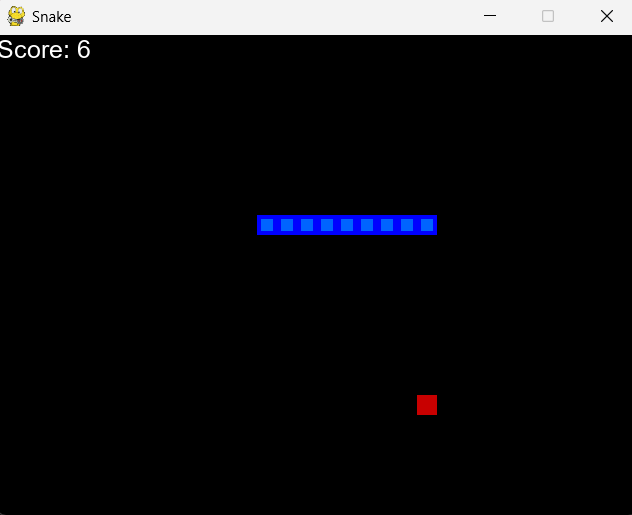
**AI Agent Snake Game training via Reinforcement Learning**

Il presente progetto ha l'obiettivo di sviluppare e addestrare un agente di intelligenza artificiale per giocare al gioco Snake, utilizzando tecniche di **Reinforcement Learning (RL)**. In particolare, viene implementato un agente AI che apprende a giocare attraverso **Deep Q-Learning**, un'estensione del classico algoritmo Q-Learning che sfrutta una rete neurale per approssimare la funzione Q.

L'agente interagisce con l'ambiente di gioco, ricevendo ricompense in base alle azioni intraprese, con l'obiettivo di massimizzare il punteggio (numero di cibi raccolti) mantenendo il serpente in vita il più a lungo possibile.

**Problem Statement**

Il gioco **Snake** prevede che un serpente, inizialmente di lunghezza fissa, si muova su una griglia e cerchi di mangiare cibo per allungarsi. Il gioco termina quando il serpente colpisce il bordo dello schermo o se stesso. L’obiettivo è quello di massimizzare il punteggio, raccogliendo quanti più cibi possibili.

L'agente AI deve imparare a evitare collisioni con i bordi e con il proprio corpo, e allo stesso tempo individuare strategie efficaci per raccogliere il cibo in modo ottimale. Questo richiede un bilanciamento tra l’esplorazione di nuove strategie e l’ottimizzazione delle mosse già note, tramite un processo iterativo di apprendimento basato su ricompense e penalità.  


**Metodologia**

L'approccio di **Reinforcement Learning** si basa su una combinazione tra:

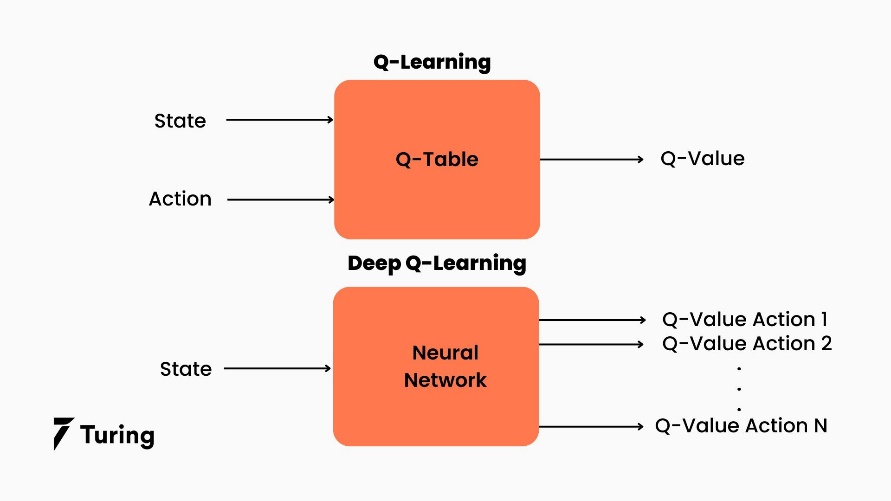
**Stato**: la rappresentazione dell’ambiente, che comprende la posizione del serpente, la posizione del cibo e la direzione attuale del serpente.

**Azione**: la decisione dell’agente di muoversi in una delle quattro direzioni (su, giù, destra, sinistra).

**Reward**: una ricompensa che l’agente riceve in base all’esito della sua azione. L’agente riceve una ricompensa positiva quando raccoglie il cibo, e una penalità se collide con il bordo dello schermo o con se stesso.

**Algoritmo: Deep Q-Learning**

L'agente utilizza il **Deep Q-Learning** per approssimare la funzione Q, che valuta il valore di ciascuna azione possibile dato lo stato corrente. L'architettura utilizzata comprende una rete neurale con due strati completamente connessi: uno strato input con 11 neuroni connesso ad uno nascosto con 256 neuroni e un livello nascosto (quello precedentemente utilizzato) connesso con uno di output di 3 neuroni che rappresenta le possibili azioni.



**Struttura dell'Agente**

L'agente è implementato utilizzando il framework **PyTorch**. I passaggi principali dell'algoritmo includono:

**Stato**: L'agente riceve in input lo stato corrente del gioco, che comprende informazioni sulla posizione del cibo, la direzione attuale e i pericoli nelle vicinanze (bordo dello schermo o parte del corpo del serpente).

**Azione**: L'agente sceglie un'azione utilizzando una strategia \*\*epsilon-greedy\*\*, bilanciando l'esplorazione di nuove azioni con lo sfruttamento di quelle già apprese.

**Memoria**: Le esperienze di gioco (stato, azione, reward, nuovo stato) vengono memorizzate in un buffer di replay, da cui l'agente può estrarre campioni casuali per il training.

**Training**: Attraverso l'aggiornamento della funzione Q, l’agente migliora progressivamente la sua capacità di prendere decisioni efficaci.

**Implementazione**

L’implementazione è suddivisa in più componenti:

**game.py**: contiene la logica del gioco Snake, inclusi il rendering grafico con \*\*Pygame\*\* e il meccanismo di gestione del serpente e del cibo.

**model.py**: definisce la rete neurale utilizzata per il Deep Q-Learning e la funzione di ottimizzazione.

**agent.py**: implementa l'agente AI, gestisce il ciclo di apprendimento e le strategie di esplorazione/esploitazione.

**helper.py**: fornisce strumenti per la visualizzazione grafica dei punteggi durante l'addestramento.

**Training**

Il training dell'agente si svolge attraverso episodi di gioco in cui l'agente impara a migliorare progressivamente le sue prestazioni. Durante ogni episodio, l'agente esegue le seguenti operazioni:

1. Acquisisce lo stato attuale dell'ambiente.
2. Seleziona un'azione utilizzando la strategia epsilon-greedy.
3. Riceve una ricompensa in base all’azione intrapresa.
4. Memorizza l’esperienza (stato, azione, reward, nuovo stato) nel buffer di replay.
5. Esegue il training a lungo termine estraendo campioni dal buffer di replay e aggiornando i pesi della rete neurale.

**Funzione di Ricompensa**

La funzione di ricompensa premia l'agente per aver mangiato il cibo (+10 punti) e lo penalizza per la collisione con il muro o con se stesso (-10 punti). Ciò incentiva l’agente a mantenere il serpente vivo il più a lungo possibile e a raccogliere cibo per massimizzare il punteggio.

**Risultati**

Durante il processo di training, l'agente ha mostrato un miglioramento progressivo delle prestazioni, misurato attraverso il punteggio medio ottenuto nei vari episodi di gioco. La visualizzazione grafica dei punteggi e delle medie mostra una chiara tendenza al rialzo, segno di un apprendimento efficace.

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

**Conclusioni**

Il progetto ha dimostrato l'efficacia del Deep Q-Learning per addestrare un agente AI a giocare a Snake, risolvendo con successo il problema dell'esplorazione di strategie efficaci in un ambiente dinamico e pieno di sfide. In prospettiva futura, sarà possibile ottimizzare ulteriormente il sistema attraverso tecniche avanzate, per migliorare l'efficienza dell'addestramento e le prestazioni dell'agente.

**REINFORCEMENT LEARNING (Apprendimento per Rinforzo)**

Un'area del machine learning che si concentra su come gli agenti software prendono decisioni in un ambiente per massimizzare la ricompensa.

**Deep Q Learning**: Estende l'apprendimento per rinforzo utilizzando una rete neurale profonda per prevedere le azioni.

**Agente**: Deve conoscere sia il gioco che il modello, in modo da poter funzionare correttamente.

**Ciclo di addestramento**: Basato sul gioco, calcoliamo il passo successivo che definisce la prossima azione. Dopo aver eseguito l'azione, otteniamo un nuovo passo che definisce un nuovo stato che viene salvato e mantenuto per il ciclo successivo.

**Gioco**: È un ciclo di gioco composto da passi. Dopo ogni passo, restituisce la ricompensa e un punteggio complessivo.

**Concetto di ricompensa**: Verrà spiegato basandosi sul gioco Snake. Dopo aver ottenuto una ricompensa, il serpente guadagna +10 punti; se muore, ottiene -10 punti e se non succede nulla, il valore rimane 0.

**Azione**: Direzioneremo il serpente in tre direzioni in base a dove sta guardando, che corrisponde a [dritto, svolta a sinistra, svolta a destra], per evitare svolte immediate di 180° che ucciderebbero il serpente.

**Stato**: I valori dello stato includono:

[pericolo dritto, pericolo a destra, pericolo a sinistra, direzione sinistra, direzione destra, direzione su, direzione giù, cibo a sinistra, cibo a destra, cibo sopra, cibo sotto]

Immagine che contiene schermata, design

Descrizione generata automaticamenteEsempio:  
[0, 0, 0,

0, 1, 0, 0,

0, 1, 0, 1]

**Modello**

Un modello di rete neurale con livelli lineari, dove utilizziamo model.predict per essere usato nell'agente. Nel nostro modello, abbiamo un livello di input, un livello nascosto e un livello di output:

* **Livello di input**: rappresenta lo stato. Dal momento che abbiamo 11 valori booleani, la dimensione iniziale della rete neurale sarà 11.
* **Livello nascosto**: definito da una dimensione nascosta.
* Immagine che contiene testo, schermata, diagramma, Carattere

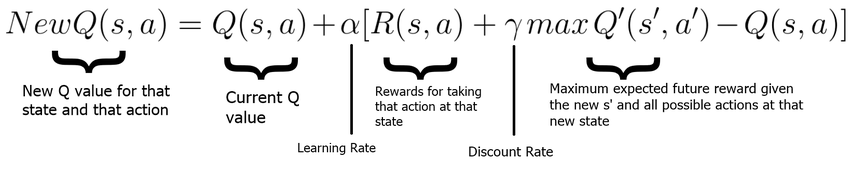
  Descrizione generata automaticamente**Livello di output**: abbiamo bisogno di tre output per prevedere l'azione. Non c'è bisogno di probabilità; sceglieremo semplicemente il numero più grande.

**DEEP Q Learning**

Q sta per **Qualità dell'azione**, che è l'obiettivo da migliorare. Divideremo i passi in:

1. Inizializzare il valore Q (= inizializzare il modello)
2. Scegliere l'azione (model.predict(state) || mossa casuale)
3. Eseguire l'azione
4. Misurare la ricompensa
5. Aggiornare il valore Q (+ addestrare il modello)

Tutto inizia inizializzando un valore Q, che in questo caso sarà un valore casuale proveniente dal modello. Successivamente, si richiama il “model.predict(state)” e talvolta sarà un valore casuale, poiché inizialmente conosciamo poco sul gioco. In seguito, ci sarà un compromesso tra mossa casuale e previsione effettiva, perché non avremo più bisogno di mosse casuali (Esplorazione ed Exploitazione). Dopo questo, eseguiamo l'azione, misuriamo la ricompensa, aggiorniamo il valore Q e addestriamo il modello, ripetendo il ciclo.

Per addestrare il modello, è necessario usare un tipo di funzione di perdita per ottimizzare la funzionalità complessiva, e utilizzeremo l'**Equazione di Bellman**:

* Il nuovo valore Q è determinato dal valore Q attuale, più il tasso di apprendimento moltiplicato per la ricompensa, più il tasso di sconto moltiplicato per la massima ricompensa futura prevista.

Immagine che contiene Carattere, testo, calligrafia, tipografia

Descrizione generata automaticamente O più semplicemente

Immagine che contiene Carattere, tipografia, design

Descrizione generata automaticamenteda cui: