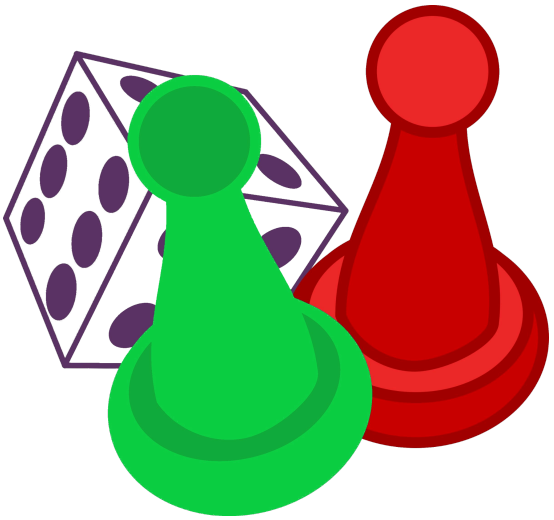
Università degli Studi di Salerno

Corso di Intelligenza Artificiale

Applicazione del Q-learning & Sarsa al Gioco da Tavolo Ludo



Progetto di Cerullo Mary e Selice Andrea

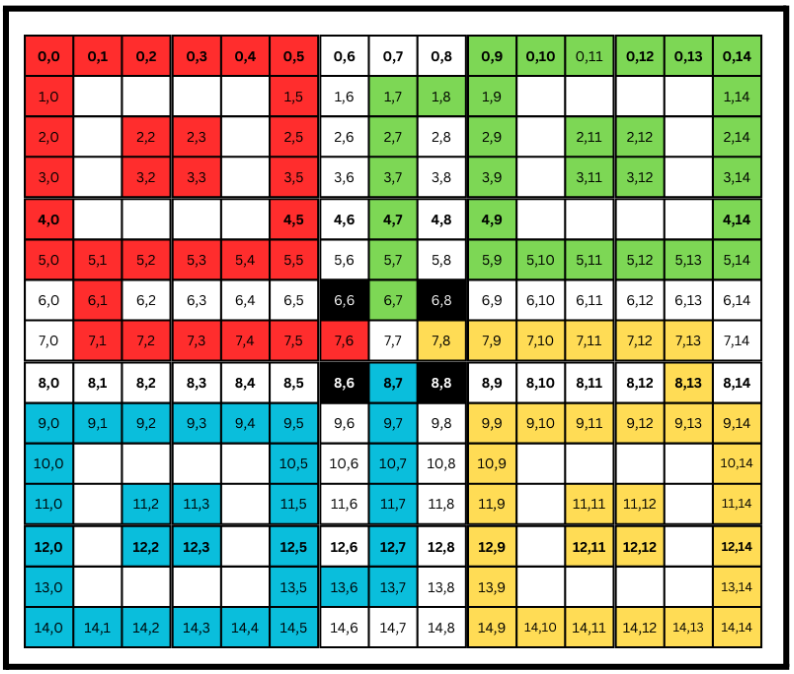
INTRODUZIONE

Questo progetto ha previsto lo sviluppo e l'implementazione di un sistema di apprendimento per rinforzo finalizzato al gioco da tavolo Ludo.
Ludo è un gioco da tavolo di percorso il cui obiettivo consiste nel condurre le proprie pedine al goal centrale del tabellone, seguendo un percorso lungo i margini dello stesso.
Di seguito, verranno esaminate le metodologie impiegate per la creazione dell'ambiente di gioco, la definizione delle regole di Ludo, lo sviluppo dell'agente addestrato mediante l'uso di Q-Learning e Sarsa, i risultati ottenuti durante il processo di addestramento, le sfide affrontate, possibili direzioni per lo sviluppo futuro e le conclusioni tratte.

METODOLOGIA

Environment

La definizione ell'environment è stata resa possibile tramite l'utilizzo della libreria Pygame.
All'interno dell'environment possiamo ritrovare quattro zone di interesse che costituiranno la base delle osservazioni:



BASE: le due pedine a disposizione dell'agente si troveranno nelle posizioni (2,2) e (2,3) rispettivamente. Le pedine della CPU in posizione (2,11) e (2,12) rispettivamente.

PATH: riguarda le posizioni in cui ciascuna pedina può muoversi. Vanno dalla posizione (6,2) alla posizione (6,1) per le pedine dell'agente e dalla posizione (2,8) alla posizione (1,8) per la CPU.
SAFE: riguarda la zona vicina al punto di fine partita. Per l'agente va dalla posizione (7,1) alla posizione (7,5).
GOAL: riguarda la posizione di fine partita e l'obiettivo dell'agente. La posizione è quella relativa alla cella (7,6).

Regole di Ludo

Inizialmente, ogni giocatore dispone le proprie pedine nella rispettiva casella base sul tabellone. Per avviare il gioco, i partecipanti, a turno, lanciano il dado. Affinché una pedina possa essere messa nel path di gioco, è necessario ottenere un risultato di 6. Fino a quando non si ottiene un 6, il turno deve essere passato. Una volta ottenuto un 6, il giocatore ha il diritto di posizionare una pedina nel path e ha anche la possibilità di effettuare un ulteriore lancio di dado.
Il valore del dado determina il numero di caselle che una pedina può avanzare nel path.
Se un giocatore ottiene nuovamente un 6, può decidere se inserire un'altra pedina nel path o muovere una pedina già posizionata nel path.
Tuttavia, se un giocatore ottiene un risultato di 6 per tre volte consecutive, è tenuto a passare il turno. Quando una pedina effettua un giro completo del path, è ammessa nella safe zone, che costituisce la via verso il goal finale.
Ludo include altre regole, come il controllo di quattro pedine, la possibilità di catturare pedine avversarie, e la perdita del turno in diverse circostanze. Tuttavia, per il contesto di questo progetto, vengono considerate esclusivamente le regole fondamentali del gioco.

Sviluppo dell'Agente

Azioni

L'agente può compiere due azioni:
- muovere la pedina 1
- muovere la pedina 2.

Stati

In un primo momento, sulla base delle osservazioni, la lista degli stati possibili è pari a 10. Le pedine a disposizione dell'agente si possono trovare (0,1 o 2) nella base, (0,1 0 2) nel path, (0, 1 o 2) nella safe e (0,1 o 2) nel goal che rappresenta il punto finale della partita nel

momento che entrambe le pedine dell'agente si trovano in questa particolare zona. La lista così definita è riportata nella tabella di seguito:

BASE	PATH	SAFE	GOAL
0	0	0	2
0	0	1	1
0	0	2	0
0	1	0	1
0	1	1	0
0	2	0	0
1	0	0	1
1	0	1	0
1	1	0	0
2	0	0	0

PROBLEMA: il problema relativo alla definizione di questi stati è legato al fatto che in questo modo non vi è possibilità di assegnare reward negativi all'agente.

SOLUZIONE: vengono introdotti due nuove opzioni: SORPASSATA 1 e SORPASSATA 2. Questa soluzione permette di fare in modo che l'agente possa imparare una strategia per la quale nel momento in cui venga sorpassata la prima pedina (SORPASSATA 1) si possa assegnare un maggior reward alla scelta dell'azione corrispondente al movimento della stessa per recuperare terreno. Lo stesso è stato applicato anche alla seconda pedina. Il totale degli stati possibili è pari a 34.
Di conseguenza la lista degli stati è stata modificata come segue:

BASE	PATH	SAFE	GOAL	SORPASSATA 1	SORPASSATA 2
0	0	0	2	0	0
0	0	1	1	0	0
0	0	1	1	0	1
0	0	1	1	1	0
0	0	2	0	0	0
0	0	2	0	0	1
0	0	2	0	1	0

0	0	2	0	1	1
0	1	0	1	0	0
0	1	0	1	1	0
0	1	0	1	0	1
0	1	1	0	0	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	0	1	1
0	2	0	0	0	0
0	2	0	0	1	0
0	2	0	0	0	1
0	2	0	0	1	1
1	0	0	1	0	0
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	0	1	1
1	1	0	0	0	0
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	0	1	1
2	0	0	0	0	0
2	0	0	0	0	1
2	0	0	0	1	0
2	0	0	0	1	1

Q-learning

Q-learning è un algoritmo di reinforcement learning usato per risolvere problemi decisionali sequenziali, in cui un agente impara a prendere decisioni interagendo con un ambiente. Effettuando un'azione, l'agente passa da uno stato ad un altro e ogni stato è associato ad una ricompensa. L'obiettivo è massimizzare la ricompensa totale.
L'algoritmo del Q-learning si basa sulla tabella Q-table, che va a memorizzare un valore per ogni coppia stato-azione. Questo valore rappresenta la stima della ricompensa attesa che l'agente otterrà se compie una certa azione in un certo stato.
L'agente può decidere quale azione intraprendere in due modi:

- Esplorazione casuale
- Sfruttamento della conoscenza: l'agente sceglie l'azione con il valore massimo della Q-table per lo stato corrente

Una volta scelta l'azione, l'agente osserverà la ricompensa ottenuta e il nuovo stato in cui si trova e sfrutta queste informazioni per effettuare l'aggiornamento della Q-table come segue:

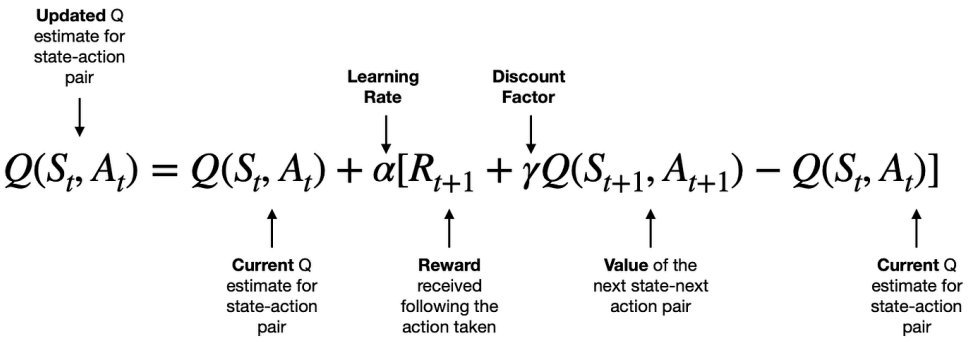
$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

dove:

- Learning rate: indica quanto l'agente dovrebbe aggiornare i valori nella Q-table con le informazioni appena ottenute. Un learning rate alto implicherà che l'agente si andrà ad adattarsi velocemente alle nuove informazioni, ma potrebbe essere meno stabile. Viceversa, un learning rate più basso renderà l'aggiornamento più stabile ma richiede più tempo per convergere.
- Discount factor: indica quanto l'agente attribuisce valore alle ricompense future. Un discount factor alto implica che l'agente tenga maggiormente conto delle ricompense future. Viceversa un valore più basso rende l'agente più orientato alle ricompense presenti.

Sarsa

Sarsa è un algoritmo di reinforcement learning simile al Q-learning, che differisce nell'aggiornamento della Q-table che viene effettuato non solo in base allo stato e all'azione corrente, ma anche in base al prossimo stato e alla prossima azione selezionata.
La Q-table viene aggiornata come segue:



RISULTATI

Rewards

Di seguito riportiamo i rewards utilizzati nel corso delle varie fasi di training:

Prima iterazione	Seconda 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]	[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]	

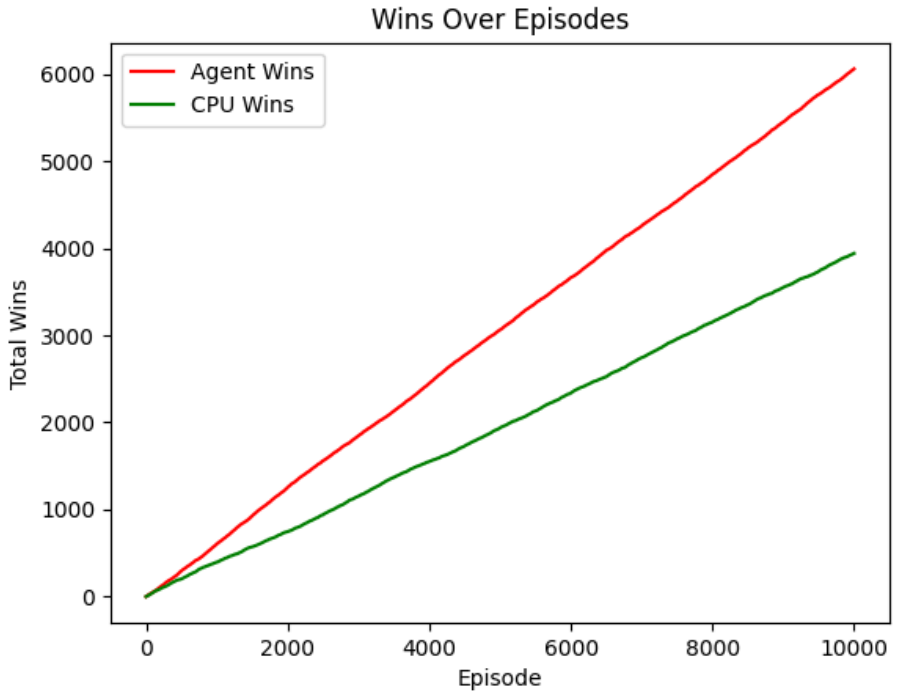
Q-Learning

Iperparametri

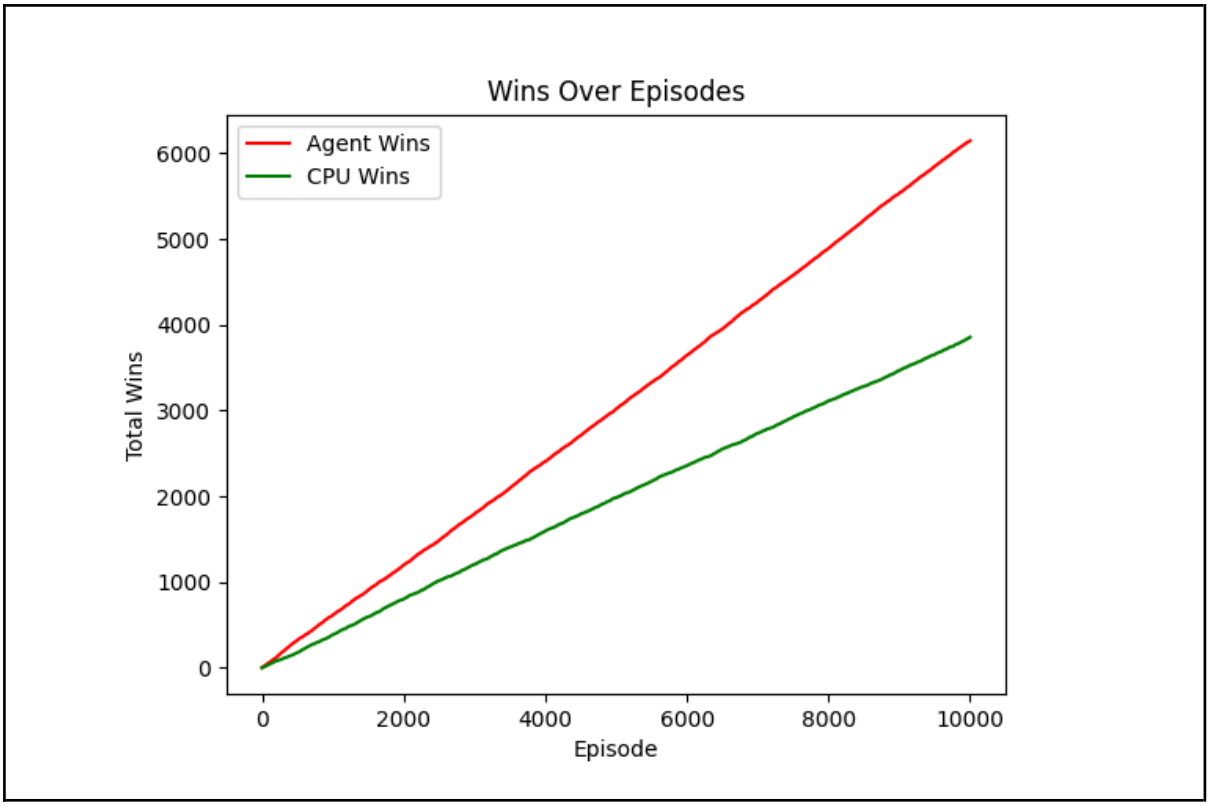
Di seguito mostriamo i risultati ottenuti utilizzando i rewards della prima iterazione.

learning_rate	0.4
discount_factor	0.95

exploration_prob	0.1
num_episodes	10000
Vittorie agente	6059 (60.59%)
Vittorie cpu	3941 (39.41%)

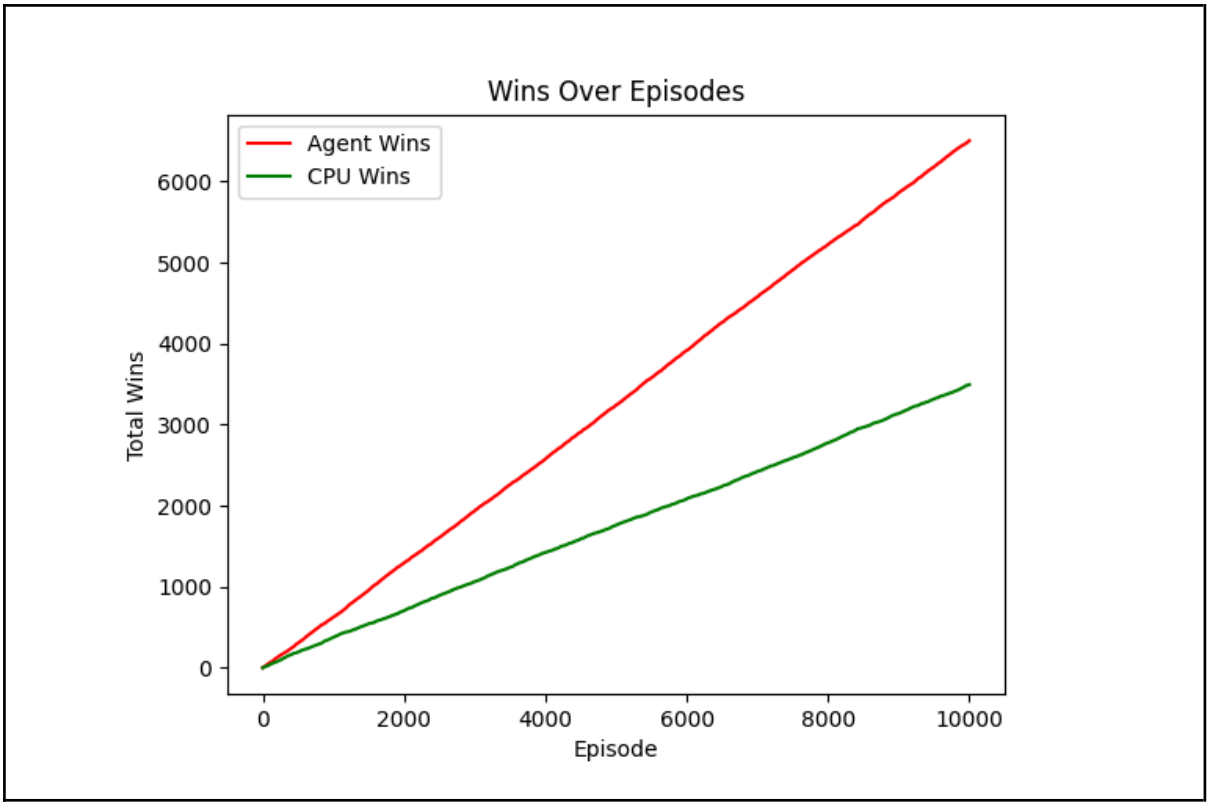


learning_rate	0.5
discount_factor	0.95
exploration_prob	0.3
num_episodes	10000
Vittorie agente	6145 (61.45%)
Vittorie cpu	3855 (38.55%)

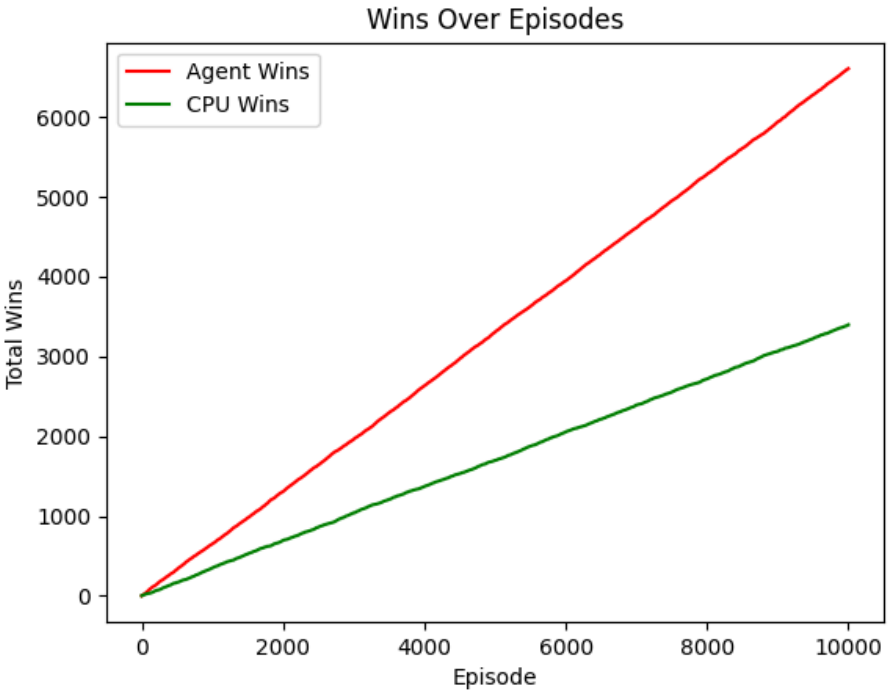


Di seguito mostriamo i risultati ottenuti con i rewards della seconda iterazione.

learning_rate	0.4
discount_factor	0.95
exploration_prob	0.1
num_episodes	10000
Vittorie agente	6504 (65.04%)
Vittorie cpu	3496 (34.96%)



learning_rate	0.3
discount_factor	0.95
exploration_prob	0.1
num_episodes	10000
Vittorie agente	6604 (66.04%)
Vittorie cpu	3396 (33.96%)

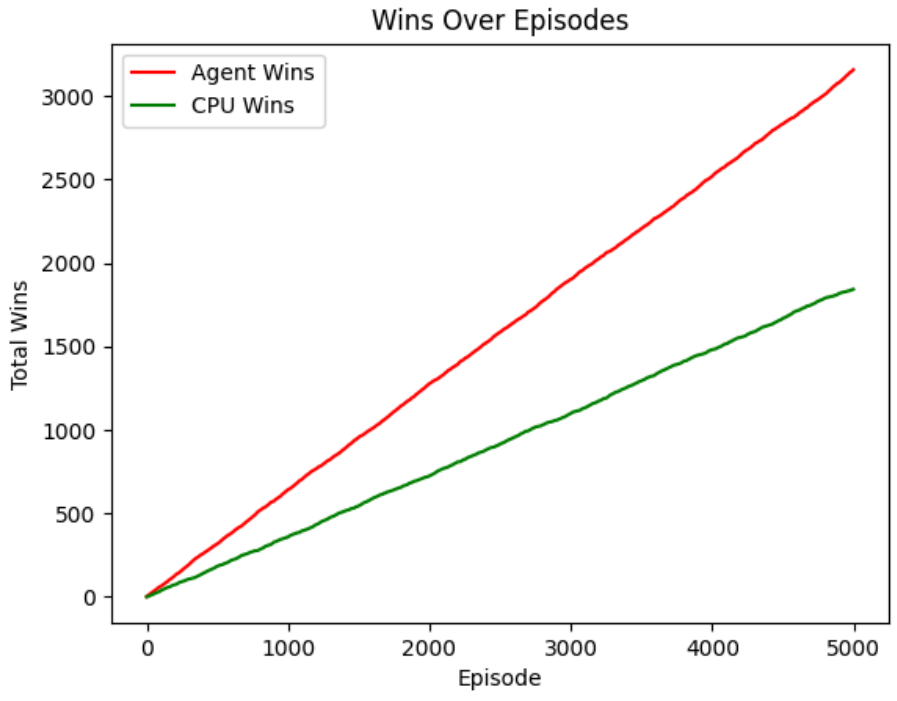


learning_rate	0.5
discount_factor	0.95
exploration_prob	0.3
num_episodes	5000
Vittorie agente	3061 (61.22%)
Vittorie cpu	1939 (38.78%)

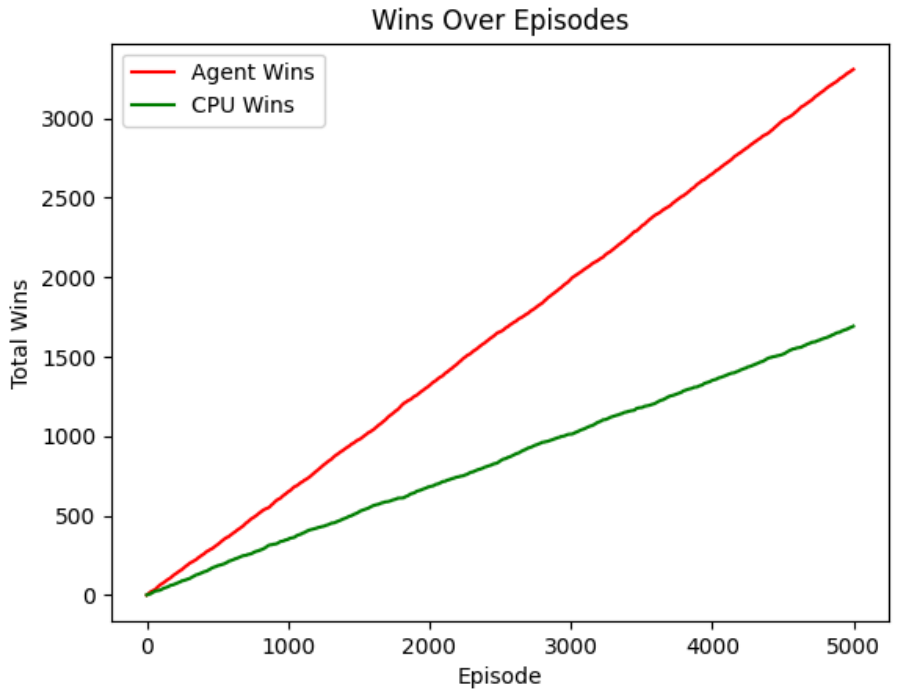
learning_rate	0.8
discount_factor	0.95
exploration_prob	0.2
num_episodes	5000

Vittorie agente	3123 (62.46%)
Vittorie cpu	1877 (37.54%)

learning_rate	0.3
discount_factor	0.50
exploration_prob	0.1
num_episodes	5000
Vittorie agente	3158 (63.16%)
Vittorie cpu	1842 (36.84%)



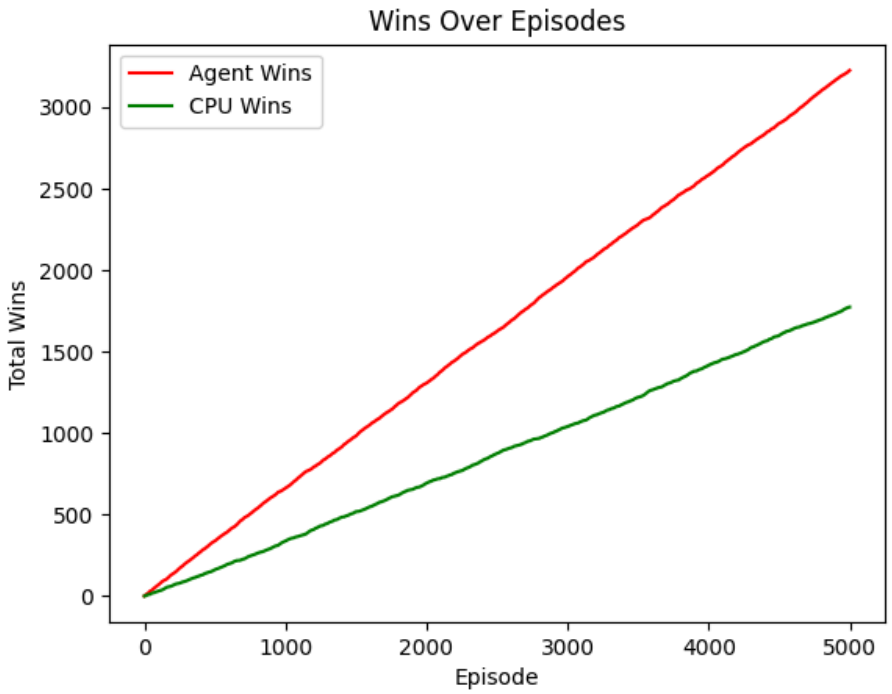
learning_rate	0.3
discount_factor	0.95
exploration_prob	0.1
num_episodes	5000
Vittorie agente	3308 (66.16%)
Vittorie cpu	1692 (33.84%)



Di seguito riportiamo i risultati ottenuti con i rewards della terza iterazione:

learning_rate	0.3
discount_factor	0.95
exploration_prob	0.1
num_episodes	5000
Vittorie agente	3226 (64.52%)

Vittorie cpu	1774 (35.48%)

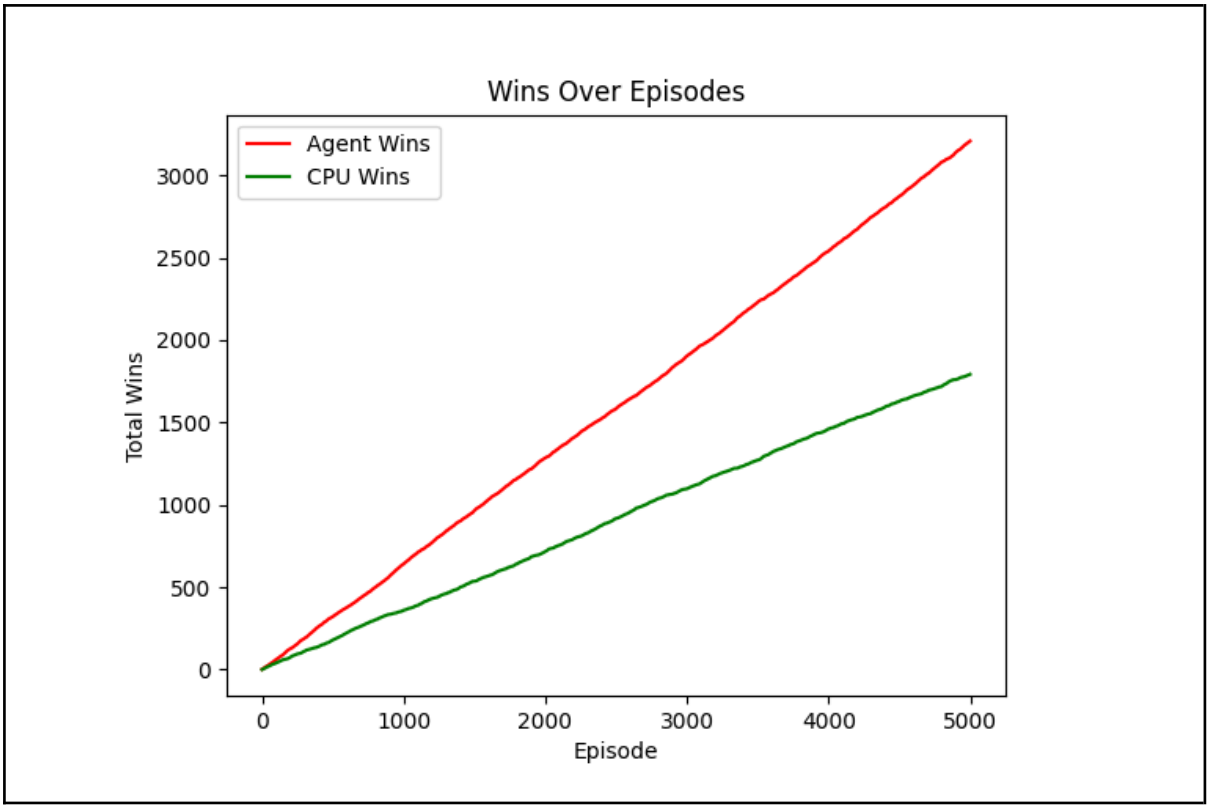


Sarsa

Iperparametri

Di seguito mostriamo i risultati ottenuti con i rewards della seconda iterazione.

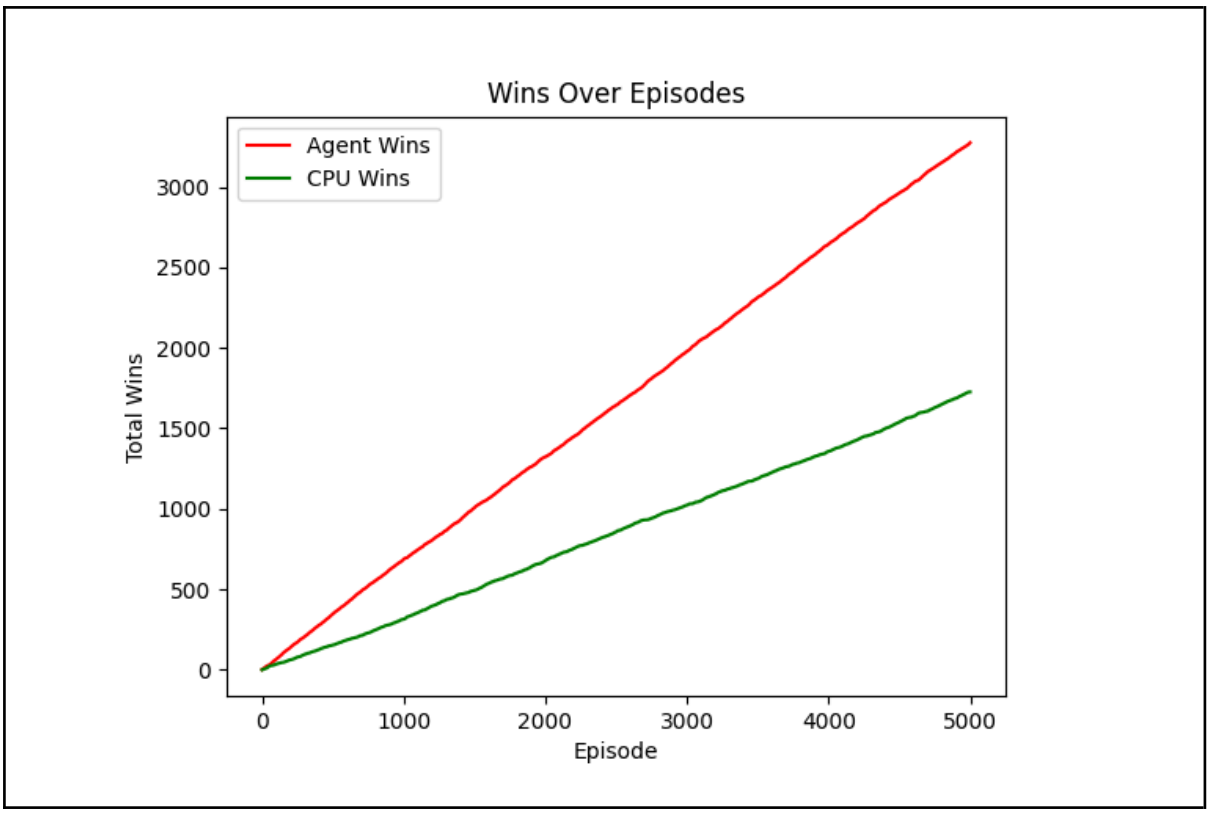
learning_rate	0.3
discount_factor	0.95
exploration_prob	0.1
num_episodes	5000
Vittorie agente	3308 (66.16%)
Vittorie cpu	1692 (33.84%)



Di seguito i risultati ottenuti con i rewards della terza iterazione:

learning_rate	0.3
discount_factor	0.99
exploration_prob	0.1
num_episodes	5000
Vittorie agente	3266 (65.32%)
Vittorie cpu	1734 (34.68%)

learning_rate	0.3
discount_factor	0.95
exploration_prob	0.1
num_episodes	5000
Vittorie agente	3274 (65.48%)
Vittorie cpu	1726 (34.52%)



Differenze tra Q-Learning e Sarsa

Sia il Q-Learning che il SARSA hanno dimostrato risultati simili nel contesto dell'ottimizzazione delle prestazioni dell'agente. Tuttavia, durante l'analisi dei risultati, è emerso che il Q-Learning ha mostrato un miglioramento più significativo nei cambiamenti dei rewards e degli iperparametri. Ciò si è tradotto in un aumento della percentuale di vittorie dell'agente dal 60.59% al 66.16%. D'altra parte, utilizzando i medesimi valori ottimali nel SARSA, abbiamo riscontrato una percentuale di vittorie simile al 66.16%, mantenendo una stabilità intorno al 65% con ulteriori tentativi.
Questa osservazione suggerisce che, sebbene entrambi gli algoritmi abbiano prodotto risultati simili, il Q-Learning ha dimostrato un maggiore potenziale di ottimizzazione delle prestazioni attraverso l'adattamento degli iperparametri e l'ottimizzazione dei rewards.

SVILUPPI FUTURI E CONCLUSIONI

I risultati ottenuti mostrano come l'agente riesca ad ottenere una buona percentuale di vittorie rispetto alla CPU. La complessità del problema riscontrato però non permette di ottenere risultati progressivamente migliori. Di conseguenza, tra gli sviluppi futuri si potrebbe pensare di impiegare modelli più complessi come il Deep Q-Learning combinato con l'implementazione di altre regole di Ludo non considerate nello sviluppo del progetto.