Prima Parte Progetto IA

Super Mario Bross.

Data	19/01/2025
Destinatario	Prof. V. Deufemia, Dott. G. Cimino
Presentato da	Arcangeli Giovanni, Ciancio Vittorio, Di Maio Marco

Sommario

1.Introduzione

1.1 Contesto del progetto

Il presente progetto si colloca nell'ambito dell'esame di Intelligenza Artificiale e mira ad applicare tecniche avanzate di Reinforcement Learning (RL) in un ambiente dinamico e complesso: il videogioco **Super Mario Bros**. Questo titolo, celebre per la sua combinazione di controllo sequenziale, gestione delle risorse e navigazione in ambienti ricchi di ostacoli e obiettivi, rappresenta una sfida stimolante per gli algoritmi di RL. La complessità del gioco lo rende ideale per valutare l'efficacia di approcci diversi, spingendo gli algoritmi a sviluppare strategie intelligenti e ad adattarsi a scenari in evoluzione.

1.2 Obiettivi e motivazioni

L'obiettivo principale è lo sviluppo di un agente in grado di completare autonomamente un livello di Super Mario Bros, ottimizzando le sue prestazioni attraverso l'addestramento con tecniche di RL avanzate. Per raggiungere questo traguardo, sono stati definiti i seguenti sotto-obiettivi:

1. **Implementazione e confronto di algoritmi di RL:** Si sono presi in esame due approcci principali, **DDQN** (Deep Double Q-Network) e **PPO** (Proximal Policy Optimization), per valutare i loro punti di forza e debolezza in un contesto pratico.
2. **Integrazione di un sistema di visione artificiale:** Si è fatto uso di **YOLOv5**, un framework avanzato per il riconoscimento visivo, per fornire all'agente informazioni dettagliate sull'ambiente e migliorare la qualità delle sue decisioni.

Questi obiettivi non solo mirano a massimizzare le prestazioni dell'agente, ma esplorano anche le possibilità offerte dalla combinazione di RL e visione artificiale, aprendo la strada a nuove applicazioni in scenari complessi.

1.3 Struttura della relazione

La relazione è organizzata nei seguenti capitoli, ciascuno dedicato a un aspetto chiave del progetto:

- **Capitolo 2:** Introduzione teorica ai concetti fondamentali del Reinforcement Learning e dei modelli utilizzati, tra cui DDQN, PPO e YOLOv5.
- **Capitolo 3:** Dettagli sulle metodologie adottate, con un focus sulla creazione del dataset YOLO, le architetture implementate e le configurazioni degli algoritmi.
- **Capitolo 4:** Analisi approfondita dei risultati ottenuti, con confronti tra i modelli e una valutazione dell'impatto di YOLOv5 sull'agente.
- **Capitolo 5:** Discussione dei problemi incontrati durante il progetto e delle soluzioni proposte.
- **Capitolo 6:** Conclusioni che riassumono i risultati principali e offrono spunti per sviluppi futuri.

2. Background teorico

2.1 Reinforcement Learning: Principi di base

Il **Reinforcement Learning (RL)** è una branca dell'apprendimento automatico in cui un agente apprende a interagire con un ambiente al fine di massimizzare una funzione obiettivo, detta *reward*. Il processo segue il paradigma stato-azione-ricompensa e può essere riassunto nei seguenti passaggi:

- L'agente percepisce lo stato dell'ambiente al tempo t.
- Seleziona un'azione sulla base di una politica, che può essere deterministica o stocastica.
- Riceve una ricompensa e transita in uno stato successivo.

L'obiettivo principale dell'RL è apprendere una politica ottimale che massimizzi il valore atteso cumulativo delle ricompense future.

2.2 Modelli utilizzati: DDQN e PPO

2.2.1 DDQN (Deep Double Q-Network)

Il **DDQN** è un'evoluzione dell'algoritmo **DQN** (Deep Q-Network), progettata per affrontare il problema della sovrastima dei valori Q. In un DQN standard, un'unica rete neurale è utilizzata sia per selezionare le azioni sia per stimare i valori Q, il che può portare a valutazioni errate delle ricompense future.

Il DDQN introduce una separazione tra la rete principale e quella di target:

- La rete principale (*Qonline*) seleziona l'azione a'.
- La rete target (*Qtarget*) calcola il valore Q associato all'azione a'.

Questa separazione riduce il bias e migliora la stabilità dell'apprendimento. Nel contesto del progetto, il DDQN è stato implementato utilizzando una **CNN** (Convolutional Neural Network) per elaborare i frame di gioco, combinata con un livello denso per stimare i valori Q.

2.2.2 PPO (Proximal Policy Optimization)

PPO (Proximal Policy Optimization) è un algoritmo di RL basato su policy che migliora la stabilità degli aggiornamenti utilizzando un approccio a gradiente. A differenza degli algoritmi basati su funzioni Q, come il DQN, il PPO apprende direttamente una politica che massimizza una funzione di vantaggio A.

La funzione obiettivo del PPO limita gli aggiornamenti drastici della politica, controllando la variazione attraverso un *clipping*. Questo approccio bilancia esplorazione e sfruttamento, evitando il collasso dell'apprendimento.

Il PPO è stato scelto per la sua efficienza nell'addestramento e la capacità di gestire spazi di azione e stato sia continui che discreti, risultando ideale per il contesto di **Super Mario Bros**.

2.3 YOLOv5: Riconoscimento Oggetti e Applicazioni

2.3.1 Principi di funzionamento di YOLOv5

YOLO (*You Only Look Once*) è un framework di visione artificiale progettato per il riconoscimento di oggetti in tempo reale. Il modello divide un'immagine in una griglia e assegna un bounding box e una classe a ciascun oggetto rilevato.

YOLOv5, una delle versioni più avanzate, utilizza un'architettura CNN ottimizzata per:

- Identificare oggetti con alta precisione e velocità.

- Minimizzare i *false positive* e i *false negative* grazie a un migliore bilanciamento tra precisione e richiamo.

2.3.2 Creazione del dataset YOLO

Un elemento fondamentale del progetto è stato l'addestramento di YOLOv5 su un dataset personalizzato, progettato per rilevare e riconoscere 10 classi specifiche presenti nel gioco.

Di seguito le classi individuate:

- **castle**: Il castello posizionato alla fine del livello.
- **interactable**: I blocchi che contengono un power-up.
- **fm**: (Fire Mario) Mario potenziato con l'abilità del fuoco.
- **fpole**: La bandiera di fine livello.
- **hole**: Le buche presenti lungo il percorso.
- **goomba**: Nemici a forma di fungo marrone.
- **pipe**: I classici tubi verdi.
- **mr:** (Mushroom Mario) Mario potenziato con il fungo.
- **sm:** (Small Mario) Mario nella sua forma base, senza power-up.
- **turtle**: Nemici a forma di tartaruga.

La pipeline per la creazione del dataset ha incluso:

1. **Raccolta delle immagini**: Immagini estratte da dataset esistenti o frame catturati dal gioco.
2. **Annotazione manuale**: Utilizzo di strumenti come *Roboflow* per definire i bounding box e le coordinate degli oggetti.
3. **Addestramento del modello**: Configurazione dei parametri YOLOv5, come img_size, batch_size, e epochs, per adattarsi al contesto del videogioco.

2.4 Integrazione di YOLOv5 con PPO

L'integrazione di YOLOv5 con PPO è stata motivata dalla necessità di migliorare l'interpretazione dell'ambiente da parte dell'agente. Nello specifico, abbiamo deciso di utilizzare la versione YOLOv5l, ossia la più precisa tra le versioni, nonostante impiegasse più tempo per l'addestramento. È stata presa questa decisione, dopo aver confrontato i risultati con le altre due versioni (YOLOv5s e YOLOv5m) e aver accertato la qualità superiore della versione YOLOv5l.

In uno scenario tradizionale, PPO utilizza direttamente i frame raw come input, il che può limitare la capacità di cogliere dettagli specifici rilevanti per la strategia di gioco.

Con YOLOv5, l'agente è stato dotato di un ulteriore canale di osservazione contenente i bounding box rilevati, consentendogli di:

- Identificare oggetti chiave come nemici (goomba, turtle) o ostacoli (pipe, hole).
- Adattare dinamicamente la sua politica in funzione delle informazioni visive più strutturate.
- Calcolare la distanza tra Mario e gli oggetti quali nemici o ostacoli.

L'obiettivo era verificare se un input più "semantico" potesse migliorare le decisioni e ridurre i tempi di apprendimento.

3. Metodologia

3.1 Descrizione dell'ambiente di lavoro

3.1.1 Gym e Gym-Super-Mario-Bros

Per il progetto è stato scelto il framework **Gym**, con il pacchetto aggiuntivo **Gym-Super-Mario-Bros**, progettato per simulare ambienti basati sul classico videogioco **Super Mario Bros**. L'ambiente utilizzato è il livello iniziale **SuperMarioBros-1-1-v0**, configurato con due set di controlli:

- **SIMPLE_MOVEMENT**: un insieme ridotto di azioni che semplifica l'interazione.
- **COMPLEX_MOVEMENT**: un set completo che permette un controllo più preciso.

Non sono state introdotte modifiche sostanziali all'ambiente, preferendo un approccio che permettesse all'agente di apprendere autonomamente, anche in presenza di sfide complesse come il controllo del salto.

Sono stati introdotti alcuni wrapper personalizzati per migliorare l'efficacia dell'apprendimento:

- **FrameStack**: combina più frame consecutivi per fornire un contesto temporale.
- **ResizeObservation** e **GrayscaleObservation**: riducono la complessità delle osservazioni trasformando i frame in immagini in scala di grigi con dimensioni 84x84 pixel.
- **Reward Shaping**: incentiva l'agente a progredire nel livello e penalizza cadute e morti.

Questa configurazione ha garantito che l'agente lavorasse su dati più gestibili e rilevanti, senza però alterare la logica del gioco.

3.2 Creazione del dataset YOLO

3.2.1 Raccolta delle immagini e annotazione

Per l'addestramento del modello **YOLOv5**, è stato creato un dataset personalizzato di circa **350 immagini**, raccolte da:

- **Frame di gioco** acquisiti direttamente.
- Dataset preesistenti utilizzati come integrazione.

L'annotazione è stata eseguita utilizzando lo strumento **Roboflow**, che ha permesso di:

1. Definire con precisione i bounding box.
2. Generare i file di configurazione nei formati richiesti da YOLO.

Le immagini sono state etichettate in 10 classi principali, rilevanti per il contesto di Super Mario Bros:

- **castle**: Il castello posizionato alla fine del livello.
- **interactable**: I blocchi che contengono un power-up.
- **fm**: (Fire Mario) Mario potenziato con l'abilità del fuoco.
- **fpole**: La bandiera di fine livello.
- **hole**: Le buche presenti lungo il percorso.
- **goomba**: Nemici a forma di fungo marrone.
- **pipe**: I classici tubi verdi.
- **mr:** (Mushroom Mario) Mario potenziato con il fungo.

- **sm:** (Small Mario) Mario nella sua forma base, senza power-up.
- **turtle**: Nemici a forma di tartaruga.

3.2.2 Suddivisione del dataset

Il dataset è stato suddiviso seguendo una proporzione standard:

- **85%** delle immagini per il training.
- **15%** delle immagini per la validazione.

Questa suddivisione ha permesso di garantire un buon equilibrio tra addestramento e valutazione, massimizzando l'utilizzo dei dati disponibili senza compromettere la capacità di generalizzazione del modello.

3.2.3 Addestramento di YOLOv5

L'addestramento è stato eseguito su una macchina Windows con GPU, utilizzando la configurazione seguente:

- **Dimensione immagine**: 320x320 pixel.
- **Batch size**: 8.
- **Epoche**: 50.
- **Learning rate**: 0.01.

Il comando utilizzato per l'addestramento è il seguente:

```
python train.py --data "C:\path\to\data.yaml" --cfg models/yolov5l.yaml --weights yolov5l.pt --batch-size 8 --epochs 50 --device 0 --project runs/train
```

Il modello risultante è stato ottimizzato per rilevare oggetti chiave nel gioco, migliorando l'osservazione dell'ambiente da parte dell'agente.

3.3 Implementazione dei modelli

3.3.1 DDQN e PPO

I modelli **DDQN** e **PPO** sono stati implementati utilizzando le librerie **PyTorch** e **Stable-Baselines3**.

DDQN

Per il **DDQN** è stata progettata un'architettura basata su:

- **CNN**: per elaborare i frame di gioco e ridurre la dimensionalità dello spazio osservato.
- **Replay Buffer**: per memorizzare esperienze di gioco e migliorare l'efficienza dell'apprendimento tramite l'esperienza passata.

I principali iperparametri utilizzati per l'addestramento di DDQN sono:

- **Learning rate**: 0.0001.
- **Gamma**: 0.99 (fattore di sconto per le ricompense future).
- **Replay buffer size**: 100.000 esperienze.

L'addestramento è stato effettuato su una macchina Windows con GPU, per garantire tempi di calcolo ragionevoli e consentire esperimenti su più configurazioni di iperparametri.

PPO

Il modello **PPO** è stato implementato tramite il framework **Stable-Baselines3**. La configurazione ha sfruttato due diverse impostazioni per confrontare le prestazioni:

1. **n_steps=512** e **learning_rate=0.0000005**.
2. **n_steps=2048** e **learning_rate=0.000005**.

Entrambe le configurazioni hanno utilizzato:

- **Batch size**: 64.
- **Clip range**: 0.2.

Gli esperimenti sono stati eseguiti per **10 milioni di timesteps**, sempre su una macchina con GPU. Questa infrastruttura ha permesso di:

- Eseguire il training in tempi accettabili.
- Testare l'impatto delle diverse configurazioni di iperparametri sulle prestazioni dell'agente.

3.3.2 Integrazione di YOLOv5 con PPO

L'integrazione tra YOLOv5 e PPO ha richiesto la creazione di un wrapper personalizzato, in cui il modello di visione artificiale elaborava ogni frame di gioco, identificando e segmentando oggetti rilevanti. Il risultato è stato un **canale aggiuntivo** che si affianca ai frame raw:

- Il frame originale viene elaborato in scala di grigi e ridimensionato.
- Un ulteriore livello (channel) è stato aggiunto per evidenziare gli oggetti rilevati, come bounding box per i nemici o ostacoli.

Questo approccio ha permesso all'agente PPO di acquisire informazioni più semantiche sull'ambiente, migliorando potenzialmente la capacità di pianificare azioni in base agli oggetti rilevanti nel contesto del gioco.

3.4 Strumenti e hardware utilizzati

Il progetto ha fatto ampio uso di strumenti e framework moderni, tra cui:

- **Librerie e framework**: PyTorch, Stable-Baselines3, YOLOv5l, OpenCV, Gym, CUDA.
- **Strumenti per il dataset**: Roboflow per l'annotazione automatica.
- **Hardware**:
    - Macchina Windows con NVIDIA GeForce RTX 4050 Laptop GPU con 6GB di memoria dedicata e 15,7 GB di memoria condivisa, 32GB di RAM, Intel(R) Core(TM) Ultra 7 155H per l'addestramento di DDQN e PPO.
    - Mac Studio con Chip Apple M2 Ultra con CPU 24-core, GPU 60-core, Neural Engine 32-core e 64GB di memoria unificata per l'addestramento del modello PPO integrato con YOLOv5.

Questa infrastruttura ha garantito prestazioni ottimali per l'addestramento e i test, permettendo di affrontare le diverse sfide del progetto.

4. Risultati e Analisi

4.1 Prestazioni del modello DDQN

L'addestramento del modello Deep Double Q-Network (DDQN) ha prodotto risultati significativi, evidenziando una convergenza graduale durante il processo. Di seguito vengono analizzate le principali metriche osservate:

4.1.1 Lunghezza degli episodi

- **Inizio dell'addestramento:** Durante le prime iterazioni, la lunghezza media degli episodi si è ridotta drasticamente. Questo andamento suggerisce che l'agente ha rapidamente acquisito strategie più efficienti per progredire nel livello, riducendo i tempi sprecati in comportamenti inefficaci.
- **Fase avanzata dell'addestramento:** Nella parte finale, la lunghezza media degli episodi si è stabilizzata intorno ai 500 passi, indicando che l'agente ha raggiunto una certa stabilità nel suo comportamento. Tuttavia, questa strategia, pur essendo più efficiente rispetto all'inizio, rimane subottimale.

4.1.2 Ricompensa cumulativa

- La ricompensa media cumulativa ha mostrato un andamento coerente con la lunghezza degli episodi. Dopo un iniziale calo, si è gradualmente stabilizzata, con un miglioramento progressivo che riflette un apprendimento costante.
- La media mobile delle ricompense ha evidenziato un trend di crescita, confermando che il modello stava ottimizzando le sue azioni per massimizzare i guadagni cumulativi.

4.1.3 Valori Q e perdite

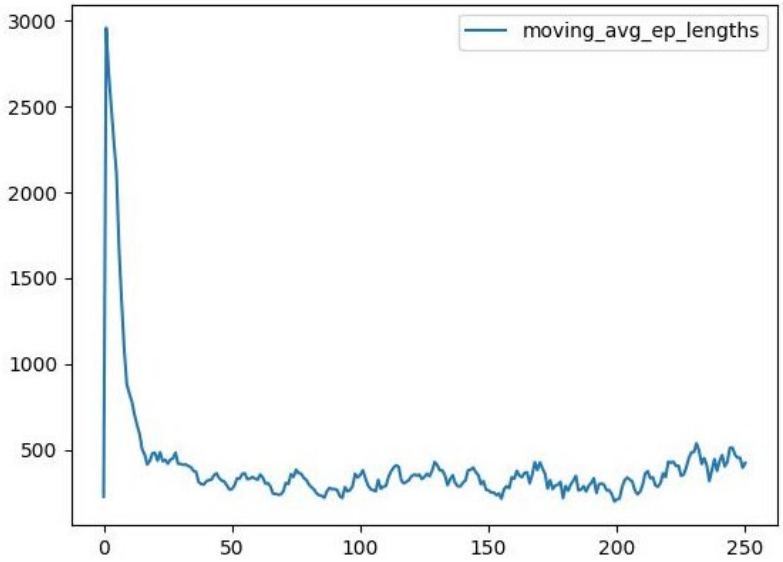
- **Valori Q medi:** Durante l'addestramento, i valori Q medi hanno mostrato un aumento costante, suggerendo una crescente fiducia del modello nelle proprie decisioni. Questo progresso riflette la capacità dell'agente di apprendere le dinamiche dell'ambiente in modo sempre più accurato.
- **Perdite:** Le perdite sono rimaste sotto controllo per tutta la durata dell'addestramento, indicando un processo di apprendimento stabile. Tuttavia, l'assenza di una completa saturazione nella curva delle perdite lascia spazio a ulteriori ottimizzazioni.

4.1.4 Completamento del livello

Il modello DDQN si è dimostrato capace di completare il livello con una vittoria su 1000 episodi. Questo risultato rappresenta una pietra miliare significativa, considerando la complessità dell'ambiente e l'assenza di semplificazioni che potessero agevolare l'apprendimento. Tuttavia, è importante notare che la vittoria è avvenuta intorno al trecentesimo episodio e in modo apparentemente casuale. A causa di questa instabilità, si è deciso di non migliorare ulteriormente il modello in questa fase.

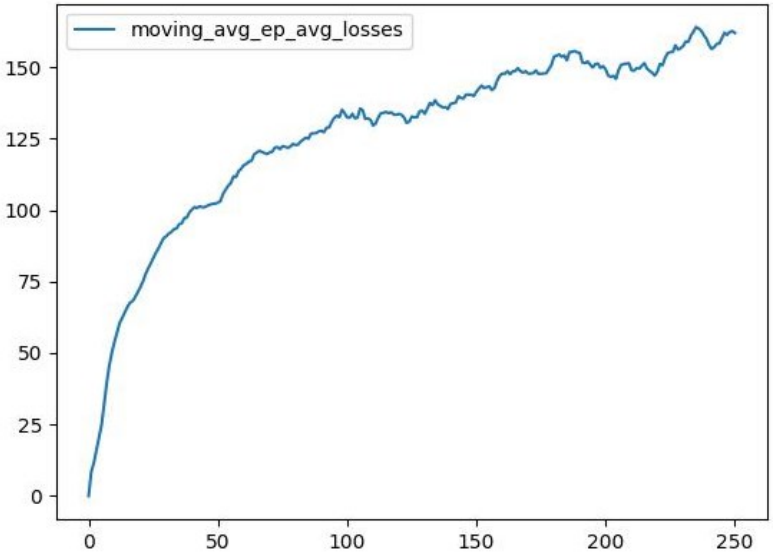
4.1.5 Grafici di supporto

1. **Lunghezza degli episodi**:



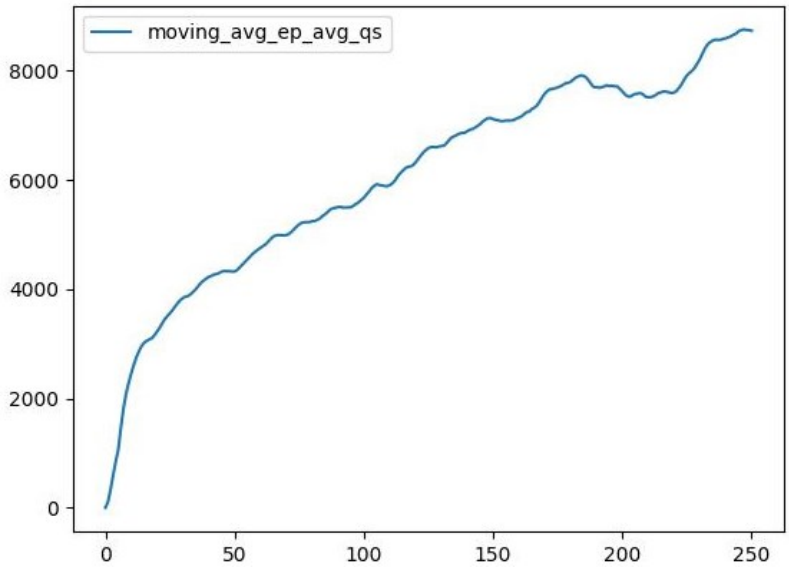
- Il grafico mostra una drastica riduzione iniziale della lunghezza degli episodi, indicando una rapida transizione da un comportamento inefficace a uno più strutturato.
- Dopo questa fase iniziale, si osserva una stabilizzazione intorno ai 400-500 passi, suggerendo che l'agente abbia sviluppato una strategia basilare, pur non raggiungendo l'ottimalità.

2. **Perdita media per episodio**:



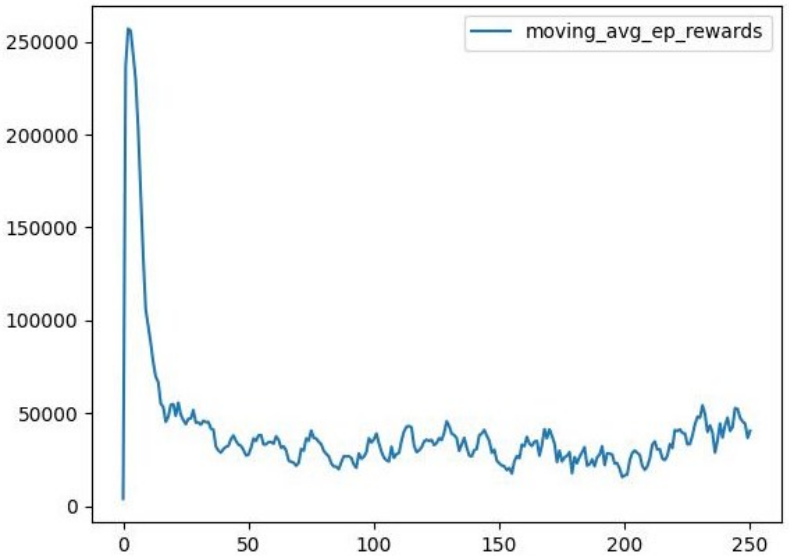
- Le perdite aumentano progressivamente nel corso dell'addestramento, riflettendo un continuo aggiornamento delle stime Q da parte del modello.
- L'assenza di saturazione nella curva potrebbe indicare che il modello non abbia ancora raggiunto il massimo delle sue potenzialità, lasciando aperte opportunità di miglioramento.

3. **Valore medio Q per episodio**:



- Il valore Q medio cresce costantemente, dimostrando che l'agente diventa progressivamente più fiducioso nelle sue scelte.
- Questo andamento positivo sottolinea il progresso del modello nell'apprendimento delle dinamiche dell'ambiente. Tuttavia, un aumento troppo rapido potrebbe anche indicare un potenziale rischio di overfitting.

4. **Ricompensa media per episodio**:



- Il grafico della ricompensa mostra un rapido calo iniziale, seguito da una fase di stabilizzazione a valori medi inferiori.
- Questo andamento suggerisce che, sebbene l'agente abbia migliorato la sua efficienza complessiva, continua a incontrare difficoltà nel massimizzare le ricompense, probabilmente a causa di ostacoli specifici o di strategie non ottimali.

4.2 Prestazione del modello PPO

L'addestramento del modello Proximal Policy Optimization (PPO) è stato eseguito in due configurazioni principali, utilizzando un totale di 10 milioni di passi ciascuna:

- **Configurazione 1**: 512 passi per aggiornamento.
- **Configurazione 2**: 2048 passi per aggiornamento.

Abbiamo scelto di effettuare queste due prove per evidenziare i vantaggi di entrambe le opzioni.

- Con 512 passi, gli episodi risultano più brevi, riducendo i tempi di addestramento. Questa configurazione può favorire strategie più dirette, come il correre rapidamente verso la fine del livello, limitando esplorazioni inutili.
- Con 2048 passi, il modello ha invece più tempo per esplorare il livello, affrontare segmenti complessi o lontani e sviluppare strategie più sofisticate.

In generale, la scelta della configurazione dipende dall'obiettivo dell'allenamento e dalla complessità del livello:

- **512 passi** è adeguato per apprendere comportamenti di base, come saltare, evitare ostacoli o raggiungere una determinata distanza.
- **2048 passi** è più indicato per livelli lunghi o complessi.

Differenze tra le due configurazioni

Le due configurazioni si differenziano anche per i parametri di apprendimento utilizzati:

- **Configurazione 1**: n_steps=512 e learning_rate=0.0000005.
- **Configurazione 2**: n_steps=2048 e learning_rate=0.000005.

Questa differenza riflette un bilanciamento tra esplorazione a breve termine e apprendimento più approfondito. La configurazione con 512 passi utilizza un tasso di apprendimento più basso per garantire aggiornamenti graduali su episodi brevi, mentre quella con 2048 passi utilizza un tasso più alto per accelerare l'apprendimento su sequenze più lunghe.

4.2.1 Addestramento con 512 passi

Durante l'addestramento con una finestra di 512 passi per ogni aggiornamento della rete, il modello ha mostrato un andamento non lineare nelle vittorie, distribuite lungo i 10 milioni di passi come segue:

Passi	Vittorie
1M	1
2M	3
3M	3
4M	14
5M	2
6M	6
7M	9
8M	3
9M	11
10M	2

Analisi dei risultati

1. **Andamento delle vittorie**:
    - L'addestramento ha mostrato un andamento oscillante. Miglioramenti significativi sono stati osservati attorno ai 4 milioni di passi (14 vittorie) e nuovamente ai 9 milioni di passi (11 vittorie).
    - La flessione a 5 milioni e a 10 milioni di passi (2 vittorie in entrambi i casi) potrebbe indicare problemi di stabilità o difficoltà nel consolidare strategie chiave del livello.
2. **Picchi prestazionali e convergenza**:
    - I picchi prestazionali si verificano periodicamente, suggerendo che il modello individua strategie vantaggiose, senza però consolidarle nel lungo termine.
    - La mancanza di convergenza stabile potrebbe dipendere da:
        - **Reward Shaping insufficiente**: Ricompense non sufficientemente specifiche per incentivare il completamento del livello.
        - **Iperparametri subottimali**: La finestra di 512 passi potrebbe aver limitato la capacità del modello di generalizzare strategie su periodi più lunghi.
3. **Implicazioni dei risultati**:
    - Il modello ha mostrato la capacità di ottenere vittorie in modo sporadico, ma senza miglioramenti consistenti nella frequenza delle vittorie.
    - La variabilità suggerisce un potenziale conflitto tra esplorazione e sfruttamento, impedendo al modello di focalizzarsi su strategie consolidate.

Conclusioni intermedie

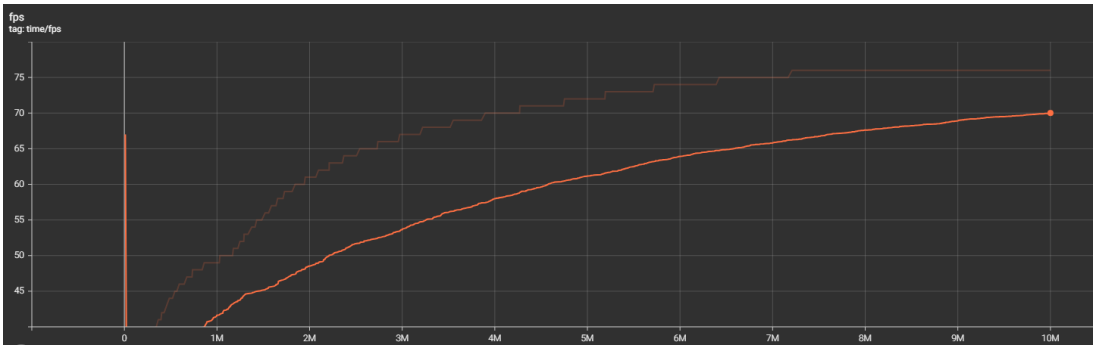
Nonostante alcune vittorie promettenti, l'addestramento con 512 passi non ha portato a una convergenza stabile. Le strategie apprese sembrano temporanee e non sufficientemente robuste per garantire prestazioni consistenti.

4.2.1.1 Grafici di supporto

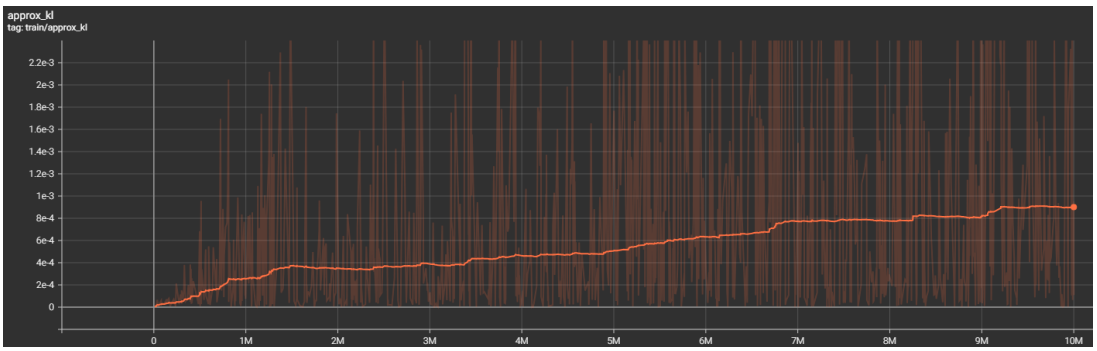
In sintesi, possiamo osservare che:

- **Il modello impara e si adatta:** Molte metriche, come approx_kl, explained_variance, entropy_loss e policy_gradient_loss, mostrano un andamento che indica un progressivo adattamento del modello all'ambiente.
- **L'addestramento è stabile:** Metriche come clip_fraction e clip_range suggeriscono che l'addestramento procede in modo stabile, senza oscillazioni eccessive.

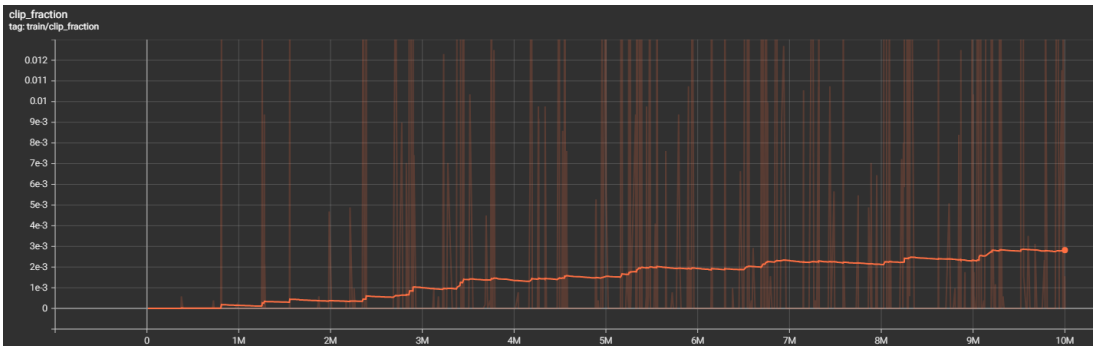
Analizzando i grafici più nel dettaglio:



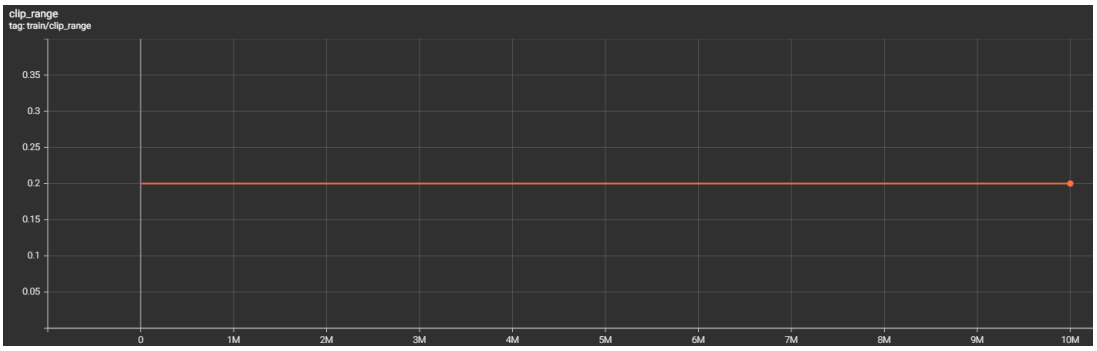
- **time/fps:** Mostra un iniziale miglioramento delle prestazioni (aumento degli FPS), seguito da una leggera stabilizzazione. Questo indica che il modello diventa più efficiente nell'elaborazione dei dati.



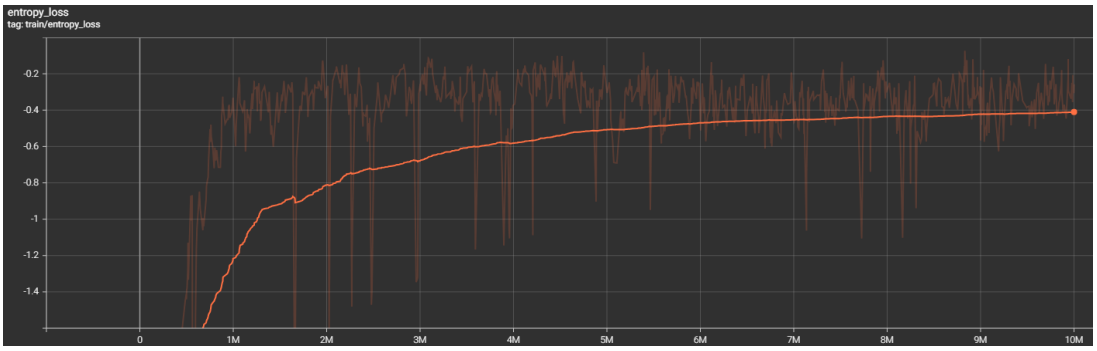
- **approx_kl:** Aumenta gradualmente, suggerendo un continuo adattamento della policy del modello.



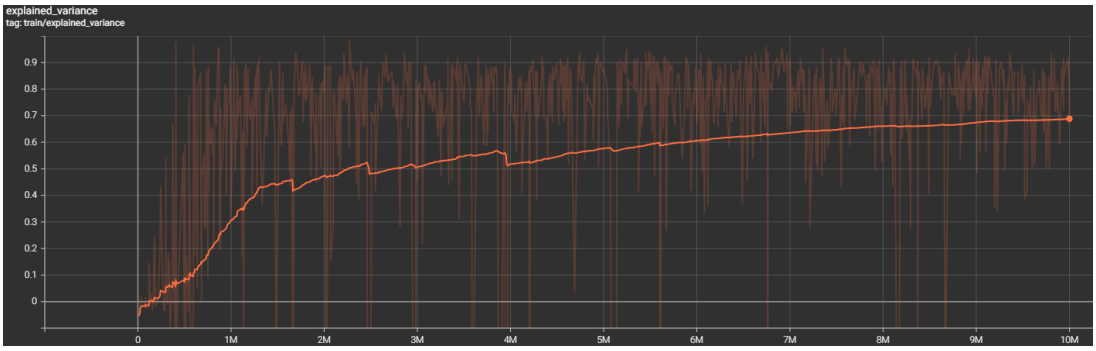
- **clip_fraction:** Si stabilizza dopo una fase iniziale, indicando un apprendimento stabile.



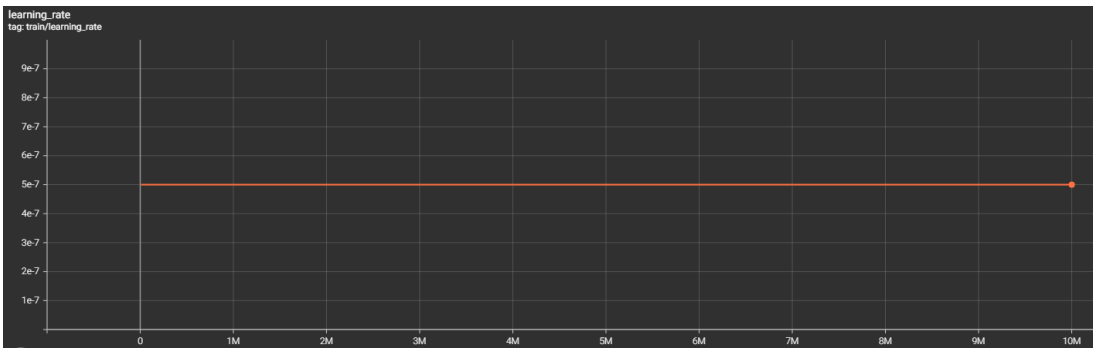
- **clip_range:** Rimane costante, confermando la stabilità dell'apprendimento.



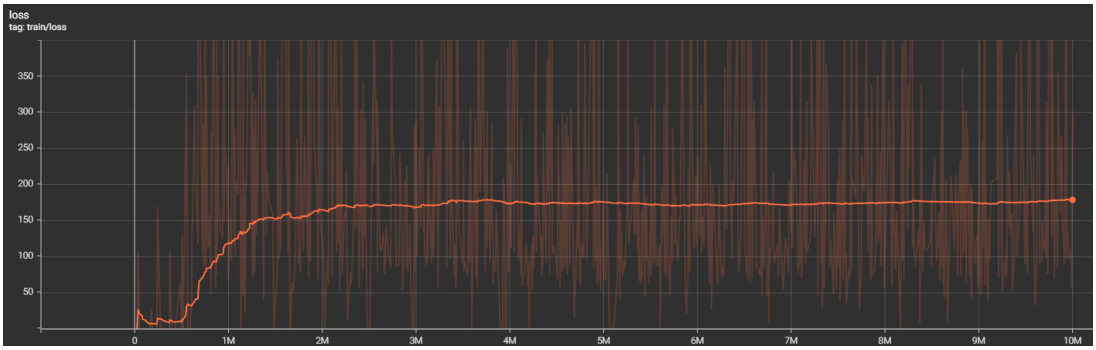
- **entropy_loss:** Mostra un leggero aumento, suggerendo che il modello sta esplorando nuove strategie.



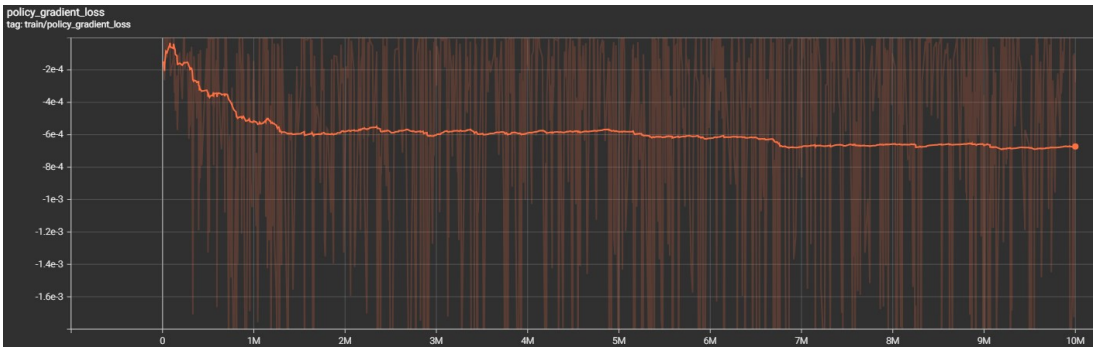
- **explained_variance:** Aumenta costantemente, indicando una maggiore capacità del modello di prevedere le ricompense.



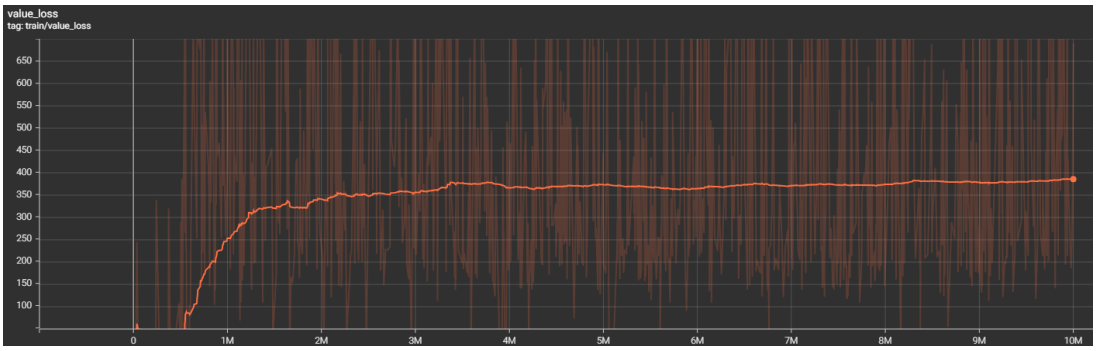
- **learning_rate:** Rimane costante, indicando una velocità di apprendimento stabile.



- **loss:** Presenta un andamento non lineare, con un incremento iniziale seguito da una stabilizzazione.



- **policy_gradient_loss:** Mostra una diminuzione, indicando un efficace aggiornamento della policy del modello.



- **value_loss:** Aumenta progressivamente e poi si stabilizza, suggerendo che il modello sta migliorando la sua capacità di stimare il valore degli stati.

4.2.2 Addestramento con 2048 passi

L'addestramento con una finestra di 2048 passi per ogni aggiornamento della rete non ha prodotto alcuna vittoria durante i 10 milioni di passi, evidenziando difficoltà significative nell'apprendimento di strategie efficaci.

Analisi dei risultati

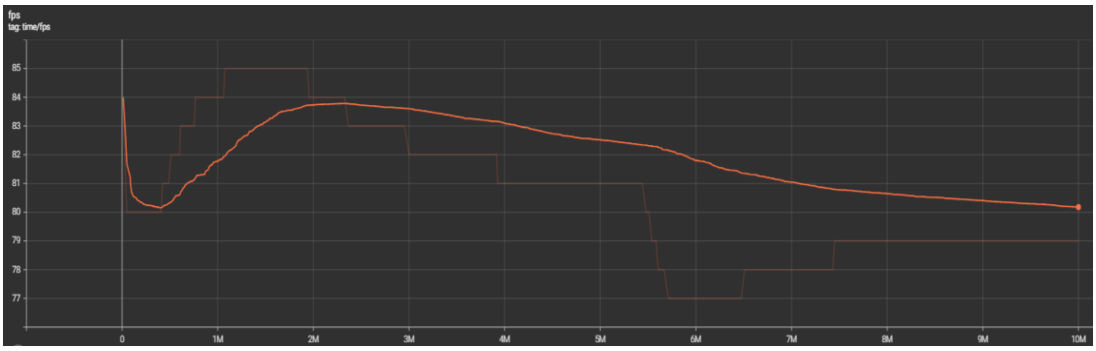
1. **Nessuna vittoria**:
    - L'assenza totale di vittorie suggerisce che il modello non è riuscito ad apprendere strategie utili per avanzare nel livello. Questo risultato potrebbe derivare dalla difficoltà del modello di adattarsi a un ambiente complesso e dinamico come Super Mario Bros, dove sono richieste azioni precise e un apprendimento strategico costante.
2. **Impatto della finestra di aggiornamento**:
    - La finestra più ampia (2048 passi) ha probabilmente ridotto la capacità del modello di adattarsi rapidamente a nuove informazioni. In ambienti complessi come Super Mario Bros, una finestra troppo grande può ritardare l'aggiornamento delle politiche, rendendo il modello meno reattivo ai cambiamenti ambientali.
    - Inoltre, la funzione obiettivo adottata potrebbe essere troppo conservativa. L'algoritmo PPO, limitando drasticamente i cambiamenti nella politica per garantire stabilità, potrebbe aver ostacolato l'esplorazione di strategie nuove e rischiose, necessarie per superare ostacoli critici.
3. **Implicazioni dei risultati**:
    - Il mancato successo suggerisce che il modello non è stato in grado di bilanciare efficacemente l'esplorazione e lo sfruttamento. Questo squilibrio ha compromesso l'apprendimento, impedendo al modello di consolidare strategie vincenti e adattarsi in modo efficace all'ambiente.

Conclusioni intermedie

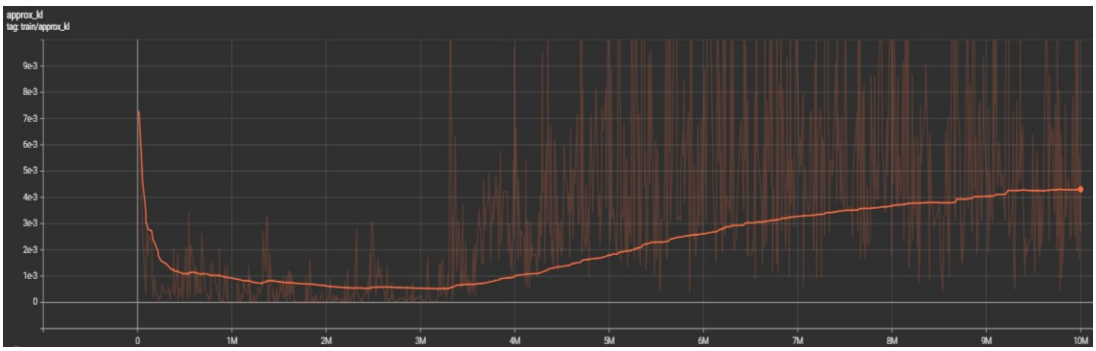
La configurazione con 2048 passi ha dimostrato prestazioni inferiori rispetto a quella con 512 passi. Non è stata osservata alcuna capacità di completare il livello, evidenziando limiti significativi nell'apprendimento del modello con questa configurazione. Questo risultato sottolinea la necessità di ottimizzare ulteriormente gli iperparametri e la funzione obiettivo per favorire l'apprendimento.

4.2.2.2 Grafici di supporto

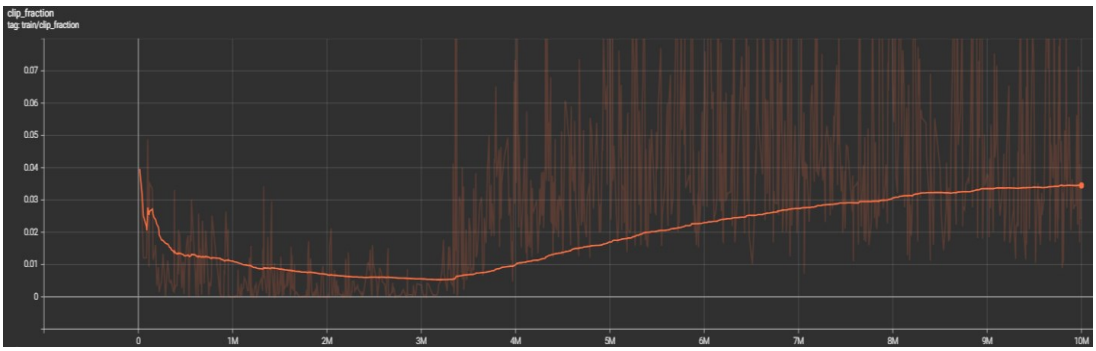
Analizzando i grafici nel dettaglio:



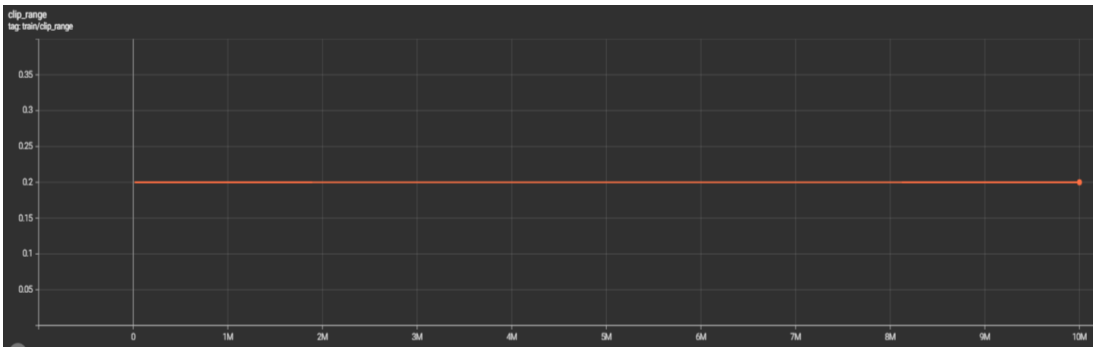
- **time/fps:** Gli FPS inizialmente alti calano, per poi risalire e stabilizzarsi intorno a 84. Successivamente, diminuiscono gradualmente fino a circa 80. Questo andamento può essere attribuito a diversi fattori, come l'aumento della complessità computazionale del modello, l'ottimizzazione del codice e le risorse hardware disponibili.



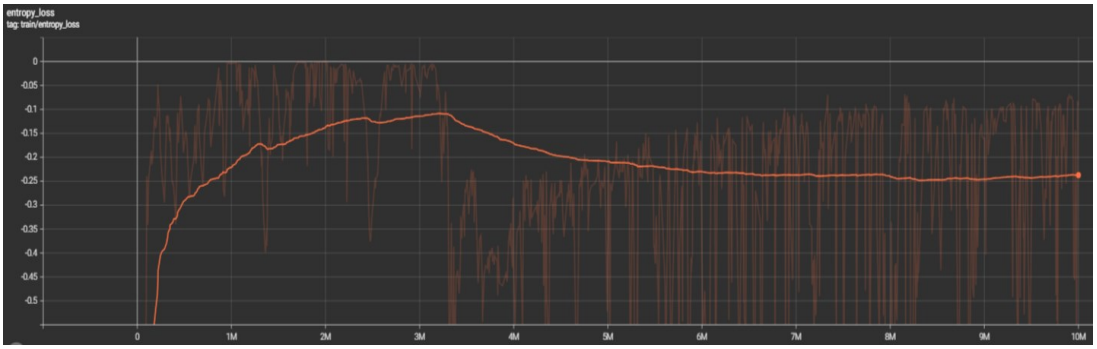
- **approx_kl:** Il valore inizia alto, diminuisce rapidamente e poi aumenta gradualmente. Questo andamento suggerisce che il modello esplora intensamente nelle fasi iniziali, trova una strategia iniziale e la affina gradualmente, senza però raggiungere una convergenza chiara.



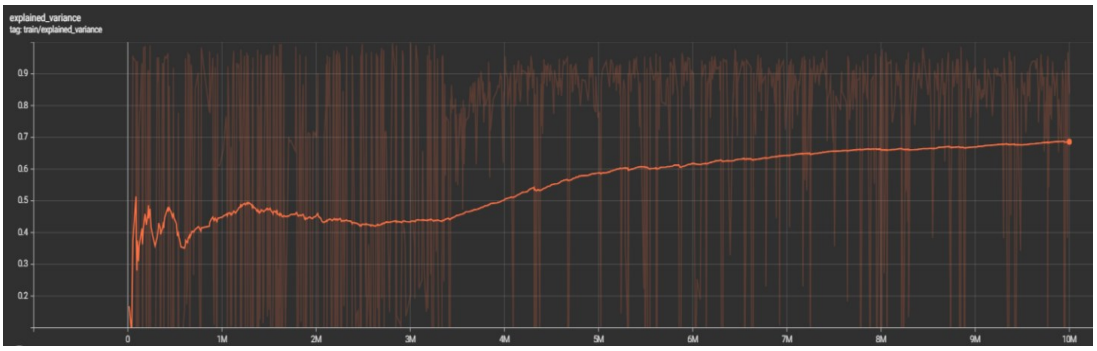
- **clip_fraction:** Il valore inizia alto, diminuisce rapidamente e poi aumenta lentamente. Questo indica che il modello inizialmente applica aggiornamenti significativi alla politica, per poi adattarsi con modifiche più graduali, riflettendo una stabilizzazione progressiva.



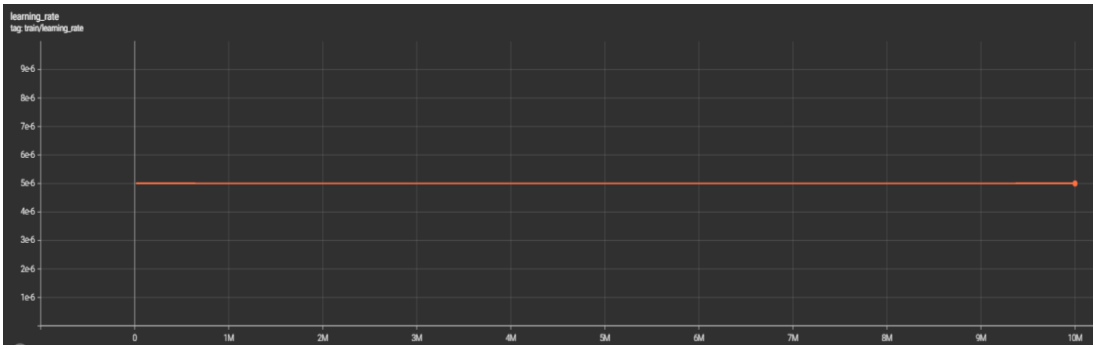
- **clip_range:** Rimane costante durante tutto l'addestramento, confermando che il modello ha mantenuto una stabilità generale nell'apprendimento.



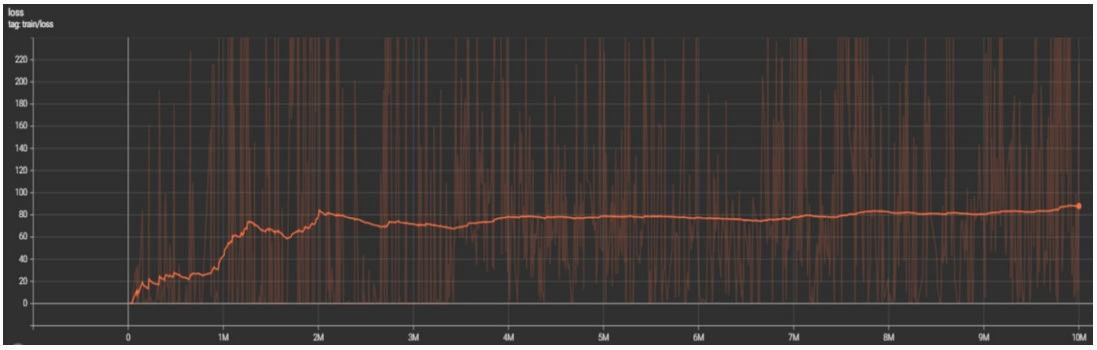
- **entropy_loss:** Il valore inizia basso, aumenta e poi diminuisce gradualmente, stabilizzandosi verso la fine. Questo comportamento riflette una fase iniziale di maggiore esplorazione seguita da una progressiva focalizzazione su strategie specifiche.



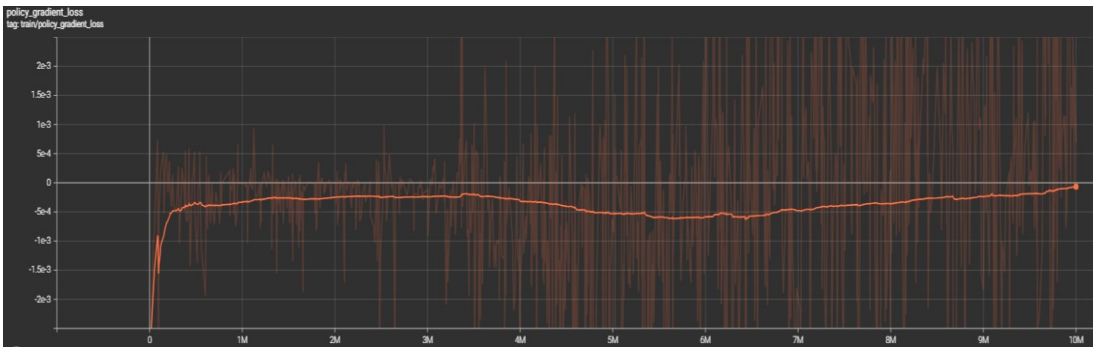
- **explained_variance:** Il valore aumenta gradualmente, indicando che il modello migliora la sua capacità di prevedere ricompense e prendere decisioni più informate.



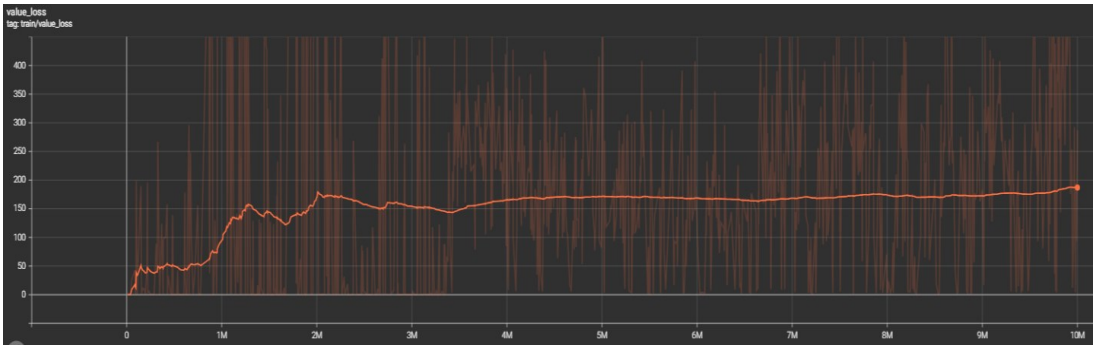
- **learning_rate:** Rimane costante durante tutto l'addestramento, indicando che il modello ha mantenuto una velocità di apprendimento stabile.



- **loss:** Dopo un inizio basso, il valore aumenta rapidamente per poi stabilizzarsi. Questo suggerisce che il modello stia ancora affinando le sue stime, senza però raggiungere un errore minimo ottimale.



- **policy_gradient_loss:** Dopo un inizio negativo, il valore aumenta rapidamente e si stabilizza intorno a zero. Questo andamento indica che gli aggiornamenti della politica sono diventati progressivamente più efficaci.



- **value_loss:** Il valore aumenta nelle fasi iniziali e poi si stabilizza, suggerendo che il modello ha raggiunto una certa stabilità nella stima dei valori degli stati, pur con margini di miglioramento residui.

4.2.3 Confronto tra le due configurazioni

Il confronto tra le due configurazioni evidenzia differenze significative nelle prestazioni e nella stabilità dell'apprendimento:

Configurazione	Passi totali	Vittorie totali	Stabilità	Note principali
512 passi	10M	Variabile (54)	Media	Oscillazioni marcate nelle prestazioni.
2048 passi	10M	0	Nessuna	Nessun miglioramento significativo.

- La configurazione con **512 passi** ha mostrato un maggiore potenziale nell'ottenere vittorie, totalizzando 54 vittorie durante i 10 milioni di passi. Tuttavia, le prestazioni non sono risultate costanti, caratterizzate da oscillazioni e una mancanza di convergenza stabile.
- La configurazione con **2048 passi**, invece, non ha prodotto alcuna vittoria. Questo risultato sottolinea una possibile incompatibilità tra questa configurazione e la complessità del problema da risolvere, evidenziando la necessità di ulteriori ottimizzazioni per sfruttare appieno questa finestra di aggiornamento.

Conclusioni Il modello configurato con 512 passi si è dimostrato più efficace nel raggiungere risultati tangibili, anche se con prestazioni instabili. Al contrario, la configurazione con 2048 passi non ha evidenziato alcun miglioramento significativo, suggerendo che una finestra di aggiornamento più grande non sia adatta per questo specifico contesto.

4.3 Prestazioni del modello PPO integrato con YOLOv5

L'integrazione tra Proximal Policy Optimization (PPO) e YOLOv5 ha fornito un approccio innovativo per il controllo dell'agente e il riconoscimento degli oggetti nell'ambiente di gioco.

4.3.1 YOLOv5: Riconoscimento degli oggetti

L'addestramento di YOLOv5 ha prodotto metriche elevate, dimostrando un'efficace capacità di rilevamento degli oggetti:

- **Precisione (P)**: 94,2%

  La precisione misura la percentuale di predizioni corrette rispetto a tutte le predizioni effettuate. Un alto valore di precisione indica che il modello effettua pochi falsi positivi, identificando correttamente gli oggetti senza rilevamenti errati.

- **Recall (R)**: 100%

  Il recall rappresenta la percentuale di oggetti effettivamente presenti che sono stati correttamente identificati dal modello. Un valore di 100% significa che il modello non ha perso nessun oggetto durante il rilevamento (assenza di falsi negativi).

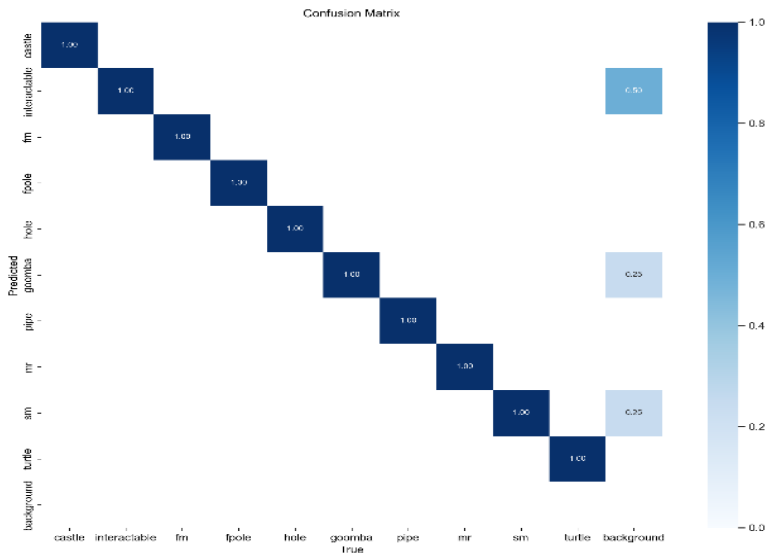
- **mAP@50-95**: 87,6%

  Il mean Average Precision (mAP) è una metrica che combina precisione e recall, calcolata su diverse soglie di confidenza (tra il 50% e il 95%). Un mAP elevato rappresenta una buona capacità del modello di bilanciare precisione e recall su tutte le classi.

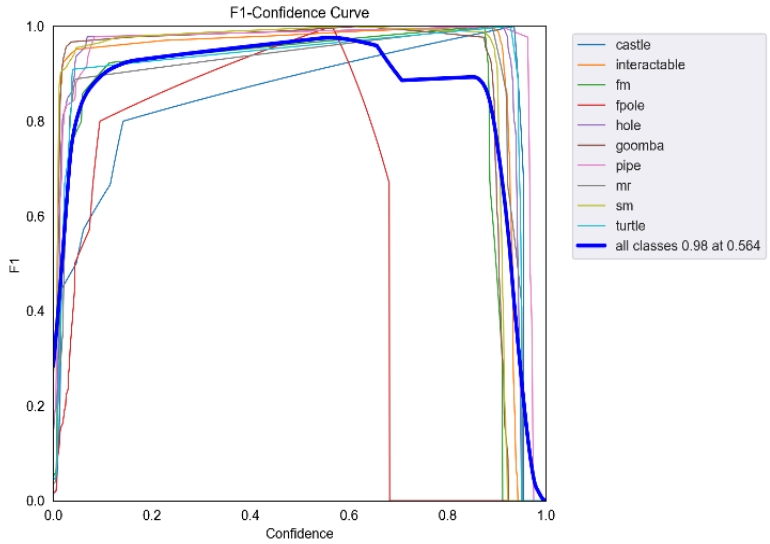
Performance per classe:

- Le classi come *castle*, *goomba* e *turtle* hanno ottenuto recall perfetti (100%).
- La classe *fpole* ha mostrato un mAP@50-95 relativamente basso (54,7%), suggerendo difficoltà nel rilevamento accurato di questa categoria, potenzialmente dovute alla scarsità di dati o alla complessità visiva della classe stessa.

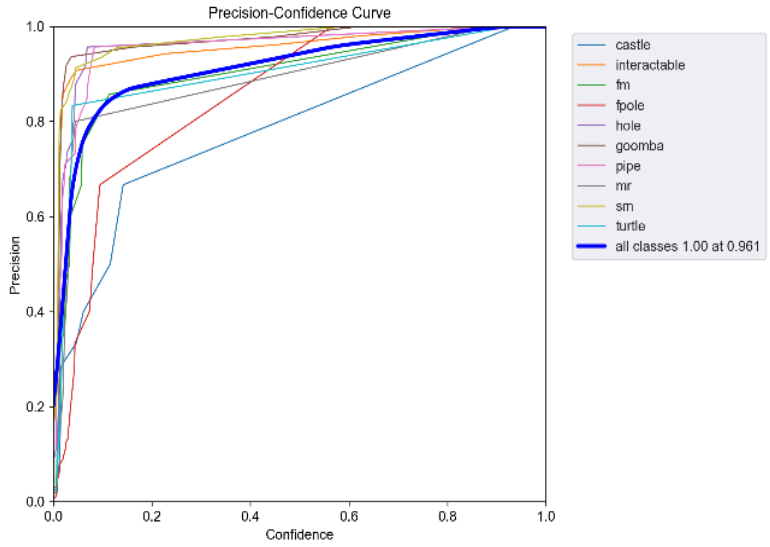
4.3.1.1 Grafici di Supporto



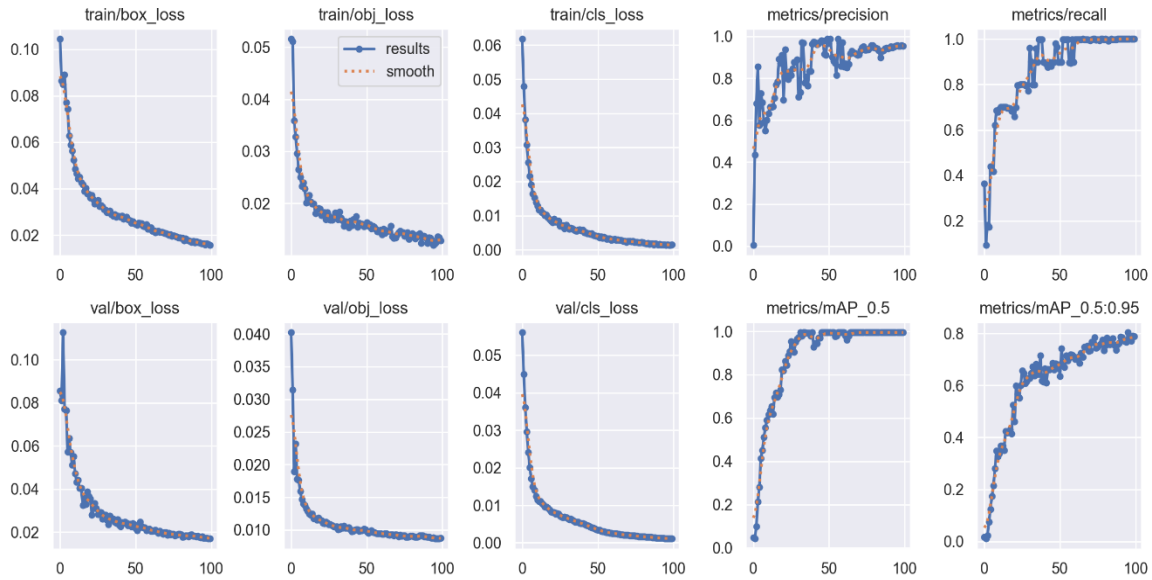
- **Matrice di confusione:** Il grafico evidenzia buone prestazioni generali, con la maggior parte delle previsioni corrette (valori alti sulla diagonale). Tuttavia, sono presenti alcune confusioni, in particolare tra "turtle" e "interactable".



- **Curva F1-Confidence:** Il modello mostra un F1 score elevato per la maggior parte delle classi. Tuttavia, alcune classi come "fm" e "fpole" evidenziano un andamento più irregolare, suggerendo difficoltà nell'identificazione affidabile.



- **Curva Precision-Confidence**: La precision-confidence curve conferma un’ottima precisione nel riconoscimento degli oggetti da parte del modello, con alcune oscillazioni nelle classi più difficili.



- **Andamento delle perdite:** Le curve di perdita (*train/box_loss*, *train/obj_loss*, *train/cls_loss*) mostrano una decrescita costante, indicando un apprendimento efficace. Analogamente, le perdite sul set di validazione (*val/...*) diminuiscono, seppur con qualche oscillazione, suggerendo una buona capacità di generalizzazione. Tuttavia, si osserva un possibile lieve overfitting che potrebbe richiedere ulteriori accorgimenti.

4.3.2 Prestazioni del modello PPO

L'addestramento del PPO ha coinvolto un totale di 10 milioni di passi, con risultati significativi:

- **Explained variance: 91,1%**

  Questo valore indica che il modello è riuscito a spiegare la maggior parte della variazione nei valori della funzione valore. Un explained variance vicino a 1 (o 100%) suggerisce una stima accurata della funzione valore, prevedendo correttamente i valori futuri della ricompensa cumulativa.

- **Policy gradient loss: 0,000987**

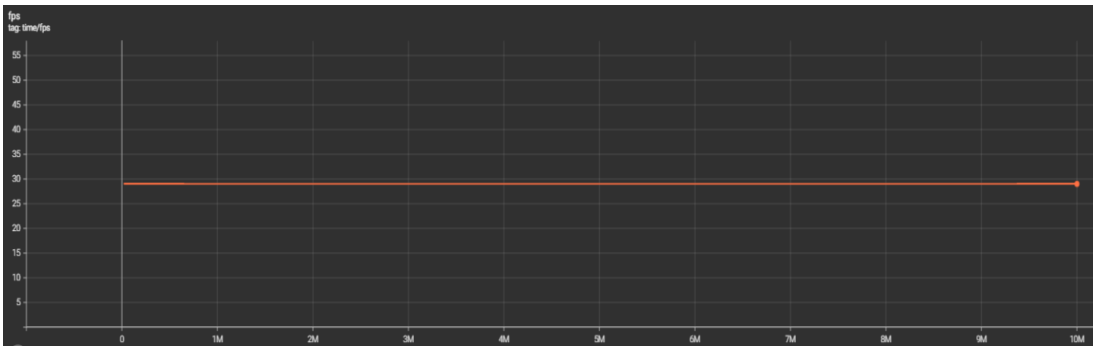
  Questo valore suggerisce un aggiornamento stabile della politica durante l'addestramento. Un policy gradient loss molto basso indica che la politica si è stabilizzata durante l'addestramento, con un apprendimento coerente rispetto all'ambiente.

- **Stabilità dell'apprendimento**

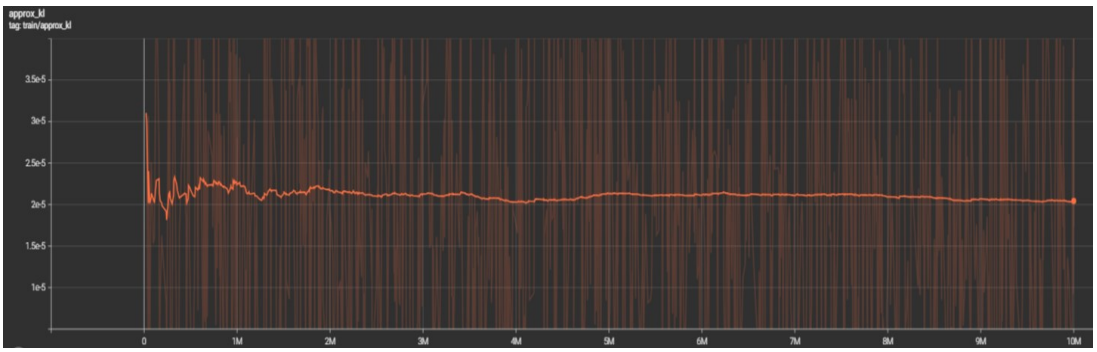
  La bassa perdita totale e l'elevata frequenza di aggiornamenti hanno consentito di mantenere una politica esplorativa, evitando fenomeni di overfitting.

4.3.2.1 Grafici di supporto

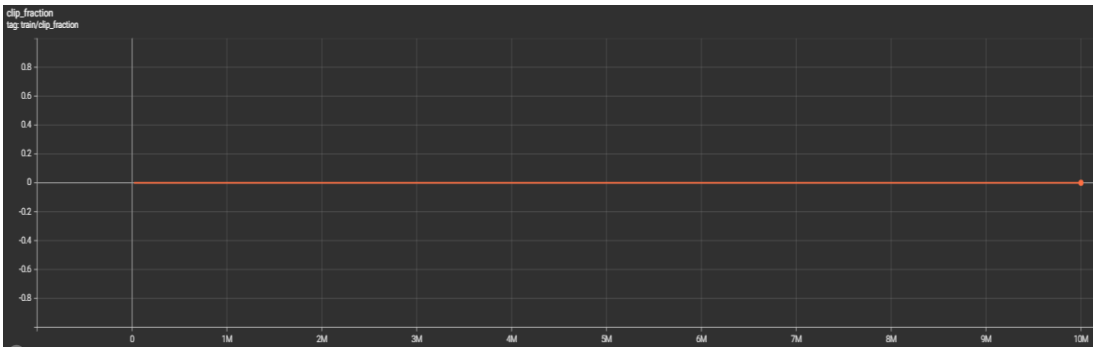
Analizzando i grafici nel dettaglio:



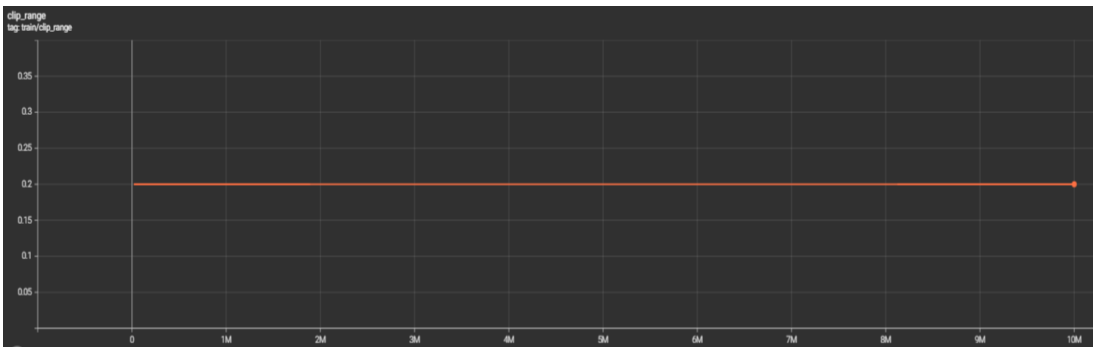
- **time/fps:** Gli FPS rimangono costanti durante l'addestramento, indicando una velocità di elaborazione stabile e invariata.



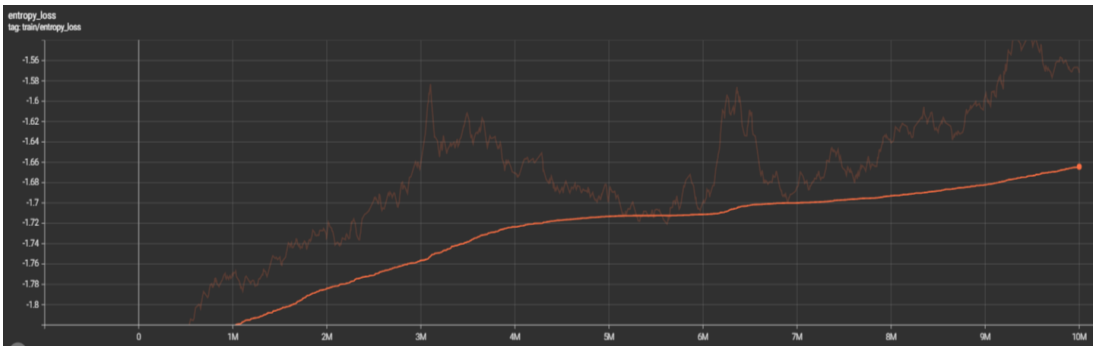
- **approx_kl:** Dopo un picco iniziale, il valore si stabilizza su una soglia bassa e costante, indicando che il modello apprende gradualmente con aggiornamenti ridotti alla policy.



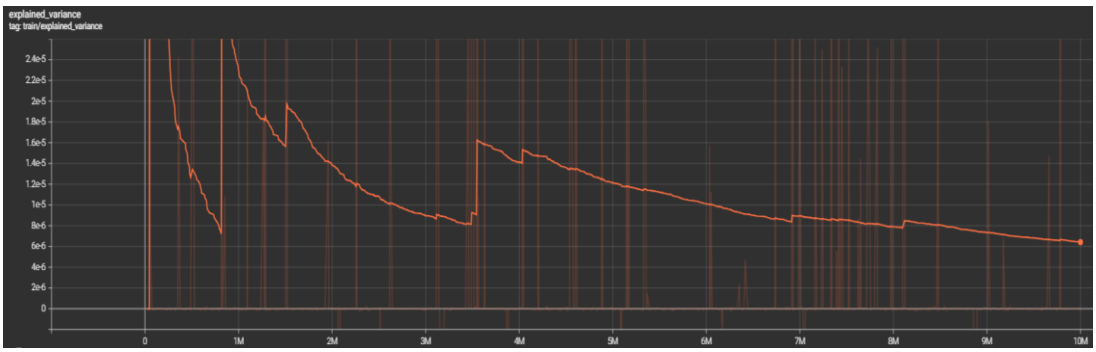
- **clip_fraction:** Rimane costante a zero, suggerendo che nessun aggiornamento alla policy è stato "tagliato", segno di stabilità nell'apprendimento.



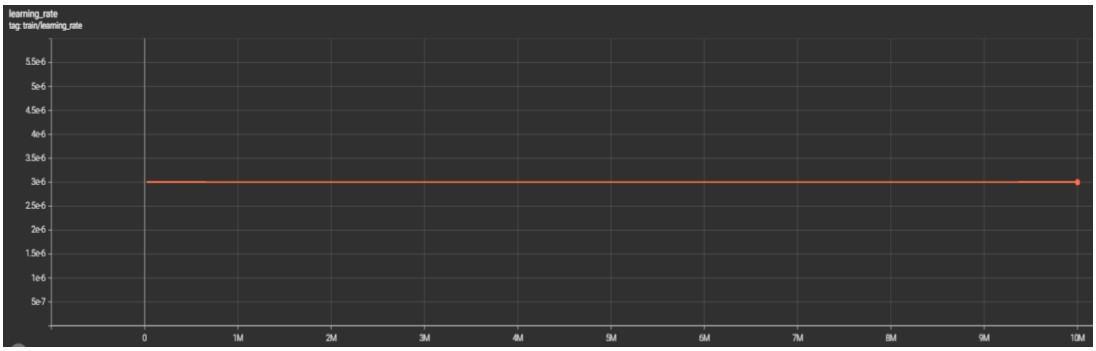
- **clip_range:** Rimane costante per tutta la durata dell’addestramento, confermando la regolarità degli aggiornamenti.



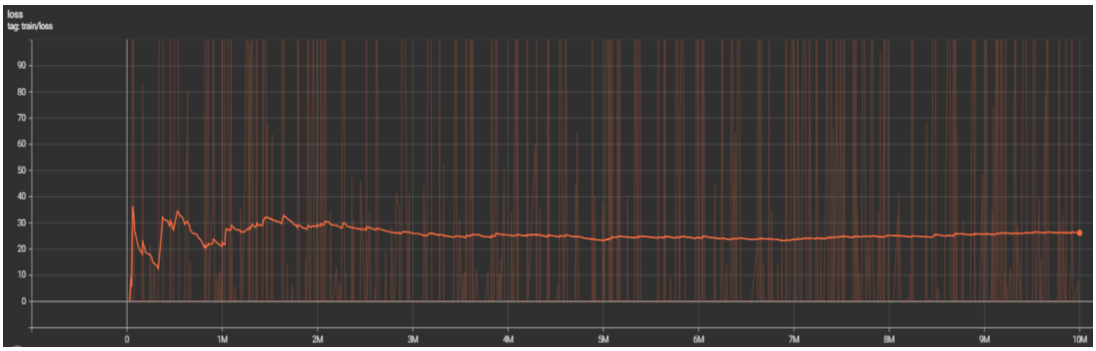
- **entropy_loss:** L'entropia aumenta progressivamente, suggerendo che il modello esplora nuove strategie o affronta difficoltà nell'apprendimento.



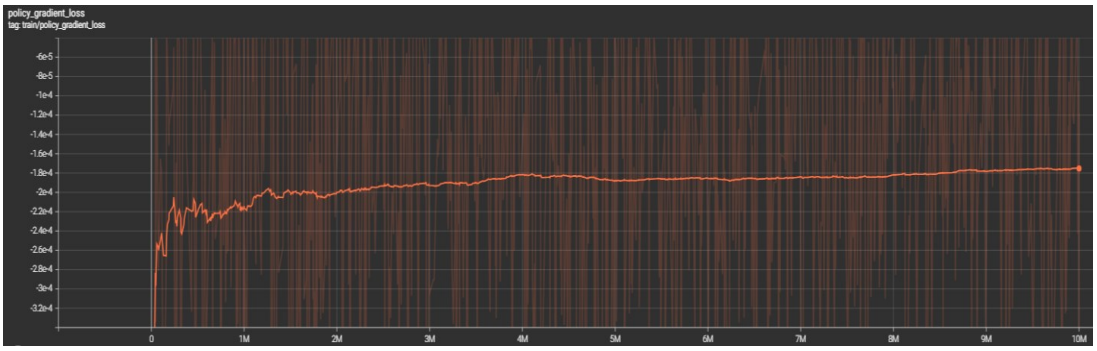
- **explained_variance:** Nonostante un andamento inizialmente positivo, si registrano momenti di instabilità, con una diminuzione del valore durante l'addestramento.



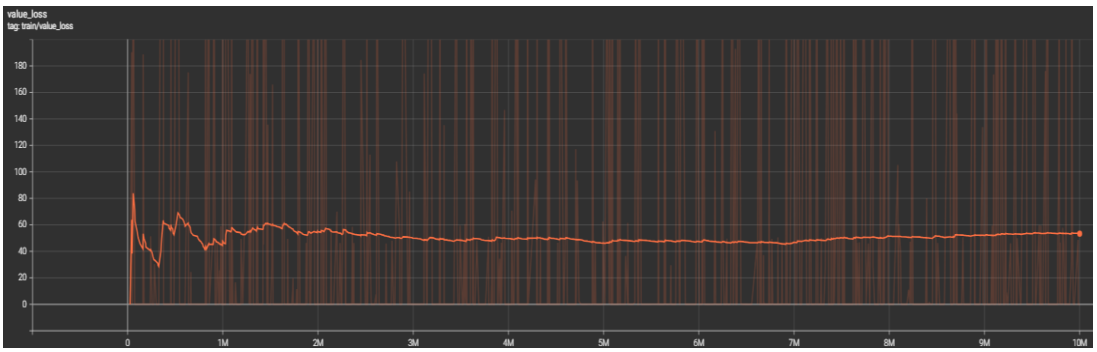
- **learning_rate:** Rimane costante, indicando una velocità di apprendimento stabile.



- **loss:** Dopo un picco iniziale, diminuisce progressivamente e si stabilizza su un valore basso, suggerendo che l'errore complessivo si riduce nel tempo.



- **policy_gradient_loss:** Si stabilizza intorno a un valore negativo, evidenziando che gli aggiornamenti alla policy, in media, migliorano l'apprendimento.



- **value_loss:** Dopo un incremento iniziale, il valore si stabilizza su una soglia moderata, suggerendo un miglioramento costante nella stima dei valori degli stati.

4.3.3 Difficoltà specifiche

Nonostante l'accuratezza del riconoscimento degli oggetti fosse eccellente, il PPO integrato con YOLOv5 non è riuscito a completare alcun episodio con una vittoria, anche dopo oltre 1000 episodi (circa 48 ore di addestramento).

- **Ostacoli complessi**: Il principale limite è stato il salto del terzo tubo (*pipe*), il più alto del livello. L'agente non è stato in grado di eseguire correttamente l'azione richiesta.
- **Traduzione delle informazioni**: Sebbene YOLOv5 fornisse informazioni accurate sull'ambiente, l'agente non è riuscito a tradurre queste informazioni in azioni complesse come il salto, evidenziando limiti nell'integrazione tra riconoscimento visivo e controllo delle azioni.

4.4 Confronto tra DDQN e PPO

L'analisi delle prestazioni dei modelli Deep Double Q-Network (DDQN) e Proximal Policy Optimization (PPO) ha evidenziato differenze significative nei loro comportamenti e risultati nel completamento del livello *SuperMarioBros-1-1-v0*. In questa sezione vengono esaminati i punti di forza, le debolezze e le caratteristiche principali dei due approcci.

4.4.1 Approccio e caratteristiche principali

Caratteristica	DDQN	PPO
Tipo di apprendimento	Basato su funzione Q (value-based)	Basato su politica (policy-based)
Architettura di base	Rete CNN per stimare i valori Q	Rete CNN per apprendere una politica diretta
Obiettivo	Massimizzare Q(s, a)	Massimizzare una funzione di vantaggio A(s, a)
Stabilità	Maggiore stabilità, ma apprendimento più lento	Aggiornamenti più rapidi, ma con instabilità

4.4.2 Risultati comparativi

Metrica	DDQN	PPO (512 passi)	PPO (2048 passi)
Vittorie totali	1 su 1000 episodi	54 in 10M passi	0 in 10M passi
Convergenza	Lenta, ma stabile	Variabile, oscillante	Nessuna
Robustezza delle strategie	Elevata	Moderata	Nessuna

1. **DDQN**:
   - È riuscito a completare il livello con una vittoria su 1000 episodi.
   - Ha mostrato una convergenza lenta ma stabile, con un comportamento dell'agente verso la fine dell'addestramento più affidabile.
   - Le strategie apprese erano robuste e ripetibili, anche se i tempi di addestramento erano significativamente più lunghi rispetto al PPO.
2. **PPO con 512 passi**:
   - Ha ottenuto prestazioni superiori rispetto al DDQN in termini di vittorie totali (54 vittorie in 10 milioni di passi).
   - Tuttavia, il comportamento era altamente oscillante, con una stabilità non garantita. Il modello non è riuscito a consolidare strategie vincenti in modo consistente.
   - Questo evidenzia una maggiore velocità di apprendimento ma una difficoltà nel mantenere strategie robuste.
3. **PPO con 2048 passi**:
   - Non ha raggiunto alcuna vittoria durante l'addestramento, dimostrando gravi difficoltà nel completamento del livello.
   - L'assenza di convergenza evidenzia una configurazione subottimale degli iperparametri per questo ambiente.
   - Questo risultato suggerisce che un aggiornamento basato su una finestra di 2048 passi potrebbe essere inadatto per ambienti dinamici e complessi come *SuperMarioBros-1-1-v0*.

4.5 Confronto tra DDQN e PPO + YOLOv5

Sulla base dei risultati ottenuti:

- Il modello **DDQN** si è dimostrato efficace nel completamento del livello, ottenendo una vittoria in 1000 episodi. Il comportamento verso la fine dell'addestramento era stabile, ma i tempi richiesti per raggiungere tali prestazioni sono stati significativamente lunghi.
- Il modello **PPO + YOLOv5**, nonostante la precisione e il recall elevati ottenuti da YOLOv5 nel riconoscimento degli oggetti, non è riuscito a completare il livello. Le difficoltà con azioni complesse, come il salto, hanno limitato la capacità dell'agente di avanzare nel gioco.
- Tuttavia, l'integrazione di YOLOv5 ha dimostrato un chiaro vantaggio nell'interpretazione semantica dell'ambiente, un aspetto che potrebbe risultare utile per futuri miglioramenti delle capacità dell'agente.

4.6 Confronto tra PPO e PPO + YOLOv5

L'integrazione di YOLOv5 con l'algoritmo Proximal Policy Optimization (PPO) è stata progettata per migliorare le capacità dell'agente grazie a un'interpretazione semantica più dettagliata dell'ambiente di gioco. Questa sezione confronta le prestazioni di PPO senza YOLOv5 e PPO con YOLOv5, analizzando vantaggi, svantaggi e impatti dell'integrazione.

4.6.1 Differenze metodologiche

Caratteristica	PPO (solo)	PPO + YOLOv5
Input del modello	Frame raw	Frame raw + bounding box degli oggetti
Elaborazione visiva	Nessuna	Identificazione oggetti con YOLOv5
Numero di canali di input	1 (scala di grigi)	2 (scala di grigi + bounding box)
Richiesta computazionale	Media	Alta

L'integrazione di YOLOv5 ha introdotto un secondo canale di input per il modello PPO, fornendo informazioni strutturate sugli oggetti rilevanti nell'ambiente, come nemici (*goomba*, *turtle*) e ostacoli (*pipe*, *hole*).

4.6.2 Risultati comparativi

Metrica	PPO (solo)	PPO + YOLOv5
Vittorie totali (10M passi)	54	0
Explained variance	~91%	~89%
Policy gradient loss	~0.000987	~0.001045
Capacità di salto complesso	Moderata	Insufficiente

1. **Prestazioni generali**:
    - **PPO senza YOLOv5** è stato in grado di ottenere 54 vittorie durante l'addestramento, dimostrando una certa capacità di apprendere strategie vincenti, sebbene con stabilità limitata.
    - **PPO + YOLOv5**, nonostante l'integrazione di un potente sistema di riconoscimento visivo, non ha raggiunto alcuna vittoria durante i test su 10 milioni di passi.
2. **Spiegazione della varianza**:
    - L'**explained variance** del PPO + YOLOv5 è risultata leggermente inferiore rispetto al PPO senza YOLOv5, suggerendo che il modello integrato ha avuto difficoltà a sfruttare pienamente le informazioni aggiuntive fornite da YOLOv5.

3. **Problematiche specifiche**:
    - Una delle principali limitazioni riscontrate nel modello PPO + YOLOv5 è stata l'incapacità di superare ostacoli complessi, come il terzo tubo del livello, che richiede una sequenza di salti precisa. Nonostante YOLOv5 fornisse bounding box accurati per identificare i tubi, il modello non è riuscito a tradurre queste informazioni in azioni efficaci.

4.6.3 Analisi dei risultati

- **Vantaggi di PPO + YOLOv5**:
    - L'integrazione di YOLOv5 ha migliorato l'interpretazione semantica dell'ambiente, fornendo una rappresentazione dettagliata e strutturata degli oggetti presenti.
    - Questo approccio potrebbe risultare particolarmente vantaggioso in contesti dove la distinzione visiva tra oggetti è cruciale per ottimizzare le strategie.
- **Svantaggi di PPO + YOLOv5**:
    - L'aumento della complessità computazionale ha rallentato l'addestramento, rendendo meno efficienti le iterazioni rispetto a PPO senza YOLOv5.
    - L'agente non è riuscito a sfruttare appieno le informazioni aggiuntive fornite da YOLOv5, suggerendo che l'integrazione richiede ulteriori ottimizzazioni.
- **PPO senza YOLOv5**:
    - Sebbene meno sofisticato dal punto di vista visivo, il modello PPO senza YOLOv5 ha mostrato una maggiore capacità di apprendimento delle strategie fondamentali per completare il livello.
    - Tuttavia, le vittorie sono risultate distribuite in modo non uniforme, evidenziando difficoltà di stabilità e convergenza.

5. Problemi incontrati e test vari

5.1 Problemi Tecnici e Limitazioni Hardware

5.1.1 Limitazioni del Mac e utilizzo della GPU

Uno dei principali problemi affrontati durante il progetto è legato all'utilizzo della GPU su Mac. Sebbene il sistema fosse dotato di una GPU, il supporto per librerie come PyTorch in modalità GPU era limitato. Questo ha comportato due conseguenze principali:

- **Addestramento di PPO + YOLOv5 senza GPU**: L'assenza di accelerazione hardware ha significativamente prolungato i tempi di addestramento, riducendo la possibilità di eseguire test più approfonditi o iterazioni rapide per ottimizzare il modello.
- **Incompatibilità dei risultati YOLOv5 tra sistemi operativi**: Il training di YOLOv5, inizialmente eseguito su Windows, è stato completato in meno di un'ora, restituendo ottimi risultati. Tuttavia, il trasferimento al sistema Mac si è rivelato impossibile a causa dell'incompatibilità dei risultati generati su Windows. Questo ha reso necessario ripetere l'intero addestramento su Mac, impiegando oltre 24 ore (senza sfruttare la GPU) e ottenendo risultati simili.

Questi problemi hanno evidenziato la necessità di un ambiente hardware più flessibile, spingendo a un utilizzo strategico delle risorse disponibili.

5.2 Problematiche generali

5.2.1 Difficoltà nel salto del terzo tubo

Una delle principali limitazioni riscontrate durante gli addestramenti dei vari modelli è stata l'incapacità di superare un ostacolo critico del livello di gioco: il terzo tubo (*pipe*), il più alto tra quelli presenti. Per cercare di risolvere tale problema, sono stati esplorati due approcci principali.

Tentativi di soluzione

Sono stati effettuati diversi esperimenti per affrontare questo problema:

1. **Definizione di CUSTOM_MOVEMENT**:
   Sono state create sequenze personalizzate di azioni, come passi all'indietro seguiti da salti consecutivi, per aumentare la probabilità di superare l'ostacolo. Tuttavia, questi approcci hanno portato a una rigidità nel comportamento dell'agente, limitando la generalizzazione a altre parti del livello.
2. **Reward shaping mirato**:
   È stato introdotto un sistema di *reward* specifico per incentivare il modello a saltare più frequentemente in prossimità del tubo. Sebbene questa tecnica abbia temporaneamente migliorato il comportamento, non ha risolto il problema in modo stabile, evidenziando una limitazione intrinseca nella capacità del modello PPO di apprendere azioni complesse in questa configurazione.

Conclusioni

Considerando i risultati ottenuti, non ancora ottimali, si è deciso di non implementare soluzioni forzate per non compromettere l'obiettivo principale del progetto: addestrare un modello tramite RL evitando interventi diretti che forzino comportamenti specifici.

5.3 Problemi con l'Addestramento di DDQN

5.3.1 Tempi di addestramento lunghi

L'addestramento del modello DDQN, pur avendo prodotto risultati promettenti (incluso il completamento del livello con una vittoria), è stato caratterizzato da tempi molto lunghi per raggiungere la convergenza. Questo problema è stato amplificato dalla mancanza di tecniche di accelerazione e da un'esplorazione iniziale inefficiente.

Nonostante le ottimizzazioni tentate, i tempi di addestramento sono rimasti una limitazione significativa per il modello DDQN.

5.4 Problemi con il Dataset YOLOv5

5.4.1 Bilanciamento delle classi

Un problema minore, ma significativo, è stato il bilanciamento delle classi nel dataset YOLOv5. Alcune classi avevano un numero limitato di esempi annotati, portando a un *mAP@50-95* relativamente basso.

Tentativi di soluzione

- Sono stati aggiunti manualmente esempi per alcune classi sottorappresentate, ma il numero complessivo di immagini nel dataset è rimasto un limite intrinseco.
- Sono state esplorate tecniche di *data augmentation* su immagini esistenti, come ritagli e rotazioni, ma i risultati sono stati limitati dal contesto specifico del gioco.
- Per affrontare la difficoltà nell'individuare correttamente la classe *goomba*, sono stati utilizzati più dataset disponibili in rete. Tuttavia, questi non si sono dimostrati sufficienti. Pertanto, è stato creato un dataset personalizzato combinando immagini dai dataset preesistenti con altre generate attraverso catture durante il gioco.
- L'utilizzo di diversi dataset ha richiesto un processo di uniformazione, aggiungendo le classi mancanti e rianalizzando le annotazioni esistenti. Questo ha portato alla creazione di un "DatasetFinale" più coeso e bilanciato.

5.5 Lezioni Apprese e Limiti del Progetto

1. **Rilevanza dell'hardware**:
   Il progetto ha dimostrato quanto sia cruciale un hardware adeguato per progetti che richiedono un'elevata potenza computazionale. La mancanza di GPU utilizzabili su Mac ha limitato notevolmente la velocità di iterazione, penalizzando in particolare il training di PPO + YOLOv5.
2. **Complessità delle azioni in PPO + YOLOv5**:
   Il problema del salto del tubo ha evidenziato i limiti del sistema ibrido PPO + YOLOv5 in ambienti con dinamiche complesse.
3. **Dimensione del dataset**:
   Un dataset più ampio e bilanciato sarebbe stato essenziale per migliorare ulteriormente le prestazioni di YOLOv5, specialmente per classi problematiche come *fpole*.
4. **Approcci alternativi**:
   I test falliti, come CUSTOM_MOVEMENT e il *reward shaping* mirato, si sono rivelati utili per comprendere meglio i limiti degli algoritmi e per esplorare strategie alternative, sebbene non abbiano fornito soluzioni definitive.

Questi problemi, pur rappresentando delle sfide, hanno permesso di acquisire conoscenze fondamentali utili per migliorare gli approcci adottati. Essi rappresentano una base solida per ulteriori sviluppi futuri.

6. Conclusioni

Il progetto ha esplorato l'applicazione di tecniche avanzate di Reinforcement Learning (RL) per affrontare la complessità dell'ambiente di gioco di *Super Mario Bros*. In particolare, sono stati analizzati i modelli Deep Double Q-Network (DDQN) e Proximal Policy Optimization (PPO), con e senza l'integrazione del sistema di visione artificiale YOLOv5. I risultati ottenuti offrono una panoramica delle potenzialità e delle sfide legate all'uso di questi approcci in ambienti dinamici e ricchi di variabili.

6.1 Risultati principali

1. **Deep Double Q-Network (DDQN)**:
    - Ha dimostrato una maggiore stabilità e robustezza rispetto al PPO, completando il livello una volta su 1000 episodi.
    - Tuttavia, i tempi di addestramento si sono rivelati molto lunghi, e il modello non ha mostrato miglioramenti significativi dopo aver raggiunto la convergenza.
2. **Proximal Policy Optimization (PPO)**:
    - In configurazione ottimale (512 passi), il PPO ha mostrato una maggiore capacità di esplorazione rispetto al DDQN, ottenendo 54 vittorie su 10 milioni di passi.
    - La configurazione con 2048 passi, invece, non ha prodotto alcuna vittoria, evidenziando una sensibilità elevata agli iperparametri.
    - Le prestazioni del PPO sono risultate meno stabili rispetto al DDQN, con oscillazioni significative durante l'addestramento.
3. **PPO + YOLOv5**:
    - L'integrazione di YOLOv5 ha fornito un miglioramento significativo nella capacità dell'agente di interpretare visivamente l'ambiente, con metriche elevate nel riconoscimento degli oggetti (precisione del 94,2% e recall del 100%).
    - Nonostante queste potenzialità, il PPO + YOLOv5 non ha completato alcun livello, evidenziando difficoltà nel tradurre le informazioni semantiche in strategie efficaci.
    - L'aumento della complessità computazionale e i tempi di addestramento più lunghi hanno rappresentato ulteriori limitazioni.

6.2 Problemi incontrati

Durante il progetto sono emerse alcune problematiche chiave che hanno influenzato le prestazioni dei modelli:

- **Limitazioni nell'apprendimento delle azioni complesse**:
    - Nessun modello è stato in grado di superare in modo affidabile il salto del terzo tubo, un ostacolo critico nel livello.
- **Sensibilità agli iperparametri**:
    - Il PPO ha mostrato una marcata dipendenza dalla configurazione degli iperparametri, influenzando negativamente la stabilità dell'apprendimento.
- **Restrizioni hardware**:
    - L'assenza di un supporto GPU ottimale su alcune piattaforme ha rallentato l'addestramento, in particolare durante l'implementazione del PPO con YOLOv5.
- **Dataset YOLOv5**:

- Il bilanciamento delle classi e il numero limitato di immagini hanno influito negativamente sul riconoscimento di alcune categorie.

6.3 Sviluppi futuri

I risultati di questo progetto rappresentano un punto di partenza per futuri miglioramenti:

1. **Ottimizzazione degli iperparametri**:
   - Approfondire l'analisi delle configurazioni di PPO (ad esempio, passi per aggiornamento, *learning rate*) per migliorare la stabilità e le prestazioni.
2. **Miglioramento del dataset YOLOv5**:
   - Espandere il dataset con un numero maggiore di immagini e migliorare il bilanciamento delle classi per ottenere un riconoscimento più accurato.
3. **Modelli ibridi**:
   - Esplorare architetture ibride che combinino le capacità esplorative del PPO con la robustezza del DDQN.
4. **Gestione delle azioni complesse**:
   - Sperimentare con *reward shaping* più mirati e tecniche di esplorazione avanzate, come la *curiosity-driven exploration*, per superare ostacoli critici.
5. **Riduzione della complessità computazionale**:
   - Studiare metodi per ottimizzare l'integrazione tra RL e YOLOv5, riducendo i tempi di addestramento senza compromettere le prestazioni.