Documentazione Mine-FIA

Prof. Fabio Palomba

Giuseppe Russo 0512116396

Marco Renella 0512117806

Marco Meglio 0512118178

Anno Accademico 2024/2025

Repository: https://github.com/MarcoMeg03/MineFIA

Indice

1 Introduzione

1.1 Panoramica del progetto

MineFIA è un progetto sviluppato con l'obiettivo di esplorare le potenzialità dell'intelligenza artificiale applicata a Minecraft. Il progetto sfrutta la libreria MineRL e segue gli standard del programma BASALT per addestrare agenti capaci di svolgere compiti complessi in un ambiente simulato. Attraverso l'uso di tecniche di clonazione comportamentale e l'integrazione di funzioni di reward personalizzate.

Ci siamo ispirati alla competizione NeurIPS 2022 minerl BASALT competition, abbiamo definito un nostro obiettivo, un nostro ambiente ed abbiamo ottimizzato un modello per un task semplificato, adatto alla potenza computazionale di cui disponiamo.

1.2 Obiettivi

L'obiettivo principale del progetto è sviluppare un agente intelligente in grado di giocare a Minecraft, con particolare attenzione alle prime fasi del gioco. L'agente deve essere capace di:

- Raccogliere legno dagli alberi;
- Trasformarlo in assi;
- Creare un banco da lavoro.

Questi compiti rappresentano le operazioni fondamentali che un giocatore umano eseguirebbe per iniziare il proprio progresso nel gioco. Attraverso l'addestramento basato su tecniche di clonazione comportamentale e l'integrazione di funzioni di reward personalizzate, l'agente viene ottimizzato per completare queste attività in maniera autonoma ed efficiente.

1.3 Tecnologie Utilizzate

Le tecnologie utilizzate per lo sviluppo di questo progetto includono:

- **OpenAI Video Pre-Training (VPT)**: Modello pre-addestrato per migliorare le capacità degli agenti in ambienti complessi come Minecraft, si tratta di un modello estremamente inefficente , che sembra comportarsi a caso ma che di tanto in tanto riesce a compiere azioni come rompere blocchi, e la rottura di un blocco è una sequenza continuata di azioni.
- **Gym**: Libreria standard per l'interazione con ambienti di reinforcement learning.
- **MineRL**: Libreria per creare task personalizzati in Minecraft.
- **PyTorch**: Framework per lo sviluppo e l'addestramento di reti neurali.
- **NumPy**: Libreria per la manipolazione di array e matrici multidimensionali.
- **Pygame**: Utilizzato per implementare un sistema di registrazione manuale delle azioni dell'utente.
- **OpenCV**: Libreria per l'elaborazione di immagini e video.

1.4 VPT Model

VPT è un modello di rete neurale profonda progettato per interagire efficacemente con l'ambiente complesso di Minecraft.

`Architettura della rete`: La rete è composta da quattro componenti principali, ciascuna realizzata attraverso strati distinti:

1. **Strati convoluzionali (CNN)**: Per l'estrazione delle caratteristiche visive.
2. **Strato denso**: Per l'aggregazione delle caratteristiche visive.
3. **Transformer**: Per la modellazione temporale.
4. **Strato di output**: Per la distribuzione delle probabilità delle azioni.

1.4.1 Strati convoluzionali (CNN)

Gli strati convoluzionali elaborano input visivi costituiti da immagini RGB di dimensioni 128x128. La rete convoluzionale è suddivisa in 3 stack, ciascuno dei quali è composto da:

- **Un layer convoluzionale iniziale**:
  - Nel primo stack, il layer iniziale utilizza 64 filtri con kernel di dimensioni 3x3.
  - Nei successivi stack, il numero di filtri aumenta a 128, per catturare rappresentazioni più complesse.
- **Due blocchi residui**: Migliorano il flusso dei gradienti tramite connessioni dirette ("skip connections").

1.4.2 Layer Densi

Dopo i layer convoluzionali, i dati vengono appiattiti e passano attraverso due strati densamente connessi, chiamati `dense` e `linear` nella rete:

- **Dense Layer**:
  - Normalizzazione tramite `LayerNorm` per stabilizzare l'output delle convoluzioni.
  - Riduzione della dimensionalità a un vettore di dimensione 256.
- **Linear Layer**:
  - Normalizzazione tramite `LayerNorm`
  - Espansione della dimensionalità dell'output a un vettore di dimensione 1024, preparandolo per i layer ricorrenti.

Il primo layer denso (*dense*) si occupa di compattare e riassumere le informazioni complesse provenienti dai layer convoluzionali.

Il secondo layer denso (*linear*) prepara i dati per il passo successivo, ovvero i layer ricorrenti (*transformer*). L'espansione a 1024 garantisce una rappresentazione più ricca e adeguata al modello.

1.4.3 Layer Ricorrenti (Transformer)

I layer ricorrenti sono composti da 4 blocchi transformer principali, progettati per modellare le dipendenze temporali tra le osservazioni. Ogni blocco contiene:

- **Self-Attention Layer**: calcola l'importanza relativa tra le rappresentazioni temporali, permettendo alla rete di identificare elementi significativi nella sequenza.
- **MLP Residuo**: due strati completamente connessi (fully connected) per elaborare ulteriormente le rappresentazioni.
- **Layer di normalizzazione**: Migliora la stabilità.

I trasformatori consentono alla rete di catturare relazioni temporali tra osservazioni successive, un aspetto cruciale per compiti complessi come l'interazione in Minecraft.

1.4.4 Layer di Output

Dopo i layer ricorrenti, la rete utilizza uno strato di transizione chiamato *lastlayer*, che:

- Applica una normalizzazione (`LayerNorm`).
- Processa l'output normalizzato attraverso un layer *Linear*, mantenendo la dimensione del vettore a 1024, senza alterarne la dimensionalità, ma trasformandolo per adattarlo alle teste di output.

Dopodiché, il modello termina con due teste di output:

- **Pi Head**:
  - Produce probabilità per azioni discrete (come "avanti", "salta", "attacca") utilizzando distribuzioni categoriali.
  - Modella azioni continue (come il movimento della telecamera) utilizzando distribuzioni gaussiane.
- **Value Head**:
  - Stima il valore (la qualità) dello stato corrente.

1.5 Specifica PEAS

Un ambiente rappresenta un'istanza di un problema dove un agente razionale deve trovare una soluzione ottimale. Nel contesto del progetto, l'ambiente è una simulazione in Minecraft focalizzata su obiettivi specifici. La formulazione PEAS (Performance, Environment, Actuators, Sensors) descrive le caratteristiche fondamentali di questo ambiente.

Performance (Prestazioni) Le prestazioni dell'agente vengono misurate in base alla capacità di completare obiettivi predefiniti nell'ambiente, come:

- Raccogliere legno.
- Trasformare il legno in assi.
- Creare un banco da lavoro.

Il successo è valutato in termini di episodi completati con successo entro un limite di step. In questo caso, valutiamo quante volte l'agente riesce a portare a termine l'obiettivo in 50 tentativi, ciascuno di 2000 step (circa 2 minuti).

Environment (Ambiente) L'ambiente è simulato in Minecraft, generato con specifiche condizioni iniziali e condizioni di terminazione.

Actuators (Attuatori) L'agente utilizza una libreria MineRL per interagire con l'ambiente, tra cui:

- **Movimento**: avanti, indietro, sinistra, destra, salto, corsa.
- **Interazione**: attacco, utilizzo di oggetti, apertura di interfacce come l'inventario, posizionamento e raccolta di blocchi, crafting di oggetti.

Sensors (Sensori) L'agente percepisce l'ambiente attraverso la vista in prima persona del mondo di gioco (immagini del POV).

1.6 Caratteristiche dell'ambiente

L'ambiente presenta le seguenti caratteristiche:

- **Parzialmente osservabile**: L'agente non ha una visione completa del mondo di gioco; osserva solo ciò che è visibile dalla prospettiva in prima persona. Elementi nascosti o lontani richiedono esplorazione.
- **Stocastico**: Gli eventi e la disposizione degli oggetti nell'ambiente presentano componenti casuali, rendendo le dinamiche imprevedibili e variabili.
- **Sequenziale**: Le azioni dell'agente seguono una sequenza in cui ogni scelta influenza lo stato successivo, rendendo l'ordine delle operazioni fondamentale per il raggiungimento dell'obiettivo.
- **Dinamico**: L'ambiente può evolvere durante l'episodio, in risposta alle azioni dell'agente o in modo autonomo.

- **Discreto/Continuo**: Le azioni e gli stati sono prevalentemente rappresentati da insiemi discreti, tuttavia alcune azioni sono descritte mediante insiemi continui (movimento del mouse).
- **Singolo-Agente**: L'ambiente è progettato per un singolo agente che interagisce con il mondo virtuale senza la presenza di altri agenti che possano influenzarne le decisioni o il comportamento.

2 Analisi dei dati

2.1 Introduzione al Dataset

Per lo sviluppo del progetto è stato utilizzato un mix di dati preesistenti e dati generati manualmente:

1. **Dataset Preesistente**: È stato utilizzato il dataset `MineRLObtainDiamondShovel`, che contiene sessioni registrate di gameplay Minecraft, contiene tutte le osservazioni necessarie per arrivare alla raccolta del diamante e la creazione di una pala in diamante. La parte di dati estratta da questo dataset rappresenta il 60% dei dati totali utilizzati.
2. **Dati Generati Manualmente**: Per completare il dataset e arricchirlo di dati che interessano il nostro task specifico, è stato sviluppato uno script personalizzato denominato `manual_recorder.py` che consente di registrare video (in formato .mp4) e azioni correlate ad ogni fotogramma (in formato .jsonl) durante l'interazione diretta con l'ambiente di Minecraft. Questo strumanto è stato utile per generare nuove osservazioni e per bilanciare quelle presenti.Per esempio abbiamo osservato che il dataset era povero di esperienze per quanto riguarda la raccolta di legno di acacia, e di legno della giungla.

   Nello specifico

I dati sono strutturati in due formati:

- **Video**: File .mp4 che rappresentano il gameplay registrato.
- **Azioni**: File .jsonl contenenti il mapping dettagliato delle azioni effettuate per ogni fotogramma(tasti premuti, movimenti del mouse, interazioni con l'ambiente).

2.2 Pulizia e Preprocessing

La qualità dei dati preesistenti è stata adattata ai nostri scopi tramite tecniche di estrazione e pulizia, utilizzando in particolare due script:

- **Intallazione_selezionata.py** : Poiché il dataset ha una dimensione enorme (1130GB), impossibile da gestire con le nostre risorse, abbiamo deciso di effettuare una prima pulizia direttamente durante la fase di installazione.Abbiamo installato soltanto i file JSONL , li abbiamo esaminati tramite uno scritpt e abbiamo eliminato subito tutti quelli che non fossero utili al nostro scopo, Dalla lista così ottenuta, abbiamo installato solo i video correlati ai file JSONL selezionati (molto più pesanti), riducendo l'installazione al 40% delle osservazioni totali.

- **CutData.py** : Ogni video è di circa 4 minuti , ma soltanto una piccola parte di questi video era utile al nostro scopo, per questo motivo abbiamo realizzato uno script che estraesse soltanto la porzione di video utile al nostro fine. Questo garantisce che i dati siano ottimizzati per il processo di addestramento e non portino l'agente a compiere azioni diverse dal nostro obiettivo.

2.2.1 Codice CutVideo

```
def process_videos(input_folder, output_folder):
    """
    Processa i video e i relativi JSONL nella cartella di input.
    """
    os.makedirs(output_folder, exist_ok=True)
    video_paths = [f for f in os.listdir(input_folder) if f.endswith(".mp4")]
    for video_name in video_paths:
        jsonl_name = video_name.replace(".mp4", ".jsonl")
        video_path = os.path.join(input_folder, video_name)
        jsonl_path = os.path.join(input_folder, jsonl_name)

        if not os.path.exists(jsonl_path):
            print(f"JSONL non trovato per {video_name}, salto...")
            continue

        last_tick, cut_time, is_useful = get_cut_timestamp(jsonl_path)

        if not is_useful:
            print(f"Scartato video {video_name} per contenere oggetti non utili.")
            continue

        if cut_time is None or last_tick is None:
            print(f"Nessun taglio necessario per {video_name}, salto...")
            continue

        output_video_path = os.path.join(output_folder, f"trimmed_{video_name}")
        cut_frame = trim_video(video_path, output_video_path, cut_time)

        output_jsonl_path = os.path.join(output_folder, f"trimmed_{jsonl_name}")
        trim_jsonl(jsonl_path, output_jsonl_path, last_tick)
```

2.3 Data Augmentation

Per aumentare la variabilità del dataset, è stato implementato uno script denominato `MirrorData.py`. Questo script specchia i dati (video e azioni corrispondenti), garantendo una maggiore diversità senza modificare la semantica delle azioni.In altre parole abbiamo duplicato le osservazioni specchiando i video e le azioni associate evitando di specchiare fotogrammi non specchiabili, come per esempio l'istante in cui una qualsiasi GUI è aperta.

2.3.1 Codice mirroring video

```python
def mirror_video(input_path, output_path, json_data):
    # Apri il video originale
    cap = cv2.VideoCapture(input_path)

    if not cap.isOpened():
        print(f"Errore nell'aprire il video: {input_path}")
        return

    # Ottieni le propriet  del video
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)
    codec = cv2.VideoWriter_fourcc(*'mp4v')  # Codec per output in formato .mp4

    if fps == 0 or width == 0 or height == 0:
        print(f"Errore: il video {input_path} ha FPS o dimensioni non validi.")
        cap.release()
        return

    # Configura il writer per salvare il video specchiato
    out = cv2.VideoWriter(output_path, codec, fps, (width, height))

    frame_index = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Controlla isGuiOpen per decidere se specchiare il frame
        if frame_index < len(json_data) and not json_data[frame_index].get("isGuiOpen",
            False):
            mirrored_frame = cv2.flip(frame, 1)  # Specchia solo se isGuiOpen    False
        else:
            mirrored_frame = frame  # Non specchiare se isGuiOpen    True

        out.write(mirrored_frame)
        frame_index += 1

    # Chiudi tutto
    cap.release()
    out.release()
    print(f"Video specchiato salvato in: {output_path}")
```

2.3.2 Codice mirroring JSONL

```python
def mirror_json(input_json_path, output_json_path, video_path):
    cap = cv2.VideoCapture(video_path)
    width = 1280  # Dimensione dello schermo
    cap.release()

    with open(input_json_path, "r") as infile:
        lines = infile.readlines()

    mirrored_lines = []
    for line in lines:
        data = json.loads(line)

        if not data.get("isGuiOpen", False):  # Specchia solo se isGuiOpen = False
            if "mouse" in data:
                data["mouse"]["dx"] = -data["mouse"]["dx"]
                data["mouse"]["scaledX"] = -data["mouse"]["scaledX"]
                data["mouse"]["x"] = width - data["mouse"]["x"]

            if "keyboard" in data and "keys" in data["keyboard"]:
                keys = data["keyboard"]["keys"]
                new_keys = []
                new_chars = []
                for key in keys:
                    if key == "key.keyboard.a":
                        new_keys.append("key.keyboard.d")
                        new_chars.append("d")
                    elif key == "key.keyboard.d":
                        new_keys.append("key.keyboard.a")
                        new_chars.append("a")
                    else:
                        new_keys.append(key)
                        if "chars" in data["keyboard"]:
                            new_chars.append(data["keyboard"]["chars"])

                data["keyboard"]["keys"] = new_keys
                if "chars" in data["keyboard"]:
                    data["keyboard"]["chars"] = "".join(new_chars)

            if "hotbar" in data:
                data["hotbar"] = 8 - data["hotbar"]

        mirrored_lines.append(json.dumps(data))
    with open(output_json_path, "w") as outfile:
        outfile.write("\n".join(mirrored_lines) + "\n")
```

2.4 Data Quality

2.4.1 Correttezza osservazioni specchiate

Per verificare la correttezza del mirroring dei video , abbiamo deciso di specchiare l'osservazione e di specchiare nuovamente l'osservazione specchiata , in questo modo abbiamo stabilito la correttezza , in quanto l'osservazione specchiata due volte appariva identica all'osservazione di partenza.

2.4.2 Correttezza osservazioni generate

Per verificare la qualità dei dati raccolti, è stato utilizzato lo script `visualize_mouse_movement.py`. Questo strumento consente di visualizzare graficamente i movimenti del mouse registrati durante le sessioni di interazione, garantendo che i dati creati siano coerenti e privi di errori.

2.4.3 Codice visualize_mouse_movement

```
def main(file_path):
    # Estrai le coordinate
    coordinates = extract_coordinates(file_path)
    if not coordinates:
        print("Nessuna coordinata valida trovata nel file.")
        return

    # Inizializza Pygame
    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Mouse Movement Visualization")
    clock = pygame.time.Clock()

    running = True
    index = 0

    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        # Pulisci lo schermo
        screen.fill(BACKGROUND_COLOR)

        # Disegna il pallino
        if index < len(coordinates):
            x, y = coordinates[index]
            x = min(max(0, int(x)), WIDTH - 1)  # Mantieni le coordinate nei limiti
            y = min(max(0, int(y)), HEIGHT - 1)
            print(f"Coordinata attuale: x={x}, y={y}")  # Stampa le coordinate
            pygame.draw.circle(screen, BALL_COLOR, (x, y), BALL_RADIUS)
            index += 1
        else:
            print("Fine delle coordinate, chiusura del programma.")
            running = False  # Termina

        pygame.display.flip()  # Aggiorna lo schermo
        clock.tick(FPS)
    pygame.quit()
```

2.5 Tipologia del problema

Il problema affrontato nel progetto MineFIA rientra nella categoria di **apprendimento sequenziale decisionale**.

Si tratta di un problema di **decision-making basato su sequenze temporali**, dove l'obiettivo dell'agente è imparare a prendere decisioni ottimali in maniera iterativa in un ambiente complesso, come Minecraft, basandosi sulla sola osservazione dell'ambiente.

Caratteristiche del problema

- **Apprendimento supervisionato con behavioural cloning**:
  - La rete è addestrata a imitare il comportamento umano.
  - Questo processo si basa su dataset pre-registrati che contengono sequenze di osservazioni e azioni eseguite da giocatori umani esperti.
- **Previsione delle azioni (multi-output)**:
  - L'obiettivo non è classificare un dato in una singola classe, ma prevedere:
    - **Azioni discrete**, come "muoviti avanti", "salta" o "attacca".
    - **Azioni continue**, come la rotazione della telecamera (parametri x, y).
  - Questo rende il problema una combinazione di **classificazione multi-classe** (per azioni discrete) e **regressione** (per azioni continue).
- **Dipendenza temporale**:
  - Ogni decisione dipende non solo dall'osservazione corrente, ma anche dalle azioni e osservazioni passate.
  - Questo introduce una natura **sequenziale** al problema, che viene modellata utilizzando trasformatori (Transformer).
- **Apprendimento da dati ad alta dimensionalità**:
  - Gli input sono immagini RGB di dimensioni 128x128 e un inventario codificato.
  - La rete deve estrarre caratteristiche visive rilevanti e mappare queste informazioni a una serie di possibili azioni.

2.6 Funzione di perdita

L'addestramento del modello utilizza una funzione di perdita per calcolare l'errore e aggiornare i parametri del modello durante l'addestramento. La funzione di perdita è composta da due componenti principali:

2.6.1 Log-Likelihood Loss

```
def logprob(self, action_sample: torch.Tensor, pd_params: torch.Tensor) -> torch.Tensor:
    """Log-likelihood"""
    means = pd_params[..., 0]
    log_std = pd_params[..., 1]

    std = torch.exp(log_std)

    z_score = (action_sample - means) / std

    return -(0.5 * ((z_score ** 2 + self.LOG2PI).sum(dim=-1)) + log_std.sum(dim=-1))
```

Questa parte misura quanto il modello è vicino a prevedere correttamente le azioni osservate nei dati di addestramento. La funzione restituisce il logaritmo della probabilità dell'azione osservata data la distribuzione di probabilità predetta.
La formula implementata per calcolare la log-probabilità è quella di una distribuzione normale:

$$\log\_\text{prob}(a) = -\left(0.5 \cdot \left(\frac{(a-\mu)^2}{\sigma^2} + \log(2\pi)\right) + \log(\sigma)\right)$$

Dove:

- a è l'azione osservata,
- μ è la media della distribuzione (`means`),
- σ è la deviazione standard $\sigma = e^{\texttt{log\_std}}$,
- $\log(\sigma)$ è il logaritmo della deviazione standard.

La formula restituisce un valore negativo proporzionale alla distanza fra l'azione osservata e la media della distribuzione predetta.

2.6.2 KL Divergence (Kullback-Leibler Divergence)

Questa componente penalizza il modello se la distribuzione predetta si discosta troppo dalla distribuzione di riferimento.

```
kl_div = policy.get_kl_of_action_dists(pi_distribution, original_pi_distribution)
loss += KL_LOSS_WEIGHT * kl_div / BATCH_SIZE
```

- `pi_distribution`: La distribuzione predetta dal modello corrente.
- `original_pi_distribution`: La distribuzione predetta dal modello originale (di riferimento).
- `KL_LOSS_WEIGHT` è un fattore che bilancia l'importanza di questa componente.

Confronta la distribuzione predetta dal modello attuale $\pi_{\text{distribution}}$ con quella di riferimento π_{original}. Questo regolarizzatore aiuta il modello a mantenere una certa somiglianza con il comportamento iniziale.

2.6.3 Funzione di perdita totale

La funzione di perdita totale combina entrambe le componenti:

```
loss = (-log_prob + KL_LOSS_WEIGHT * kl_div) / BATCH_SIZE
```

- `-log_prob`: Riduce l'errore nel prevedere azioni corrette.
- `KL_LOSS_WEIGHT * kl_div`: Penalizza grandi deviazioni dalla distribuzione di riferimento.

3 Reinforcement Learning in Fase di Esecuzione

In fase di esecuzione, il modello addestrato viene ulteriormente ottimizzato tramite tecniche di reinforcement learning per migliorare le sue prestazioni in ambienti complessi. Questa fase sfrutta i concetti di reward personalizzate, stima dei vantaggi tramite Generalized Advantage Estimation (GAE) e aggiornamento della policy tramite Proximal Policy Optimization (PPO).

3.1 Definizione delle Reward

Le reward sono state progettate per guidare il modello verso comportamenti desiderabili e penalizzare azioni non produttive.

3.1.1 Reward Basate sull'Inventario

Il modello assegna reward positive per l'accumulo di materiali utili e penalità per la raccolta di materiali non rilevanti. La funzione `compute_reward` definisce un sistema di ricompense basato sull'inventario dell'agente. Di seguito, il codice implementato:

```
def compute_reward(inventory, prev_inventory):
    reward = 0
    for material, value in MATERIAL_REWARDS.items():
        current_quantity = inventory.get(material, 0)
        previous_quantity = prev_inventory.get(material, 0)
        if current_quantity > previous_quantity:
            reward += (current_quantity - previous_quantity) * value
    return reward / (abs(reward) + 1e-6) if abs(reward) > 1 else reward
```

- `inventory`: Lo stato corrente dell'inventario.
- `prev_inventory`: L'inventario al passo precedente.
- `MATERIAL_REWARDS`: Una mappa che associa un valore numerico a ciascun materiale.

La reward totale viene calcolata sommando i contributi di ogni materiale presente nell'inventario. Gli incrementi di materiali predefiniti ("logs", "planks", ecc.) generano ricompense positive proporzionali, mentre materiali meno utili, come sabbia o ghiaia, comportano penalizzazioni.

3.1.2 Reward Basate sulle Azioni

Un sistema aggiuntivo di reward guida il modello a intraprendere azioni utili, come l'apertura dell'inventario in momenti opportuni o l'evitare azioni ripetitive come i salti inutili:

```
def action_based_reward(action, inventory, jump_window, inventory_reward_given):
    reward = 0
    if (any(inventory.get(log, 0) > 0 for log in LOG_TYPES) and
        inventory.get("crafting_table", 0) == 0 and
        action.get("inventory", 0) == 1 and not inventory_reward_given):
        reward += 0.5
        inventory_reward_given = True
    if sum(jump_window) > 10:
        reward -= 0.18
    if not any(np.any(value == 1) if isinstance(value, np.ndarray) else value == 1 for
        value in action.values()):
        reward -= 0.1
    return reward / (abs(reward) + 1e-6) if abs(reward) > 1 else reward,
        inventory_reward_given
```

3.1.3 Formula della Reward Normalizzata

L'agente per alcune azioni potrebbe ricevere una reward sproporzionata, per esempio all'atto di costruzione delle assi di legno, non costruisce un uico blocco alla volta, ma a gruppi di 4 e addirittura, in alcuni casi anche 40 in un colpo solo.
La reward calcolata viene normalizzata per evitare valori estremi:

$$\text{Reward Normalizzata} = \frac{R}{|R| + \epsilon} \quad \text{se } |R| > 1$$

Dove:

- R è la reward non normalizzata.
- ϵ è un piccolo valore (ad esempio, 10^{-6}) per evitare divisioni per zero.

3.2 Generalized Advantage Estimation (GAE)

La Generalized Advantage Estimation (GAE) è una tecnica utilizzata per calcolare il vantaggio $A(s_t, a_t)$, che rappresenta quanto un'azione è migliore rispetto alla media delle azioni possibili in uno stato dato. GAE combina ricompense immediate con stime delle ricompense future, bilanciando l'influenza di entrambe. Questo approccio consente di ottenere una stima del vantaggio più stabile, riducendo la varianza senza introdurre un bias significativo. La formula implementata è:

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

Dove:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

```
def compute_gae(rewards, values, gamma=0.99, lambda_=0.95):
    advantages = []
    gae = 0
    for t in reversed(range(len(rewards))):
        delta = rewards[t] + gamma * (values[t + 1] if t + 1 < len(values) else 0) -
            values[t]
        gae = delta + gamma * lambda_ * gae
        advantages.insert(0, gae)
    return th.tensor(advantages)
```

- γ: Fattore di sconto per la reward futura. Un valore tipico è 0.99, che riduce l'importanza delle ricompense future man mano che si allontanano temporalmente.
- λ: Parametro che bilancia bias e varianza. Valori tipici sono compresi tra 0.9 e 0.95.
- δ_t: Errore di vantaggio temporale, dato dalla differenza tra la reward osservata più il valore futuro scontato $\gamma V(s_{t+1})$ e il valore stimato $V(s_t)$.

La GAE migliora la stabilità dell'ottimizzazione riducendo le fluttuazioni nelle stime del vantaggio.

3.3 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) è un metodo di policy gradient che utilizza una funzione di clipping per limitare aggiornamenti drastici della policy.
La funzione obiettivo è:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t) \right]$$

Dove:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

```
ratio = th.exp(log_probs_tensor - old_log_probs_tensor)
clipped_ratio = th.clamp(ratio, 1 - epsilon, 1 + epsilon)
loss_clip = -th.min(ratio * advantages, clipped_ratio * advantages).mean()
```

- ϵ: Parametro di clipping (tipicamente 0.2), che limita la variazione del rapporto $r_t(\theta)$.
- $r_t(\theta)$: Rapporto tra la probabilità delle azioni correnti e quelle precedenti, indicativo di quanto la policy è cambiata.
- A_t: Vantaggio calcolato tramite GAE.

Il meccanismo di clipping assicura che gli aggiornamenti della policy siano stabili e che le probabilità non varino drasticamente tra un'iterazione e l'altra.

3.4 Normalizzazione del Vantaggio

Per migliorare la stabilità numerica e garantire che il modello apprenda correttamente, il vantaggio A_t è normalizzato:

$$A_t \leftarrow \frac{A_t - \mu(A)}{\sigma(A) + \epsilon}$$

Dove:

- $\mu(A)$: Media del vantaggio sull'intero batch.
- $\sigma(A)$: Deviazione standard del vantaggio.
- ϵ: Termine di stabilità numerica, tipicamente 10^{-8}.

Il codice corrispondente è:

```
def normalize(tensor):
    return (tensor - tensor.mean()) / (tensor.std() + 1e-8)
```

3.5 Ottimizzazione della Funzione Valore

Il modello ottimizza anche la funzione valore per migliorare la stima degli stati. Contestualmente, vengono aggiornati anche i parametri del layer **pi_head**, responsabile della distribuzione delle probabilità delle azioni:

```
for param in agent.policy.pi_head.parameters():
    param.requires_grad = True
for param in agent.policy.value_head.parameters():
    param.requires_grad = True
```

3.6 Ottimizzazione con RMSProp

Per ottimizzare i parametri del modello, il codice utilizza l'algoritmo di ottimizzazione **RMSProp**. Questo algoritmo è particolarmente adatto per problemi di reinforcement learning, poiché gestisce efficacemente gradienti rumorosi e oscillanti, tipici in ambienti complessi come Minecraft.

La configurazione di **RMSProp** è la seguente:

```
optimizer = th.optim.RMSprop(
    filter(lambda p: p.requires_grad, agent.policy.parameters()),
    lr=0.00001,
    alpha=0.99,
    eps=1e-8
)
scheduler = th.optim.lr_scheduler.StepLR(optimizer, step_size=100, gamma=0.9)
```

- `lr (learning rate)`: Il tasso di apprendimento è impostato a 10^{-5}, un valore basso per garantire stabilità durante gli aggiornamenti.
- `alpha`: Coefficiente per la media esponenziale dei gradienti quadrati ($\alpha = 0.99$). Aiuta a mantenere una stima accurata della varianza dei gradienti.
- `eps`: Termine di stabilità numerica ($\epsilon = 10^{-8}$) per evitare divisioni per zero durante il calcolo.

3.6.1 Approccio Batch-wise e Utilizzo dello Scheduler

L'ottimizzazione viene effettuata ogni 15 passi utilizzando un approccio `batch-wise`; durante questo intervallo, il modello accumula rewards, valori stimati degli stati - V(s) -, e log-probabilità delle azioni.

Lo scheduler `StepLR` è utilizzato per regolare dinamicamente il tasso di apprendimento durante l'addestramento:

- `step_size`: Specifica ogni quanto diminuire il tasso di apprendimento (ogni 100 passi in questo caso).
- `gamma`: Fattore di riduzione del learning rate, qui impostato a 0.9, che riduce progressivamente il passo di apprendimento per stabilizzare l'ottimizzazione nelle fasi avanzate.

3.7 Conclusione Reinforcement Learning

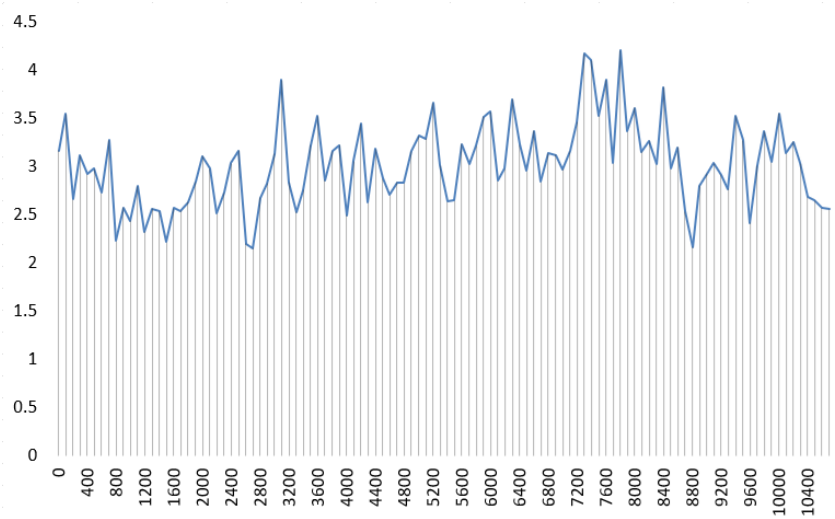
L'approccio descritto utilizza metodi avanzati come PPO e GAE per ottimizzare la performance dell'agente in fase di esecuzione. Queste tecniche garantiscono stabilità, riducono la varianza e migliorano l'efficienza complessiva del modello, rendendolo adatto ad ambienti complessi come Minecraft.

4 Testing

4.0.1 Training loss

Una prima indicazione delle performance del modello è stata ottenuta direttamente dalla fase di training con clonazione comportamentale.

Nota: il training è durato molto più a lungo, ma abbiamo riportato solo una porzione del grafico.

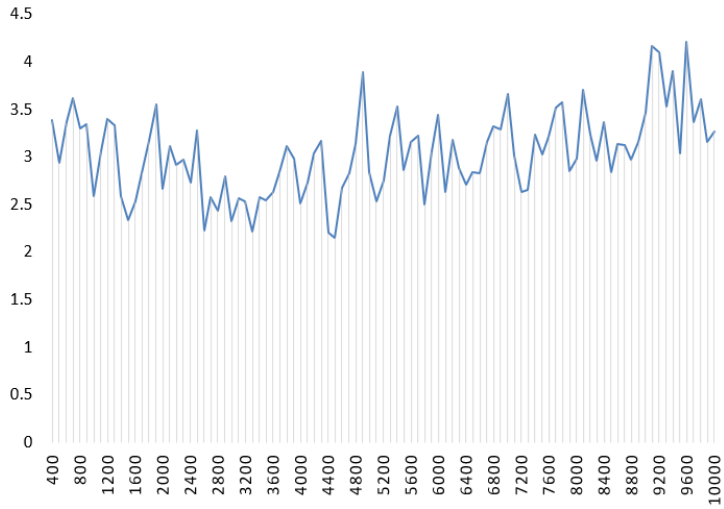


Il risultato non è soddisfacente. Fin da subito ci siamo accorti che, sebbene monitorare la funzione di loss sia utile per addestrare e correggere il modello, non è altrettanto efficace per testarlo.

4.1 Test set

Abbiamo messo da parte una porzione del dataset, pari al 25% del training set. Abbiamo eseguito il modello e monitorato la funzione di loss.

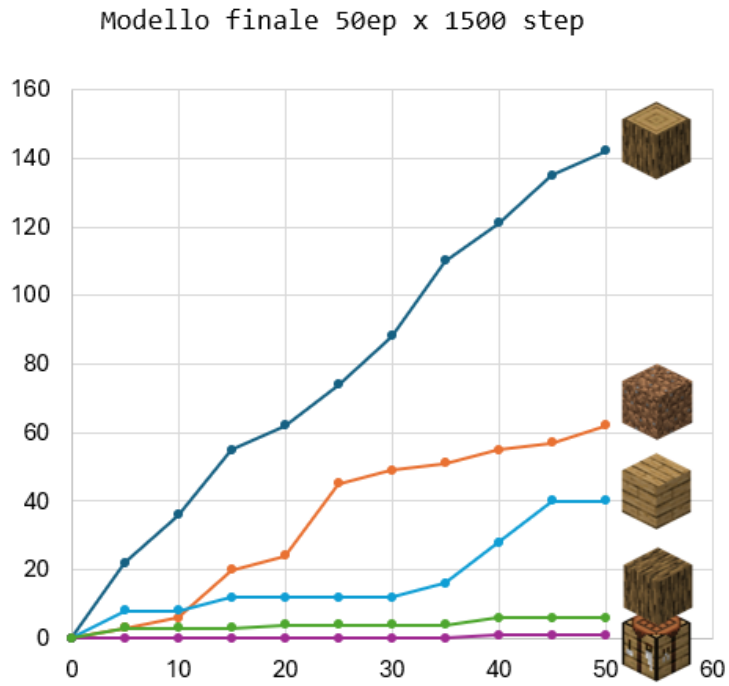
Nota: l'esecuzione del test set è durata molto più a lungo, ma abbiamo riportato solo una porzione del grafico.



Secondo noi, la loss non fornisce un'indicazione affidabile sulla bontà dell'agente, poiché in una determinata situazione l'agente potrebbe compiere molte azioni completamente diverse e comunque corrette.
Ad esempio, se in un determinato fotogramma l'agente si trova di fronte a un bosco, potrebbe voltarsi verso qualsiasi albero nelle vicinanze. Qualunque scelta sarebbe corretta, ma potrebbe differire dal risultato atteso.

4.2 Esecuzione all'interno dell'ambiente

Abbiamo eseguito il modello direttamente all'interno dell'ambiente, avviando 50 episodi da 1500 step ciascuno. Ogni step corrisponde a un'azione e, in media, ha una durata di circa un minuto e mezzo.



Abbiamo prestato attenzione non tanto alla quantità dei materiali raccolti, ma al comportamento dell'agente e alla frequenza di azioni "sensate". In un minuto e 30 secondi, nel 75% dei casi riesce a raccogliere il legno, nel 10% dei casi riesce a costruire assi di legno e soltanto nel 2% dei casi è riuscito a costruire un banco da lavoro, anche se spesso ci è andato vicino.
Nonostante ciò, secondo noi il reale comportamento dell'agente può essere giudicato soltanto osservandolo all'opera e fornendo un giudizio umano.