

UNIVERSITÀ DEGLI STUDI DI BARI
ALDO MORO



DIPARTIMENTO DI INFORMATICA
CORSO DI LAUREA TRIENNALE IN INFORMATICA

TESI DI LAUREA
In Sviluppo dei Videogiochi

**ALGORITMI DI REINFORCEMENT LEARNING PER
L'IMPLEMENTAZIONE DI NPC NEI VIDEOGIOCHI**

Relatori:

Prof. PIERPAOLO BASILE

Laureando:
STEFANO ROMANELLI
MATRICOLA: 709725

ANNO ACCADEMICO 2023/2024

Indice

Elenco delle figure	vii
1 Introduzione	1
2 Stato dell'arte	5
2.1 Introduzione	5
2.1.1 Comportamenti NPC Deterministici	5
2.1.2 Comportamenti NPC Non Deterministici	6
2.2 Reinforcement Learning	7
2.3 Markov Decision Process	8
Policy	9
Return	9
Discounted Return	9
2.4 Value Function	10
2.5 Categorie Algoritmi di RL	11
2.5.1 Metodi Value-Based	11
2.5.2 Metodi Policy-based	12
Policy Gradient	12
Impiego delle Reti Neurali	13
2.6 Proximal Policy Optimization	13
2.6.1 Panoramica PPO	14
2.6.2 Passaggi Algoritmo PPO	14
Inizializzazione della Rete Policy	14
Raccolta delle Traiettorie	14
Calcolo delle Stime dei Vantaggi	14
Calcolo della Funzione Obiettivo Clipped Surrogate	15
Ottimizzazione della Rete Policy	15
2.7 Reinforcement Learning nei Videogiochi	15
ML-Agents	16
3 Metodologia	17
3.1 Introduzione	17

3.2	Unity	18
3.3	ML-Agents	18
3.3.1	Architettura Toolkit ML-Agents	19
3.4	Learning Environment	20
3.4.1	Agente	20
3.4.2	Scenario di Gioco Simmetrico	22
3.4.3	Compatibilità tra Agenti e Giocatori Umani	22
3.4.4	Osservazioni e Azioni Agente	23
	Osservazioni	23
	Azioni	24
3.4.5	Ricompense Agente	24
	Bias della Funzione di Ricompensa	24
3.5	Metodologie di Addestramento	25
3.5.1	Parallelizzazione	25
3.5.2	Curriculum Learning	26
3.5.3	Randomizzazione dello Scenario di Addestramento	27
3.6	Algoritmi di Addestramento	27
3.6.1	Curiosity Module	28
3.6.2	Self-Play	29
	RL negli Adversarial Games	29
	Considerazioni Pratiche e Configurazione	30
3.6.3	Imitation Learning	31
	Campionamento dell’Insieme di Dimostrazioni	32
	Generative Adversarial Imitation Learning(GAIL)	32
	Behaviour Cloning	33
3.7	Sperimentazione	34
3.7.1	Configurazioni Addestramenti	34
	Analisi dei Risultati degli Addestramenti	34
3.8	Test Utente	35
4	Progettazione	37
4.1	Ciclo di Apprendimento Agente	37
4.1.1	Definizione Features Agente	37
	Osservazioni Agente	37
	Azioni Agente	38
	Richiesta delle Decisioni	39
4.1.2	Implementazione dello Script dell’Agente	39
	Implementazione delle Osservazioni	40

Implementazione delle Azioni	42
Implementazione Euristica per il Testing dell'Environment	43
Implementazione Fine Episodio	43
Implementazione Ripristino dell'Episodio	43
4.2 Meccaniche di gioco	44
Implementazione delle Ricompense e delle Penalità . . .	45
Settings Agente	46
4.3 Implementazione Metodologie di Addestramento	47
4.3.1 Curriculum Learning	47
4.3.2 Randomizzazione degli Scenari	48
4.3.3 Self-Play	48
4.3.4 Imitation Learning	50
Campionamento delle Dimostrazioni	50
GAIL	50
Iperparametri Behaviour Cloning	51
4.4 Setting di Addestramento	51
5 Sperimentazione	53
5.1 Introduzione	53
5.2 Metriche di Valutazione	53
5.2.1 Episode Length	53
Interpretazione	54
Comportamenti Tipici in Fase di Addestramento	54
5.2.2 Cumulative Reward	55
5.2.3 Curiosity Inverse Loss	55
Interpretazione	55
Comportamenti Tipici in Fase di Addestramento	56
5.2.4 GAIL Reward	56
Interpretazione	56
5.2.5 Self-Play/ELO	56
Comportamenti Tipici in Fase di Addestramento	57
5.3 Configurazioni Addestramenti	58
5.4 Analisi dei Risultati degli Esperimenti	59
Modelli Addestrati tramite Imitation Learning	59
Modello Addestrato tramite Reinforcement Learning . .	59
5.4.1 Episode Length	60
Analisi Andamento Risultati	60
Analisi Differenze dei Risultati	61

5.4.2	Cumulative Reward	62
	Analisi Andamento Risultati	62
	Analisi Differenze dei Risultati	62
	Curiosity Inverse Loss	63
	Analisi Andamento Risultati	63
	GAIL Reward	64
	Analisi Andamento Risultati	64
	Analisi Differenze dei Risultati	64
	Self-Play/ELO	65
	Analisi Andamento Risultati	65
	Analisi Differenze dei Risultati	66
5.5	Test Utente	66
5.6	Conclusioni	67
5.6.1	Self-Play	67
5.6.2	Behaviour Cloning	68
5.6.3	Risultati Test Utente	68
6	Conclusioni	69
6.1	Valutazioni Finali	69
6.2	Sviluppi Futuri	70
	Bibliografia	73

Elenco delle figure

2.1	Interazione Agente-Ambiente del modello base del Reinforcement Learning	7
3.1	Architettura Software per il Learning Environment	19
3.2	Vari tipi di behaviour associabili ad ogni agente	20
3.3	Scenario di gioco simmetrico	22
3.4	L'insieme delle istanze parallele che hanno contribuito all'addestramento	26
3.5	Differenza in termini di ricompense cumulative medie rispetto agli step di addestramento	26
3.6	Con il Curriculum Learning lo scenario di training si complica progressivamente con l'avanzare degli step totali di addestramento	27
3.7	Confronto di Problema Tradizionale di RL e un Problema di RL in un Adversarial Game	30
4.1	Parametri componente Behaviour Parameters	38
4.2	Parametri componente Decision Requester	39
4.3	Metodo per fornire le osservazioni dell'agente al learner	41
4.4	Debug grafico del componente <i>RayPerceptionSensor3D</i> , usato per raccogliere osservazioni sul campo visivo	41
4.5	Estrazione delle azioni ricevute dal learner per ogni branch	42
4.6	Movimenti del giocatore/agente	44
4.7	Comportamento palla	45
4.8	Penalità uniforme sul tempo	45
4.9	Penalità per contatti con muri o giocatori avversari	46
4.10	Parametri curriculum learning	47
4.11	Superficie di spawn degli agenti rappresentata dall'area azzurra in ogni scenario del curriculum learning	48
4.12	configurazione iperparametri self-play	48
4.13	Metodo per terminare l'episodio per ogni agente nell'ambiente di addestramento	49
4.14	Iperparametri del modulo GAIL	50

4.15 Iperparametri del modulo BC	51
4.16 Comando per avviare l'addestramento	51
5.1 Durata media degli episodi	60
5.2 Ricompensa cumulativa media per episodio	62
5.3 Loss modulo Curiosity	63
5.4 Ricompensa cumulativa intrinseca GAIL	64
5.5 Punteggio ELO	65
5.6 Risultati test utente	67

Capitolo 1

Introduzione

Fin dagli albori dei videogiochi digitali, lo sviluppo di NPC (Non-Playable Characters) ha avuto l'obiettivo di simulare entità intelligenti capaci di collaborare, interagire e sfidare il giocatore. L'intento è sempre stato quello di offrire un'esperienza coinvolgente, che desse al giocatore l'impressione di essere parte di un ambiente complesso e dinamico. Con il progresso delle tecnologie, gli ambienti di gioco e le regole che determinano il comportamento degli NPC sono diventati sempre più articolati. In particolare, la recente diffusione del Machine Learning ha permesso sperimentazioni interessanti nel mondo dei videogames. Ad esempio, recentemente l'introduzione della tecnologia NVIDIA ACE ha dimostrato come l'IA generativa possa migliorare l'interazione con gli NPC, rendendoli più naturali e credibili, aumentando il coinvolgimento dei giocatori (NVIDIA, 2023).

Tradizionalmente, per implementare i comportamenti degli NPC, la maggior parte degli sviluppatori adotta tecniche deterministiche, suddividendone gli stati comportamentali in modo rigido. Tuttavia, queste soluzioni tendono a semplificare eccessivamente i comportamenti degli NPC, impedendo loro di sfruttare appieno le meccaniche di gioco offerte dall'ambiente. Di conseguenza, il gameplay spesso risulta poco dinamico, meccanico e ripetitivo, riducendo il fattore di immedesimazione del giocatore e, in molti casi, il divertimento. Per esempio, in giochi open-world come The Elder Scrolls V: Skyrim, i comportamenti scriptati degli NPC sono stati spesso criticati per la loro ripetitività, evidenziando come la mancanza di variabilità possa ridurre il senso di credibilità del mondo di gioco. Per mitigare questi limiti, gli sviluppatori hanno proposto più metodologie, come introdurre elementi di stocasticità nelle azioni degli NPC, rendendole meno prevedibili rispetto allo stato in cui si trovano. Un'altra strategia comunemente adottata è l'uso di tecniche deterministiche avanzate, come il Goal-Oriented Action Planning (GOAP) che vedremo nel Capitolo 2, che permette di simulare un processo decisionale

volto a pianificare strategie per raggiungere obiettivi specifici. Un esempio significativo è rappresentato dagli NPC del gioco F.E.A.R.[9], che adottano GOAP per creare tattiche di combattimento dinamiche, adattandosi al contesto del giocatore in modo convincente.

Sebbene queste soluzioni migliorino alcuni aspetti del comportamento degli NPC, queste non riescono comunque a sfruttare appieno la complessità e la dinamicità degli ambienti di gioco moderni. Con la crescente complessità degli ambienti e delle meccaniche di gioco, gli sviluppatori si trovano spesso a dover gestire strutture deterministiche sempre più complesse. Questo aumenta la probabilità di errori, come comportamenti contraddittori, comportamenti indesiderati o conflitti tra entità di gioco, poiché diventa difficile prevedere e gestire tutti gli stati possibili e le interazioni all'interno dell'ambiente, rischiando di restare invisi chiati nella complessa logica implementata. Per evitare queste difficoltà, gli sviluppatori ricorrono talvolta a semplificazioni che, pur riducendo la complessità degli stati logici dell'NPC, finiscono per rendere i comportamenti troppo prevedibili e meccanici.

Queste sfide evidenziano la necessità di tecniche più avanzate, in grado di elaborare e costruire strategie che sfruttino appieno le potenzialità offerte dall'ambiente di gioco. Tra queste tecniche spicca l'uso del Machine Learning, in particolare dell'Apprendimento per Rinforzo (Reinforcement Learning o RL). Il RL è una tecnica di Machine Learning che consente agli agenti di apprendere comportamenti complessi attraverso l'interazione con l'ambiente, adattandosi dinamicamente a situazioni di gioco diverse e adottando scelte ottimali in base al contesto.

Il RL tramite addestramento multi-agente può essere utilizzato per addestrare agenti capaci di competere contro altri agenti in scenari di gioco simmetrici, sfruttando tecniche di addestramento multi-agente come il self-play, tecniche sperimentate e analizzate nel contesto di questa tesi. Tuttavia, l'addestramento multi-agente basato esclusivamente sul RL in scenari di questo tipo presenta alcune problematiche: gli agenti potrebbero apprendere strategie efficaci solo nel contesto di addestramento, ma meno valide o credibili quando si confrontano con giocatori umani. Questo limite deriva dal fatto che applicando il solo RL, tende a ottimizzare una politica basata esclusivamente sulla ricerca della massimizzazione della ricompensa, spesso a scapito del realismo. Di conseguenza, gli agenti possono adottare comportamenti poco realistici, riducendo la loro efficacia e adattabilità in contesti reali. L'introduzione

di tecniche di Imitation Learning può mitigare queste problematiche. In particolare, queste tecniche consentono agli agenti di apprendere politiche che non solo ottimizzano il rendimento in termini di ricompensa, ma risultano anche più credibili e realistiche quando utilizzate per sfidare giocatori umani.

Questa tesi si propone di affrontare il problema specifico dell’addestramento multi-agente, in cui gli agenti imparano a interagire in contesti di giochi competitivi e simmetrici con altri agenti, utilizzando tecniche di Reinforcement Learning (RL), Imitation Learning e metodologie di addestramento avanzate, come il Curriculum Learning e il Self-Play. L’obiettivo principale è dimostrare che un agente addestrato con queste tecniche sia in grado di competere con esseri umani in modo realistico, replicando comportamenti simili a quelli umani e offrendo un livello di sfida soddisfacente per il giocatore.

Le tecniche e le metodologie adottate vengono introdotte nel Capitolo 3 e implementate nel Capitolo 4. Nel Capitolo 5, tutte le tecniche proposte dallo studio vengono sperimentate e confrontate tramite test per valutarne l’efficacia e le differenze. Inoltre, il Capitolo 5 include una sottosezione dedicata ai test utente, finalizzati a valutare il grado di realismo percepito dagli utenti durante l’interazione con gli agenti(NPC).

Oggi, i ritmi serrati del mercato videoludico limitano la possibilità di sperimentare approcci innovativi nella gestione dei comportamenti degli NPC, favorendo l’adozione di tecniche più semplici e collaudate. Pochi prodotti dell’industria osano introdurre nuove modalità per sviluppare comportamenti realistici. In questo contesto, sperimentazioni come quelle descritte in questa tesi non solo dimostrano le potenzialità dell’addestramento di NPC tramite Reinforcement Learning, ma mirano anche ad anticipare le opportunità future nell’utilizzo di queste tecniche per migliorare la profondità delle interazioni degli NPC con i giocatori nei prodotti videoludici.

Capitolo 2

Stato dell'arte

2.1 Introduzione

2.1.1 Comportamenti NPC Deterministici

Come già detto nell'introduzione, tradizionalmente codificare il comportamento di un NPC(Personaggio non giocante) nei prodotti videoludici equivale a costruire rappresentazioni strutturate come automi deterministici, che consistono nel variare il loro stato in base ai feedback ricevuti dall'ambiente o in base agli obiettivi portati a termine. Questi approcci sono particolarmente popolari perché sono semplici da implementare, sono comprensibili per gli umani, sono adatti per modellare comportamenti discreti e sequenze di azioni. Se correttamente configurate, queste tecniche permettono quasi sempre di concludere le attività in modo controllato. Inoltre, lo stato degli NPC è sempre noto e prevedibile nelle fasi di testing-debugging in quanto questi metodi seguono regole esplicite. Di seguito una lista delle tecniche deterministiche:

- **Finite State Machine:** Automa a stati finiti basato su stati e transizioni
- **Hierarchical FSM (HFSM):** Suddivide il comportamento in sottogruppi, semplificando la gestione di comportamenti complessi.
- **Behavior Trees:** In cui ogni elemento costitutivo di un comportamento è un'attività piuttosto che uno stato. In questa tecnica possono essere intraprese decisioni stocastiche ma sono comunque implementate in modo da essere prevedibili dai progettisti.
- **Goal-Oriented Action Planning (GOAP):** Usa algoritmi di pianificazione intelligenza artificiale(come A*) per pianificare e determinare dinamicamente una sequenza di azioni per raggiungere un obiettivo prefissato.

La credibilità e l'efficacia degli NPC generalmente, dipende dalla complessità delle meccaniche di gioco e dalla correttezza della rappresentazione degli stati da parte del progettista. Nonostante le possibili astrazioni offerte, queste tecniche in alcuni generi di videogiochi possono risultare eccessivamente meccaniche e poco credibili. Questo abbassa il grado di immedesimazione o l'effetto sorpresa che una strategia costruita dinamicamente potrebbe offrire.

Esempio: Supponiamo di creare un gioco multigiocatore a tema bellico in cui i giocatori controllano i soldati. Le partite potrebbero non essere completamente riempite da giocatori per vari motivi e quindi si decide di sostituirli momentaneamente con degli agenti(NPC). Se il ruolo dell'NPC è quello del medico, questo dovrà cercare e rianimare i giocatori feriti. Il comportamento di un medico è complesso. Innanzitutto dovrà evitare di ricevere danni, che richiederebbe riconoscere quando esso si trova in pericolo per poi spostarsi in un luogo sicuro. Inoltre deve essere consapevole di quali sono i membri del suo team che sono feriti e che necessitano di assistenza. In caso di feriti multipli, l'agente dovrà valutare e decidere in base al grado di lesione chi aiutare per primo. Infine un buon medico sceglierà una posizione strategica in cui può aiutare rapidamente i membri del team. Considerare tutti questi fattori significa che l'agente deve analizzare diverse caratteristiche e configurazioni dell'ambiente in cui si trova e decidere quale azione effettuare. Dato il gran numero di configurazioni dell'ambiente e il gran numero di azioni che l'agente può intraprendere, definire e implementare manualmente tali comportamenti complessi in modo efficace può risultare molto difficile e soggetto a errori di sviluppo.

2.1.2 Comportamenti NPC Non Deterministici

Le tecniche di Reinforcement Learning e di Imitation Learning permettono di addestrare agenti in grado di esibire comportamenti più complessi e realistici, come quelli nell'esempio. Il punto debole delle tecniche basate su modelli appresi è che i progettisti non sono in grado di comprendere il perché delle scelte intraprese in una sequenza di azioni, e quindi risulta difficile prevedere gli effetti degli agenti in un ambiente di gioco. In alcuni casi questo aspetto potrebbe essere gestito costruendo dei sistemi ibridi basati su strategie deterministiche che integrano l'utilizzo di modelli appresi, in modo da adottare un diverso modello appreso in base allo stato dell'automa deterministico(Questa possibile metodologia viene accennata nella sezione sugli sviluppi futuri di questa tesi 6.2.

2.2 Reinforcement Learning

A differenza dell'apprendimento supervisionato, dove un algoritmo apprende da un dataset pre-fornito di esempi etichettati, l'apprendimento tramite RL si basa sull'esplorazione dell'ambiente, raccogliendo campioni e compiendo azioni, il cui feedback è fornito sotto forma di ricompense o penalità. Questo rende il RL particolarmente utile in scenari in cui l'ambiente è sconosciuto a priori e necessita di essere esplorato per apprendere progressivamente come risolvere al meglio un compito massimizzando una ricompensa cumulativa.

Nel RL l'attenzione è rivolta alla ricerca di un equilibrio tra esplorazione (di territorio inesplorato) e sfruttamento (della conoscenza attuale) con l'obiettivo di costruire un comportamento ottimale (policy) in grado di massimizzare la ricompensa cumulativa. La ricerca di questo equilibrio è nota come dilemma esplorazione-sfruttamento (oppure noto anche come tradeoff esplorazione-sfruttamento), un concetto fondamentale nel processo decisionale [12]. È rappresentato come l'atto di bilanciamento tra due strategie opposte. Lo sfruttamento implica la scelta dell'opzione migliore in base alla conoscenza attuale del sistema (che può essere incompleta o fuorviante), mentre l'esplorazione implica la sperimentazione di nuove opzioni che possono portare a risultati migliori in futuro a scapito di un'opportunità di sfruttamento. Trovare l'equilibrio ottimale tra queste due strategie è una sfida cruciale in molti problemi decisionali il cui obiettivo è massimizzare i benefici a lungo termine.

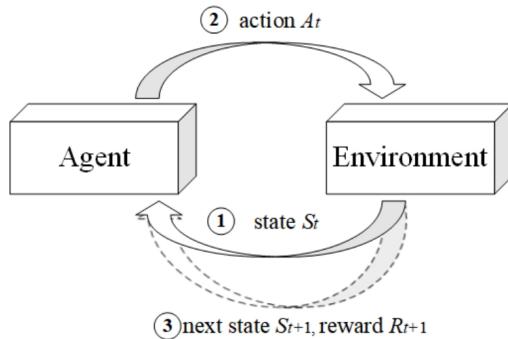


FIGURA 2.1: Interazione Agente-Ambiente del modello base del Reinforcement Learning

Il Reinforcement Learning può essere definito come un modello di interazione agente-ambiente. Il modello di base del RL utilizza i seguenti concetti:

- **Ambiente e agente:** L'agente e l'ambiente sono due parti che coesistono

nel modello del RL. Un agente migliora il suo comportamento(policy) interagendo con l’ambiente. Più specificamente, esso intraprendono una serie di azioni nell’ambiente attraverso una policy cercando di massimizzare la ricompensa o raggiungere un certo obiettivo.

- **Passo temporale:** L’intero processo del RL viene discretizzato in diversi passi temporali. Ad ogni passo temporale, l’ambiente e l’agente interagiscono.
- **Stato:** Lo stato riflette le osservazioni dell’agente sull’ambiente. Quando l’agente intraprende un’azione, anche lo stato cambierà. In altre parole l’ambiente passa ad uno stato successivo dopo un’azione intrapresa.
- **Azioni:** L’agente può valutare l’ambiente, prendere decisioni e infine intraprendere determinate azioni. Queste azioni sono prese in relazione allo stato osservato dell’ambiente.
- **Ricompensa:** Dopo aver ricevuto l’azione dell’agente, l’ambiente fornirà all’agente lo stato dell’ambiente corrente e la ricompensa per l’azione precedente. La ricompensa rappresenta una valutazione dell’azione intrapresa dall’agente.

2.3 Markov Decision Process

Quasi tutti i problemi di RL possono essere formalizzati in termini del processo decisionale di Markov(MDP). Un MDP è un modello per il processo decisionale sequenziale quando i risultati sono incerti [11]. Gli MDP sono utili per lo studio di una vasta gamma di problemi di ottimizzazione, risolti con la programmazione dinamica e tramite RL. Più precisamente, un processo decisionale di Markov è un processo di controllo stocastico a tempo discreto. Se gli spazi degli stati e delle azioni sono finiti, allora il problema è chiamato MDP finito. Gli MDP finiti sono particolarmente importanti per la teoria del RL.

Un MDP finito è definito da:

- uno spazio degli stati S .
- uno spazio delle azioni A disponibili dallo stato s .
- $P_s(s, s')$ che a livello intuitivo, rappresenta la probabilità che l’azione a nello stato s al momento t porti allo stato s' al momento $t + 1$.

- una ricompensa immediata $R_a(s, s')$ ricevuta dopo la transizione dallo stato s allo stato s' tramite l'azione a .

Policy

La selezione delle azioni di un'agente è modellata come una funzione chiamata policy π ed è una mappa:

$$\pi : A \times S \rightarrow [0, 1]$$

$$\pi(a, s) = P(A_t = a | S_t = s)$$

La mappa policy fornisce qual è la probabilità di intraprendere un'azione a quando l'ambiente si trova nello stato s .

Return

Poiché il nostro obiettivo è massimizzare la ricompensa totale, possiamo quantificare questa ricompensa totale dell'agente su tutti i passi temporali t , chiamata Return(ritorno) con:

$$G_t = R_{t+1} + R_{t+2} + \dots + R_T$$

Dove T è l'ultimo passo, e quindi S_T è lo stato di terminazione. Il ritorno G_t calcola la somma delle ricompense future che l'agente si aspetta di ricevere seguendo la policy π . Un episodio è completato quando l'agente completa l'azione di terminazione.

Esempio: Supponiamo che un agente riceva le seguenti ricompense a partire dal tempo $t : 2, 3, 4, 5$, il ritorno sarà $G_t = 2 + 3 + 1 + 5 = 11$

Discounted Return

Oltre a questo tipo di task episodico, esiste un altro tipo di task che non ha uno stato di terminazione, in altre parole, può teoricamente funzionare per sempre. Questo tipo di task è chiamato task continuo. Per i task continui, poiché non esiste uno stato di terminazione, la definizione di ritorno può essere divergente. Viene quindi introdotto un altro modo per calcolare il ritorno, chiamato ritorno **discounted return**(ritorno scontato), cioè:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Il cui **discount factor** γ soddisfa $0 < \gamma < 1$. Quando $\gamma = 1$, l'agente può ottenere il valore pieno di tutti i passi futuri, mentre quando $\gamma = 0$, l'agente può

solo vedere la ricompensa corrente. Quando γ è inferiore a 1, le ricompense nel lontano futuro hanno peso inferiore rispetto alle ricompense nel futuro immediato(osservare esponente di γ). Far tendere γ da 0 a 1, renderà l'agente lungimirante(nel caso in cui ci interessi prevedere le ricompense più lontane nel futuro). L'agente quindi non cercherà di massimizzare la ricompensa immediatamente ottenibile ma cercherà di farlo sul lungo periodo guardando al proprio futuro.

Esempio: usando le ricompense dell'esempio precedente, il *discounted return* sarà $G_t = 2 + 0.9 \cdot 3 + 0.9^2 \cdot 1 + 0.9^3 \cdot 5 = 9.155$

2.4 Value Function

La Value Function è la previsione media dell'agente delle ricompense future, questa viene utilizzata per valutare la qualità dello stato in cui l'agente si trova e selezionare le azioni più convenienti. La differenza tra la Value Function e una ricompensa, è che una ricompensa R è definita come una valutazione immediata di una certa interazione A_t , mentre la Value Function è definita come il "return" medio delle azioni su un lungo periodo di tempo. In altre parole, la Value Function dello stato corrente $S_t = s$ è il return medio a lungo termine. Ci sono due importanti funzioni di valore nel campo dell'RL:

- **La Value Function Dello Stato $V_\pi(s)$:** Rappresenta il return medio che si ottiene se l'agente continua a seguire la strategia π dopo aver raggiunto un certo stato S_t . In altre parole la value function $V_\pi(s)$ per uno specifico stato s indica quanto è utile o vantaggioso trovarsi in quello stato, in termini di ricompense future attese, se l'agente segue la policy π . Essa è definita come:

$$V_\pi(s) = E_\pi[G_t | S_t = s]$$

Dove E_π rappresenta l'Aspettazione(o valore atteso) rispetto alla policy

- **La Value Function Dell'Azione $Q_\pi(s, a)$:** Rappresenta invece il ritorno medio che si ottiene se una certa azione $A_t = a$ viene intrapresa da un certo stato corrente $S_t = s$ in cui le azioni successive vengono intraprese secondo la policy π . La sua definizione è:

$$Q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$

L'obiettivo nel RL è imparare la policy ottimale π^* che può massimizzare la value function interagendo con l'ambiente. L'obiettivo dell'agente non deve

essere ottenere la massima ricompensa ad ogni singola azione del breve termine, ma ottenere la ricompensa più alta nel lungo periodo. Pertanto la policy può essere formulata come:

$$\pi^* = \arg \max_{\pi} V_{\pi}(s)$$

2.5 Categorie Algoritmi di RL

Nel RL ci sono diverse categorie di algoritmi con strategie differenti, il cui obiettivo comune è ottenere una policy ottimale π^* . Qualunque metodo si utilizzi per risolvere un problema di RL, si avrà una policy. Due dei metodi principali sono:

- **Value-Based:** Nel caso dei metodi Value-Based non si addestra una policy: In questo caso la policy è una semplice funzione pre-specificata(ad esempio, la Greedy Policy seleziona l'azione con il massimo return atteso) che usa i valori forniti dalla value function per selezionare l'azione successiva. Questi metodi si basano sul trovare una value function ottimale (indicata con Q^* o V^*) che appunto portino ad ottenere una politica ottimale.
- **Policy-Based:** Nei metodi Policy-Based viene addestrata direttamente una policy, che selezioni direttamente l'azione da intraprendere dato uno stato(oppure una distribuzione di probabilità), senza alcuna value function.

2.5.1 Metodi Value-Based

Nei metodi Value-Based, l'idea è di apprendere la value function che stimi quanto è "vantaggioso" essere in un dato stato oppure quanto è vantaggioso eseguire una certa azione in un dato stato. In altre parole, l'obiettivo della tecnica Value-Based è quello di trovare la migliore value function Q^* che rappresenti il return massimo atteso per ogni coppia stato-azione. Poiché la policy non è addestrata/appresa, se vogliamo intraprendere azioni che porteranno sempre più alla ricompensa più grande, creeremo una Greedy Policy. Una Greedy Policy seleziona l'azione con il return medio più alto. Trovare una value function ottimale equivale quindi a ottenere una politica ottimale.

Un algoritmo tipico value-based è il Q-learning. Il problema principale di metodi value-based come il Q-learning è che essi possono funzionare solo in spazi

di azione discreti¹, di conseguenza hanno difficoltà nel gestire valori stato-azione continui.

2.5.2 Metodi Policy-based

Nei metodi Policy-Based, viene modellata direttamente una policy ottimale π^* senza l'uso di una value function. Questo approccio riduce il problema a una ricerca locale nello spazio delle policy, trasformandolo in un caso di ottimizzazione stocastica (sia basata su gradiente che non).

Un algoritmo policy-based aggiorna i parametri della policy tramite addestramento e genera la policy ottimale π^* . Nei metodi policy-based, immettiamo uno stato e otteniamo direttamente l'azione corrispondente. Uno degli algoritmi policy-based più basilari è il Policy Gradient.

Policy Gradient

Gli algoritmi di Policy Gradient ottimizzano direttamente la policy aggiornando i parametri di una rete neurale tramite il calcolo del gradiente della ricompensa attesa[15]. La funzione obiettivo da massimizzare $J(\theta)$ è la ricompensa cumulativa attesa, cioè:

$$J(\theta) = E_{\tau \sim \theta(\tau)}[r(\tau)] = \int_{\tau \sim \pi(\tau)} r(\tau) \pi_\theta(\tau) d\tau$$

Dove:

- θ rappresenta i parametri della policy.
- τ indica una traiettoria, ovvero una sequenza di stati e azioni.
- $r(\tau)$ è la ricompensa totale ottenuta lungo la traiettoria τ .
- $\pi_\theta(\tau)$ è la probabilità di osservare la traiettoria τ seguendo la policy π_θ .
- $E_{\tau \sim \theta(\tau)}[r(\tau)]$ indica che stiamo calcolando l'aspettazione della ricompensa totale $r(\tau)$ su tutte le possibili traiettorie τ che l'agente può seguire, secondo la distribuzione delle traiettorie seguendo la politica π_θ . Una traiettoria è una sequenza di stati, azioni e ricompense.

¹Lavori recenti hanno sviluppato architetture di reti neurali che consentono di approssimare funzioni di valore stato-azione continue (traiettorie continue) nel contesto del Reinforcement Learning, utilizzando l'errore di Bellman[2]

- L'integrale $\int_{\tau \pi(\tau)} r(\tau) \pi_\theta(\tau) d\tau$ rappresenta la somma delle ricompense ponderate dalla probabilità su tutte le possibili traiettorie τ seguendo la politica π_θ . L'integrale è la forma continua della somma delle ricompense attese.

Da cui ottenere la derivata di $J(\theta)$, ovvero il gradiente della funzione obiettivo:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^T \nabla_\theta \log \pi_\theta(A_t | S_t) \sum_{t=1}^T r(S_t, A_t) \right]$$

I vantaggi degli algoritmi Policy Gradient sono la loro applicabilità in spazi in cui le azioni sono continue. Poiché non è disponibile un'espressione analitica per il gradiente, è possibile ottenerne solo una stima rumorosa. Tale stima può essere costruita in vari modi, utilizzando algoritmi come il metodo REINFORCE di Williams [17] o il Proximal Policy Optimization (PPO), che sarà introdotto nelle sezioni successive e utilizzato tramite il toolkit ML-Agents.

Un'ultima importante osservazione sui metodi Policy Gradient è che questi algoritmi, essendo basati sulla ricerca locale, possono bloccarsi in ottimi locali.

Impiego delle Reti Neurali

Con la continua espansione dell'applicazione del deep learning e delle reti neurali, la loro crescita ha investito anche il campo del Reinforcement Learning, portando all'utilizzo di reti neurali multi-layer per approssimare la value function o la policy function negli algoritmi di RL[5]. Nei metodi Policy Gradient, la rete neurale viene utilizzata per approssimare la funzione di policy. Nel caso specifico di questa tesi, l'addestramento dell'esperimento si basa su un metodo Policy Gradient, in cui viene impiegata una rete neurale per apprendere la policy ottimale π^* .

2.6 Proximal Policy Optimization

Ottener buoni risultati con i metodi Policy Gradient è generalmente difficile, poiché questi algoritmi sono sensibili alla scelta della dimensione del passo(learning rate). Passi troppo piccoli rallentano il processo di convergenza verso una soluzione ottimale, mentre passi troppo grandi possono causare il superamento di soluzioni ottimali e introdurre un eccessivo rumore. Inoltre, questi metodi spesso presentano un'efficienza di campionamento molto scarsa, richiedendo milioni o miliardi di timestep per apprendere solo semplici attività.

2.6.1 Panoramica PPO

I Proximal Policy Optimization (PPO) sono una classe di algoritmi di Reinforcement Learning che rientrano nei metodi Policy Gradient [13]. Questi algoritmi si basano sull’addestramento di una rete neurale per approssimare la funzione di policy di un agente. Molti esperti lo definiscono i PPO lo stato dell’arte in quanto offrono prestazioni comparabili o superiori rispetto agli approcci all’avanguardia, pur essendo relativamente semplici da implementare e ottimizzare.

L’idea alla base dei PPO è quella di limitare gli aggiornamenti della policy troppo ampi su una nuova funzione obiettivo chiamata funzione obiettivo Clipped Surrogate, la quale limiterà la modifica della policy in un intervallo limitato usando una clip.

2.6.2 Passaggi Algoritmo PPO

Inizializzazione della Rete Policy

L’algoritmo inizializza una rete neurale che rappresenta la policy. Questa rete legge un’osservazione fornita(stato attuale) e restituisce come input la distribuzione di probabilità sulle possibili azioni.

Raccolta delle Traiettorie

L’agente, seguendo la policy attuale, interagisce con l’ambiente raccogliendo traiettorie formate da stati, azioni e ricompense.

Calcolo delle Stime dei Vantaggi

Utilizzando le traiettorie raccolte, vengono calcolate le stime dei vantaggi per ogni coppia stato-azione. Le stime dei vantaggi rappresentano quanto un’azione sia migliore o peggiore rispetto all’azione media intrapresa da un dato stato. Il vantaggio $A(s, a)$ misura quanto sia buono o cattivo intraprendere un’azione specifica a in un particolare stato s .

Il vantaggio è calcolato come:

$$A(s, a) = G_t - V(s)$$

dove G_t è il return al passo t e $V(s)$ è il valore dello stato. Valori positivi del vantaggio indicano che l'azione è stata migliore della media, mentre i valori negativi indicano che l'azione è stata peggiore della media.

Calcolo della Funzione Obiettivo Clipped Surrogate

Questa funzione obiettivo approssima il miglioramento della policy, assicurando al contempo che l'aggiornamento della policy rimanga entro un intervallo sicuro.

$$L^{\text{clip}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \cdot A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \cdot A_t \right) \right]$$

Per evitare aggiornamenti della policy di dimensioni troppo elevate, il rapporto tra la probabilità di azioni in base alla nuova policy e a quella precedente viene ridotto a un intervallo specificato $[1 - \epsilon, 1 + \epsilon]$. ϵ è il parametro di clipping. $r_t(\theta)$ rappresenta il rapporto tra la probabilità di intraprendere un'azione in base alla nuova policy e quella in base alla vecchia policy.

$$r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\text{old}}(a_t | s_t)}$$

Ottimizzazione della Rete Policy

Per aggiornare i parametri della rete policy si utilizza la discesa del gradiente stocastico(SGD) o una sua variante, riducendo al minimo l'obiettivo surrogato (stima della ricompensa cumulativa media).

2.7 Reinforcement Learning nei Videogiochi

Come discusso nella sottosezione 2.1.1, la maggior parte delle produzioni di sviluppo di videogiochi preferisce approcci tradizionali per implementare il comportamento degli NPC. Tuttavia, negli ultimi anni, anche l'industria videoludica ha iniziato a sperimentare e adottare tecnologie basate sul machine learning, tra cui il RL per gestire compiti specifici degli NPC. Nonostante i potenziali vantaggi, l'adozione del RL per progettare comportamenti realistici, simili a quelli dei giocatori umani, è ancora limitata. Questa poca fiducia nella loro adozione potrebbe derivare dalla mancanza di maturità di tali tecnologie, che ad oggi non garantisce di soddisfare le esigenze e gli standard dell'industria su larga scala. Di seguito, vengono presentati alcuni studi che esplorano l'applicazione del RL nel settore videoludico, con l'obiettivo di migliorare l'esperienza di gioco attraverso comportamenti più naturali e realistici degli NPC.

Un primo esempio è fornito da un articolo che analizza le limitazioni dei sistemi di navigazione tradizionali utilizzati nei motori di gioco. Questi sistemi, spesso rigidi, sono difficili da integrare con le capacità di movimento avanzate degli NPC. Tramite l'utilizzo del Deep Reinforcement Learning (DRL), gli autori propongono un'alternativa ai sistemi classici, rendendo gli agenti capaci di spostarsi utilizzando un'ampia gamma di abilità di navigazione [1].

Un altro studio dimostra come il DRL possa diventare più accessibile al settore dei videogiochi. Gli autori evidenziano che gli agenti addestrati con tecniche di RL mostrano una capacità di adattamento degli NPC superiore rispetto alle architetture tradizionali, rendendo framework basati su RL più robusti ai cambiamenti dinamici del gameplay durante le fasi di sviluppo [14].

Infine, un ulteriore esperimento esplora la possibilità di sostituire completamente gli NPC con agenti di intelligenza artificiale addestrati tramite algoritmi di RL. I risultati suggeriscono che tali agenti possono migliorare l'esperienza di gioco, offrendo interazioni più coinvolgenti e dinamiche [18]. Studi come questi mirano ad anticipare le opportunità di queste tecniche nell'industria videoludica, mostrando il potenziale di queste tecnologie nel superare le limitazioni dei metodi tradizionali e nel creare esperienze di gioco più ricche e realistiche. Le problematiche principali riguardano come tecnologie come queste siano poco prevedibili e spesso difficili da configurare (corretta configurazione delle ricompense e delle osservazioni).

Non sono stati individuati studi che sperimentino l'integrazione di tecnologie come l'apprendimento per imitazione e il RL al fine di migliorare la percezione di comportamenti umani degli NPC all'interno di un videogioco. La presente tesi si propone quindi di esplorare, sperimentare e dimostrare l'efficacia di tali tecniche.

ML-Agents

Le ottime prestazioni di PPO e il suo impiego nel toolkit ML-Agents(Unity) hanno permesso di condurre molteplici studi ed esperimenti sul RL negli ultimi anni. Nel prossimo capitolo, analizzeremo come questo toolkit consenta l'accesso ad algoritmi all'avanguardia, permettendo di sperimentare metodologie usate in questo lavoro per addestrare agenti in grado di sfidare giocatori umani in un gioco simmetrico, replicando comportamenti umani realistici, pur offrendo un livello di sfida soddisfacente per i giocatori.

Capitolo 3

Metodologia

3.1 Introduzione

Le tecniche proposte nel capitolo precedente affrontano vari approcci per addestrare gli agenti tramite RL, consentendo loro di svolgere task all'interno del gioco in modo più efficace e dinamico attraverso il Reinforcement Learning. La scarsa disponibilità di articoli o studi focalizzati su questo tema rappresenta una delle ragioni principali per cui questa tesi propone di sperimentare tecniche volte ad addestrare agenti capaci di sfidare giocatori umani in modo realistico e competitivo. Per affrontare questa sfida, questa tesi propone una metodologia che combina tecniche di RL con metodologie di Imitation Learning(fornite dal toolkit ML-Agents). L'obiettivo è sviluppare agenti che non solo siano in grado di competere contro altri giocatori, ma che possano anche apprendere e replicare comportamenti complessi osservati in dimostrazioni di partite giocate da esseri umani. Attraverso questa combinazione, si mira a influenzare la policy appresa tramite RL utilizzando i comportamenti umani campionati, al fine di restituire un'esperienza di gioco realistica senza perdere la capacità di improvvisazione e adattamento tipica del RL.

Il primo passo di questo processo è stato adottare la tecnica del Self-Play, che consente di addestrare agenti in scenari simmetrici, dove imparano sia a orientarsi nel mondo di gioco sia a competere contro avversari simmetrici, ovvero agenti con capacità equivalenti. Successivamente, sono state integrate tecniche di Imitation Learning, originariamente introdotte per accelerare la convergenza dell'apprendimento. Nella tesi, queste tecniche sono state utilizzate per insegnare agli agenti a imitare comportamenti osservati in dimostrazioni di gioco tra giocatori umani, consentendo loro di adottare strategie più sofisticate. L'obiettivo è dimostrare che, combinando Self-Play e Imitation Learning, gli agenti possono migliorare sia l'efficacia che il realismo delle strategie apprese.

La metodologia proposta sarà validata attraverso test con utenti, in cui giocatori umani sfideranno gli agenti addestrati durante la sperimentazione. I partecipanti forniranno un feedback sul livello di sfida percepito e sul realismo delle strategie adottate dagli agenti. Le implementazioni specifiche e i dettagli tecnici saranno discussi nel Capitolo 4.

3.2 Unity

Gli addestramenti sono stati realizzati all'interno di uno scenario progettato utilizzando Unity, una piattaforma di sviluppo software ampiamente utilizzata per la creazione di giochi, esperienze di realtà virtuale, simulazioni e altre applicazioni interattive. Unity, grazie alla sua flessibilità e al supporto per il plugin ML-Agents, si è affermata come un potenziale strumento per la progettazione e la sperimentazione di algoritmi di Reinforcement Learning.

L'integrazione di ML-Agents in Unity consente di creare rapidamente ambienti personalizzati e scalabili, rendendo possibile il test e l'ottimizzazione di algoritmi complessi in scenari simulati. Questo approccio permette non solo di accelerare il ciclo di sviluppo degli agenti, ma anche di garantire un elevato grado di controllo sulle variabili ambientali e sulle dinamiche di interazione, caratteristiche essenziali per la validazione di algoritmi di apprendimento automatico.

3.3 ML-Agents

ML-Agents è un progetto open source che include un set di strumenti di Machine Learning allo stato dell'arte, progettato per utilizzare simulazioni e giochi sviluppati in Unity come ambienti per l'addestramento di agenti intelligenti. La libreria offre a ricercatori e sviluppatori un'interfaccia tramite API Python, consentendo l'accesso a implementazioni basate su PyTorch. Gli algoritmi disponibili includono tecniche di Reinforcement Learning, Imitation Learning, Neuroevolution e altri metodi avanzati.

Gli agenti addestrati possono essere utilizzati per molteplici scopi, come il controllo del comportamento degli NPC (personaggi non giocanti) e i test automatizzati delle build di gioco. Inoltre, le attività di ricerca nel campo dell'intelligenza artificiale evidenziano sempre più spesso come i moderni motori di gioco siano particolarmente adatti a simulare ambienti ricchi e complessi, rendendoli utili per compiere progressi in diversi ambiti di ricerca [7].

3.3.1 Architettura Toolkit ML-Agents

Il toolkit ML-Agents è composto da cinque componenti principali di alto livello:

- **Learning Environment:** Rappresenta la scena Unity e tutte le entità presenti in essa. ML-Agents permette di trasformare qualsiasi scena Unity in un ambiente di apprendimento per gli agenti, consentendo di definire ciò che osservano e le azioni che possono intraprendere.
- **Python Low-Level API:** È un’interfaccia Python di basso livello che permette di interagire con l’environment di apprendimento. Essa si trova all’esterno di Unity e comunica con esso tramite l’*External Communicator*
- **External Communicator:** Collega il Learning Environment con l’API Low-Level Python. Risiede all’interno del Learning Environment.
- **Python Trainers:** Contiene tutti gli algoritmi di Machine Learning che permettono di addestrare agenti. Questi algoritmi comunicano con il Learning Environment tramite la Python Low-Level API.

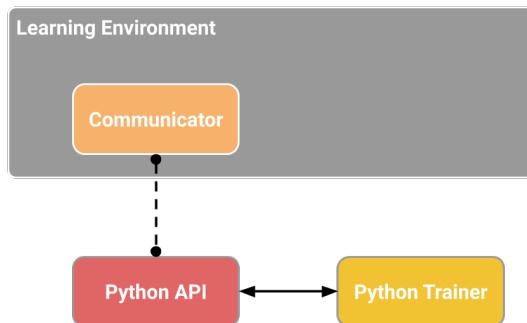


FIGURA 3.1: Architettura Software per il Learning Environment

Il Learning Environment contiene due componenti Unity che aiutano a organizzare la scena Unity:

- **Agents:** È un componente associato a un GameObject di Unity (qualsiasi entità capace di compiere azioni e osservazioni all’interno di una scena). Questo componente gestisce la generazione delle osservazioni dell’agente, l’esecuzione delle azioni ricevute dal trainer e l’assegnazione di una ricompensa appropriata. Ogni agente è collegato a un Behaviour.
- **Behavior:** Permette di definire attributi specifici dell’agente. Ogni Behaviour è identificato in modo univoco da un id. Un Behaviour può essere pensato come una funzione che riceve osservazioni e ricompense dall’agente e restituisce delle azioni. Un Behaviour può essere di tre tipi:

- Learning: Un behaviour di tipo Learning è un comportamento ancora in fase di addestramento non ancora definito. Nella figura 3.2 rappresenta il Behaviour A.
- Heuristic: Un behaviour euristico è definito da un set di regole "hard-coded" implementate da codice.
- Inference: Un behavior di inferenza è un comportamento guidato dal funzionamento di una rete neurale addestrata. In altre parole, dopo aver addestrato una rete neurale questa stabilisce il behaviour (policy) dell'agente in fase di inferenza.

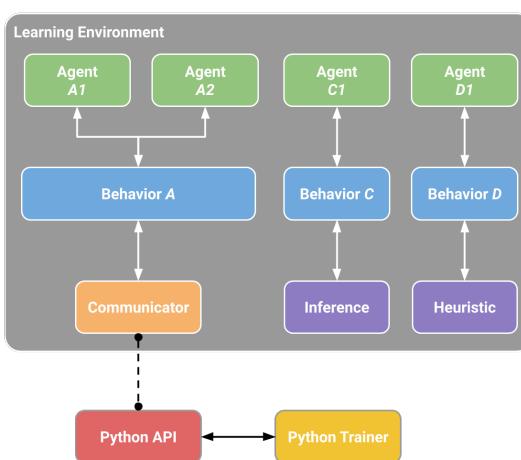


FIGURA 3.2: Vari tipi di behaviour associabili ad ogni agente

3.4 Learning Environment

Il Learning Environment rappresenta l'intera scena in Unity che comprende tutte le entità che influenzano direttamente le osservazioni dell'agente. Le azioni dell'agente all'interno dell'environment ne modificano lo stato, i Reward sono ottenibili dopo una corretta sequenza delle azioni di un agente. Modellare correttamente un Learning Environment durante la fase di progettazione è un aspetto fondamentale. Un'adeguata progettazione del Learning Environment consente l'addestramento di policy efficaci che abbiano una buona capacità di generalizzazione.

3.4.1 Agente

Un agente in ML-Agents è un'entità capace di interagire nell'ambiente di gioco che può:

- Osservare l'ambiente circostante.
- Prendere decisioni basate su queste osservazioni.
- Eseguire azioni all'interno dell'ambiente di gioco.

L'obiettivo principale di un agente durante l'addestramento è quello di imparare a massimizzare le ricompense cumulative attraverso il compimento di osservazioni e azioni in un Learning Environment. L'addestramento di un agente tramite RL nel toolkit ML-Agents avviene seguendo un ciclo iterativo basato su:

- **Observation:** In questa fase l'agente raccoglie osservazioni nel suo Environment(scena Unity) e le usa come input per l'addestramento. L'agente dopo aver raccolto le osservazioni le invia al trainer che basandosi su di esse, restituisce all'agente un'azione da compiere. Quante e quali osservazioni sono stabilite dal progettista. Le osservazioni vengono ottenute campionando valori dal mondo di gioco, come distanze, campi visivi e altre informazioni rilevanti per l'addestramento. Così come in molti task di Machine Learning la scelta delle osservazioni dovrebbe includere tutte e sole le informazioni rilevanti per lo specifico task di addestramento. La scelta di quali osservazioni dipenderà dal problema specifico che si vuole risolvere. Essenzialmente bisogna pensare quali sono le osservazioni che consentono all'agente di prendere una decisione informata in modo ottimale, senza altre informazioni estranee. Empiricamente si potrebbe anche immaginare a grandi linee quali informazioni servirebbero ad un essere umano per risolvere un determinato problema.
- **Decision:** Dopo aver fornito le osservazioni, l'agente richiede una decisione dal learner, corrispondente all'azione che esso deve effettuare. La richiesta di una decisione avviene utilizzando il componente Unity Decision Requester.
- **Action:** Ad ogni iterazione del Decision Period, l'agente dopo aver inviato al trainer le osservazioni e la richiesta di una decisione, il trainer fornisce all'agente l'azione da intraprendere. Il componente Behavior Parameters consente di configurare le azioni e le osservazioni che l'agente può eseguire nell'ambiente.
- **Reward:** Se l'azione, o la sequenza di azioni, intrapresa dall'agente nell'ambiente è corretta, l'agente otterrà una ricompensa. I reward vengono usati solo in fase di addestramento¹. Generalmente la progettazione delle

¹Le ricompense non vengono utilizzate durante l'inferenza(l'agente utilizza un modello addestrato) e nemmeno durante l'apprendimento per imitazione(Behavioral Cloning).

ricompense si struttura iniziando a inserire ricompense in modo semplice, man mano si aggiungono ricompense quando necessario.

Questo ciclo continuo permette all'agente di apprendere una policy circa ottimale che suggerisca quali azioni portano alla massimizzazione delle ricompense.

3.4.2 Scenario di Gioco Simmetrico

Lo scenario di gioco utilizzato per la sperimentazione è stato progettato per garantire una competizione equa tra due partecipanti. Entrambi i partecipanti sono dotati delle stesse caratteristiche in termini di azioni, osservazioni e ricompense. Uno scenario di gioco simmetrico si definisce come un gioco che assicuri pari opportunità ai giocatori durante la partita.

Il gioco consiste nel consentire ai due giocatori di muoversi liberamente all'interno di uno scenario composto da più stanze. L'obiettivo è colpire l'avversario sparando una palla. Riuscire a colpire l'avversario equivale a vincere la partita. La vittoria di un giocatore rappresenta il fallimento del suo avversario. I giocatori dispongono di un numero limitato di munizioni (palle), il primo che le esaurisce perde automaticamente la partita.

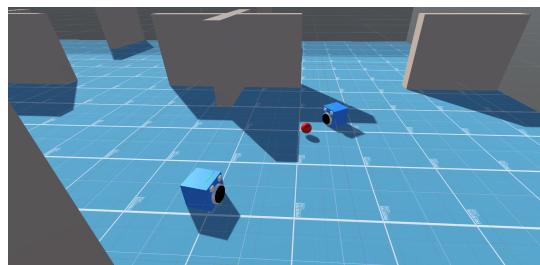


FIGURA 3.3: Scenario di gioco simmetrico

3.4.3 Compatibilità tra Agenti e Giocatori Umani

Lo scenario di gioco è progettato per consentire la partecipazione sia di giocatori umani, tramite strumenti di input e output per interagire con il gioco, sia di agenti addestrati utilizzando algoritmi di RL.

Per i giocatori umani, l'interazione con il gioco avviene attraverso un dispositivo di controllo e uno schermo che visualizza una porzione della scena di gioco, fornendo le informazioni necessarie per agire all'interno della partita. Per gli agenti, invece, le osservazioni e le azioni vengono codificate

e strutturate per essere utilizzate durante il processo di apprendimento. I modelli addestrati tramite il processo di apprendimento possono infine essere usati per controllare(inferenza) i giocatori all'interno di una partita, con la capacità di competere con un altro giocatore agente o un altro giocatore umano.

Poiché l'obiettivo finale è addestrare l'agente a competere contro un altro giocatore o agente in modo realistico, le azioni e, soprattutto, le osservazioni codificate per l'agente devono essere quanto più simili possibile a quelle disponibili per un giocatore umano. Questo approccio garantisce che l'agente apprenda comportamenti e strategie che siano compatibili ed equi rispetto ai giocatori umani. Entrambi i partecipanti, che siano umani o agenti, dovranno imparare a interagire con l'ambiente attraverso osservazioni simili.

Esempio: Fornire all'agente l'informazione sulla posizione assoluta dell'avversario durante la fase di addestramento potrebbe creare uno squilibrio a svantaggio del giocatore umano. Questa osservazione, infatti, non sarebbe accessibile a un giocatore umano, che potrebbe ottenerla solo esplorando attivamente lo scenario.

3.4.4 Osservazioni e Azioni Agente

Osservazioni

Per evitare lo scenario di sbilanciamento delle osservazioni da parte di agenti o giocatori descritto nell'esempio precedente, la soluzione adottata per selezionare le osservazioni dell'agente consiste nella selezione empirica delle osservazioni nello scenario, limitandole esclusivamente a quelle che un giocatore umano potrebbe dedurre osservando lo schermo dal proprio punto di vista durante il gioco. Tra queste si includono, ad esempio, il campionamento del campo visivo, la distinzione dei tipi di oggetti inquadrati nel campo visivo e la stima della velocità dell'avversario. Il resto delle osservazioni impiegate per l'addestramento e i dettagli sulla loro implementazione sono spiegati nella sottosezione del Capitolo dedicato alla progettazione 4.1.2. Identificare le giuste osservazioni informative può aumentare di parecchio la stabilità e la velocità di convergenza di un addestramento a una policy ottimale.

Le osservazioni provenienti dall'ambiente potrebbero presentare scale numeriche diverse. Questa disparità può portare a una convergenza più lenta o addirittura impedire la convergenza dell'algoritmo. La normalizzazione delle osservazioni fornite alla rete neurale è cruciale per garantire un addestramento efficiente e una buona capacità di generalizzazione del modello.

Azioni

Per quanto riguarda le azioni che gli agenti possono compiere, queste corrispondono a quelle descritte in 3.4.2, ovvero sono esattamente le stesse che un giocatore umano potrebbe eseguire utilizzando un controller di gioco. Questo garantisce la condizione di parità, rispettando il principio del gioco simmetrico.

3.4.5 Ricompense Agente

L'assegnazione di ricompense così come per le osservazioni consiste in una progettazione empirica, non ci sono regole precise che funzionano per tutti gli scenari di addestramento. Ogni scenario è un caso a sé stante. Ci sono però alcuni aspetti fondamentali da tenere a mente quando si progetta una funzione di ricompensa.

Bias della Funzione di Ricompensa

La funzione di ricompensa è una funzione che determina come l'agente viene ricompensato per le sue azioni. Se la funzione di ricompensa è affetta da bias, l'agente potrebbe imparare a comportarsi in modo non ottimale per l'ambiente. Ad esempio, se la funzione di ricompensa premia l'agente solo per aver intrapreso azioni che portano a ricompense a breve termine, l'agente potrebbe imparare a ignorare le azioni che portano a ricompense a lungo termine.

È quindi fondamentale progettare con attenzione la funzione di ricompensa, in modo che questa rifletta il comportamento desiderato dall'agente nell'ambiente. Di seguito viene fornito un esempio concreto che evidenzia le problematiche associate a una funzione di ricompensa affetta da bias.

Esempio: Consideriamo l'ambiente di gioco utilizzato nell'esperimento condotto. Se, durante l'addestramento, associamo una penalità ogni volta che l'agente collide con un muro, terminando la partita, e allo stesso tempo imponiamo una penalità distribuita uniformemente rispetto alla durata dell'episodio (per incentivare l'agente a terminare gli episodi più rapidamente), potrebbe emergere un comportamento subottimale. In particolare, se la durata delle partite durante l'addestramento diventasse eccessivamente lunga, l'agente potrebbe imparare a collidere intenzionalmente contro un muro per terminare rapidamente la partita ed evitare la penalità temporale accumulata. Questo

comportamento, pur massimizzando la ricompensa nell'immediato, ignora la possibilità di ottenere una ricompensa cumulativa maggiore cercando di vincere la partita, ovvero sconfiggendo l'avversario.

Come illustrato nell'esempio, l'utilizzo di un numero eccessivo di ricompense e penalità, o un loro cattivo bilanciamento, potrebbe costituire una funzione di ricompensa affetta da bias. L'approccio seguito in questo lavoro prevede di partire con una funzione di ricompensa semplice, aggiungendo complessità solo quando strettamente necessario. In generale, le linee guida per la progettazione delle funzioni di ricompensa suggeriscono di premiare i risultati piuttosto che le azioni che si presume conducano ai risultati desiderati.

Un ultimo aspetto fondamentale è l'intervallo delle ricompense assegnate, che dovrebbero essere comprese (normalizzate) tra $[-1, 1]$. Valori al di fuori di questo intervallo possono compromettere la stabilità dell'addestramento.

3.5 Metodologie di Addestramento

Agli addestramenti effettuati tramite RL sono state affiancate metodologie specifiche, con l'obiettivo di migliorare le prestazioni degli agenti sotto diversi aspetti:

1. **Facilitare la convergenza** a politiche ottimali in tempi più brevi, accelerando il processo di apprendimento.
2. **Ottimizzare l'efficacia** degli agenti nel portare a termine i task, migliorandone la capacità di raggiungere gli obiettivi prefissati.
3. **Garantire robustezza** agli agenti, rendendoli capaci di generalizzare meglio e di adattarsi a nuovi ambienti mai osservati durante la fase di addestramento.

3.5.1 Parallelizzazione

Il toolkit supporta l'addestramento parallelo su più istanze. Parallelizzare gli ambienti di addestramento è una tecnica molto importante che permette di aumentare di molto la velocità di addestramento verso un ottimo locale. Questo approccio consente di migliorare l'efficienza dell'apprendimento grazie a una maggiore diversità di esperienze raccolte (campionamento) dagli agenti in un tempo ridotto.

Con la presenza di molteplici ambienti di addestramento nello scenario di Unity, è fondamentale che gli agenti elaborino informazioni relative/locali piuttosto che globali e che essi agiscano tutti in modo indipendente tra loro.

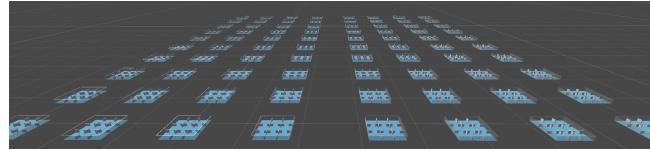


FIGURA 3.4: L’insieme delle istanze parallele che hanno contribuito all’addestramento

3.5.2 Curriculum Learning

Il Curriculum Learning è una metodologia per l’addestramento di modelli di apprendimento automatico che prevede l’introduzione graduale degli aspetti più complessi di un problema. Introdurre una variante troppo complessa dell’ambiente nelle prime fasi del training potrebbe quasi sempre richiedere molte più iterazioni prima di raggiungere una politica ottimale o, peggio, potrebbe bloccare l’agente in politiche subottimali per periodi prolungati(stagnation). Affrontare progressivamente ambienti con complessità crescente permette di guidare l’agente verso politiche più efficaci in maniera più efficiente.

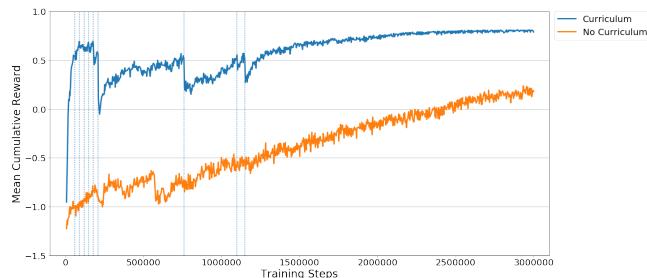


FIGURA 3.5: Differenza in termini di ricompense cumulativa medie rispetto agli step di addestramento

In questa tesi l’addestramento di un agente con l’obiettivo di competere contro un altro agente ha richiesto un approccio progressivo. È stato più utile costruire gradualmente le abilità desiderate, partendo da competenze più generiche e affinando progressivamente la specializzazione dell’agente. Questo approccio si è rivelato particolarmente importante in presenza di più agenti, poiché l’interazione tra agenti aumenta le dinamiche non stazionarie nell’ambiente², rendendo imprevedibili gli esiti delle azioni e ottenendo addestramenti

²Nelle sottosezioni successive il concetto di stazionarietà viene spiegato più specificamente.

instabili. Inoltre, complicare progressivamente gli scenari generalmente aiuta a migliorare la capacità del modello di generalizzare.

In questo lavoro, l'adozione del Curriculum Learning si è dimostrata una tecnica efficace per guidare l'agente verso una politica ottimale in tempi molto più rapidi rispetto a un addestramento diretto e privo di progressività.

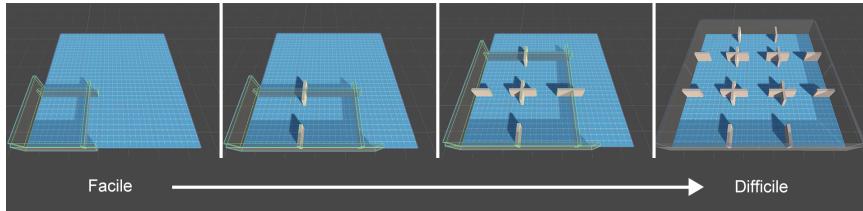


FIGURA 3.6: Con il Curriculum Learning lo scenario di training si complica progressivamente con l'avanzare degli step totali di addestramento

3.5.3 Randomizzazione dello Scenario di Addestramento

Un agente addestrato in un ambiente specifico potrebbe non essere in grado di generalizzare rispetto a piccole modifiche o variazioni dell'environment. Questo significa che l'agente in fase di inferenza potrebbe non essere capace di prendere decisioni ottimali in scenari simili ma diversi da quello usato per l'addestramento. Introdurre fattori di casualità negli scenari è essenziale per permettere all'addestramento di costruire un modello robusto capace di generalizzare ed evitare l'overfitting. Un modello ben generalizzato è in grado di riconoscere pattern simili rispetto alle esperienze passate e di prendere decisioni ottimali anche in presenza di stati mai osservati durante l'addestramento.

Per aumentare la generalizzazione del modello, le situazioni di partenza di ogni episodio sono programmate per essere sempre differenti. A tal fine, gli agenti partono da posizioni e rotazioni casuali a ogni inizio di episodio.

3.6 Algoritmi di Addestramento

Come algoritmo di base per il RL è stato utilizzato il Proximal Policy Optimization (PPO), fornito dal toolkit ML-Agents e descritto nella Sezione 2.6. Tuttavia, per raggiungere l'obiettivo di creare agenti realistici e competitivi, il PPO è stato integrato con ulteriori tecniche. Queste hanno consentito di guidare gli agenti verso politiche in grado di simulare dinamiche di partite tra esseri umani, offrendo un'esperienza di gioco realistica e al tempo stesso capace di competere efficacemente contro giocatori umani.

Le sezioni successive approfondiranno ciascuna metodologia, evidenziando il loro ruolo nel processo di addestramento e l'impatto sulle capacità finali degli agenti.

3.6.1 Curiosity Module

In ambienti in cui l'agente può ricevere ricompense rare o poco frequenti (ricompense sparse), è possibile che, durante un episodio, l'agente non ottenga alcuna ricompensa su cui basare il processo di addestramento, specialmente se si trova in un ambiente complesso. L'integrazione del modulo curiosità introduce **ricompense intrinseche** che possono aiutare l'agente a esplorare meglio l'ambiente, migliorando così l'efficacia del suo apprendimento [10].

Il Curiosity Module è un modulo che addestra due reti:

- un *modello inverso*, che prende l'osservazione corrente e successiva dell'agente, le codifica e utilizza la codifica per prevedere l'azione intrapresa tra le due osservazioni.
- un *modello avanzato*, che prende l'osservazione e l'azione correnti codificate e prevede l'osservazione successiva codificata.

La loss del *modello inverso* è l'errore commesso nel prevedere l'azione. La loss del *modello avanzato*, la differenza tra le osservazioni codificate previste e quelle effettivamente osservate, viene utilizzata come ricompensa intrinseca. Quindi, più il modello osserva nuove situazioni che non aveva previsto, maggiore sarà la sua ricompensa.

Il Curiosity Module incentiva quindi l'agente a prediligere l'esplorazione, permettendogli di esplorare il mondo più di quanto farebbe altrimenti e, di conseguenza, aiutare l'agente ad ottenere ricompense estrinseche (quelle fornite dall'ambiente) in misura maggiore. Nel campo del RL, i ricercatori hanno dedicato molta attenzione allo sviluppo di sistemi efficaci per fornire ricompense intrinseche agli agenti, dotandoli di motivazioni analoghe a quelle osservabili negli esseri viventi in natura.

Un esempio spesso utilizzato è quello di un bambino che, grazie alla sua voglia di conoscere il mondo, esso ha maggiori probabilità di comprenderlo meglio. L'idea alla base dell'esplorazione guidata dalla curiosità è quella di replicare questo tipo di motivazione negli agenti artificiali. Per maggiori dettagli, si rimanda all'articolo di riferimento [10].

3.6.2 Self-Play

Nella maggior parte della teoria esistente nel RL si applica solo lo scenario in cui l’agente gioca in un environment statico. Le sole tecniche di RL generalmente non sono sufficienti per addestrare un agente in grado di sfidare giocatori umani in modo soddisfacente. In altre parole, un agente, per imparare a competere contro un altro agente, necessita della presenza di una supervisione diretta da parte di agenti o giocatori umani che lo sfidino e contribuiscano durante la fase di apprendimento. Addestrare un agente capace di competere contro un giocatore umano può essere molto complicato nella pratica, poiché nel caso di supervisione umana questa richiederebbe la partecipazione di giocatore umano durante la fase di addestramento in milioni di partite, che non potrebbero essere né parallelizzate né accelerate.

Il Self-Play è una tecnica di addestramento che permette ad un agente ad imparare in un ambiente multi-agente, ottenendo ottimi risultati [16]. La funzionalità di Self-Play fornita dal toolkit ML-Agents consente di addestrare agenti in sfida tra loro in giochi simmetrici tramite RL. Il Self-Play utilizza le versioni attuali e passate dell’agente come avversari, consentendo all’agente di sviluppare le proprie capacità utilizzando algoritmi di RL.

Questa tecnica offre un environment di apprendimento simile a quello utilizzato dagli esseri umani per strutturare la competizione. Ad esempio, un giocatore umano trarrebbe maggiore beneficio allenandosi con un avversario di abilità simile, poiché un avversario troppo debole o troppo forte limiterebbe le opportunità di crescita. Di conseguenza, l’agente, una volta raggiunto un livello di abilità sufficiente, progredisce sfidando un avversario di livello superiore. Il Self-Play può essere considerato una forma di curriculum learning, in cui l’agente si allena sfidando versioni di sé stesso come avversari di forza sempre crescente.

Il Self-Play è una tecnica responsabile di molti risultati contemporanei nel RL. Nei risultati ottenuti utilizzando il Self-Play, i ricercatori sottolineano spesso che gli agenti scoprono strategie in grado di sorprendere gli umani [3].

RL negli Adversarial Games

In un tradizionale problema di RL, un agente cerca di apprendere una politica che massimizzi la ricompensa cumulativa. In genere, l’agente deve imparare a gestire aspetti come i vincoli imposti dall’environment e l’influenza delle proprie azioni, al fine di ottenere una ricompensa elevata. In altre parole, l’agente si

confronta con le dinamiche dell'environment per visitare le sequenze di stati che offrono il massimo rendimento.

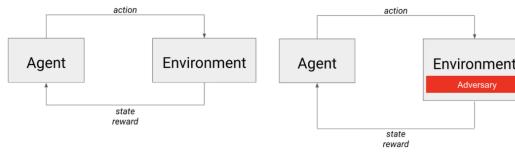


FIGURA 3.7: Confronto di Problema Tradizionale di RL e un Problema di RL in un Adversarial Game

In un Adversarial Game, l'agente non si confronta solo con le dinamiche dell'environment ma anche con un altro agente intelligente. Si può pensare all'agente avversario come fosse un tutt'uno con l'environment, poiché le sue azioni influenzano direttamente lo stato successivo che l'agente osserva. Nell'immagine 3.7, a sinistra è rappresentato il tipico scenario di RL. A destra, lo scenario di apprendimento include la competizione con un avversario che, dal punto di vista dell'agente, è considerato parte dell'environment. L'addestramento con Self-Play permette di addestrare comportamenti di agenti complessi, sfidando istanze di se stesso senza usare un algoritmo multi-agente ma con semplici algoritmi di RL.

Considerazioni Pratiche e Configurazione

Le potenzialità del Self-Play sono interessanti ma esso introduce due problematiche che, se non correttamente gestite, possono peggiorare i risultati dell'apprendimento:

- **Overfitting a uno stile di gioco specifico:** Durante un addestramento con Self-Play, l'agente potrebbe adattarsi troppo a un particolare avversario o stile di gioco, sviluppando strategie che funzionano bene solo contro un tipo specifico di avversario. Se questi avversari non sono sufficientemente vari, l'agente non sviluppa strategie generalizzate, diventando incapace di affrontare avversari con stili diversi. Per risolvere questa problematica il Self-Play del toolkit introduce un buffer di politiche passate. L'agente si allena quindi contro versioni precedenti di se stesso, garantendo una maggiore varietà di avversari.
- **Instabilità dovuta alla non stazionarietà della funzione di transizione:** Durante l'addestramento tramite self-play, l'environment cambia costantemente perché gli avversari(copie o versioni dell'agente stesso)

migliorano o modificano il loro comportamento nel tempo. Se questi avversari variano la loro versione troppo spesso, questo porterà ad funzione di transizione non stazionaria, dove le stesse azioni dell'agente non portano agli stessi risultati nel tempo. Generalmente gli algoritmi di RL assumono che l'environment sia stazionario, ovvero che le azioni dell'agente producano sempre risultati prevedibili. Con il self-play questa assunzione viene violata, rendendo difficile per l'agente la convergenza verso una politica ottimale. La problematica legata al rischio di un addestramento instabile, si risolve facendo competere l'agente contro la versione di un certo avversario(se stesso) per periodi più lunghi, stabilizzando l'apprendimento.

Nel Capitolo 4 dedicato alla progettazione sono illustrate le decisioni prese per gestire gli aspetti relativi al Self-Play.

3.6.3 Imitation Learning

L'utilità principale dell'Imitation Learning è quella di fornire all'agente una dimostrazione del comportamento desiderato, anziché farlo apprendere esclusivamente tramite tentativi, errori e una ricompensa estrinseca. Questo approccio utilizza coppie di osservazioni e azioni estratte da una dimostrazione per apprendere una politica.

L'apprendimento per imitazione può essere impiegato autonomamente o in combinazione con il RL. Quando utilizzato da solo, permette di apprendere un comportamento specifico, come uno stile particolare per risolvere un compito, ignorando le ricompense estrinseche. Se combinato con l'apprendimento per rinforzo, può ridurre drasticamente il tempo necessario per addestrare l'agente a risolvere un ambiente complesso.

Il toolkit ML-Agents offre questi strumenti sia per apprendere direttamente dalle dimostrazioni, sia per utilizzarle come supporto per accelerare l'addestramento basato sulle ricompense (RL).

Se lo scopo iniziale dell'approccio basato sull'Imitation Learning è quello di aiutare l'agente ad apprendere una policy tramite dimostrazioni registrate da esseri umani, lo scopo di questa tesi è dimostrare che una fusione appropriata tra Reinforcement Learning e Imitation Learning, nel contesto degli Adversarial Games , può consentire l'addestramento di agenti capaci di imitare parzialmente specifici comportamenti presenti nelle dimostrazioni (partite giocate da esseri umani), pur mantenendo un buon livello di competitività.

Gli agenti addestrati con RL integrato da queste metodologie hanno inoltre migliorato la percezione di realismo e di presenza umana nelle partite, come emerso dai risultati dei test utente riportati nella Sezione 5.5.

Campionamento dell’Insieme di Dimostrazioni

Le dimostrazioni da fornire al processo di addestramento possono essere registrate tramite uno specifico componente fornito da ML-Agents dall’editor di Unity e salvate come asset(file). Per applicare le tecniche di Imitation Learning, sono stati campionati 217 episodi all’interno dell’ambiente di gioco progettato per le sperimentazioni. Durante questi episodi, due giocatori umani si sono sfidati in modo naturale, cercando di sconfiggersi a vicenda. Il campionamento del set di dimostrazioni consiste nel registrare le azioni, le osservazioni e le ricompense generate durante le fasi di gioco, salvandolo in un file. Tali elementi corrispondono esattamente a quelli definiti durante la progettazione dell’agente.

Il set di dimostrazioni ottenuto è stato successivamente fornito al processo di apprendimento dell’Imitation Learning, fase descritta in dettaglio nelle sottosezioni successive.

Generative Adversarial Imitation Learning(GAIL)

GAIL, o Generative Adversarial Imitation Learning, utilizza un approccio simile alle reti neurali avversarie [4] per premiare l’agente quando il suo comportamento è simile a quello osservato in un set di dimostrazioni.

In questo framework, una seconda rete neurale, chiamata discriminatore, viene addestrata a distinguere se una coppia osservazione/azione proviene da una dimostrazione o è generata dall’agente. Il discriminatore, quindi, esamina nuove osservazioni/azioni e assegna una ricompensa in base alla somiglianza con le dimostrazioni fornite [6].

Durante ogni fase di addestramento:

- L’agente cerca di massimizzare la ricompensa assegnata dal discriminatore, imparando così a imitare sempre più accuratamente le dimostrazioni.
- Il discriminatore viene addestrato a migliorare la sua capacità di distinguere tra dimostrazioni e osservazioni/azioni dell’agente.

Questo processo competitivo porta a un ciclo continuo: mentre l’agente migliora nel replicare le dimostrazioni, il discriminatore diventa sempre

più severo, costringendo l'agente a perfezionarsi ulteriormente per "ingannarlo".

Negli esperimenti condotti, GAIL ha dimostrato di essere in grado di apprendere una policy che produce stati e azioni simili a quelli osservati nelle dimostrazioni, imparando ad emulare comportamenti specifici.

Il segnale di ricompensa fornito da GAIL è stato combinato con un segnale di ricompensa estrinseco (ricompense ambiente), guidando ulteriormente il processo di apprendimento. Questa configurazione ha consentito di addestrare agenti in grado di generalizzare efficacemente, evitando di limitarsi a gestire situazioni specifiche. Gli agenti hanno appreso comportamenti complessi, come attacchi indiretti, evasione da situazioni di attacco diretto, utilizzo di ripari per nascondersi, ed esplorazione rapida dell'ambiente 5.4, mantenendo al contempo un buon livello di competitività.

Come nota finale su GAIL, la capacità di imitare le dimostrazioni fornite introduce un Survivor Bias nel processo di apprendimento[8]. Questo significa che, per ottenere ricompense positive basate sulla somiglianza con le dimostrazioni, l'agente è incentivato a rimanere in vita il più a lungo possibile.

Per mitigare questo effetto, in linea con le linee guida empiriche, è stato scelto di utilizzare segnali di ricompensa forniti da GAIL che favorissero l'imitazione delle dimostrazioni senza prevalere sulle ricompense estrinseche. In questo modo, si è evitato il rischio che l'agente perdesse di vista l'obiettivo principale del task, ovvero vincere la partita.

La configurazione dei parametri è stata quindi progettata per bilanciare un trade-off tra competitività ed imitazione, combinando le ricompense estrinseche tipiche del Reinforcement Learning con quelle intrinseche fornite da GAIL, al fine di ottenere un buon livello di imitazione dei comportamenti osservati nelle dimostrazioni e garantendo un adeguato livello di competitività.

Behaviour Cloning

Behaviour Cloning (BC) è una tecnica relativamente semplice che si basa sull'addestramento della policy dell'agente per imitare esattamente le azioni mostrate nel set di dimostrazioni. Tuttavia, poiché BC non è in grado di generalizzare oltre gli esempi forniti, tende a funzionare meglio in presenza di dimostrazioni che coprono quasi tutti gli stati che l'agente può incontrare. In alternativa, BC può essere utilizzato in combinazione con GAIL e/o con una ricompensa estrinseca per migliorare le sue prestazioni.

A causa della sua limitata capacità di generalizzazione e a causa dalla limitata quantità di dimostrazioni, BC ha prodotto risultati poco peggiori delle altre configurazioni di addestramento negli esperimenti condotti. Per questo motivo, è stato necessario ridurre notevolmente i livelli di intensità della sua applicazione, poiché tendeva a rendere l'agente inefficiente per il task, risultando complessivamente poco competitivo.

3.7 Sperimentazione

La sperimentazione è stata strutturata in due fasi principali. Nella prima fase, gli algoritmi sono stati testati durante la progettazione del gioco simmetrico. Sono state esplorate diverse configurazioni di iperparametri per l'addestramento, seguendo le linee guida fornite. In particolare, sono stati analizzati i singoli algoritmi di RL, come il semplice RL base, il Self-Play e l'Imitation Learning. L'obiettivo è stato identificare fin da subito le problematiche che rendevano gli addestramenti instabili o poco efficaci. Questa fase ha consentito di costruire uno scenario di addestramento solido e di ottimizzare gli iperparametri, garantendo risultati soddisfacenti.

Nella seconda fase, le tecniche collaudate nella prima fase sono state combinate per trovare un equilibrio ottimale tramite la configurazione degli iperparametri, tra capacità di imitazione e abilità nel completare il task. Trovare questo equilibrio è stato necessario per supportare la tesi.

3.7.1 Configurazioni Addestramenti

Sulla base dei risultati ottenuti nella prima fase, sono state selezionate tre configurazioni differenti di addestramento per il confronto, con l'obiettivo di evidenziare le differenze tra le varie modalità di training, sia imitative che non imitative. Le configurazioni scelte sono descritte nella sezione 5.3.

Analisi dei Risultati degli Addestramenti

Durante l'analisi dei risultati, sono stati valutati gli esiti ottenuti dalle tre configurazioni sopra descritte. La valutazione si è concentrata sulla stabilità degli addestramenti e sulle differenze nei risultati tra le diverse tecniche, cercando di spiegare le motivazioni alla base delle variazioni osservate.

3.8 Test Utente

La tesi mira a dimostrare che gli agenti addestrati siano in grado di competere in modo realistico e sfidante. Per valutare aspetti non misurabili attraverso metriche tecniche, è stato condotto un test utente. Durante il test, un gruppo di partecipanti ha giocato 15 partite, sfidando gli agenti addestrati nelle tre configurazioni e un giocatore umano, senza sapere contro chi stessero giocando. Al termine delle partite, hanno espresso valutazioni sulla difficoltà, sul realismo percepito e sulla soddisfazione complessiva del grado di sfida del gioco. I dettagli sui risultati dei test utenti sono riportati nella sezione 5.5.

Capitolo 4

Progettazione

4.1 Ciclo di Apprendimento Agente

L’implementazione del ciclo di apprendimento consiste nell’implementare uno script dedicato all’agente e nell’aggiungere i componenti forniti dal pacchetto ML-Agents al gameobject dell’agente. Questi componenti sono necessari per formalizzare le osservazioni e le azioni che verranno proposte dal modello in fase di apprendimento.

4.1.1 Definizione Features Agente

La definizione delle azioni e delle osservazioni di un agente viene effettuata tramite la configurazione di due componenti sul *GameObject* che rappresenta l’agente. È importante notare che questi componenti permettono solo la configurazione di tali aspetti; l’implementazione vera e propria dell’agente utilizzerà questa configurazione per mandare osservazioni al learner e interagire con l’ambiente.

Il componente *BehaviourParameters*, in fase di esecuzione, genera gli oggetti policy dell’agente in base alle impostazioni specificate tramite Editor(parametri componente). In altre parole, componente permette di settare quali osservazioni è in grado di fornire e quali azioni è in grado di compiere l’agente, durante il ciclo di apprendimento.

Osservazioni Agente

Il parametro *SpaceSize* consente di definire il numero di osservazioni fornite al processo di training a ogni academy step. Nel contesto di questa sperimentazione, il valore è stato impostato a 12. Nella sottosezione 4.1.2 dedicata all’implementazione dello script dell’agente viene illustrato in dettaglio il codice attraverso il quale le osservazioni vengono fornite al learner.

Azioni Agente

Questo parametro permette di definire sia azioni continue che discrete. Nel caso di azioni discrete, è possibile configurare gruppi di azioni discrete, detti *DiscreteBranches*, che l'agente può compiere. Ogni branch discreto contiene un numero finito di azioni disponibili.

Per questa sperimentazione sono stati creati tre branch discreti, ognuno rappresentante un gruppo di azioni specifiche fornite dal learner all'agente presente nell'ambiente. Durante l'addestramento, ciascun branch può assumere un valore che corrisponde all'azione da eseguire in un determinato academy step. Le azioni disponibili per l'agente sono definite come segue:

- Il primo branch corrisponde all'azione di rotazione dell'agente nell'ambiente e può assumere 3 valori finiti che corrispondono a 3 azioni differenti sull'asse di rotazione dell'agente(asse Y):
 - Valore 0: l'agente non ruota in nessuna direzione
 - Valore 1: l'agente ruota a destra
 - Valore 2: l'agente ruota a sinistra
- Il secondo branch corrisponde ai movimenti in avanti e indietro dell'agente:
 - Valore 0: l'agente resta fermo
 - Valore 1: l'agente si muove in avanti
 - Valore 2: l'agente si muove all'indietro
- L'ultimo branch corrisponde alle azioni relative allo sparare le palle:
 - Valore 0: l'agente non spara
 - Valore 1: l'agente spara



FIGURA 4.1: Parametri componente Behaviour Parameters

Richiesta delle Decisioni

Come già spiegato, nella fase *Decision*, il componente *DecisionRequester* serve a richiedere una decisione. Il parametro *DecisionPeriod* indica il numero di passi dell’Academy che devono trascorrere tra due richieste di decisione consecutive. L’Academy è il processo che gestisce il ciclo di simulazione nell’environment di ML-Agent. Ogni Academy Step rappresenta l’unità di tempo simulata all’interno dell’environment.

- Ogni volta che trascorre il numero di Academy Steps specificato dal Decision Period, l’agente:
 - Raccoglie osservazioni dall’environment
 - Le invia al modello per calcolare l’azione da eseguire
 - L’agente riceve un’azione da seguire
- Durante gli Academy steps intermedi (tra una decisione e l’altra):
 - l’agente continua a eseguire l’ultima azione. Questo comportamento di default è attivo tramite il parametro booleano *TakeActionsBetweenDecisions*

La configurazione del parametro del *DecisionPeriod* può essere settata sperimentalmente nel caso si desideri che l’agente mantenga la stessa azione per un certo numero di passi consecutivi, ad esempio come continuare a camminare in una direzione. Nel caso di simulazioni in tempo reale dove l’agente agisce continuamente in un environment, è fondamentale che richieda una decisione a intervalli regolari. In altri giochi basati su turni sarebbe appropriato invece richiedere una decisione solo ad ogni fine turno.

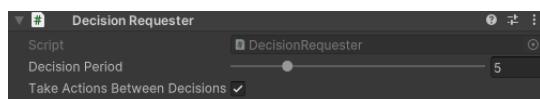


FIGURA 4.2: Parametri componente Decision Requester

4.1.2 Implementazione dello Script dell’Agente

Uno script contenente la classe dell’agente definisce tutti i metodi essenziali per l’addestramento, inclusi quelli necessari per fornire le osservazioni e eseguire le azioni ricevute dal learner. Per implementare la comunicazione con il learner, si estende la classe *Agent* (fornita dal toolkit), che consente di ereditare e sovrascrivere (override) i metodi necessari.

Implementazione delle Osservazioni

Il modo per raccogliere e inviare le osservazioni al trainer avviene riscrivendo il metodo *CollectObservations*. Le osservazioni vengono raccolte sotto forma di vettore (configurato nel componente Unity *BehaviourParameters*) ad ogni academy step. Queste includono:

- **Munizioni:** La quantità di munizioni a disposizione dell'agente/giocatore, normalizzata rispetto al numero massimo di munizioni iniziali. Questa osservazione fornisce al learner informazioni sul numero di munizioni residue durante l'episodio, dato che la partita può essere persa se l'agente esaurisce le munizioni. L'osservazione contribuisce a informare il learner che un valore in diminuzione è associato a un avvicinamento alla sconfitta, che comporta una penalità.
- **Ultima posizione dell'avversario:** La posizione dell'avversario è nota all'agente solo quando esso entra nel campo visivo dell'agente. Se l'avversario esce dal campo visivo, l'osservazione resta fissa all'ultimo valore rilevato, simulando una memoria temporanea. Questa informazione, combinata con la *posizione dell'agente*, può aiutare l'agente ad apprendere strategie di ricerca, indirizzando l'agente a cercare l'avversario nei pressi della posizione in cui è stato perso di vista.
- **Posizione agente:** La posizione locale dell'agente all'interno dell'environment di addestramento. Questa osservazione può aiutare l'agente a orientarsi, specialmente se combinata con informazioni relative alla posizione dell'avversario.
- **Direzione dell'avversario:** La direzione dell'agente avversario, normalizzata (diviso 360) è fornita solo quando l'avversario è nel campo visivo dell'agente. Se l'avversario non è inquadrato, il valore dell'osservazione fornita al learner è pari a -1.
- **Direzione agente:** La direzione dell'agente anch'essa normalizzata. Questa informazione, insieme alla direzione dell'avversario, è stata fornita con l'obiettivo empirico di aiutare l'agente con l'esperienza a comprendere che attaccare alle spalle dell'avversario è meno rischioso e generalmente più conveniente rispetto a un attacco frontale.
- **Vettore velocità dell'agente:** È un vettore tridimensionale che rappresenta direzione il cui modulo ne misura la velocità dell'agente. Questa

informazione aiuta l’agente a imparare a sparare le palle, anticipando le traiettorie in funzione della velocità e della direzione.

- **Vettore velocità dell’avversario:** Questo vettore è fornito solo quando l’agente inquadra l’avversario nel suo campo visivo. Combinando questa informazione con il vettore velocità dell’agente, il learner può imparare ad anticipare le traiettorie di tiro in base alla direzione e alla velocità dell’avversario. Questo comportamento imita quello di giocatori umani, che utilizzano la percezione della direzione e della velocità dell’avversario per prevedere e regolare le linee di tiro.

```
Guidance
public override void CollectObservations(RaycasterSensor sensor) {
    sensor.AddObservation(observation: agentAmmo / GetMaxAgentAmmo()); // ammagine normalizzata
    // rileva direzione agente avversario se nel campo visivo
    OppositeAgentInformation oppositeAgentInformation = DetectsEnemyAgent();

    // posizione degli agenti
    sensor.AddObservation(observation: new Vector3(gameObject.transform.localPosition.x, y: gameObject.transform.localPosition.z));
    // posizione dell’agente avversario
    sensor.AddObservation(observation: oppositeAgentInformation.agentLocalPosition);
    sensor.AddObservation(observation: oppositeAgentInformation.oppositeAgentInformation.agentLocalPosition);

    // direzione agente
    sensor.AddObservation(observation: gameObject.transform.localEulerAngles.y / 360.0f);
    // ultima direzione neutra normalizzata
    sensor.AddObservation(observation: oppositeAgentInformation.agentDirection / 360.0f);

    // ultima velocità dell’agente avversario
    sensor.AddObservation(observation: oppositeAgentInformation.agentVelocity);
    // velocità dell’agente
    sensor.AddObservation(observation: gameObject.GetComponent().velocity);
}
```

FIGURA 4.3: Metodo per fornire le osservazioni dell’agente al learner

Per raccogliere le informazioni sul campo visivo, ho avuto a disposizione diverse opzioni, come l’uso di un flusso di pixel da fornire a una rete neurale convoluzionale (CNN). Tuttavia, questa scelta avrebbe comportato un significativo aumento dei tempi di addestramento. Considerata la semplicità delle dinamiche di gioco, è stato preferito l’utilizzo di un componente fornito dal toolkit ML-Agents, chiamato *RayPerceptionSensor3D*.

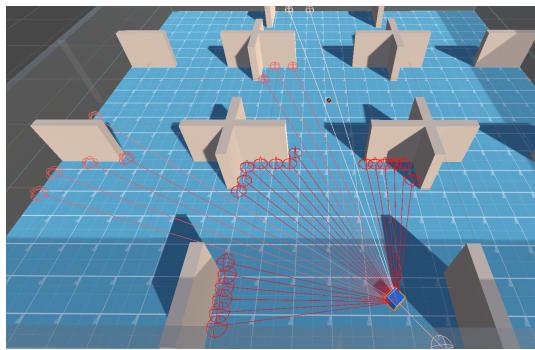


FIGURA 4.4: Debug grafico del componente *RayPerceptionSensor3D*, usato per raccogliere osservazioni sul campo visivo

Questo componente consente di configurare un numero n di raggi distribuiti su un campo visivo. Ogni raggio misura la distanza dall’oggetto con cui collide

e identifica il tag associato. Il *RayPerceptionSensor3D* si è dimostrato utile per approssimare il campo visivo dell'agente, fornendo al learner informazioni relative alla profondità dell'ambiente "scannerizzato". Tutte le osservazioni legate al campo visivo vengono raccolte e inviate al learner a ogni academy step.

Implementazione delle Azioni

Nel metodo *OnActionReceived*, il modello fornisce un buffer in formato vettoriale contenente tutte le azioni da eseguire. Ogni valore nel buffer rappresenta l'azione da compiere sul determinato branch, ma il modello non conosce il significato semantico di tali azioni, come ad esempio "spostarsi in una direzione". Il modello si limita a suggerire le azioni in base alle esperienze apprese (osservazioni, azioni e ricompense). È responsabilità del progettista definire quali azioni corrispondono ai valori restituiti dal modello.

```

0 references
public override void OnActionReceived(ActionBuffers actions) {
    // Ottieni gli input
    int moveYRaw = actions.DiscreteActions[ index: 1]; // Movimento avanti/indietro
    int moveY = 0;
    int rotateRaw = actions.DiscreteActions[ index: 0]; // Rotazione
    int rotate = 0;
    int shootRaw = actions.DiscreteActions[ index: 2]; // Sparo

    // Mappa i valori dell'input
    if(shootRaw == 0) {
    } else if(shootRaw == 1 && agentCannon.CanShoot())...
    if(rotateRaw == 0) {
        rotate = 0;
    } else if(rotateRaw == 1) {
        rotate = 1;
    } else if(rotateRaw == 2) {
        rotate = -1;
    }

    if(moveYRaw == 0) {
        moveY = 0;
    } else if(moveYRaw == 1) {
        moveY = 1;
    } else if(moveYRaw == 2) {
        moveY = -1;
    }
}

```

FIGURA 4.5: Estrazione delle azioni ricevute dal learner per ogni branch

In questo caso, le informazioni sulle azioni da eseguire su ciascun branch vengono raccolte e mappate in azioni concrete da compiere nell'environment di gioco.

Ogni giocatore o agente può sparare un colpo con una frequenza di fuoco predefinita, il che significa che l'azione di sparare può essere eseguita solo una volta per intervallo di tempo specificato. Per implementare questa limitazione, viene applicata un'operazione di masking sull'azione di sparo,

associata al branch che rappresenta questa azione. Il masking disabilita temporaneamente l’azione di sparo al learner fintanto che il metodo *CanShoot()*¹ restituisce false.

Implementazione Euristica per il Testing dell’Environment

Il metodo *Heuristic()* in Unity ML-Agents sovrascrive le azioni suggerite dal learner in fase di apprendimento (o dal file di inferenza del modello .onnx) con input esterni, come tastiera, mouse o controller. Per avviare la simulazione in modalità Heuristic, è necessario configurare il parametro *BehaviourType* su *HeuristicOnly* nel componente *BehaviourParameters*. Questo consente di trasferire nel mondo di gioco le azioni eseguite da uno o più giocatori reali.

Questa implementazione è stata essenziale per raccogliere le azioni eseguite da esseri umani durante la registrazione degli episodi utilizzati per addestrare i modelli imitativi. Inoltre, l’implementazione è stata impiegata per testare e verificare il corretto funzionamento delle azioni, dei rewards e delle condizioni di terminazione dell’episodio, permettendo di testare il funzionamento previsto.

Implementazione Fine Episodio

L’episodio termina quando uno degli agenti colpisce l’avversario, quando uno di essi collide con i muri dello scenario di addestramento, oppure quando uno dei due giocatori esaurisce le munizioni.

Implementazione Ripristino dell’Episodio

Quando un episodio di addestramento termina, lo stato dell’environment viene ripristinato per consentire l’avvio di un nuovo episodio. Questo processo avviene sovrascrivendo il metodo *OnEpisodeBegin()*, all’interno del quale vengono reinizializzate tutte le variabili dell’environment che rappresentano lo stato dell’episodio.

Ogni palla sparata da un giocatore o un agente è rappresentata da un’istanza contenuta in un array. Al termine di ogni episodio, tutte le palle presenti nell’array vengono disabilitate.

La mancata reinizializzazione corretta di tutte le variabili può compromettere l’intero processo di addestramento, poiché eventuali informazioni residue degli episodi precedenti potrebbero influenzare quelli successivi, rendendo l’addestramento instabile o inefficace.

¹Il metodo *CanShoot()* restituisce true o false in base alla frequenza di fuoco e all’intervallo di tempo, consentendo o meno l’esecuzione dell’azione.

4.2 Meccaniche di gioco

Come già detto nell'implementazione delle azioni, le azioni dell'agente che siano input forniti da controller o azioni fornite dal learner vengono convertite in movimenti che l'agente effettua nel mondo di gioco.

```
// Calcola la direzione del movimento e della rotazione
Vector3 dirToGo = transform.forward * moveY;
Vector3 rotateDir = transform.up * rotate;

// Applica la rotazione
if(moveY > 0 || moveY == 0) {
    transform.Rotate(axis, rotateDir, angle: Time.deltaTime * agentRotationSpeed);
} else { // in retro marcia viene applicata una rotazione inversa
    transform.Rotate(axis, rotateDir, angle: Time.deltaTime * agentRotationSpeed * -1);
}

// Applica il movimento basato su AddForce
Rigidbody rb = gameObject.GetComponent<Rigidbody>();
rb.AddForce(force: dirToGo * agentMoveSpeed, mode: ForceMode.VelocityChange);
```

FIGURA 4.6: Movimenti del giocatore/agente

Le palle sparate da ogni agente non vengono istanziate a runtime, poiché l'operazione di istanziazione di un nuovo GameObject comporta un ritardo dovuto al recupero dei dati sulla mesh e sull'asset. Considerando la quantità totale di episodi (centinaia di migliaia), tali ritardi accumulati potrebbero rallentare drasticamente le prestazioni dell'addestramento, introducendo colli di bottiglia e costringendo il learner ad attendere l'inizializzazione del gameobject in Unity.

Per ovviare a questo problema e sfruttare appieno le risorse senza compromettere le prestazioni in fase di addestramento, tutte le palle di cannone vengono istanziate una sola volta in una pool (una coda circolare di GameObject) allo start dell'addestramento. Quando un academy episode termina, tutte le palle di cannone vengono disabilitate. Durante l'addestramento, le palle vengono abilitate e posizionate nel punto di sparo solo quando necessario, seguendo un ordine FIFO (First In, First Out), cioè la prima palla istanziata è la prima a essere riutilizzata.

Le palle, una volta abilitate, eseguono uno spostamento nella direzione frontale con una certa velocità all'interno del metodo *FixedUpdate()*. Questo metodo è stato scelto perché è basato sulla fisica, quindi è fondamentale per rilevare correttamente le collisioni. La moltiplicazione per *Time.fixedDeltaTime* garantisce che il movimento dell'oggetto (o dell'agente) sia indipendente dal frame rate, assicurando un comportamento coerente anche quando il frame rate varia.

Questo accorgimento è particolarmente importante durante la fase di addestramento, dove il tempo delle simulazioni viene accelerato. In tali condizioni, è essenziale che le palle si spostino in modo coerente, indipendentemente dalla velocità della simulazione o dal frame rate.

```
Unity Message | 0 references
void FixedUpdate() {
    // Movimento costante del proiettile
    Vector3 moveDirection = transform.forward * speed * Time.fixedDeltaTime;
    rb.MovePosition(position: rb.position + moveDirection);
}
```

FIGURA 4.7: Comportamento palla

Quando la palla collide con qualsiasi oggetto dello scenario, viene disabilitata.

Implementazione delle Ricompense e delle Penalità

Come descritto nella sottosezione 3.4.5, le ricompense e le penalità sono state progettate evitando funzioni di ricompensa affette da bias. Tuttavia, le ricompense devono comunque essere utili a modellare i comportamenti desiderati per il raggiungi.

Ricompense:

- **Colpire l'avversario:** L'agente riceve una ricompensa ogni volta che riesce a colpire l'avversario.

Penalità

- **Penalità sul tempo:** Per incentivare l'agente a completare l'episodio rapidamente, viene applicata una penalità uniforme distribuita sul numero massimo di step dell'episodio. Se l'agente non completa il task entro il limite massimo di step, riceve la penalità massima relativa.

```
// Penalizza per il tempo trascorso
float timePenalty = -1f / MaxStep;
AddReward(increment: timePenalty);
```

FIGURA 4.8: Penalità uniforme sul tempo

- **Contatti indesiderati:** Per disincentivare l'agente dal collidere con i muri o con il giocatore avversario, viene applicata una penalità in entrambi i casi. Questa penalità aiuta l'agente a imparare a evitare tali contatti.

```

Unity Message | references
private void OnCollisionEnter(Collision collision) {
    if(collision.gameObject.tag == "wall") {
        // penalità per la collisione con un muro
        if(!gameEnvironmentController.wallMatchFailDisabled) {
            gameEnvironmentController.EndEnvironmentEpisodeWithOneLose(
                loserAgent: gameObject.GetComponent<BehaviorParameters>().TeamId,
                penalty: -10f
            );
        }
    } else if(collision.gameObject.tag == "agent") {
        // penalità per la collisione con un altro agente
        gameEnvironmentController.EndEnvironmentEpisodeWithOneLose(
            loserAgent: gameObject.GetComponent<BehaviorParameters>().TeamId,
            penalty: -10f
        );
    }
}

```

FIGURA 4.9: Penalità per contatti con muri o giocatori avversari

- **Colpi mancati:** L’agente riceve una penalità (normalizzata sul numero massimo di colpi) ogni volta che spara un colpo che non va a segno. Questa penalità incentiva l’agente a:
 - Migliorare la mira.
 - Sparare solo quando il colpo ha una buona probabilità di andare a segno.
 - Ridurre l’uso superfluo delle munizioni.
- **Esaурimento dei colpi:** Un ulteriore penalità viene applicata quando l’agente esaurisce tutte le munizioni a sua disposizione.

Settings Agente

Estendendo la classe Agent si eredita il parametro *MaxStep* che permette di specificare il numero massimo di passi per episodio, ovvero gli Academy Steps per episodio:

- Se impostato a zero un episodio durerà fino a quando l’agente non raggiunge l’obiettivo o un altro criterio di terminazione definito manualmente dal progettista tramite *EndEpisode()*.
- Altrimenti quando l’agente raggiunge il numero massimo di passi specificato in *MaxStep*, Unity termina automaticamente l’episodio, anche se l’agente non ha ancora completato il task.

In questo caso Max Step viene impostato su un valore che permetta all’agente di esplorare in modo soddisfacente l’environment prima di concludere l’episodio.

4.3 Implementazione Metodologie di Addestramento

4.3.1 Curriculum Learning

Per proporre le diverse varianti di environment durante la fase di training, è integrata via script una versione del curriculum learning che aumenta progressivamente la complessità degli environments in ordine crescente come in figura 3.6. Il primo environment viene proposto per il primo 20% del training, il secondo fino al 35% del training, il terzo fino al 50% del training e il quarto(environment più complesso) per la restante percentuale rimanente. Il Curriculum Learning è stato adottato su ogni configurazione di addestramento sperimentata.

```
environment_parameters:  
  my_environment_parameter:  
    curriculum:  
      # Prima lezione (iniziale)  
      - name: lesson0  
        completion_criteria:  
          measure: progress  
          behavior: BehaviorTest  
          threshold: 0.20 # Cambia lezione al 20% del training  
          value: 0  
      # Seconda lezione  
      - name: lesson1  
        completion_criteria:  
          measure: progress  
          behavior: BehaviorTest  
          threshold: 0.35 # Cambia lezione al 35% del training  
          value: 1  
      # Terza lezione  
      - name: lesson2  
        completion_criteria:  
          measure: progress  
          behavior: BehaviorTest  
          threshold: 0.50 # Cambia lezione al 50% del training  
          value: 2  
      # Quarta lezione  
      - name: lesson3  
        completion_criteria:  
          measure: progress  
          behavior: BehaviorTest  
          threshold: 0.99  
          value: 3
```

FIGURA 4.10: Parametri curriculum learning

4.3.2 Randomizzazione degli Scenari

Per aumentare la robustezza(capacità di generalizzare) del modello addestrato, le posizioni e le rotazioni di spawn degli agenti o giocatori vengono generate casualmente. Nello scenario sono presenti muri, che servono a diversificare l'ambiente e fungono da ripari strategici. Tuttavia, questi muri possono entrare in conflitto con la fase di generazione degli agenti o giocatori all'inizio di ogni episodio, causando compenetrazione con i giocatori. Lo spawn consiste nella generazione casuale di una coordinata all'interno di un determinato range di coordinate.

Per evitare problemi di compenetrazione durante la scelta casuale dello spawn, è stato utilizzato il sistema di NavMesh fornito dalla libreria "*Navigation*" di Unity. Questa libreria consente di generare superfici correttamente distanziate dalle barriere dello scenario, permettendo di effettuare un campionamento casuale di una posizione(coordinata usata per lo spawn) sulla superficie generata, integrando una piccola porzione di logica aggiuntiva. Questo approccio ha eliminato il rischio che agenti o giocatori venissero generati all'interno dei muri o in posizioni non valide.

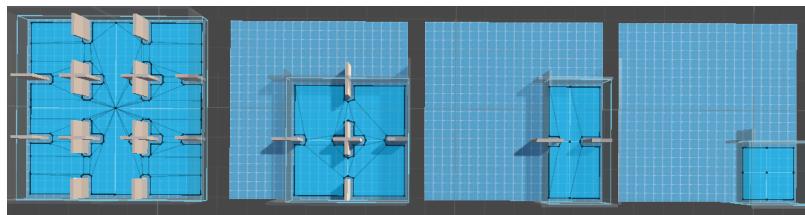


FIGURA 4.11: Superficie di spawn degli agenti rappresentata dall'area azzurra in ogni scenario del curriculum learning

4.3.3 Self-Play

In relazione alle considerazioni fatte sul self-play nella sottosezione 3.6.3, la sua configurazione consiste nel settare gli iperparametri che bilanciano le problematiche di diversità e stabilità.

```
self_play:
  save_steps: 60000
  team_change: 360000
  play_against_latest_model_ratio: 0.5
  window: 12
```

FIGURA 4.12: configurazione iperparametri self-play

Iperparametri per il self-play utilizzati:

- *save_steps*: Indica ogni quanti step salvare lo snapshot della policy corrente. Questo parametro permette di configurare la soluzione al problema legato alla "diversità" descritto nelle considerazioni precedenti.
- *team_change*: Indica dopo quanti step avviene il cambio di agente avversario. Questo parametro configura la soluzione al problema di "stabilità".
- *play_against_latest_model_ratio*: Configura la probabilità che l'agente giochi contro un avversario che adotti l'ultima versione della politica. Invece con probabilità $1 - play_{against latest model ratio}$ l'agente giocherà contro un'istantanea di se del passato.
- *window*: Rappresenta la dimensione di "finestra scorrevole" degli snapshot delle policy passate. Ad esempio una *window* con dimensione 5 salverà gli ultimi 5 snapshot acquisiti. Aumentare il valore di *window* significa che la pool di avversari conterrà una maggiore diversità di policy. Aumentare il parametro richiede più step di addestramento complessivi, ma potrebbe portare ad addestrare una policy più robusta con più capacità di generalizzazione.

Per addestrare un agente usando il modulo di addestramento Self-Play, si introduce per ogni environment di addestramento parallelo, la presenza di un ulteriore agente. In ogni environment i due agenti nel component Behaviour Parameters devono configurare un valore di *TeamId* univoco. In presenza di due agenti per scena, il primo agente viene assegnato al team 0 e il secondo al team 1.

La progettazione prevede che la terminazione dell'episodio dei due agenti sia sincronizzata. La terminazione dell'episodio dei due agenti appartenenti allo stesso environment viene delegata alla classe *EnvironmentController*, chiamando il metodo *EndEnvironmentEpisode*:

```
+reference
public void EndEnvironmentEpisode(int winningAgent) {
    Debug.Log(message: "EndEnvironmentEpisode");
    // premia l'agente vincitore e penalizza gli altri(perdenti)
    for(int i = 0; i < agents.Length; i++) {
        if(agents[i].gameObject.GetComponent<BehaviorParameters>().TeamId == winningAgent) {
            agents[i].AddReward(increment: 20F);
        } else {
            agents[i].AddReward(increment: -20F);
        }
    }

    // termina episodio per tutti gli agenti
    for(int i = 0; i < agents.Length; i++) {
        agents[i].EndEpisode();
    }
}
```

FIGURA 4.13: Metodo per terminare l'episodio per ogni agente nell'ambiente di addestramento

La terminazione dell'episodio per ogni agente avviene quando si verifica un evento di fine gioco.

4.3.4 Imitation Learning

Campionamento delle Dimostrazioni

Per campionare le dimostrazioni da fornire alle tecniche di imitation learning è stato utilizzato il componente offerto dal toolkit Demonstration Recorder. Il componente associato all'agente ha permesso di registrare le partite tra due giocatori umani (con i behaviour in modalità heuristic per rispondere ai comandi forniti dai controller comandati da giocatori umani). La directory del file contenente le dimostrazioni viene fornita nella fase di configurazione degli iperparametri delle tecniche di addestramento. Sono stati campionati 217 episodi.

GAIL

Per abilitare la rete Generative Adversarial nella configurazione di addestramento basata su imitazione, nel file di configurazione dell'addestramento viene aggiunta, nella sezione reward signals, una sottosezione dedicata agli iperparametri specifici per la rete addestrata (discriminatore).

```
gail:
    gamma: 0.99
    strength: 0.3
    learning_rate: 0.0003
    use_actions: true
    demo_path: Assets/Demos/Demos1(player1)/recs.demo
```

FIGURA 4.14: Iperparametri del modulo GAIL

Gli iperparametri utilizzati sono i seguenti:

- *gamma*: Fattore di sconto per le ricompense future.
- *strength*: Fattore che regola l'intensità della ricompensa intrinseca. È stato configurato in modo tale che le ricompense intrinseche utilizzate per addestrare l'agente a imitare le dimostrazioni, non prevalessero sulle ricompense estrinseche (che definiscono gli obiettivi del gioco).
- *learning_rate*: Tasso di apprendimento per aggiornare il discriminatore.

- *user_actions*: Permette al discriminatore di considerare non solo le osservazioni, ma anche le azioni. Questo ha consentito all'agente di apprendere ad imitare anche le azioni(oltre che le osservazioni) registrate nelle dimostrazioni.
- *demo_path*: Campo obbligatorio che specifica la directory del file contenente le dimostrazioni registrate.

Iperparametri Behaviour Cloning

Per abilitare il modulo di BC nella configurazione dell'addestramento, viene aggiunta una sezione dedicata agli iperparametri di BC. Il modulo non è basato su ricompense.

```
behavioral_cloning:
  demo_path: Assets/Demos/Demos1(player1)/recs.demo
  strength: 0.025
  samples_per_update: 0
```

FIGURA 4.15: Iperparametri del modulo BC

I cui iperparametri:

- *strength*: Corrisponde a quanto fortemente viene permesso a BC di influenzare la policy dell'agente in fase di addestramento
- *demo_path*: Campo obbligatorio che specifica la directory del file contenente le dimostrazioni registrate.
- *samples_per_update*: Numero massimo di campioni da utilizzare durante ogni aggiornamento di imitazione.

4.4 Setting di Addestramento

Per ciascuna delle tre configurazioni di addestramento scelte per la valutazione (5.3), è stato creato un file di configurazione. Ogni configurazione salvata in un file con estensione .yaml contiene una definizione specifica degli iperparametri.

Per avviare gli addestramenti, è stato utilizzato il seguente comando:

```
PS C:\Users\roman> magents-learn --torch-device cuda config/RL+Curiosity+GAIL.yaml
--run-id="RL+Curiosity+GAIL" --force
```

FIGURA 4.16: Comando per avviare l'addestramento

I parametri del comando sono i seguenti:

- *torch-device cuda*: Imposta la GPU come dispositivo di calcolo per l’addestramento.
- *file.yaml*: Specifica il file contenente la configurazione degli iperparametri per lo specifico addestramento.
- *id=""*: Definisce un identificativo per l’istanza di addestramento.

Capitolo 5

Sperimentazione

5.1 Introduzione

La fase di sperimentazione si è basata sull'esecuzione di tre diverse configurazioni di addestramento, come descritto nella sezione 5.3. I risultati di questi addestramenti sono stati analizzati nella sezione Analisi dei Risultati degli Esperimenti (5.4). Inoltre, nella sezione dedicata ai Test Utente (5.5), un gruppo di partecipanti ha valutato i tre modelli addestrati attraverso sessioni di gioco 3.8.

5.2 Metriche di Valutazione

Prima di illustrare i risultati, vengono descritte le metriche utilizzate per tracciare i progressi durante gli addestramenti. Queste metriche si sono dimostrate fondamentali per monitorare e interpretare al meglio i risultati ottenuti, consentendo di identificare eventuali problematiche, eseguire il debug e approfondire l'effetto di diverse configurazioni degli iperparametri.

Le metriche introdotte sono rese disponibili tramite la TensorBoard di ML-Agents.

5.2.1 Episode Length

La metrica Environment/Episode Length rappresenta la durata media degli episodi durante l'addestramento dell'agente. Misura il numero di passi temporali medi che intercorrono dall'inizio di un episodio fino alla sua terminazione, che può avvenire per il raggiungimento dell'obiettivo, per un fallimento o per il superamento di un timeout.

Questa metrica è utile per valutare la capacità dell'agente di completare gli episodi in tempi brevi. Inoltre, può fornire indicazioni su potenziali situazioni di stagnazione, in cui l'agente rimane bloccato senza riuscire a completare efficacemente gli obiettivi richiesti.

Interpretazione

Di seguito, i valori assunti da questa metrica permettono di osservare:

- Valori alti: Possono suggerire che l'agente fatica a completare il compito richiesto, o che l'ambiente è troppo complesso per il livello corrente di apprendimento. Tuttavia, in alcuni contesti, valori alti possono anche indicare un comportamento esplorativo, utile in fasi iniziali dell'addestramento.
- Valori troppo alti: Quando i valori sono troppo alti e convergono a *max_step* (massimo numero di passi per episodio), può voler dire che l'agente tende a non riuscire a completare i task, stagnando in comportamenti subottimali.
- Valori bassi: Indicano che gli episodi terminano rapidamente. Questo può essere un segnale positivo se l'agente sta completando i compiti in modo efficiente. Tuttavia, valori eccessivamente bassi potrebbero indicare che l'agente termina gli episodi prematuramente a causa di errori frequenti o strategie subottimali. Ad esempio, se viene associata una penalità troppo forte sulla durata totale dell'episodio, l'agente potrebbe apprendere una strategia che lo porta a far terminare intenzionalmente l'episodio per fallire il compito, pur di subire una penalità minore.
- Oscillazioni dei valori: Possono essere comuni durante fasi di apprendimento dinamico. Oscillazioni frequenti o eccessive, però, potrebbero indicare instabilità nell'apprendimento o difficoltà nell'adattarsi a nuovi scenari.

Comportamenti Tipici in Fase di Addestramento

Durante l'addestramento, la Environment/Episode Length può seguire comportamenti caratteristici che variano in base al task di addestramento. Se l'obiettivo del task si basa sulla risoluzione di un compito in modo efficiente e ottimale in un ambiente statico, idealmente la Environment/Episode Length dovrebbe diminuire fino a stabilizzarsi su un valore costante. Tuttavia, in presenza di tecniche come il Curriculum Learning o il Self-Play, dove la complessità degli scenari evolve progressivamente, è normale osservare fasi alternate di crescita e decrescita fino al raggiungimento di una stabilità finale.

In particolare, nel caso del Self-Play, dove l'addestramento si basa sullo sfidare un avversario che utilizza istanze di politiche precedenti dell'agente stesso,

l’Episode Length tende ad aumentare. Questo accade perché entrambi gli agenti cercano di massimizzare le proprie probabilità di vittoria, evitando la sconfitta, il che porta a una durata media degli episodi più lunga. Una durata crescente degli episodi può quindi indicare che l’agente sta progressivamente apprendendo strategie più sofisticate ed efficaci contro il proprio avversario (una copia di sé stesso). Inoltre anche GAIL introduce un fattore di survivor bias, allungando la durata media degli episodi 3.6.3.

5.2.2 Cumulative Reward

La Cumulative Reward rappresenta la ricompensa cumulativa (estrinseca) media ricevuta dall’ambiente per episodio. Generalmente, questa è considerata un’ottima metrica quando l’agente opera in un ambiente stazionario. L’Extrinsic Reward generalmente aumenta man mano che l’agente impara ad eseguire sempre meglio il task. Tuttavia, come spiegato nella sottosezione 3.6.3, nei contesti degli Adversarial Games, la ricompensa cumulativa media non può essere considerata una metrica assolutamente significativa per la valutazione. Questo accade perché, con il progressivo miglioramento delle abilità dell’avversario, la ricompensa cumulativa dell’agente media tende a diminuire, dato che la competizione rende sempre più difficile per l’agente compiere azioni vantaggiose senza incontrare resistenza.

5.2.3 Curiosity Inverse Loss

La Curiosity Inverse Loss è l’indicatore chiave legato al Intrinsic Curiosity Module (ICM), il modulo integrato nelle sperimentazioni; come già approfondito nella Sottosezione 3.6.1 esso è utilizzato per incentivare l’esplorazione da parte dell’agente nell’ambiente. Il Modello Inverso cerca di prevedere quale azione l’agente ha compiuto per passare dallo stato A allo stato B. La Curiosity Inverse rappresenta l’errore di questa predizione. La Curiosity Inverse Loss è stata usata solo negli addestramenti che hanno integrato il Curiosity Module.

Interpretazione

- Valori alti di Inverse Loss indicano che il modello ha difficoltà a predire le azioni dell’agente, suggerendo che l’ambiente potrebbe essere altamente imprevedibile o che le osservazioni sono ancora troppo complesse da interpretare rispetto all’esperienza dell’agente.

- Un valore basso di Inverse Loss significa che il modello inverso riesce a comprendere bene la relazione tra stati e azioni, implicando che l'ambiente è prevedibile per l'agente e che quindi questo riesca a reagire rispetto alle sue esperienze.

Comportamenti Tipici in Fase di Addestramento

La curiosità di un agente aumenta quando l'agente non riesce a prevedere completamente gli effetti delle sue azioni nell'ambiente, quindi adotta un approccio più esplorativo, raccogliendo altre informazioni dall'ambiente. Idealmente, quando la Inverse Loss raggiunge un valore basso e stabile, indica che il modello inverso ha imparato efficacemente la relazione tra stati e azioni.

5.2.4 GAIL Reward

La GAIL Reward rappresenta la ricompensa cumulativa media che misura quanto l'agente è in grado di imitare le dimostrazioni fornite durante l'addestramento per imitazione.

Interpretazione

- Un aumento dei valori della GAIL Reward indica che l'agente sta migliorando nell'ingannare il discriminatore, imitando sempre più fedelmente le dimostrazioni fornite.
- La stabilizzazione o la diminuzione dei valori suggerisce che l'agente non sta più progredendo nel migliorare la propria capacità di imitare e ingannare il discriminatore.

Una configurazione adeguata dell'iperparametro *strength* dovrebbe favorire un aumento della GAIL Reward. Eventuali irregolarità nella crescita possono verificarsi se l'agente viene esposto a scenari più complessi (ad esempio, tramite il curriculum learning), ma si prevede che i valori tornino a salire man mano che l'agente impara ad ingannare il discriminatore.

5.2.5 Self-Play/ELO

Introducendo il Self-Play come tecnica di addestramento in presenza di più agenti, la ricompensa cumulativa, che generalmente viene utilizzata come metrica di valutazione principale per tracciare i progressi dell'apprendimento, potrebbe non essere più così rappresentativa. Questo perché la ricompensa cumulativa

non dipende unicamente dalle azioni dell’agente nell’ambiente, ma anche dall’abilità dell’avversario. Di conseguenza, la ricompensa ottenuta dall’agente può variare notevolmente in base al livello di abilità dell’avversario. Per affrontare il caso di apprendimento Self-Play, ML-Agents fornisce un sistema di valutazione ELO¹, che consente di calcolare il livello di abilità relativo tra due giocatori in un gioco a somma zero. I giochi a somma zero sono quelli in cui il guadagno di un agente corrisponde esattamente alla perdita dell’altro. Lo scenario di gioco progettato non può essere considerato un vero e proprio gioco a somma zero, ma si avvicina molto a questa definizione, poiché:

- Danneggiare un avversario equivale a guadagnare un punto per l’agente, facendo contemporaneamente perdere un punto all’avversario.
- Il punteggio guadagnato dall’agente vincente corrisponde esattamente al punteggio perso, in negativo, dall’agente perdente.

Tuttavia, l’agente riceve una penalità se il colpo sparato non va a segno, questa penalità non si traduce in un guadagno per l’avversario. Per questo motivo, il gioco non può essere classificato come un gioco a somma zero nel senso stretto del termine. Con il progredire degli episodi, comunque, l’agente impara a ridurre al minimo i colpi mancati(i colpi mancati equivalgono a una penalità), quindi lo scenario di gioco dovrebbe comunque convergere verso un gioco a somma zero. Date le premesse, la ELO non è stata considerata una metrica assolutamente rappresentativa per il monitoraggio delle performance, pur rimanendo un buon punto di riferimento per valutare il comportamento dell’agente.

Comportamenti Tipici in Fase di Addestramento

I comportamenti caratteristici del punteggio ELO devono evidenziare una progressione nel tempo, mostrando come l’agente addestrato riesca ad adattarsi progressivamente alle diverse istanze della propria policy all’interno dell’ambiente.

1. Fase iniziale: I punteggi ELO potrebbero tendere a fluttuare in modo significativo, poiché gli agenti inizialmente esplorano l’ambiente e sperimentando strategie diverse, risultando una misura poco informativa sulle capacità dell’agente.
2. Fase intermedia: L’ELO di un agente vincente dovrebbe diminuire la decrescita o crescere progressivamente, segnalando un miglioramento delle

¹Dettagli sul sistema di valutazione Elo: [urlhttps://en.wikipedia.org/wiki/Elo_rating_system](https://en.wikipedia.org/wiki/Elo_rating_system)

sue capacità strategiche rispetto all'avversario. Fluttuazioni in questa fase possono essere normali, soprattutto se in presenza di aggiornamenti frequenti della politica avversaria o di un cambio di complessità dello scenario dovuto al Curriculum Learning.

3. Fase avanzata: I punteggi ELO tendono a stabilizzarsi quando gli agenti hanno raggiunto un equilibrio competitivo. Uno stallo nell'ELO può indicare che entrambi gli agenti stanno raggiungendo una politica ottimale.

Nel caso della sperimentazione, nonostante le decrescite iniziali dovute alla natura del gioco non propriamente a somma zero, una stabilizzazione del valore ELO indicherà che l'agente sta imparando a utilizzare in modo efficiente le munizioni a sua disposizione. Dopo tale stabilizzazione, continuando con gli step di addestramento, ci si aspetta che questa misura possa persino aumentare.

5.3 Configurazioni Addestramenti

Per la sperimentazione, sono state definite tre configurazioni di addestramento distinte. Ogni configurazione prevede l'abilitazione di vari moduli. Ogni configurazione è stata progettata con l'obiettivo di evidenziare le differenze imitative e non imitative tra i modelli addestrati. Lo scopo finale è dimostrare la capacità degli algoritmi di training imitativi integrati con RL di far percepire ai giocatori che gli avversari mostrino comportamenti simili a quelli umani.

Le tre configurazioni definite per le sperimentazioni sono:

- **RL + Curiosity:** L'agente viene addestrato utilizzando solo il modello RL base integrato con il modulo Curiosity, per valutare la dinamicità di gioco e la capacità dell'agente di competere contro un avversario in un gioco simmetrico(Self-Play). In questa configurazione, l'agente non è stato addestrato per imitare dimostrazioni, ma esclusivamente per completare il task, ovvero sconfiggere l'avversario(se stesso).
- **RL + Curiosity + GAIL:** Oltre ad addestrare l'agente a completare il task, a questa configurazione viene aggiunto il modulo GAIL per permettere all'agente di apprendere comportamenti basati sulle dimostrazioni.
- **RL + Curiosity + GAIL + Behaviour Cloning:** In questa configurazione viene introdotto anche il Behaviour Cloning, per tentare di

marcare ulteriormente l'imitazione dei comportamenti delle dimostrazioni fornite.

5.4 Analisi dei Risultati degli Esperimenti

In generale, tutti e tre gli addestramenti hanno prodotto modelli efficaci, ovvero agenti addestrati in contesti multi-agente capaci di competere l'uno contro l'altro in modo mediamente performante. Inoltre, ciascun modello evidenzia chiaramente tre approcci distinti per competere e affrontare l'avversario.

Modelli Addestrati tramite Imitation Learning

I modelli addestrati tramite imitation learning hanno dimostrato di far emergere comportamenti simili a quelli osservati nelle dimostrazioni di partite umane. Dai test utente e dall'esecuzione delle inferenze sono emerse strategie quali:

- Attacchi indiretti per sfuggire ai colpi diretti dei nemici
- Esplorazione dell'ambiente più rapida rispetto ai modelli non imitativi
- Evasione da situazioni di attacco diretto

Inoltre, i modelli basati su tecniche imitative hanno mostrato una maggiore capacità di evitare lo spreco di colpi quando non sono avvistati nemici, un miglioramento rispetto ai modelli addestrati tramite il solo reinforcement learning. Tuttavia, il modello imitativo basato su Behaviour Cloning ha evidenziato prestazioni inferiori sia in termini di sfida che di realismo percepito. Questo può essere attribuito alla scarsa capacità di generalizzazione tipica di questa tecnica. In un ambiente altamente non stazionario, come quello sperimentato, tale incapacità di generalizzazione ha penalizzato significativamente i risultati ottenuti.

Modello Addestrato tramite Reinforcement Learning

Il modello addestrato tramite reinforcement learning ha adottato strategie molto più aggressive e dirette, mirate principalmente al compimento efficace del task. Ad esempio, un utilizzo più frequente delle munizioni ha aumentato le probabilità di successo negli scontri diretti. Tuttavia, il modello non ha imparato a razionare le munizioni durante l'intera partita, il che lo conduce a esaurire i colpi, determinandone la sconfitta.

5.4.1 Episode Length

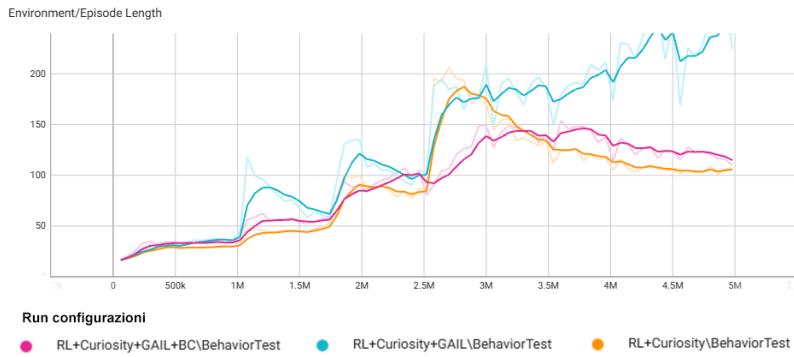


FIGURA 5.1: Durata media degli episodi

Analisi Andamento Risultati

Le tre configurazioni di addestramento hanno mostrato risultati simili in alcune fasi, mentre in altre hanno evidenziato differenze significative. Di seguito si analizzano le fasi della metrica:

- Fase iniziale: L’andamento nella fase iniziale è pressoché simile tra le diverse configurazioni. Gli episodi tendono a durare poco, un risultato previsto poiché il curriculum learning nelle fasi iniziali propone spazi ristretti. In tali ambienti, l’agente tende a terminare rapidamente gli episodi esaurendo le munizioni o colpendo facilmente l’avversario. Nel primo milione di step, si osserva un leggero aumento della durata media degli episodi, attribuibile al fatto che l’agente inizia ad imparare ad evitare le collisioni con i muri (che terminerebbero l’episodio), a sprecare meno colpi e a migliorare la precisione nel colpire l’avversario.
- Fase intermedia: Durante le fasi intermedie, l’aumento della complessità degli scenari proposti dal curriculum learning (ad esempio, dopo il primo milione di step) porta l’agente di tutte e tre le configurazioni a trovarsi in ambienti non familiari. In queste fasi, gli episodi tendono a durare più a lungo poiché l’agente deve esplorare e imparare i nuovi stati introdotti dal curriculum learning. Tuttavia, man mano che l’agente acquisisce familiarità con i nuovi scenari, la durata degli episodi diminuisce gradualmente. Questo comportamento di crescita e successiva stabilizzazione è periodico e coincide con i cambiamenti di scenario sempre introdotti dal curriculum learning.

- Fase conclusiva: Nelle fasi conclusive, si osservano convergenze differenti tra i modelli, evidenziando che ciascuna configurazione ha appreso policy differenti, con conseguenti stili di gioco differenti.

Analisi Differenze dei Risultati

Di seguito vengono analizzate le ragioni di queste discrepanze tra i tre addestramenti:

- RL+Curiosity: Questo addestramento dimostra di ridurre la durata degli episodi molto più rapidamente rispetto alle altre. Ciò avviene perché l'unico obiettivo dei modelli addestrati tramite reinforcement learning è massimizzare la ricompensa estrinseca, equivalente a sconfiggere l'avversario con il minor numero di colpi e nel minor tempo possibile. Anche nella fase di convergenza finale, si osserva come questa configurazione di addestramento diventi altamente efficiente, concludendo gli episodi molto rapidamente, suggerendo l'adozione di strategie nette e ben definite.
- RL+Curiosity+GAIL: Nonostante la presenza del modulo GAIL, l'agente dimostra di essere rapido nell'adattarsi ai cambiamenti dell'environment durante l'addestramento (curriculum learning). Si osservano massimi locali più elevati, attribuibili al fatto che GAIL introduce una ricompensa intrinseca basata sull'ingannare il discriminatore (Survivor Bias, descritto in 3.6.3), incentivando l'agente a rimanere in vita più a lungo. Nelle fasi finali, gli episodi tendono a durare mediamente più a lungo. Questo comportamento può essere ricondotto al survivor bias e alla capacità degli agenti addestrati con GAIL, di cogliere dalle partite registrate l'intenzione dei giocatori umani di prolungare la propria sopravvivenza. Gli agenti, infatti, non affrontano sempre lo scontro in modo diretto, ma in alcune situazioni preferiscono evitare il conflitto per poi ritentare successivamente. Di conseguenza, si registra un aumento medio della durata degli episodi, maggiore rispetto a tutte le altre tecniche. La durata media maggiore delle partite in questo caso non è un fatto negativo dal momento che la durata di sopravvivenza da parte degli agenti non deriva dall'incapacità di portare a termine il task, ma dal tentativo di tentare strategie di attacco meno dirette.
- RL+Curiosity+GAIL+BC: L'aggiunta del Behaviour Cloning mostra come la durata media degli episodi è più bassa rispetto agli altri. Ciò è dovuto a una maggiore tendenza della tecnica a replicare anche gli errori

presenti nelle dimostrazioni, come collisioni contro i muri, che portano al fallimento e alla conclusione anticipata degli episodi. Usando BC l'agente impara ad imitare esattamente le dimostrazioni con nessuna capacità di generalizzare, portando ad imparare anche gli errori.

5.4.2 Cumulative Reward

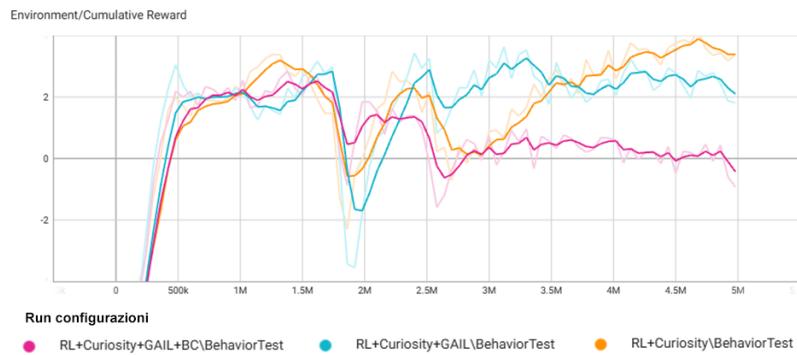


FIGURA 5.2: Ricompensa cumulativa media per episodio

Analisi Andamento Risultati

Anche in questo caso, le tre configurazioni di addestramento hanno mostrato andamenti simili in alcune fasi, mentre in altre si sono evidenziate differenze significative. Come osservato per la metrica Episode Length, è possibile identificare degli avvallamenti nelle ricompense medie (in questo caso, minimi locali), attribuibili alla temporanea difficoltà del modello nello sfruttare le conoscenze acquisite nel nuovo scenario proposto dal curriculum learning. Tuttavia, dopo una fase di esplorazione e accumulo di esperienza nell'ambiente, le ricompense cumulative tornano a crescere mediamente per tutte le configurazioni.

Analisi Differenze dei Risultati

- **RL+Curiosity:** L'addestramento si distingue per la capacità di totalizzare le ricompense cumulative medie più elevate rispetto agli altri. Questo risultato è attribuibile alla sua efficienza nel completare i match in tempi molto più brevi, favorendo scontri diretti. Partite di breve durata permettono di ridurre l'accumulo di penalità, dato che match più lunghi comportano penalità maggiori.
- **RL+Curiosity+GAIL:** Questa configurazione presenta prestazioni simili a RL+Curiosity. Questa configurazione rispetto alla configurazione

RL+Curiosity dimostra un'efficacia più bassa nello sconfiggere gli avversari. Il modello RL+Curiosity+GAIL però, si distingue per la capacità di risparmiare colpi. Sebbene i match risultino mediamente più lunghi, con un conseguente aumento delle penalità legate alla durata dell'episodio, l'accuratezza nello sparo permette di evitare penalità associate a colpi errati, contribuendo a mantenere alta la ricompensa cumulativa media.

- RL+Curiosity+GAIL+BC: L'aggiunta del Behaviour Cloning (BC) non migliora le ricompense cumulative medie. Al contrario, con l'avanzare degli step di addestramento, la ricompensa cumulativa media tende a diminuire, mostrando come l'agente non riesca ad imparare una policy ottimale integrando BC. BC a causa della mancanza di generalizzazione, per funzionare bene necessiterebbe di più dimostrazioni.

Curiosity Inverse Loss

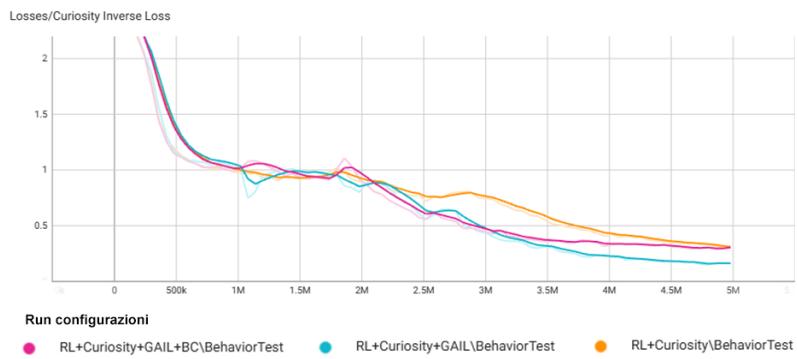


FIGURA 5.3: Loss modulo Curiosity

Analisi Andamento Risultati

La Inverse Loss mostra un andamento generale decrescente per tutti gli addestramenti, con fluttuazioni caratterizzate da piccoli incrementi e successivi decrementi circoscritti a specifici intervalli, che generano massimi locali nei punti in cui il Curriculum Learning introduce nuovi scenari per gli agenti. Questi incrementi, tuttavia, sono temporanei, poiché man mano che l'agente si adatta al nuovo scenario, la loss tende nuovamente a decrescere. Questo comportamento evidenzia come i cambiamenti di scenario introdotti dal Curriculum Learning aumentino la difficoltà per l'agente nel prevedere gli stati successivi basandosi sull'esperienza precedente (incrementando l'errore nella previsione delle azioni). Questo porta l'agente a esplorare maggiormente il nuovo ambiente, stimolando la curiosity e aiutando l'agente ad adattarsi efficacemente.

GAIL Reward

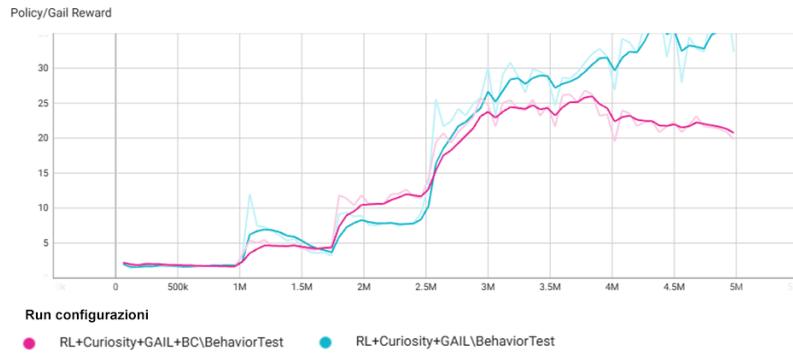


FIGURA 5.4: Ricompensa cumulativa intrinseca GAIL

Analisi Andamento Risultati

La misura GAIL Reward è osservabile solo negli addestramenti che hanno abilitato il modulo GAIL. Poiché le dimostrazioni tra giocatori reali sono state registrate direttamente negli scenari più complessi del Curriculum Learning, nelle fasi iniziali dell’addestramento entrambe le configurazioni faticano a imparare ad imitare correttamente le dimostrazioni in ambienti differenti, ovvero quelli più semplici proposti dal Curriculum Learning. La GAIL Reward mostra una crescita più significativa solo quando vengono introdotti scenari di complessità maggiore, più simili a quelli utilizzati per campionare le dimostrazioni. Le crescite in entrambi i modelli evidenziano come entrambi abbiano imparato ad ingannare discretamente il discriminatore.

Analisi Differenze dei Risultati

- **RL+Curiosity+GAIL:** L’addestramento che utilizza esclusivamente il modulo GAIL mostra risultati migliori nell’imparare ad ingannare il discriminatore e, di conseguenza, nell’imitare le dimostrazioni fornite. L’andamento finale della metrica suggerisce che ulteriori step di addestramento potrebbero portare a margini di crescita, migliorando ulteriormente la capacità di imitare le dimostrazioni.
- **RL+Curiosity+GAIL+BC:** L’aggiunta del modulo Behaviour Cloning (BC) produce risultati abbastanza soddisfacenti, ma evidenzia una capacità inferiore dell’agente di rispondere dinamicamente alle situazioni di gioco rispetto alla configurazione RL+Curiosity+GAIL, che si dimostra molto più flessibile e credibile. Questo peggioramento potrebbe essere attribuito al fatto che BC richiede una quantità molto elevata di dimostrazioni per

funzionare efficacemente, necessaria per coprire il maggior numero possibile di combinazioni osservazione-azione. Tale limite è dovuto alla scarsa capacità di generalizzazione di BC, a differenza di GAIL, che è in grado di operare efficacemente anche con un set limitato di dimostrazioni.

Self-Play/ELO

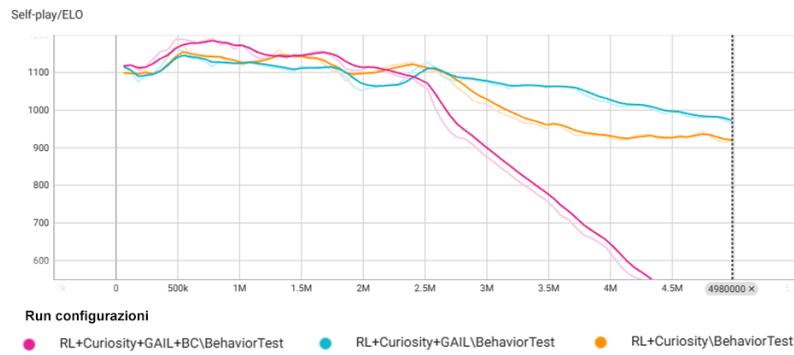


FIGURA 5.5: Punteggio ELO

Analisi Andamento Risultati

Come già accennato, lo scenario di gioco progettato non può essere considerato un vero e proprio gioco a somma zero, poiché le penalità per i colpi non andati a segno o per le collisioni contro i muri dello scenario non corrispondono a ricompense per l'agente avversario. Tuttavia, con il progredire degli addestramenti, man mano che gli agenti imparano ad usare i colpi in modo efficiente e a evitare le collisioni con i muri, il gioco tende ad assumere dinamiche simili a quelle di un gioco a somma zero.

Nonostante questa limitazione, la metrica del punteggio ELO si è rivelata un buon indicatore per valutare le performance degli agenti nelle diverse configurazioni di addestramento.

In un vero gioco a somma zero, ci si aspetterebbe che il punteggio ELO mostri una progressione più evidente. Ma in questo caso specifico, data la natura del gioco (non propriamente a somma zero), il comportamento del punteggio ELO non segue esattamente questo schema e mostra invece una tendenza generale alla decrescita, con pendenze variabili. Tuttavia, si sono osservate tendenze più stabili (ovvero decrescite meno marcate) nei casi in cui gli agenti hanno adottato comportamenti più competitivi nei confronti dell'avversario.

Analisi Differenze dei Risultati

Come si può notare, le configurazioni RL+Curiosity e RL+Curiosity+GAIL mostrano punteggi ELO simili, riflettendo un grado di competitività degli agenti simile.

Nel caso dell’addestramento con il modulo Behaviour Cloning (BC), si osserva invece che l’agente non riesce a diventare sufficientemente competitivo. Questo peggioramento delle prestazioni è attribuibile a una policy poco efficace contro gli avversari (sempre a causa di una bassa capacità di generalizzare).

5.5 Test Utente

L’ipotesi centrale della tesi è dimostrare la capacità degli agenti addestrati di competere in modo realistico e sfidante. Poiché le metriche tecniche di valutazione non sono adatte a misurare il grado di realismo percepito e il livello di sfida, è stato condotto un test utente su un campione di partecipanti.

Durante il test, gli utenti hanno affrontato partite sia contro le tre configurazioni di agenti addestrati sia contro un giocatore umano. Con ordine casuale e senza sapere se stessero affrontando un avversario umano o un agente addestrato, 5 utenti hanno giocato a 16 partite esprimendo per ognuna una valutazione sui seguenti aspetti:

- **Difficoltà del match:** La percezione della sfida offerta dall’avversario.
- **Realismo percepito:** Quanto l’utente ha ritenuto che l’avversario fosse un giocatore umano.
- **Grado di soddisfazione del match:** Quanto il match è stato appagante dal punto di vista competitivo.

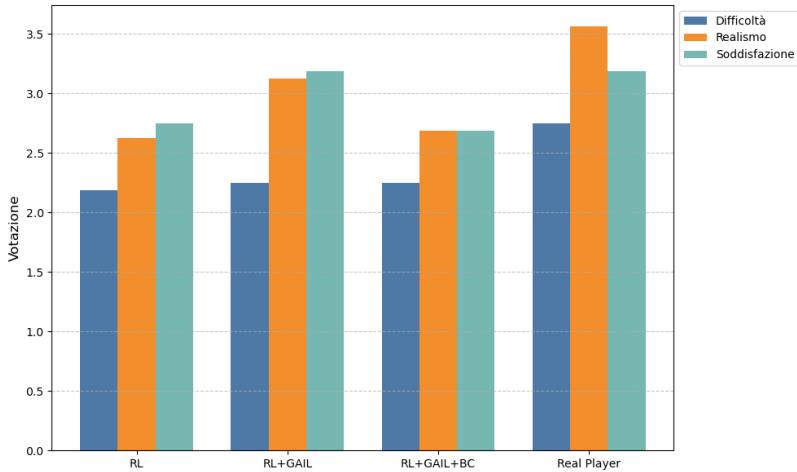


FIGURA 5.6: Risultati test utente

Complessivamente, le valutazioni sui tre modelli addestrati indicano che l’esperienza di gioco offerta tende ad avvicinarsi a quella percepita con un giocatore umano (Real Player, rappresentato nell’ultimo grafico a barre multiple sulla destra). Questo risultato dimostra che, in media, tutti i metodi sono stati considerati simili alle partite giocate contro un avversario reale. Tra i tre modelli, i risultati delle votazioni su GAIL hanno mostrato la maggiore somiglianza con il comportamento umano, risultando il più realistico e convincente.

5.6 Conclusioni

Nella configurazione RL+GAIL, è stato utilizzato un valore di intensità delle ricompense intrinseche GAIL relativamente basso. L’andamento della cumulative reward, stabile rispetto a una GAIL reward ancora in crescita, suggerisce che sia possibile sperimentare ulteriormente. In particolare, si potrebbe aumentare l’intensità delle ricompense intrinseche fornite da GAIL e incrementare il numero massimo di step per valutare fino a che punto il modello riesca a integrare efficacemente il comportamento imitativo, mantenendo al contempo un buon livello di competitività.

5.6.1 Self-Play

Nelle sperimentazioni, la tecnica del curriculum learning si è dimostrata fondamentale per guidare l’addestramento degli agenti verso politiche ottimali fin dalle prime fasi. Senza l’utilizzo di questa tecnica, gli addestramenti si sono rivelati molto più instabili, con una velocità di convergenza verso una policy ottimale significativamente più lunga.

Generalmente, gli algoritmi di RL assumono che l’ambiente sia stazionario, tuttavia, negli adversarial games questa assunzione diventa problematica poiché la presenza di agenti avversari riduce drasticamente la prevedibilità delle conseguenze delle azioni dell’agente addestrato sull’ambiente. In altre parole, la presenza di avversari aumenta la non stazionarietà dello scenario di training, complicando il processo di apprendimento.

Nel contesto degli adversarial games (Self-Play), il curriculum learning, basato sulla progressiva complessità degli scenari di addestramento, si è rivelato una tecnica particolarmente efficace per affrontare il problema della non stazionarietà. Partendo da scenari semplici, l’agente ha potuto apprendere interazioni di base che hanno poi costituito una solida base per affrontare scenari più complessi, imparando a generalizzare bene. Questo approccio si è dimostrato particolarmente utile in presenza di altri agenti nello scenario di training, dove altrimenti l’addestramento sarebbe risultato instabile o avrebbe richiesto tempi significativamente più lunghi. Complessivamente rispetto alle sperimentazioni, il curriculum learning ha contribuito a migliorare in modo significativo l’efficacia e l’efficienza del processo di addestramento.

5.6.2 Behaviour Cloning

La scarsa efficacia di BC è probabilmente attribuibile all’incapacità di generalizzare, aggravata dalla ridotta quantità di dimostrazioni fornite per l’addestramento. Questa limitazione impedisce agli agenti di sviluppare una policy ottimale per il task. In particolare, la configurazione BC si dimostra inefficace in ambienti altamente non stazionari, come nel caso del gioco simmetrico sperimentato, con conseguente abbassamento delle prestazioni complessive degli agenti.

5.6.3 Risultati Test Utente

I risultati dei test utente e i feedback raccolti dai partecipanti che hanno sperimentato le varianti degli agenti addestrati evidenziano che le tecniche di Reinforcement Learning e di Imitation Learning, in particolare GAIL, sono in grado di offrire un’esperienza di gioco realistica e dinamica. Questi risultati dimostrano come tali tecniche riescano a creare una percezione di presenza umana all’interno delle partite, mantenendo al contempo un livello di sfida soddisfacente per il giocatore. Tali risultati suggeriscono che questi algoritmi potrebbero essere impiegati per l’addestramento di NPC nei moderni videogiochi.

Capitolo 6

Conclusioni

6.1 Valutazioni Finali

Le sperimentazioni condotte in questa tesi hanno dimostrato che l'utilizzo combinato di tecniche di Reinforcement Learning, Imitation Learning e metodologie avanzate come il Self-Play può portare alla creazione di NPC capaci di offrire un'esperienza di gioco più realistica e dinamica. L'analisi delle metriche di valutazione, insieme ai risultati dei test utente, ha evidenziato che gli agenti addestrati con queste tecniche, in particolare con GAIL, possono non solo essere capaci di simulare comportamenti credibili e naturali, ma anche di competere efficacemente contro giocatori umani.

In particolare, il lavoro svolto ha fatto emergere i seguenti risultati:

- Il Curriculum Learning si è dimostrato un'ottima tecnica di addestramento per il reinforcement learning nel contesto degli adversarial games o in particolare negli ambienti non stazionari. Ha migliorato significativamente la stabilità dell'addestramento e accelerato la convergenza degli agenti verso politiche ottimali, anche in scenari complessi.
- Gli agenti addestrati esclusivamente con Reinforcement Learning hanno prodotto risultati soddisfacenti, mostrando comunque l'efficacia di questa tecnica anche senza integrazioni aggiuntive.
- L'integrazione di GAIL ha consentito di sviluppare modelli imitativi credibili, mantenendo le capacità di generalizzazione tipiche degli approcci di Reinforcement Learning.
- È stato messo in evidenza come le tecniche adottate possano contribuire a una percezione più umana degli NPC da parte degli utenti, contribuendo a rispondere ad esigenze di immersività e di sfida nelle esperienze di intrattenimento digitali.

- Tecniche come il Behaviour Cloning non si sono dimostrate particolarmente efficaci nel contesto studiato, nel contribuire alla creazione di modelli di agenti realistici. Questa limitazione è probabilmente dovuta alla scarsa capacità della tecnica di generalizzare e alla sua conseguente necessità di un numero significativamente maggiore di dimostrazioni per coprire una gamma più ampia di stati e azioni possibili. Questa vulnerabilità riduce la dinamicità dell'esperienza di gioco.

In conclusione, spero che le metodologie e i risultati esplorati in queste sperimentazioni possano dimostrare, anche al di fuori di questa tesi, le potenzialità dell'addestramento di NPC tramite le tecniche presentate. Questi approcci hanno dimostrato come possano contribuire a evidenziare nuove opportunità per migliorare la profondità delle interazioni con gli NPC, arricchendo i mondi di gioco nelle esperienze di intrattenimento digitale.

6.2 Sviluppi Futuri

Durante la realizzazione del progetto di tesi sono emersi diversi interrogativi e vari possibili sviluppi futuri:

- **Integrazione di strategie ibride:** Per combinare i punti di forza delle strategie deterministiche con i modelli appresi tramite RL, si potrebbe adottare un approccio ibrido all'interno di framework deterministiche, incorporando metodi di Reinforcement Learning. In questo contesto, i comportamenti appresi dai modelli RL potrebbero essere suddivisi e associati a specifici stati deterministiche. Un NPC potrebbe adottare un comportamento specifico, rappresentato da uno specifico modello addestrato, in base allo stato in cui si trova. Questo approccio mirerebbe a mitigare il problema legato alla difficoltà dei progettisti nel comprendere le situazioni di gioco, rendendo più prevedibili gli effetti degli agenti nell'ambiente.
- **Bilanciamento della difficoltà:** Un limite dei modelli di machine learning è l'impossibilità di bilanciare direttamente il livello di difficoltà del gioco. Tuttavia, un modo per adottare livelli di difficoltà minore potrebbe essere quello di utilizzare modelli addestrati in modo meno accurato. Questi modelli, meno sofisticati, potrebbero commettere errori con maggiore frequenza, emulando comportamenti simili a quelli di un giocatore umano inesperto. Questo approccio consentirebbe di modulare la difficoltà del gioco in base al livello dei giocatori.

- **Studio più approfondito sui giochi moderni:** Un interessante lavoro di ricerca potrebbe riguardare l'applicazione delle tecniche di Imitation Learning e di Reinforcement Learning ai moderni giochi multiplayer. Utilizzando dimostrazioni di partite reali per l'addestramento, questi modelli potrebbero essere addestrati per esplorare il potenziale di tali tecniche in giochi più complessi, caratterizzati da sequenze di azioni e strategie articolate. Un aspetto rilevante di una ricerca come questa sarebbe valutare come queste strategie vengano percepite da giocatori esperti del gioco in questione.

Bibliografia

- [1] Eloi Alonso et al. «Deep Reinforcement Learning for Navigation in AAA Video Games». In: *CoRR* abs/2011.04764 (2020). arXiv: 2011 . 04764. URL: <https://arxiv.org/abs/2011.04764>.
- [2] Kavosh Asadi et al. *Deep Radial-Basis Value Functions for Continuous Control*. 2021. arXiv: 2002.01883 [cs.LG]. URL: <https://arxiv.org/abs/2002.01883>.
- [3] Bowen Baker et al. «Emergent Tool Use From Multi-Agent Autocurricula». In: *arXiv preprint arXiv:1909.07528* (2019).
- [4] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: <https://arxiv.org/abs/1406.2661>.
- [5] Peter Henderson et al. «Deep Reinforcement Learning That Matters». In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (2018). DOI: 10.1609/aaai.v32i1.11694. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/11694>.
- [6] Jonathan Ho e Stefano Ermon. «Generative Adversarial Imitation Learning». In: *CoRR* abs/1606.03476 (2016). arXiv: 1606 . 03476. URL: <http://arxiv.org/abs/1606.03476>.
- [7] Arthur Juliani et al. «Unity: A General Platform for Intelligent Agents». In: *arXiv preprint arXiv:1809.02627* (2020). URL: <https://arxiv.org/pdf/1809.02627.pdf>.
- [8] Ilya Kostrikov et al. *Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning*. 2018. arXiv: 1809.02925 [cs.LG]. URL: <https://arxiv.org/abs/1809.02925>.
- [9] Jeff Orkin. «Three States and a Plan: The A.I. of F.E.A.R.» In: *Proceedings of the Game Developers Conference*. International Game Developers Association, 2006.
- [10] Deepak Pathak et al. «Curiosity-driven Exploration by Self-supervised Prediction». In: *CoRR* abs/1705.05363 (2017). arXiv: 1705 . 05363. URL: <http://arxiv.org/abs/1705.05363>.

- [11] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. USA: John Wiley & Sons, Inc., 1994.
ISBN: 0471619779.
- [12] Mooweon Rhee e Tohyun Kim. «Exploration and Exploitation». In: *The Palgrave Encyclopedia of Strategic Management*. A cura di Mie Augier e David J. Teece. London: Palgrave Macmillan UK, 2018, pp. 543–546.
ISBN: 978-1-137-00772-8.
DOI: 10.1057/978-1-137-00772-8_388. URL: https://doi.org/10.1057/978-1-137-00772-8_388.
- [13] John Schulman et al. «Proximal Policy Optimization Algorithms». In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: <http://arxiv.org/abs/1707.06347>.
- [14] Alessandro Sestini, Alexander Kuhnle e Andrew D. Bagdanov. «Deep Policy Networks for NPC Behaviors that Adapt to Changing Design Parameters in Roguelike Games». In: *CoRR* abs/2012.03532 (2020). arXiv: 2012.03532. URL: <https://arxiv.org/abs/2012.03532>.
- [15] David Silver et al. «Deterministic Policy Gradient Algorithms». In: *Proceedings of the 31st International Conference on Machine Learning*. A cura di Eric P. Xing e Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, 2014, pp. 387–395. URL: <https://proceedings.mlr.press/v32/silver14.html>.
- [16] David Silver et al. «Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm». In: *arXiv preprint arXiv:1712.01815* (2017).
- [17] Ronald J. Williams. «Simple statistical gradient-following algorithms for connectionist reinforcement learning». en. In: *Machine Learning* 8.3 (mag. 1992), pp. 229–256.
ISSN: 1573-0565.
DOI: 10.1007/BF00992696. URL: <https://doi.org/10.1007/BF00992696> (visitato il 04/12/2024).
- [18] Dinmukhammed Zhasulanov et al. «Enhancing Gameplay Experience Through Reinforcement Learning in Games». In: *2024 IEEE 4th International Conference on Smart Information Systems and Technologies (SIST)*. 2024, pp. 175–180.
DOI: 10.1109/SIST61555.2024.10629511.

Ringraziamenti

Desidero dedicare qualche riga ai ringraziamenti che, più che una semplice formalità retorica, rappresentano per me un sincero e profondo gesto di gratitudine.

Innanzitutto, vorrei ringraziare i professori che mi hanno trasmesso il valore della professionalità, intesa come una responsabilità incondizionata nei confronti della scienza e del ruolo professionale che ricopriamo. Un pensiero particolare va ai professori che hanno riaccesso le mie passioni, essenziali per il mio corretto funzionamento. Grazie all'esperienza universitaria, che mi ha offerto l'opportunità di migliorarmi più di quanto avrei mai immaginato, permettendomi di imparare a fidarmi delle mie potenzialità.

Un ringraziamento al mio relatore per avermi coinvolto in attività extrauniversitarie e per la fiducia dimostrata nel mio lavoro di tesi.

Infine, rivolgo un ringraziamento particolare alla mia famiglia. Specialmente, ai miei genitori, che con pazienza e sostegno mi hanno aiutato a raggiungere questo traguardo.