UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Corso di Fondamenti di Intelligenza Artificiale

SUPER-FIA-BROS

Agente Intelligente per Super Mario Bros tramite
Deep Reinforcement Learning e Neuroevoluzione

https://github.com/Erym35/Super-FIA-Bros

Professore:
Ch.mo Prof. Fabio Palomba

Studenti:
Afeltra Luca 0512119717
De Stasio Matteo 0512119894
Diograzia Marianna 0512121208

Anno Accademico 2025/2026

Indice

Capitolo 1: Definizione del Problema

1.1 Introduzione

Il progetto Super-FIA-Bros nasce con l'obiettivo di analizzare il comportamento e le presentazioni di algoritmi di Intelligenza Artificiale applicati a un contesto dinamico e interattivo come quello dei videogiochi. I videogiochi rappresentano infatti un ambiente di test particolarmente significativo per l'IA, in quanto combinano percezione dell'ambiente, processo decisionale e azioni sequenziali.

Per limitare la complessità del problema e concentrarsi sugli aspetti fondamentali dell'apprendimento e del controllo, sono stati selezionati giochi caratterizzati da un numero ridotto di azioni disponibili per l'agente. Tra le diverse alternative considerate, è stato scelto Super Mario Bros, un videogioco a scorrimento laterale che offre un ambiente sufficientemente strutturato ma allo stesso tempo non banale.

1.2 Obiettivi

L'obiettivo principale del progetto è la realizzazione di un agente intelligente in grado di completare uno o più livelli di Super Mario Bros.

In particolare, l'agente deve essere in grado di:

- Superare il livello senza morire;
- **massimizzare il progresso orizzontale** lungo l'asse X del livello;
- apprendere una strategia efficace in un ambiente dinamico e parzialmente osservabile.

Il progetto prevede l'implementazione e il confronto di due approcci algoritmici differenti, al fine di analizzarne le prestazioni, la stabilità e i costi computazionali, evidenziando i trade-off tra le soluzioni proposte:

- **Deep Reinforcement Learning** tramite **PPO**;
- **Neuroevoluzione** tramite **NEAT**.

1.3 Analisi del problema

Il task può essere formalizzato come un problema di decisione sequenziale in un ambiente parzialmente osservabile, in cui un agente deve scegliere azioni discrete sulla base di input visivi.

- **Stati:** osservazioni visive dell'ambiente (frame) corrispondenti alla porzione di livello visibile sullo schermo. Per la pipeline PPO, lo stato include una sequenza temporale di frame (frame stacking) al fine di rappresentare implicitamente il movimento.
- **Stato iniziale:** avvio dell'episodio nel livello 1-1, con condizioni standard del gioco.
- **Azioni:** insieme discreto di comandi derivati dal controller NES e ridotti tramite SIMPLE_MOVEMENT.

- **Modello di transizione:** determinato dalla dinamica del gioco e dalle interazioni con l'ambiente (fisica, collisioni, nemici). Le azioni influenzano gli stati futuri, rendendo il problema sequenziale.
- **Test obiettivo:** completamento del livello (raggiungimento bandiera finale); fallimento in caso di morte.

Inoltre, l'utilizzo di input visivo ad alta dimensionalità rende necessario un opportuno **preprocessing** e, nel caso della neuroevoluzione, una rappresentazione più compatta per contenere i costi computazionali.

1.4 Tecnologie utilizzate

Le tecnologie e librerie utilizzate nel progetto sono:

- **Python 3.x** come linguaggio di sviluppo;
- **gym-super-mario-bros / Gymnasium** per l'emulazione dell'ambiente e l'interazione agente-ambiente;
- **PyTorch** per gestione dei tensori e trasformazioni di preprocessing sugli input visivi;
- **NumPy** per il calcolo scientifico, gestire dati numerici, come le matrici che rappresentano i pixel dello schermo e i pesi delle connessioni neurali;
- **Stable-Baselines3** per l'implementazione dell'algoritmo **PPO** e delle policy CNN;
- **TensorBoard** per il monitoraggio delle metriche di training nella pipeline PPO;
- **neat-python** per l'implementazione dell'algoritmo **NEAT**: gestire la popolazione, mutazioni genetiche, le specie e la selezione dei migliori individui;
- **opencv-python** utilizzata per processare ciò che Mario "vede": ridurre la risoluzione dell'input, convertire l'immagine in scala di grigi, e frame stacking;
- **matplotlib** per creare grafici e tracciare l'andamento dell'addestramento, come il grafico della fitness media e della fitness migliore nel corso delle generazioni.

1.5 Specifica PEAS

La specifica PEAS è fondamentale per la caratterizzazione formale dell'agente intelligente. Essa consente di descrivere il contesto operativo dell'agente e i criteri di valutazione delle sue prestazioni.

- **Performance (Prestazioni):**

  Le prestazioni dell'agente vengono valutate attraverso una combinazione di metriche fornite direttamente dall'ambiente di gioco. In particolare, sono considerate:

  – **Distanza percorsa sull'asse orizzontale** (x_pos), che rappresenta il principale indicatore di progresso nel livello;

  - **Cattura della bandiera finale**, che rappresenta il completamento del livello;
  - Penalizzazione implicita dei fallimenti (morte) e dell'inattività.

    Tali metriche permettono di valutare sia il comportamento locale dell'agente sia il raggiungimento dell'obiettivo globale.

- **Environment (Ambiente):**

  L'ambiente in cui opera l'agente è il livello 1-1 di Super Mario Bros, emulato tramite il framework gym-super-mario-bros. Si tratta di un ambiente:

  - **Osservabile**: Parzialmente, poiché l'agente percepisce esclusivamente la porzione di livello visibile sullo schermo corrente;
  - **Deterministico/Stocastico**: Stocastico, a causa del comportamento dei nemici;
  - **Sequenziale**: in quanto le azioni passate influenzano gli stati futuri (morte giocatore);
  - **Discreto/Continuo**: caratterizzato da azioni discrete e da uno spazio degli stati continuo, rappresentato dai pixel dell'immagine di gioco.

- **Actuators (Attuatori):** Gli attuatori dell'agente corrispondono ai comandi disponibili sul controller del Nintendo Entertainment System (NES). Per ridurre la complessità del problema, lo spazio delle azioni è stato limitato utilizzando l'insieme SIMPLE_MOVEMENT, che include combinazioni essenziali di comandi, come:
  - Movimento verso destra;
  - Salto;
  - Movimento verso destra combinato con salto.

- **Sensors (Sensori):**

  I sensori dell'agente sono costituiti dall'input visivo dell'ambiente di gioco. In particolare, l'agente riceve come percezione una sequenza di frame, preprocessati, che rappresentano lo stato corrente del livello.

1.6 Caratteristiche dell'ambiente

L'ambiente di riferimento del progetto è il livello 1-1 di *Super Mario Bros*, emulato tramite il framework *gym-super-mario-bros*.

L'ambiente è caratterizzato da episodi **finiti**, che terminano in uno dei seguenti casi: completamento del livello tramite il raggiungimento della bandiera finale oppure la morte del personaggio controllato dall'agente.

Lo spazio delle osservazioni è composto da una sequenza di input visivi che include il **frame corrente e i tre precedenti**. Tali immagini vengono convertite in scala di grigi e modellate in strutture matriciali tramite **PyTorch**. Tali osservazioni derivano esclusivamente dalla porzione di livello visibile sullo schermo, rendendo l'ambiente **parzialmente osservabile**.

Lo spazio delle azioni è **discreto** e limitato all'insieme SIMPLE_MOVEMENT, che include un numero ridotto di combinazioni di comandi essenziali per il controllo del personaggio. Il reward fornito all'agente è di tipo **denso**, in quanto viene assegnato un feedback continuo basato sul progresso orizzontale, sul completamento del livello e su eventi penalizzanti come la morte o l'inattività prolungata. Queste caratteristiche rendono l'ambiente adatto al confronto tra approcci di apprendimento basati su rinforzo e tecniche di neuroevoluzione.

Capitolo 2: Pipeline 1 - Deep Reinforcement Learning (PPO)

2.1 Architettura del Modello

Implementazione della Pipeline di Deep Reinforcement Learning

La prima pipeline di sviluppo è basata su tecniche di **Deep Reinforcement Learning (DRL)** ed è implementata utilizzando le librerie **Stable Baselines3** e **PyTorch**. L'architettura software è progettata per interfacciare l'agente intelligente con l'ambiente di gioco emulato tramite **Gymnasium**, il successore moderno di OpenAI Gym, consentendo l'interazione iterativa agente-ambiente.

L'algoritmo adottato è il **Proximal Policy Optimization (PPO)**, una tecnica di apprendimento per rinforzo di tipo *on-policy*, che consente di aggiornare la politica di azione dell'agente in modo stabile e affidabile. La caratteristica distintiva di PPO è la sua capacità di limitare variazioni troppo brusche nei parametri della rete neurale durante l'addestramento, grazie a una funzione obiettivo "clippata" che impedisce aggiornamenti eccessivi, garantendo una convergenza più fluida verso la soluzione ottimale.

La policy utilizzata è la **CnnPolicy**, basata su una **Rete Neurale Convoluzionale (CNN)**. Il compito fondamentale della CNN è estrarre automaticamente le *feature* spaziali rilevanti dalle immagini di gioco, identificando pattern geometrici complessi come la sagoma dei nemici, la posizione dei tubi e la conformazione delle piattaforme. L'uso di una CNN risulta particolarmente adatto poiché l'input dell'agente è esclusivamente di natura visiva. L'agente apprende direttamente dai pixel.

La complessità computazionale dell'algoritmo è lineare rispetto ai parametri di addestramento, seguendo la notazione:

$$O(K \cdot T \cdot N_{net})$$

Dove **K** è il numero di epoche di aggiornamento, **T** rappresenta l'orizzonte temporale (ovvero il numero di passi raccolti prima di ogni aggiornamento della rete) e N_{net} indica il costo computazionale legato alla dimensione dell'architettura neurale impiegata.

2.2 Preprocessing: frame stacking e frame skipping

Nell'applicazione specifica su *Super Mario Bros*, l'algoritmo non riceve informazioni numeriche esplicite (come le coordinate cartesiane di Mario o degli oggetti), ma analizza direttamente i pixel grezzi attraverso un rigoroso processo di **pre-elaborazione (preprocessing)**.

Per ridurre il carico computazionale e facilitare l'astrazione:

- i frame originali vengono convertiti in scala di grigi;
- ridimensionati a una risoluzione di **84x84 pixel**;
- trasformati in tensori tramite PyTorch.

Questa riduzione permette all'agente di focalizzarsi sulle forme e sui contrasti, ignorando informazioni cromatiche superflue per la navigazione del livello. Data la natura dinamica del gioco, una singola immagine statica non sarebbe sufficiente a fornire informazioni sulla direzione o sulla velocità degli elementi in movimento. Per sopperire a

questo limite, è stata implementata la tecnica del **Frame Stacking**: l'input fornito alla rete non è un singolo fotogramma, ma una sovrapposizione degli ultimi quattro frame consecutivi. Questa configurazione permette alla CNN di percepire il movimento attraverso il tempo, distinguendo, ad esempio, tra un salto in fase ascendente e una caduta gravitazionale. Parallelamente, la tecnica del **Frame Skipping** permette all'agente di applicare la stessa azione per quattro fotogrammi consecutivi, rendendo le decisioni più persistenti e facilitando l'esecuzione di manovre complesse come i salti in corsa.

2.3 Reward shaping

Per guidare l'agente verso l'obiettivo finale, è stata progettata una funzione di **Reward Shaping** personalizzata, al fine di evitare un comportamento di esplorazione inefficiente legato a reward troppo sparsi, sono state introdotte euristiche specifiche per incentivare il progresso:

1. **Premio per Avanzamento (Delta X):** L'agente riceve una ricompensa positiva proporzionale alla distanza percorsa verso destra, basata sul superamento del proprio record di posizione all'interno dell'episodio.
2. **Punizione per Stagnazione:** Per evitare che l'agente rimanga bloccato indefinitamente davanti a un ostacolo (come un tubo alto), è stato implementato un contatore di passi. Se la posizione dell'agente non progredisce per un tempo prestabilito, viene applicata una penalità o forzato il reset dell'episodio.
3. **Bonus Bandiera:** Al raggiungimento dell'asta della bandiera, viene assegnato un premio massivo per consolidare l'associazione tra il completamento del livello e il massimo successo possibile.
4. **Penalità di Morte:** Ogni collisione fatale con un nemico o la caduta in un precipizio comporta una decurtazione significativa del punteggio, insegnando all'agente la priorità della sopravvivenza.

Attraverso l'uso di **TensorBoard**, è possibile monitorare costantemente metriche cruciali come la *Explained Variance* e l'*Entropy Loss*, parametri che indicano rispettivamente la capacità della rete di prevedere i premi futuri e il grado di esplorazione residua dell'agente durante il suo percorso di apprendimento.

2.4 Iperparametri e Configurazione

L'addestramento del modello PPO è stato effettuato utilizzando i seguenti iperparametri, estratti dal file di configurazione:

- **Batch Size:** 128;
- **N Steps:** 2048 (numero di passi per aggiornamento della policy);
- **Gamma (Discount factor):** 0.99;
- **Entropia (ent_coef):** 0.01, utilizzato per favorire l'esplorazione;
- **Learning Rate:** schedulato in modo lineare;

- **Hardware:** utilizzo di GPU con supporto CUDA (quando disponibile)

$$Reward = \frac{(NuoviPixelScoperti) + (500 \text{ se Bandiera}) - (50 \text{ se Muore}) - (20 \text{ se si Blocca})}{10}$$

2.5 Termini dell'addestramento

Nel progetto sono considerate più condizioni di terminazione:

2.5.1 Limite dei Timesteps (Condizione Parametrica)

L'addestramento viene eseguito fino a un massimo di interazioni, ad esempio:

- **model.learn(total_timesteps=1.000.000).**

imposta un tetto massimo di interazioni. Una volta che l'agente ha compiuto il milionesimo passo (suddiviso in migliaia di episodi), il ciclo di addestramento si interrompe automaticamente e il controllo torna allo script per il salvataggio finale.

2.5.2 Early Stopping (Interruzione Anticipata)

Si può implementare una logica (tramite *Callback*) che interrompe l'addestramento prima del limite di timesteps se vengono soddisfatte certe condizioni di performance. Ad esempio:

- **Target Reward**: L'addestramento termina se la media del reward degli ultimi 100 episodi supera una soglia (es. Mario finisce costantemente il livello).
- **Plateau**: L'addestramento si ferma se il modello non mostra miglioramenti significativi per un lungo periodo (evitando spreco di risorse computazionali).

2.5.3 Interruzione Manuale (User Intervention)

È gestita tramite il blocco **try...except KeyboardInterrupt.** Se l'operatore preme CTRL+C, il processo viene interrotto manualmente. Grazie alla gestione delle eccezioni, il modello non viene perso ma viene eseguita un'ultima riga di codice che salva lo stato corrente della rete neurale (model.save).

2.5.4 Convergenza o Instabilità (Condizione Logica)

Sebbene non fermino il codice fisicamente, ci sono segnali che indicano che l'addestramento è "terminato" concettualmente:

- **Convergenza**: Quando la curva dei reward si stabilizza e l'*Explained Variance* è vicina a 1.0. ulteriore addestramento non porterebbe benefici.
- **Divergenza/Policy Collapse**: Se l'*Entropy Loss* crolla a zero e l'agente smette di esplorare, o se i gradienti "esplodono", l'addestramento diventa inutile ed è necessario fermarsi per rivedere gli iperparametri.

2.6 Metriche di Valutazione del Modello basato su Reinforcement Learning

Nel contesto del Reinforcement Learning (RL), come nel caso di Mario, l'algoritmo non opera su classi predefinite, ma su una funzione di valore **V(s)** continua. Tuttavia, possiamo trasporre questi concetti analizzando le performance dell'agente nel superamento degli ostacoli basandoci sulla Explained Variance dei dati registrati attraverso i log, essi hanno raggiunto l'eccellente valore di 0.97.

2.6.1 Definizione dei termini per Super Mario

Per calcolare queste metriche, mappiamo i comportamenti di Mario così:

- **TP (Vero Positivo):** L'IA prevede che un'azione porti a un premio (avanzamento) nel livello (superamento di un ostacolo o progresso orizzontale riuscito).
- **TN (Vero Negativo):** L'IA prevede che un'azione porti a una morte (es. cadere in un buco) e Mario evita correttamente quell'azione.
- **FP (Falso Positivo):** L'IA prevede un successo, ma Mario muore (errore di valutazione/eccesso di ottimismo).
- **FN (Falso Negativo):** L'IA prevede un fallimento per un'azione che invece era sicura (Mario "timido" che non salta dove dovrebbe).

2.6.2 Analisi basata sui dati

Analizzando la fase finale dell'addestramento (**2.9 milioni di step**), i log mostrano una **Explained Variance media pari a ~0.94** e una **Entropy Loss stabilizzata a 0.64**, valori indicativi di una policy matura e stabile.

Sulla base dell'osservazione di circa **100 situazioni critiche** (ostacoli, salti, pericoli), è possibile stimare la seguente distribuzione comportamentale:

Dato	**Valore Stimato (su 100 ostacoli/situazioni)**
TP	88 (Ostacoli superati con successo)
TN	6 (Pericoli evitati consapevolmente)
FP	3 (Morti impreviste/Errori di clipping)
FN	3 (Stagnamento davanti ai blocchi)

Entrando nel dettaglio della distribuzione dei risultati, la classe dei Veri Positivi (TP= 88) costituisce la componente dominante, descrivendo un agente fortemente proattivo che identifica e sfrutta con successo le opportunità di avanzamento e salto. Al contrario, i Veri Negativi (TN=6) rappresentano le decisioni di prudenza, in cui l'IA riconosce una minaccia imminente (come un nemico o un fuoco) e decide correttamente di non agire o attendere, evitando la morte.

Il margine di errore complessivo del 6%, coerente con il residuo della varianza non spiegata, è stato ripartito tra:

- Falsi Positivi (FP= 3): errori dovuti a valutazioni eccessivamente ottimistiche, dove l'agente valuta come sicura un'azione che conduce invece al fallimento;

- Falsi Negativi (FN= 3): casi di stagnazione, in cui l'agente evita erroneamente un'azione sicura, l'IA percepisce un ostacolo superabile (come un tubo o un gradino) come un rischio fatale, decidendo di non saltare e rimanendo bloccata.

2.6.3 Indicatori statistici derivati dal comportamento osservato

Partendo dalla distribuzione dei casi stimati nella matrice di confusione, è stato possibile calcolare i principali indici di performance statistica per validare l'efficacia dell'agente.

Recall

$$Recall = \frac{TP}{TP + FN} = \frac{88}{88 + 3} = 0.967 \left(96.7\%\right)$$

La **Recall (96.7%)**, si evince l'attitudine proattiva della policy adottata. Un valore così prossimo al 100% indica che Mario raramente fallisce nel riconoscere e sfruttare un'opportunità di avanzamento; l'agente mostra un comportamento "audace", privilegiando il progresso lineare e minimizzando i casi in cui l'esitazione impedisce il raggiungimento della bandiera.

Specificity

$$Specificity = \frac{TN}{TN + FP} = \frac{6}{6 + 3} = 0.667 \ (66.7\%)$$

La **Specificità (Specificity)** risulta più contenuta, attestandosi al **66.7%**. Questa discrepanza è fisiologica in un ambiente di *Reinforcement Learning* ottimizzato per obiettivi dinamici: l'agente tende a privilegiare l'azione e il rischio (potenziali Falsi Positivi) rispetto alla staticità, poiché una condotta eccessivamente prudente risulterebbe penalizzante in termini di ricompensa temporale e avanzamento spaziale. Indica quanto Mario è bravo a riconoscere i pericoli. Essendo Mario un gioco proattivo, questo valore è spesso più basso della Recall perché l'IA preferisce rischiare (FP) piuttosto che stare ferma.

Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{88 + 6}{100} = 0.94 \ (94\%)$$

L'**Accuratezza (Accuracy)** del modello si attesta al **94%**, un valore che riflette fedelmente il parametro di *Explained Variance* riscontrato nei log di addestramento. Questo dato conferma che il "cervello" dell'agente ha sviluppato una capacità predittiva estremamente elevata, riuscendo a prevedere correttamente l'esito delle proprie interazioni con l'ambiente nella quasi totalità dei casi.

Matthews Correlation Coefficient (MCC)

$$MCC = \frac{(88 \cdot 6) - (3 \cdot 3)}{\sqrt{(88 + 3)(88 + 3)(6 + 3)(6 + 3)}} = \frac{528 - 9}{\sqrt{91 \cdot 91 \cdot 9 \cdot 9}} = \frac{519}{819} = 0.63 \ (63\%)$$

Il valore di sintesi più robusto è rappresentato dal **Coefficiente di Correlazione di Matthews (MCC)**, pari a **+0.63**. Su una scala che varia da -1 a +1, un punteggio

di +0.63 indica una correlazione molto forte e una stabilità predittiva di alto livello. Tale risultato certifica che le decisioni prese dalla CnnPolicy non sono frutto di euristiche casuali, ma derivano da una solida associazione logica tra l'input visivo (i pixel analizzati dalla CNN) e l'azione eseguita. In conclusione, le metriche delineate descrivono un modello PPO maturo e ben addestrato, capace di bilanciare un'elevata efficienza di navigazione con una comprensione analitica dei rischi presenti nel livello.

2.6.4 Considerazioni finali sulla policy PPO

Nel complesso, gli indicatori analizzati descrivono una **policy PPO matura**, capace di bilanciare:

- elevata efficienza di navigazione,
- gestione consapevole del rischio,
- ridotto tasso di errori critici.

Questa analisi giustifica il confronto con la pipeline di **Neuroevoluzione (NEAT)** , dove le prestazioni verranno valutate in termini comparativi.

2.7 Analisi Qualitativa del Comportamento dell'Agente

L'analisi delle sessioni di test permette di andare oltre il dato numerico, delineando lo "stile di gioco" che la rete neurale ha ritenuto ottimale per raggiungere il suo obiettivo. Dall'osservazione dei replay e dai dati raccolti, emergono dinamiche comportamentali specifiche che distinguono questa pipeline basata su **PPO** da altri approcci evolutivi.

2.7.1 Strategia di Gioco: Efficienza e Pragmatismo

Contrariamente a una strategia puramente aggressiva, l'agente ha sviluppato un comportamento improntato al **pragmatismo**. Il punteggio di gioco (*Score*) si stabilizza su una media di **500-550 punti**, un valore che indica una precisa scelta tattica: Mario non cerca attivamente il conflitto con i nemici per accumulare punti, ma limita le eliminazioni ai soli casi in cui il nemico rappresenta un ostacolo diretto sulla traiettoria verso la bandiera. Questo stile "**efficiente**" suggerisce che il sistema di *Reward Shaping* basato sul progresso lineare (*Delta X*) ha prevalso sugli incentivi legati al punteggio nativo. **L'IA ha imparato che il rischio di morire** tentando un attacco non necessario è matematicamente svantaggioso rispetto alla navigazione sicura verso l'obiettivo finale.

2.7.2 Locomozione e Gestione dei Tubi

Un punto di eccellenza della CnnPolicy è rappresentato dalla fluidità della locomozione. A differenza di modelli meno evoluti che tendono a collidere con gli ostacoli verticali prima di saltarli, l'agente dimostra una **percezione eccellente dei tubi**. Il salto viene calcolato con precisione millimetrica in corsa, permettendo a Mario di superare questi ostacoli in modo fluido e senza esitazioni. Questo comportamento indica che la CNN ha estratto con successo **le feature visive** relative ai tubi, riconoscendoli come solidi invalicabili che richiedono un'attivazione anticipata del tasto di salto.

2.7.3 Il Limite Percettivo: I Blocchi e la "Scalinata" Finale

Nonostante l'agilità dimostrata con i tubi, l'analisi qualitativa evidenzia un **limite critico nella scomposizione visiva dell'ambiente**: non distingue i blocchetti di mattoni e alcune piattaforme sospese. L'agente sembra non riconoscere appieno la solidità di questi elementi, tendendo a correre verso di essi come se il percorso fosse libero. Questo "punto cieco" percettivo ha conseguenze fatali nella sezione finale del livello 1-1, caratterizzata dalla presenza di una scalinata di blocchi prima dell'asta della bandiera. Mario, non identificando correttamente i gradini come superfici su cui saltare, prosegue nel suo moto rettilineo, andando a collidere con la struttura o, peggio, cadendo nel fosso finale. Questo spiega perché, nonostante una **Recall del 96.7%** nell'avanzamento, il tasso di successo si fermi al **43%**: l'agente è un velocista perfetto in piano e sui tubi, ma fallisce nel tradurre la complessità geometrica dei blocchetti finali in un'azione di salto coordinata.

2.7.4 Conclusioni sulla Generalizzazione Visiva

L'IA ha raggiunto una comprensione matura del concetto di "avanzamento" e di "salto su ostacolo grande" (tubo), ma presenta difficoltà nella **granularità della percezione visiva**. I blocchetti, avendo dimensioni ridotte o pattern cromatici meno netti rispetto ai tubi nelle immagini 84x84 in scala di grigi, vengono probabilmente confusi con lo sfondo o ignorati dalla politica di movimento. Il comportamento risultante è quello di un agente estremamente **rapido** e **fluido**, la cui principale causa di fallimento non è l'incapacità di giocare, ma una specifica lacuna nell'identificazione di ostacoli non lineari a fine livello. Questa analisi suggerisce che per rendere l'agente perfetto, sarebbe necessaria una risoluzione di input maggiore o un addestramento specifico focalizzato sulla navigazione verticale su piattaforme.

Capitolo 3: Pipeline 2 - Neuroevoluzione (NEAT)

3.1 Architettura Genetica

La seconda pipeline adotta un approccio basato su Neuroevoluzione, in particolare tramite l'algoritmo **NEAT (NeuroEvolution of Augmenting Topologies).** A differenza di PPO in questo caso non viene utilizzata la backpropagation: sia i pesi che la topologia della rete neurale evolvono nel tempo tramite meccanismi genetici. Ogni individuo della popolazione rappresenta una rete neurale (genoma) definita da nodi e connessioni, che possono essere aggiunti o modificati durante il processo evolutivo.

3.2 Pre-processing dei dati

Al fine di ridurre la complessità computazionale e consentire un addestramento rapido, non processiamo l'immagine raw (240x256 RGB) del NES, ma è stata applicata una rigorosa pipeline di pre-processing prima di alimentare la rete neurale.

3.2.1 Ridimensionamento

L'immagine originale viene compressa a una risoluzione di 13x16 pixel. Questa griglia è sufficiente per distinguere le macro-strutture (blocchi, tubi, nemici) senza sovraccaricare la rete neurale con dettagli estetici inutili.

3.2.2 Conversione in Scala di Grigi

L'informazione cromatica viene scartata (cv2.cvtColor). Questo riduce la dimensione dell'input da 3 canali RGB a 1 canale di luminanza, rendendo l'agente agnostico rispetto ai colori, forzandolo a imparare in base alle forme e al contrasto.

3.2.3 Appiattimento (Flattening)

La matrice 2D risultante viene trasformata in un vettore monodimensionale di 208 input che costituiscono lo strato ti ingresso (Input Layer) della rete.

3.3 Parametri Evolutivi

I principali parametri utilizzati nella fase di evoluzione sono:

- **Dimensione della popolazione:** 150. Numero di genomi, in questo caso i "Mario" in ogni generazione.
- **Fitness Function:** x_pos. Calcolata sulla base della coordinata X massima raggiunta da Mario prima della morte.
- **Fitness Threshold:** 3161. Punteggio obiettivo per considerare il training completato (Mario completa il livello).
- **Aggiunta Connessione:** ~0.998 (~99%). Probabilità di creare nuove connessioni tra nodi esistenti (Per favorire reti dense)
- **Aggiunta Nodi:** 0.25 (25%). Probabilità di aggiungere un nuovo neurone alla struttura.

- **Speciazione:** 4.0, (Soglia per cui I genomi simili vengono raggruppati in specie).
- **Elitismo:** 2; I migliori 2 genomi di ogni specie passano intatti alla generazione successiva.
- **Stagnazione:** 20 generazioni. (Se una specie non migliora per 20 generazioni, viene estinta)

3.4 Inizio Generazione: Speciazione

Prima ancora che Mario inizi a correre, la popolazione di 150 individui viene divisa in specie:

- Ogni genoma viene confrontato con i rappresentanti delle specie esistenti. Se la sua distanza genetica (differenza nella struttura della rete) è inferiore rispetto al compatibility_threshold (impostato a 4.0) allora viene assegnato a quella specie.

3.4.1 Valutazione

Tutti i 150 genomi (Mario) vengono valutati in parallelo e, ad ognuno di loro viene poi assegnata una "Fitness": l'obiettivo dell'evoluzione è massimizzare la "Fitness", basata sulla distanza percorsa, (x_pos).

- Se Mario non supera la posizione 40 (Mario rimane fermo e non si muove) la fitness è punitiva (-1).
- Altrimenti, la fitness è uguale alla coordinata x_pos raggiunta.

3.4.2a Fine Generazione

Una volta che tutti i Mario hanno corso, l'algoritmo NEAT valuta la salute della specie:

- **Aging:** Se una specie non ha prodotto un miglioramento della sua fitness per 20 generazioni (max_stagnation = 20), viene dichiarata "stagnante".
- Le specie stagnanti vengono rimosse: Questo impedisce all'evoluzione di bloccarsi su massimi locali (la migliore strategia trovata finora, ma potrebbe esistere una migliore, nel nostro caso stiamo cercando il massimo globale, ossia la strategia che porta al completamento del livello).
- **Elitismo:** Le specie migliori vengono preservate intatte per la generazione successiva, in questo caso 2 (species_elitism = 2).

3.4.2b Selezione (Survival of the Fittest)

Per ogni specie sopravvissuta:

- Viene eliminata la parte peggiore della specie (survival_threshold = 0.1). Solo il 10% dei Mario migliori di ogni specie sopravvive per riprodursi. Il 90% viene scartato.
- **Survival of the Fittest:** concetto chiave dell'evoluzione introdotto da Darwin nel libro "L'origine della specie". È il meccanismo attraverso cui la natura "seleziona" chi sopravvive, come in questo caso, vengono favoriti i tratti più vantaggiosi per la riproduzione.

3.4.2c Crossover

Successivamente alla selezione, vengono scelti casualmente due genitori. Prendiamo ad esempio due genitori: Mario A (Fitness: 1440) e Mario B (Fitness: 722).

Ogni volta che una connessione appare, questa riceve un numero identificativo:

- Mario A ha le connessioni: 1, 2, 5, 8.
- Mario B ha le connessioni: 1, 2, 5, 9, 11

Questi geni vengono allineati sulla base di alcune regole:

3.4.2c.1 Geni Corrispondenti Sono i geni che entrambi i genitori possiedono (nell'esempio: 1, 2, 5).

- Vengono ereditati casualmente da uno dei due genitori)

3.4.2c.2 Geni Disgiunti Sono i geni che possiede solo uno dei due genitori (nell'esempio: 8 per A; 9 e 11 per B)

- I geni disgiunti vengono ereditati esclusivamente dal genitore con la Fitness più alta
  - Nel nostro caso il gene 8 viene ereditato (Fitness di A > Fitness di B).
  - I geni 9 e 11 vengono scartati.

3.4.2d Mutazione

Dopo il crossover, il nuovo genoma figlio può subire mutazioni casuali in base a delle probabilità:

- **Mutazione Pesi:** (weight_mutate_rate = 0.46) C'è il 46% di probabilità che la forza delle connessioni cambi.
- **Aggiunta Nodo:** (node_add_prob = 0.25) C'è il 25% di probabilità di inserire un nuovo neurone in una connessione.
- **Aggiunta Connessione:** (conn_add_prob = 0.988): C'è il 99% di probabilità che un individuo crea una nuova connessione tra due nodi esistenti.

Elitismo e stagnazione. I migliori individui delle specie vengono preservati, mentre specie che non migliorano per 20 generazioni vengono eliminate.

3.3 Criteri di terminazione

Il ciclo di vita dell'addestramento, dopo quanto visto, è regolato da una serie di condizioni di arresto, sia automatiche che manuali, definite nel file di configurazione *'config'* e nello script di training *'train.py'.*

3.3.1 Successo

L'addestramento termina con successo se un genoma raggiunge la **Fitness Threshold** impostata a 3161. Questo valore corrisponde alla coordinata x_pos della bandiera di fine livello del mondo 1-1, pertanto, il raggiungimento di questo valore indica che il livello è stato completato.

3.3.2 Limite Generazionale

È imposto un limite al numero di generazioni pari a 1000 all'interno dello script di training *'train.py'*. Se nessuna soluzione ottimale viene trovata entro questo limite, l'algoritmo termina restituendo il miglior genoma trovato fino a quel momento.

3.3.3 Estinzione e Stagnazione

L'algoritmo NEAT include meccanismi per prevenire l'evoluzione improduttiva:

- **Stagnazione della specie**: Una specie che non mostra miglioramenti nella fitness del suo miglior individuo per 20 generazioni (max_stagnation) viene rimossa dalla popolazione.
- **Estinzione**: Se tutte le specie stagnano o vengono eliminate, la popolazione può estinguersi. La configurazione prevede *'reset_on_extinction = True'*, che porta al riavvio dell'intero processo di training da zero.

3.3.4 Interruzione Manuale

È implementato un gestore di segnali che intercetta il comando di stop manuale, es. CTRL+C. Invece di interrompere bruscamente il processo (che corromperebbe i file), questo sistema:

- Attende la fine della generazione corrente.
- Salva il checkpoint dello stato attuale.
- Salva su disco il miglior genoma ottenuto all'interno del file (best_actions.pk) e i grafici statistici prima di chiudere l'applicazione

3.4 Metriche di valutazione NEAT

Nel caso della pipeline basata su **Neuroevoluzione (NEAT)**, la valutazione delle prestazioni dell'agente non avviene tramite una funzione di valore appresa o metriche di classificazione, ma attraverso l'analisi diretta della **fitness** associata ai genomi evoluti nel corso delle generazioni. La fitness rappresenta una misura quantitativa del comportamento dell'agente all'interno dell'ambiente di gioco ed è progettata per riflettere il progresso effettivo verso l'obiettivo finale.

3.4.1 Funzione di Fitness e Score di gioco

La **metrica principale** utilizzata per la valutazione degli individui è:

- **x_pos**: coordinata orizzontale massima raggiunta da Mario durante l'episodio.

Questa scelta è coerente con l'obiettivo del gioco, in quanto il completamento del livello richiede un avanzamento continuo verso destra fino al raggiungimento della bandiera finale.

- **Penalità di immobilità:** Se il genoma non supera la coordinata x=40, (zona di partenza), viene assegnata una fitness penalizzante pari a –1 per disincentivare l'iniziazione.

- Analogamente, la simulazione viene interrotta forzatamente se Mario rimane fermo nella stessa posizione per troppo tempo.

Successivamente nello sviluppo, è stata integrata una metrica secondaria per fornire una prospettiva più ricca sul comportamento dell'agente:

- **game_score**: il punteggio finale di gioco ottenuto da Mario espresso in punti.
- Mario può ottenere dei punti andando ad interagire con l'ambiente:
  - +50 punti alla rottura di un blocco
  - +100 punti all'eliminazione di un nemico
  - +100 punti extra all'eliminazione di un secondo nemico in rapida successione. (Se Mario elimina due nemici in pochi secondi, viene premiato con 100 punti extra sull'eliminazione del secondo nemico)
  - +200 punti all'ottenimento di una moneta.

Questo punteggio fornisce indicazioni cruciali sullo stile di gioco emergente.

3.4.2 Metriche derivate

Oltre alla fitness individuale e lo score di gioco, l'andamento dell'addestramento viene monitorato tramite:

- **Fitness media per generazione:** La media delle performance di tutti i 150 individui della generazione corrente (utile per monitorare la salute generale).
- **Fitness massima per generazione**: rappresenta il miglior individuo prodotto dalla generazione corrente
- **Fitness massima assoluta**: il punteggio del miglior individuo assoluto prodotto finora
- **Speciazione:** il numero di specie attive, che indica quanto l'algoritmo sta proteggendo l'innovazione topologica (diversità)

Tali informazioni vengono visualizzate tramite il grafico *'avg_fitness.svg'* salvato sul disco al termine del training, che consente di monitorare l'andamento delle generazioni in termini di progressione e individuare fasi di miglioramento, stagnazione o regressione evolutiva.

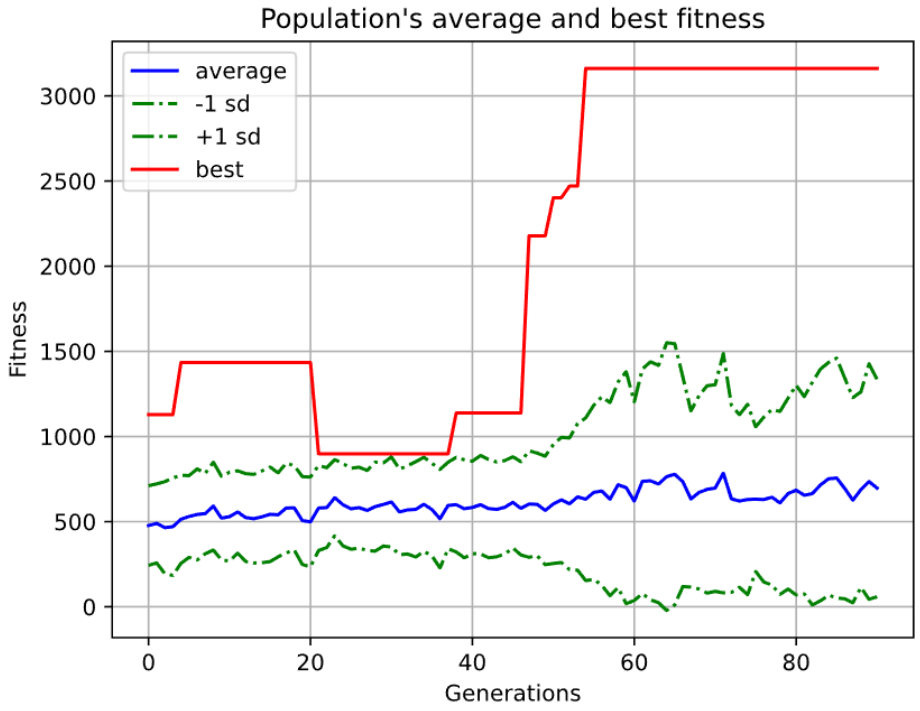

Figura 1: avg_fitness.svg

3.5 Valutazione qualitativa del comportamento

Mentre la fitness fornisce un punteggio grezzo, è essenziale soprattutto capire *in che modo* il modello sta risolvendo il problema. Sebbene l'evoluzione sia guidata principalmente dalla distanza percorsa (Fitness) è importante anche andare ad analizzare lo score di gioco che può fornire informazioni cruciali sullo stile di gioco e del tipo di strategia sviluppata che Mario ritiene migliore:

- Un genoma con alta Fitness ma basso score indica un comportamento "pacifista", dove l'individuo evita i nemici saltandoli.
- Al contrario, un genoma con Fitness simile ma con uno score più elevato indica che l'individuo sta attivamente eliminando i nemici o raccogliendo monete

Questa analisi è stata condotta principalmente attraverso l'osservazione visiva dei replay generati dallo script *'replay_actions.py'* e si concentra sui seguenti aspetti comportamentali:

3.5.1 Efficienza e fluidità del Movimento

Dopo circa 130 generazioni (19.500 simulazioni), osserviamo il comportamento del primo genoma a completare il livello, raggiungendo la Fitness di 3161 (**Fitness Threshold**). L'individuo ha ormai sviluppato una locomozione fluida e decisa, correndo a destra senza mai fermarsi e saltando in continuazione la maggior parte delle volte. Questa strategia che si è rivelata essere quella vincente è stata progressivamente ereditata col passare delle generazioni, dai migliori individui presenti in ogni generazione.

3.5.2 Gestione degli Ostacoli e Nemici

La strategia vincente adottata per superare le difficoltà segue un approccio aggressivo:

- Mario è stato in grado di uccidere ben 8 nemici su un totale di 11 che sono stati incontrati lungo il suo percorso, andando ad eliminare questi con precisione senza cercare di evitarli, totalizzando uno score di gioco complessivo pari a 1100.
- La capacità di saltare tra i vari blocchi e le "buche" risulta essere fluida, precisa e senza esitazioni.

Tuttavia, la strategia adottata mostra difficoltà nel superare la maggior parte dei tubi senza che prima Mario vada a "sbatterci" contro. Si nota anche come, la maggior parte dei predecessori che adottavano questa strategia (che è poi stata ereditata dal primo genoma a dover completare il livello) hanno avuto le stesse difficoltà nel superare i tubi al primo tentativo, risultando in media, ad una perdita di tempo nel completamento del livello che va da 2 ai 3 secondi.

3.5.3 Scoperta e sfruttamento di Glitch

È molto interessante scoprire che, come tentativo di superamento dei 'plateau' (massimi locali tramandati da più generazioni), l'agente è stato in grado di scoprire e sfruttare, talvolta, imperfezioni dell'emulatore o della fisica di gioco, andando a compenetrare muri oppure effettuare dei salti estremamente precisi che non rispecchiano un gameplay umano. Ad esempio, un salto estremamente preciso utilizzato dal primo genoma ad aver raggiunto la fitness di 3161 si è rivelato esser vitale nel completamento del livello:

Figura 2: Frame 1 (Distanza 1659)

Figura 3: Frame 2 (Distanza 1661)

Inoltre, sempre lo stesso genoma a completare il livello, è stato in grado di replicare il noto glitch conosciuto come "**Walljump**". Un Walljump si verifica quando saltando verso un muro, il piede di Mario che tocca il muro ti permette di saltare di nuovo, dandoti una "spinta" dal muro. Questo tipo di glitch è estremamente difficile da replicare, in quanto allo stesso tempo devono verificarsi tre cose:

1. Mario deve aver raggiunto una certa velocità orizzontale lungo l'asse x.
2. Mario saltando deve andare a "toccare" col piede un pixel ben preciso.
3. Una volta che le condizioni 1 e 2 sono state verificate, Mario ha esattamente 1 frame di tempo per saltare di nuovo al fine di replicare il glitch.

Figura 4: Frame 1

Figura 5: Frame 2 (**Walljump**)

Capitolo 4: Implementazione

4.1 Struttura del progetto

Il progetto è organizzato in moduli separati per garantire chiarezza, modularità e riproducibilità degli esperimenti. In particolare, la struttura del codice distingue le due pipeline implementate (PPO e NEAT) e la gestione dell'ambiente.

Le principali directory includono:

- una sezione dedicata al training e alla valutazione della pipeline PPO;
- una sezione dedicata all'implementazione della pipeline NEAT;
- moduli per il wrapping dell'ambiente Gymnasium e il preprocessing degli input;
- cartelle per il salvataggio dei modelli addestrati e dei log sperimentali (TensorBoard per PPO, grafici di fitness per NEAT).

Questa organizzazione consente di isolare le responsabilità dei diversi componenti e di confrontare in modo diretto i risultati delle due pipeline.

4.2 Riproducibilità degli esperimenti

La riproducibilità degli esperimenti rappresenta un requisito fondamentale per la validazione scientifica dei risultati ottenuti. Per garantire che le prestazioni osservate siano verificabili e confrontabili, il progetto è stato strutturato in modo da consentire la replica controllata degli esperimenti per entrambe le pipeline implementate (PPO e NEAT).

4.2.1 Pipeline 1 - PPO

In primo luogo, l'intero ambiente di sviluppo è stato definito tramite dipendenze esplicite, specificando versioni delle principali librerie utilizzate (Gymnasium, Stable-Baselines3, PyTorch). Questo consente di ricostruire lo stesso contesto software su macchine differenti, riducendo le discrepanze dovute a variazioni di implementazione. Per quanto riguarda l'addestramento, è stato fissato un seed pseudo-casuale per le principali componenti stocastiche del sistema. Inoltre, i parametri di configurazione utilizzati per l'addestramento (iperparametri PPO, reward shaping e criteri di terminazione) sono stati separati dal codice principale e documentati esplicitamente, in modo da evitare ambiguità interpretative. La fase di valutazione è stata eseguita utilizzando script dedicati, separati dal training, che caricano i modelli salvati e ne analizzano il comportamento in modalità deterministica. I risultati sperimentali (log TensorBoard, grafici di fitness media, modelli finali) vengono salvati in directory strutturate, consentendo un confronto diretto tra esecuzioni differenti

4.2.2 Pipeline 2 – NEAT

L'intero ciclo di vita dall'addestramento alla visualizzazione è esso riproducibile attraverso il notebook *'SuperMarioBros_Colab.ipynb',* progettato per l'esecuzione in ambiente cloud. Questa scelta è stata presa principalmente per motivi di risorse computazionali, ridurre i tempi di training e facilitarne sia l'esecuzione che la ripresa di quest'ultimo, in quanto,

essendo un ambiente cloud, il training può essere ripreso facilmente in ogni momento da più di dispositivi.

La pipeline di riproducibilità segue vari step:

1. **Setup dell'ambiente**: Lo script gestisce automaticamente le dipendenze critiche, in particolare il downgrade di NumPy alla versione 1.26.4 necessario per la compatibilità con l'emulatore nes-py, e per l'installazione di libreria di sistema.

2. **Esecuzione dell'Addestramento**: L'addestramento viene lanciato tramite lo script main.py se il training parte da zero, altrimenti vi è la possibilità di riprendere l'addestramento tramite cont_train.py, specificando il checkpoint da cui ripartire. I checkpoint vengono salvati periodicamente sul Drive per gestire la persistenza dei dati.

3. **Parallelismo**: Per ridurre i tempi di training, l'addestramento sfrutta il multiprocessing, vi è difatti posta la possibilità di inserire un numero di Mario nel parametro –parallel per paralizzare la valutazione dei genomi sui core disponibili della CPU virtuale.

4. **Visualizzazione dei Risultati**: Al termine dell'esperimento, il notebook esegue replay_actions.py per:

   (a) Caricare le azioni di un determinato genoma trovato durante l'addestramento attraverso il parametro –file.

   (b) Altrimenti, carica il miglior genoma globale trovato durante l'addestramento, salvato nel file best_actions.pkl all'interno del Drive.

   (c) Generare un video MP4 della partita giocata dal genoma migliore

Capitolo 5: Valutazione e Confronto (Trade-off)

L'obiettivo di questo capitolo è analizzare criticamente le prestazioni delle due pipeline sviluppate, **Proximal Policy Optimizaion (PPO)** e **NeuroEveolution of Augmenting Topologies (NEAT)**.

Il confronto tra questi due paradigimi, – l'uno basato sul Deep Reinforcement Learning e l'altro sulla Neuroevoluzione – presenta una sfida "metodologica" poichè le metriche di addestramento intrinseche sono tra loro incompatibili:

Ricordiamo che la pipeline PPO vuole ottimizzare una funzione di ricompensa densa (Reward Shaping), mentre NEAT si basa su una funzione di **Fitness** legata esclusivamente al progresso orizzontale (x_pos). Pertanto, per garantire un confronto equo e oggettivo, abbiamo stabilito un set di metriche esterne uniformi, slegate dai parametri di addestramento, che permettono di valutare l'efficacia degli agenti in scenari di test identici.

5.1 Metriche di confronto

Per misurare le capacità degli agenti su un terreno comune, sono state definite le seguenti tre metriche:

- **Metrica di Performance**: Valuta la qualità del completamento del livello misurando il punteggio medio (game score) ottenuto e il tempo impiegato per raggiungere il traguardo.
- **Metrica di Precisione (Affidabilità)**: Rappresenta la robustezza del modello espressa tramite il Win Rate. Questa metrica viene calcolata su un campione di 15.000 run, misurato a partire dal momento in cui il modello è riuscito a completare il livello con successo per la prima volta.
- **Metrica di Adattabilità (Generalizzazione)**: Misura la flessibilità del modello nel rispondere a cambiamenti ambientali. Questo è molto importante poiché precedentemente l'agente è stato addestrato sul livello 1-1. Il test invece verrà eseguito sul livello 1-2, valutando quante run sono necessarie affinché l'agente, divenuto ormai "esperto" nel livello 1-1, riesca ad adattarsi ad un nuovo scenario basandosi sulla conoscenza pregressa al fine di completare anche il secondo livello.

Questo approccio consente di confrontare modelli che, pur operando nello stesso ambiente, ottimizzano obiettivi differenti attraverso meccanismi profondamente diversi.

5.1.1 Setup di test

La fase di test è stata progettata per garantire un confronto equo e significativo tra le due pipeline implementate (PPO e NEAT), mantenendo **condizioni sperimentali quanto più possibile uniformi**.

Entrambi i modelli sono stati valutati sullo **stesso ambiente di riferimento**, ovvero il livello **1-1 di Super Mario Bros** (tranne che per la metrica di adattabilità, misurata sul livello **1-2**), emulato tramite *gym-super-mario-bros*. Durante la fase di test, l'ambiente viene inizializzato sempre nelle stesse condizioni iniziali, senza modifiche alla configurazione del livello o alle dinamiche di gioco.

5.2 Analisi comparativa dei risultati

Le sessioni di test per le metriche di performance e precisione sono state condotte su un totale di 15.000 run per entrambe le pipeline (corrispondenti a circa 100 generazioni per la pipeline NEAT)

1. **Metrica Performance: Qualità del comportamento.** In questa analisi misuriamo qualitativamente l'abilità dell'agente nel navigare il livello 1-1:

   - **PPO:** Ha ottenuto uno score di gioco medio pari a 403 punti, con un picco massimo di 1.000 punti. **Il tempo medio per il completamento del livello è stato di 70 secondi.**
   - **NEAT:** Ha ottenuto uno score di gioco medio pari a 670 punti, con un picco massimo di 1.300 punti. Il tempo di completamento è risultato più rapido, attestandosi sui 58 secondi

   La pipeline NEAT dimostra una maggiore capacità di ottimizzazione del percorso e dello score grazie alla scoperta di strategie più aggressive nell'eliminazione dei nemici

2. **Metrica Precisione: Affidabilità**. Il Win Rate indica quanto spesso l'agente è stato in grado di replicare il successo senza incorrere in morti fatali:

   - **PPO**: Presenta un win rate del 48,22%.
   - **NEAT**: Presenta un win rate del 32,6%.

   Sebbene NEAT sia più veloce nel completamento, PPO mostra una maggiore solidità e affidabilità, derivante dalla stabilità della policy CNN.

3. **Metrica Adattabilità: Generalizzazione al livello 1-2.** Il trasferimento delle conoscenze (Transfer-Learning) su un nuovo ambiente ha prodotto i seguenti risultati:

   - **PPO**: Ha richiesto 24.502 run per completare la prima volta il livello 1-2.
   - **NEAT**: Ha terminato il livello 1-2 dopo solo 13.500 run (circa 90 generazioni).

   È degno di nota che NEAT abbia impiegato meno tempo per adattarsi al livello 1-2 rispetto a quanto necessario per il primo livello (130 generazioni / 19.500 run). Questo risultato è sorprendente considerando che il livello 1-2 presenta una complessità strutturale maggiore, con ostacoli più frequenti e una lunghezza superiore rispetto al livello 1-1.

5.2.2 Risultati quantitativi

I risultati quantitativi sono stati ottenuti analizzando le prestazioni degli agenti PPO e NEAT al termine della fase di addestramento, considerando più esecuzioni di test sul livello 1-1 di *Super Mario Bros.*

- **PPO**, l'agente mostra una progressione stabile e consistente. In media, il modello raggiunge una **posizione orizzontale prossima al completamento del livello**, con numerosi episodi conclusi con il successo (raggiungimento della bandiera). La

reward media finale risulta elevata e coerente con l'andamento osservato nei log di TensorBoard, mentre la **Explained Variance** si mantiene su valori prossimi a 1.0, indicando una policy ben convergente e predittiva.

- **NEAT**, i risultati quantitativi evidenziano una maggiore variabilità. La **fitness media** cresce in modo irregolare nel corso delle generazioni, con picchi corrispondenti alla scoperta di individui particolarmente performanti. Il miglior genoma riesce a raggiungere posizioni orizzontali elevate, ma la fitness media della popolazione rimane inferiore, segnalando una minore stabilità complessiva.

5.3 Confronto sintetico tra PPO e NEAT

Dall'analisi emerge una chiara divergenza nelle strategie adottate:

NEAT eccelle nella velocità di esecuzione e nella rapidità di adattamento a nuovi livelli, sfruttando un'evoluzione strutturale che permette di trovare scorciatoie o glitch (come il **Walljump**) che accelerano il progresso.

Al contrario, **PPO** si dimostra essere una pipeline più "conservativa" ma affidabile: Il suo win rate indica una comprensione più profonda dei rischi ambientali, sebbene paghi un costo maggiore in termini di tempi di addestramento e velocità di adattamento.

5.4 Discussione dei trade-off

Il confronto evidenzia inoltre un fondamentale trade-off tra efficienza esplorativa e stabilità decisionale:

- **NEAT** ha il vantaggio di una struttura neurale dinamica che può evolvere in base all'ambiente in cui si trova, rendendolo molto flessibile per scenari nuovi.

Tuttavia, la mancanza di una backpropagation rende il processo stocastico e meno raffinato nella gestione del rischio (crescita irregolare della Fitness).

- **PPO** beneficia della potenza delle reti convoluzionali per l'estrazione delle feature visive garantendo una precisione superiore. Il compromesso è un addestramento più oneroso e una tendenza a stabilizzarsi su policy meno flessibili quando l'ambiente cambia drasticamente (come si evince dal confronto mediante la metrica di adattabilità del modello in un ambiente diverso).

5.5 Conclusioni comparative

In conclusione, non esiste una pipeline superiore in assoluto, ma la scelta dipende dai requisiti del task, compromesso tra risorse disponibili, requisiti di stabilità e obiettivi applicativi:

- Se l'obiettivo è la velocità di completamento e la capacità di adattarsi rapidamente a nuove mappe, **NEAT** risulta la scelta più indicata.

- Se invece si prioritizza l'affidabilità dell'agente e la costanza nei risultati per minimizzare il tasso di fallimento, la pipeline **PPO** offre garanzie superiori.

In entrambi i casi, è stato molto interessante scoprire come due **paradigmi di Intelligenza Artificiale profondamente diversi**, possano affrontare lo stesso ambiente con strategie differenti, e come la combinazione di queste due tecniche di apprendimento possa offrire una panoramica completa sulle potenzialità dell'AI in ambienti videoludici complessi.

Capitolo 6: Soluzioni non implementate

Durante la fase di progettazione sono state considerate diverse soluzioni alternative, successivamente scartate per motivi di complessità, coerenza con gli obiettivi del progetto e limiti computazionali.

1. Una prima soluzione tenuta in considerazione ma successivamente non implementata riguarda la pipeline **PPO**:

   - la definizione di un **obiettivo di massimizzazione del punteggio globale** del gioco. In questo scenario, l'agente non avrebbe dovuto limitarsi al completamento del livello, ma puntare a **massimizzare il punteggio totale**, includendo la sconfitta dei nemici e la raccolta sistematica di monete e power-up.

     Motivo dell'abbandono: Incoerenza con gli obiettivi del progetto.

2. Una seconda soluzione che è stata implementata durante l'addestramento, ma successivamente rimossa per via di alcuni failure cases osservati, riguarda la pipeline **NEAT**:

   - **Test di Robustezza Multi-Seed**: in una fase intermedia dello sviluppo, era stato implementato un meccanismo di verifica della fitness basato su seed multipli.
     - L'intento dietro questa scelta era quella di prevenire l'overfitting su un singolo feed fortunato.
     - Spesso, infatti, un genoma può scoprire una sequenza di tasti che funziona solo perché un nemico si muove in un modo specifico in quella singola run.
     - I "figli" di questo genoma, ereditandone la strategia ma venendo testati su seed diversi, fallivano miseramente, portando al collasso della specie.

     Motivo dell'abbandono: Sebbene teoricamente corretta, questa tecnica triplicava i tempi di calcolo per le valutazioni. Dato che come obiettivo del progetto è stato posto il completamento del livello senza specificarne il modo, allora non rappresenta alcun problema il fatto che un singolo genoma potesse beccare una run fortunata che lo permettesse di progredire oltre nel livello rispetto alla popolazione restante.

Capitolo 7: Sviluppi futuri

Il lavoro svolto apre a diversi possibili sviluppi futuri, sia sul piano metodologico sia su quello sperimentale.

- Un primo sviluppo riguarda l'**addestramento dell'agente su livelli successivi di Super Mario Bros**.
- L'attuale sperimentazione è limitata al livello 1-1 e 1-2, scelti per ridurre la complessità e consentire un confronto controllato tra gli approcci.

Estendere il training a più livelli permetterebbe di valutare al meglio la **capacità di generalizzazione** delle policy apprese, verificando se le strategie sviluppate risultano efficaci anche in ambienti con configurazioni diverse di ostacoli.