UNIVERSITÀ DEGLI STUDI
DI SALERNO

Ludo: Progetto IA

Eduardo Scarpa
e.scarpa6@studenti.unisa.it

Alfredo Cannavaro
a.cannavaro@studenti.unisa.it

Catello Staiano
c.staiano14@studenti.unisa.it

Abstract

L'apprendimento per rinforzo (Reinforcement Learning, RL) è una tecnica chiave dell'intelligenza artificiale, utilizzata per sviluppare agenti in grado di apprendere strategie ottimali tramite interazione con un ambiente. Questo progetto applica il RL al gioco da tavolo Ludo, sfruttando algoritmi come Q-learning, SARSA e le loro varianti Double. Gli agenti sono stati allenati in un ambiente simulato implementato con *Pygame*, ottenendo percentuali di vittorie superiori all'80%.

Il lavoro include la formalizzazione delle regole di gioco, la definizione degli stati e delle ricompense, e un'analisi comparativa degli algoritmi implementati. Le conclusioni evidenziano l'efficacia degli approcci proposti e suggeriscono futuri sviluppi, come l'uso di Deep RL e scenari multi-agente.

1 Introduzione

1.1 Contesto e motivazioni

L'apprendimento per rinforzo (**Reinforcement Learning**, RL) si è affermato come una delle principali metodologie nell'ambito dell'intelligenza artificiale, grazie alla sua capacità di consentire agli agenti di apprendere comportamenti ottimali attraverso l'interazione con un ambiente. Questa tecnica ha dimostrato il suo potenziale in numerosi campi, tra cui la robotica, il controllo dinamico e, soprattutto, i giochi. I giochi da tavolo, in particolare, rappresentano un campo di studio ideale per il RL, poiché combinano dinamiche strategiche complesse, interazioni tra agenti e meccaniche stocastiche.

1.2 Ludo: Progetto IA

Questo progetto ha riguardato la progettazione e realizzazione di un sistema basato sull'apprendimento per rinforzo, applicato al gioco da tavolo Ludo.

Il Ludo è un gioco di percorso in cui i giocatori devono guidare le proprie pedine verso un'area centrale del tabellone, completando un tragitto prestabilito lungo i bordi del tabellone stesso.

Nel presente lavoro verranno analizzate le tecniche adottate per la creazione dell'ambiente di gioco, la formalizzazione delle regole di Ludo, lo sviluppo dell'agente attraverso gli algoritmi di Q-learning, SARSA, Double Q-learning e Double Sarsa, i risultati ottenuti durante le fasi di training, le difficoltà incontrate, le possibili prospettive per miglioramenti futuri e le considerazioni conclusive.

1.3 Obiettivi del progetto

Il progetto si propone di applicare le tecniche di apprendimento per rinforzo al gioco del **Ludo**, un gioco da tavolo tradizionale che, nonostante la semplicità apparente delle sue regole, presenta notevoli sfide strategiche. Il progetto esplora come agenti intelligenti possano essere sviluppati per competere in modo autonomo ed efficace, adattandosi alle dinamiche del gioco e alle interazioni con altri agenti. Gli obiettivi principali sono:

Strategia: Creare un agente intelligente capace di apprendere una strategia di gioco ottimale, migliorando nel tempo le decisioni tattiche durante le partite.

Stati del gioco: Definire gli stati del gioco in modo preciso, considerando la posizione delle pedine dell'agente e degli avversari, nonché le condizioni del tabellone di gioco.

Azioni: Definire azioni tattiche avanzate che consentano all'agente di prendere decisioni tattiche mirate.

Apprendimento: Sviluppare un sistema di apprendimento per rinforzo.

2 Background Teorico

2.1 Ludo: Il Gioco

Il **Ludo** è un gioco da tavolo tradizionale che si basa su un tabellone suddiviso in quadranti colorati, ciascuno associato a un giocatore. L'obiettivo del gioco è far arrivare tutte le proprie pedine alla "casa base" percorrendo un tragitto definito sul tabellone.

All'inizio del gioco, ogni giocatore posiziona le proprie pedine nella rispettiva casella base sul tabellone. Per poter mettere una pedina nel percorso di gioco, un giocatore deve ottenere un 6 al lancio del dado. Fino a quando non si ottiene un 6, il turno passa al giocatore successivo. Il numero ottenuto al dado determina quante caselle una pedina può avanzare lungo il percorso. Se un giocatore ottiene un altro 6, può scegliere se:

Inserire: Inserire un'altra pedina nel percorso (se ci sono pedine ancora nella base).

Muovere: Muovere una pedina già presente nel percorso.

Quando un giocatore ottiene un 6 ha diritto a un ulteriore tiro del dado. Il tiro del dado serve per determinare i movimenti delle pedine, ma è necessario scegliere con attenzione quale pedina spostare per massimizzare le proprie possibilità di vittoria. Inoltre, se un giocatore ottiene un 6 per tre volte consecutive, è obbligato a passare il turno senza effettuare ulteriori mosse. Quando una pedina completa un giro del percorso, può entrare nella zona sicura, che rappresenta la parte finale del tragitto verso il *goal*. Le regole fondamentali prevedono anche la possibilità di catturare le pedine avversarie e la perdita del turno in alcune circostanze. Per il contesto di questo progetto, vengono considerate esclusivamente le regole fondamentali del gioco. Gli elementi principali del gioco sono:

- **Tabellone**: Un percorso suddiviso in caselle, con punti strategici come la zona di partenza, le caselle sicure e la casa base;
- **Pedine**: Ogni giocatore ha un numero fisso di pedine che devono completare il percorso;
- **Strategia e fortuna**: Sebbene il tiro del dado aggiunga una componente casuale, le decisioni strategiche giocano un ruolo cruciale, ad esempio nello scegliere quando attaccare, difendere o avanzare rapidamente.

Q-learning

Il Q-learning è un algoritmo di apprendimento per rinforzo (Reinforcement Learning) progettato per risolvere problemi decisionali sequenziali. Questo algoritmo consente a un agente di apprendere come prendere decisioni ottimali interagendo con un ambiente. Ogni azione compiuta dall'agente lo porta da uno stato s_t a un altro stato s_{t+1}, ricevendo una ricompensa associata r_t. L'obiettivo principale del Q-learning è massimizzare la ricompensa totale accumulata.

L'algoritmo si basa su una struttura nota come Q-table, una tabella in cui ogni coppia stato-azione (s, a) viene associata a un valore numerico che rappresenta la stima della ricompensa futura attesa per quella coppia. L'agente decide le sue azioni in due modi principali:

- **Esplorazione casuale**: l'agente seleziona un'azione casuale per esplorare l'ambiente.
- **Sfruttamento della conoscenza**: l'agente sceglie l'azione con il valore massimo nella Q-table per lo stato corrente.

Dopo aver scelto un'azione, l'agente aggiorna la Q-table basandosi sulla ricompensa ricevuta e sul valore stimato dello stato futuro. La formula per aggiornare il valore nella Q-table è:

$$Q^{\text{new}}(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \Big(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \Big)$$

dove:

- α è il **learning rate**, che controlla quanto velocemente l'agente apprende dai nuovi dati. Un valore alto implica un adattamento rapido, ma può causare instabilità. Un valore basso garantisce maggiore stabilità ma rallenta la convergenza.
- γ è il **discount factor**, che determina l'importanza delle ricompense future rispetto a quelle immediate. Un valore alto favorisce una visione a lungo termine, mentre un valore basso dà priorità alle ricompense immediate.
- r_t è la ricompensa ricevuta per aver compiuto l'azione a_t nello stato s_t.
- $\max_a Q(s_{t+1}, a)$ rappresenta il valore massimo stimato delle azioni nello stato successivo s_{t+1}.

Sarsa

L'algoritmo Sarsa (*State-Action-Reward-State-Action*) è un'altra tecnica di apprendimento per rinforzo, simile al Q-learning, ma differisce nel modo in cui aggiorna la Q-table. In Sarsa, l'aggiornamento tiene conto sia dello stato e dell'azione correnti, sia dello stato e dell'azione selezionati successivamente.

La formula di aggiornamento per Sarsa è la seguente:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

dove:

- $Q(s_t, a_t)$ è il valore stimato della coppia stato-azione corrente.
- r_t è la ricompensa ottenuta compiendo l'azione a_t nello stato s_t.

- $Q(s_{t+1}, a_{t+1})$ rappresenta il valore stimato della coppia stato-azione successiva.
- α e γ hanno lo stesso significato indicato per il Q-learning.

Differenze tra Q-learning e Sarsa

La principale differenza tra i due algoritmi è nel modo in cui viene calcolato il valore futuro:

- In Q-learning, l'aggiornamento usa il valore massimo della Q-table per il prossimo stato ($\max_a Q(s_{t+1}, a)$), indipendentemente dall'azione effettivamente scelta.
- In Sarsa, invece, l'aggiornamento usa il valore associato all'azione effettivamente selezionata ($Q(s_{t+1}, a_{t+1})$).

Questa distinzione rende il Q-learning una tecnica **off-policy**, mentre Sarsa è un approccio **on-policy**.

Nei metodi **off-policy**, l'agente apprende una politica ottimale π^* basandosi su azioni che non seguono necessariamente la politica attuale. Ad esempio, il Q-learning aggiorna il valore $Q(s, a)$ utilizzando il massimo valore possibile per lo stato successivo ($\max_a Q(s', a)$).

Nei metodi **on-policy**, invece, l'agente apprende valutando e migliorando la stessa politica che sta seguendo. Ad esempio, Sarsa aggiorna il valore $Q(s, a)$ usando la ricompensa e il valore associato all'azione effettivamente scelta ($Q(s', a')$).

3 Double Q-learning

Il **Double Q-learning** è una variante del Q-learning progettata per affrontare il problema del **bias ottimistico** introdotto dall'uso della stessa funzione $Q(s, a)$ sia per selezionare l'azione migliore ($\arg\max_{a'} Q(s', a')$) sia per stimarne il valore. Questo bias può portare a sovrastime nelle stime dei valori Q.

Caratteristiche principali

- **Utilizza due funzioni Q separate**: Q_A e Q_B, che vengono aggiornate in modo alternato.
- **Riduce il bias ottimistico** rompendo la correlazione tra la selezione e la valutazione dell'azione migliore.

Regole di aggiornamento

L'algoritmo alterna l'aggiornamento delle due funzioni Q_A e Q_B.

Se decidiamo di aggiornare Q_A:

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha \cdot \big[r + \gamma \cdot Q_B\big(s', \arg\max_{a'} Q_A(s',a')\big) - Q_A(s,a)\big] \quad (1)$$

Se invece aggiorniamo Q_B:

$$Q_B(s,a) \leftarrow Q_B(s,a) + \alpha \cdot \big[r + \gamma \cdot Q_A\big(s', \arg\max_{a'} Q_B(s',a')\big) - Q_B(s,a)\big] \quad (2)$$

Dettagli del funzionamento

1. **Selezione casuale dell'aggiornamento**: a ogni passo, l'algoritmo sceglie casualmente quale delle due funzioni (Q_A o Q_B) aggiornare.
2. **Decoupling della selezione dell'azione e della sua valutazione**: l'azione $\arg\max_{a'} Q(s', a')$ viene selezionata sulla base della funzione Q che non viene aggiornata. Questo riduce il rischio di sovrastima.

Vantaggi

- Riduce il bias ottimistico presente nel Q-learning standard.
- Migliora l'accuratezza delle stime dei valori Q in ambienti rumorosi o con grande variabilità.

Limiti

- Può richiedere più risorse computazionali rispetto al Q-learning standard, poiché mantiene due funzioni separate.
- È un algoritmo **off-policy**, quindi il comportamento dell'agente può differire dalla politica appresa.

4 Double SARSA

Il **Double SARSA** è una variante dell'algoritmo SARSA (on-policy) che, come il Double Q-learning, utilizza **due funzioni** Q (Q_A e Q_B) per ridurre il bias ottimistico. A differenza del Double Q-learning, Double SARSA opera **on-policy**, basandosi sull'azione a' effettivamente eseguita dall'agente in conformità alla politica corrente.

Caratteristiche principali

- **On-policy**: utilizza transizioni effettive generate dalla politica corrente.
- **Riduzione del bias**: grazie all'uso di due funzioni separate, rompe la correlazione tra selezione e valutazione dell'azione.

Regole di aggiornamento

Anche qui l'aggiornamento alterna Q_A e Q_B.

Se aggiorniamo Q_A:

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha \cdot \big[r + \gamma \cdot Q_B(s',a') - Q_A(s,a)\big] \quad (3)$$

Se aggiorniamo Q_B:

$$Q_B(s,a) \leftarrow Q_B(s,a) + \alpha \cdot \big[r + \gamma \cdot Q_A(s',a') - Q_B(s,a)\big] \quad (4)$$

Dettagli del funzionamento

1. **Aggiornamento alternato**: a ogni passo, viene scelta casualmente quale funzione Q aggiornare (Q_A o Q_B).

2. **Valutazione basata sulla politica corrente**: Double SARSA utilizza l'azione a' che è stata effettivamente scelta dall'agente, diversamente dal Double Q-learning che utilizza arg max.

Vantaggi

- Adatto ad ambienti on-policy, dove l'agente segue la politica corrente per l'apprendimento.
- Riduce il bias ottimistico senza modificare il comportamento on-policy.

Limiti

- Come SARSA, può essere più lento nell'apprendimento rispetto al Q-learning o Double Q-learning, poiché segue la politica corrente, che può includere esplorazione inefficiente.

5 Differenze tra Double Q-learning e Double SARSA

Caratteristica	Double Q-learning	Double SARSA
Tipo	Off-policy	On-policy
Valutazione	$\arg\max_{a'} Q$	Azione a'
Bias	Ridotto (Q_A/Q_B)	Ridotto (Q_A/Q_B)
Applicazione	Esplorazioni varie	Politiche stabili
Efficienza	Più rapido	Più lento

Table 1: Differenze tra Double Q-learning e Double SARSA

6 Metodologia

6.1 Environment

L'ambiente è un elemento chiave nell'apprendimento per rinforzo (Reinforcement Learning, RL), poiché rappresenta lo spazio in cui l'agente opera e con cui interagisce per massimizzare una ricompensa cumulativa. Nel contesto di questo progetto, l'ambiente è ispirato al gioco da tavolo Ludo e viene implementato utilizzando la libreria *Pygame*. Esso è suddiviso in quattro aree principali:

- **START:** Posizioni iniziali delle pedine dell'agente $(2, 2$ e $2, 3)$ e della CPU $(2, 11$ e $2, 12)$.
- **PATH:** Percorso di movimento delle pedine. Per l'agente, va da $(6, 2)$ a $(6, 1)$, mentre per la CPU si estende da $(2, 8)$ a $(1, 8)$.
- **SAFE:** Zona di sicurezza vicina al *goal*, destinata all'agente, che si estende da $(7, 1)$ a $(7, 5)$.
- **FINISH:** Posizione finale, corrispondente alla cella $(7, 6)$, che rappresenta l'obiettivo del gioco.

Le interazioni tra agente e ambiente sono governate da stati, azioni e ricompense. Nello specifico:

- **Stati (s):** Configurazioni che descrivono la posizione delle pedine nell'ambiente. Gli stati includono le aree **START**, **PATH**, **SAFE**, **FINISH** e due stati aggiuntivi (**SUPERATA 1** e **SUPERATA 2**), introdotti per gestire situazioni in cui una pedina viene superata.
- **Azioni (a):** L'agente può compiere due azioni: muovere la pedina 1 o muovere la pedina 2.
- **Ricompensa (r):** Un segnale numerico che guida l'apprendimento. Le ricompense positive incentivano il raggiungimento del *finish*, mentre ricompense negative penalizzano situazioni indesiderate, come il superamento da parte della CPU.

6.1.1 Costruzione Stati del Gioco

Inizialmente, la lista degli stati possibili era composta da 10 configurazioni, basate sul numero di pedine dell'agente presenti in ciascuna delle quattro aree principali: **START**, **PATH**, **SAFE** e **FINISH**. Ogni stato rappresentava una combinazione tra 0, 1 o 2 pedine in ciascuna zona. Questa rappresentazione non consentiva di assegnare ricompense negative, rendendo difficile per l'agente apprendere strategie efficaci in situazioni sfavorevoli.

Per risolvere il problema, sono stati introdotti due nuovi stati, **SUPERATA 1** e **SUPERATA 2**, che indicano rispettivamente il superamento della prima o della seconda pedina da parte della CPU. Questo approccio consente di penalizzare tali eventi, incoraggiando l'agente a recuperare terreno con azioni appropriate. Con questa modifica, il totale degli stati possibili è passato da 10 a 34, ampliando la capacità dell'agente di apprendere strategie più complesse.

7 Risultati

In questa sezione vengono illustrati i risultati ottenuti durante il processo di training dell'agente nell'ambiente definito. L'obiettivo principale del training era quello di sviluppare un comportamento ottimale che consentisse all'agente di massimizzare la ricompensa cumulativa e minimizzare il numero di mosse necessarie per raggiungere l'obiettivo.

Le tre iterazioni del training sono state progettate per testare diverse configurazioni di rewards e valutare l'efficacia delle strategie apprese dall'agente in risposta ai cambiamenti introdotti. Ogni iterazione è stata caratterizzata da una diversa combinazione di penalizzazioni e incentivi, permettendo di osservare l'impatto diretto

di queste modifiche sul processo di apprendimento. Il comportamento dell'agente è stato analizzato attraverso metriche chiave come il tasso di successo, il numero medio di mosse per episodio e la ricompensa cumulativa media.

7.1 Rewards

Di seguito riportiamo i rewards utilizzati nel corso delle varie fasi di training, organizzati a mo' di matrice:

Prima iterazione

$[1, 1]$,
$[0.5, 0.5], [0.6, 0], [0, 0.6]$,
$[0.5, 0.5], [0.6, 0.3], [0.3, 0.6], [0.5, 0.5]$,
$[0.5, 0.5], [0, 0.6], [0.6, 0]$,
$[0.5, 0.5], [0.7, 0.3], [0.3, 0.7], [0.5, 0.5]$,
$[0.5, 0.5], [0.6, 0.2], [0.2, 0.6], [0.5, 0.5]$,
$[0.6, 0.6], [0, 0.6], [0.5, 0.5]$,
$[0.5, 0.3], [0.3, 0.5], [0.5, 0.5], [0.5, 0.5]$,
$[0.4, 0.5], [0.5, 0.4], [0.3, 0.6], [0.5, 0.5]$,
$[0.5, 0.7], [0.7, 0.5], [0.6, 0.6]$.

Seconda iterazione

$[5, 5]$,
$[0.2, 0.2], [0, 0.3], [0.3, 0]$,
$[0.2, 0.2], [-0.9, 0.5], [0.5, -0.9], [0.2, 0.2]$,
$[0, 0], [0.2, 0], [0, 0.2]$,
$[0.2, 0.2], [-0.9, 0.5], [0.5, -0.5], [0.2, 0.2]$,
$[0.3, 0.3], [0.5, -0.9], [-0.9, 0.5], [0.2, 0.2]$,
$[0, 0], [-0.9, 0.5], [0.5, -0.9]$,
$[0, 0], [-0.9, 0.5], [0.5, -0.9], [0.2, 0.2]$,
$[0.2, 0.2], [-0.7, 0.4], [0.4, -0.7], [0.2, 0.2]$,
$[0.3, 0.3], [-0.5, 0.6], [0.6, -0.5], [0.3, 0.3]$.

Terza iterazione

$[7, 7]$,
$[0.5, 0.5], [0, 0.7], [0.7, 0]$,
$[0.5, 0.5], [-2.0, 0.7], [0.7, -2.0], [0.5, 0.5]$,
$[0, 0], [0.5, 0], [0, 0.5]$,
$[0.5, 0.5], [-2.0, 0.9], [0.9, -0.9], [0.5, 0.5]$,
$[0.7, 0.7], [0.9, -2.0], [-2.0, 0.7], [0.5, 0.5]$,
$[0, 0], [-2.0, 0.9], [0.9, -2.0]$,
$[0, 0], [-2.0, 0.9], [0.9, -2.0], [0.5, 0.5]$,
$[0.5, 0.5], [-1.5, 0.6], [0.6, -1.5], [0.5, 0.5]$,
$[0.7, 0.7], [-0.9, 1.0], [1.0, -0.9], [0.7, 0.7]$.

Iperparametri utilizzati

Gli iperparametri selezionati hanno avuto un ruolo cruciale nell'influenzare il comportamento dell'agente e la velocità di apprendimento. Di seguito sono riportati i valori utilizzati:

Learning rate (α): 0.4,
Discount factor (γ): 0.97,
Exploration probability iniziale (ϵ): 1.0,
Exploration probability finale (ϵ_{final}): 0.01,
Fattore di decrescita (ϵ_{decay}): 0.995,
Numero massimo di episodi: 5000,
Dimensione matrice Q: 34×2,
Reward per goal: $+50$,
Penalità per vicinanza a un nemico: -10.

7.2 QLearning

Prima iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Q-learning.

Percentuale vittorie agente: 82.54%
Percentuale vittorie CPU: 17.46%

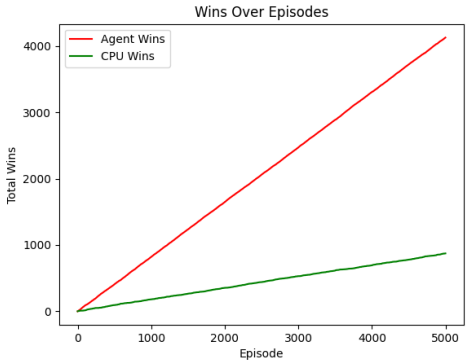

Figure 1: Prima iterazione: andamento delle vittorie durante il training con Q-learning.

Seconda iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Q-learning.

Percentuale vittorie agente: 82.70%
Percentuale vittorie CPU: 17.30%

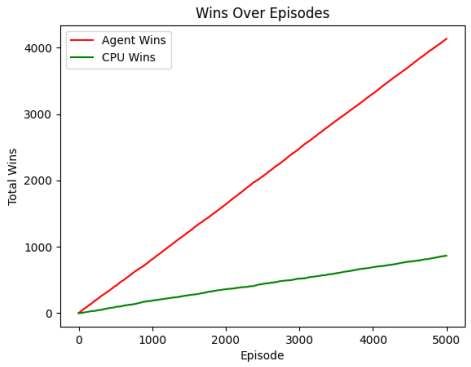

Figure 2: Seconda iterazione: andamento delle vittorie durante il training con Q-learning.

Terza iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Q-learning.

Percentuale vittorie agente: 82.80%
Percentuale vittorie CPU: 17.20%

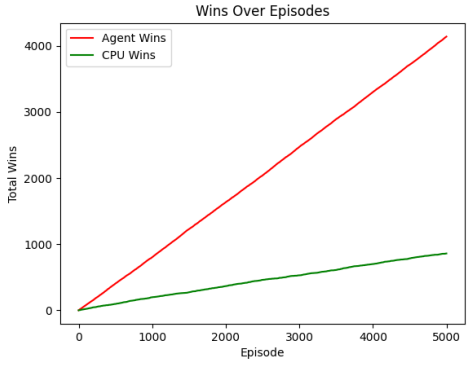

Figure 3: Terza iterazione: andamento delle vittorie durante il training con Q-learning.

7.3 Sarsa

Prima iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Sarsa.

Percentuale vittorie agente: 81.56%
Percentuale vittorie CPU: 18.44%

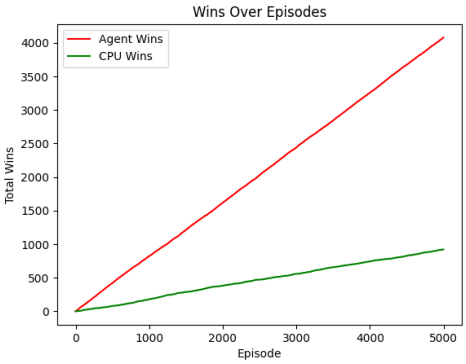

Figure 4: Prima iterazione: andamento delle vittorie durante il training con Sarsa.

Seconda iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Sarsa.

Percentuale vittorie agente: 82.96%
Percentuale vittorie CPU: 17.04%

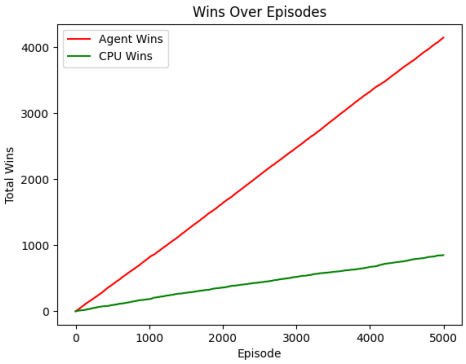

Figure 5: Seconda iterazione: andamento delle vittorie durante il training con Sarsa.

Terza iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Sarsa.

Percentuale vittorie agente: 82.72%
Percentuale vittorie CPU: 17.28%

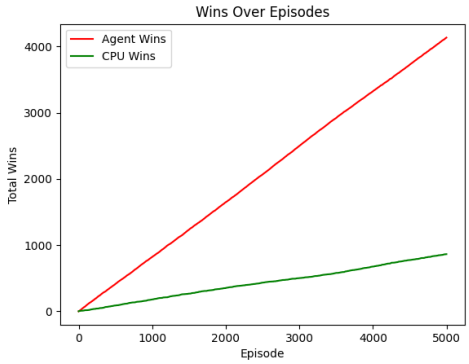

Figure 6: Terza iterazione: andamento delle vittorie durante il training con Sarsa.

7.4 Double Q-learning

Prima iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Double Q-learning.

Percentuale vittorie agente: 82.04%
Percentuale vittorie CPU: 17.96%

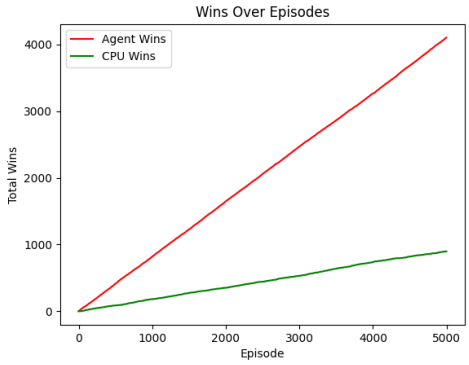

Figure 7: Prima iterazione: andamento delle vittorie durante il training con Double Q-learning.

Seconda iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Double Q-learning.

Percentuale vittorie agente: 81.54%
Percentuale vittorie CPU: 18.46%

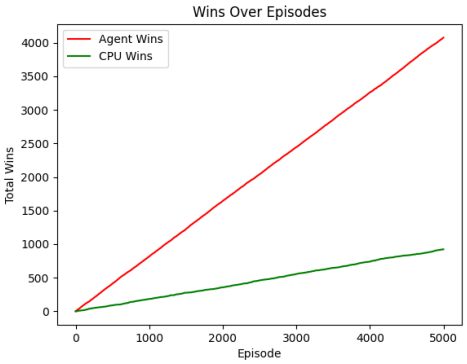

Figure 8: Seconda iterazione: andamento delle vittorie durante il training con Double Q-learning.

Terza iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Double Q-learning.

Percentuale vittorie agente: 81,43%
Percentuale vittorie CPU: 18,57%

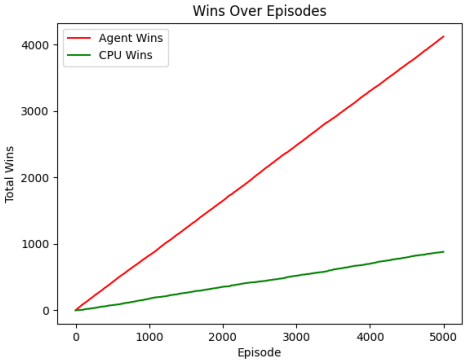

Figure 9: Terza iterazione: andamento delle vittorie durante il training con Double Q-learning.

7.5 Double Sarsa

Prima iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Double Sarsa.

Percentuale vittorie agente: 82,38%
Percentuale vittorie CPU: 17,62%

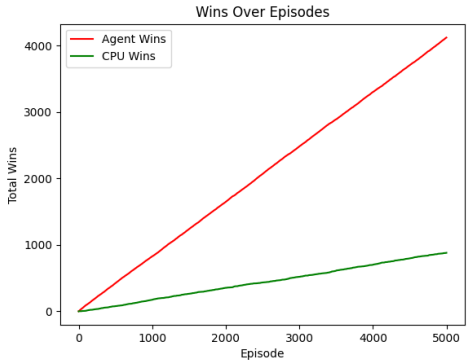

Figure 10: Prima iterazione: andamento delle vittorie durante il training con Double Sarsa.

Seconda iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Double Sarsa.

Percentuale vittorie agente: 82,78%
Percentuale vittorie CPU: 17,22%

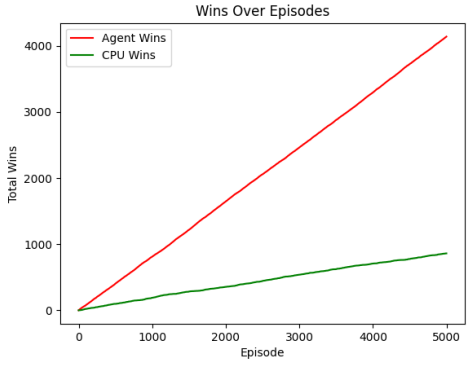

Figure 11: Seconda iterazione: andamento delle vittorie durante il training con Double Sarsa.

Terza iterazione

Il seguente grafico mostra l'andamento delle vittorie dell'agente e della CPU durante il processo di training con l'algoritmo Double Sarsa.

Percentuale vittorie agente: 81,94%
Percentuale vittorie CPU: 18,06%

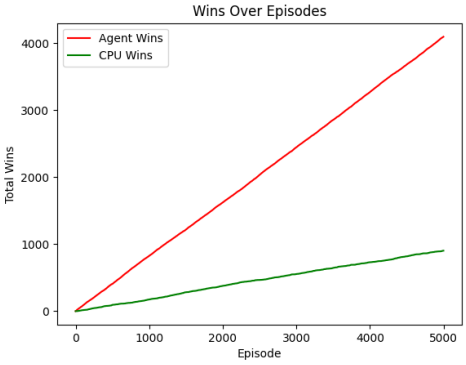

Figure 12: Terza iterazione: andamento delle vittorie durante il training con Double Sarsa.

8 Discussione

L'analisi dei risultati ottenuti durante il training degli algoritmi di apprendimento per rinforzo evidenzia comportamenti distintivi e prestazioni comparabili tra i diversi metodi implementati. Di seguito sono riportati i principali punti emersi:

Confronto tra gli algoritmi

Q-learning L'algoritmo Q-learning ha mostrato una costante crescita nelle prestazioni dell'agente durante il training, raggiungendo una percentuale di vittorie dell'82.80% al termine della terza iterazione. Questo dimostra che il metodo, basato sull'aggiornamento delle stime dei valori d'azione attraverso la massimizzazione, è efficace nel contesto analizzato. Inoltre, il miglioramento tra una iterazione e l'altra è risultato marginale, indicando una possibile saturazione nelle capacità di apprendimento del modello.

Sarsa L'algoritmo Sarsa ha prodotto risultati comparabili al Q-learning, con una percentuale di vittorie massima dell'82.96%. Il comportamento dell'agente è risultato più conservativo rispetto al Q-learning, grazie alla politica on-policy utilizzata durante il training. Questo può aver ridotto il rischio di esplorazioni eccessivamente rischiose, rendendo l'apprendimento più stabile.

Double Q-learning Double Q-learning ha registrato percentuali di vittorie leggermente inferiori rispetto al Q-learning tradizionale, con un massimo dell'81.54% nella seconda iterazione. Ciò può essere attribuito al processo di separazione dei valori Q per mitigare il

bias positivo tipico del Q-learning. Sebbene questa tecnica riduca il rischio di sovrastime, potrebbe rallentare l'apprendimento in determinati scenari.

Double Sarsa Double Sarsa ha fornito prestazioni competitive, raggiungendo un massimo dell'82.78% di vittorie nella seconda iterazione. L'approccio combinato tra on-policy e riduzione del bias positivo sembra aver bilanciato efficacemente esplorazione e sfruttamento. Anche in questo caso, le percentuali di vittorie sono rimaste leggermente inferiori rispetto al Sarsa standard.

Osservazioni generali

L'analisi dei risultati ottenuti nel contesto del gioco da tavolo Ludo ha evidenziato che tutti gli algoritmi analizzati, sia Q-learning che SARSA, insieme alle loro varianti Double, sono stati in grado di apprendere strategie efficaci, con differenze marginali nelle prestazioni complessive. Le percentuali di vittorie ottenute dagli agenti mostrano che le tecniche di apprendimento per rinforzo implementate sono adatte a gestire le dinamiche complesse e stocastiche del gioco.

La scelta degli iperparametri, come il tasso di apprendimento, il fattore di sconto e la strategia di esplorazione, ha avuto un impatto determinante sull'efficacia dell'apprendimento. In particolare, la strategia di decrescita graduale della probabilità di esplorazione (ϵ) si è dimostrata cruciale per bilanciare adeguatamente l'esplorazione di nuove strategie con lo sfruttamento delle conoscenze acquisite. È interessante notare che, sebbene le varianti Double abbiano mostrato una riduzione del bias positivo nel processo di aggiornamento dei valori Q, questo non si è tradotto in un incremento significativo delle percentuali di vittorie.

9 Conclusioni

Limitazioni e possibili sviluppi

Il progetto presenta alcune limitazioni legate principalmente alla configurazione dell'ambiente e alla complessità del dominio. Sebbene gli algoritmi abbiano dimostrato di adattarsi al gioco di Ludo, i risultati ottenuti sono fortemente influenzati dalla scelta degli iperparametri. Pertanto, un'analisi sistematica di tuning potrebbe migliorare ulteriormente l'efficacia degli agenti. Inoltre, il progetto è stato implementato seguendo principalmente le regole base di Ludo; un'evoluzione significativa potrebbe derivare dall'introduzione di regole più dettagliate, in grado di aumentare la complessità strategica e simulare meglio scenari reali di gioco.

Un'altra possibile estensione riguarda l'aumento del numero di partecipanti all'interno dell'ambiente di simulazione, passando da scenari a due giocatori a quelli con tre o quattro partecipanti. Questo cambiamento non solo amplificherebbe la variabilità delle strategie, ma consentirebbe anche di analizzare più approfonditamente le interazioni multi-agente e il potenziale degli algoritmi di apprendimento per rinforzo in contesti più competitivi e complessi.

Infine, in contesti più articolati, l'adozione di algoritmi avanzati come Deep Q-learning, che sfruttano reti neurali per gestire spazi di stato più ampi e dinamici, potrebbe rappresentare un passo evolutivo significativo. Anche l'introduzione di metriche aggiuntive per una valutazione più approfondita del comportamento degli agenti, come il tempo medio necessario per raggiungere un obiettivo o l'impatto delle penalità nelle strategie di gioco, potrebbe offrire un quadro più dettagliato delle performance degli algoritmi utilizzati.

10 Data availability

Codice workflow: GitHub.