



# Ipomoea Aquatica

*Un agente intelligente per Space Invaders.*

LUIGINA COSTANTE

ANGELO NAZZARO



Università degli Studi di Salerno

Anno Accademico 2024/2025

---

# INDICE

<b>1 Introduzione</b>	<b>3</b>
Cos'è Ipomoea Aquatica?.....	3
Space Invaders.....	3
1.2.1 Obiettivi .....	3
1.2.2 Comandi .....	4
1.2.3 Varianti di gioco .....	4
1.2.4 Punteggio .....	4
<b>2 Definizione del Problema</b>	<b>6</b>
Obiettivi.....	6
<b>3 Metodologia</b>	<b>7</b>
Ambiente .....	7
3.1.1 Stato dell'ambiente .....	7
3.1.2 Azioni .....	8
3.1.3 Dinamiche e ricompense .....	9
Agenti .....	10
3.2.1 Q-Learning .....	10
3.2.2 Deep Q-Network (DQN) .....	11
3.2.3 Asynchronous Advantage Actor-Critic (A3C) .....	15
<b>4 Risultati e Discussioni</b>	<b>19</b>
Setup Sperimentale .....	19
Metriche di Valutazione.....	20
4.2.1 Metriche comuni a tutti i modelli .....	20
4.2.2 Metriche Specifiche per DQN .....	21
4.2.3 Metriche Specifiche per A3C .....	21
Risultati .....	22
4.3.1 Q-Learning .....	22
4.3.2 DQN .....	26
4.3.3 A3C .....	31
<b>5 Conclusioni</b>	<b>39</b>

Cos'è Ipomoea Aquatica? .....	3	1.2.2 Comandi . . . . .	4
Space Invaders .....	3	1.2.3 Varianti di gioco . . . .	4
1.2.1 Obiettivi .....	3	1.2.4 Punteggio . . . . .	4

CAPITOLO N°

1

## INTRODUZIONE

### 1.1 Cos'è Ipomoea Aquatica?

Ipomea Aquatica è un agente intelligente progettato per giocare a Space Invaders, sviluppato attraverso l'applicazione avanzata di tecniche di apprendimento per rinforzo (RL). Il progetto si basa sull'implementazione e il confronto di tre algoritmi distinti: *Q-Learning* [watkins1992:q-learning], *Deep Q-Network (DQN)* [mnih2015:dqn] e *Advantage Actor-Critic (A3C)* [mnih2016:a3c], ciascuno addestrato per ottimizzare le proprie performance nell'ambiente del gioco. L'architettura è stata costruita con un approccio modulare e supporta funzionalità come la registrazione video delle sessioni di gioco e il logging delle metriche tramite *Weights & Biases* (wandb) [biweald2020:wandb]. L'obiettivo è stato quello di esplorare le capacità di generalizzazione, adattamento e strategia di ciascun agente in un ambiente complesso e dinamico.

### 1.2 Space Invaders

Space Invaders è un classico gioco arcade in cui il giocatore controlla un "cannone laser" mobile per difendere la Terra da un'invasione aliena. L'obiettivo principale è abbattere le navicelle degli invasori prima che raggiungano la Terra (fondo dello schermo) o prima che colpiscono il giocatore tre volte con le loro "bombe laser".

#### 1.2.1 Obiettivi

Gli invasori devono essere eliminati prima che:

- Raggiungano il fondo dello schermo, completando così l'invasione.
- Colpiscono il giocatore per tre volte, causando la fine della partita.

L'obiettivo a lungo termine del gioco è segnare il maggior numero possibile di punti. Il punteggio viene incrementato ogni volta che si distrugge un invasore, e il valore del punteggio dipende dalla posizione iniziale dell'invasore sullo schermo.

### 1.2.2 Comandi

I comandi disponibili per il giocatore sono illustrati nella Tabella 1.1 seguente.

Numero	Azione	Descrizione
0	NOOP	Nessuna azione, l'agente rimane fermo
1	FIRE	Spara un colpo
2	RIGHT	Sposta il cannone a destra
3	LEFT	Sposta il cannone a sinistra
4	RIGHTFIRE	Sposta il cannone a destra e spara simultaneamente
5	LEFTFIRE	Sposta il cannone a sinistra e spara simultaneamente

Tabella 1.1: Comandi disponibili in Space Invaders.

### 1.2.3 Varianti di gioco

Space Invaders include diverse varianti che aumentano la complessità e la sfida per il giocatore:

- **Scudi mobili:** gli scudi protettivi si muovono avanti e indietro sullo schermo, aumentando il rischio durante il loro utilizzo.
- **Bombe laser a zigzag:** le bombe laser seguono traiettorie irregolari, rendendo più difficile prevedere dove cadranno.
- **Bombe laser veloci:** le bombe scendono rapidamente, richiedendo riflessi pronti per evitarle.
- **Invasori invisibili:** gli invasori diventano invisibili dopo l'inizio del gioco. Appaiono solo brevemente ogni volta che uno di essi viene colpito.

Nel nostro studio, consideriamo la variante base del gioco.

### 1.2.4 Punteggio

Ogni invasore distrutto assegna un punteggio che varia in base alla sua riga di partenza. In Tabella 1.2 sono illustrati i valori dei punteggi per ciascun tipo di nemico. In aggiunta ai nemici mostrati in Tabella 1.2, periodicamente, apparirà una "Nave del Mistero", inerme e molto veloce, che vale 200 punti ogni volta che viene abbattuta.

Nemico	Riga	Punteggio
	1	5
	2	10
	3	15
	4	20
	5	25
	6	30

Tabella 1.2: Tipi di nemici e relativi punteggi.

## DEFINIZIONE DEL PROBLEMA

### 2.1 Obiettivi

Il progetto mira a sviluppare e implementare un sistema di RL con l'obiettivo di creare un agente in grado di interagire con l'ambiente di gioco e di apprendere strategie avanzate per massimizzare il proprio punteggio. Gli obiettivi specifici includono:

- *Definizione degli stati del gioco*: identificare gli stati rilevanti del gioco, considerando informazioni come la posizione e il movimento dei nemici, lo stato e la posizione del giocatore, e altri elementi critici dell'ambiente di gioco.
- *Azioni tattiche avanzate*: definire un set di azioni ottimizzate, inclusi movimenti precisi, tattiche di tiro efficace e reazioni rapide ai cambiamenti dinamici dell'ambiente.
- *Apprendimento di strategie avanzate di gioco*: progettare un agente intelligente capace di sviluppare strategie che consentano di massimizzare il punteggio e migliorare la capacità di sopravvivenza nel gioco.
- *Implementazione degli agenti*: sviluppare agenti basati su tre diverse tecniche di apprendimento per rinforzo, sfruttando algoritmi avanzati come QLearning [[watkins1992:q-learning](#)], DQN [[mnih2015:dqn](#)] e A3C [[mnih2016:a3c](#)] per migliorare progressivamente le prestazioni.
- *Addestramento iterativo e fine-tuning*: condurre sessioni iterative di addestramento e tuning degli iperparametri per ottimizzare le strategie dell'agente, con un'attenzione particolare all'esplorazione di scenari complessi.
- *Valutazione delle prestazioni*: misurare l'efficacia dell'agente attraverso metriche tra cui il punteggio totale ottenuto, la reward cumulativa, la variazione dei Q-values, e altre misure che saranno descritte in dettaglio nelle sezioni successive.

Questo progetto si propone, inoltre, di confrontare le performance dei diversi algoritmi di RL implementati, di analizzare i punti di forza e le limitazioni di ciascun approccio, e di documentare le sfide affrontate durante lo sviluppo, insieme alle soluzioni adottate.

# 3

CAPITOLO N°

Ambiente .....	7	Agenti .....	10
3.1.1 Stato dell'ambiente . . . . .	7	3.2.1 Q-Learning . . . . .	10
3.1.2 Azioni . . . . .	8	3.2.2 Deep Q-Network (DQN)	11
3.1.3 Dinamiche e ricompense	9	3.2.3 Asynchronous Advantage Actor-Critic (A3C) . . . . .	15

## METODOLOGIA

### 3.1 Ambiente

L'ambiente SpaceInvaders-v5, fornito dalla libreria OpenAI Gym [[openaigym](#)], si basa sul classico gioco arcade *Space Invaders* e appartiene alla categoria degli ambienti *Atari*. Quest'ultimo è caratterizzato da una rappresentazione a stati discreti, un insieme finito di azioni, e un ambiente dinamico. Di seguito, viene fornita una descrizione dettagliata dei componenti principali dell'ambiente: stati, azioni e dinamiche.

#### 3.1.1 Stato dell'ambiente

Lo stato dell'ambiente (*observation*) è rappresentato da una matrice tridimensionale  $S \in \mathbb{R}^{210 \times 160 \times 3}$ , dove  $210 \times 160$  sono le dimensioni del frame video e 3 rappresenta i canali di colore RGB. Ogni stato corrisponde a un frame del gioco, che include la posizione del giocatore (*cannon*), dei nemici (*invaders*), dei proiettili in volo, e degli eventuali ostacoli parzialmente distrutti. Nella Figura 3.1 è mostrato un esempio di stato, che rappresenta un frame tipico dell'ambiente di gioco.

##### 3.1.1.1 Observation Wrapper

La matrice appena descritta cattura l'intero contenuto visivo dell'ambiente, inclusi gli oggetti dinamici come il giocatore, i nemici, i proiettili e gli scudi. Tuttavia, per rendere il processo di apprendimento più efficace, il frame è stato arricchito con informazioni derivanti da un'analisi semantica dei principali elementi di gioco.

Il wrapper implementato consente di individuare la posizione precisa del giocatore (*cannon*), che viene utilizzata non solo per tracciare i suoi movimenti, ma anche per determinare la sua interazione con gli scudi e i proiettili. La posizione dei nemici (*invaders*) viene identificata attraverso una griglia, che rappresenta la disposizione spaziale degli invasori su una matrice  $6 \times 6$ . Questa matrice è calcolata a partire dalla *bounding box* che delimita l'area occupata dagli invasori nel frame corrente.

Un'attenzione particolare è dedicata agli scudi, che svolgono un ruolo significativo nel gameplay. Il wrapper analizza lo stato degli scudi confrontando le loro posizioni tra frame consecutivi, identificando eventuali danni subiti. Questo processo si basa sulla determinazione della *bounding box* degli scudi e sulla valutazione delle differenze pixel-per-pixel, permettendo di rilevare con precisione i colpi

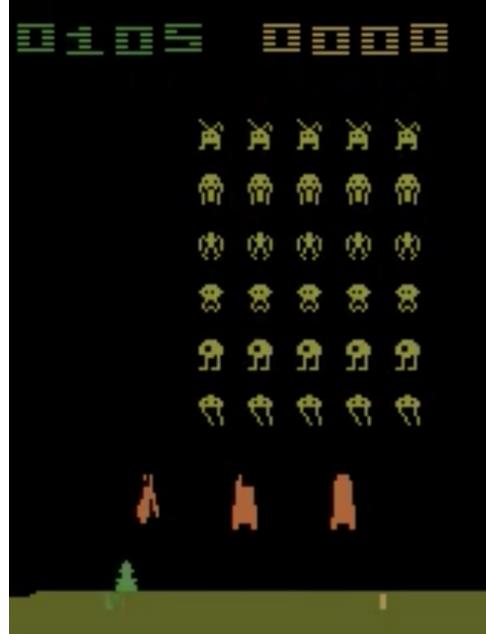


Figura 3.1: Esempio di stato dell’ambiente.

ricevuti. Inoltre, viene stabilita la fonte del danno, distinguendo tra proiettili sparati dal giocatore e quelli provenienti dai nemici.

Un altro elemento rilevante è la posizione della ‘Nave del Mistero’ (*mothership*), che viene rilevata nel frame corrente per permettere al giocatore di reagire alla sua apparizione. Lo stato include anche il numero di vite residue del giocatore, una variabile direttamente ricavata dall’API dell’ambiente ALE (Arcade Learning Environment), che fornisce un’indicazione chiara del progresso e della performance dell’agente.

Combinando queste informazioni, lo stato dell’ambiente non solo offre una rappresentazione visiva del gioco, ma include anche un set di dati strutturati che catturano gli eventi chiave e le interazioni dinamiche. Questo livello di dettaglio permette agli agenti di apprendere strategie avanzate in modo più efficiente, concentrandosi sulle componenti rilevanti dell’ambiente e ignorando i dettagli superflui.

### 3.1.2 Azioni

L’insieme delle azioni disponibili, denotato come  $A = \{0, 1, 2, 3, 4, 5\}$ , è costituito da sei opzioni discrete, ognuna delle quali rappresenta una scelta specifica che l’agente può intraprendere durante il gioco. Le azioni possibili sono le stesse illustrate in Tabella 1.1. Per facilitarne la consultazione, le riportiamo di seguito:

- **0 - NOOP:** nessuna azione, l’agente rimane fermo;
- **1 - FIRE:** l’agente spara un colpo verso i nemici;
- **2 - RIGHT:** l’agente si sposta verso sinistra;
- **3 - LEFT:** l’agente si sposta verso destra;
- **4 - RIGHTFIRE:** l’agente si sposta verso sinistra e spara simultaneamente;
- **5 - LEFTFIRE:** l’agente si sposta verso destra e spara simultaneamente.

Tali azioni sono selezionabili dall'agente in modo esclusivo, il quale deve decidere quale eseguire in ogni passo temporale. Poiché lo spazio delle azioni risulta finito e numerabile, l'agente è vincolato a scegliere tra un numero limitato di opzioni, ciascuna con effetti diretti sullo stato e sulla strategia del gioco.

### 3.1.2.1 Action Wrapper

Per gestire le azioni in modo efficiente, è stato implementato anche in questo caso un wrapper personalizzato. Quest'ultimo, tramite il metodo `step`, sovrascrive il comportamento standard dell'ambiente per notificare quando l'agente spara un colpo. L'azione di sparare è identificata dai tre ID specifici: FIRE (1), RIGHTFIRE (4) e LEFTFIRE (5). Quando una di queste azioni viene selezionata, il wrapper chiama il metodo `notify_agent_shot`, che segnala all'ambiente il verificarsi dell'evento.

Inoltre, il wrapper gestisce gli stati terminali del gioco. Quando l'ambiente rileva che la partita è terminata o interrotta (`terminated` o `truncated`), il metodo `step` si assicura che il punteggio dell'agente venga resettato, preparando così l'ambiente per una nuova sessione di gioco.

### 3.1.3 Dinamiche e ricompense

L'agente inizia l'episodio avendo a disposizione 3 vite. Durante il corso dell'episodio, la posizione dei nemici e dei loro proiettili. L'episodio termina in uno dei seguenti casi:

- Tutte le vite del giocatore sono esaurite a causa dell'impatto con i proiettili nemici o della collisione con i nemici stessi;
- I nemici raggiungono la superficie terrestre, segnando una condizione di sconfitta.

Le ricompense disponibili per l'agente sono le stesse illustrate in Tabella 1.2. Per facilitarne la consultazione, le riportiamo di seguito:

- **Distruzione nemico in riga 1:** 5;
- **Distruzione nemico in riga 2:** 10;
- **Distruzione nemico in riga 3:** 15;
- **Distruzione nemico in riga 4:** 20;
- **Distruzione nemico in riga 5:** 25;
- **Distruzione nemico in riga 6:** 30;
- **Distruzione "Nave del Mistero":** 200;

L'obiettivo dell'agente è quello di sopravvivere più a lungo, massimizzando il punteggio ottenuto.

### 3.1.3.1 Reward Wrapper

Il wrapper del sistema di ricompense fornisce un controllo adattivo sulle ricompense durante l'episodio, integrando diversi fattori che influenzano l'assegnazione delle ricompense. Ogni azione intrapresa dall'agente è valutata considerando:

- *La perdita di vite:* quando l'agente perde una vita, la ricompensa subisce una penalizzazione di -100, incoraggiando l'adozione di strategie difensive per evitare errori fatali.

- *I danni agli scudi:* se i proiettili sparati dal giocatore danneggiano gli scudi, viene applicata una penalizzazione aggiuntiva di  $-25$ , promuovendo una maggiore precisione nei colpi.
- *L'avanzamento dei nemici:* il wrapper calcola una penalità progressiva basata sull'avvicinamento degli invasori al giocatore, misurando la distanza tra la loro nuova posizione e quella del giocatore. Se gli invasori avanzano rispetto al frame precedente, la penalità aumenta proporzionalmente al numero di nemici rimanenti e alla riduzione della distanza, scoraggiando comportamenti passivi dell'agente.

Oltre a queste penalizzazioni, il wrapper offre la possibilità di normalizzare le ricompense utilizzando una tecnica di *min-max normalization*, che comprime il range delle ricompense tra  $[0, 1]$ . La scelta di normalizzare la reward è stata dettata da un'analisi della letteratura corrente, che ha rivelato un miglioramento della stabilità durante l'apprendimento con le reward normalizzate rispetto alle reward normali.

Con questo approccio si mira a considerare anche l'evoluzione dell'episodio oltre alla valutazione immediata delle azioni, fornendo feedback continui e adattivi per guidare l'agente verso il raggiungimento del suo obiettivo finale: sopravvivere il più a lungo possibile e ottenere il massimo punteggio. In questo modo, l'agente è incentivato a bilanciare comportamenti offensivi e difensivi, adattandosi alle condizioni del gioco.

## 3.2 Agenti

Nella sezione seguente presentiamo e approfondiamo i modelli impiegati. Nello specifico, abbiamo deciso di addestrare e confrontare tre tipologie diverse di agenti:

- **Metodo Tabulare: Q-Learning** [[watkins1992:q-learning](#)];
- **Metodo Approssimato - Off-Policy: DQN** [[mnih2015:dqn](#)];
- **Metodo Approssimato - Policy Gradient: A3C** [[mnih2016:a3c](#)];

### 3.2.1 Q-Learning

Q-Learning [[watkins1992:q-learning](#)] è un algoritmo *off-policy* progettato per approssimare la funzione azione-valore ottimale  $Q^*(S_t, A_t)$ , che rappresenta il massimo valore atteso di ricompensa cumulativa ottenibile partendo dallo stato  $S_t$  ed eseguendo l'azione  $A_t$ . L'algoritmo aggiorna iterativamente i valori  $Q(S_t, A_t)$  secondo la seguente regola di aggiornamento:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (3.1)$$

dove  $\alpha$  è il tasso di apprendimento (*learning rate*), che determina quanto velocemente il modello si adatta alle nuove informazioni;  $\gamma$  è il fattore di sconto (*discount factor*), che controlla il peso dato alle ricompense future;  $R_t$  è la ricompensa ricevuta dopo aver eseguito l'azione  $A_t$  nello stato  $S_t$ ;  $S_{t+1}$  è lo stato successivo risultante dall'esecuzione di  $A_t$  nello stato  $S_t$  e  $\max_a Q(S_{t+1}, a)$  rappresenta il massimo valore atteso per lo stato successivo, considerando tutte le possibili azioni  $a$ . Lo pseudocodice dell'algoritmo per l'implementazione del Q-Learning è presentato nell'Algoritmo 1.

## Q-Learning

```
Input :  $\alpha \in (0, 1]$ ,  $\epsilon > 0$  piccolo
Inizializza  $Q(s, a), \forall s \in S^+, a \in A(s)$  in maniera arbitraria, tranne che  $Q(\text{terminal}, \cdot) = 0$ ;
for ogni episodio do
    Inizializza  $S$ ;
    while l'episodio non è terminato do
        Scegli  $A_t$  da  $S_t$  usando la policy derivata da  $Q$ ;
        Esegui  $A_t$ , osserva  $R_t, S_t$  e  $S_{t+1}$ ;
        Aggiorna  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$ ;
         $S_t \leftarrow S_{t+1}$ ;
    end
end
```

### 3.2.1.1 Policy

La policy seguita dalla nostra implementazione è una  $\epsilon$ -greedy policy. Riferisi a 4.1 per maggior dettagli sulla scelta di  $\epsilon$ .

### 3.2.1.2 Motivazioni

Sebbene Q-Learning sia un metodo tabulare, in genere applicato ad ambienti con un insieme degli spazi sufficientemente piccolo da poter essere memorizzato in una tabella, abbiamo deciso di considerarlo in quanto la sua semplicità di implementazione e l'enorme ammontare di studi dietro, lo rendono una solida base di partenza per confrontare metodi più avanzati e complessi. Inoltre, abbiamo ritenuto interessante esplorare le prestazioni di un algoritmo tabulare in un contesto con uno spazio di stato significativamente ampio, al fine di valutarne i limiti pratici e le capacità effettive.

## 3.2.2 Deep Q-Network (DQN)

aggiungere early stopping

La Deep Q-Network (DQN) rappresenta un progresso significativo nell'apprendimento per rinforzo per ambienti ad alta dimensionalità, come i videogiochi Atari. Basandosi sul lavoro di Mnih et al. [mnih2015:dqn], il modello sviluppato combina l'uso di reti neurali convoluzionali (CNN) per l'estrazione di caratteristiche visive con un meccanismo di apprendimento Q-learning per stimare la funzione  $Q$ .

### 3.2.2.1 Architettura del modello

L'architettura della rete è costituita da tre layer convoluzionali seguiti da due layer fully connected. Ogni layer convoluzionale utilizza la funzione di attivazione ReLU per introdurre non linearità. La struttura dettagliata è descritta come segue:

- *Layer 1 (Convoluzione):*
  - **Input:** Tensore  $(B, C_{\text{in}}, H_{\text{in}}, W_{\text{in}})$ , dove  $B$  è il batch size,  $C_{\text{in}}$  = numero di canali,  $H_{\text{in}} = W_{\text{in}} = 84$ .
  - **Operazione:** Convoluzione 2D con 32 filtri, kernel  $8 \times 8$  e stride 4.

- **Output:** Tensore  $(B, 32, 20, 20)$ .
- *Layer 2 (Convoluzione):*
  - **Input:** Tensore  $(B, 32, 20, 20)$ .
  - **Operazione:** Convoluzione 2D con 64 filtri, kernel  $4 \times 4$  e stride 2.
  - **Output:** Tensore  $(B, 64, 9, 9)$ .
- *Layer 3 (Convoluzione):*
  - **Input:** Tensore  $(B, 64, 9, 9)$ .
  - **Operazione:** Convoluzione 2D con 64 filtri, kernel  $3 \times 3$  e stride 1.
  - **Output:** Tensore  $(B, 64, 7, 7)$ .
- *Layer Fully Connected 1:*
  - **Input:** Tensore vettorizzato di dimensione  $64 \times 7 \times 7 = 3136$ .
  - **Output:** Vettore di dimensione 512.
- *Layer di Output:*
  - **Input:** Vettore di dimensione 512.
  - **Output:** Vettore di dimensione  $n_{\text{actions}} = 6$ , rappresentante i valori Q per ciascuna azione disponibile.

L'architettura della rete implementata è illustrata in Figura 3.2.

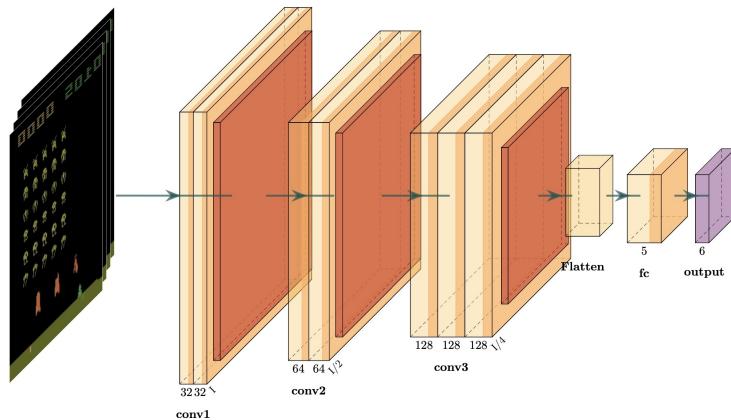


Figura 3.2: Architettura della Deep Q-Network (DQN)

### 3.2.2.2 Replay Memory

Il modello integra una *replay memory*, che memorizza esperienze in forma di tuple con lo scopo di rimuovere correlazioni sequenziali e migliorare la stabilità dell'apprendimento. La replay memory è stata implementata come una deque con una capacità massima predefinita e gestisce due tipi di esperienze. Le *esperienze normali* vengono memorizzate nella deque principale, mentre le *esperienze held-out* vengono conservate separatamente in un sottoinsieme specifico, destinato alla valutazione offline della qualità della policy appresa.

Ogni esperienza è rappresentata da una tupla denominata Transition, che include cinque elementi:

$$(\text{state}, \text{action}, \text{reward}, \text{next\_state}, \text{done}).$$

### 3.2.2.3 Policy

L'agente utilizza un approccio  $\epsilon$ -greedy per bilanciare esplorazione ed sfruttamento:

- **Esplorazione:** con probabilità  $\epsilon$ , l'agente seleziona un'azione casuale.
- **Sfruttamento:** con probabilità  $1 - \epsilon$ , l'agente seleziona l'azione con il massimo valore Q stimato.

Durante il training,  $\epsilon$  viene ridotto linearmente da 1.0 a 0.1. Riferisi a [4.1](#) per maggior dettagli sulla scelta di  $\epsilon$ .

### 3.2.2.4 Ottimizzazione della rete

La funzione obiettivo per l'ottimizzazione è la *Mean Squared Error (MSE)* [[das2004mean:mse](#)] tra i valori Q predetti e i target Q calcolati come segue:

$$Q_{\text{target}} = \text{reward} + \gamma \cdot \max Q_{\text{next}} \cdot (1 - \text{done}),$$

dove  $\gamma$  è il fattore di sconto. Per stabilizzare l'apprendimento, la rete target viene aggiornata periodicamente con i pesi della policy network.

### 3.2.2.5 Training (DQ-Learning)

L'algoritmo di training consiste nei seguenti passi principali:

1. Accumulare esperienze nella Replay Memory fino al raggiungimento di una soglia iniziale (*Replay Start Size*).
2. Campionare mini-batch di esperienze e aggiornare i pesi della rete Q utilizzando la retropropagazione.
3. Aggiornare la rete target ogni  $N$  passi.
4. Monitorare la convergenza utilizzando metriche come il valore medio di Q e la variazione di Q ( $\Delta Q$ ).

Lo pseudocodice relativo all'implementazione della DQN, mostrato in [[mnih2015:dqn](#)], è riportato nell'Algoritmo [2](#) che ne riassume l'intera sequenza di passi.

### 3.2.2.6 Valutazione e salvataggio del modello

Il modello include due funzionalità principali: la *valutazione offline* e il *salvataggio*. La valutazione offline consente di calcolare il valore medio della funzione Q su un insieme di esperienze tenute da parte (held-out), fornendo così una misura dell'efficacia del modello su dati non utilizzati direttamente per il training. D'altra parte, la funzionalità di salvataggio permette di serializzare i parametri del modello insieme allo stato corrente del checkpoint, facilitando il ripristino dell'agente in un secondo momento.

### 3.2.2.7 Motivazioni

L'uso della Deep Q-Network (DQN) è stato motivato dalla necessità di affrontare ambienti con spazi di stato ad alta dimensionalità, come quelli rappresentati dai videogiochi Atari, in cui i metodi tradizionali tabulari risultano di difficile applicazione. Sebbene il Q-learning sia una solida base teorica per l'apprendimento per rinforzo, la sua dipendenza dalla memorizzazione esplicita della funzione  $Q(s, a)$  in una tabella rende il metodo impraticabile per spazi continui o di grandi dimensioni.

Abbiamo scelto la DQN in quanto combina i vantaggi del Q-learning con la capacità delle reti neurali convoluzionali di estrarre caratteristiche rilevanti da input ad alta dimensionalità. Inoltre, l'utilizzo della *Replay Memory* e della rete target, ha garantito una maggiore stabilità nell'apprendimento rispetto ai metodi tradizionali.

#### DQN

```

Input : Replay memory  $D$  di capacità massima  $N$ , fattore di sconto  $\gamma \in (0, 1)$ , frequenza  

        di aggiornamento rete target  $C$ , batch size  $B$ , learning rate  $\alpha$ ,  $\epsilon$  iniziale.  

Inizializza la rete  $Q$  con pesi casuali  $\theta$ ;  

Inizializza la rete target  $\hat{Q}$  con pesi  $\theta^- \leftarrow \theta$ ;  

Inizializza la Replay Memory  $D$ ;  

for ogni episodio do  

    Inizializza lo stato iniziale  $s_1$  preprocessato come  $\phi(s_1)$ ;  

    while l'episodio non è terminato do  

        Con probabilità  $\epsilon$ , seleziona un'azione casuale  $a_t$ ;  

        Altrimenti, seleziona  $a_t \leftarrow \arg \max_a Q(\phi(s_t), a; \theta)$ ;  

        Esegui  $a_t$ , osserva la ricompensa  $r_t$  e il nuovo stato  $s_{t+1}$ ;  

        Calcola  $\phi(s_{t+1})$ ;  

        Memorizza la transizione  $(\phi(s_t), a_t, r_t, \phi(s_{t+1}), d_t)$  in  $D$ ;  

        if Replay Memory  $D$  contiene almeno  $B$  transizioni then  

            Estrai un minibatch casuale  $B$  da  $D$ ;  

            for ogni transizione  $(\phi(s_j), a_j, r_j, \phi(s_{j+1}), d_j) \in B$  do  

                Calcola il target  $y_j \leftarrow \begin{cases} r_j, & \text{se } d_j = 1, \\ r_j + \gamma \max_{a'} \hat{Q}(\phi(s_{j+1}), a'; \theta^-), & \text{altrimenti.} \end{cases}$   

            end  

            Aggiorna i pesi  $\theta$  minimizzando la perdita:  


$$L(\theta) = \frac{1}{|B|} \sum_{j \in B} (y_j - Q(\phi(s_j), a_j; \theta))^2.$$
  

        end  

        if  $t \bmod C == 0$  then  

            Aggiorna la rete target  $\theta^- \leftarrow \theta$ ;  

        end  

    end  

end
```

### 3.2.3 Asynchronous Advantage Actor-Critic (A3C)

L'Asynchronous Advantage Actor-Critic (A3C) rappresenta un approccio innovativo per il deep reinforcement learning, come descritto nel lavoro di Mnih et al. [mnih2016:a3c]. Esso combina i vantaggi delle architetture actor-critic con un framework asincrono, permettendo un'esplorazione più efficiente e una stabilità maggiore nel processo di apprendimento. L'algoritmo è stato progettato per affrontare i limiti degli approcci basati sull'utilizzo della replay memory, come per le Deep Q-Networks (DQN), sfruttando l'esecuzione parallela di più agenti.

La principale innovazione introdotta risiede nell'utilizzo di più *actor-learner threads* che integrano simultaneamente con le loro istanze dell'ambiente, aggiornando in modo asincrono una rete globale condivisa. Questa strategia riduce le dipendenze tra gli aggiornamenti, promuove una decorrelazione dei dati e consente l'utilizzo di metodi on-policy con deep neural network. Inoltre, l'A3C elimina la necessità di hardware specializzato, essendo progettato per funzionare su CPU multi-core.

#### 3.2.3.1 Architettura del modello

L'architettura dell'A3C include due reti principali: *Actor* (policy network) e *Critic* (value network). Actor è la rete responsabile della generazione della policy, implementata tramite una softmax che restituisce una distribuzione di probabilità sulle azioni possibili. Il compito dell'Actor è quello di massimizzare il vantaggio atteso, rappresentato dalla differenza tra il valore stimato dello stato corrente e il ritorno effettivo (*advantage*). Critic rappresenta invece la rete dedicata alla stima del valore dello stato (*value function*), che guida l'Actor nella scelta delle azioni ottimali. Questo modulo stima il valore futuro atteso di uno stato e contribuisce alla retropropagazione dell'advantage.

Entrambe le reti condividono una struttura convoluzionale iniziale per l'estrazione di caratteristiche visive dall'ambiente di gioco ma si differenziano per i layer di output ad esse associati. La configurazione dettagliata è la seguente:

- *Layer 1 (Convoluzione):*
  - **Input:** Tensore  $(B, C_{\text{in}}, H_{\text{in}}, W_{\text{in}})$ , dove  $B$  è il batch size,  $C_{\text{in}}$  è il numero di canali di input, e  $H_{\text{in}} = W_{\text{in}} = 84$ .
  - **Operazione:** Convoluzione 2D con 32 filtri, kernel  $8 \times 8$  e stride 4.
  - **Output:** Tensore  $(B, 32, 20, 20)$ .
- *Layer 2 (Convoluzione):*
  - **Input:** Tensore  $(B, 32, 20, 20)$ .
  - **Operazione:** Convoluzione 2D con 64 filtri, kernel  $4 \times 4$  e stride 2.
  - **Output:** Tensore  $(B, 64, 9, 9)$ .
- *Layer 3 (LSTM):*
  - **Input:** Tensore vettorizzato di dimensione  $64 \times 9 \times 9 = 5184$ .
  - **Operazione:** Strato ricorrente LSTM con 128 unità nascoste per catturare le dipendenze temporali tra gli stati.
  - **Output:** Vettore di dimensione 128.
- *Layer di Output per Actor:*
  - **Input:** Vettore di dimensione 128.

- **Operazione:** Strato fully connected con  $n_{actions}$  unità, seguito da una funzione Softmax per produrre una distribuzione di probabilità sulle azioni.
  - **Output:** Vettore di dimensione  $n_{actions}$ , rappresentante la probabilità di ciascuna azione.
- *Layer di Output per Critic:*
    - **Input:** Vettore di dimensione 128.
    - **Operazione:** Strato fully connected con 1 unità per stimare il valore dello stato corrente.
    - **Output:** Scalare, rappresentante il valore stimato dello stato.

Le architetture delle reti implementate sono illustrate nelle Figure 3.3 e 3.4.

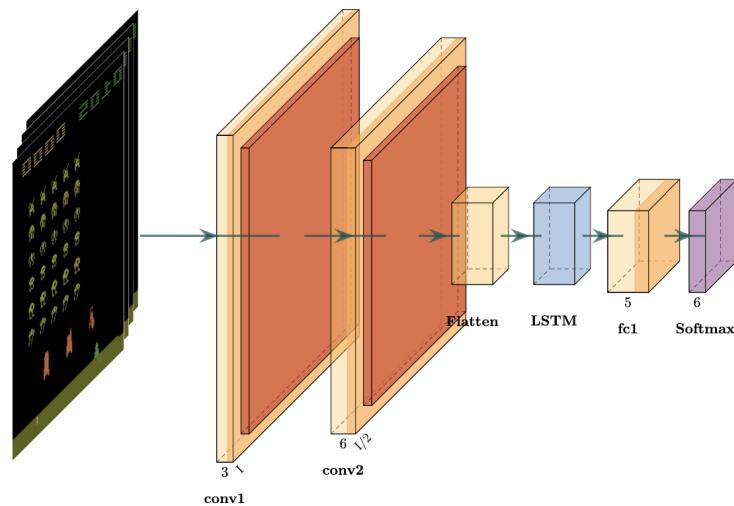


Figura 3.3: Architettura dell'A3C Actor

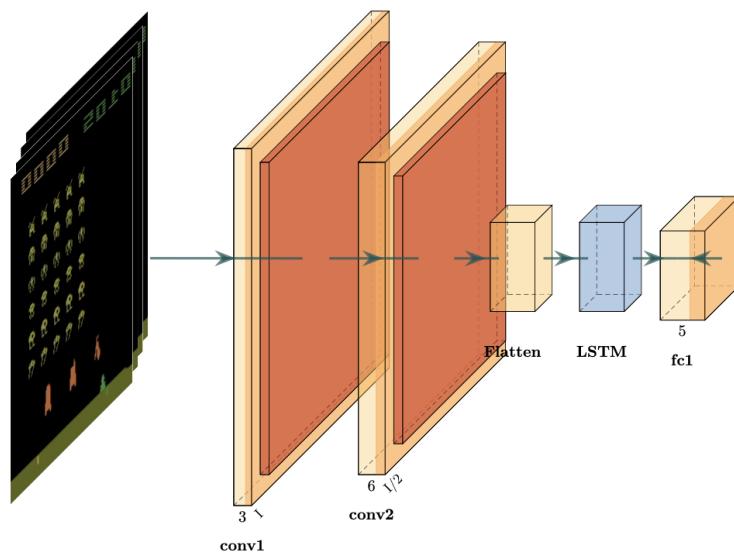


Figura 3.4: Architettura dell'A3C Critic

### 3.2.3.2 Training

L'A3C segue un framework asincrono dove più processi (worker) aggiornano una rete globale in parallelo. Ogni worker utilizza una propria copia della rete locale per interagire con l'ambiente, accumulare esperienze e calcolare gradienti. La rete globale viene aggiornata in modo asincrono utilizzando i gradienti aggregati.

Lo pseudocodice relativo all'implementazione della DQN, mostrato in [[mnih2016:a3c](#)], è riportato nell'Algoritmo 3 che ne riassume l'intera sequenza di passi.

```

A3C

Input : Parametri condivisi  $\theta$  e  $\theta_v$ , contatore globale  $T = 0$ , iperparametri  $\gamma, \beta$ , massimo numero di step  $t_{max}$ .
Inizializza i gradienti:  $d\theta \leftarrow 0, d\theta_v \leftarrow 0$ ;
Sincronizza i parametri locali con quelli globali:  $\theta' \leftarrow \theta, \theta'_v \leftarrow \theta_v$ ;
while  $T < T_{max}$  do
    Ottieni lo stato  $s_t$ ;
     $t_{start} \leftarrow t$ ;
    while  $t - t_{start} < t_{max}$  e  $s_t$  non terminale do
        Seleziona l'azione  $a_t \sim \pi(a_t | s_t; \theta')$ ;
        Osserva la reward  $r_t$  e lo stato successivo  $s_{t+1}$ ;
        Aggiorna  $t \leftarrow t + 1, T \leftarrow T + 1$ ;
    end
    Calcola il valore bootstrap  $R = \begin{cases} 0 & \text{se } s_t \text{ terminale,} \\ V(s_t, \theta'_v) & \text{altrimenti.} \end{cases}$ ;
    for  $i \in \{t - 1, \dots, t_{start}\}$  do
         $R \leftarrow r_i + \gamma R$ ;
        Accumula il gradiente della policy:  $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i | s_i; \theta') \cdot (R - V(s_i; \theta'_v))$ ;
        Accumula il gradiente del value:  $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i; \theta'_v))^2 / \partial \theta'_v$ ;
    end
    Esegui l'aggiornamento asincrono di  $\theta$  e  $\theta_v$  usando  $d\theta$  e  $d\theta_v$ ;
end
```

### 3.2.3.3 Parallelizzazione e Worker

La parallelizzazione nell'A3C si basa sull'utilizzo di worker implementati come processi separati, ciascuno dei quali opera su un'istanza indipendente dell'ambiente. Ogni worker avvia e mantiene una propria copia locale della rete, che viene periodicamente sincronizzata con la rete globale. Gli aggiornamenti alla rete globale avvengono in modo asincrono: ciascun worker contribuisce con i gradienti calcolati sulla base delle proprie esperienze, permettendo di integrare le informazioni raccolte da ambienti differenti. Operando in questo modo il modello non solo migliora la stabilità dell'apprendimento, ma garantisce anche l'efficienza computazionale.

### 3.2.3.4 Funzione obiettivo

La funzione obiettivo combina le perdite di policy e value, con un termine di entropia per incentivare l'esplorazione ed è definita come:

$$L = L_\pi + \beta \cdot H - L_V$$

Dove:

- $L_\pi = -\log \pi(a_t|s_t; \theta) \cdot A(s_t, a_t)$  è la perdita relativa alla policy.
- $L_V = (R - V(s_t; \theta_v))^2$  è la perdita relativa al value.
- $H$  rappresenta l'entropia della policy bilanciata dall'iperparametro  $\beta$ .

### 3.2.3.5 Motivazioni

L'algoritmo A3C rappresenta una soluzione avanzata che combina due aspetti chiave per affrontare le sfide dell'apprendimento per rinforzo in ambienti complessi: l'efficienza della parallelizzazione asincrona e l'uso di deep neural network per modellare sia la policy che la state value function. Ciò risulta particolarmente indicato per contesti caratterizzati da alta dimensionalità e dinamiche complesse, come i giochi Atari ed in particolare come il gioco Space Invaders, dove è essenziale bilanciare l'esplorazione di nuove strategie e lo sfruttamento delle conoscenze acquisite per massimizzare le prestazioni dell'agente.

Uno dei principali limiti degli algoritmi di apprendimento per rinforzo tradizionali, come il Q-Learning, è la difficoltà di gestire dati non stazionari e l'instabilità dell'apprendimento quando si utilizzano reti neurali profonde come funzione approssimante. Algoritmi come il Deep Q-Network (DQN), pur avendo ottenuto buoni risultati, si basano sull'utilizzo di una replay memory per ridurre la correlazione tra le esperienze. Tuttavia, questa scelta comporta un maggiore consumo di memoria e richiede algoritmi off-policy, limitando la flessibilità nell'implementazione di altre strategie di apprendimento.

L'A3C supera queste limitazioni adottando un framework asincrono in cui molteplici worker interagiscono in parallelo con istanze indipendenti dell'ambiente. Questo design elimina la necessità di una replay memory, poiché la diversità degli stati visitati dai worker in esecuzione simultanea genera dati decorrelati in modo naturale. Questo non solo stabilizza l'apprendimento, ma consente anche l'applicazione di algoritmi on-policy, come l'Actor-Critic, che tradizionalmente soffrivano di problemi di instabilità quando combinati con deep neural network.

Inoltre, il framework asincrono offre vantaggi significativi rispetto al DQN e al Q-Learning in termini di efficienza computazionale. Gli aggiornamenti alla rete globale vengono effettuati in modo asincrono, riducendo il carico computazionale rispetto a soluzioni che richiedono un accesso sincrono alle risorse. Questo consente all'A3C di sfruttare appieno le CPU multi-core, eliminando la dipendenza da GPU. Come evidenziato anche in [[mnih2016:a3c](#)] questo modello è in grado di superare le prestazioni della DQN utilizzando solo CPU e in un tempo di addestramento inferiore.

Un altro punto di forza dell'A3C è la sua capacità di esplorare simultaneamente diverse aree dello spazio delle policy. Ogni worker aggiorna in modo indipendente i parametri della rete globale sulla base delle esperienze accumulate, consentendo all'agente di apprendere da un ventaglio di stati e strategie più ampio rispetto a un singolo agente centralizzato. Ciò consente anche di ridurre il rischio di convergenza verso politiche sub-ottimali, un problema noto negli algoritmi come il Q-Learning e le sue varianti.

Infine, l'architettura dell'A3C, che integra una rete ricorrente (LSTM) per catturare le dipendenze temporali, rende l'algoritmo adatto ad ambienti con dinamiche non stazionarie e azioni dipendenti dal contesto.

# 4

CAPITOLO N°

Setup Sperimentale . . . . .	19	4.2.3 Metriche Specifiche per A3C . . . . .	21
Metriche di Valutazione . . . . .	20	Risultati . . . . .	22
4.2.1 Metriche comuni a tutti i modelli . . . . .	20	4.3.1 Q-Learning . . . . .	22
4.2.2 Metriche Specifiche per DQN . . . . .	21	4.3.2 DQN . . . . .	26
		4.3.3 A3C . . . . .	31

## RISULTATI E DISCUSSIONI

### 4.1 Setup Sperimentale

**Ricerca degli Iper-parametri** I valori e gli intervalli degli iper-parametri considerati sono stati scelti sulla base dei valori impiegati in [mnih2015:dqn]. Gli iper-parametri sono stati adattati sfruttando l’ottimizzazione bayesiana [snoek2012:bayesian], eseguendo 5 iterazioni per modello. La Tabella 4.1 mostra i valori e gli intervalli considerati per gli iper-parametri dei modelli considerati. Sia per la DQN che per A3C, ci siamo attenuti all’implementazione originale, optando per RMSProp [tieleman2012:rmsprop] come ottimizzatore.

Modello	$\alpha$	$\gamma$	$\epsilon$
Q-Learning	[0.025, 0.1]	{1.0, 0.8, 0.5}	[0.1, 1.0]
DQN/A3C	[0.00025, 0.1]	{1.0, 0.8, 0.5}	[0.1, 1.0]

Tabella 4.1: Valori e intervalli degli iper-parametri considerati per ogni modello.

**Modelli** La combinazione migliore degli iperparametri trovata per Q-Learning comprende  $\alpha = 0.074103$ ,  $\gamma = 0.7$  e  $\epsilon = 0.8$ . Mentre, per la DQN la combinazione migliore comprende  $\alpha = 0.033823$ ,  $\gamma = 0.5$  e  $\epsilon = 0.8$ . La combinazione migliore per A3C è la seguente:  $\alpha = 0.048264$ ,  $\gamma = 0.3$  e  $\epsilon = 0.5$ . Tutti i modelli sono stati addestrati per 5.000 episodi, con 3.000 steps massimi per episodio e con decadimento lineare della  $\epsilon$  pari a 1.000.000 di steps. Sia la DQN che A3C sono stati addestrati con una batch size pari a 32. La capacità massima dell’experience replay (ER) della DQN è stata impostata a 200.000. I valori della batch size e dell’ER sono stati determinati a causa di limitazioni hardware.

**Early Stopping** : Durante l’addestramento di ciascun modello, è stato applicato l’early stopping per prevenire l’overfitting e ottimizzare l’uso delle risorse, evitando di proseguire l’addestramento oltre il necessario in caso di convergenza anticipata (prima del limite di 5000 episodi). In particolare, è stata monitorata la variazione del Delta Q (si veda la sezione successiva) a intervalli di 100 episodi. Un basso valore di tale metrica indica la stabilizzazione della policy, mentre un’alta varianza suggerisce che il modello si trovi ancora in una fase esplorativa. Nei nostri esperimenti, l’addestramento è stato

interrotto quando il valore di Delta Q è sceso sotto la soglia di 0.0003. Per il modello A3C, invece, il criterio di early stopping si è basato sul monitoraggio della loss, in linea con le best practices del deep learning.

## 4.2 Metriche di Valutazione

La scelta delle metriche di valutazione riveste un ruolo centrale nella quantificazione delle prestazioni dei modelli e nell'analisi del processo di apprendimento. Una valutazione accurata consente di identificare tendenze, anomalie (quali il *catastrophic forgetting*, l'*overfitting*, ecc.), ottimizzare gli iperparametri e rilevare eventuali instabilità nel comportamento degli algoritmi.

In questa sezione vengono descritte le metriche utilizzate per ciascun modello (Q-Learning, DQN, A3C), integrando quelle proposte in Oliveira et al. [[oliveira2019difference:metrics](#)] per ottenere un'analisi più espressiva e interpretabile.

### 4.2.1 Metriche comuni a tutti i modelli

**Cumulative Reward** Rappresenta la somma cumulativa delle ricompense non scontate su tutti gli episodi:

$$\text{Cumulative Reward} = \sum_{t=1}^T R_t$$

Questa metrica fornisce una misura aggregata dell'efficacia complessiva della policy. Un valore elevato riflette strategie vincenti, mentre incrementi lenti suggeriscono un'esplorazione inefficiente.

**Average Reward** Rappresenta la media delle ricompense non scontate calcolata ogni  $N$  episodi:

$$\text{Average Reward} = \frac{1}{N} \sum_{t=1}^N R_t$$

Essa fornisce una misura della performance media del modello durante il processo di apprendimento.

**Difference Based Score (DBS)** Questa metrica è definita come la differenza tra le ricompense di finestre consecutive di episodi:

$$\text{DBS}(i) = \text{RawReward}(i + h, n) - \text{RawReward}(i, n)$$

Dove  $h$  è l'ampiezza del salto tra le finestre (nel caso in questione  $h = 1$ , poiché vengono valutati episodi consecutivi),  $n$  rappresenta la dimensione della finestra (nel caso in questione  $n = 100$ ), e RawReward è la ricompensa grezza. Valori positivi indicano un miglioramento, mentre valori negativi indicano una regressione delle prestazioni.

**Weighted Difference Count (WDC)** Le metriche derivate dal DBS permettono di quantificare la direzione e l'intensità delle variazioni. Esse sono definite come segue:

- $\text{WDC}_p$ : rappresenta la somma dei valori DBS positivi in un intervallo e misura il miglioramento cumulativo.
- $\text{WDC}_n$ : rappresenta la somma dei valori DBS negativi e misura il deterioramento.

Le espressioni matematiche per queste due metriche sono:

$$\text{WDC}_p = \sum_i \max(\text{DBS}(i), 0), \quad \text{WDC}_n = \sum_i \min(\text{DBS}(i), 0)$$

**MinMax Average (MMAVG)** Questa metrica quantifica il rapporto tra l'escursione massima e la media delle ricompense in una finestra di dimensione  $N$ , per valutare la varianza normalizzata:

$$\text{MMAVG}(k, n) = \frac{\max_{k \leq i \leq k+n} R_i - \min_{k \leq i \leq k+n} R_i}{\text{AvgReward}(k, n)}$$

Un valore basso di MMAVG, associato a una media alta di ricompense, indica stabilità, mentre un valore elevato di MMAVG suggerisce instabilità.

**Delta Q** Questa metrica misura la variazione media assoluta dei valori  $Q$  tra episodi consecutivi, fornendo una stima della velocità di convergenza del modello:

$$\Delta Q_t = \frac{1}{|\mathcal{S}||\mathcal{A}|} \sum_{s,a} |Q_t(s, a) - Q_{t-1}(s, a)|$$

Dove  $\mathcal{S}$  e  $\mathcal{A}$  rappresentano rispettivamente lo spazio degli stati e delle azioni. Una variazione ridotta indica una convergenza rapida, mentre una variazione elevata suggerisce un apprendimento più instabile.

#### 4.2.2 Metriche Specifiche per DQN

**Average Loss** La metrica della perdita media (MSE) tra i valori  $Q$  target e quelli predetti è la seguente:

$$\mathcal{L} = \frac{1}{B} \sum_{i=1}^B (y_i - Q(s_i, a_i; \theta))^2$$

Dove  $B$  è la dimensione del batch,  $y_i$  è il valore target per l'azione  $a_i$  e lo stato  $s_i$ , e  $Q(s_i, a_i; \theta)$  è il valore predetto dalla rete neurale per l'azione  $a_i$  in  $s_i$ .

#### 4.2.3 Metriche Specifiche per A3C

**Entropy Loss** La metrica di entropia viene utilizzata per regolarizzare l'esplorazione del modello A3C. Essa misura l'incertezza nella distribuzione delle probabilità delle azioni, incentivando il modello a esplorare diverse azioni piuttosto che convergere troppo rapidamente verso una strategia deterministica. L'entropia è calcolata come segue:

$$\mathcal{L}_{\text{entropy}} = - \sum_a \pi(a|s) \log \pi(a|s)$$

Dove  $\pi(a|s)$  è la probabilità di selezionare l'azione  $a$  dato lo stato  $s$ . Un decadimento controllato dell'entropia attraverso un iperparametro  $\beta$  permette al modello di esplorare inizialmente, per poi concentrarsi sulle azioni più promettenti senza incorrere nell'*overfitting*.

**Policy Loss** La policy loss misura l'errore nella selezione delle azioni, calcolando il gradiente delle probabilità. La formula è:

$$\mathcal{L}_{\text{policy}} = -\mathbb{E}[\log \pi(a|s) A(s, a)]$$

Dove  $A(s, a)$  è l'advantage, ovvero la differenza tra il ritorno osservato e il valore stimato dello stato. La loss è negativa per stimolare il modello ad aumentare la probabilità delle azioni che hanno prodotto un risultato positivo (o "advantage").

**Value Loss** La value loss misura la differenza quadratica tra il valore predetto dal modello e il ritorno atteso. In A3C, il valore è predetto dalla rete *Critic* e confrontato con il ritorno calcolato tramite bootstrapping (ossia, utilizzando il valore predetto per il prossimo stato). La formula è:

$$\mathcal{L}_{\text{value}} = \frac{1}{B} \sum_{i=1}^B (V_{\text{target}}(s_i) - V(s_i; \theta))^2$$

Dove  $V_{\text{target}}(s_i)$  è il ritorno target calcolato durante la traiettoria e  $V(s_i; \theta)$  è il valore predetto dalla rete per lo stato  $s_i$ . La loss viene minimizzata affinché il valore predetto si avvicini al ritorno effettivo.

## 4.3 Risultati

In questa sezione vengono analizzati i risultati ottenuti dai tre modelli di apprendimento per rinforzo: Q-Learning, DQN e A3C. Ogni sottosezione seguente presenta un approfondimento sui risultati specifici per ciascun modello, con un focus sulle performance relative in termini di ricompensa cumulativa, velocità di apprendimento e stabilità, utilizzando le metriche descritte nella precedente Sezione 4.2. Per garantire una comparazione equa, tutti i modelli sono stati addestrati e testati nelle stesse condizioni sperimentali.

Il Q-Learning, come metodo classico di apprendimento per rinforzo, funge da baseline per l'analisi, offrendo una comprensione delle capacità e delle limitazioni di un approccio tabulare e senza rete neurale. D'altra parte, la DQN, che integra le reti neurali con Q-Learning, dovrebbe teoricamente superare le difficoltà del Q-Learning tradizionale, specialmente in ambienti complessi. Infine, l'A3C, un algoritmo avanzato basato su approccio actor-critic, mira a migliorare ulteriormente l'efficacia nell'esplorazione e nella convergenza, grazie alla sua capacità di apprendere in parallelo e sfruttare diverse politiche di esplorazione.

### 4.3.1 Q-Learning

In questa sottosezione, vengono presentati i risultati ottenuti utilizzando il Q-Learning. L'analisi si concentra sulle caratteristiche peculiari di questo approccio tabulare, mettendo in evidenza le sue potenzialità e le limitazioni, specialmente in ambienti complessi dove la rappresentazione esplicita di tutte le coppie stato-azione diventa difficile o inefficiente.

**Cumulative Reward** La Cumulative Reward, mostrata in Figura 4.1, evidenzia un andamento quasi costante, con valori leggermente inferiori nei primi 1000 episodi. Questa fase iniziale riflette il "costo" di una politica che privilegia l'esplorazione, necessaria per identificare azioni vantaggiose. Superata questa soglia, la ricompensa cumulativa tende a stabilizzarsi, con alcune oscillazioni e picchi occasionali, attribuibili a scelte dell'agente che, per via della casualità dell'esplorazione, hanno portato a risultati migliori. Questo comportamento suggerisce che il modello ha progressivamente appreso a massimizzare le ricompense nel lungo termine. La crescita sostenuta indica un buon equilibrio tra esplorazione ( $\epsilon = 0.8$  iniziale) e sfruttamento, favorendo una strategia sempre più efficace.

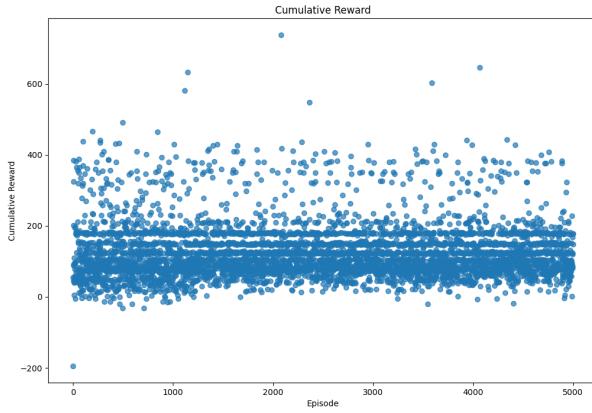


Figura 4.1: Andamento della Cumulative Reward al crescere degli episodi.

**Average Reward** L’Average Reward, illustrata in Figura 4.2, segue un andamento coerente con quanto osservato per la ricompensa cumulativa. Nei primi 1000 episodi, il grafico mostra oscillazioni più marcate, che tendono a ridursi con l’aumentare del numero di episodi di addestramento.

Picchi sporadici, come quello massimo pari a 178.6 nell’episodio 1400, suggeriscono l’acquisizione temporanea di strategie avanzate, come l’eliminazione sequenziale di gruppi di nemici o altre tattiche ottimali. Tuttavia, la difficoltà nel mantenere questi livelli di performance evidenzia le limitazioni intrinseche del Q-Learning tabulare, specialmente in ambienti con stati ad alta dimensionalità, dove la rappresentazione esplicita di tutte le coppie stato-azione diventa meno efficace.

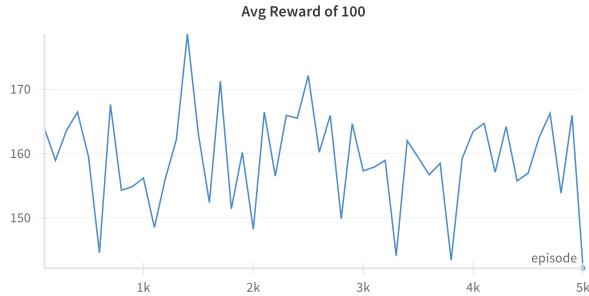


Figura 4.2: Andamento dell’Average Reward al crescere degli episodi.

**Difference Based Score (DBS)** Il grafico riportato in Figura 4.3 mostra valori oscillanti tra  $-200$  e  $+250$ . La fase iniziale (tra 0 e 500) presenta variazioni ampie ( $\Delta DBS \approx 450$ ), riflettendo un’esplorazione aggressiva. Successivamente (tra 1.000 e 5.000), le oscillazioni si riducono ( $\Delta DBS \approx 100$ ), indicando una transizione verso lo sfruttamento.

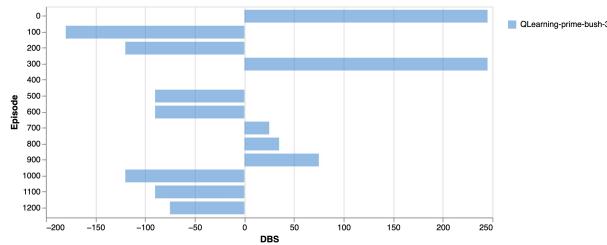


Figura 4.3: Andamento del DBS al crescere degli episodi.

**Weighted Difference Count (WDC)** Le metriche Positive Weighted Difference Count ( $WDC_p$ ) e Negative Weighted Difference Count ( $WDC_n$ ) quantificano rispettivamente l'accumulo di miglioramenti ( $WDC_p = 1.950$ ) e il deterioramento ( $WDC_n = -2.175$ ) cumulativi nel corso dell'addestramento. L'ampia differenza tra questi due valori suggerisce una significativa variazione nelle ricompense tra gli episodi, riflettendo un processo di apprendimento relativamente stabile, come affermato anche in [oliveira2019difference:metrics].

I grafici relativi a queste due metriche sono riportati nella Figura 4.4 seguente.

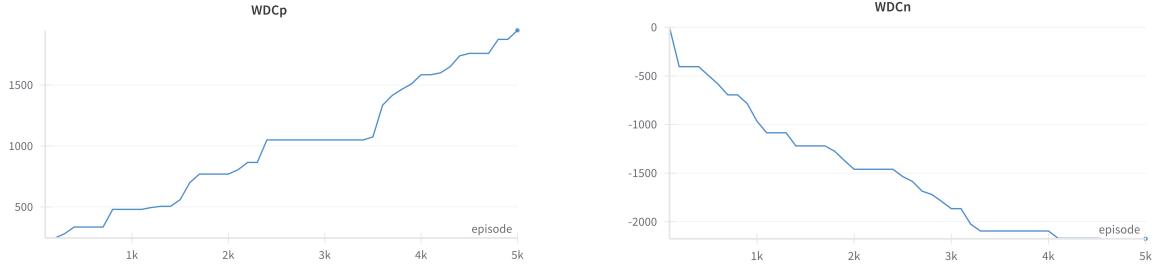


Figura 4.4: Andamento di  $WDC_p$  e  $WDC_n$  al crescere degli episodi.

**MinMax Average (MMAVG)** Questa metrica, considerata isolatamente, fornisce informazioni limitate. Per un'interpretazione più significativa, è necessario metterla in relazione con altre metriche, come DBS e Average Reward.

Osservando il grafico in Figura 4.5 e confrontandolo con quelli delle Figure 4.2 e 4.3, si nota che fasi caratterizzate da una bassa varianza della ricompensa corrispondono generalmente a valori elevati di Average Reward e a un DBS positivo. Questo comportamento suggerisce che l'agente ha appreso strategie efficaci e le sta sfruttando in modo più stabile.

Al contrario, nei periodi in cui la varianza è più elevata, si osserva una tendenza crescente dell'Average Reward e un DBS oscillante. Questo potrebbe indicare fasi di esplorazione più intensa, in cui l'agente sta ancora affinando la propria strategia e testando nuove azioni prima di convergere su una soluzione più ottimale.

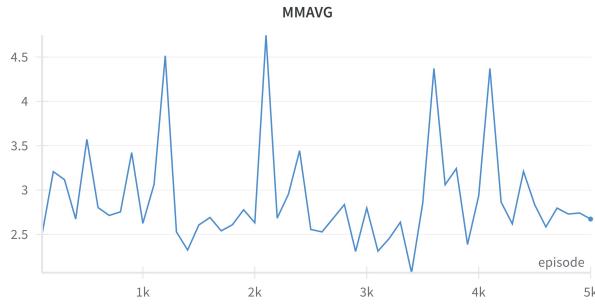


Figura 4.5: Andamento di MinMax Average al crescere degli episodi.

**Delta Q** La variazione media assoluta dei valori Q, illustrata in Figura 4.6, mostra un drastico calo da 60.000 a 10.000 unità nei primi 2.500 episodi, segnalando una rapida convergenza iniziale. Questo comportamento è tipico del Q-Learning, che aggiorna direttamente la tabella Q senza ricorrere ad approssimazioni.

Successivamente, la metrica continua a oscillare in maniera leggermente meno marcata fino al termine dell'addestramento. Queste fluttuazioni residue sono dovute principalmente al decadimento

lineare di  $\epsilon$ , che mantiene una quota ridotta di esplorazione anche nelle fasi avanzate del training ( $\epsilon_{final} = 0.1$ ) , impedendo al modello di convergere prematuramente su una politica subottimale.

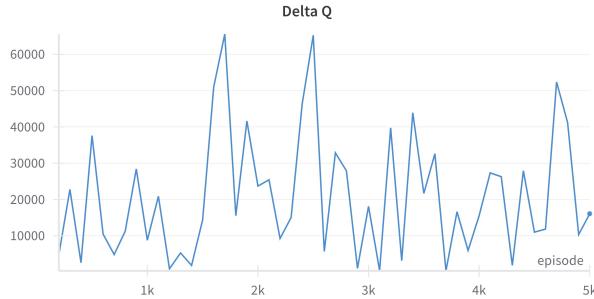


Figura 4.6: Andamento di  $\Delta Q$  al crescere degli episodi.

**Average Q-Value** Il grafico riportato in Figura 4.7 mostra l'andamento del Q-Value medio durante l'addestramento dell'agente in esame. Il valore medio della funzione Q presenta un'oscillazione significativa nel corso dell'addestramento, indicando una variabilità nelle decisioni prese dall'agente. Si osservano picchi superiori a 120.000 e minimi che scendono sotto i 40.000, segno di una fase di esplorazione e apprendimento non ancora stabilizzata. L'andamento non monotono suggerisce che l'agente sta ancora cercando di bilanciare esplorazione e sfruttamento, senza aver raggiunto una convergenza definitiva. Tuttavia, si nota una tendenza generale a valori mediamente più elevati nelle fasi avanzate, suggerendo un miglioramento nella qualità delle decisioni prese dall'agente man mano che l'addestramento procede.

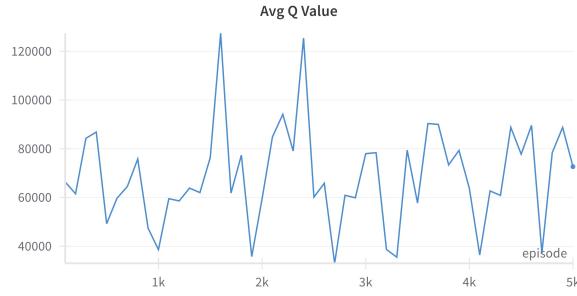


Figura 4.7: Andamento del Q-value medio al crescere degli episodi.

**Analisi integrata** La combinazione delle metriche evidenzia un apprendimento robusto ma non privo di criticità. Il Q-Learning raggiunge prestazioni accettabili grazie alla sua semplicità computazionale, ma soffre di limitazioni nella gestione di stati rari o complessi, come dimostrato dai picchi intermittenti in Average Reward e dalle fluttuazioni in DBS. Il decadimento di  $\epsilon$  ha contribuito a bilanciare esplorazione e sfruttamento, ma l'elevato  $WDC_n$  suggerisce una frequente regressione nelle prestazioni, indicando la necessità di ulteriori ottimizzazioni. In particolare, la crescita del numero di stati con cui l'agente deve interagire rende il Q-Learning tabulare sempre meno efficace, suggerendo la possibile adozione di modelli più avanzati, come DQN o altre varianti basate su funzioni di approssimazione.

### 4.3.2 DQN

In questa sottosezione, vengono presentati i risultati ottenuti con la DQN. L'uso di reti neurali consente a DQN di affrontare ambienti più complessi e ad alta dimensionalità, ma comporta anche sfide relative alla stabilità dell'addestramento e alla capacità di generalizzazione. Verranno analizzati gli effetti dell'integrazione di deep learning sull'efficacia e sulla velocità di apprendimento, nonché sulle difficoltà incontrate durante l'addestramento.

**Average Loss** Il grafico Avg Loss of 100 (Figura 4.8) mostra l'andamento della loss media calcolata su batch di 100 episodi. Nei primi 1.000 episodi, la perdita oscilla tra 7.000 e 8.000 unità, riflettendo l'instabilità iniziale tipica della DQN, dovuta alla combinazione di esplorazione casuale e aggiornamenti della rete neurale. Tra 1.000 e 3.000 episodi, la perdita diminuisce gradualmente fino a 5.000 unità, indicando un miglioramento nella stima dei valori Q. Tuttavia, dopo 3.000 episodi, la perdita tende ad aumentare mantenendo una media di circa 6.000 unità, senza ulteriori riduzioni significative. Questo andamento suggerisce una convergenza subottimale, probabilmente causata da una esplorazione insufficiente.



Figura 4.8: Andamento della loss media al crescere degli episodi.

**Average Q-Value** Il grafico Avg Q-Value (Figura 4.9) mostra l'andamento del valore Q medio predetto dalla rete neurale. Nei primi 1.000 episodi, la metrica presenta valori più bassi ( $\approx 0.5$ ), riflettendo l'iniziale mancanza di conoscenza dell'ambiente. Tra 1.000 e 4.000 episodi, il valore Q aumenta con picchi che raggiungono anche le 3.18 unità, indicando che la rete sta imparando a predire ricompense future più elevate. Tuttavia, dopo 4.000 episodi, il valore Q si stabilizza, senza ulteriori incrementi significativi. Questo comportamento è coerente con l'andamento finale osservato nell'Average Loss e conferma una convergenza incompleta.

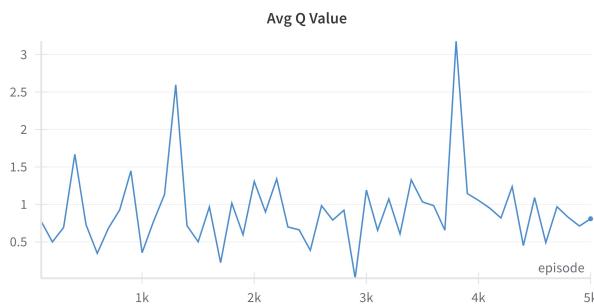


Figura 4.9: Andamento della valore Q-value medio al crescere degli episodi.

**Average Reward** Il grafico Avg Reward of 100 (Figura 4.10) rappresenta la ricompensa media calcolata su finestre di 100 episodi. L'andamento della curva risulta simile a quello osservato per l'agente basato su Q-Learning (Figura 4.2), evidenziando un comportamento oscillante con picchi occasionali. Tuttavia, si nota una minore variabilità nelle fluttuazioni, indicando un miglioramento nelle prestazioni dell'agente. Nonostante ciò, tale miglioramento non è sufficiente a raggiungere livelli prestazionali elevati.

Inoltre, si osserva che, dopo circa 3.000 episodi, la ricompensa media non mostra incrementi significativi rispetto ai valori precedenti. Questo comportamento suggerisce che l'algoritmo DQN abbia raggiunto un limite nelle sue capacità di apprendimento.

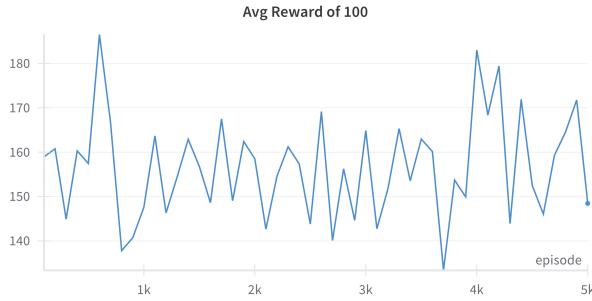


Figura 4.10: Andamento della valore ricompensa media al crescere degli episodi.

**Difference Based Score (DBS)** Il grafico DBS-Step (Figura 4.11) mostra l'andamento del DBS in una finestra compresa tra 5.200 e 6.400 step. Durante questa fase, il DBS oscilla tra -400 e +200, con picchi negativi che coincidono con improvvisi cali di performance. Queste fluttuazioni riflettono l'instabilità intrinseca della DQN, dovuta alla combinazione di aggiornamenti della rete neurale e campionamento casuale dall'experience replay. In particolare, il crollo a -400 intorno a 5.800 step potrebbe essere riconducibile ad un episodio di *catastrophic forgetting*, in cui l'aggiornamento della rete ha sovrascritto conoscenze precedentemente apprese.

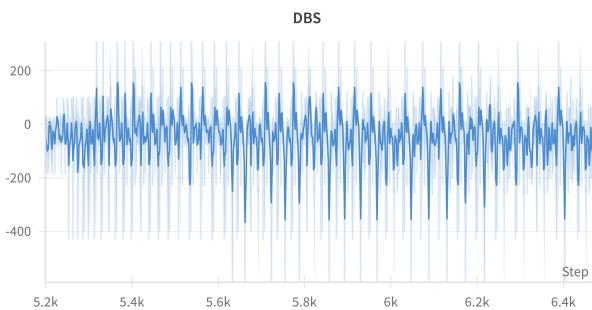


Figura 4.11: Andamento del DBS in una finestra tra 5.200 e 6.400 step.

Il grafico DBS (Figura 4.12) conferma il comportamento osservato in precedenza, mostrando valori che oscillano nell'intervallo compreso tra -200 e +450 per l'intera durata dell'addestramento. La presenza di picchi positivi elevati ( $> +400$ ) suggerisce fasi di apprendimento accelerato in cui l'agente riesce temporaneamente a migliorare le proprie prestazioni in modo significativo. Al contrario, i picchi negativi ( $< -200$ ) evidenziano momenti di regressione, in cui l'agente sperimenta un deterioramento delle sue performance, probabilmente a causa di una politica d'azione non ancora ottimizzata o di esplorazioni inefficaci.

Tuttavia, rispetto al grafico ottenuto per l'agente basato su Q-Learning (Figura 4.3), si osserva una distribuzione più favorevole dei valori, con una predominanza di valori positivi rispetto a quelli negativi. Questo indica che, pur persistendo oscillazioni e instabilità, l'algoritmo DQN mostra complessivamente un miglioramento delle prestazioni rispetto al metodo tabulare. Tale comportamento suggerisce una maggiore capacità dell'agente DQN di adattarsi all'ambiente e apprendere strategie più efficaci, sebbene permangano limiti strutturali nel processo di apprendimento che impediscono una convergenza più stabile e uniforme.

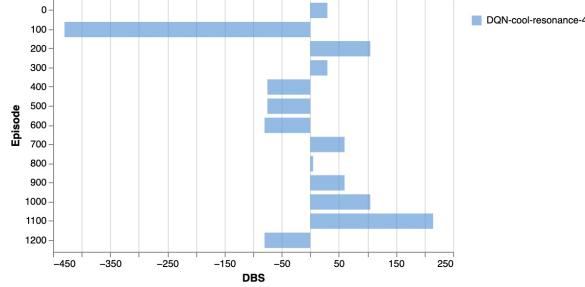


Figura 4.12: Andamento del DBS al crescere degli episodi.

**Delta Q** Il grafico Delta Q (Figura 4.13) mostra la variazione media assoluta dei valori Q tra episodi consecutivi. Nei primi 1.000 episodi, Delta Q è elevato raggiungendo picchi di circa 2.5 unità, riflettendo aggiornamenti significativi nella rete neurale mentre l'agente esplora l'ambiente e impara strategie basilari. Tra 1.000 e 3.000 episodi, Delta Q diminuisce gradualmente fino a 1.0, indicando una progressiva stabilizzazione della policy. Tuttavia, dopo 3.000 episodi, la metrica si stabilizza attorno a 0.5, ad eccezione del picco relativo all'episodio 3800, confermando l'andamento osservato precedentemente nella Average Loss e nell'Avg Q Value. Questo suggerisce che la rete neurale ha smesso di apprendere nuove strategie, limitandosi a raffinarne le esistenti senza miglioramenti sostanziali.

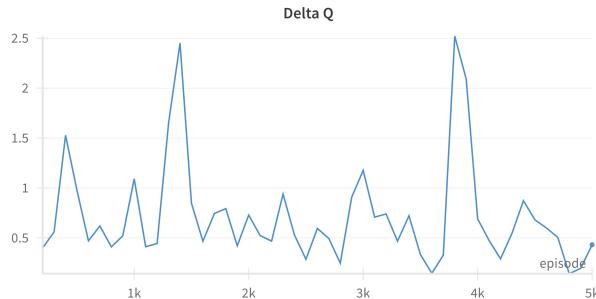


Figura 4.13: Andamento del  $\Delta Q$  al crescere degli episodi.

**MinMax Average (MMAVG)** Il grafico MMAVG (Figura 4.14) rappresenta la varianza normalizzata delle ricompense nel corso dell'addestramento. Nei primi 1.000 episodi, la metrica assume valori superiori a 4, evidenziando un'elevata variabilità nelle ricompense, principalmente dovuta alla fase iniziale di esplorazione casuale. Successivamente, tra i 1.000 e i 3.000 episodi, si osserva una progressiva riduzione di MMAVG fino a circa 3, segnalando una maggiore stabilità della policy man mano che l'agente apprende strategie più consolidate. Tuttavia, oltre i 3.000 episodi, la metrica mostra fluttuazioni comprese tra 3 e 5, indicando la presenza di residui periodi di instabilità, probabilmente causati da sovra-adattamenti temporanei o da transizioni critiche nell'ambiente.

Analogamente a quanto osservato per l'agente basato su Q-Learning, si rileva che fasi caratterizzate da una bassa varianza di MMAVG corrispondono tipicamente a valori elevati di Average Reward e a un DBS positivo, suggerendo che l'agente sta consolidando il proprio apprendimento. Al contrario, valori più elevati di MMAVG tendono ad accompagnarsi a ricompense inferiori e a un DBS fortemente oscillante, segnalando fasi in cui l'agente sta ancora affinando la propria strategia.

Considerando che l'agente DQN mostra in generale ricompense medie più elevate e che, in particolare tra i 3.400 e i 4.500 episodi, i valori di MMAVG risultano più bassi e meno soggetti a picchi rispetto a quelli dell'agente Q-Learning, si può concludere che anche questa metrica evidenzia un miglioramento delle prestazioni. Tuttavia, tale miglioramento non è ancora sufficiente a garantire una stabilità ottimale nell'apprendimento dell'agente.

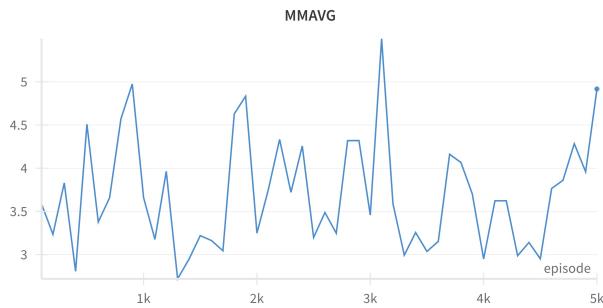


Figura 4.14: Andamento del MinMax Average al crescere degli episodi.

**Raw Reward** Il grafico Raw Reward (Figura 4.15) mostra le ricompense grezze per ogni episodio. Si osservano picchi sporadici fino a 800 unità e valli fino a 0, evidenziando la natura stocastica e ad alta varianza dell'ambiente. Nonostante ciò, la tendenza generale è positiva: dopo 2.500 episodi, i picchi positivi diventano più frequenti ( $> 500$ ), mentre le ricompense nulle si riducono.

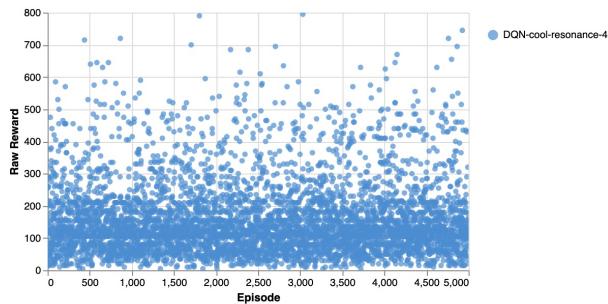


Figura 4.15: Andamento della reward al crescere degli episodi.

**Smoothed AvgReward** Il grafico Smoothed AvgReward (Figura 4.16) conferma le tendenze osservate nelle altre metriche. I valori relativi a tale metrica crescono da 20 a 160 unità nei primi 1.000 episodi, dimostrando un apprendimento progressivo. Tuttavia, negli episodi successivi, la crescita si arresta, stabilizzandosi attorno a 160. Questo plateau è coerente con il comportamento delle altre metriche e suggerisce che la DQN ha raggiunto un limite nelle sue capacità di miglioramento, probabilmente a causa della saturazione dell'experience replay o della complessità intrinseca dell'ambiente.

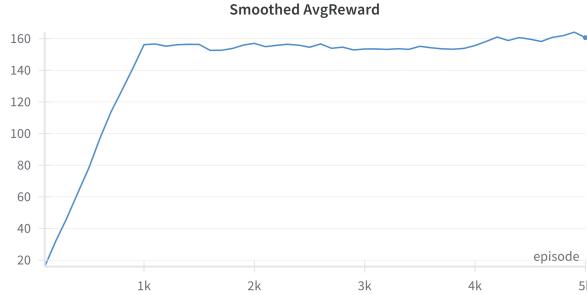


Figura 4.16: Andamento di Smoothed AvgReward al crescere degli episodi.

**Weighted Difference Count (WDC)** I grafici  $WDC_p$  e  $WDC_n$  (Figura 4.17) completano l’analisi della dinamica di apprendimento:

- $WDC_p$  cresce da 0 a 1.830 unità tra 1.000 e 5.000 episodi, indicando un accumulo progressivo di miglioramenti nella policy. Tuttavia, la crescita non è lineare: tra 2.000 e 4.000 episodi, l’aumento rallenta, segnalando una riduzione dell’efficacia degli aggiornamenti.
- $WDC_n$  precipita da -1.000 a -4.310 unità dopo i 1.000 episodi, riflettendo un deterioramento cumulativo significativo.

Rispetto all’algoritmo Q-Learning, si osserva una differenza ancora più marcata tra i due valori, indicando una maggiore variabilità nelle ricompense tra gli episodi. Questo comportamento riflette, anche in questo caso, un processo di addestramento relativamente stabile.

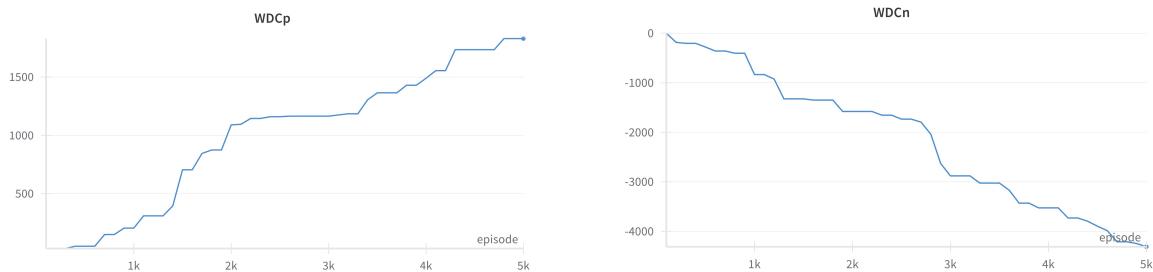


Figura 4.17: Andamento di  $WDC_p$  e  $WDC_n$  al crescere degli episodi.

**Analisi integrata** L’analisi congiunta di tutte le metriche fornisce un quadro coerente ma critico del comportamento della DQN, evidenziando sia i suoi punti di forza che le sue limitazioni. Uno degli aspetti più rilevanti è la *convergenza subottimale*, evidenziata dall’andamento negli ultimi 1.000 episodi osservato nelle metriche *Average Loss* ( $\approx 5.000$ ), *Avg Q Value* ( $\approx 2.5$ ), *Smoothed AvgReward* ( $\approx 160$ ) e *Delta Q* ( $\approx 0.5$ ). Tale fenomeno indica che, dopo circa 3.000 episodi, la rete neurale ha cessato di apprendere nuove strategie. Questa stagnazione può essere attribuita a vincoli hardware, come una dimensione del *batch* ridotta e un *Experience Replay* limitato.

Un ulteriore aspetto da considerare è l’*instabilità intrinseca* del modello. Le oscillazioni significative nei valori di *DBS*, con picchi compresi tra -400 e +450, unite alle variazioni osservate nella metrica *MMAVG*, che oscilla tra 3 e 5.5, e al progressivo deterioramento della *WDCn* fino a -4.000, indicano una sensibilità elevata della DQN agli aggiornamenti della rete. Questo comportamento è probabilmente il risultato della natura stocastica del campionamento nell’*Experience Replay*, che può portare a fenomeni di *catastrophic forgetting*, come evidenziato dal crollo improvviso della performance (es. *DBS* pari a -400).

Un altro elemento di rilievo è lo *squilibrio tra miglioramenti e regressioni*. Sebbene la DQN sia in grado di ottenere miglioramenti significativi, come dimostrato dalla metrica *WDCp*, che raggiunge 1.830 unità, il rapporto tra *WDCp* e il valore assoluto di *WDCn* è pari a 0.425. Ciò implica che ogni fase di apprendimento positivo è seguita da regressioni circa 3 volte più intense, il che spiega il plateau nelle ricompense medie e la mancata ottimizzazione della policy.

Nonostante queste criticità, l'analisi dei dati mostra che la DQN possiede un *potenziale non sfruttato*. In particolare, i picchi elevati nella metrica *Raw Reward* ( $> 800$ ) suggeriscono che l'algoritmo è in grado di raggiungere buone prestazioni in determinate condizioni. Tuttavia, la sua incapacità di mantenere tali livelli indica una scarsa generalizzazione e una forte dipendenza da esperienze specifiche memorizzate nel *replay buffer*.

Sebbene la DQN mostri una capacità di apprendimento superiore rispetto al Q-Learning tabulare, essa presenta ancora limitazioni strutturali che ne influenzano la stabilità e ne ostacolano la convergenza verso una policy ottimale. Al fine di superare tali criticità, si è optato per l'adozione di un approccio di tipo *actor-critic*, i cui risultati saranno illustrati nella sezione seguente.

#### 4.3.3 A3C

In questa sottosezione, vengono illustrati i risultati ottenuti con l'A3C. Quest'ultimo si distingue per la sua capacità di eseguire più agenti in parallelo, il che consente una migliore esplorazione e una convergenza più rapida. L'analisi includerà la valutazione delle sue prestazioni, i benefici derivanti dalla parallelizzazione e l'impatto sulla qualità del modello finale.

**Average Reward of 100** Il grafico Avg Reward of 100 (Figura 4.18) mostra una crescita della ricompensa media, che passa da 50 a 350 unità nell'arco di 4.000 episodi. A differenza della DQN, che non mostra progressi dopo i 3.000 episodi, l'A3C mantiene un tasso di miglioramento all'aumentare degli episodi e presenta valori più elevati rispetto agli altri due agenti analizzati precedentemente. Questo andamento riflette l'efficacia della parallelizzazione: gli agenti multipli esplorano diverse regioni dell'ambiente contemporaneamente, condividendo conoscenze attraverso gradienti asincroni.

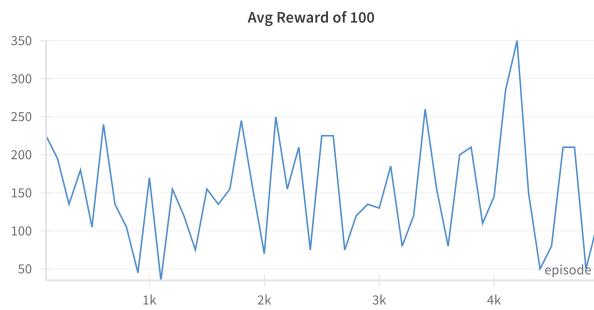


Figura 4.18: Andamento della reward media misurata ogni 100 episodi.

**Reward/Score Mean** La metrica Reward/Score Mean (Figura 4.19), che rappresenta la media normalizzata delle ricompense cumulative (corrispondenti al valore degli score del gioco), cresce da 0.15 a 0.4 in 4.000 episodi. A differenza delle oscillazioni osservate nella DQN, questa curva è priva di picchi negativi significativi, indicando una policy stabile e prevedibile. Valori superiori a 0.35 dopo 3.000 episodi segnalano che l'agente ha imparato a massimizzare le ricompense senza sovra-adattarsi a stati specifici, garantendo una buona generalizzazione.



Figura 4.19: Andamento della reward cumulativa / score medi al crescere degli episodi.

**Difference Based Score (DBS)** Il grafico DBS-Step (Figura 4.20) mostra l’andamento del DBS in una finestra ristretta di step (5.694.460–5.694.500). I valori oscillano tra –200 e +200, con variazioni contenute ( $\Delta DBS \approx 100$ ) e assenza di picchi estremi. Questo comportamento riflette la stabilità dell’A3C: gli aggiornamenti asincroni della policy e della critic network riducono il rischio di *catastrophic forgetting*, mentre la regolarizzazione dell’entropia previene esplorazioni eccessivamente aggressive. A differenza della DQN, non si osservano crolli improvvisi, confermando la robustezza dell’architettura.

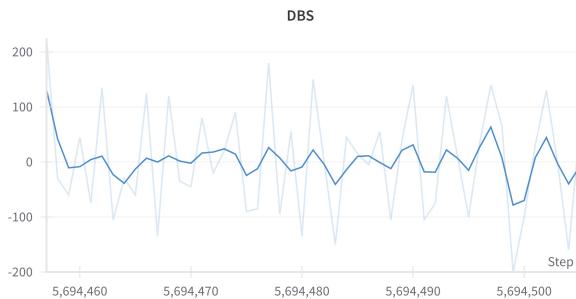


Figura 4.20: Andamento del DBS al crescere degli step.

**MinMax Average (MMAVG)** Il grafico MMAVG (Figura 4.21) evidenzia una progressiva riduzione della varianza, che passa da 0.5 a 0.2 nel periodo compreso tra 1.000 e 4.000 episodi. Si osservano valori significativamente inferiori rispetto a quelli registrati per gli altri due agenti, il che suggerisce una maggiore stabilità dell’apprendimento. Questo andamento è direttamente correlato alla crescita costante della ‘Average Reward’ e alla riduzione dell’entropia della policy. In particolare, valori di MMAVG inferiori a 0.3 dopo 2.000 episodi indicano che l’agente ha raggiunto una strategia stabile, caratterizzata da ricompense prevedibili e da un ridotto livello di rumore nelle azioni eseguite. Inoltre, la diminuzione della varianza è favorita dalla parallelizzazione, che consente di campionare esperienze diversificate, mitigando il rischio di sovra-adattamento. Complessivamente, le prestazioni ottenute con questo agente risultano superiori a quelle precedentemente osservate con DQN e Q-Learning.

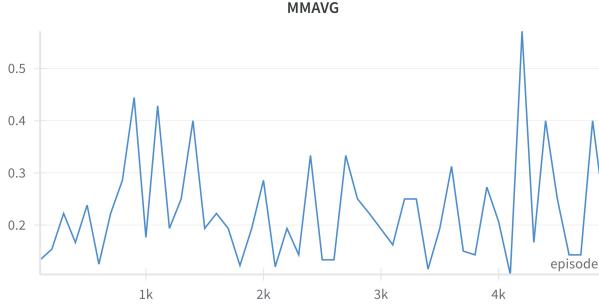


Figura 4.21: Andamento del MinMax Average al crescere degli episodi.

**Weighted Difference Count (WDC)** Le metriche  $WDC_p$  e  $WDC_n$  (Figura 4.22) completano l’analisi condotta finora. Risultano immediatamente evidenti le differenze rispetto ai grafici precedentemente esaminati per questa metrica, i quali mostravano un andamento pressoché lineare in crescita e decrescita. Nel caso dell’A3C, invece, si osservano i seguenti comportamenti:

- $WDC_p$  mostra una crescita da 0 a 200 unità, raggiungendo il massimo a 4.000 episodi. Questo incremento evidenzia un miglioramento continuo della politica.
- $WDC_n$  si mantiene compreso tra -50 e -200 unità, con un rapporto  $WDC_p/|WDC_n| = 1.0$ . A differenza della DQN, in cui le regressioni risultavano tre volte più intense dei miglioramenti, l’A3C dimostra un efficace bilanciamento tra apprendimento e stabilità, evitando deterioramenti critici.

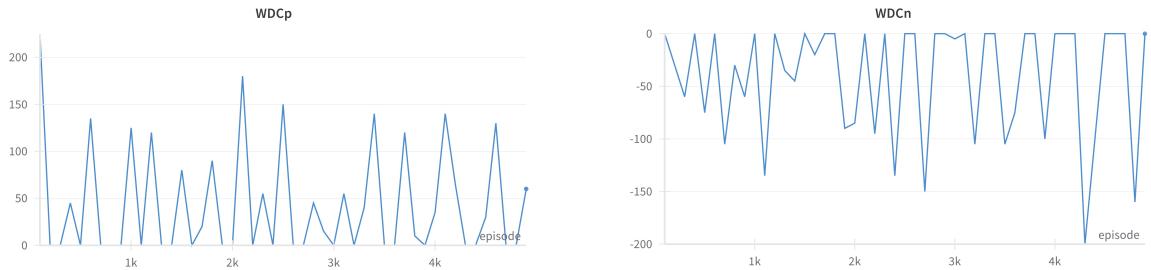


Figura 4.22: Andamento di  $WDC_p$  e  $WDC_n$  al crescere degli episodi.

**Entropy Loss** Il grafico Entropy Loss (Figura 4.23) mostra un decadimento controllato da  $-0.017902$  a  $-0.017906$  nell’arco di 4.000 episodi. Questa lieve riduzione riflette una progressiva specializzazione della policy, in cui l’agente diventa più confidente nelle azioni ottimali, senza tuttavia cadere in un overfitting. La regolarizzazione dell’entropia, introdotta per preservare un livello minimo di esplorazione, garantisce che la perdita non collassi bruscamente, mantenendo un equilibrio tra esplorazione e sfruttamento. Questo comportamento è critico in ambienti dinamici, dove strategie troppo rigide potrebbero fallire in scenari non precedentemente osservati.

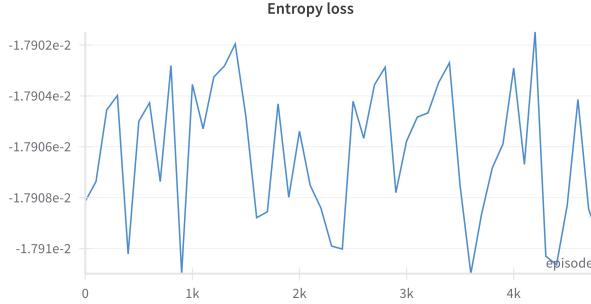


Figura 4.23: Andamento dell'Entropy Loss al crescere degli episodi.

**Policy Loss** La Policy Loss (Figura 4.24), che misura l'errore nella selezione delle azioni basato sull'advantage, diminuisce da 0.8 a 0.4 tra 1.000 e 4.000 episodi. Questo trend indica che l'agente sta ottimizzando la policy per massimizzare i ritorni attesi, riducendo la frequenza di azioni subottimali. A differenza della DQN, dove la policy loss mostrava fluttuazioni ampie, l'A3C mantiene un andamento regolare, grazie alla separazione tra actor (selezione delle azioni) e critic (valutazione degli stati), che riduce la varianza negli aggiornamenti.

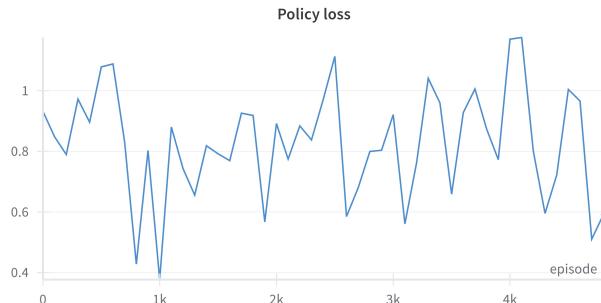


Figura 4.24: Andamento dell'Entropy Loss al crescere degli episodi.

**Value Loss** Il grafico Value Loss (Figura 4.25), che misura l'errore della critic network nella stima dei valori degli stati, assume valori compresi tra 0 e 10, con un picco di 55.6 unità registrato all'episodio 4.100, seguito da una progressiva stabilizzazione. Questa rapida riduzione dell'errore indica un miglioramento nell'accuratezza della stima dei ritorni futuri, fattore determinante per orientare gli aggiornamenti della policy correttamente. La convergenza verso zero dopo 2.000 episodi suggerisce che la critic network abbia raggiunto una comprensione stabile dell'ambiente, fornendo segnali affidabili per l'ottimizzazione dell'actor.



Figura 4.25: Andamento della Value Loss al crescere degli episodi.

**Advantage Mean** Il grafico Advantage Mean (Figura 4.26) mostra la media dell'advantage, calcolato come differenza tra il ritorno osservato e il valore predetto dello stato. La metrica presenta delle oscillazioni tra 0.6 a 0.3 tra 1.000 e 4.000 episodi, indicando che l'agente sta riducendo progressivamente lo scarto tra aspettative e risultati reali. Questo trend riflette una policy sempre più accurata: azioni inizialmente sottostimate (advantage positivo elevato) vengono sostituite da scelte ottimizzate, dove il valore predetto dallo stato ( $V(s)$ ) si allinea ai ritorni effettivi. La riduzione controllata dell'advantage è un segnale di maturità della policy, tipico di un critic network ben addestrato.

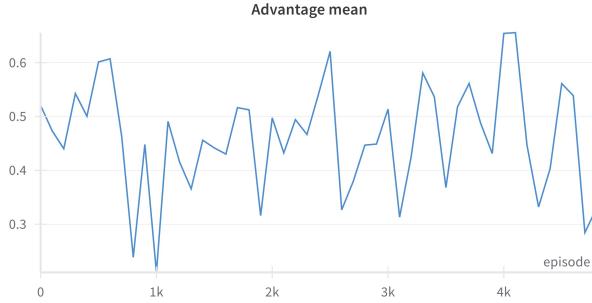


Figura 4.26: Andamento dell'Advantage medio al crescere degli episodi.

**Return Mean** La metrica Return Mean (Figura 4.27), che rappresenta la media dei ritorni scontati, oscilla tra 0.2 e 0.5 in 4.000 episodi. Questo incremento, seppur moderato, è significativo in un ambiente complesso come Space Invaders, dove le ricompense sono sparse e legate a sequenze di azioni coordinate. La crescita, priva di oscillazioni brusche, conferma la capacità dell'A3C di massimizzare le ricompense a lungo termine attraverso una strategia coerente, a differenza della DQN, dove picchi isolati non si traducono in miglioramenti sostenuti.



Figura 4.27: Andamento del Return medio al crescere degli episodi.

**Analisi integrata e comparativa** L'analisi complessiva delle metriche conferma in modo inequivocabile i vantaggi distintivi dell'A3C, evidenziando la sua capacità di adattarsi agli ambienti complessi e di superare le limitazioni di altri approcci. Dall'analisi dei risultati sopra mostrati è emerso l'allineamento tra aspettative e risultati. La riduzione dell'Advantage Mean da 0.6 a 0.3 indica che la critic network è in grado di stimare con sempre maggiore precisione i valori degli stati, contribuendo a una progressiva ottimizzazione degli aggiornamenti della policy. Parallelamente, l'aumento del Return Mean da 0.4 a 0.5 dimostra che l'agente non si limita a migliorare le singole azioni, ma riesce a coordinarle in sequenze strategicamente vantaggiose, massimizzando così le ricompense cumulative. L'analisi integrata delle metriche rivela inoltre una forte sinergia tra i vari parametri. La stabilizzazione della Value Loss a zero dopo 2.000 episodi spiega la riduzione dell'Advantage Mean: una stima più accurata dei valori degli stati consente di ridurre l'incertezza nella selezione delle azioni,

favorendo decisioni più affidabili. Contestualmente, la crescita dello Score (da 50 a 350) e del Return Mean è sostenuta dalla riduzione progressiva della Policy Loss (da 0.8 a 0.4), segnale di una minore incidenza di errori nelle scelte strategiche dell'agente. Un ulteriore elemento distintivo dell'A3C è la sua capacità di bilanciare esplorazione e sfruttamento. La gestione controllata della Entropy Loss e la riduzione della MMAVG (0.2) garantiscono che l'agente esplori l'ambiente in modo efficace, senza cadere nel rischio di sovra-adattarsi a particolari stati e compromettere la generalizzazione. Il confronto definitivo con Q-Learning e DQN evidenzia il netto vantaggio dell'A3C in termini di stabilità ed efficienza. Mentre il Q-Learning soffre di limitazioni tabulari e la DQN risulta instabile a causa dell'aggiornamento basato su una singola rete, l'A3C si distingue per la sua precisione nella stima del valore degli stati. La critic network riduce la varianza degli aggiornamenti, garantendo una maggiore coerenza tra advantage e ritorni. Inoltre, la chiara separazione dei ruoli tra actor e critic permette di ottimizzare la policy ( $L_{policy} \approx 0.4$ ) mentre il critic fornisce valutazioni accurate degli stati ( $L_{value} \approx 0$ ), evitando le inefficienze tipiche della DQN, dove un'unica rete deve gestire entrambi i compiti. Infine, l'A3C si dimostra particolarmente efficace nella parallelizzazione. Il raggiungimento di uno Score pari a 350 in 4.000 episodi è reso possibile dall'esplorazione asincrona e dalla condivisione efficiente delle conoscenze tra le istanze parallele dell'agente. Tuttavia, è importante sottolineare che il processo di addestramento è stato significativamente vincolato dalle risorse computazionali disponibili e dalle limitazioni temporali. Questo elemento assume un'importanza significativa considerando che sia l'A3C che la DQN sono modelli basati su reti neurali convoluzionali sviluppate senza pesi pre-addestrati, il che richiede un'elevata capacità di calcolo per garantire una convergenza ottimale. Nonostante queste limitazioni, il modello A3C ha comunque dimostrato prestazioni notevoli, che risultano ancora più evidenti se messe a confronto diretto con quelle degli altri due modelli, come illustrato in Figura 4.28 in cui sono riportati i grafici delle principali metriche precedentemente analizzate.

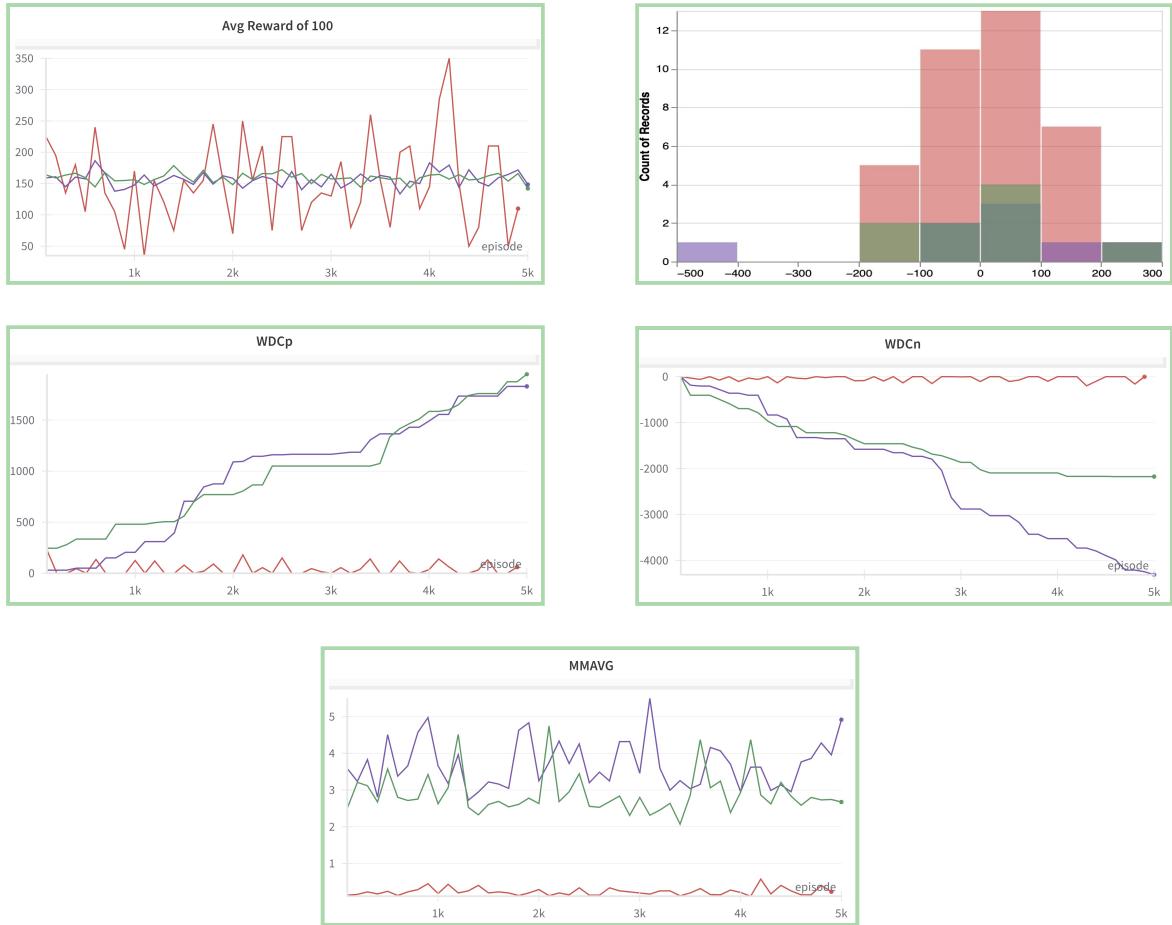


Figura 4.28: Confronto delle metriche Average Reward, DBS,  $WDC_p$ ,  $WDC_n$  e MinMax Average per i modelli Q-Learning (verde), DQN (viola) e A3C (rosso). I grafici mostrano l'andamento di ciascuna metrica nel tempo, evidenziando le performance relative dei tre modelli.

**Risultati Qualitativi** Per consolidare ulteriormente l'analisi comparativa, la Tabella 4.2 riporta le metriche di performance ottenute dai tre algoritmi considerati, valutati su 100 episodi casuali.

Modello	Score Medio	Reward Media	Tempo di gioco medio (secondi)
Q-Learning	3.8	-80.60	<b>50</b>
DQN	132.25	132.25	21.1
A3C	<b>177.6</b>	<b>177.6</b>	35.6

Tabella 4.2: Confronto delle prestazioni tra Q-Learning, DQN e A3C. I migliori risultati sono in grassetto.

I dati riportati in tabella confermano la superiorità dell'A3C rispetto agli altri due approcci. Lo score medio e la reward media ottenuti dall'A3C sono significativamente superiori rispetto a quelli della DQN e del Q-Learning, con un incremento del 34.3% rispetto alla DQN e oltre il 4000% rispetto al Q-Learning. Il tempo medio di gioco offre un'ulteriore chiave di lettura sulle differenze tra gli algoritmi. Il Q-Learning, con un valore di 50 secondi, registra il tempo più elevato, probabilmente

a causa della sua natura tabulare: incontrando frequentemente stati sconosciuti, l'agente fatica a selezionare l'azione ottimale, trascorrendo gran parte del tempo senza compiere scelte efficaci. La DQN e l'A3C, invece, mostrano tempi di gioco più contenuti, rispettivamente 21.1 e 35.6 secondi, a fronte di performance nettamente superiori. Questo suggerisce che entrambi gli algoritmi apprendono policy strategiche più efficienti. Tuttavia, il tempo di gioco leggermente più alto dell'A3C rispetto alla DQN, correlato con i corrispettivi score, indica che l'agente adotta una strategia più bilanciata tra esplorazione e sfruttamento, evitando di convergere prematuramente su soluzioni subottimali e garantendo un apprendimento più efficace nel lungo periodo.

---

## CONCLUSIONI

*Ipomoea Aquatica* ha rappresentato un'opportunità per esplorare le potenzialità dell'apprendimento per rinforzo in un contesto dinamico e complesso come quello del gioco *Space Invaders*. Attraverso l'implementazione e il confronto di tre approcci distinti, quali Q-Learning [[watkins1992:q-learning](#)], Deep Q-Network (DQN) [[mnih2015:dqn](#)] e Asynchronous Advantage Actor-Critic (A3C) [[mnih2016:a3c](#)], è stato possibile delineare vantaggi, limiti e scenari di applicabilità per ciascun algoritmo, offrendo al contempo spunti critici per sviluppi futuri.

Il Q-Learning [[watkins1992:q-learning](#)], sebbene efficace come metodo tabulare per ambienti a bassa dimensionalità, ha evidenziato limitazioni significative nella gestione di spazi di stato complessi. Nonostante un apprendimento inizialmente promettente, la sua incapacità di generalizzare in contesti ad alta varianza ha portato a prestazioni stagnanti, con ricompense medie che si sono stabilizzate precocemente. Questo risultato sottolinea l'importanza di approcci approssimati in scenari dove la rappresentazione esplicita degli stati diventa impraticabile.

La Deep Q-Network (DQN) [[mnih2015:dqn](#)], grazie all'integrazione di reti neurali convoluzionali, ha superato molte delle criticità del Q-Learning, dimostrando una maggiore capacità di gestire stati visivi e raggiungendo ricompense medie più elevate. Tuttavia, l'instabilità negli aggiornamenti —evidenziata da picchi nella Value Loss e oscillazioni nel Difference Based Score (DBS) — ha messo in luce sfide legate alla convergenza e alla stabilità. È doveroso precisare che tali limitazioni sono state amplificate da vincoli infrastrutturali, come l'impossibilità di utilizzare batch size più ampi o un Experience Replay sufficientemente capiente. Risorse hardware più potenti, unitamente a tempistiche di addestramento estese, avrebbero potuto mitigare questi problemi, consentendo aggiornamenti più frequenti della rete e una migliore ottimizzazione.

A3C [[mnih2016:a3c](#)], d'altro canto, si è distinto per robustezza e efficienza. Grazie alla parallelizzazione asincrona e alla separazione tra actor e critic, questo algoritmo ha raggiunto prestazioni superiori, con una crescita costante delle ricompense e una varianza ridotta. La regolarizzazione dell'entropia e la convergenza della Value Loss verso zero hanno confermato una politica bilanciata, capace di adattarsi alle dinamiche del gioco senza sacrificare l'esplorazione. Questi risultati pongono A3C come approccio privilegiato per ambienti ad alta complessità, dove la scalabilità e la stabilità sono fattori determinanti.

Le limitazioni infrastrutturali, in particolare l'assenza di GPU dedicate e risorse computazionali avanzate, hanno inevitabilmente condizionato i risultati, specialmente per la DQN, la cui architettura richiede aggiornamenti frequenti e un Experience Replay di grandi dimensioni. Nonostante ciò,

il progetto ha dimostrato che l'apprendimento per rinforzo, se supportato da scelte algoritmiche ponderate, può produrre agenti capaci di interagire con ambienti dinamici in modo sempre più sofisticato. Per scopi futuri, ci proponiamo di:

- Esplorare ulteriori ottimizzazioni derivanti dall'adozione di hardware dedicato.
- Esplorare l'effetto dell'integrazione di architetture più moderne e complesse, come il Trasformer [**vaswani2023attentionneed**], in sostituzione alla CNN impiegata nella DQN e in A3C e/o l'applicazione di tecniche di transfer learning per accelerare la fase iniziale di addestramento.
- Estendere lo studio a varianti di gioco più complesse, come quelle con scudi mobili o invasori invisibili, rappresenterebbe un benchmark ideale per valutare la capacità degli agenti di adattarsi a scenari non stazionari.