Figure 1: Università degli Studi di Salerno

PongAI: Implementazione del Pong con il Q-Learning

Autore: Emanuele Luigi Amore
Matricola: 05121/16309

https://github.com/NumberZeroo/PongAI

Sommario

1 Introduzione

Il progetto si concentra sull'implementazione di un sistema di apprendimento per il gioco di Pong utilizzando l'algoritmo di Q-Learning. L'obiettivo principale è addestrare due agenti autonomi insieme in grado di competere tra loro ottimizzando le proprie strategie di gioco tramite la massimizzazione delle ricompense cumulative.

L'ambiente di gioco è stato progettato ex novo utilizzando Pygame, con un'interfaccia compatibile con OpenAI Gym per facilitare l'integrazione degli algoritmi di apprendimento. Inoltre, sono stati sviluppati strumenti di visualizzazione per analizzare l'andamento delle performance degli agenti, fornendo una chiara panoramica dei risultati ottenuti durante le fasi di addestramento e testing.

Questo progetto non solo vuole fornire un'implementazione funzionale del gioco di Pong con intelligenza artificiale, ma rappresenta anche una base solida per approfondire i concetti teorici e pratici dell'apprendimento con rinforzo.

1.1 Tool utilizzati

1. **Linguaggio**: Python
2. **Librerie**:
   (a) Pygame: Creazione dell'ambiente di gioco e gestione del rendering.
   (b) NumPy: Gestione di array e calcoli numerici per l'algoritmo Q-Learning.
   (c) Matplotlib: Visualizzazione dei risultati tramite grafici.
   (d) Pickle: Salvataggio e caricamento delle Table.
3. **Struttura modulare**: Codice organizzato in moduli distinti per facilitare la leggibilità e la manutenzione. I moduli principali includono:
   - Ambiente di simulazione (Pong).
   - Algoritmo di apprendimento (Q-Learning).
   - Funzioni per l'analisi dei dati e la creazione di grafici.

2 Definizione del problema

2.1 Obiettivo

L'obiettivo principale del progetto è addestrare due agenti autonomi che competano in un ambiente simulato, imparando a massimizzare le ricompense attraverso un processo iterativo di esplorazione e sfruttamento. Questo permette di dimostrare come l'apprendimento con rinforzo possa essere applicato in scenari competitivi.

2.2 Specifica P.E.A.S.

- **Performance**: Numero di vittorie, ricompense totali accumulate e numero di tocchi consecutivi in un singolo episodio. Verranno analizzati questi aspetti attraverso grafici e statistiche raccolti durante l'addestramento e il testing.
- **Environment**: Simulazione del gioco Pong tramite l'ambiente di Pygame.

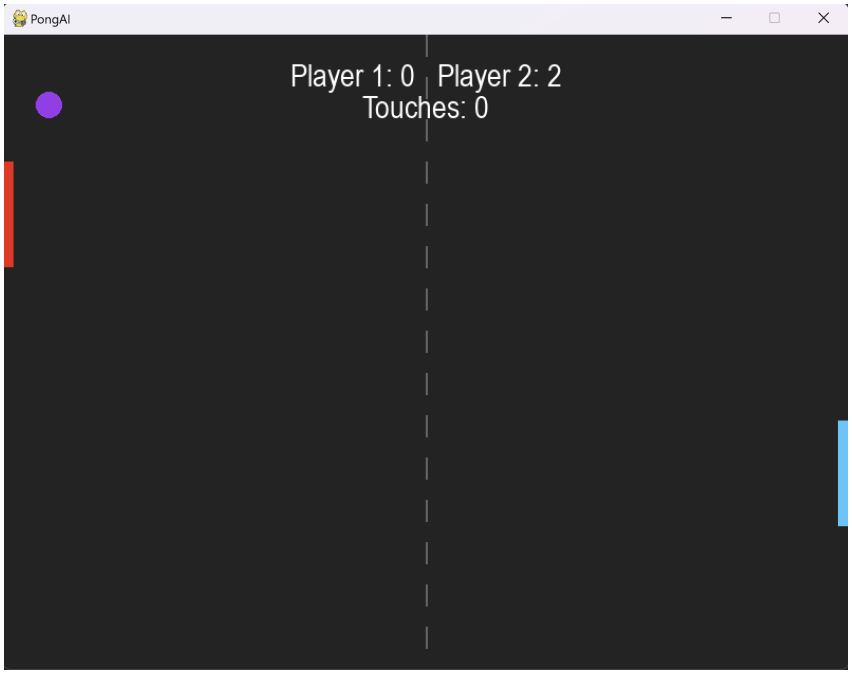

Figure 2: Schermata di esempio di una partita simulata.

- **Actuators**: Movimenti dei paddle (su, giù, fermo).
- **Sensors**: Osservazioni dell'ambiente, come posizione della palla, velocità e paddle.

2.3 Caratteristiche dell'ambiente

L'ambiente di simulazione utilizzato per il progetto è stato sviluppato da zero utilizzando la libreria **Pygame**, con una struttura compatibile con l'API di OpenAI Gym. Di seguito vengono descritte le principali caratteristiche:

- **Multi-agente**: L'ambiente include due agenti, Player 1 e Player 2, che competono tra loro controllando i paddle per massimizzare le proprie ricompense. Ogni agente è indipendente e prende decisioni basate sul proprio stato osservato.

- **Completamente osservabile**: L'ambiente è completamente osservabile per ciascun agente, poiché ogni paddle può osservare tutte le informazioni, come la posizione e la velocità della palla, oltre alla propria posizione e a quella dell'altro paddle.
- **Dinamico**: L'ambiente è dinamico, poiché la palla si muove continuamente e le interazioni tra palla e paddle aggiornano costantemente lo stato. Gli agenti devono prendere decisioni rapide per adattarsi ai cambiamenti dell'ambiente.
- **Discreto**: Sebbene l'ambiente sia fisicamente continuo, lo spazio degli stati è stato discretizzato in un numero finito di intervalli per permettere l'uso di una Q-Table. Le azioni degli agenti sono discrete (fermo, muovi in alto, muovi in basso).
- **Sequenziale**: Le decisioni prese dagli agenti influenzano direttamente gli stati successivi. Ad esempio, un colpo ben calibrato cambia la traiettoria della palla, influenzando le opportunità dell'avversario.
- **Deterministico**: L'ambiente è deterministico dato uno stato iniziale e una sequenza di azioni identiche, ma alcune condizioni iniziali (ad esempio, direzione e velocità della palla) sono stocastiche.
- **Competitivo**: L'ambiente è competitivo, in quanto i due agenti hanno obiettivi opposti: massimizzare il proprio punteggio minimizzando quello dell'avversario.

2.3.1 PongAI vs Pong Arcade

Per comprendere le differenze tra il gioco originale e l'ambiente sviluppato in questo progetto, è doveroso un confronto tra **Pong classico** e **PongAI**:

Proprietà	**Pong Arcade**	**PongAI**
Giocatori	Umano vs CPU	AI vs AI
Paddle	Pre-programmato	Strategie adattive
Ricompense	Nessuna logica	Tocchi, angoli e penalità per goal
Adattabilità	Comportamento fisso e prevedibile	Gli agenti si adattano

Table 1: Confronto tra Pong classico e PongAI

2.3.2 Principali differenze e sfide

Il **Pong classico** è basato su un comportamento *deterministico*, in cui la CPU segue regole pre-programmate senza capacità di apprendimento. Qui, invece, gli agenti imparano nel tempo tramite un approccio **trial & error**, sviluppando comportamenti adattivi e ottimizzando le proprie strategie.

Una delle principali sfide affrontate è stata la **definizione di un sistema di reward efficace**, capace di incentivare comportamenti ottimali senza far cadere l'agente in strategie non desiderate (come rimanere fermo per evitare penalità).

2.4 Gestione della prospettiva degli agenti

Uno degli aspetti critici dell'addestramento riguarda la loro percezione dell'ambiente. Per garantire un training equo e simmetrico tra i due giocatori, è stato necessario **specchiare la visione del Player 2** rispetto al Player 1.

Nel gioco originale, il Player 1 osserva la palla muoversi da sinistra a destra, mentre il Player 2 la vede muoversi nella direzione opposta. Se entrambi gli agenti fossero stati addestrati senza considerare questa differenza, avrebbero potuto apprendere strategie sbilanciate. Per evitare questo problema, ogni osservazione dell'ambiente per il Player 2 è stata trasformata come segue:

- La coordinata x della palla viene invertita rispetto all'asse verticale del campo.
- La velocità della palla lungo l'asse x viene negata per mantenere coerenza nel movimento.
- La posizione dei paddle viene scambiata, in modo che ogni agente "creda" di essere sempre sul lato sinistro.

Questa tecnica garantisce che Player 2 abbia un'esperienza di gioco identica a quella del Player 1, prevenendo bias nell'addestramento.

2.5 Analisi e definizione del Problema

Dato l'obiettivo del progetto di simulare il gioco di Pong e addestrare due agenti autonomi in grado di competere, possiamo analizzare il problema sotto i seguenti aspetti:

2.5.1 Sfide principali

- **Esplorazione ed esploitazione**: Gli agenti devono bilanciare la ricerca di nuove strategie (esplorazione) con lo sfruttamento di quelle già apprese per massimizzare le ricompense a lungo termine.
- **Ambiente competitivo**: Ogni agente deve ottimizzare le proprie azioni considerando le strategie dell'avversario.

2.5.2 Aspetti caratteristici

- **Azioni**: Ciascun agente può scegliere tra tre possibili azioni: restare fermo, muovere il paddle verso l'alto o verso il basso.
- **Osservazioni**: Ogni agente osserva le informazioni rilevanti al suo comportamento, come la posizione della palla e dei paddle, oltre alla velocità della palla.
- **Ricompense**: Le ricompense vengono assegnate per incentivare comportamenti utili, ma approfondiremo bene il discorso nella sezione sulla Metodologia.

3 Metodologia

Per permettere al gioco di "giocare" da solo, possiamo utilizzare diversi approcci. La scelta più semplice sarebbe sicuramente utilizzare un algoritmo ad hoc, per poi spaziare verso approcci più avanzati come il Deep Q-Learning, le reti neurali, e infine il Q-Learning, che è stato il metodo scelto per questo progetto.

3.1 Algoritmo ad hoc

Un algoritmo ad hoc si basa su euristiche predefinite. Ad esempio, il paddle può muoversi semplicemente verso la posizione della palla lungo l'asse verticale, ignorando altre informazioni come velocità o traiettoria. Questo tipo di approccio non prevede un apprendimento, ma è sufficiente per comportamenti di base.

Vantaggi	**Svantaggi**
Facile da implementare	Mancanza di strategia avanzata
Bassa complessità computazionale	Prestazioni limitate

Table 2: Vantaggi e svantaggi di un algoritmo ad hoc.

3.2 Deep Q-Learning (DQL)

Il Deep Q-Learning utilizza una rete neurale profonda per approssimare la funzione Q. Rispetto al Q-Learning tradizionale, questo approccio è utile quando lo spazio degli stati è molto grande o continuo. La rete neurale apprende una rappresentazione compatta degli stati e delle azioni, permettendo di scalare verso problemi più complessi.

Vantaggi	**Svantaggi**
Scalabile a spazi degli stati continui	Maggiore complessità computazionale
Capacità di generalizzazione	Richiede molte risorse per l'addestramento

Table 3: Vantaggi e svantaggi del Deep Q-Learning.

3.3 Reti Neurali

Le reti neurali possono essere utilizzate per predire direttamente le azioni ottimali (ad esempio, su, giù, fermo) dato uno stato osservato. L'addestramento può avvenire utilizzando dataset di partite precedenti o tramite apprendimento supervisionato. Tuttavia, le reti neurali non sono ideali per apprendere strategie adattive senza un'integrazione con tecniche di rinforzo.

Vantaggi	Svantaggi
Adatto per problemi complessi e non lineari	Dipendenza dalla qualità dei dati
Capacità di generalizzazione	Difficoltà ad adattarsi dinamicamente

Table 4: Vantaggi e svantaggi delle reti neurali nel contesto del Pong.

3.4 Discretizzazione degli Stati

Nel contesto del Q-Learning, uno dei passi fondamentali è la discretizzazione dello spazio degli stati. Questo consente di trasformare uno spazio continuo in un numero finito di stati, rendendo possibile l'uso di una Q-Table. La discretizzazione è stata implementata dividendo ogni dimensione dello stato in un numero finito di intervalli, come descritto nella funzione `discretize_state()` nel codice.

Vantaggi	Svantaggi
Riduce la complessità dello spazio degli stati	Perdita di precisione
Adatto per ambienti piccoli e medi	Non scalabile per spazi continui grandi

Table 5: Vantaggi e svantaggi della discretizzazione degli stati.

3.4.1 Discretizzazione, nel dettaglio

La discretizzazione dello spazio degli stati avviene suddividendo ogni dimensione dello stato (es. posizione della palla, velocità della palla, posizione dei paddle) in un numero finito di intervalli. Ad esempio, la posizione verticale della palla viene mappata in 10 intervalli discreti lungo l'asse y. Questa tecnica permette di rappresentare uno spazio continuo in una tabella Q finita e gestibile.
La funzione di discretizzazione implementata è la seguente:

```
def discretize_state(state, bins):
    :param state: Stato continuo dell'ambiente
    :param bins: Valori di divisione per ciascuna dimensione dello stato
    :return: Stato discretizzato (tuple)
    """
    player1_y, player2_y, ball_x, ball_y, ball_dx, ball_dy = state
    discrete_state = (
        np.digitize(player1_y, bins[0]) - 1,
        np.digitize(player2_y, bins[1]) - 1,
        np.digitize(ball_x, bins[2]) - 1,
        np.digitize(ball_y, bins[3]) - 1,
        np.digitize(ball_dx, bins[4]) - 1,
        np.digitize(ball_dy, bins[5]) - 1,
    )
    discrete_state = tuple(np.clip(discrete_state, 0, len(bins[0]) - 2))
    return discrete_state
```

3.5 Algoritmo Q-Learning

Dopo aver descritto vantaggi e svantaggi dei possibili approcci, l'algoritmo scelto per il progetto è il **Q-Learning**, un metodo basato su apprendimento per rinforzo che utilizza una Q-Table per memorizzare i valori delle azioni ottimali. Le principali componenti del Q-Learning sono:

- **Q-Table**: Una tabella che mappa ogni stato e azione al valore Q stimato, indicando la "qualità" di una determinata azione in uno stato specifico.
- **Stato attuale**: La configurazione osservata dell'ambiente (es., posizione della palla, posizione dei paddle).
- **Tasso di apprendimento** (α): Controlla quanto rapidamente l'agente aggiorna la propria conoscenza.
- **Fattore di sconto** (γ): Pondera l'importanza delle ricompense future rispetto a quelle immediate, anch'esso ha un valore che fluttua tra 0 e 1 dove 0 fa in modo che l'agente dia importanza solo a premi a "breve termine", mentre 1 il contrario.
- **Tasso di esplorazione** (ϵ): Determina la probabilità di esplorare nuove azioni rispetto a sfruttare quelle già note.
- **Ricompense** (R): Punteggio ottenuto dopo aver effettuato delle azioni, può assumere sia valore positivo che negativo. Nell'ambito del progetto, il reward potrà avere quattro valori:
  - **-5**: L'agente subisce un "goal".
  - **+1**: L'agente fa punto.
  - **+1**: L'agente tocca la palla evitando di subire "goal".
  - **+2**: Reward bonus se l'agente tocca la palla con uno degli spigoli (calcolati tramite l'altezza dei rect in Pygame) causando un "colpo ad effetto" velocizzando la palla e, probabilmente, facendo punto.
- **Q-Function**: La funzione di aggiornamento della Q-Table è definita come:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

  Dove s' è il nuovo stato risultante dall'azione a.

3.5.1 Strategia di esplorazione e decadimento di Epsilon

Uno degli aspetti fondamentali dell'addestramento con Q-Learning è la gestione del **decadimento di** ϵ, il parametro che controlla il bilanciamento tra esplorazione e sfruttamento.
Inizialmente, è stato implementato un decadimento **costante**, con ϵ che si riduceva ad ogni episodio seguendo la regola:

$$\epsilon = \epsilon \cdot 0.999 \tag{1}$$

Tuttavia, con questo approccio ϵ scendeva troppo rapidamente, raggiungendo il valore minimo dopo appena 10.000 episodi, impedendo agli agenti di esplorare strategie alternative nelle fasi avanzate del training.

Per migliorare la qualità dell'apprendimento, è stato adottato un **decadimento adattivo basato sul numero di episodi**, calcolato come:

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) \cdot \left(1 - \frac{\text{episode}}{\text{episodes}_{tot}}\right) \tag{2}$$

dove:

- $\epsilon_{max} = 1.0$, valore iniziale.
- $\epsilon_{min} = 0.05$, valore finale per garantire un minimo di esplorazione.
- episodes_{tot} è il numero totale di episodi del training.

Questo approccio garantisce che l'agente esplori più a lungo nelle prime fasi e riduca gradualmente l'esplorazione man mano che impara strategie più efficaci.

3.6 Implementazione dell'algoritmo

L'algoritmo Q-Learning è stato implementato con una struttura modulare che utilizza una Q-Table per apprendere le migliori azioni. Di seguito è mostrata una porzione del codice che aggiorna la Q-Table:

```
max_future_q = np.max(q_table[next_state])
current_q = q_table[state, action]
new_q = current_q + alpha *
    (reward + gamma * max_future_q - current_q)
q_table[state, action] = new_q
```

3.7 Implementazione del reward

L'implementazione del reward è molto semplice, ma efficace. Come elencato in precedenza, abbiamo diversi tipi di reward che vengono assegnati agli agenti in base alle loro scelte. Di seguito è mostrata una porzione di codice:

```
# Ricompensa basata sul punto di impatto
if impact_point < 0.15 or impact_point > 0.85:
    reward_player2 += 2
else:
    reward_player2 += 1

# Paddle 2 ha toccato, penalità per Player 1
if self.paddle2_touched:
    self.score_player2 += 1
    reward_player1 -= 5
    reward_player2 += 1
else:  # Solo penalità per Player 1
    reward_player1 -= 5
```

Il codice mostrato riguarda il player2, ma ovviamente è identico a quello inerente al player1. Dopo aver definito una variabile **impactpoint**, controlliamo se questo indica o meno che la palla ha colpito uno degli spigoli del paddle. Se la palla ha colpito uno degli spigoli, allora assegnamo +2 al player, altrimenti +1.

Il "goal" invece viene interpretato in modo tale da essere assegnato solo se il player ha effettivamente toccato prima la palla (e quindi causato il goal), altrimenti non viene assegnato nessun reward al player, ma solo una penalità a chi ha subito il punto perché, giustamente, non ha parato la palla.

3.8 Metodologia, conclusioni

La scelta dell'algoritmo di Q-Learning si è rivelata efficace per affrontare il problema di un ambiente competitivo e deterministico come Pong. Attraverso l'uso della Q-Table e una strategia di esplorazione bilanciata, gli agenti sono stati in grado di apprendere comportamenti adattivi e ottimizzati, dimostrando la potenza dell'apprendimento per rinforzo in contesti simulativi. I risultati ottenuti verranno discussi nella prossima sezione.

4 Risultati

4.1 Risultati desiderati

L'obiettivo principale era dimostrare che i due agenti possano apprendere strategie di gioco competitive attraverso il Q-Learning. I risultati desiderati includono:

- Miglioramento progressivo delle ricompense medie per ciascun agente durante il training.
- Riduzione delle penalità ricevute dagli agenti (ad esempio, subire meno goal).
- Aumento del numero di tocchi consecutivi, segnale di scambi più complessi e competitivi.
- Confronto significativo tra agenti addestrati e non addestrati durante il testing.

4.2 Metriche di valutazione delle performance

Per analizzare le performance degli agenti, utilizziamo le seguenti metriche:

- **Ricompensa media per episodio**: Misura il valore medio della ricompensa ottenuta durante ogni episodio.
- **Percentuale di vittorie**: Calcola la proporzione di episodi vinti da ciascun agente.
- **Numero medio di scambi**: Rappresenta il livello di interazione tra gli agenti durante il gioco.

4.3 Training

Il training degli agenti è stato condotto per 120.000 episodi con i seguenti parametri:

- **Tasso di apprendimento** (α): 0.1
- **Fattore di sconto** (γ): 0.99
- **Tasso di esplorazione iniziale** (ϵ): 1.0, con decadimento verso 0.05

4.3.1 Configurazione dei due agenti

I due agenti sono stati configurati con una politica di esplorazione iniziale alta, che consente loro di esplorare lo spazio delle azioni. Durante il training, il tasso di esplorazione è stato gradualmente ridotto per favorire lo sfruttamento delle strategie apprese.

4.3.2 Comportamento degli agenti

All'inizio del training, gli agenti mostrano comportamenti casuali. Con il progredire degli episodi, iniziano a sviluppare strategie che massimizzano le ricompense, come il posizionamento ottimale dei paddle per intercettare la palla e l'esecuzione di colpi ad effetto.

4.4 Risultati ottenuti

I risultati del training sono stati analizzati durante l'addestramento:

- **Ricompensa media per episodio**:

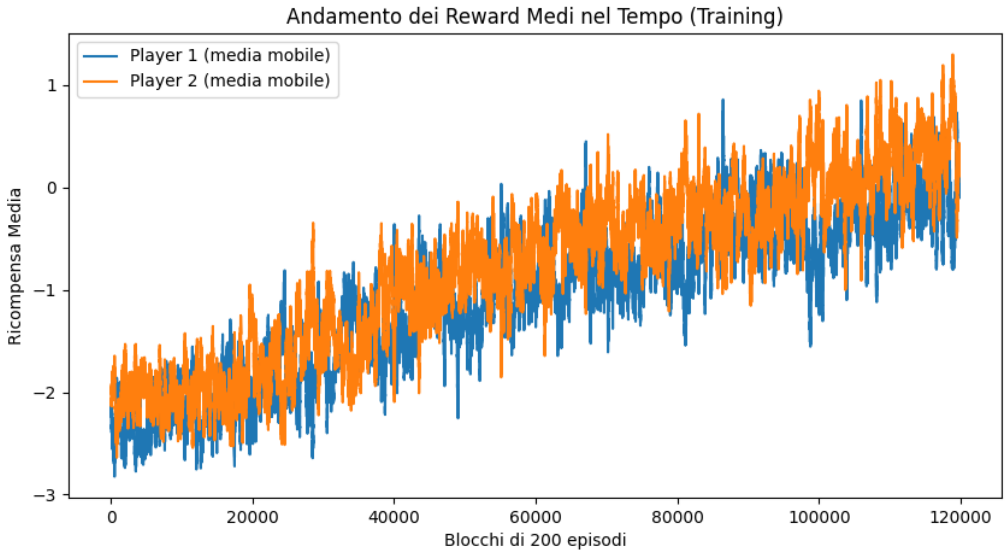

Figure 3: Andamento della ricompensa media per episodio durante il training.

- **Numero di scambi medi**:

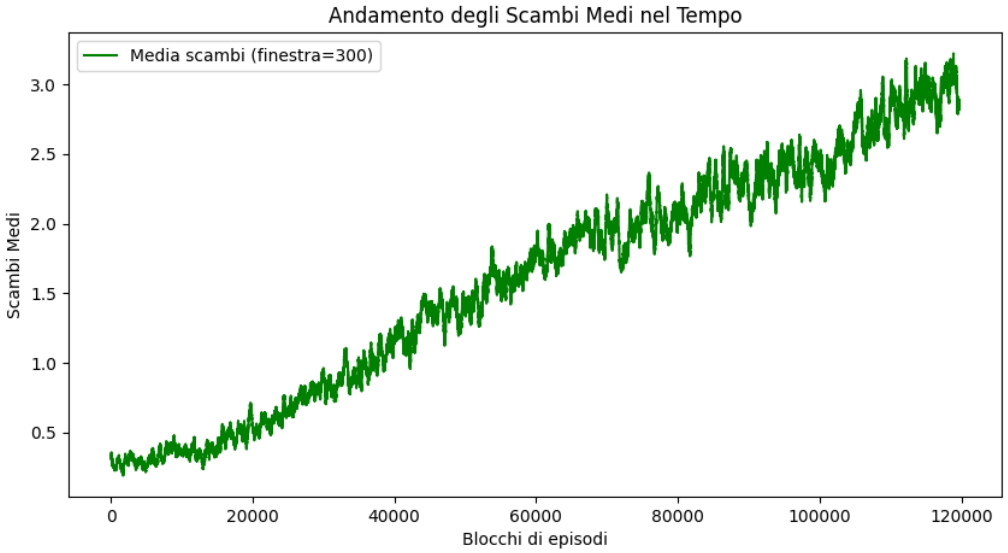

Figure 4: Scambi medi.

- **Percentuale vittorie totali**:

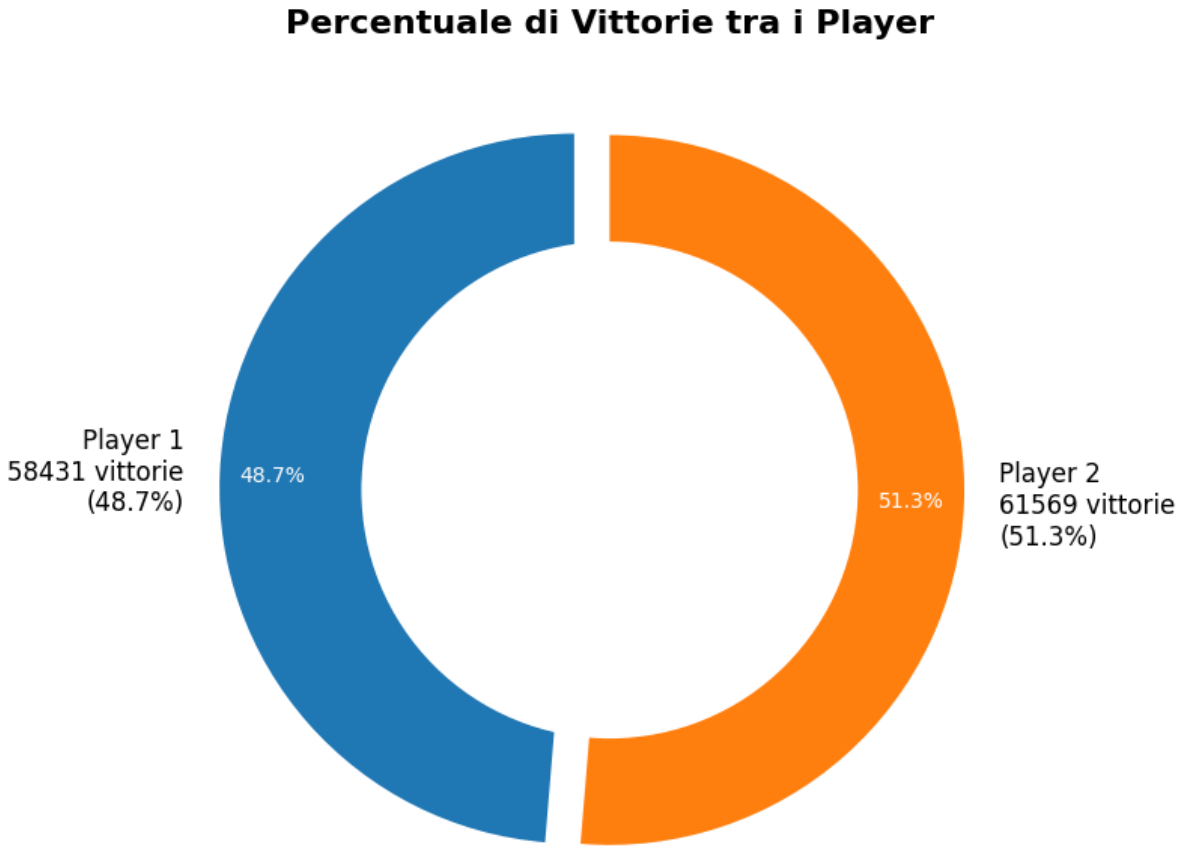

Figure 5: Percentuali vittorie.

Il training ha soddisfatto le aspettative, mostrando un miglioramento progressivo delle performance degli agenti. In particolare:

- La ricompensa media è aumentata costantemente durante gli episodi, segno che gli agenti hanno imparato a massimizzare il reward ottimizzando le loro strategie.
- Il numero medio di scambi è cresciuto nel tempo, suggerendo che gli agenti sono diventati più abili nel mantenere il controllo della palla.
- La percentuale di vittorie si è stabilizzata, indicando che entrambi gli agenti hanno sviluppato strategie competitive.

4.5 Testing

Dopo la fase di training, gli agenti sono stati testati in 10 e 20.000 episodi per valutare la loro effettiva capacità di gioco. I risultati ottenuti mostrano alcune particolarità interessanti:

4.5.1 Comportamento Stocastico degli Agenti

Durante il testing, si è osservato un fenomeno interessante: **in alcune sessioni di testing è il Player 1 a prevalere, mentre in altre è il Player 2**. Questo comportamento è spiegabile attraverso i seguenti fattori:

- La casualità nello **spawn iniziale della palla** può fornire un leggero vantaggio iniziale a uno dei due agenti.
- Piccole differenze nei valori della **Q-Table** possono portare un agente a sviluppare una strategia dominante in un particolare test.
- Il processo di apprendimento competitivo può portare uno degli agenti a trovare strategie più efficaci rispetto all'altro, in modo del tutto casuale.

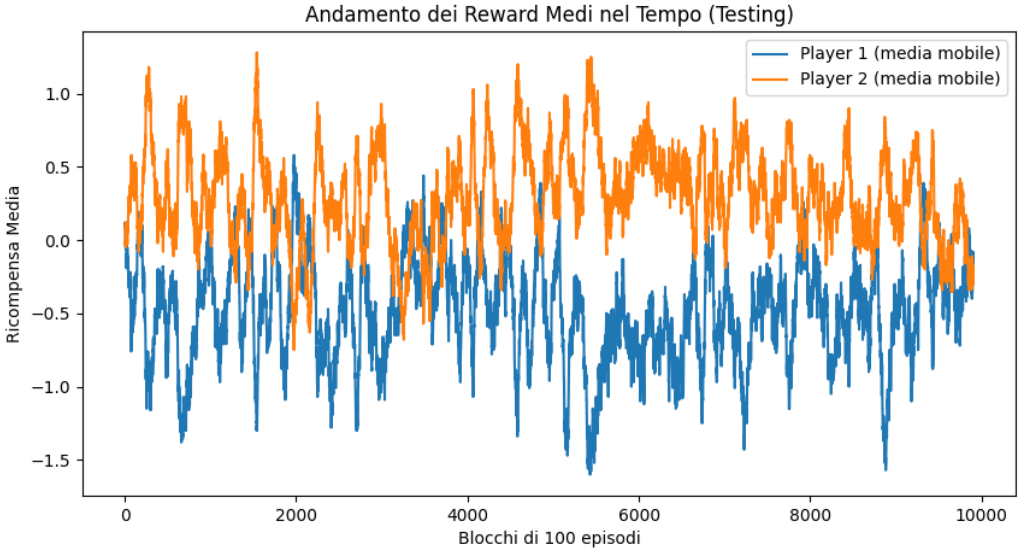

Figure 6: Esempio di distribuzione delle vittorie tra Player 1 e Player 2 durante il testing.

4.5.2 Confronto tra Modello Trainato e Non Trainato

Per valutare l'efficacia del Q-Learning, è stato effettuato un confronto tra il modello trainato e un modello non addestrato (che prende decisioni casuali). I risultati evidenziano una chiara differenza di performance:

- Il modello addestrato riesce a mantenere la palla in gioco per un numero significativamente maggiore di tocchi.
- Il reward medio per episodio del modello trainato è molto più alto rispetto al modello non trainato.
- Il modello non trainato mostra un comportamento erratico, mentre il modello addestrato esegue movimenti più precisi e strategici.

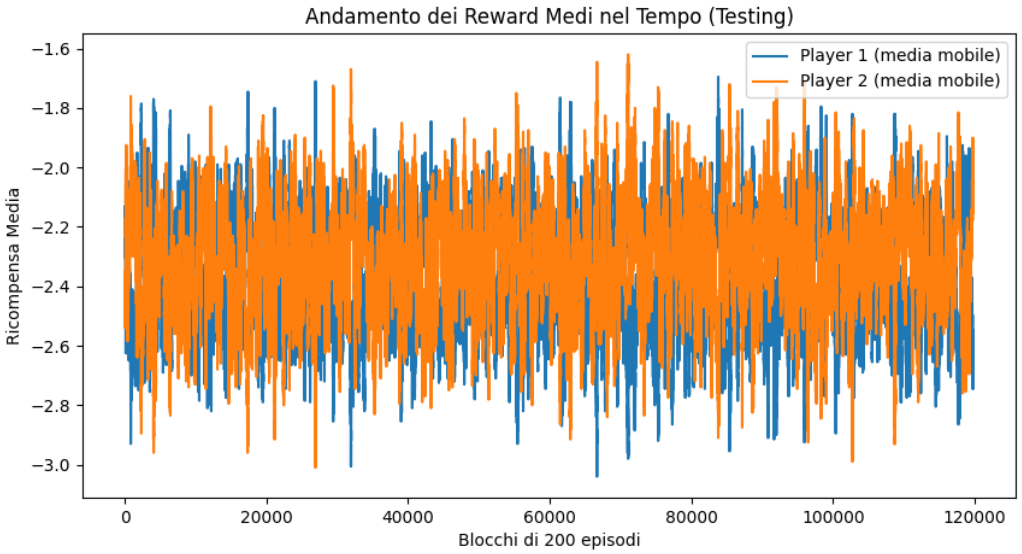

Figure 7: AvgReward di un modello non trainato.

I risultati confermano l'efficacia dell'algoritmo di Q-Learning, dimostrando che gli agenti hanno sviluppato strategie in grado di massimizzare le ricompense e migliorare la loro competitività.

4.6 Conclusione sui Risultati

In sintesi:

- Il training ha portato a un miglioramento evidente delle capacità degli agenti, con un aumento del reward medio e della stabilità delle performance.
- Il testing ha rivelato una natura stocastica nell'apprendimento, dove a seconda della sessione uno dei due agenti tende a prevalere.
- Il confronto con un modello non trainato dimostra in modo chiaro l'efficacia del Q-Learning nel migliorare le performance di gioco.
- Sono state testate diverse configurazioni dei parametri, come:
  - **Tasso di apprendimento** (α): sono stati provati valori come 0.5, ma hanno mostrato instabilità rispetto a 0.1.
  - **Fattore di sconto** (γ): l'uso di valori vicini a 1.0 ha permesso di massimizzare le ricompense a lungo termine.
  - **Decadimento di epsilon**: valori più lenti di decadimento hanno portato a una migliore esplorazione nelle fasi iniziali.

Questi esperimenti hanno evidenziato che un valore $\alpha = 0.1$ e $\gamma = 0.99$ forniscono un buon bilanciamento tra stabilità e apprendimento efficace. Tuttavia, potrebbero essere esplorate ulteriori configurazioni per migliorare l'adattabilità degli agenti.

5 Conclusioni

L'obiettivo del progetto era creare un'intelligenza artificiale capace di giocare autonomamente a Pong, utilizzando il Q-Learning per apprendere strategie efficaci. Dopo un addestramento di 120.000 episodi, possiamo trarre le seguenti considerazioni:

- Il modello ha dimostrato la capacità di **apprendere strategie competitive**, migliorando il numero di tocchi e il reward medio episodio dopo episodio.
- L'**andamento delle vittorie** ha mostrato una natura stocastica: in base alle condizioni iniziali e all'addestramento, un agente può sviluppare una strategia dominante sull'altro.
- Il confronto con un modello **non addestrato** evidenzia chiaramente l'efficacia del Q-Learning: un agente allenato prende decisioni migliori, mantenendo la palla più a lungo e massimizzando il reward.

5.1 Possibili miglioramenti

Nonostante i buoni risultati, ci sono diversi aspetti che potrebbero essere migliorati:

- **Double Q-Learning**: potrebbe ridurre il bias nell'aggiornamento dei valori Q, migliorando l'affidabilità della strategia appresa.
- **Utilizzo di una rete neurale (DQN)**: sostituire la Q-Table con un modello più scalabile, capace di gestire stati continui e situazioni più complesse.
- **Diversi approcci di training**: allenare gli agenti separatamente prima di testarli insieme, per verificare se emergono strategie più avanzate.

5.2 Conclusione finale

In conclusione, PongAI ha dimostrato come il Reinforcement Learning possa essere applicato anche in un ambiente competitivo, con agenti che imparano strategie vincenti in modo autonomo. Il modello attuale rappresenta una base solida, ma ulteriori miglioramenti potrebbero renderlo ancora più efficace e adattabile.