Università degli Studi di Salerno

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

Migliorare l'interazione ed il reasoning di un NPC in un gioco a turni tramite Large Language Models e Reinforcement Learning

Relatori

Prof. Vincenzo Deufemia

Dott. Gaetano Cimino

Candidato

Andrea Selice

Matricola: 0522501424

Anno Accademico 2023/2024

Indice

Elenco delle figure

Elenco delle tabelle

Abstract

I Non-Player Character (NPC), o personaggi non giocanti, sono entità presenti nei videogiochi, gestite dal sistema, che contribuiscono a rendere il mondo di gioco più immersivo e realistico, arricchendo l'esperienza complessiva del giocatore. Ad esempio, in un videogioco che presenta combattimenti a turni, la presenza di un NPC con il quale conversare per ottenere suggerimenti è un qualcosa che potrebbe rendere maggiormente interattivo lo scontro. Gli NPC sono progettati per interagire con i giocatori in molteplici modi, ricoprendo ruoli diversi. Tuttavia, sebbene siano una componente essenziale nei videogiochi, la loro implementazione può spesso risultare limitata, con comportamenti statici e dialoghi poco realistici. Questo accade perché si basano su set predefiniti di risposte che, durante lunghe sessioni di gioco, possono diventare ripetitivi e frustranti per il giocatore. Questa tesi affrontata la possibilità di usare i Large Language Models (LLM) ed il Reinforcement Learning (RL) per creare un NPC in grado di assistere un giocatore, rappresentato da un agente, suggerendo la prossima azione da intraprendere. Uno dei principali limiti degli LLM è la tendenza a generare allucinazioni, ovvero risposte sintatticamente corrette ma disconnesse dalla realtà. Per alleviare il problema delle allucinazioni, è stato chiesto al NPC di fornire il suo reasoning e, grazie ad un ulteriore agente di LLM addestrato tramite *Proximal Policy Optimization* (PPO), sono stati forniti istruzioni per effettuare il rephrasing della risposta iniziale, con l'obiettivo di rendere i consigli più efficaci possibili. Per testare l'efficacia dell'approccio proposto sono stati condotti esperimenti su un numero variabile di episodi. Inoltre, è stato condotto un test con una configurazione in due fasi in cui, nella prima, l'agente opera in autonomia per un determinato numero di episodi, mentre nella seconda è supportato dalla metodologia proposta, al fine di analizzare l'impatto dei consigli sulla strategia finale. I risultati dimostrano un significativo incremento nella qualità delle azioni proposte, evidenziando il potenziale dell'integrazione tra LLM e RL per migliorare l'interazione con un NPC nei videogiochi.

Capitolo 1

Introduzione

Per Non-Player Character (NPC) si intende ogni personaggio all'interno di un videogioco che non viene controllato dal giocatore [1], come ad esempio il Mercante di Resident Evil 4 [2]. Molto spesso gli NPC sono per lo più statici, hanno un set stabilito di risposte e spesso rimangono impassibili dinnanzi alle azioni del giocatore [3], come accade, ad esempio, con i cittadini del gioco Skyrim [4], che continuano le loro routine quotidiane indipendentemente dal caos circostante.

Negli ultimi anni, i Large Language Models (LLM) hanno aperto nuove possibilità per rendere le interazioni con gli NPC più naturali e coinvolgenti [5]. Grazie alla loro capacità di generare risposte coerenti in base al contesto fornito, gli LLM possono trasformare gli NPC rendendoli più dinamici, in grado di rispondere in modo personalizzato e di fornire suggerimenti utili al giocatore. Tuttavia, l'utilizzo di LLM presenta alcuni limiti. Tra questi, spiccano le allucinazioni [6], ossia risposte semanticamente corrette ma scollegate dal contesto, e la difficoltà di mantenere un reasoning coerente con le richieste dell'utente [7].

Il Reinforcement Learning (RL) rappresenta una delle metodologie più efficaci per insegnare agli agenti a prendere decisioni in ambienti complessi, attraverso un processo di trial-and-error guidato da una reward function. Finora il RL ha mostrato grande potenziale principalmente nel giocare a giochi con obiettivi ben definiti [8]; tuttavia, nei contesti con spazi d'azione ampi e molteplici strategie, come i giochi a turni, definire una reward function ottimale può risultare particolarmente complesso. Questo può portare a un apprendimento lento, con agenti che hanno difficoltà a sviluppare strategie vincenti. Alla luce di questi limiti, nasce la domanda di ricerca che guida la seguente tesi, ovvero:

RQ *E' possibile utilizzare il RL per migliorare il reasoning di un LLM nel suggerimento della prossima azione da eseguire?*

Per rispondere alla RQ, la presente tesi propone una metodologia che integra LLM e RL. In particolare, tale metodologia è stata impiegata in un gioco con sistema di combattimento CTB (Conditional Turn Based) basato su scelte sequenziali, in cui un NPC assiste un agente fornendo consigli, turno dopo turno, sulla prossima azione da intraprendere. Per migliorare la qualità del reasoning delle risposte fornite dall'NPC, è stato introdotto un Reviewer, un LLM progettato per fornire istruzioni per effettuare il rephrasing delle risposte iniziali. Il Reviewer è stato inizialmente addestrato tramite learning supervisionato, utilizzando un dataset costruito a partire dalle risposte generate dall'NPC durante la fase sperimentale. Successivamente, è stato eseguito un fine-tuning tramite Reinforcement Learning, utilizzando l'algoritmo Proximal Policy Optimization (PPO). La reward function è stata progettata per valutare se le istruzioni fornite dal Reviewer fossero corrette e prive di allucinazioni, garantendo così un miglioramento concreto della qualità dei suggerimenti.

Per una valutazione più oggettiva, è stata ideata una metrica di scoring che misura l'efficacia di ogni azione suggerita, tenendo conto delle caratteristiche specifiche delle azioni e dello stato della partita in un dato turno. Le prestazioni dell'NPC sono state valutate attraverso diversi episodi, in cui il nemico selezionava le proprie azioni in modo casuale. Questo approccio ha consentito di rendere le partite più dinamiche e imprevedibili, garantendo una valutazione più robusta delle strategie adottate. Successivamente, è stato condotta un'analisi comparativa per confrontare la metodologia proposta con due scenari distinti, il primo con un agente di RL senza alcun supporto, e il secondo con l'uso dei consigli iniziali forniti dall'NPC senza revisione. La sperimentazione condotta mostra che, con le istruzioni fornite dal Reviewer, l'output dell'NPC si adatta maggiormente all'evoluzione del gioco.

La struttura della tesi è organizzata in cinque capitoli. Nel capitolo 2 viene fornita una panoramica sul Reinforcement Learning e sui Large Language Models, con un approfondimento sullo stato dell'arte relativo all'uso dei LLM nei videogiochi. Viene inoltre discusso come RL e LLM possano essere combinati per affrontare problemi complessi. Il capitolo 3 introduce la metodologia sviluppata, descrivendo nel dettaglio le fasi di progettazione dell'environment, l'implementazione dei moduli e la definizione della metrica di valutazione. Il capitolo 4 è dedicato all'analisi dei risultati ottenuti, evidenziando le prestazioni dei vari approcci e con-

frontando l'efficacia delle strategie proposte. Infine, il capitolo 5 presenta le conclusioni del lavoro svolto e discute le possibili direzioni per sviluppi futuri.

Capitolo 2

Background e Stato dell'arte

All'interno dei videogiochi, gli NPC ricoprono un ruolo fondamentale e possono essere suddivisi in diverse categorie in base alla loro funzione e comportamento. Esistono, ad esempio, nemici che attaccano il giocatore, alleati che lo supportano in battaglia o durante missioni specifiche, personaggi secondari che assegnano incarichi o vendono oggetti, e comparse che popolano l'ambiente per renderlo più realistico. Anche il comportamento degli NPC può variare: alcuni sono statici, con dialoghi limitati e privi di interazione significativa con l'ambiente; altri seguono percorsi predefiniti o agiscono come pattugliatori pronti ad attaccare; infine, vi sono NPC dinamici, che reagiscono in modo più imprevedibile, contribuendo a un'esperienza di gioco più immersiva. In giochi con combattimenti a turni, gli NPC possono assumere ruoli ancora più specifici. I compagni di squadra, ad esempio, partecipano attivamente ai combattimenti, contribuendo con abilità uniche e strategie mirate. Altri NPC, come guaritori o supporti strategici, forniscono rispettivamente cure o potenziamenti temporanei per migliorare le capacità del team. Vi sono anche diverse tipologie di nemici che spaziano dai mob, avversari minori incontrati frequentemente, ai boss, nemici particolarmente impegnativi che rappresentano vere e proprie sfide. Infine, altre figure come mercanti, allenatori e informatori offrono al giocatore risorse, miglioramenti o informazioni utili per avanzare nella trama. La ricerca sull'utilizzo combinato di LLM e algoritmi di Reinforcement Learning nel contesto videoludico ha registrato un notevole sviluppo negli ultimi anni [5]. In particolare, numerosi studi sono stati proposti per sfruttare i Large Language Models per migliorare l'efficienza di agenti di Reinforcement Learning (2.3). Tuttavia, l'impiego esclusivo di LLM per migliorare l'interazione degli NPC nei videogiochi è stato finora limitato a piccole produzioni indie [5],

[7], [9] (independent developer [10]). Al contrario, l'uso esclusivo del Reinforcement Learning nei videogiochi per migliorare gli NPC è un campo ancora poco esplorato, ma ultimamente tra le grandi software house si sta iniziando a guardare all'uso del Reinforcement Learning per migliorare lo sviluppo dell'IA nei videogiochi, sebbene sia ancora presto per vederlo in azione in una produzione AAA, ovvero videogiochi che richiedono un grande budget per la produzione e il marketing [10].

In questo capitolo verranno introdotti il Reinforcement Learning e i Large Language Models, con particolare attenzione alle loro rispettive applicazioni. Infine, saranno descritte le soluzioni esistenti che li integrano per risolvere specifici problemi.

2.1 Reinforcement Learning

Il Reinforcement Learning è una branca del machine learning, in cui un agente interagisce con un ambiente al fine di apprendere come comportarsi per raggiungere un obiettivo specifico. In questo modo l'agente impara a prendere decisioni anche in scenari complessi, in cui i dati di training potrebbero essere inesistenti [11].

Come mostrato in Fig. 2.1 l'agente interagisce con l'ambiente, sperimentando diverse azioni e ricevendo feedback e ricompense in base alle conseguenze di tali azioni. Questa interazione è continua e iterativa, e l'agente apprende attraverso l'esperienza. Il compito dell'agente è massimizzare la ricompensa cumulativa, ovvero la somma delle varie ricompense ad ogni iterazione.

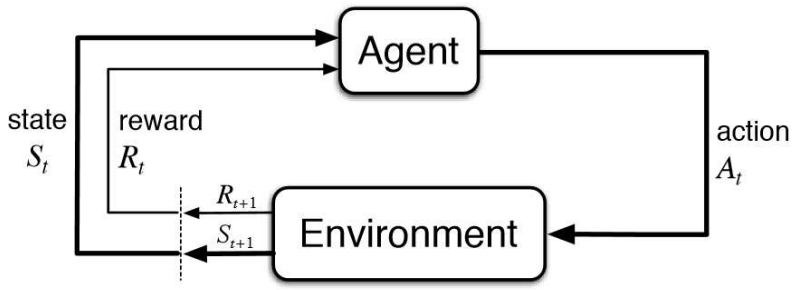

Figura 2.1: Meccanismo del Reinforcement Learning [11].

Un agente basato su Reinforcement Learning è composto da varie componenti chiave, che cooperano per guidare il comportamento dell'agente e supportare il processo decisionale. Tali componenti sono:

- **Policy** π: rappresenta la strategia del piano di azione dell'agente. È una funzione che indica all'agente quale azione dovrebbe eseguire in una determinata situazione o in un determinato stato dell'ambiente. L'obiettivo dell'agente è apprendere la policy che massimizzi la sua ricompensa complessiva nel tempo.

  Vi sono due tipologie di policy:

  1. Policy Deterministica: a ciascuno stato dell'agente è associata un'azione specifica. In questo caso, la policy è una funzione diretta che mappa gli stati alle azioni e viene rappresentata come $\alpha = \pi(s)$, dove α rappresenta l'azione ed s lo stato.
  2. Policy Stocastica: invece di avere un'azione specifica per ciascuno stato, la policy assegna una distribuzione di probabilità su un insieme di azioni possibili per ciascuno stato. Quando l'agente si trova in uno stato, la policy stocastica fornisce la probabilità di eseguire ciascuna azione disponibile in quel determinato stato.

     Ciò consente una maggiore flessibilità nell'esplorazione e nell'adattamento a situazioni incerte. Tale policy viene rappresentata come: $\pi(a \mid s) = P[A_t = a \mid S_t = s]$, dove s rappresenta lo stato, α rappresneta l'azione e P rappresenta la probabilità di eseguire l'azione α nello stato s.

- **Value Function** $v_\pi(s)$: funzione che assegna un valore numerico a ciascuno stato o coppia stato-azione nell'ambiente. Tale valore indica quanto è vantaggioso trovarsi in uno stato specifico o intraprendere una data azione in uno stato.

  Essa permette di valutare le conseguenze a lungo termine delle azioni intraprese dall'agente. Tale funzione può essere espressa in due varianti:

  1. State Value Function: valuta i singoli stati
  2. Action Value Function: valuta le coppie stato-azione

- **Fattore di sconto** γ: è un elemento chiave nel Reinfercement Learning.

  Si basa sulla considerazione che, in molti scenari, le ricompense future dovrebbero avere un valore inferiore rispetto alle ricompense immediate.

  Il motivo di questo sconto delle ricompense future è che, in generale, una ricompensa ottenuta in un momento futuro è considerata meno preziosa di una ricompensa ottenuta immediatamente.

  È un numero compreso tra 0 e 1, solitamente stabilito a priori. Più si avvicina a 1, meno

viene scontato il valore delle ricompense future, il che significa che l'agente darà più peso alle ricompense future, infatti la valutazione viene definita lungimirante. Viceversa, se è più vicino a 0, le ricompense future saranno scontate in misura maggiore, quindi l'agente darà maggior peso alle ricompense immediate. In questo caso la valutazione viene definita miope. Il suo valore dipende dal contesto specifico dell'applicazione e dalle preferenze dell'agente.

- **Modello**: è una rappresentazione dell'ambiente che simula o prevede come si evolverà l'ambiente in risposta alle azioni dell'agente.
  Vi sono due componenti principali in un modello:

  1. Modello di transizione di stato $\mathcal{P}$: parte del modello che predice come evolverà lo stato dell'ambiente in risposta alle azioni dell'agente. Calcola la probabilità condizionata che l'ambiente passi da uno stato s ad uno stato s', data l'azione α.

  $$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a].$$

  2. Modello di ricompensa $\mathcal{R}$: parte del modello che prevede la ricompensa immediata che l'agente riceverà dopo aver eseguito un'azione specifica in uno stato specifico.

  $$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a].$$

Nel Reinforcement Learning, i **Markov Decision Process** vengono utilizzati per descrivere l'ambiente in cui l'agente interagisce [11].
Un Markov Decision Process è formalmente definito da:

- Insieme finito di stati S
- Insieme finito delle azioni A
- Matrice di transizione di stato $\mathcal{P}$: descrive la probablità di transizione da uno stato ad un altro eseguendo una specifica azione.
- Funzione di ricompensa R
- Fattore di sconto γ

In un MDP lo stato successivo S_{t+1} dipende solo dallo stato attuale S_t e non è influenzato da tutti gli stati precedenti, $S_1, S_2, ..., S_{t-1}$. Inoltre, lo stato corrente di un MDP raccoglie tutte le informazioni necessarie per prevedere il futuro, quindi non è necessario conservare l'intera sequenza di stati, azioni e osservazioni precedenti.
In tale modo viene semplificato il calcolo delle policy e delle value function.

2.1.1 Applicazioni del Reinforcement Learning nei Giochi

Xuyouyang Fan [8] spiega che, finora, il Reinforcement Learning ha mostrato il suo potenziale nel giocare a giochi che hanno un obiettivo da raggiungere, come giochi di strategia in tempo reale (RTS), platform, giochi di scacchi e altre tipologie simili.
L'applicazione più famosa è AlphaGo, che ha sconfitto i migliori giocatori umani.

AlphaGo

AlphaGo utilizza la Convolutional Neural Network (CNN [12]) per estrarre le caratteristiche dallo stato del gioco e usa quest'ultime per assistere la decisione.
Come mostrato nella figura 2.2, AlphaGo utilizza due Deep Neural Networks (DNN [13]), policy network e value network, per prendere decisioni.

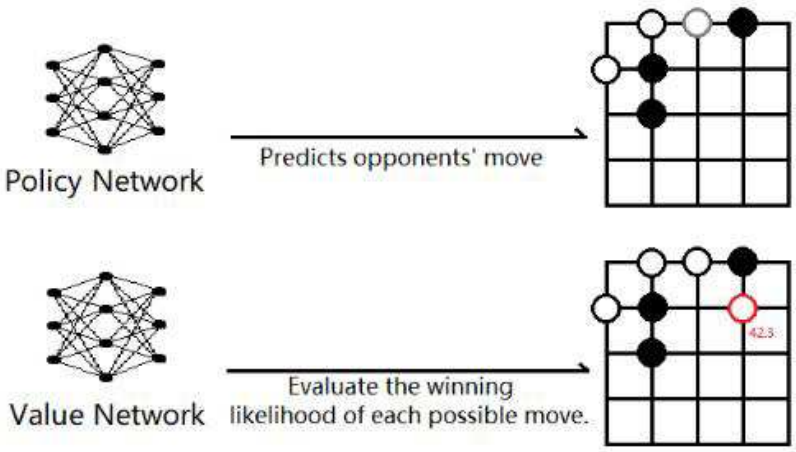

Figura 2.2: Come AlphaGo prende le decisioni [12].

AlphaStar

Un algoritmo di Reinforcement Learning multi-agente è stato sviluppato da Oriol Vinyals et al [8], utilizzando dati tratti sia dai giocatori umani che dai giocatori computerizzati per applicazioni in videogiochi RTS.
L'agente AlphaStar è stato addestrato usando una combinazione di apprendimento supervisionato e Reinforcement Learning. Tuttavia, nel contesto del Reinforcement Learning, sono stati affrontati alcuni problemi dovuti, da un lato, alla complessità delle mappe e alla scarsità di ricompense, che hanno reso l'esplorazione difficile per gli agenti; dall'altro lato, il gioco aveva una lunga durata e lo spazio d'azione delle unità era complesso, rendendo l'apprendimento off-policy un compito difficile.

Deep Q Network

Un tipo di agente intelligente chiamato Deep Q Network (DQN) è stato sviluppato da Volodymyr Mnih et al. [8] per applicazioni in giochi platform. Il DQN combina Reinforcement Learning con DNN, che ha risolto il problema dell'instabilità nel Reinforcement Learning quando si utilizzano approssimatori di funzioni non lineari.
In questo caso, la DNN sostituisce il ruolo della Q function.

Applicazione nei MOBA

Il team sperimentale di Tencent ha sviluppato un modello AI dividendo la sua architettura di deep learning in quattro sottomoduli:

- Server di Intelligenza Artistica
- Modulo di Dispatch
- Memory Pool
- Reinforcement Learning Learner

Per modellare le decisioni di azione nei MOBA (Multiplayer Online Battle Arena), hanno progettato una rete neurale attore-critico.
La rete è stata ottimizzata utilizzando un multilabel Proximal Policy Optimization (PPO [14]) e ha incorporato tecniche come i metodi di disaccoppiamento per le dipendenze delle azioni, i

meccanismi di attenzione per la selezione dei bersagli, le maschere d'azione per un'esplorazione efficiente, Long Short Term Memory (LSTM [15]) per l'apprendimento delle combinazioni di skills e una versione avanzata del Proximal Policy Optimization, chiamata dual-clip PPO, per migliorare la convergenza dell'addestramento.

2.2 Large Language Models

Per Large Language Models (LLM) si intendono reti neurali di grandi dimensioni che vengono addestrate su grandi quantità di dati testuali [16].

Negli ultimi anni, nell'ambito del Machine Learning, i Large Language Models hanno guadagnato crescente attenzione, grazie alle loro capacità di poter comprendere e generare linguaggio naturale con un alto grado di accuratezza e fluidità.

I Large Language Models sono in grado di eseguire una varietà di compiti legati al Natural Language Processing (NLP [17]) quali, ad esempio, text classification, document summarization, question answering, text generation, sentiment analysis [16].

Questi modelli utilizzano l'architettura dei Transformer [18], composta da due componenti principali, ovvero l'encoder e il decoder (Fig. 2.3).

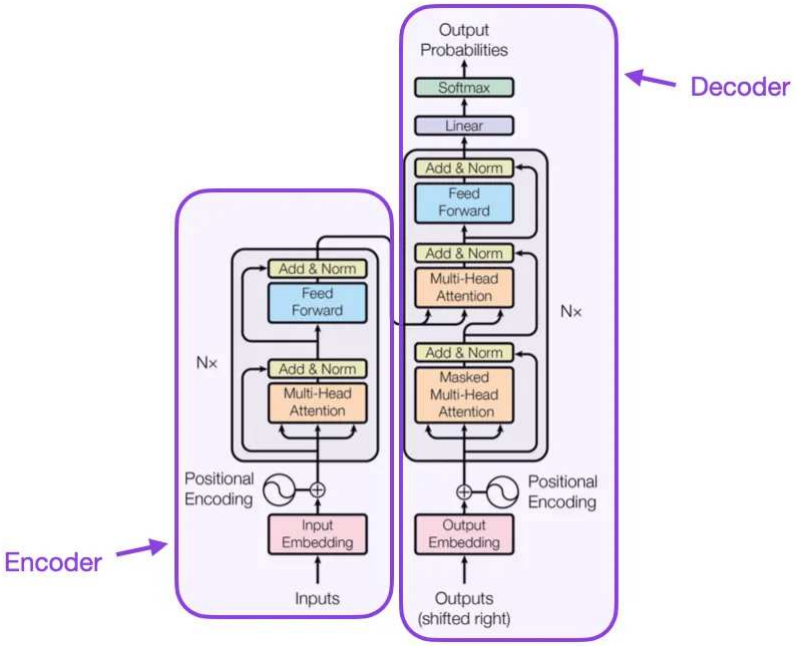

Figura 2.3: Architettura dei Transformer [M. Seiranian].

L'encoder elabora il testo in input, mappando ogni parola in una rappresentazione vettoriale

che incorpora il contesto delle parole vicine, mentre il decoder genera il testo in output, passo dopo passo, basandosi sulle rappresentazioni create dall'encoder.
I Transformer, rispetto alle RNN e le LSTM, utilizzano il layer di self-attention, che permette al modello di poter considerare al meglio il contesto delle parole, e determinare cosa è maggiormente rilevante.

2.2.1 Applicazione dei Large Language Models nei Giochi

Roberto Gallotta et al [19] evidenziano il crescente interesse nell'uso dei LLM in vari settori, tra i quali il settore videoludico.
Qui i LLM possono assumere diversi ruoli, quali ad esempio personaggi non giocanti, assistenti per il giocatore [19], game master [20] o design assistant [21].

Vaudeville

L'utilizzo dei soli LLM all'interno dei videogiochi per migliorare l'interazione degli NPC è un fenomeno emerso negli ultimi anni solo per piccole produzioni come, ad esempio, Vaudeville [7], videogioco in cui il giocatore utilizza input di linguaggio naturale per comunicare con gli NPC al fine di risolvere un misterioso delitto (Fig. 2.4).

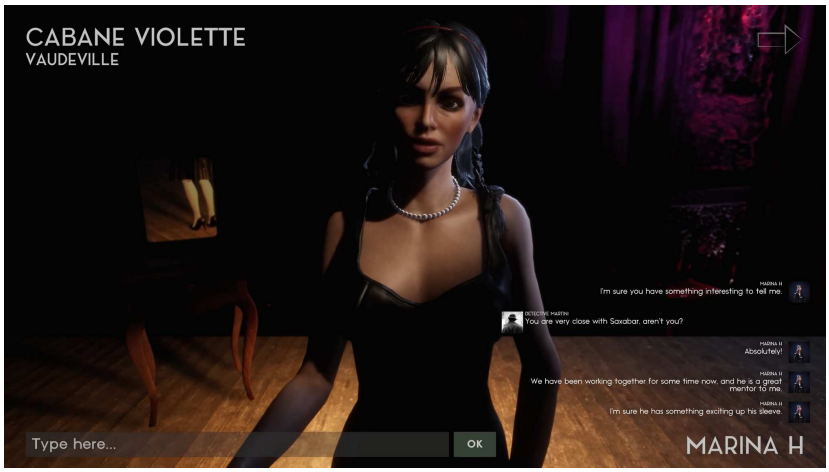

Figura 2.4: Screenshot di una conversazione in Vaudeville [7].

Lo studio evidenzia le limitazioni riscontrate dai giocatori in merito alle allucinazioni, ovvero output che sono sintatticamente corretti ma sono disconnessi dalla realtà e basati su falsi presupposti, e la mancanza di memoria dei LLM.

Tali problematiche, uniti alla natura meno strutturata delle conversazioni aperte, hanno portato alcuni giocatori a segnalare difficoltà nel tracciare e discernere informazioni significative dalle conversazioni, così come difficoltà nel decidere quali percorsi di conversazione perseguire con gli NPC.

2.3 Reinforcement Learning e Large Language Models

Numerosi studi presenti in letteratura mostrano come il Reinforcement Learning e i Large Language Models possano essere combinati per affrontare e risolvere una varietà di compiti. Bin Hu et al [22] propongono When2Ask, un approccio per determinare quando è necessario interrogare i Large Language Models per istruzioni di alto livello, al fine di completare un task. When2Ask utilizza il framework Planner-Actor-Mediator (Fig. 2.5) ed ottimizza i tempi di interazione, riducendo i costi di comunicazione e migliorando l'efficacia nell'esecuzione dei compiti target. Il Framework Planner-Actor-Mediator facilita l'interazione tra un agente e un pianificatore basato su LLM in ambienti di decisione complessi.

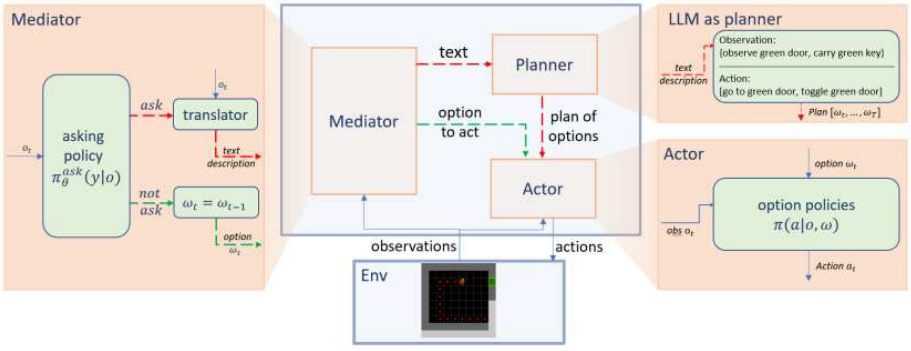

Figura 2.5: Esempio di interazione del Planner-Actor-Mediator [22].

È composto da tre componenti:

- Planner: genera istruzioni di alto livello usando un LLM pre-addestrato, basandosi sullo stato attuale dell'ambiente.
- Actor: traduce le istruzioni del pianificatore in azioni concrete e di basso livello, che l'agente esegue.
- Mediator: coordina la comunicazione tra planner e actor, decidendo quando l'agente dovrebbe chiedere nuove istruzioni.

La decisione relativa a quando l'agente dovrebbe chiedere istruzioni è basata su una policy di richiesta, addestrata con RL, minimizzando le interazioni inutili e massimizzando il successo delle azioni dell'agente.

Il problema è formulato come un Markov Decision Process, con azioni "Chiedi" e "Non chiedere".

La policy di richiesta viene addestrata usando il Proximal Policy Optimization per richiedere nuovi piani solo quando necessario, bilanciando il successo del compito con le penalità per richieste superflue. La metodologia proposta, rispetto a When2Ask, mira non solo a tradurre le istruzioni di alto livello in azioni concrete ma analizza anche l'efficacia delle strategie adottate; inoltre, mostra come il PPO possa essere utilizzato per migliorare progressivamente la qualità delle indicazioni fornite.

Chaofan Pan et al [23] propongono il framework Hi-Core per facilitare il trasferimento di conoscenze tra compiti diversi nel Reinforcement Learning Continuo.

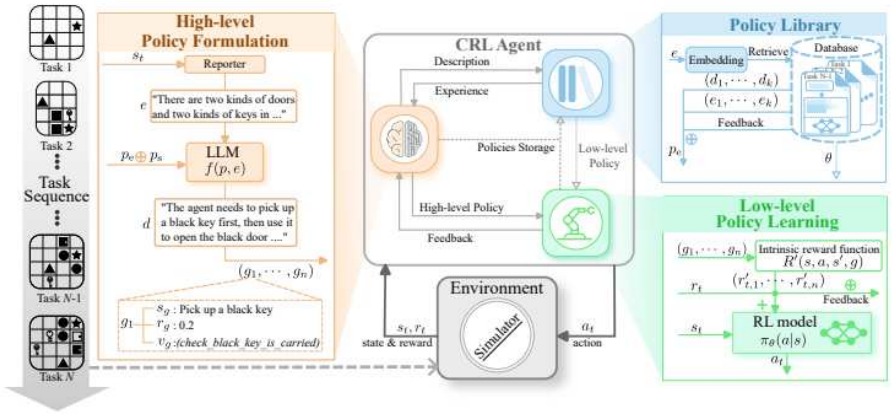

Figura 2.6: Struttura di Hi-Core [23].

È strutturato su due livelli principali (Fig. 2.6):

- Policy ad alto livello: utilizza un LLM per mappare descrizioni testuali dell'ambiente e prompt in una sequenza di obiettivi. Gli obiettivi includono descrizione testuale, funzione di verifica e valore di ricompensa intrinseca. La policy ad alto livello è iterativamente regolata dal LLM attraverso il feedback.

- Policy a basso livello: guida l'agente utilizzando input di stato grezzo e ricompense intrinseche basate sugli obiettivi. La ricompensa intrinseca totale è calcolata combinando ricompense interne ed esterne, incentivando l'agente a imparare una policy allineata con gli obiettivi.

Hi-Core dispone di una libreria di policy che memorizza policy e feedback per recuperare esperienze simili in compiti futuri.

In questo studio, alcune limitazioni riscontrate sono una soluzione non soddisfacente per il rischio di catastrophic forgetting [24], nonché ulteriori margini di miglioramento nella trasferibilità delle policy a basso livello. In risposta alla catastrophic forgetting, la metodologia proposta utilizza LLM insieme al RL per ridurre il rischio di decisioni subottimali, mitigando il rischio di catasrophic forgetting.

Zihao Zhou et al [25] propongono il framework LLM4Teach, in cui utilizzano un LLM preaddestrato per accelerare il processo di addestramento di un agente studente basato su Reinforcement Learning su piccola scala, specializzato per un compito target.

L'obiettivo del framework è permettere all'agente studente di apprendere rapidamente una policy efficace per prendere decisioni in tempo reale e completare il compito assegnato (Fig. 2.7).

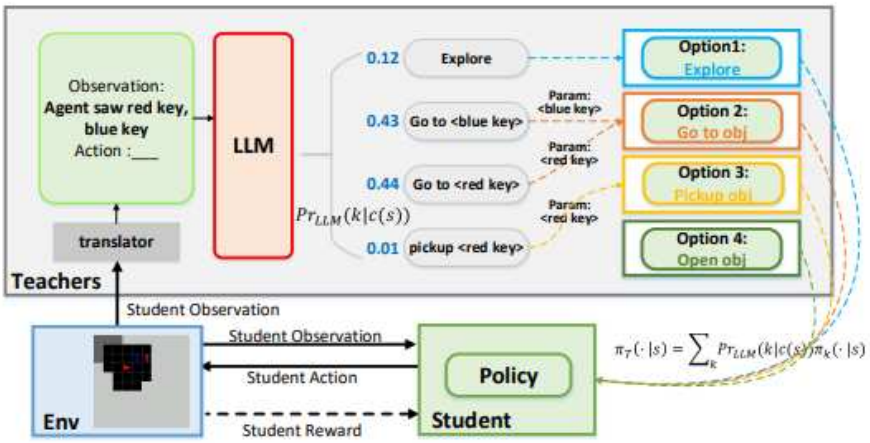

Figura 2.7: LLM4Teach utilizzato nell'environment MiniGrid [25].

Il processo di addestramento avviene in diverse fasi:

- Interazione con l'agente insegnante LLM: l'agente studente richiede indicazioni all'agente insegnante LLM, che fornisce una decisione soft basata su una descrizione testuale dello stato corrente.
- Generazione di istruzioni consapevoli dell'incertezza: l'agente insegnante fornisce istruzioni che incorporano l'incertezza, migliorando l'efficienza dell'apprendimento dell'agente studente.

- Apprendimento della policy: l'agente studente apprende la sua policy, minimizzando una loss function che include una componente di RL e un termine che misura le differenze con la policy dell'insegnante LLM.
- Controllo dell'influenza dell'insegnante: un parametro di riscaldamento λ controlla l'influenza dell'insegnante, riducendosi gradualmente durante l'addestramento per permettere all'agente studente di focalizzarsi sul ritorno atteso.
- Fase di test: quando λ raggiunge 0, l'agente studente opera autonomamente basandosi solo sul feedback dell'ambiente.

Nello studio viene riportato che sia LLM che RL hanno limitazioni nel gestire problemi complessi di decisioni sequenziali.
Il RL spesso manca di efficienza campionaria e comporta costi elevati di esplorazione, mentre i LLMs sono soggetti a errori decisionali e costi elevati di implementazione. In merito alle seguenti limitazioni, la metodologia proposta integra l'uso combinato di due modelli di LLM al fine di ridurre gli errori decisionali. Inoltre, l'architettura non si limita a un supporto temporaneo ma fornisce un feedback continuo, in modo tale da ridurre i costi elevati di esplorazione del RL.

ZihaoWang et al. [26] indagano la sfida della pianificazione delle attività per agenti incorporati multi-task in ambienti open-world, proponendo Describe Explain Plan Select (DEPS) un approccio interattivo alla pianificazione basato su LLM.

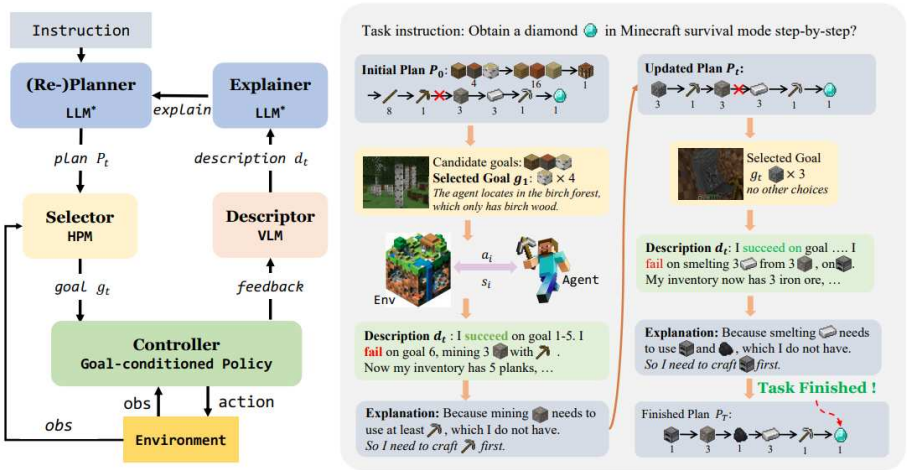

Figura 2.8: Architettura di DEPS [26].

DEPS è composto da quattro componenti (Fig. 2.8):

- Descriptor: responsabile della raccolta di informazioni sull'ambiente corrente e sull'esecuzione dei compiti. Queste informazioni vengono trasformate in un testo descrittivo e inviate al resto del sistema per elaborare i piani successivi. Un LLM viene utilizzato per elaborare la descrizione del Descriptor, rendendola comprensibile e interpretabile per il sistema.

- Explainer: incaricato di analizzare e interpretare il feedback fornito dal Descriptor. Esamina la situazione corrente e identifica le cause degli eventuali fallimenti nell'esecuzione dei task. Utilizza un LLM per comprendere il contesto della situazione, individuare le cause dei fallimenti passati e fornire spiegazioni comprensibili al Planner per consentire la correzione dei piani.

- Planner: responsabile della generazione dei piani per raggiungere obiettivi specifici. Usa le informazioni fornite dal Descriptor e dall'Explainer per elaborare strategie ottimali. Quando viene identificato un errore o un fallimento nell'esecuzione del piano corrente, il Planner utilizza le spiegazioni fornite dall'Explainer per aggiornare e migliorare il piano esistente. Utilizza un LLM per elaborare i piani, decomporre i compiti in sotto-obiettivi e garantire che siano eseguibili.

- Selector: usa un LLM per valutare i piani generati dal Planner e seleziona il percorso più efficiente e appropriato da seguire.

Vengono riportate due limitazioni principali all'interno del metodo. Il framework si basa su LLM detenuti privatamente come GPT-3 e ChatGPT, il che lo rende meno accessibile.
La seconda limitazione è la pianificazione esplicita passo dopo passo nel sistema.
Il collo di bottiglia della pianificazione può anche impedire al modello di essere ulteriormente scalato. Alla luce delle limitazioni descritte, la metodologia proposta si è basata su modelli di LLM open-source, rendendo la soluzione più accessibile e personalizzabile. Inoltre, l'uso coordinato di due LLM, descritto in precedenza, semplifica la gestione dei piani e delle strategie.

Shaoteng Liu et al [27] introducono RL-GPT, un framework che combina LLM e RL per affrontare compiti complessi in ambienti open-world, come il gioco Minecraft.
Questo framework mira a superare le limitazioni dei LLM in ambienti interattivi, sfruttando un approccio gerarchico a due livelli con due tipologie di agenti (Fig. 2.9):

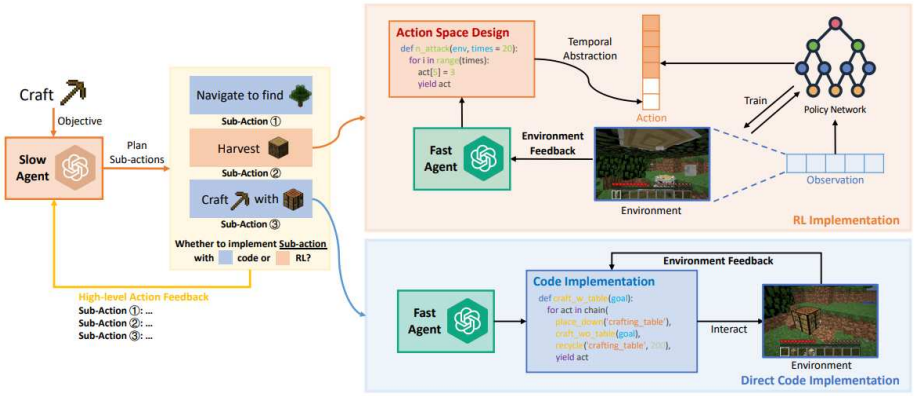

Figura 2.9: Architettura di RL-GPT [27].

- Agente lento: si occupa della pianificazione ad alto livello, decomponendo i compiti complessi in sotto-azioni e determinando quali parti possono essere codificate direttamente e quali richiedono l'uso del RL. Questo agente riduce il carico computazionale e migliora l'efficienza operativa.

- Agente veloce: esegue le sotto-azioni attraverso la scrittura di codice Python ed interagisce direttamente con l'ambiente per implementare le azioni a basso livello. In caso di fallimenti o inefficienze, si affida al feedback dell'ambiente per iterare e migliorare.

Il framework si basa su un processo iterativo, che migliora costantemente il comportamento degli agenti attraverso feedback continui.
Inoltre, un agente critico fornisce valutazioni e suggerimenti per migliorare ulteriormente le prestazioni. RL-GPT risolve uno dei principali problemi dei LLM in ambito open-world, ovvero la Degeneration-of-Thought (DoT [28]), che si verifica quando un modello diventa eccessivamente sicuro delle proprie risposte e non riesce a correggersi.
Attraverso il feedback esterno e la struttura gerarchica del framework, i LLM riescono a eseguire azioni più precise e ad adattarsi dinamicamente all'ambiente, riducendo gli errori. Mentre l'approccio descritto si concentra sulla decomposizione dei compiti complessi e sull'automazione tramite codice, la metodologia proposta sfrutta l'interazione diretta tra due LLM per il supporto decisionale, inoltre, a differenza dell'agente critico usato per la valutazione e successiva correzione, il Reviewer presente nell'architettura proposta fornisce istruzioni che permettono di ridurre suggerimenti errati prima che si verifichino.
Yuqing Du et al [29] utilizzano LLM per guidare l'esplorazione degli agenti di RL. ELLM (Exploring with LLMs) mira a superare una delle principali sfide del RL, ovvero l'esplorazione

efficace in ambienti con ricompense sparse o mal definite (Fig. 2.10).

ELLM sfrutta LLM preaddestrati per generare obiettivi contestualmente rilevanti per gli agenti.

Gli obiettivi sono usati per incentivare l'esplorazione tramite ricompense intrinseche, favorendo lo sviluppo di comportamenti utili senza l'intervento umano.

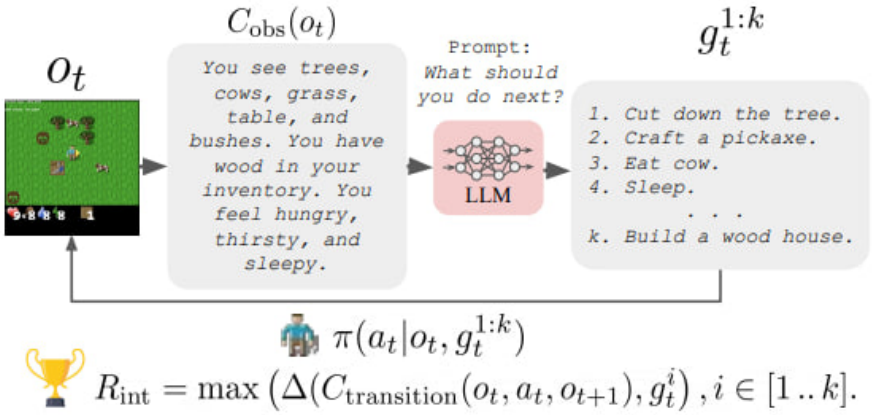

Figura 2.10: Struttura di ELLM [29].

I principali vantaggi di questo approccio sono:

- Diversità degli obiettivi: i LLM generano una gamma di obiettivi variegati, aumentando le possibilità di esplorazione dell'agente.
- Sensibilità al senso comune: gli obiettivi suggeriti sono realistici e rilevanti, permettendo all'agente di esplorare azioni significative.
- Sensibilità al contesto: gli obiettivi sono adattati alla situazione corrente dell'ambiente, migliorando l'efficienza nell'apprendimento.

ELLM utilizza un modello come SentenceBERT per calcolare la similarità semantica tra gli obiettivi generati dai LLM e le azioni intraprese dall'agente, permettendo di determinare se un obiettivo è stato raggiunto e di assegnare una ricompensa.

Questo processo di ricompensa basata sulla similarità semantica permette agli agenti di imparare obiettivi complessi senza necessità di definizioni esplicite delle ricompense.

Il metodo offre due modalità di addestramento:

- Condizionato dagli obiettivi: l'agente riceve una lista di obiettivi e cerca di raggiungerli, facilitando la sua esplorazione in ambienti complessi.

- Senza obiettivi: l'agente esplora liberamente, ma non ha accesso agli obiettivi suggeriti.

Questo approccio permette una copertura più ampia di comportamenti utili e una transizione più fluida verso compiti successivi. Diversamente da ELLM, la metodologia proposta cerca di fornire feedback specifici che guidano un giocatore verso una strategia di gioco ottimale. Inoltre, mentre ELLM utilizza la similarità semantica per valutare se un obiettivo è stato raggiunto, la metodologia proposta valuta le azioni intraprese sulla base di una metrica specifiche che considera le risorse dell'environment. Martin Klissarov et al [30] combinano la conoscenza dei LLM con il RL per esplorare ambienti complessi, utilizzando preferenze derivate dai LLM per creare una ricompensa intrinseca.

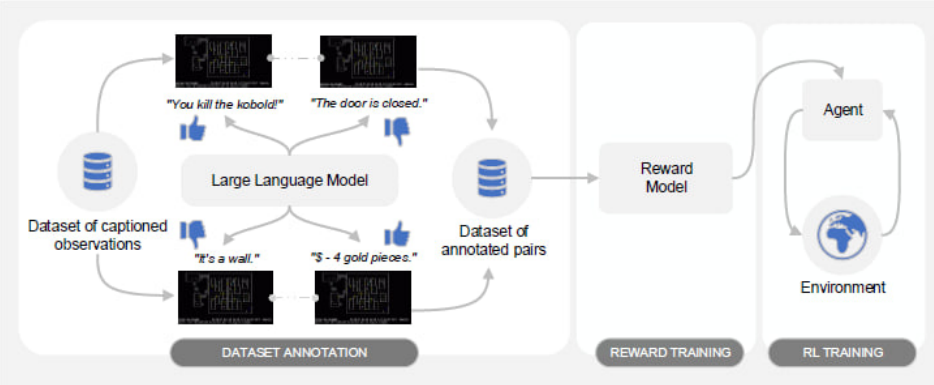

Figura 2.11: Fasi di Motif [30].

Le fasi principali di Motif (Fig. 2.11) sono le seguenti:

- Estrazione delle preferenze: un LLM esprime preferenze su coppie di eventi testuali generati dall'ambiente. Questi eventi sono descrizioni che rappresentano gli stati di gioco.

- Ricompensa intrinseca: le preferenze del LLM vengono utilizzate per estrarre una funzione di ricompensa intrinseca. Questa funzione viene addestrata utilizzando tecniche di RL basate sulle preferenze. Addestramento dell'agente: la ricompensa intrinseca è combinata con la ricompensa estrinseca dell'ambiente, e gli agenti sono addestrati a massimizzare la somma delle due ricompense. Questo processo permette agli agenti di apprendere comportamenti che bilanciano sia l'esplorazione che il raggiungimento degli obiettivi di gioco.

Motif è stato testato nell'ambiente di gioco NetHack Learning Environment (NLE), dove i messaggi di gioco sono utilizzati come input per generare le preferenze del LLM. Motif migliora

le prestazioni degli agenti di RL in ambienti complessi, tuttavia è stato osservato che combinare ricompense intrinseche ed estrinseche può portare a comportamenti non allineati. Alla luce di questa limitazione, la metodologia proposta ha adottato un meccanismo di feedback più controllato e mirato, basato su informazioni chiave dell'environment di gioco. Così facendo, è stato ridotto il rischio di comportamenti non allineati.

Zelai Xu et al [31] presentano un framework che combina il RL con i LLM per creare agenti linguistici strategici, capaci di giocare al gioco del Werewolf, un popolare gioco di deduzione sociale.

Il framework affronta due sfide principali, ovvero il superamento del bias intrinseco nelle decisioni linguistiche degli agenti basati su LLM e la capacità di dedurre informazioni nascoste attraverso la comunicazione ingannevole.

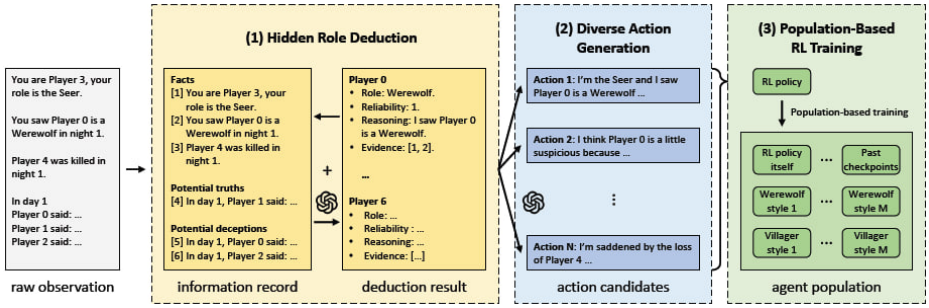

Figura 2.12: Struttura dell'agente strategico [31].

La struttura del framework (Fig. 2.12) è suddivisa in tre fasi:

- Deduzione del ruolo nascosto: viene utilizzato un LLM per organizzare le informazioni provenienti dall'osservazione del gioco, distinguendo tra dichiarazioni veritiere e ingannevoli, al fine di dedurre i ruoli nascosti degli altri giocatori. Questa componente categorizza le informazioni ricevute, strutturando il ragionamento deduttivo e fornendo una base più solida per le decisioni successive.

- Generazione diversificata delle azioni: per mitigare il bias del LLM, gli agenti generano un insieme di azioni candidate anziché una singola azione. Vengono utilizzati due metodi di generazione. Il primo prevede un prompt di base, che genera tutte le azioni in una singola inferenza per azioni semplici. Il secondo prevede un prompt iterativo, che genera azioni una alla volta per situazioni più complesse. In questo modo, l'agente esplora strategie diverse, riducendo il rischio di essere prevedibile.

- Addestramento tramite RL basato sulla popolazione: un agente viene addestrato per scegliere la migliore azione tra i candidati generati. Questo addestramento avviene giocando contro una popolazione variabile di agenti, al fine di migliorare la robustezza e le capacità decisionali dell'agente. Utilizzando una rete di auto-attenzione, l'osservazione e i candidati azione vengono processati insieme per calcolare la distribuzione ottimale delle azioni da intraprendere.

Gli agenti linguistici strategici proposti superano il bias intrinseco dei LLM e sono in grado di eseguire azioni più ottimali rispetto agli agenti basati esclusivamente su LLM.
Il framework mostra come la combinazione di RL e LLM possa produrre agenti più sofisticati, capaci di prendere decisioni strategiche in ambienti complessi e comunicativi. Rispetto al framework descritto, che si concentra sul gioco di ruolo Werewolf, la metodologia proposta crea un NPC che fornisce consigli in giochi CTB. Mentre gli agenti linguistici strategici mirano alla generazione diversificata di azioni al fine di ridurre il rischio di essere prevedibile, l'NPC mira alla creazione di una strategia ottimale, con sequenze di azioni mirate.

Minae Kwon et al [32] affrontano la complessità del reward design nel RL, proponendo un approccio che utilizza i LLM per semplificare il processo di definizione dei reward. Uno dei problemi del RL, nel contesto dei reward, è che è difficile tradurre le preferenze e i comportamenti desiderati degli utenti umani in funzioni di reward precise. Per generare reward in modo più intuitivo e flessibile viene utilizzato il prompting dei LLM, utilizzando pochi esempi o una semplice descrizione verbale del comportamento desiderato.
Questo rende il processo di design dei reward più accessibile, poiché gli utenti possono specificare le loro preferenze con il linguaggio naturale.
Il LLM agisce come una funzione di reward proxy, valutando il comportamento di un agente di RL rispetto alle preferenze descritte, e fornendo un segnale di reward che sarà utilizzato dall'agente per migliorare le sue decisioni.
L'utente fornisce un prompt testuale che descrive i suoi obiettivi (Fig. 2.13).
Durante l'addestramento, un LLM valuta il comportamento dell'agente confrontandolo con gli obiettivi definiti nel prompt e genera un segnale di reward che guida l'agente.
L'agente addestrato tramite LLM si dimostra significativamente più in linea con gli obiettivi degli utenti umani, con un'accuratezza maggiore nelle scelte rispetto agli agenti addestrati con reward functions statiche.

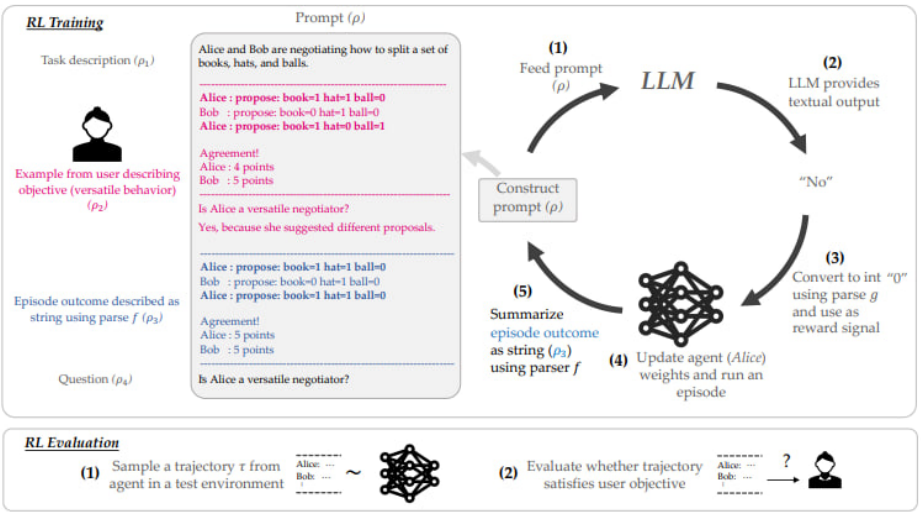

Figura 2.13: Caso di studio per il task DealORNoDeal [32].

La sfida principale riguarda la progettazione del prompt, in quanto il successo del LLM dipende fortemente dalla capacità del modello di interpretare correttamente il prompt fornito dall'utente, il che richiede un certo livello di attenzione al modo in cui viene formulato il testo. Anche la metodologia proposta affronta questa sfida, con un tuning di prompt che ha permesso di individuare i prompt migliori per effettuare le domande al NPC.

Wei Zhou et al [33] affrontano il problema della lenta convergenza degli agenti di RL nei giochi testuali, in cui lo spazio delle azioni è estremamente vasto, proponendo il framework Dialogue Shaping. La lentezza del processo di apprendimento è spesso dovuta alla natura trial-and-error del RL, che costringe gli agenti ad esplorare in modo inefficiente l'ambiente. Vengono utilizzati NPC come fonte di informazioni critiche, capaci di ridurre lo spazio di ricerca e accelerare il processo di addestramento dell'agente.

Vengono integrati LLM per facilitare il dialogo tra l'agente e gli NPC, estraendo informazioni utili per completare obiettivi all'interno del gioco.

Dialogue Shaping prevede l'uso di LLM per simulare sia gli NPC sia l'agente giocatore, sfruttando le capacità linguistiche dei modelli per rispondere a specifiche domande.

Le informazioni ottenute tramite il dialogo vengono convertite in Knowledge Graphs (KGs), che rappresentano lo stato del gioco e la relazione tra gli oggetti, i personaggi e gli eventi.

Il framework utilizza tre componenti principali (Fig. 2.14):

- NPC Prompting: un LLM che simula il comportamento dell'NPC, fornendo risposte dettagliate all'agente basate sulla conoscenza del gioco.

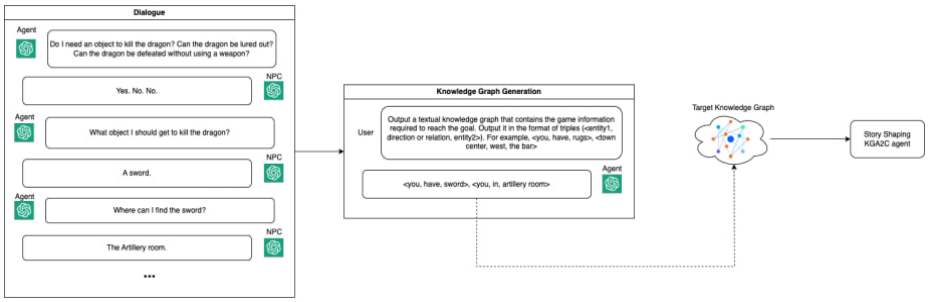

Figura 2.14: Struttura di Dialogue Shaping [33].

- Agent Prompting: l'agente di gioco, anch'esso simulato da un LLM, fa domande al NPC per ottenere informazioni cruciali. L'agente non ha conoscenza diretta del gioco e deve ottenere i dettagli necessari interagendo con l'NPC.

- Story Shaping: le informazioni raccolte dal dialogo vengono utilizzate per creare un KGs target, che funge da guida per l'agente di RL durante il suo addestramento. Il KGs target rappresenta lo stato desiderato per completare il gioco.

Questo approccio permette all'agente di imparare più rapidamente la sequenza ottimale di azioni necessarie per completare il gioco, riducendo così il numero di passi di esplorazione inefficace. Sia il framework descritto che la metodologia proposta contribuiscono a migliorare l'efficacia dell'apprendimento, ma con focus diversi. La metodologia proposta si concentra principalmente sul miglioramento del reasoning di un NPC che, tramite le azioni consigliate, guida un giocatore ad una strategia ottimale.

Houda Nait El Barj et al [34] affrontano il problema del goal misgeneralization nel RL, tramite valutazione dei LLM.

La goal misgeneralization è un tipo di fallimento che si verifica quando un agente di RL, pur mantenendo buone prestazioni al di fuori della distribuzione di addestramento, cerca di raggiungere obiettivi non desiderati, noti come obiettivi sostitutivi.

L'agente generalizza erroneamente l'obiettivo desiderato in un nuovo contesto, nonostante sia stato addestrato correttamente.

Esempi di obiettivi sostitutivi includono un agente che naviga verso una determinata posizione specifica dell'ambiente anziché raggiungere un obiettivo generalizzato, come raccogliere un oggetto specifico.

Lo studio propone l'integrazione dei LLM per analizzare la policy di un agente ed identificare

potenziali scenari di fallimento durante l'addestramento.
Il metodo proposto segue una serie di passaggi chiave:

1. Addestramento iniziale: l'agente di RL viene addestrato per un certo numero di passi temporali in un ambiente predisposto alla goal misgeneralization. Durante l'addestramento, vengono campionati roll-out di policy, ovvero sequenze di coppie stato-azione generate dalla policy dell'agente.

2. Valutazione e suggerimenti di LLM: i LLM vengono utilizzati per esaminare i roll-out delle policy ed identificare possibili scenari in cui la goal misgeneralization potrebbe verificarsi. I LLM forniscono suggerimenti su come correggere l'addestramento.

3. Etichettatura delle preferenze dei LLM: un LLM viene utilizzato per confrontare diverse coppie di roll-out di policy e fornire un feedback su quale delle due è preferibile in termini di generalizzazione dell'obiettivo. Queste preferenze vengono utilizzate per creare un modello di ricompensa basato sulle preferenze del LLM.

4. Riaddestramento dell'agente di RL: una volta costruito il modello di ricompensa, l'agente viene addestrato nuovamente, combinando la funzione di ricompensa originale con il modello derivato dal LLM. Questa nuova funzione di ricompensa consente di correggere i bias comportamentali che impediscono la corretta generalizzazione dell'obiettivo.

Il metodo descritto dimostra che i LLM possono efficacemente supervisionare agenti RL e fornire un feedback utile per mitigare i fallimenti di generalizzazione.
Wen Xiao et al [35] affrontano una delle principali sfide legate ai LLM, ovvero la difficoltà di adattare gli output generativi alle preferenze specifiche degli utenti.
Lo studio propone un framework per adattare i riassunti prodotti dai LLM alle preferenze implicite degli utenti.
Il framework è basato su un processo di generazione tri-agent (Fig. 2.15), così composto:

- Generator: un LLM che produce una prima bozza di riassunto.

- Instructor: un LLM che genera istruzioni personalizzate, mirate a migliorare l'output in base alle preferenze implicite o esplicite dell'utente.

- Editor: revisiona il riassunto basandosi sulle istruzioni dell'instructor, cercando di soddisfare al meglio le aspettative dell'utente.

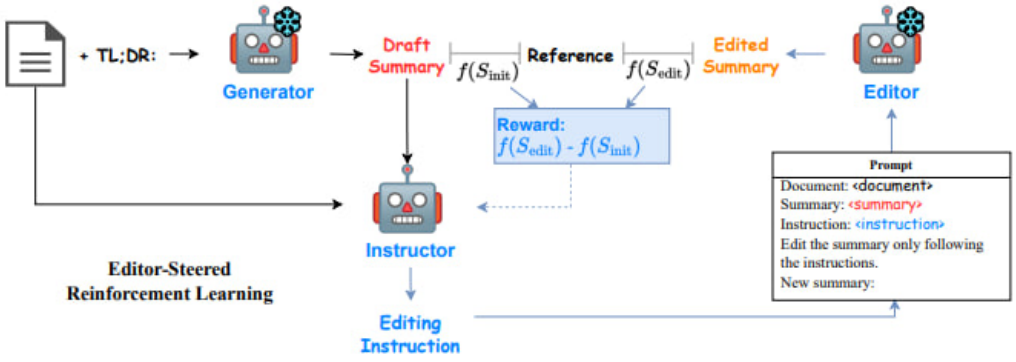

Figura 2.15: Pipeline dei Tri-Agent [35].

Un aspetto chiave di questo framework è l'utilizzo dello stesso LLM per svolgere sia il ruolo di generator che di editor, mentre un modello più piccolo viene impiegato come instructor. L'instructor viene addestrato mediante RL per valutare e ottimizzare le istruzioni generate. Durante l'addestramento supervisionato iniziale, l'instructor apprende sulla base di istruzioni oracolo e, successivamente, viene affinato tramite Reinforcement Learning utilizzando feedback generati dall'editor, con l'obiettivo di migliorare la qualità delle istruzioni.
La funzione di ricompensa viene formulata in modo da massimizzare il miglioramento della qualità del riassunto finale rispetto alla bozza iniziale, misurando la differenza nei punteggi ROUGE e nella factuality tra il riassunto originale e quello modificato.
I risultati mostrano che il framework tri-agent proposto è efficace nel generare riassunti che si allineino meglio con le preferenze dell'utente rispetto ai metodi tradizionali di generazione.
Le iterazioni di editing, supportate da istruzioni generate da un modello istruttore addestrato con RL, permettono di migliorare sia la coerenza fattuale che la copertura dei contenuti, rispondendo alle aspettative degli utenti in modo più accurato. Inoltre, il paper evidenzia come l'integrazione di modelli più piccoli, in questo caso un modello FlanT5-large, per generare istruzioni possa ridurre il costo computazionale, mantenendo elevate prestazioni in termini di personalizzazione. Sia il framework descritto che la metodologia proposta si occupano di migliorare l'output generato da un LLM tramite istruzioni personalizzate. Il framework cerca di adattare i riassunti generati dai LLM alle preferenze degli utenti, mentre la metodologia proposta si concentra sull'ottimizzazione del reasoning di un NPC.

Capitolo 3

Metodologia proposta

In questo capitolo viene presentata la metodologia adottata per il presente studio, partendo dalla descrizione dell'environment di test sviluppato, un sistema basato su combattimenti a turni, ispirato ai classici giochi di ruolo, in cui un utente e un nemico si sfidano alternando azioni strategiche. Verranno illustrate le principali caratteristiche dell'environment, incluse le tipologie di azioni disponibili e le risorse dell'utente e del nemico. Viene poi presentata l'architettura proposta, che combina LLM, per la creazione dell'NPC, con tecniche di RL al fine di migliorare il reasoning dell'NPC. Infine, vengono descritte le singole componenti dell'architettura, descrivendo le tecniche adottate e i dataset utilizzati per il loro sviluppo, oltre alla definizione della metrica impiegata per la valutazione delle performance.

3.1 Environment

Per valutare l'efficacia di un NPC nel fornire suggerimenti sulla prossima azione da intraprendere, è stato sviluppato un environment ispirato alle battaglie a turni dei primi videogiochi di Final Fantasy (Fig. 3.1). La battaglia vede protagonisti un utente, rappresentato da un agente, ed un nemico. L'NPC, denominato Helper, offre a ogni turno consigli all'agente. L'obiettivo è sconfiggere il nemico nel modo più efficace possibile, seguendo i suggerimenti forniti da Helper.

Figura 3.1: Battaglia a turni in Final Fantasy V. Immagine tratta da Tv Obsessive.

Sia l'agente che il nemico sono caratterizzati da:

- HP (health point): punti vita
- MP (magic point): punti magia
- Punti attacco
- Punti difesa
- Magie disponibili
- Items disponibili

Per le magie, l'uso di MP è necessario, mentre gli items hanno una disponibilità limitata. L'agente ha a disposizione nove diverse azioni, una di tipo attacco, cinque di tipo magia e tre items:

1. Attacco: rimuove 300 HP del nemico
2. Fire: magia che rimuove 600 HP del nemico al costo di 25 MP
3. Thunder: magia che rimuove 700 HP del nemico al costo di 30 MP
4. Blizzard: magia che rimuove 800 HP del nemico al costo di 35 MP
5. Meteor: magia che rimuove 1000 HP del nemico al costo di 40 MP

6. Cura: magia che permette di curare 1500 HP dell'agente al costo di 32 MP
7. Potion: item che permette di curare 50 HP del giocatore, in totale ne possiede tre
8. Grenade: item che permette di rimuovere 500 HP del nemico, in totale ne possiede due
9. Elixir: item che permette di recuperare tutti gli HP ed MP del giocatore, in totale ne possiede uno

Il nemico ha a disposizione tre azioni, ovvero l'attacco, la magia fire e la magia cura. Di seguito vengono riportati i valori assegnati all'agente e al nemico (Tab. 3.1):

	Utente	**Nemico**
HP	3260	5000
MP	132	701
Punti attacco	300	525
Punti difesa	34	25

Tabella 3.1: Valori assegnati all'utente e al nemico.

3.2 Architettura

L'architettura proposta comprende tre moduli principali:

- **User**: rappresenta l'utente all'interno dell'environment
- **Helper-LLM**: rappresenta il LLM incaricato di fornire consigli su quali azioni intraprendere.
- **Reviewer-LLM**: è un modello basato su LLM che fornisce istruzioni per migliorare il reasoning del modulo Helper-LLM.

Le fasi che regolano l'interazione tra questi moduli (Fig. 3.2) sono le seguenti:

1. **Generazione del prompt**: ad ogni turno di gioco il modulo User genera un prompt contenente informazioni dettagliate sullo stato dell'environment, inclusa l'ultima azione effettuata dal nemico. Questo prompt viene inviato al modulo Helper-LLM ed al modulo Reviewer-RL.

2. **Risposta iniziale del modulo Helper-LLM**: il modulo Helper-LLM elabora una risposta inziale sulla base del prompt ricevuto. Questa risposta, composta dall'azione consigliata e dal reasoning, viene inoltrata, insieme al prompt iniziale, al modulo Reviewer-LLM.

3. **Analisi e feedback del modulo Reviewer-LLM**: il modulo Reviewer-LLM analizza il prompt e la risposta iniziale del modulo Helper-LLM ed individua eventuali punti non considerati all'interno del reasoning. Sulla base di questa analisi elabora delle istruzioni che vengono restituite al modulo Helper-LLM.

4. **Riformulazione della risposta**: Il modulo Helper-LLM, sulla base delle istruzioni ricevute dal modulo Reviewer-LLM, riformula la risposta iniziale per ottimizzare il reasoning e migliorare la coerenza del suggerimento. La risposta ed il reasoning aggiornati vengono infine forniti al modulo User.

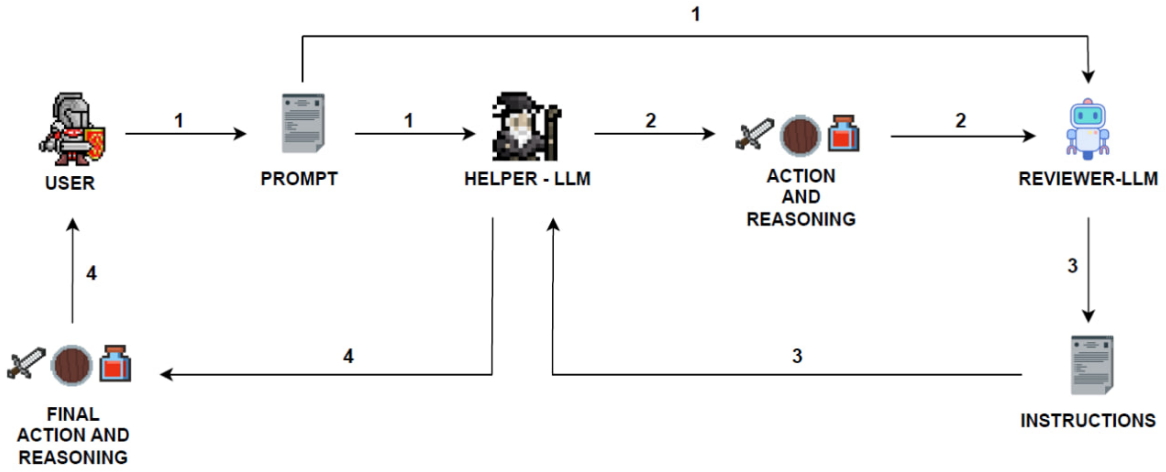

Figura 3.2: Fasi dell'architettura proposta.

Di seguito viene fornita una descrizione delle techiche e delle componenti impiegate per l'implementazione dei singoli moduli.

3.2.1 User

Il modulo User rappresenta un agente che simula il comportamento di un utente all'interno del gioco, eseguendo le azioni suggerite da Helper-LLM. Per ciascuna delle azioni disponibili, precedentemente descritte, sono stati definiti specifici valori di reward (Tab. 3.2). Questi valori

derivano dall'approccio di RL, discusso nel capitolo 4.2.1, utilizzato per l'analisi comparativa condotta.

Azione	**Reward**
Attacco	25
Magie	15
Items	15
Elixir	50

Tabella 3.2: Reward assegnati alle azioni.

Le magie e gli items hanno un valore di 15 poiché il loro utilizzo deve essere attentamente bilanciato con l'attacco, che non consuma MP e può quindi essere impiegato più frequentemente. Elixir, invece, ha un reward più elevato, pari a 50, in quanto rappresenta l'azione con il maggiore impatto positivo per l'agente e deve essere utilizzata attentamente nel corso della partita. Inoltre, è stata assegnata una reward pari a 100 in caso di vittoria, e una penalità di -100 in caso di sconfitta. La complessità del gioco, dovuta alle molteplici strategie possibili, rende difficile attribuire reward negative alle singole azioni, ad eccezione della sconfitta. Ad ogni turno, User genera un prompt che fornisce la descrizione dello stato attuale del gioco. Questo prompt, come descritto in precedenza, viene inviato sia al modulo Helper-LLM che al modulo Reviewer-RL. In particolare, il prompt include informazioni su HP ed MP dell'agente, HP del nemico, l'ultima azione eseguita dal nemico, e le azioni disponibili sulla base degli MP dell'agente. Di seguito viene illustrato un esempio (Tab. 3.3):

Prompt
Player Valos has 3260 Health Points (hp) and 170 Magic Points (mp). Enemy Magus has 5000 Health Points (hp). Available actions: attack deals 300 enemy's hp and removes 0 player's mp; fire spell deals 600 enemy's hp and removes 25 player's mp; thunder spell deals 700 enemy's hp and removes 30 player's mp; blizzard spell deals 800 enemy's hp and removes 35 player's mp; meteor spell deals 1000 enemy's hp and removes 40 player's mp; cura spell heals 1500 player's hp and removes 32 player's mp; potion heals 50 player's hp and there are 3; grenade deals 500 enemy's hp and there are 2; elixir fully restores player's hp and mp and there are 1. Last enemy move was attack.

Tabella 3.3: Esempio di prompt generato da User-RL.

3.2.2 Helper-LLM

Per il modulo Helper-LLM è stato selezionato il LLM FlanT5-Large per consigliare, turno dopo turno, l'azione successiva da intraprendere. FLAN-T5 è un LLM open source rilasciato da Google, sotto licenza Apache, alla fine del 2022 [36] e rappresenta una variante del modello T5 (Text-To-Text Transfer Transformer) [37]. Il T5 venne concepito come un modello text-to-text in cui tutti i task di NLP possono essere rappresentati come stringhe di testo. La variante Flan (Fine-Tuned Language Net) di T5 è stata addestrata su un ampio set di dati che include task diversi, tra cui question answering, problem solving e text-completion. FlanT5 è disponibile in diverse varianti, con dimensioni adattate in base alle esigenze di capacità di calcolo e di precisione del modello:

- FlanT5-Small: 80 milioni di parametri
- FlanT5-Base: 250 milioni di parametri
- FlanT5-Large: 780 milioni di parametri
- FlanT5-XL: 3 miliardi di parametri
- FlanT5-XXL: 11 miliardi di parametri

La scelta di utilizzare FlanT5-Large è stata guidata dalla necessità di bilanciare capacità computazionali e precisione nelle risposte. Ad ogni turno, il modello riceve il prompt aggiornato

sullo stato attuale del gioco ('game description') e le risorse disponibili. La domanda posta al modello (Tab. 3.4) è la seguente:

Prompt
Given the game state 'game description', what is the next action to take? Please write the chosen action like this example, [attack], and explain your reasoning briefly.

Tabella 3.4: Prompt iniziale fornito ad ogni turno ad Helper-LLM.

Al modello è stato richiesto di formattare l'azione consigliata racchiudendola tra parentesi quadre (Tab. 3.5). Questo formato ha consentito di eseguire un mapping diretto tra la stringa generata dal modello e l'azione corrispondente per User. In particolare, il mapping segue il seguente schema:

Stringa	**Azione**
[attack]	0
[fire] / [fire spell]	1
[thunder] / [thunder spell]	2
[blizzard] / [blizzard spell]	3
[meteor] / [meteor spell]	4
[cura] / [cura spell]	5
[potion]	6
[grenade]	7
[elixir]	8

Tabella 3.5: Mapping delle azioni.

Una volta ricevute le istruzioni dal Reviewer-LLM, è stato richiesto al modello di riformulare la sua risposta (Tab. 3.6) seguendo le indicazioni fornite:

Prompt
Given the game state 'game description'. Your initial response was 'initial response'. Considering this suggestion 'suggestion', rephrase your answer

Tabella 3.6: Prompt finale fornito ad ogni turno ad Helper-LLM.

dove:

- game description rappresenta la descrizione del gioco precedentemente fornita;
- initial response rappresenta la risposta iniziale fornita dal modello;
- suggestion rappresentano le istruzioni fornite dal Reviewer-LLM.

Per valutare l'efficacia delle azioni consigliate, è stata sviluppata una metrica che considera sia le caratteristiche delle azioni disponibili sia lo stato attuale della partita. Tale metrica consente di calcolare uno score associato a ciascuna azione, rappresentando il grado di efficacia di ogni azione specifica nel contesto del turno in corso.

$$\text{Score} = \begin{cases} \left(\alpha \times \frac{\mathrm{d}}{\mathrm{hp_n}}\right) - \left(\gamma \times \frac{\mathrm{mp_c}}{\mathrm{mp_g}}\right) & \text{se hp utente} > 30\% \\ \left(\alpha \times \frac{\mathrm{d}}{\mathrm{hp_n}}\right) + \left(\beta \times \frac{\mathrm{hp_r}+\mathrm{mp_r}}{\mathrm{hp_g}+\mathrm{mp_g}}\right) - \left(\gamma \times \frac{\mathrm{mp_c}}{\mathrm{mp_g}}\right) & \text{se hp utente} < 30\% \end{cases}$$

- d: danno inflitto dall'azione scelta
- hp_n: HP del nemico
- mp_c: MP per utilizzare l'azione
- mp_g: MP disponibili del giocatore
- hp_r: HP che possono essere recuperati
- mp_r: MP che possono essere recuperati
- hp_g: HP del giocatore

Sono stati definiti tre pesi, α, β, γ, che permettono di bilanciare l'importanza del danno inflitto, il costo in MP e la cura da utilizzare (Tab. 3.7).

Peso	Valore
α	0.6
β	0.4
γ	0.4

Tabella 3.7: Valori associati ai pesi.

Un'azione di attacco ottiene un punteggio dettato dal rapporto del danno inflitto rispetto agli HP del nemico ($\alpha \times \frac{\mathrm{d}}{\mathrm{hp_n}}$), sottraendo il costo in MP dell'azione rispetto ai punti magia attuali del giocatore ($\gamma \times \frac{\mathrm{mp_c}}{\mathrm{mp_g}}$). Un'azione di cura ottiene un punteggio in cui viene valuta la quantità di HP e MP recuperati, normalizzato rispetto alla somma degli HP e MP attuali del giocatore ($\beta \times \frac{\mathrm{hp_r}+\mathrm{mp_r}}{\mathrm{hp_g}+\mathrm{mp_g}}$). Le azioni di cura vengono considerate nel momento in cui gli HP del giocatore scendono al di sotto di una certa soglia, in questo caso il 30%. Lo score viene infine normalizzato in un invervallo tra 0 ed 1, dove 0 indica un'azione inefficiente, ed 1 azione particolarmente efficiente. Di seguito viene riportato un esempio (Tab. 3.8) in cui si hanno 800 HP del giocatore, 100 MP del giocatore e 2000 HP del nemico:

Attacco	Score
attack	0.06
fire spell	0.05
thunder spell	0.06
blizzard spell	0.07
meteor spell	0.09
cura spell	0.36
potion	0.01
grenade	0.10
elixir	1.0

Tabella 3.8: Esempio di score associati alle azioni.

3.2.3 Reviewer-LLM

Per implementare il modulo Reviewer-LLM è stato adottato un approccio di addestramento a due fasi, per migliorare la qualità delle istruzioni fornite:

1. Learning Supervisionato: nella prima fase il modello è stato addestrato ad analizzare il prompt di gioco e la risposta preliminare fornita dall'NPC, per poter generare istruzioni appropriate. A tal fine, è stato costruito un dataset etichettato contenente coppie di prompt, risposte e relative istruzioni. Questo dataset ha supportato la fase di addestramento iniziale, consentendo al modello di acquisire una comprensione approfondita delle istruzioni da fornire sulla base di diversi scenari di gioco.

2. Proximal Policy Optimization: nella seconda fase il modello è stato sottoposto ad un fine-tuning con Reinforcement Learning tramite la tecnica PPO, utilizzando una parte del dataset precedentemente creato. Questa fase è stata progettata per ottimizzare le prestazioni del modello, riducendo le possibilità di generare risposte contenenti allucinazioni.

Inizialmente, per il modulo Reviewer-LLM è stato selezionato il modello FlanT5-Small. Tuttavia, al termine della prima fase di learning supervisionato, si è osservata una frequente presenza di risposte contenenti allucinazioni. Questo limite ha portato ad una revisione del modello, optando per FlanT5-Large. La scelta di passare ad un modello più grande ha migliorato significativamente le capacità del Reviewer-LLM, riducendo, già dopo la prima fase di training, le risposte contenenti allucinazioni. Di seguito vengono riportati alcuni iperparametri utilizzati durante la fase di learning supervisionato:

- train batch size: 8
- eval batch size: 8
- epoche: 3
- learning rate: 5e-5

Il dataset utilizzato è stato costruito a partire dalle prime risposte generate sperimentalmente da FlanT5-Large ed è organizzato in tre colonne:

- prompt: rappresenta la descrizione del gioco in un turno specifico.
- response: contiene una possibile risposta generata da Helper-LLM.
- instruction: contiene indicazioni per migliorare la risposta iniziale.

Sono state generate casualmente 15000 descrizioni di gioco, in cui sono stati fatti variare gli HP ed MP del giocatore, gli HP del nemico e le azioni disponibili. Le istruzioni sono state generate tenendo conto del potenziale delle azioni disponibili, favorendo quelle che rimuovono il maggior numero di HP al nemico (Tab. 3.9) o che consentono al giocatore di curarsi quando i suoi HP scendono sotto una certa soglia (Tab. 3.10).

Prompt	Response	Instruction
Player has 2739 hp and 49 mp. Enemy has 1101 hp. Available actions: attack, fire spell, thunder spell, blizzard spell, meteor spell, cura spell, potion, elixir. Last enemy move was cura spell	The next action to take is [fire spell]	Consider using [meteor spell] for more damage

Tabella 3.9: Esempio di istruzioni per un attacco migliore.

Prompt	Response	Instruction
Player has 369 hp and 86 mp. Enemy has 1365 hp. Available actions: attack, fire spell, thunder spell, blizzard spell, meteor spell, cura spell, potion, elixir. Last enemy move was cura spell	The next action to take is [attack]	Consider using [elixir] to heal hp and mp

Tabella 3.10: Esempio di istruzioni per considerare una cura.

Sono state utilizzate 10000 istanze per la fase di learning supervisionato e 5000 istanze per la fase di Proximal Policy Optimization. Il Proximal Policy Optimization (PPO) è un algoritmo di Reinforcement Learning che migliora la stabilità e l'efficacia delle policy apprese attraverso un'ottimizzazione graduale e controllata [14]. Si colloca tra gli algoritmi policy-based, dove l'obiettivo è trovare la policy ottimale per ottenere la massima ricompensa possibile in un environment. Prima del PPO, un metodo comune per l'ottimizzazione della policy era il Trust Region Policy Optimization (TRPO) [38], che limitava la distanza tra

policy successive utilizzando una trust region per evitare cambiamenti drastici nella policy. Tuttavia, TRPO si è dimostrato computazionalmente costoso. PPO migliora il TRPO mantenendo il concetto di trust region, ma utilizzando una value function più semplice e facile da implementare. PPO introduce un clipped objective function che consente un controllo più semplice sui cambiamenti nella policy:

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

dove:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ rappresenta il rapporto tra la probabilità di selezionare un'azione specifica a_t nello stato s_t usando la nuova policy rispetto alla probabilità con cui la stessa azione sarebbe stata scelta dalla policy precedente $\pi_{\theta_{\text{old}}}$.
- $\hat{A}_t$ è l'advantage estimator per l'azione a_t nello stato s_t, che indica quanto sia vantaggiosa rispetto alla media delle azioni possibili nello stato corrente.
- La clip function restringe $r_t(\theta)$ all'intervallo $[1-\epsilon, 1+\epsilon]$, dove ϵ è un iperparametro che controlla la tolleranza dei cambiamenti della policy.

La funzione di clipping riduce la possibilità di aggiornamenti che causino cambiamenti drastici nella policy, rendendo l'addestramento più stabile. Una variante dell'algoritmo del PPO vede l'aggiunta di un termine di penalità basato sulla divergenza KL (Kullback-Leibler) alla value function. La divergenza KL misura quanto si discosta la nuova policy dalla policy precedente, favorendo una stabilità simile a quella ottenuta tramite il clipping.

$$L^{\text{KLPEN}}(\theta) = \mathbb{E}\left[r_t(\theta)\hat{A}_t - \beta\,\text{KL}\left(\pi_{\theta_{\text{old}}} \parallel \pi_\theta\right)\right]$$

dove:

- $\text{KL}(\pi_{\theta_{\text{old}}} \parallel \pi_\theta)$ è la divergenza KL tra la vecchia policy e la nuova policy. Questa misura indica la "distanza" tra le due distribuzioni di probabilità, ossia tra le probabilità delle azioni nella vecchia e nella nuova policy.
- β è un parametro che controlla quanto peso dare alla penalità della divergenza KL.

L'algoritmo PPO alterna tra fasi in cui avviene raccolta di esperienze dall'environment e fasi in cui avviene l'ottimizzazione della policy sulla base di queste esperienze. In particolare, le fasi si suddividono in:

1. Generazione di esperienze: l'agente interagisce con l'ambiente seguendo la policy corrente, raccogliendo esperienze di stati, azioni e ricompense.

2. Aggiornamento della Policy: per ogni batch di esperienze il PPO aggiorna la policy massimizzando il clipped objective.

3. Ripetizione del ciclo: PPO ripete i passi di esplorazione e aggiornamento della policy fino a convergenza.

Il PPO risulta essere particolarmente efficace nell'addestramento di LLM in contesti di Reinforcement Learning from Human Feedback (RLHF) [39]. Il RLHF è una tecnica di machine learning che utilizza il feedback umano per ottimizzare i modelli in modo che autoapprendano in maniera più efficiente. Così facendo i modelli risultano maggiormente allineati agli obiettivi e alle esigenze umane. Diventa quindi fondamentale definire una reward function tramite il feedback umano.

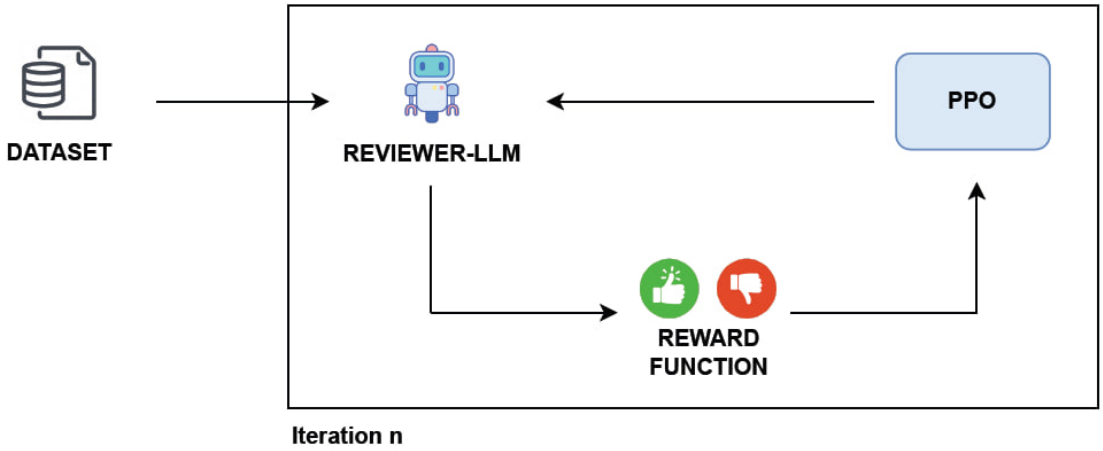

Figura 3.3: Fine-tuning tramite PPO.

Le fasi del fine-tuning del modulo Reviewer-LLM tramite PPO (Fig. 3.3) sono le seguenti:

1. Generazione dei consigli: il modello genera un consiglo a partire dal prompt e dalla response presenti nel dataset

2. Valutazione del consiglio: il consiglio viene confrontato con l'instruction presente nel dataset e la reward function assegna una reward positiva (+5) nel caso in cui l'azione consigliata corrisponde, negativa nel caso in cui non corrisponde l'azione o la risposta presenta allucinazioni (-10)

3. Aggiornamento della policy: dopo aver ottenuto la reward, la policy del modello viene aggiornata

Gli iperparametri utilizzati per l'addestramento sono riportati di seguito:

- learning rate: 5e-7
- mini batch size: 1
- batch size: 1
- epoche: 3

Capitolo 4

Risultati Sperimentali

In questo capitolo verranno illustrati e analizzati i risultati ottenuti, fornendo una risposta alla research question iniziale. La metodologia proposta è stata sottoposta a test su un numero variabile di episodi, rispettivamente 200, 500, 800 e 1000, al fine di valutare l'efficacia del sistema in scenari con differenti gradi di complessità e durata. Successivamente, è stato condotto un test specifico su 500 partite, di cui le prime 150 giocate da un approccio basato su RL, mentre le restanti 350 giocate con i consigli di Helper-LLM e Reviewer-LLM. Questo approccio ha permesso di osservare l'impatto dei suggerimenti sull'evoluzione della strategia dell'agente e sull'efficacia complessiva del sistema. Infine, è stata condotta un'analisi comparativa, in cui è stato testato l'approccio basato su RL isolato e l'efficacia dei consigli iniziali di Helper-LLM, confrontandoli con la metodologia proposta.

4.1 Helper-LLM con istruzioni di Reviewer-LLM

Di seguito sono presentati alcuni esempi di risposte iniziali fornite da Helper-LLM, insieme alle istruzioni generate da Reviewer-LLM e alla riformulazione finale, estratti dalla sperimentazione condotta (Tab. 4.1, 4.2):

Response Helper-LLM	Instruction Reviewer-LLM	Rephrasing Helper-LLM
[Attack] deals 300 enemy's hp and remove 0 player's mp so the answer is attack	You should use [elixir] to fully restore your health and mp	[Elixir] fully restore player's hp and mp so the final answer is elixir

Tabella 4.1: Esempio di riformulazione con 300 hp e 12 mp.

Response Helper-LLM	Instruction Reviewer-LLM	Rephrasing Helper-LLM
[Grenade] deals 500 enemy's hp so the answer is grenade	Consider using [meteor spell] for more damage	[Meteor spell] deals 1000 enemy's hp so the final answer is meteor

Tabella 4.2: Esempio di riformulazione con 2900 hp e 100 mp.

Come si può osservare dai grafici 4.1, 4.2, nella configurazione iniziale con 200 episodi, si registrano risultati promettenti sia in termini di success rate che di vittorie cumulative. In particolare, sono state vinte 129 partite su 200, con una percentuale di vittorie pari al 64,5%.

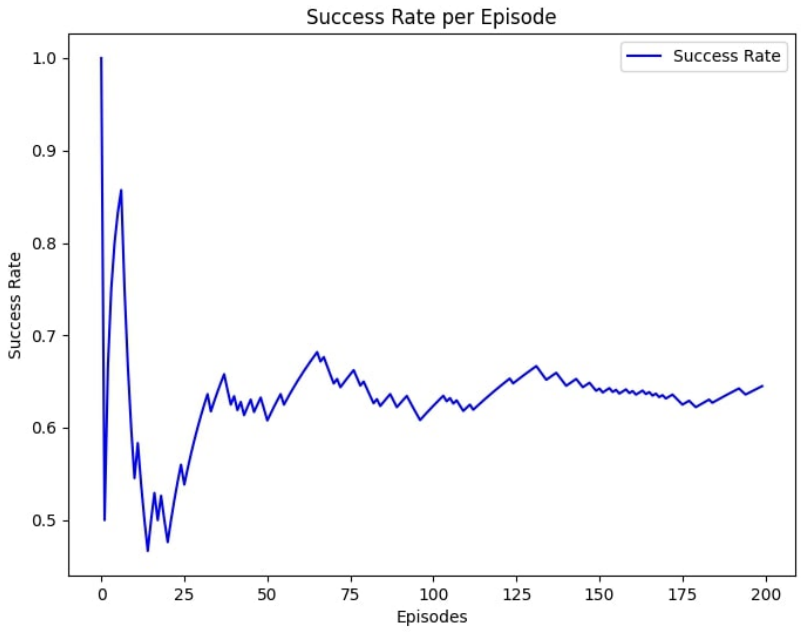

Figura 4.1: Andamento del success rate di Helper-LLM e Reviewer-LLM in 200 episodi.

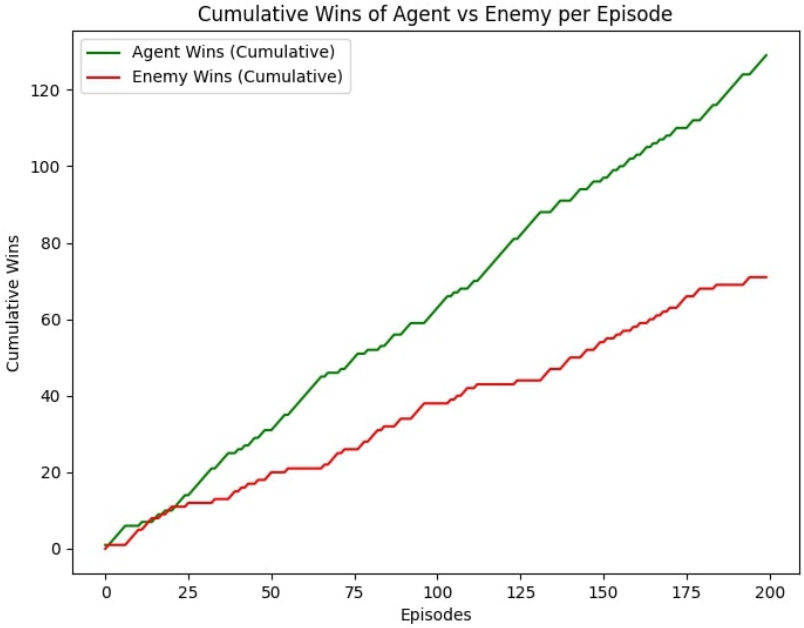

Figura 4.2: Confronto delle vittorie cumulative di Helper-LLM Reviewer-LLM e nemico in 200 episodi.

Anche le ricompense mostrano risultati elevati fin dalle 200 partite (Fig. 4.3), con una media di 259, e una media di mosse effettuate pari ad 11 (Fig. 4.4).

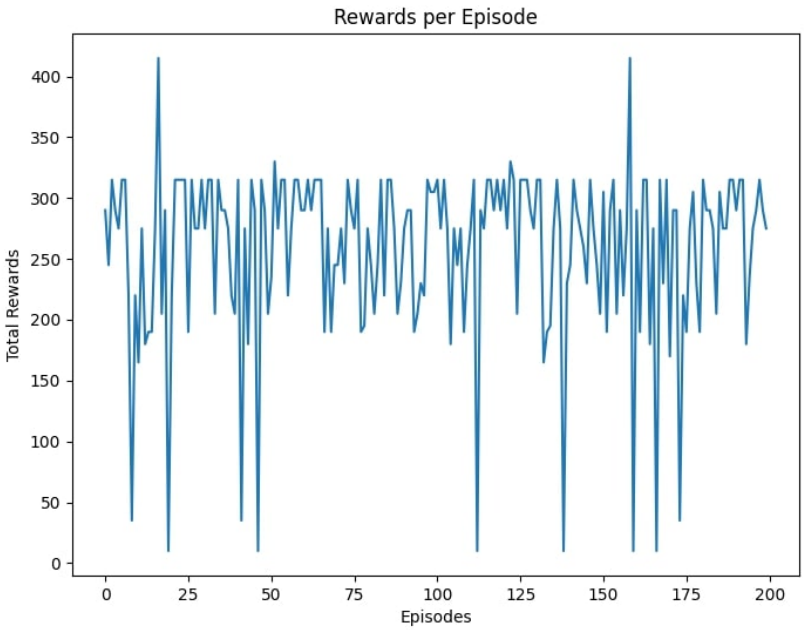

Figura 4.3: Andamento dei reward di Helper-LLM e Reviewer-LLM in 200 episodi.

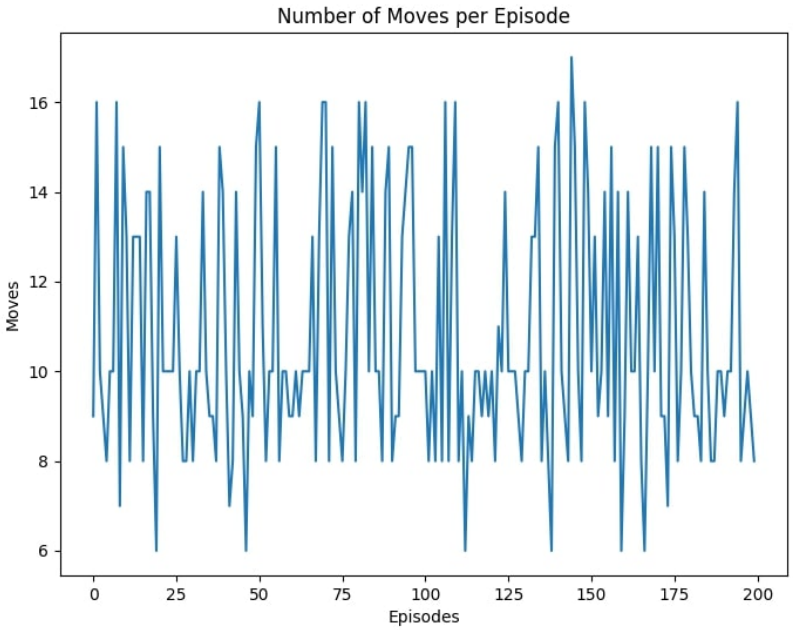

Figura 4.4: Andamento delle mosse effettuate da Helper-LLM e Reviewer-LLM in 200 episodi.

Lo score associato alle azioni finali suggerite mostra un punteggio iniziale ottimale di 0,75. Questo valore indica che le azioni suggerite da Helper-LLM, con il supporto dele istruzioni di Reviewer-LLM, risultano già significativamente efficaci sin dalle prime iterazioni. Di seguito è riportata la tabella riassuntiva (Tab. 4.3) dei risultati ottenuti:

Episodi	Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse	Score
200	129	71	64,5%	259	11	0,75

Tabella 4.3: Risultati ottenuti da Helper-LLM e Reviewer-LLM in 200 episodi.

Con 500 episodi, si è osservata una leggera diminuzione del success rate rispetto ai 200 episodi, registrando un valore del 63,8%, equivalente a 319 vittorie su 500 (Fig. 4.5, 4.6). Durante i 500 episodi sono stati registrati due casi di allucinazione da parte di Reviewer-LLM, in cui le istruzioni fornite riguardanti l'azione "[potion]" venivano erroneamente etichettate come "[Giant]". Le allucinazioni sono state gestite annullando le istruzioni errate e procedendo con la valutazione dell'azione inizialmente scelta da Helper-LLM.

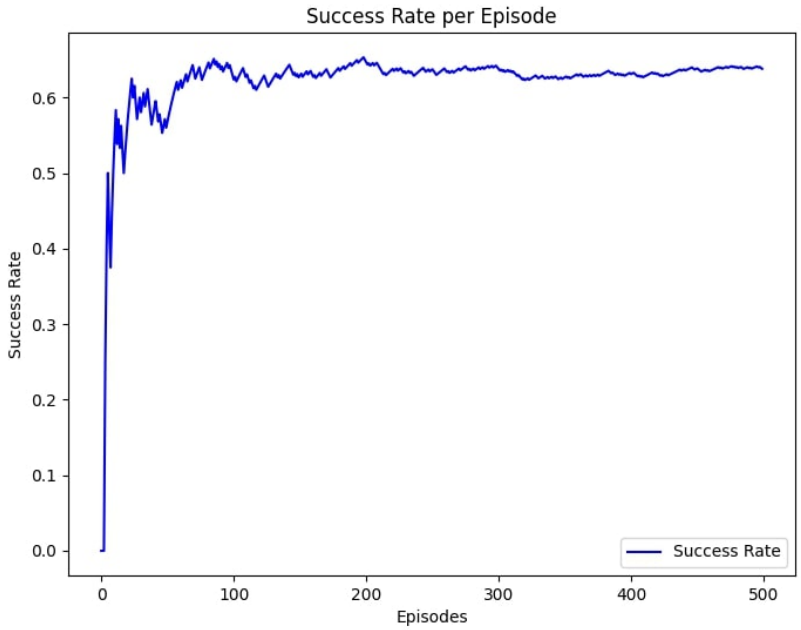

Figura 4.5: Andamento del success rate di Helper-LLM e Reviewer-LLM in 500 episodi.

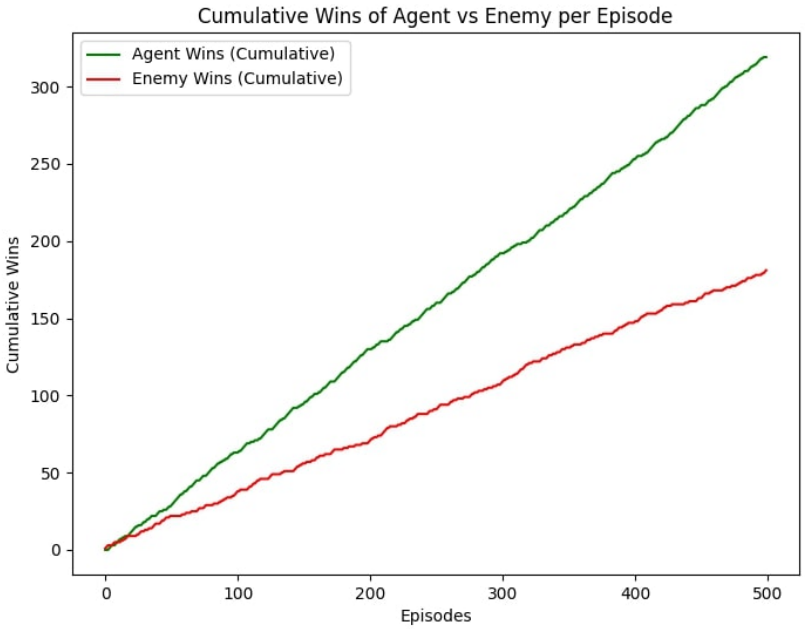

Figura 4.6: Confronto delle vittorie cumulative di Helper-LLM Reviewer-LLM e nemico in 500 episodi.

La media delle ricompense cala leggermente, passando a 258, mentre il numero medio di mosse si stabilizza a 11 (Fig. 4.7, 4.8), come confermato anche nei risultati successivi. Lo score delle azioni consigliate aumenta di 0,01 raggiungendo un valore di 0,76.

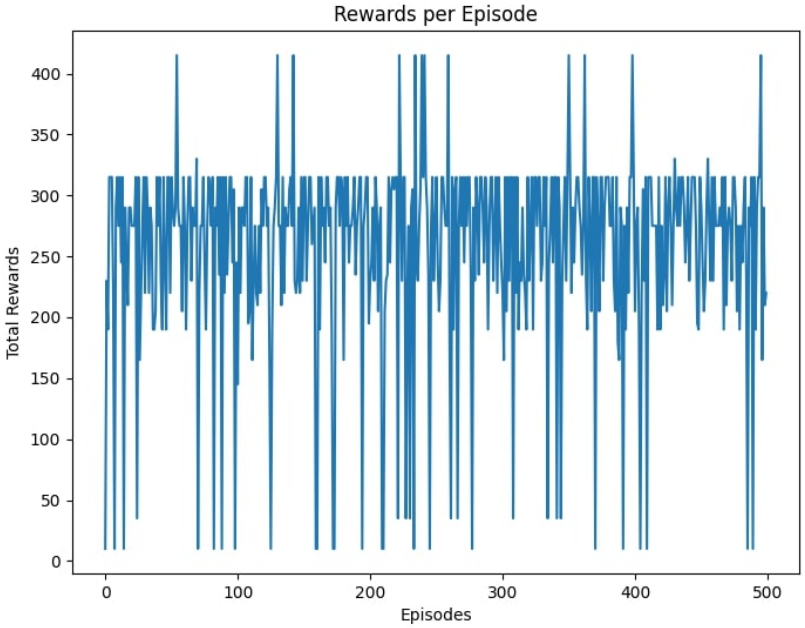

Figura 4.7: Andamento dei reward di Helper-LLM e Reviewer-LLM in 500 episodi.

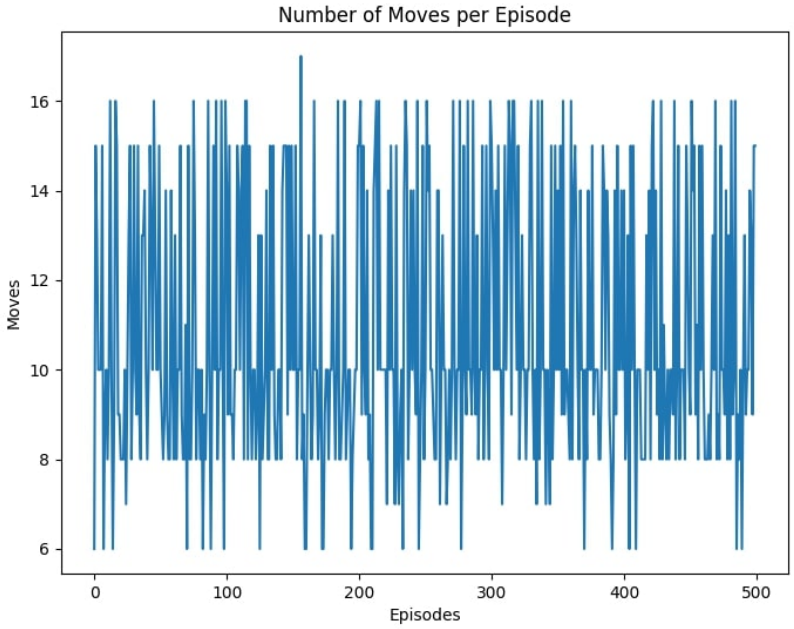

Figura 4.8: Andamento delle mosse effettuate da Helper-LLM e Reviewer-LLM in 500 episodi.

Di seguito è riportata la tabella riassuntiva (Tab. 4.4) dei risultati ottenuti:

Episodi	Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse	Score
500	319	181	63,8%	258	11	0,76

Tabella 4.4: Risultati ottenuti da Helper-LLM e Reviewer-LLM in 500 episodi.

Con 800 episodi è stato ottenuto uno dei risultati migliori in termini di success rate, con una percentuale di vittorie del 65%, corrispondente a 520 partite vinte (Fig. 4.9, 4.10). In questa fase, non sono stati riscontrati casi di allucinazioni.

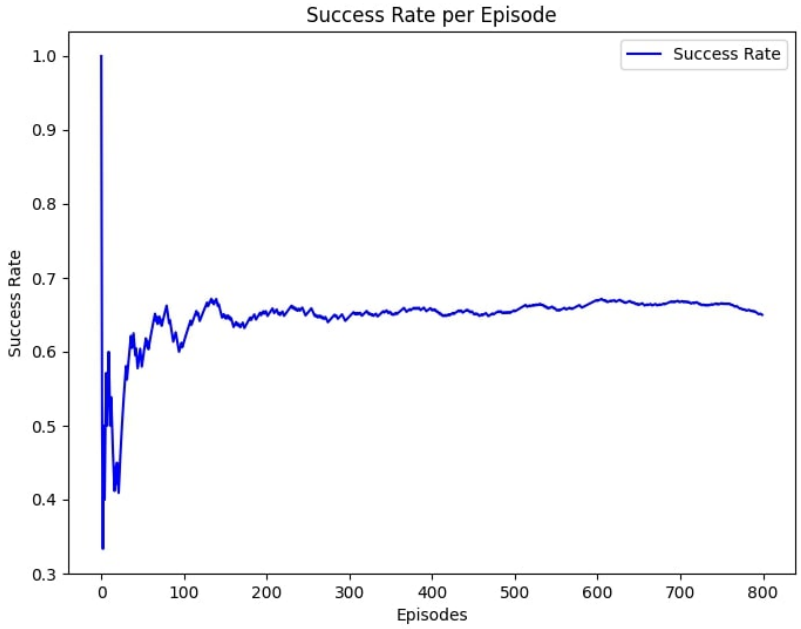

Figura 4.9: Andamento del success rate di Helper-LLM e Reviewer-LLM in 800 episodi.

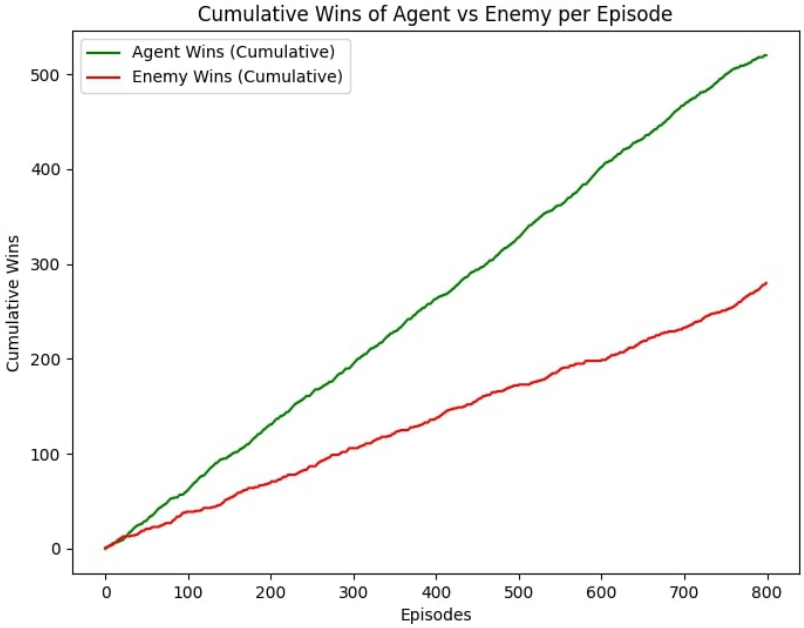

Figura 4.10: Confronto delle vittorie cumulative di Helper-LLM Reviewer-LLM e nemico in 800 episodi.

Anche la media delle ricompense ha raggiunto il valore più alto, pari a 262,5 (Fig. 4.11). La media delle mosse, come precedentemente osservato, si attesta a 11 (Fig. 4.12), mentre lo score delle mosse rimane stabile a 0,76.

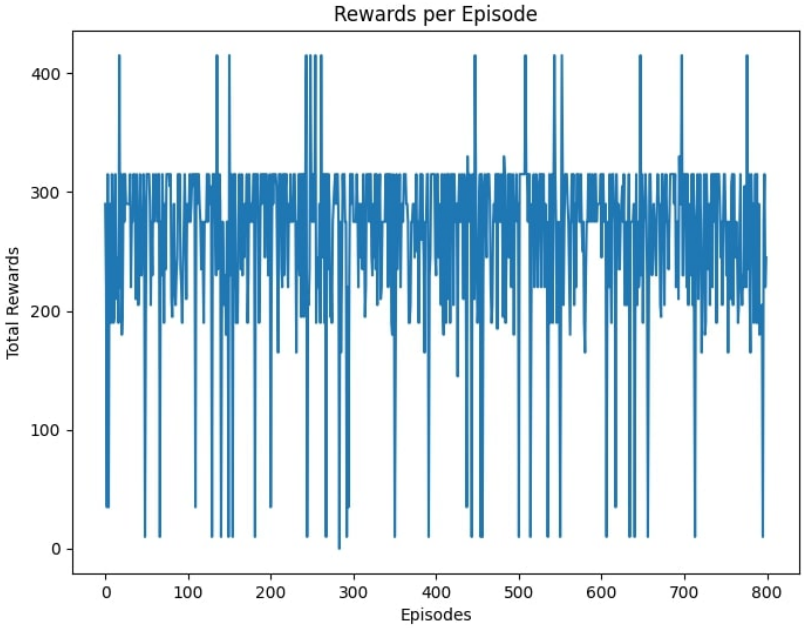

Figura 4.11: Andamento dei reward di Helper-LLM e Reviewer-LLM in 800 episodi.

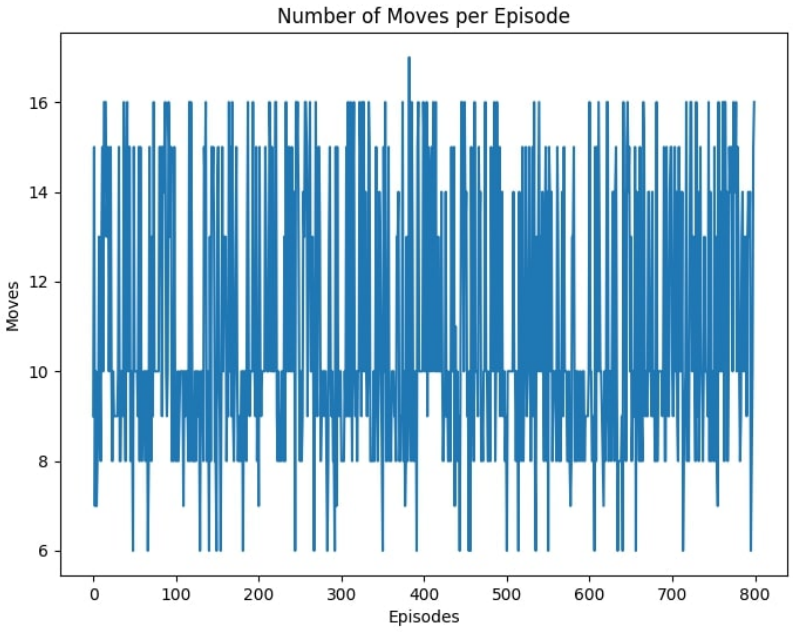

Figura 4.12: Andamento delle mosse effettuate da Helper-LLM e Reviewer-LLM in 800 episodi.

Di seguito è riportata la tabella riassuntiva (Tab. 4.5) dei risultati ottenuti:

Episodi	Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse	Score
800	520	280	65%	262,5	11	0,76

Tabella 4.5: Risultati ottenuti da Helper-LLM e le istruzioni di Reviewer-LLM in 800 episodi.

Con 1000 episodi si è registrato il miglior risultato in termini di success rate, con una percentuale di vittorie pari al 65,9%, corrispondente a 659 partite vinte (Fig. 4.13, 4.14). Sono stati osservati 3 casi di allucinazioni, gestiti come descritto in precedenza.

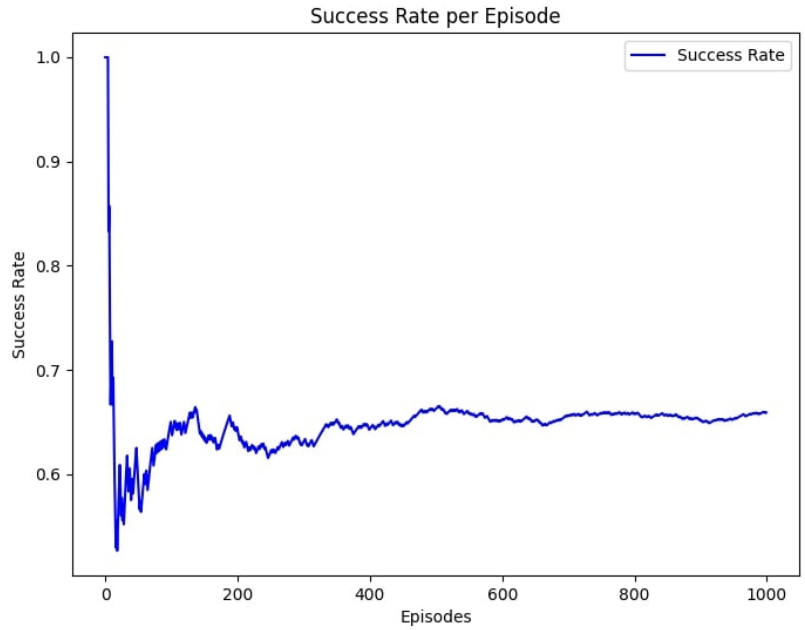

Figura 4.13: Andamento del success rate di Helper-LLM e Reviewer-LLM in 1000 episodi.

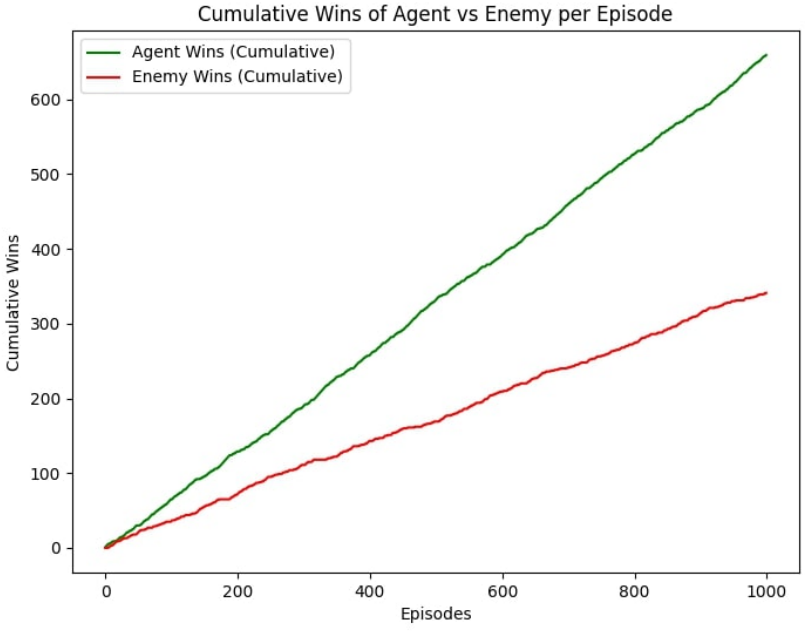

Figura 4.14: Confronto delle vittorie cumulative di Helper-LLM Reviewer-LLM e nemico in 1000 episodi.

La media delle ricompense è stata di 260 (Fig. 4.15), con una media di 11 mosse per partita (Fig. 4.16).

Lo score delle mosse ha raggiunto il valore più alto in assoluto, pari a 0,77.

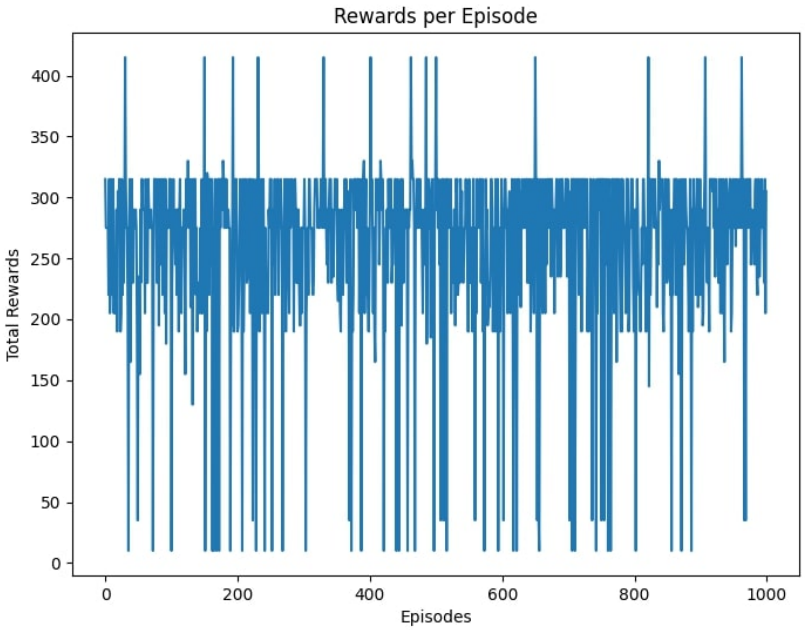

Figura 4.15: Andamento dei reward di Helper-LLM e Reviewer-LLM in 1000 episodi.

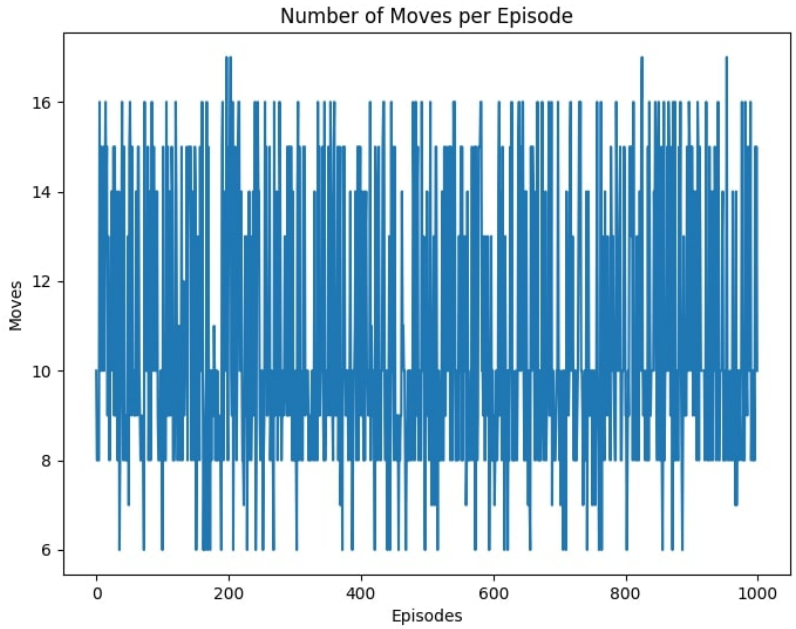

Figura 4.16: Andamento delle mosse effettuate da Helper-LLM e Reviewer-LLM in 1000 episodi.

Di seguito è riportata la tabella riassuntiva (Tab. 4.6) dei risultati ottenuti:

Episodi	Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse	Score
1000	659	341	65,9%	260	11	0,77

Tabella 4.6: Risultati ottenuti da Helper-LLM e Reviewer-LLM in 1000 episodi.

Un ulteriore test è stato condotto per analizzare l'impatto dei suggerimenti nel corso delle partite. L'esperimento è composto da 500 episodi suddivisi in due fasi: nei primi 150 episodi User è stato gestito da un approccio basato su RL (4.2.1), mentre nei successivi 350 sono stati utilizzati i consigli di Helper-LLM e Reviewer-LLM. I risultati evidenziano che, nelle prime 150 partite, l'agente ha faticato a individuare una strategia efficace, registrando una sola vittoria e un success rate estremamente basso, pari allo 0,67% (Fig. 4.17). Al contrario, durante le 350 partite successive, i suggerimenti forniti hanno portato a un notevole miglioramento delle prestazioni, con un totale di 226 partite vinte (Fig. 4.18) e un success rate pari al 64,57%. Complessivamente, sulle 500 partite, il success rate è stato del 45,4%.

Figura 4.17: Andamento del success rate su 500 episodi in cui i primi 150 giocati da RL e i successivi 350 da Helper-LLM e Reviewer-LLM.

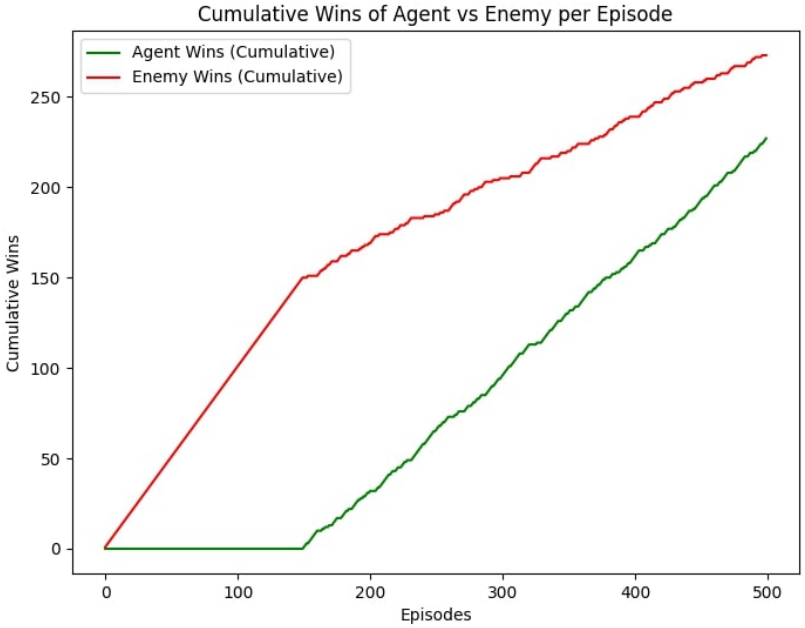

Figura 4.18: Andamento delle vittorie cumulative su 500 episodi in cui i primi 150 giocati da RL e i successivi 350 da Helper-LLM e Reviewer-LLM.

Anche la media delle ricompense (Fig. 4.19) ha mostrato un significativo incremento: da 58,3 nella fase iniziale si è passati a una media di 260. Questo aumento sottolinea l'efficacia dei consigli forniti dall'NPC nel migliorare la strategia di gioco. Inizialmente è stata registrata una media delle mosse effettuate (Fig. 4.20) pari a 11,3. Durante la fase successiva, la media è risultata leggermente inferiore, ovvero 10,8 mosse. In quest'ultimo test è stata registrata una sola allucinazione.

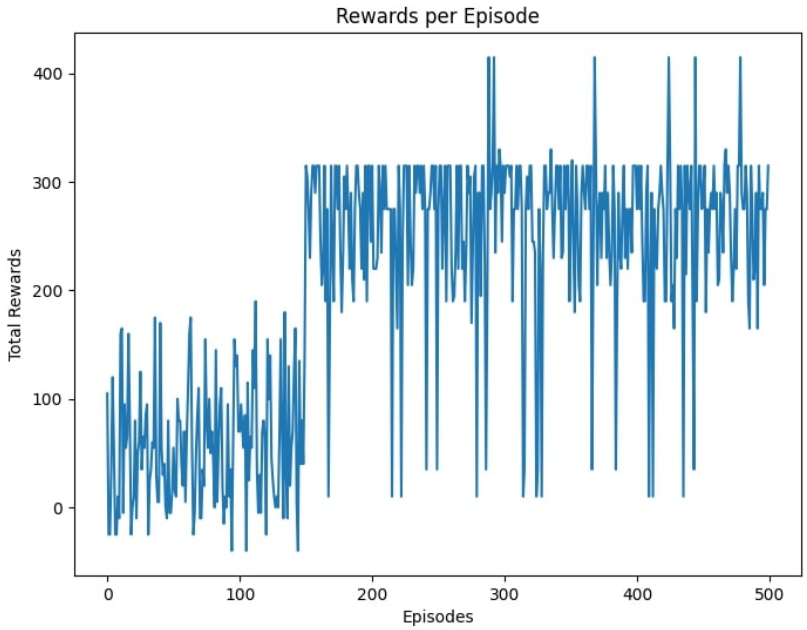

Figura 4.19: Andamento del reward su 500 episodi in cui i primi 150 giocati da RL e i successivi 350 da Helper-LLM e Reviewer-LLM.

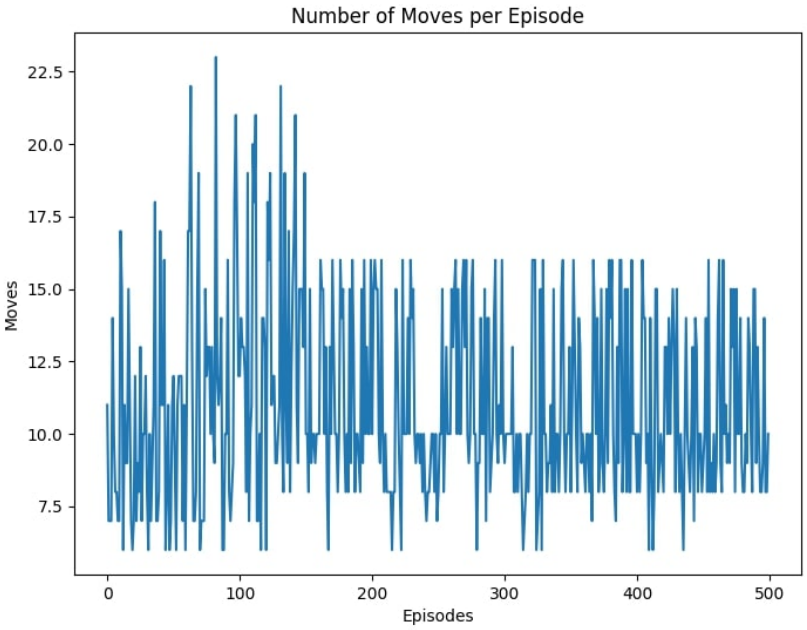

Figura 4.20: Andamento delle mosse effettuate su 500 episodi in cui i primi 150 giocati da RL e i successivi 350 da Helper-LLM e Reviewer-LLM.

Di seguito è riportata una tabella riassuntiva (Tab. 4.7) del confronto dei risultati ottenuti:

Episodi	Vittorie	% vittorie	Media ricompense	Media mosse effettuate
150	1	0,67%	58,3	11,3
350	226	64,57%	260	10,8

Tabella 4.7: Confronto dei risultati ottenuti da RL nei primi 150 episodi e nei successivi 350 da Helper-LLM e Reviewer-LLM.

4.2 Analisi comparativa

Per valutare ulteriormente l'efficacia del sistema proposto, è stata condotta un'analisi comparativa in cui è stato analizzato il comportamento di un approccio basato su RL, testato anch'esso su 200, 500, 800 e 1000 episodi. Successivamente, le stesse configurazioni sono state replicate testando i consigli iniziali forniti da Helper-LLM.

4.2.1 Confronto con un approccio basato sul Reinforcement Learning

L'approccio di RL scelto è l'algoritmo Deep Q Network [8], che ha dimostrato di poter raggiungere buone performance di livello umano su sette videogiochi della console Atari 2600 [40]. In particolare, tale algoritmo rappresenta un'evoluzione del metodo Q-Learning, in cui la tabella stato-azione viene sostituita da una rete neurale. L'apprendimento non si basa sull'aggiornamento di una tabella, ma piuttosto sull'adattamento dei pesi dei neuroni all'interno della rete attraverso il processo di backpropagation. L'apprendimento della value function nel DQN si basa sulla modifica dei pesi in funzione della seguente loss function:

$$L_t = \Big(E[r + \gamma \max_a Q(s_{t+1}, a)] - Q(s_t, a_t)\Big)^2 \tag{4.1}$$

Dove $E[r + \gamma \max_a Q(s_{t+1}, a)]$ rappresenta l'expected return, mentre $Q(s_t, a_t)$ rappresenta il valore stimato dalla rete. Gli errori derivanti da questa loss function vengono propagati all'indietro nella rete utilizzando un passo di backword, seguendo il metodo della discesa del gradiente. Il comportamento della policy viene implementato tramite un approccio e-greedy, il quale garantisce un'adeguata esplorazione dello spazio delle azioni. L'elemento cruciale che contribuisce all'efficacia del DQN è l'uso dell'experience replay. Questa tecnica consente di salvare le esperienze dell'agente, rappresentate come tuple $e_t = (s_t, a_t, r_t, s_{t+1})$, in un dataset

chiamato replay memory, definito come:

$$D = e_t, e_{t+1}, \ldots, e_n \tag{4.2}$$

Durante il training, vengono utilizzati mini-batch di esperienze selezionate casualmente dal replay memory, consentendo di riutilizzare esperienze passate per più di un aggiornamento della rete. Questo approccio aiuta ad interrompere la forte correlazione tra esperienze successive, riducendo la varianza negli aggiornamenti e rendendo l'apprendimento più stabile ed efficace. In questo caso, lo stato è definito dallo stato attuale del gioco, che include gli HP ed MP del giocatore, nonché le azioni disponibili. I reward associati alle azioni sono descritti nella tabella 3.2. Come si può osservare dai grafici 4.21 e 4.22, l'agente incontra notevoli difficoltà nei primi 200 episodi, riuscendo a vincere solo 4 partite, con un success rate pari al 2%.

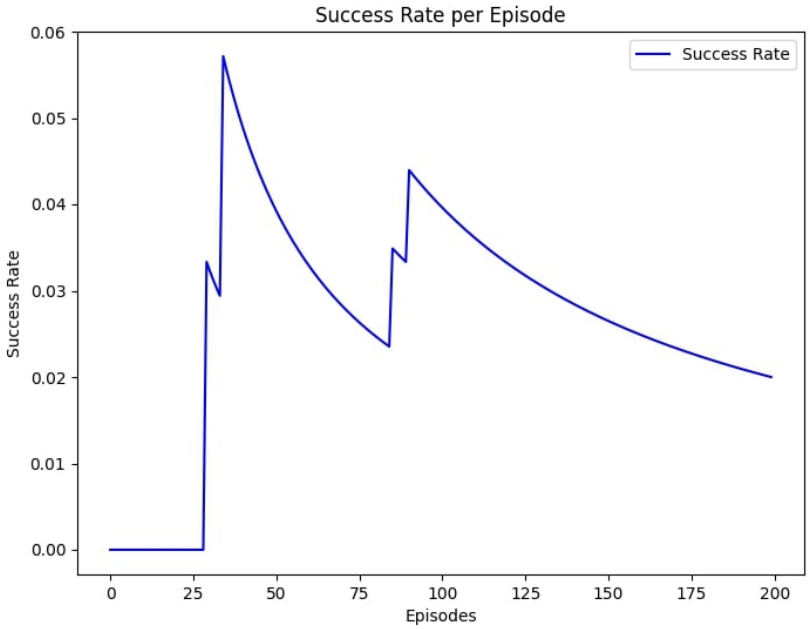

Figura 4.21: Andamento del success rate del DQN in 200 episodi.

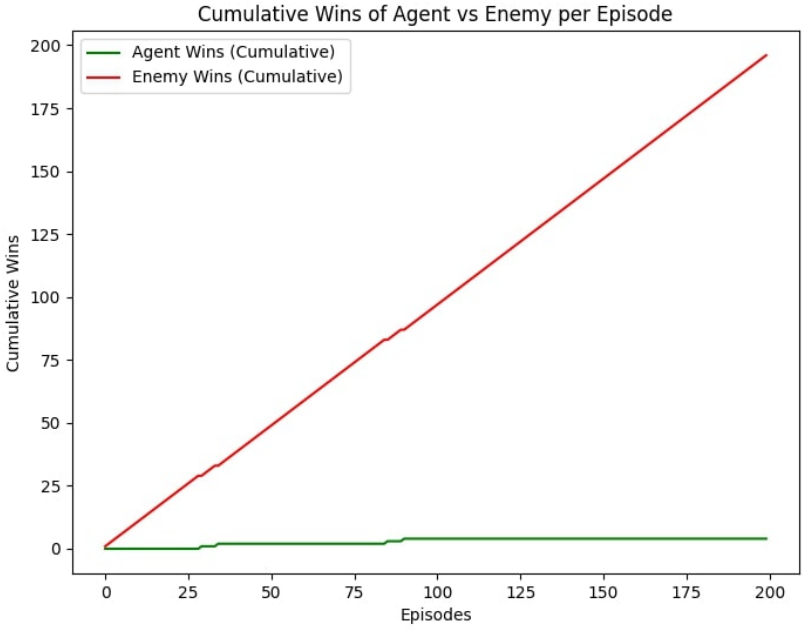

Figura 4.22: Confronto delle vittorie cumulative del DQN e nemico in 500 episodi.

I grafici 4.23 e 4.24 evidenziano un'oscillazione sia nelle ricompense che nel numero di mosse per partita, attribuibile alla natura randomica delle partite. L'agente ha ottenuto una ricompensa media di 74, con una media di 12 mosse per partita. Tali risultati sono significativamente inferiori rispetto a quelli ottenuti utilizzando il sistema proposto, evidenziando fin da subito i limiti dell'agente di RL.

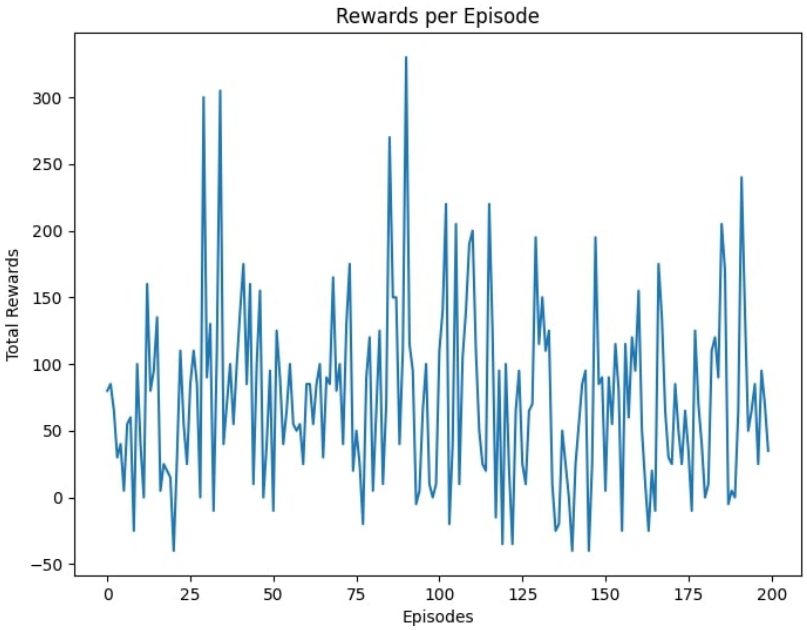

Figura 4.23: Andamento dei reward del DQN in 200 episodi.

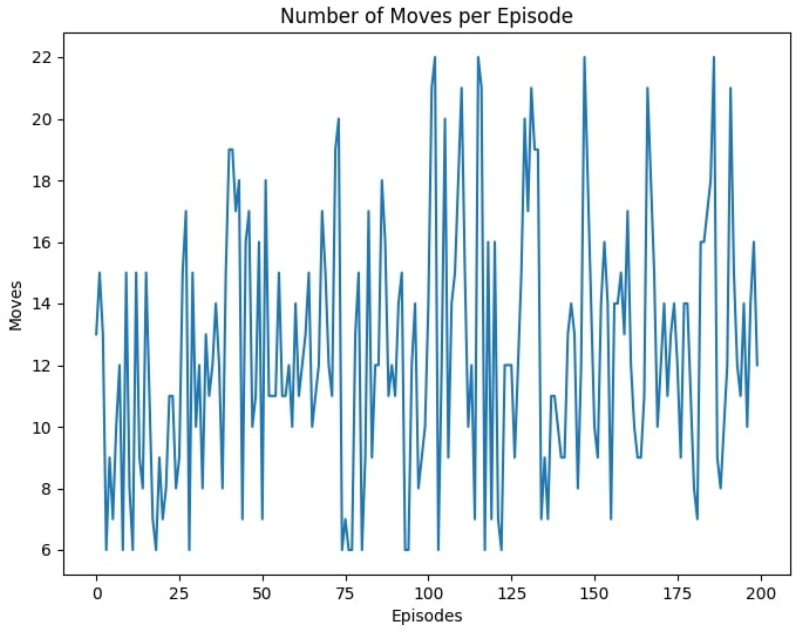

Figura 4.24: Andamento delle mosse effettuate dal DQN in 200 episodi.

Di seguito è riportata una tabella riassuntiva (Tab. 4.8) dei risultati ottenuti.

Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse effettuate
4	196	2%	74	12

Tabella 4.8: Risultati ottenuti dal DQN in 200 episodi.

Il test condotto su 500 episodi, come evidenziato nei grafici 4.25 e 4.26, conferma le difficoltà descritte in precedenza. Sebbene l'agente mostri un lieve miglioramento nel success rate nelle fasi iniziali, quest'ultimo tende a diminuire con il progredire delle partite, portando solo a un marginale incremento rispetto ai risultati ottenuti su 200 episodi. In particolare, l'agente riesce a vincere 13 partite su 500, con un success rate pari al 2,6%. Questo risultato rimane nettamente inferiore rispetto a quello raggiunto dal sistema proposto, sottolineando ancora una volta l'importanza del supporto fornito dai consigli.

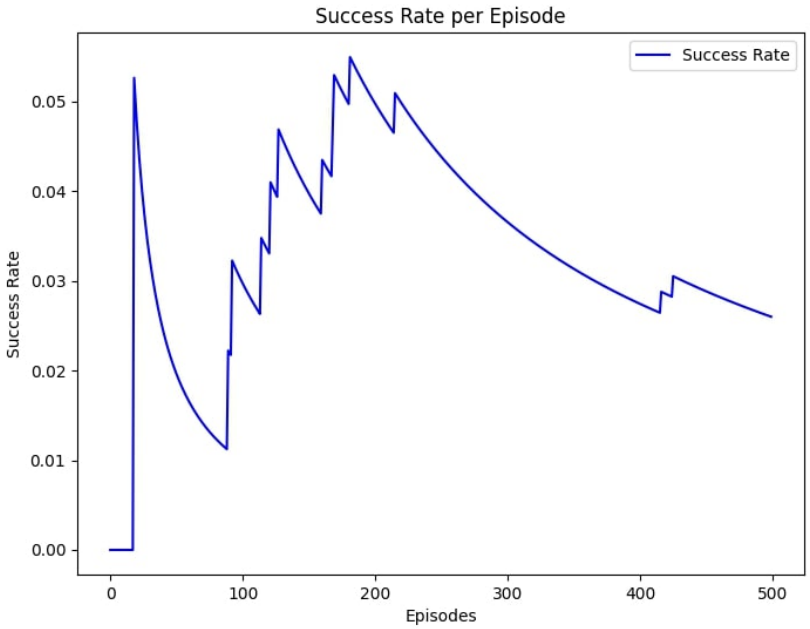

Figura 4.25: Andamento del success rate del DQN in 500 episodi.

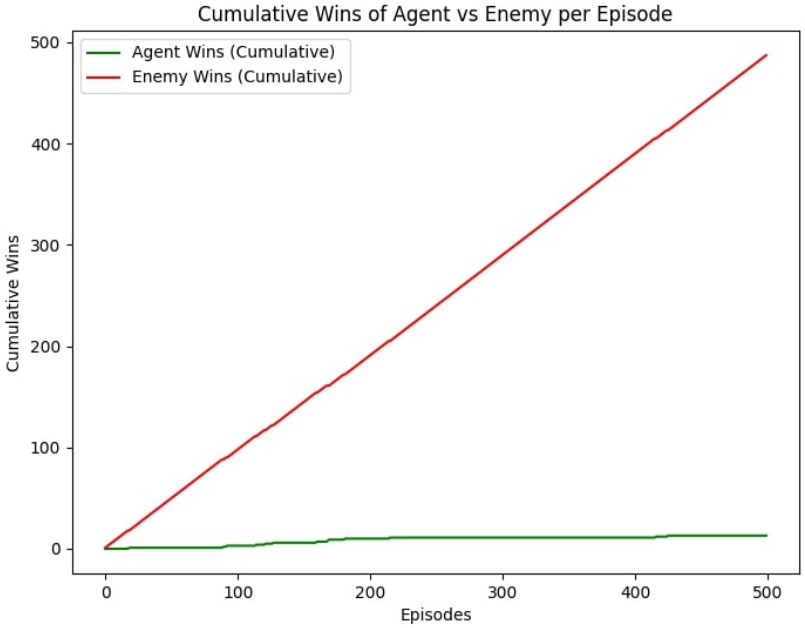

Figura 4.26: Confronto delle vittorie cumulative del DQN e nemico in 500 episodi.

In questo caso i grafici 4.27 e 4.28 mostrano dei picchi più elevati rispetto ai 200 episodi, ma la ricompensa media risulta essere più bassa, con un punteggio pari a 61 ed una media di 13 mosse per partita.

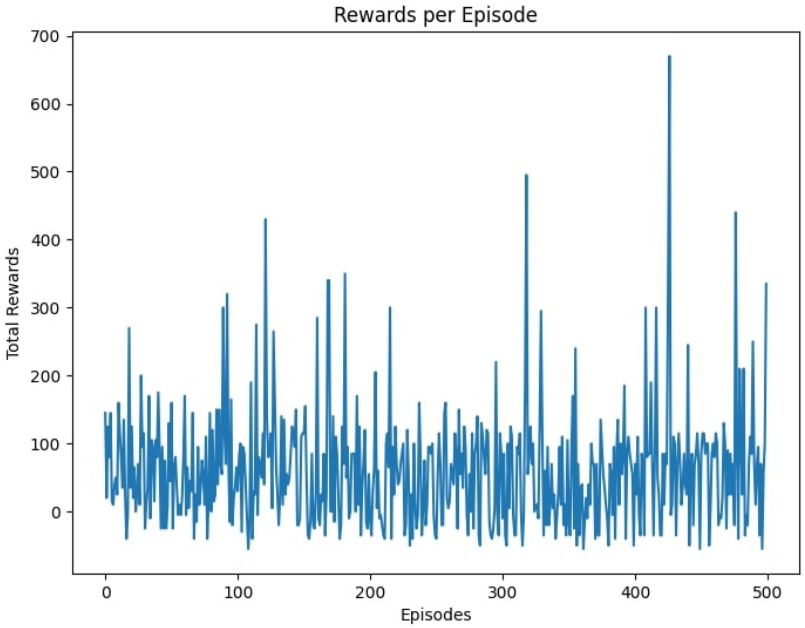

Figura 4.27: Andamento dei reward del DQN in 500 episodi.

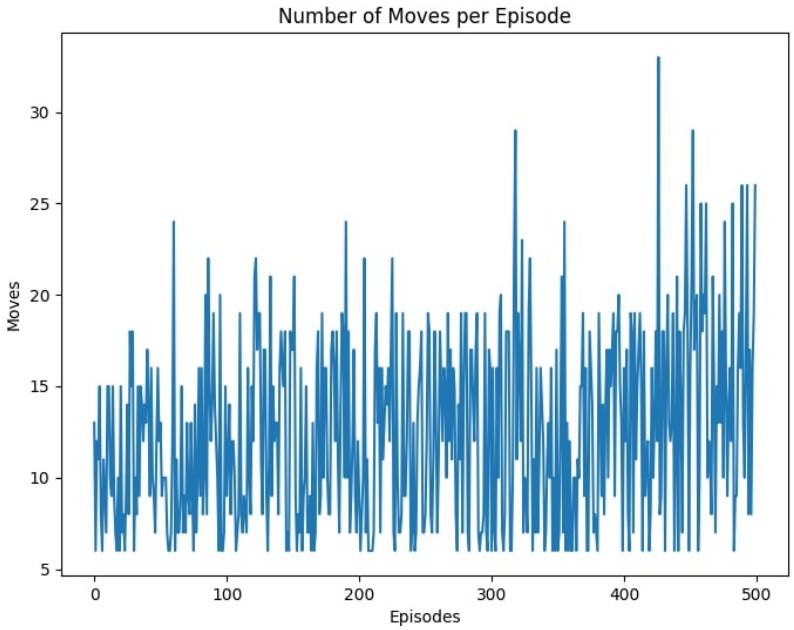

Figura 4.28: Andamento delle mosse effettuate dal DQN in 500 episodi.

Di seguito è riportata una tabella riassuntiva (Tab. 4.9) dei risultati ottenuti.

Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse effettuate
13	487	2,6%	61	13

Tabella 4.9: Risultati ottenuti dal DQN in 500 episodi.

Come si può osservare dai grafici 4.29 e 4.30, nel test con 800 episodi l'agente mostra inizialmente uno dei picchi di success rate più alti rispetto ai risultati ottenuti nei 200 e 500 episodi; tuttavia, con il progredire degli episodi, il success rate diminuisce, risultando infine inferiore rispetto a quello registrato con 500 episodi. In particolare, sono state vinte 19 partite su 800, con una percentuale di vittorie pari al 2,38%.

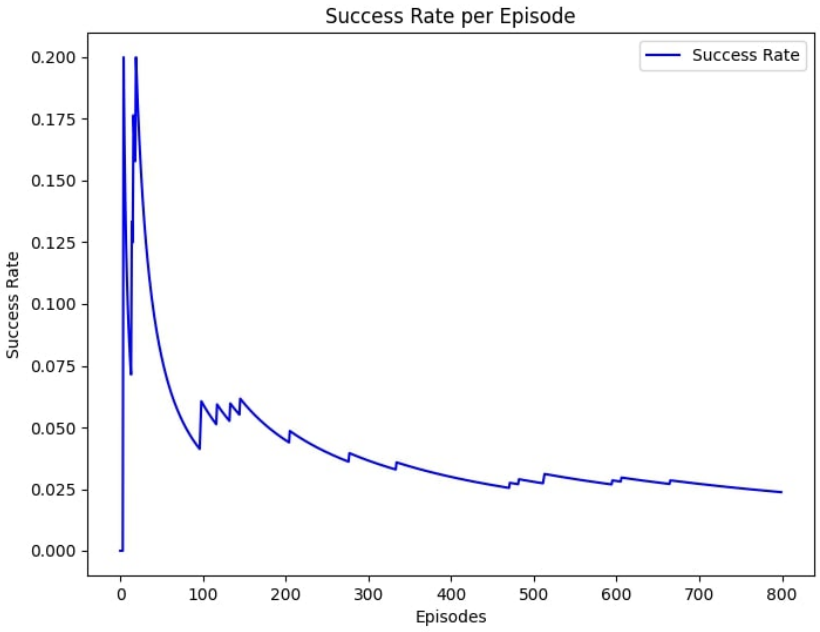

Figura 4.29: Andamento del success rate del DQN in 800 episodi.

I grafici 4.31 e 4.32 evidenziano un incremento sia delle ricompense ottenute sia delle mosse effettuate, indicando che, in questo caso, l'agente è riuscito a sostenere meglio il confronto con il nemico. Tuttavia, come evidenziato in precedenza, la strategia adottata non si è rivelata efficace, manifestando, con il progredire degli episodi, difficoltà crescenti rispetto al sistema proposto. In particolare, l'agente ha ottenuto una ricompensa media di 150, con una media di 16 mosse per partita.

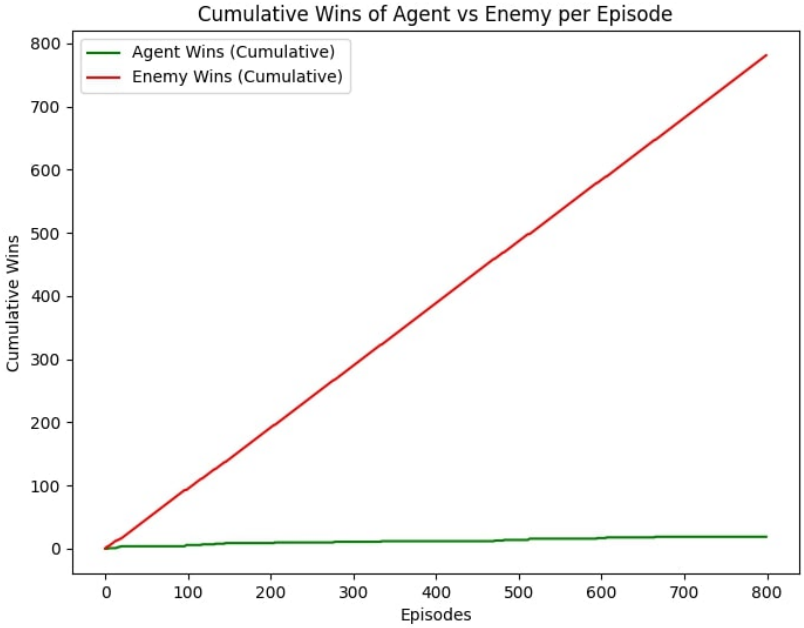

Figura 4.30: Confronto delle vittorie cumulative del DQN e nemico in 800 episodi.

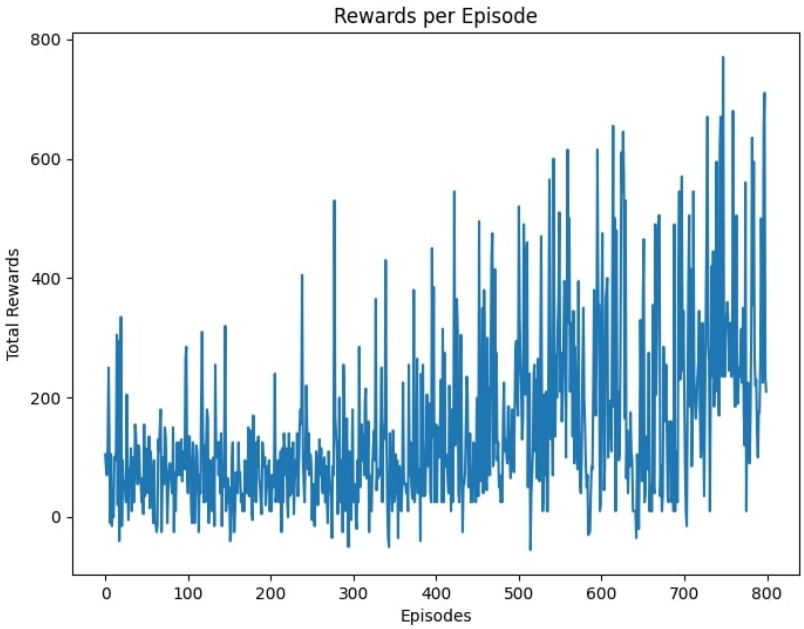

Figura 4.31: Andamento dei reward del DQN in 800 episodi.

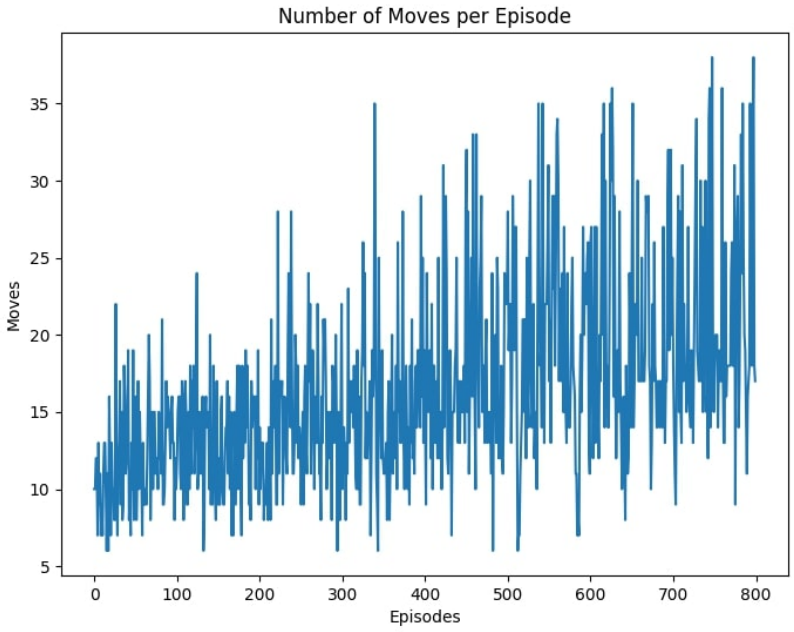

Figura 4.32: Andamento delle mosse effettuate dal DQN in 800 episodi.

Di seguito è riportata una tabella riassuntiva (Tab. 4.10) dei risultati ottenuti.

Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse effettuate
19	781	2,38%	150	16

Tabella 4.10: Risultati ottenuti dal DQN in 800 episodi.

Con il test di 1000 episodi, come evidenziato nei grafici 4.33 e 4.34, l'agente ha registrato il peggior risultato in termini di success rate. In particolare, sono state vinte 10 partite su 1000, con una percentuale di vittorie pari all'1%.

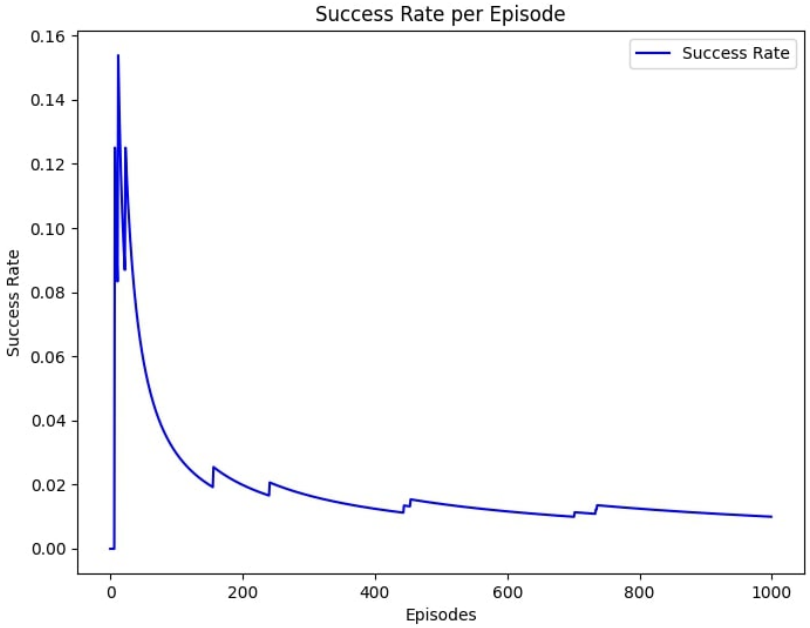

Figura 4.33: Andamento del success rate del DQN in 1000 episodi.

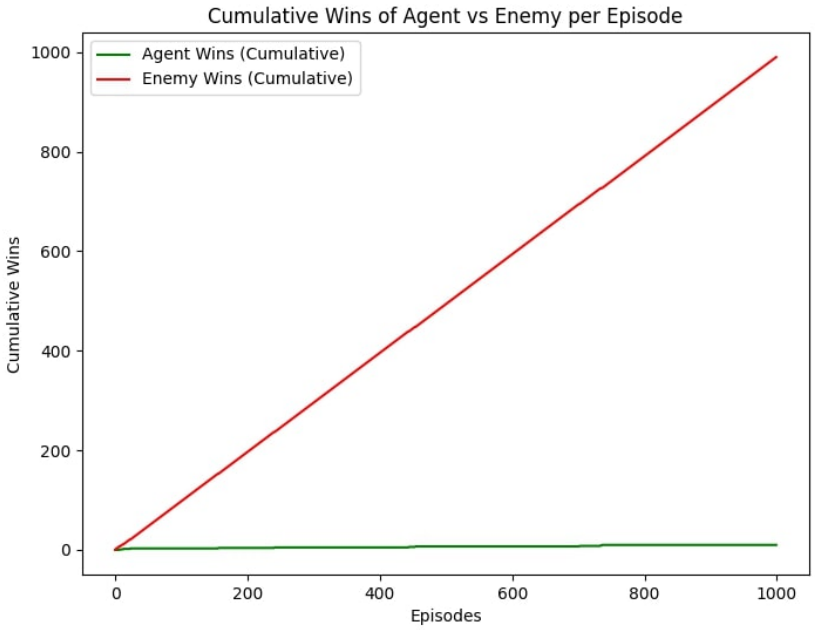

Figura 4.34: Confronto delle vittorie cumulative del DQN e nemico in 1000 episodi.

I grafici 4.35, 4.36 mostrano inoltre che l'agente ha ottenuto una ricompensa media di 124, un decremento dettato dalla media delle mosse per partita, che in questo caso è calata a 15.

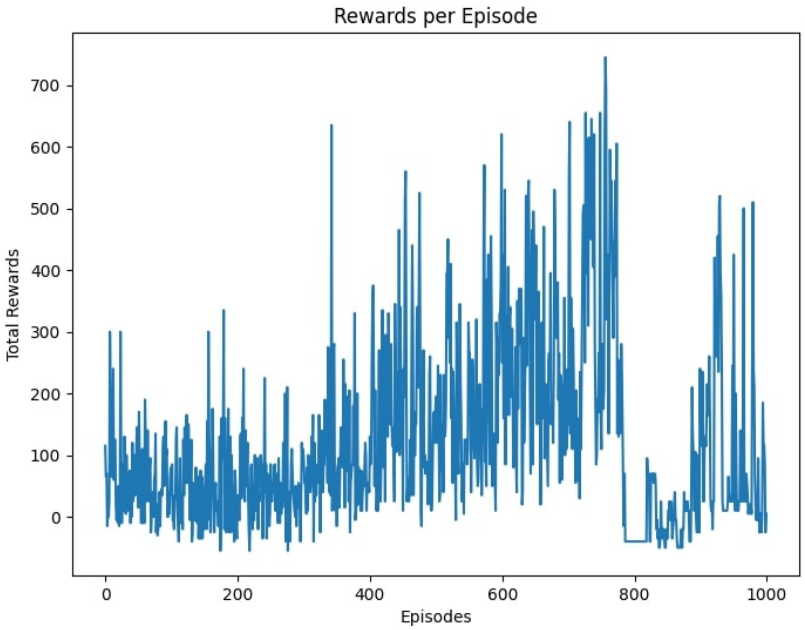

Figura 4.35: Andamento dei reward del DQN in 1000 episodi.

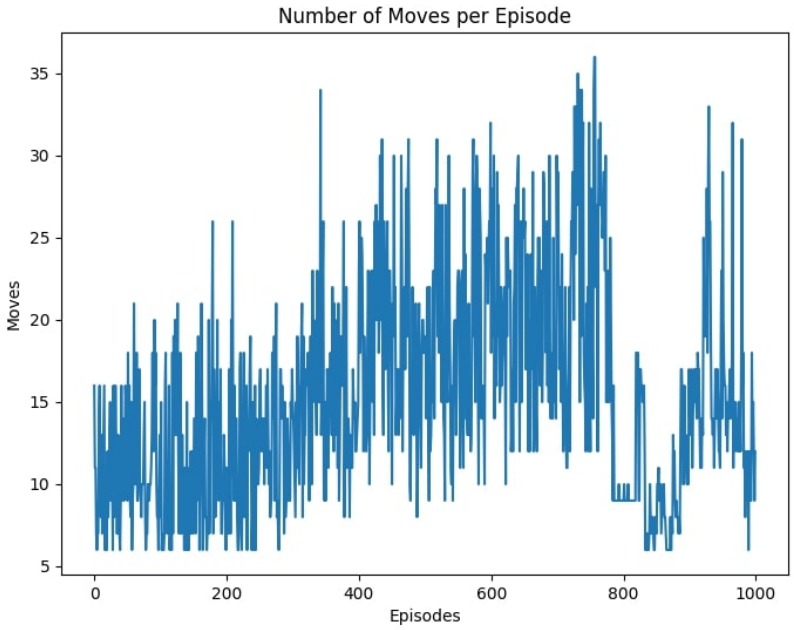

Figura 4.36: Andamento delle mosse effettuate dal DQN in 1000 episodi.

Di seguito è riportata una tabella riassuntiva (Tab. 4.11) dei risultati ottenuti.

Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse effettuate
10	990	1%	124	15

Tabella 4.11: Risultati ottenuti dal DQN in 1000 episodi.

Valutazione dei risultati. I risultati ottenuti dal DQN evidenziano una difficoltà significativa nel raggiungere performance accettabili. Come discusso nel capitolo 3, uno dei principali ostacoli nel Reinforcement Learning è la definizione di una reward function ottimale. In questo caso, considerata la complessità del gioco, è difficile assegnare reward negative all'agente a causa delle numerose strategie possibili. Con il progredire degli episodi, si è osservato un aumento sia nel numero di mosse che nella ricompensa ottenuta. Tuttavia, questo miglioramento non indica un apprendimento efficace, in quanto la strategia adottata dall'agente non converge mai verso una soluzione ottimale. Al contrario, il sistema proposto mostra risultati nettamente superiori, ottenendo strategie più avanzate e coerenti che, con l'aumentare degli episodi, facilitano un progresso continuo e più stabile verso l'ottimizzazione. Di seguito è riportata una tabella riassuntiva (Tab. 4.12) dei risultati ottenuti dal DQN in tutti i test effettuati:

Episodi	Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse effettuate
200	4	196	2%	74	12
500	13	487	2,6%	61	13
800	19	781	2,38%	150	16
1000	10	990	1%	124	15

Tabella 4.12: Risultati ottenuti dal DQN.

4.2.2 Confronto con Helper-LLM

I consigli forniti da Helper-LLM, senza il supporto delle istruzioni di Reviewer-LLM, hanno prodotto risultati significativamente inferiori rispetto sia al sistema proposto che al DQN, con un success rate invariato su tutti gli episodi e nessuna vittoria registrata (Fig. 4.37, 4.38). La media delle ricompense si colloca tra 73 e 78, mentre la media delle mosse effettuate si attesta costantemente su 7 per partita (Fig. 4.40). Inoltre, lo score medio delle azioni

consigliate oscilla tra 0,35 e 0,40, indicando una limitata efficacia strategica dei suggerimenti forniti esclusivamente da Helper-LLM.

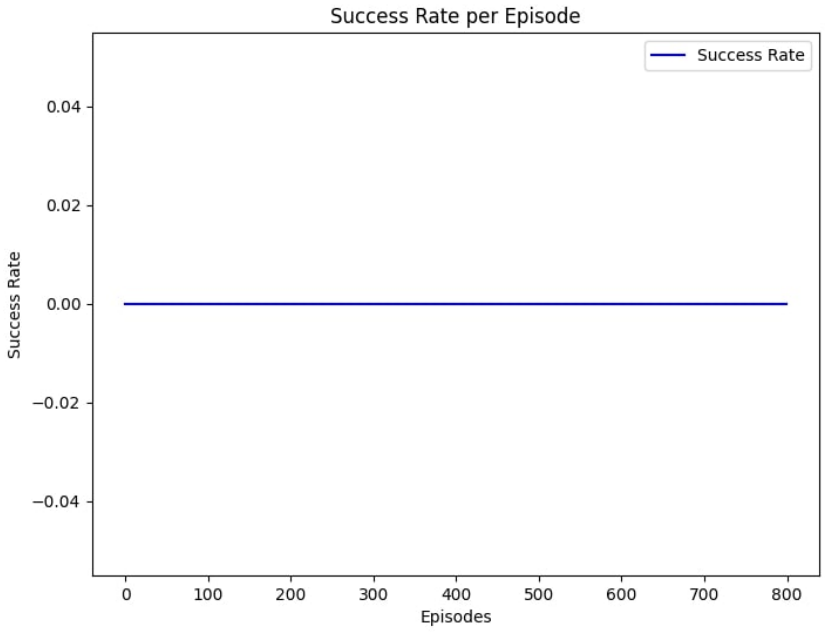

Figura 4.37: Andamento del success rate di Helper-LLM in 800 episodi.

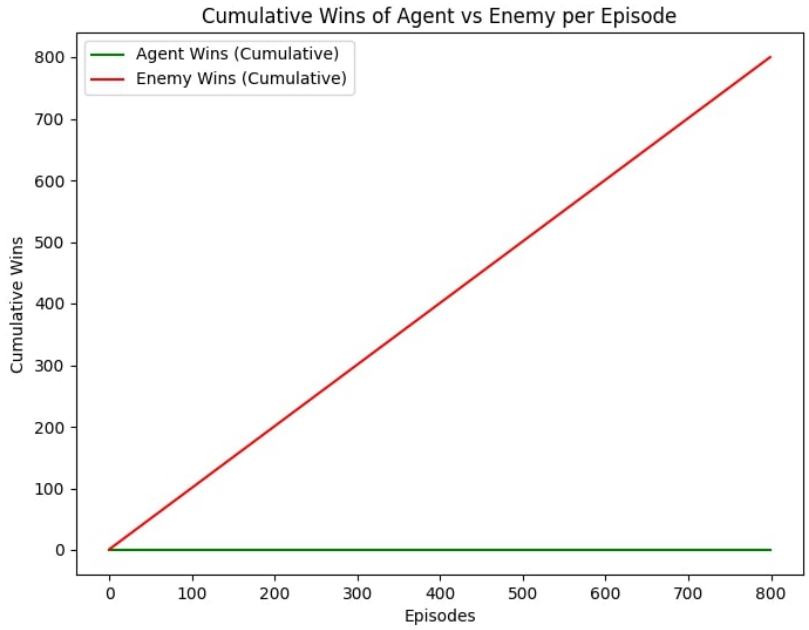

Figura 4.38: Confronto delle vittorie cumulative di Helper-LLM e nemico in 800 episodi.

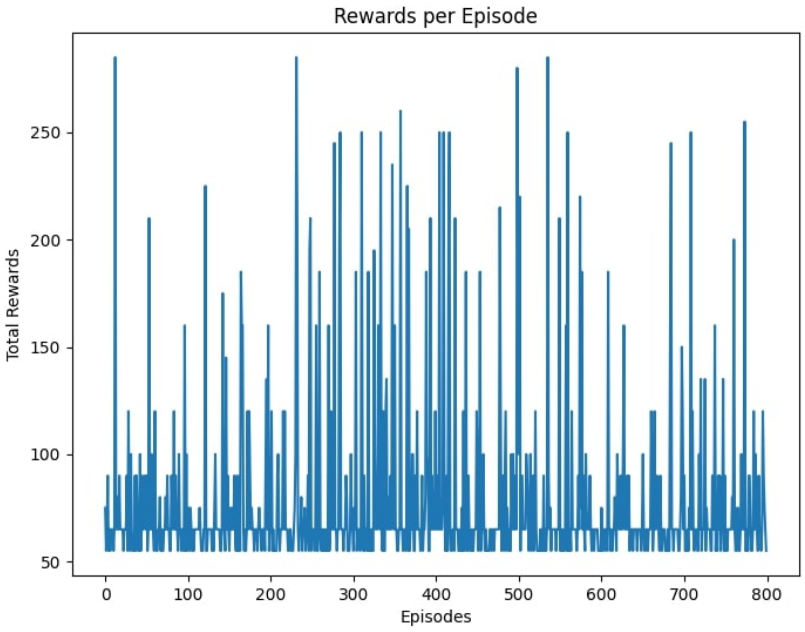

Figura 4.39: Andamento dei reward di Helper-LLM in 800 episodi.

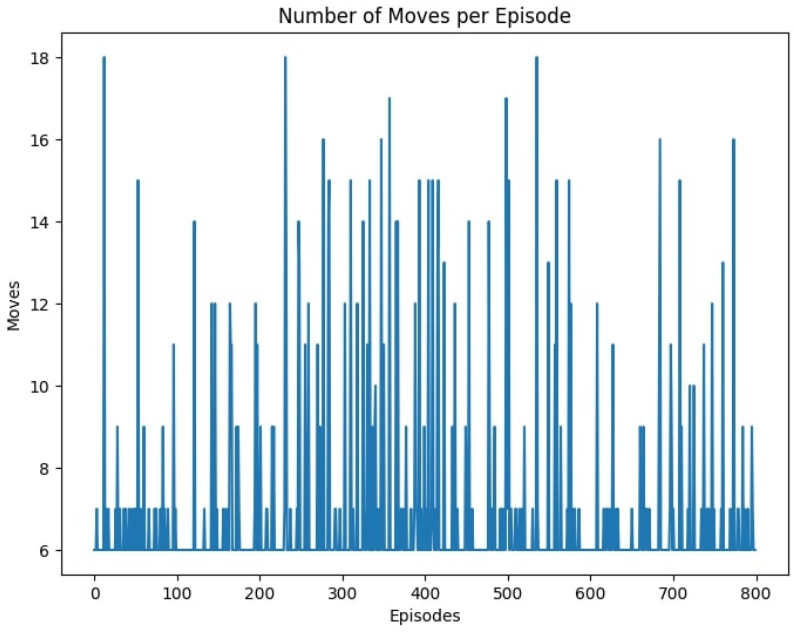

Figura 4.40: Andamento delle mosse effettuate da Helper-LLM in 800 episodi.

Di seguito è riportata una tabella riassuntiva dei risultati ottenuti:

Episodi	Vittorie	Sconfitte	% vittorie	Media ricompense	Media mosse	Score
200	0	200	0%	73	7	0,4
500	0	500	0%	78	7	0,38
800	0	800	0%	78,4	7	0,4
1000	0	1000	0%	76	7	0,35

Tabella 4.13: Risultati ottenuti da Helper-LLM.

4.3 Analisi dei risultati

Le istruzioni fornite dal Reviewer-LLM hanno contribuito a migliorare significativamente le risposte di Helper-LLM, portando a consigli che hanno ottenuto risultati nettamente superiori rispetto alle azioni consigliate inizialmente e ai risultati dell'approccio basato su DQN.

Episodi	Helper	Helper Reviewer
200	0,4	0,75
500	0,38	0,76
800	0,4	0,76
1000	0,35	0,77

Tabella 4.14: Confronto score mosse.

L'incremento dello score registrato nei diversi episodi (Tab. 4.14) riflette un corrispondente aumento sia nel success rate (Tab. 4.18, Fig. 4.41, 4.42, 4.43, 4.44) che nella media delle ricompense (Tab. 4.15), dimostrando l'efficacia delle revisioni apportate. Questi risultati sono accompagnati da un notevole aumento nel numero di partite vinte (Tab. 4.17), che cresce con il progredire degli episodi, confermando l'impatto positivo delle istruzioni fornite dal sistema. Inoltre, la media delle mosse effettuate (Tab. 4.16) evidenzia come il sistema proposto riesca ad ottenere una gestione più stabile ed efficiente del numero di azioni necessarie per completare ogni partita. Questo aspetto dimostra non solo una maggiore consistenza strategica, ma anche un'efficace ottimizzazione delle risorse impiegate.

Episodi	DQN	Helper	Helper Reviewer
200	74	73	259
500	61	78	258
800	150	78,4	262,5
1000	124	76	260

Tabella 4.15: Confronto media delle ricompense.

Episodi	DQN	Helper	Helper Reviewer
200	12	7	11
500	13	7	11
800	16	7	11
1000	15	7	11

Tabella 4.16: Confronto media delle mosse effettuate.

Episodi	DQN	Helper	Helper Reviewer
200	4	0	129
500	13	0	319
800	19	0	520
1000	10	0	659

Tabella 4.17: Confronto delle vittorie ottenute.

Episodi	DQN	Helper	Helper Reviewer
200	2	0	64,5
500	2,6	0	63,8
800	2,38	0	65
1000	1	0	65,9

Tabella 4.18: Confronto success rate.

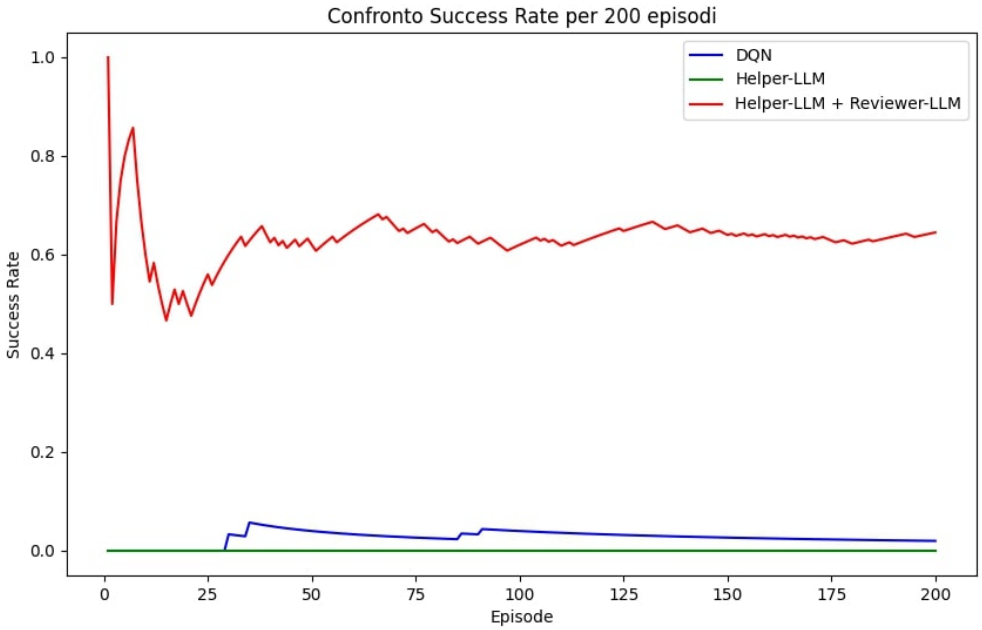

Figura 4.41: Confronto del success rate in 200 episodi.

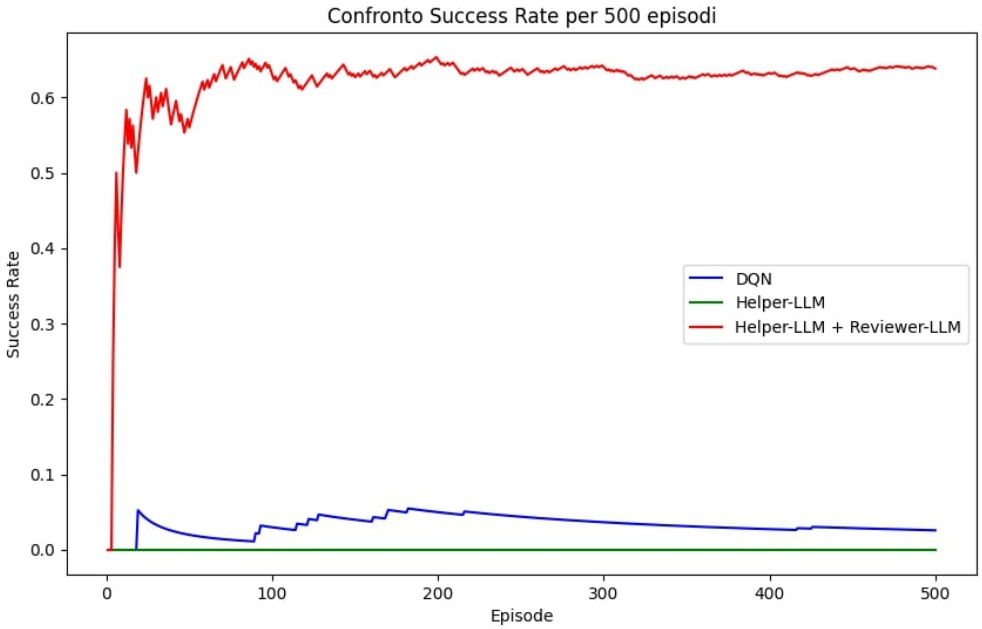

Figura 4.42: Confronto del success rate in 500 episodi.

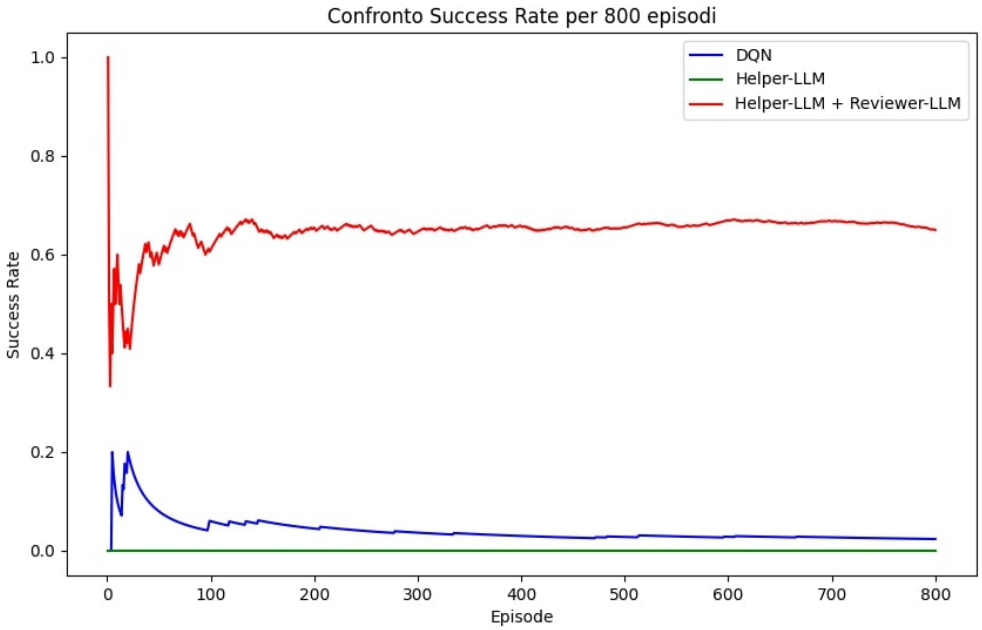

Figura 4.43: Confronto del success rate in 800 episodi.

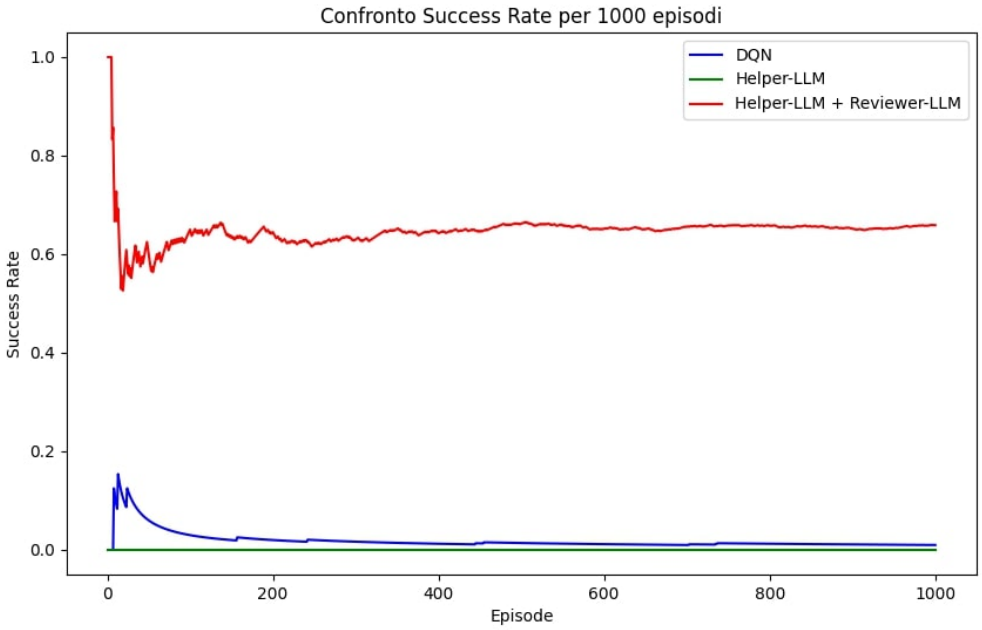

Figura 4.44: Confronto del success rate in 1000 episodi.

I risultati ottenuti mostrano le difficoltà iniziali, da parte di Helper-LLM, nel formulare suggerimenti ottimali, spesso causato da un reasoning incompleto o risposte basate su informazioni non pienamente comprese. L'uso del Reinforcement Learning ha contribuito in

modo significativo alla creazione di un Reviewer, che ha permesso di colmare queste lacune attraverso un processo di valutazione iterativa, analizzando la coerenza e la completezza del reasoning di Helper-LLM e fornendo istruzioni su come migliorarlo. La necessità di un Reviewer deriva anche dalla complessità del contesto di gioco, che richiede decisioni basate su un'analisi approfondita di diverse variabili quali HP, MP, azioni disponibili e strategia del nemico. Senza queste istruzioni, Helper-LLM tende a fornire azioni che possono non adattarsi all'evoluzione dello stato del gioco. Il Reviewer, invece, aiuta Helper-LLM ad ottenere scelte allineate con l'evoluzione del gioco, portando così ad un incremento del success rate. Questo processo iterativo permette di rispondere positivamente alla research question iniziale.

Capitolo 5

Conclusioni e sviluppi futuri

L'utilizzo dei LLM nel contesto dei videogiochi rappresenta un ambito in rapida evoluzione, che promette di rendere le interazioni con i NPC sempre più naturali e coinvolgenti. Questi modelli, grazie alla loro capacità di comprendere e generare linguaggio umano, possono trasformare il modo in cui i giocatori interagiscono con i mondi virtuali. Tuttavia, l'applicazione pratica dei LLM nei videogiochi è ancora in fase di esplorazione, specialmente per quanto riguarda la loro integrazione con altre tecnologie come il Reinforcement Learning. Questa tesi si è concentrata sull'implementazione di un sistema che utilizza LLM ed RL per migliorare le capacità strategiche di un NPC all'interno di un environment di gioco a turni. L'obiettivo primario era quello di fornire suggerimenti efficaci per sconfiggere un avversario, migliorando il reasoning e massimizzando l'efficacia delle azioni consigliate. Il lavoro ha esplorato in dettaglio come Large Language Models, nello specifico FlanT5-Large, possano essere utilizzati per generare risposte strategiche e coerenti. Per garantire una performance ottimale, il sistema è stato arricchito con un Reviewer, addestrato a fornire istruzioni utili per affinare il reasoning delle risposte suggerite. La combinazione di modelli di LLM con tecniche di RL, in particolare il Proximal Policy Optimization, ha permesso di ottimizzare il processo decisionale, riducendo le allucinazioni iniziali e aumentando la coerenza e la qualità delle risposte generate. I risultati ottenuti dalla sperimentazione condotta hanno evidenziato un miglioramento significativo sia in termini di success rate che di impatto strategico. L'NPC, supportato dal sistema proposto, ha mostrato una maggiore capacità di adottare strategie efficaci, aumentando il numero di vittorie e migliorando le prestazioni complessive rispetto alla versione senza assistenza.

Uno sviluppo futuro promettente consiste nell'integrazione di modelli di linguaggio più

avanzati, come quelli disponibili tramite API commerciali di ultima generazione, che offrono capacità significativamente migliorate in termini di comprensione e generazione del linguaggio naturale. Questi modelli potrebbero non solo aumentare la qualità dei suggerimenti forniti, ma anche migliorarne la precisione, adattandosi meglio ai contesti specifici delle interazioni di gioco. Un altro aspetto cruciale per migliorare il sistema riguarda l'ampliamento del dataset di addestramento. Attualmente, il dataset potrebbe essere limitato in termini di varietà e complessità dei contesti considerati. Estenderlo per includere un maggior numero di environment relativi a giochi a turni, caratterizzati da una gamma diversificata di situazioni strategiche, consentirebbe al modello di apprendere regole più generalizzabili e di adattarsi meglio a scenari complessi e inaspettati, aumentando così la sua robustezza.

Un ulteriore passo avanti potrebbe essere rappresentato dall'applicazione della metodologia proposta a nuove categorie di giochi, come i giochi strategici in tempo reale (RTS). A differenza dei giochi a turni, questi titoli presentano una complessità intrinseca più elevata a causa delle dinamiche fluide e non predeterminate. Questo richiederebbe di affrontare sfide aggiuntive, come la necessità di processare decisioni in tempo reale e di anticipare le mosse dell'avversario con rapidità ed efficienza. Infine, dal punto di vista dell'interazione, l'attuale sistema fornisce suggerimenti attraverso prompt testuali, che risultano funzionali ma potenzialmente limitati in termini di immersività. Un possibile sviluppo futuro potrebbe prevedere l'implementazione di una chat vocale interattiva, che consentirebbe ai giocatori di comunicare con gli NPC utilizzando la propria voce. Questa integrazione non solo renderebbe le interazioni più naturali e coinvolgenti, ma contribuirebbe anche a migliorare l'esperienza utente, creando un senso di connessione più profondo con i personaggi virtuali e il mondo di gioco.

Bibliografia

[1] Muhtar Çağkan Uludağlı e Kaya Oğuz. "Non-player character decision-making in computer games". In: *Artificial Intelligence Review* 56.12 (2023), pp. 14159–14191.

[2] Fandom Contributors. *Merchant - Resident Evil Wiki*. URL: `https://residentevil.fandom.com/wiki/Merchant`.

[3] Henrik Warpefelt. "The Non-Player Character: Exploring the believability of NPC presentation and behavior". Tesi di dott. Department of Computer e Systems Sciences, Stockholm University, 2016.

[4] Fandom Contributors. *Category: Skyrim Characters - Elder Scrolls Wiki*. URL: `https://elderscrolls.fandom.com/wiki/Category:Skyrim:_Characters`.

[5] Penny Sweetser. "Large language models and video games: A preliminary scoping review". In: *Proceedings of the 6th ACM Conference on Conversational User Interfaces*. 2024, pp. 1–8.

[6] Lei Huang et al. "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions". In: *ACM Transactions on Information Systems* (2023).

[7] Samuel Rhys Cox e Wei Tsang Ooi. "Conversational Interactions with NPCs in LLM-Driven Gaming: Guidelines from a Content Analysis of Player Feedback". In: *International Workshop on Chatbot Research and Design*. Springer. 2023, pp. 167–184.

[8] Xuyouyang Fan. "The Application of Reinforcement Learning in Video Games". In: *2023 International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2023)*. Atlantis Press. 2023, pp. 202–211.

[9] Sihao Hu et al. "A survey on large language model-based game agents". In: *arXiv preprint arXiv:2404.02039* (2024).

[10] Colin Charles Mathews e Nia Wearn. "How are modern video games marketed?" In: *The computer games journal* 5.1 (2016), pp. 23–37.

[11] Zihan Ding et al. "Introduction to reinforcement learning". In: *Deep reinforcement learning: fundamentals, research and applications* (2020), pp. 47–123.

[12] K O'Shea. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

[13] Wojciech Samek et al. "Explaining deep neural networks and beyond: A review of methods and applications". In: *Proceedings of the IEEE* 109.3 (2021), pp. 247–278.

[14] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[15] S Hochreiter. "Long Short-term Memory". In: *Neural Computation MIT-Press* (1997).

[16] Muhammad Usman Hadi et al. "A survey on large language models: Applications, challenges, limitations, and practical usage". In: *Authorea Preprints* (2023).

[17] Diksha Khurana et al. "Natural language processing: state of the art, current trends and challenges". In: *Multimedia tools and applications* 82.3 (2023), pp. 3713–3744.

[18] A Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

[19] Roberto Gallotta et al. "Large language models and games: A survey and roadmap". In: *arXiv preprint arXiv:2402.18659* (2024).

[20] Anders Tychsen et al. "The game master". In: *ACM International Conference Proceeding Series*. Vol. 123. 2005. 2005, pp. 215–222.

[21] Antonios Liapis. "Searching for sentient design tools for game development". In: (2015).

[22] Bin Hu et al. "Enabling intelligent interactions between an agent and an LLM: A reinforcement learning approach". In: *arXiv preprint arXiv:2306.03604* (2023).

[23] Chaofan Pan et al. "Hi-Core: Hierarchical Knowledge Transfer for Continual Reinforcement Learning". In: *arXiv preprint arXiv:2401.15098* (2024).

[24] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[25] Zihao Zhou et al. “Large language model as a policy teacher for training reinforcement learning agents”. In: *arXiv preprint arXiv:2311.13373* (2023).

[26] Zihao Wang et al. “Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents”. In: *arXiv preprint arXiv:2302.01560* (2023).

[27] Shaoteng Liu et al. “RL-GPT: Integrating Reinforcement Learning and Code-as-policy”. In: *arXiv preprint arXiv:2402.19299* (2024).

[28] Tian Liang et al. “Encouraging divergent thinking in large language models through multi-agent debate”. In: *arXiv preprint arXiv:2305.19118* (2023).

[29] Yuqing Du et al. “Guiding pretraining in reinforcement learning with large language models”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 8657–8677.

[30] Martin Klissarov et al. “Motif: Intrinsic motivation from artificial intelligence feedback”. In: *arXiv preprint arXiv:2310.00166* (2023).

[31] Zelai Xu et al. “Language agents with reinforcement learning for strategic play in the werewolf game”. In: *arXiv preprint arXiv:2310.18940* (2023).

[32] Minae Kwon et al. “Reward design with language models”. In: *arXiv preprint arXiv:2303.00001* (2023).

[33] Wei Zhou, Xiangyu Peng e Mark Riedl. “Dialogue shaping: Empowering agents through npc interaction”. In: *arXiv preprint arXiv:2307.15833* (2023).

[34] Houda Nait El Barj e Théophile Sautory. “Reinforcement Learning from LLM Feedback to Counteract Goal Misgeneralization”. In: *arXiv preprint arXiv:2401.07181* (2024).

[35] Wen Xiao et al. “Personalized Abstractive Summarization by Tri-agent Generation Pipeline”. In: *arXiv preprint arXiv:2305.02483* (2023).

[36] Hyung Won Chung et al. “Scaling instruction-finetuned language models”. In: *Journal of Machine Learning Research* 25.70 (2024), pp. 1–53.

[37] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[38] John Schulman. "Trust Region Policy Optimization". In: *arXiv preprint arXiv:1502.05477* (2015).

[39] Timo Kaufmann et al. "A survey of reinforcement learning from human feedback". In: *arXiv preprint arXiv:2312.14925* (2023).

[40] Volodymyr Mnih. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).