Progetto Di Sistemi Intelligenti avanzati

Edoardo Carrer

Matricola: 967549

A.A. 2022-2023

Sommario

Introduzione

Il seguente progetto riguarda l'implementazione di un agente che apprende a giocare tramite apprendimento con rinforzo al videogame "Snake" del 1976 con l'algoritmo Watkins Q(λ), cercando quindi di massimizzare il suo punteggio ad ogni partita. Per lo sviluppo è stata usata la piattaforma per lo sviluppo real time Unity e il linguaggio C#.

Il progetto si basa sul lavoro precedentemente svolto da Andre Delia nell'anno accademico 2019-20.

Definizione del Problema

Snake è un gioco che consiste nel muovere un serpente all'interno di una griglia 20x20, avendo a disposizione come azioni lo spostamento della testa del serpente in una delle 4 direzioni cardinali. L'obbiettivo dell'agente è raggiungere un frutto posizionato in una cella della griglia. Ogni qual volta si raggiunga il frutto, la lunghezza della coda del serpente viene incrementata di uno ed un altro frutto viene posizionato in una cella libera della griglia. Il gioco termina quando il serpente si muove contro la sua coda o quando finisce fuori dalla griglia. Il serpente incomincia ogni episodio con una lunghezza di tre celle. L'obbiettivo dell'agente è mangiare più frutti possibili durante una partita.

Struttura dell'agente

Per realizzare il nostro agente vogliamo utilizzare la tecnica dell'apprendimento con rinforzo, abbiamo quindi bisogno di definire lo **stato** in cui si trova in un determinato istante, delle **azioni** che gli permettano di interagire con l'ambiente, e dei **reward** ricevuti dall'ambiente per valutare le azioni.

Stato

Per permettere all'agente di apprendere è fondamentale costruire uno stato che tenga in considerazione le informazioni più importanti riguardo alla situazione in cui si trova la griglia, senza però aumentare esponenzialmente la complessità del sistema.

Considerando quindi il nostro problema un buon compromesso è definire cosa circonda la testa del serpente nelle quattro celle adiacenti in un determinato istante e un'informazione riguardo alla direzione da seguire per andare verso il frutto.

La griglia è divisa in righe e colonne, la prima riga è la più in basso nell'interfaccia, mentre la prima colonna è la più a sinistra nell'interfaccia, questa informazione ci permette di indicare la posizione relativa del frutto rispetto alla testa del serpente.

Lo stato, quindi, sarà un vettore di sei interi così definiti:

[Up, Down, Right, Left, FruitVertical, FruitHorizontal]

Up, Down, Right, Left: rappresentano le celle adiacenti nelle 4 direzioni cardinali, valgono 0 se la casella è vuota, 1 se nella casella è presente la coda del serpente o la fine della griglia, -1 se è la direzione di provenienza del serpente.

FruitVertical: rappresenta il verso da prendere nella direzione verticale per raggiungere il frutto, vale 1 se il frutto è in una riga superiore, 0 se è nella stessa riga, -1 se è in una riga inferiore rispetto al serpente.

FruitHorizontal: rappresenta il verso da prendere nella direzione orizzontale per raggiungere il frutto, vale 1 se il frutto è in una colonna successiva, 0 se è nella stessa colonna, -1 se è in una colonna precedente rispetto al serpente.

Azioni

Le azioni sono ciò che permettono all'agente di agire sull'ambiente e di raggiungere l'obbiettivo, massimizzare il punteggio del gioco. In questo caso l'agente può spostarsi ad ogni istante in tre direzioni cardinali diverse, tutte le quattro direzioni meno quella di provenienza in cui è presente la coda.

Le azioni disponibili sono:

- **Up**: l'agente si muove sulla riga superiore
- **Down**: l'agente si muove sulla riga inferiore
- **Right**: l'agente si muove sulla colonna successiva.
- **Left**: l'agente si muove sulla colonna precedente.

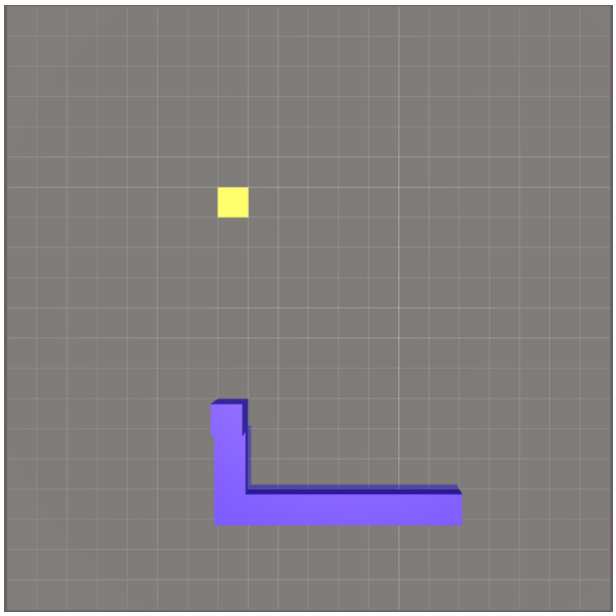
Reward

I reward sono il **feedback istantaneo** che l'ambiente fornisce all'agente riguardo all'azione, è quindi fondamentale scegliere dei reward che permettano all'agente di capire quali azioni o sequenze di azioni portino ad un buon risultato. Nel caso di Snake abbiamo deciso quindi di avere dei reward per:

- Collisione o uscita dalla griglia: -5
- Spostamento in direzione del frutto: 2
- Spostamento in direzione diversa da quella del frutto: -0.5

L'agente, quindi, riceve un feedback dall'ambiente dopo ogni azione.

Nell'esempio sottostante lo stato è [0, -1, 0, 0, 1, 0], l'azione Up sarebbe l'unica a dare un reward istantaneo positivo di due, mentre le azioni Right e Left darebbero reward negativo, l'azione Down non viene valutata in quanto l'azione precedente è stata Up e quindi la coda del serpente si trova nella cella sottostante.



Algoritmo

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a
Repeat (for each episode):
  Initialize s, a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $a^* \leftarrow \arg\max_b Q(s', b)$ (if a' ties for the max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
                    else $e(s,a) \leftarrow 0$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

Algoritmo WatkinsQ(λ)

L'algoritmo utilizzato per l'implementazione dell'agente che apprende tramite apprendimento con rinforzo è il Watkins Q(λ). L'algoritmo implementato differisce dal Q-learning per l'utilizzo della traccia, dove l'eleggibilità di tutte le coppie stato-azione visitate nell'episodio corrente vengono poste a 0 quando viene scelta un'azione diversa da quella con il Q-value migliore, detta anche azione greedy. I Q-value rappresentano il reward a lungo termine di una determinata coppia stato-azione e sono salvate in una tabella chiamata Q-table.

La Q-table è rappresentata nel codice da un Dictionary che può contenere coppie chiave-valore, dove la chiave è la coppia stato-azione, e il valore è il Q-value corrispondente. I valori della Q-table sono inizializzati a 0 all'apertura dell'applicazione e quando viene premuto il bottone di reset.

La scelta dell'azione è definita tramite la **policy**, nel progetto è stata implementata la policy Pursuit, in cui la probabilità di scegliere una determinata azione è proporzionale al Q-value associato alla specifica coppia stato-azione, nel caso in cui una coppia stato-azione avesse valore negativo viene ignorata, nel caso in cui le coppie stato-azione avessero tutte Q-value negativo, viene scelta l'azione corrispondente al Q-value maggiore.

L'algoritmo ha bisogno dei seguenti parametri per funzionare:

- α: Tasso di apprendimento
- γ: Fattore di sconto
- λ: Fattore di sconto per la traccia

l'aggiornamento dei Q-value avviene con la seguente formula, dove al valore di Q al tempo k viene sommato il delta, che rappresenta l'errore sulla stima, moltiplicato per il tasso di apprendimento,

$$Q^{\pi}_{k+1}(s,a) = Q^{\pi}_{k}(s,a) + \alpha\delta_t(s_t,a_t)e_t(s,a)$$

$$\delta_t(s_t,a_t) = \left[r' + \gamma Q^{\pi}(s_{t+1},a_{t+1}) - Q^{\pi}(s_t,a_t)\right]$$

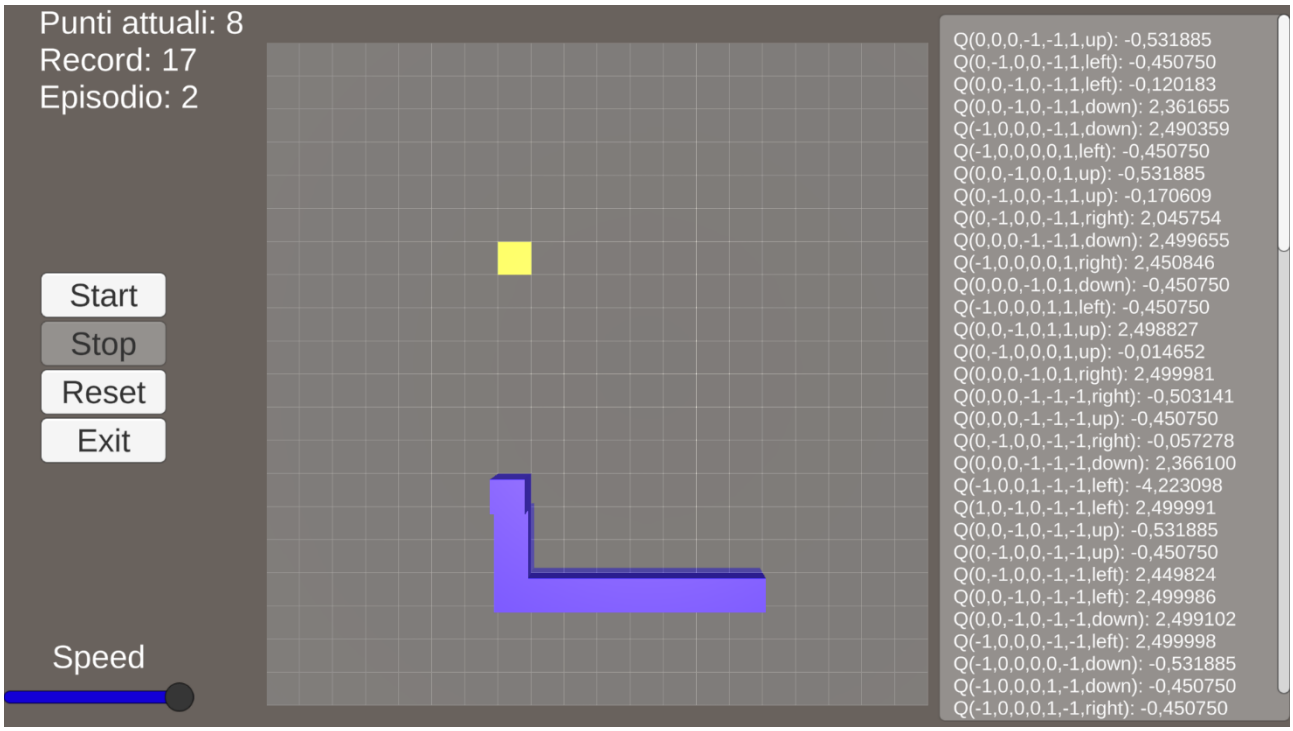
L'errore sulla stima è calcolato come il reward istantaneo appena ricevuto dall'agente più il Q-value associato alla coppia stato-azione successiva, scontato di γ, meno il Q-value della coppia stato azione attuale. La coppia stato-azione successiva è data dallo stato in cui l'agente arriva dopo aver completato l'azione corrente, e la scelta dell'azione che non avviene secondo policy, ma in base al Q-value maggiore, questa scelta nell'aggiornamento rende gli algoritmi basati sul Q-learning **off-policy**.

L'eleggibilità viene viene incrementata di uno per la coppia-stato azione corrente, scontata con i fattori γ e λ nel caso in cui l'azione attuale fosse un'azione greedy oppure posta a 0 altrimenti.

$$e_t(s,a) = \mathcal{I}_{ss_t} \cdot \mathcal{I}_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a); \\ 0 & \text{otherwise,} \end{cases}$$

L'algoritmo implementato utilizza i parametri α = 0.9, γ = 0.2, λ = 0.9 che sono risultati quello più performanti nei test svolti, sono stati fatte delle analisi anche su policy ε-greedy e parametri differenti.

Interfaccia Grafica



L'interfaccia grafica è molto semplice e presenta sulla sinistra dall'alto verso il basso:

- Info sulla situazione attuale, punti dell'episodio corrente, record di punti e il numero dell'episodio corrente.
- Bottoni per incominciare, stoppare o resettare l'addestramento, e il bottone per chiudere la finestra.
- Slider per cambiare la velocità dell'agente.

Sulla destra invece è presenta la Q-table con i valori in aggiornamento.

Risultati

Abbiamo testato l'agente utilizzando due policy differenti, ε-greedy e pursuit, per analizzarne il comportamento.

Policy ε-greedy

ε-greedy è una policy che sceglie con probabilità 1-ε l'azione migliore disponibile, e con probabilità ε un'azione casuale.

Parametri: α = 0.8, γ = 0.2, λ = 0.8, ε = 0.1.

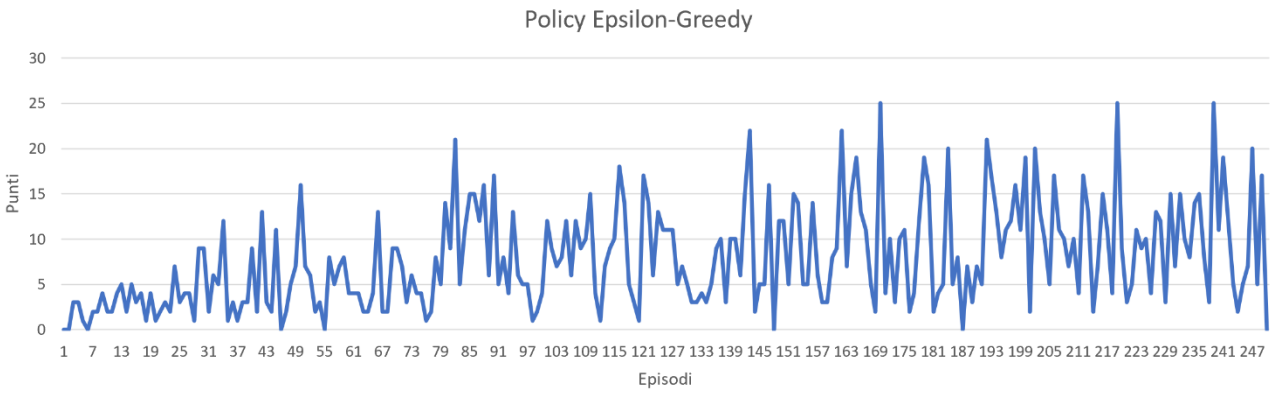


Grafico del punteggio di un'agente con policy Epsilon-greedy e parametri $\alpha = 0.9$, $\gamma = 0.2$, $\lambda = 0.9$, $\varepsilon = 0.1$

Come è ben visibile dal grafico, la policy ε-greedy permette un apprendimento lento nel tempo, l'utilizzo di mosse randomiche porta l'agente a sbagliare più spesso e ad avere quindi valori bassi di punteggio anche dopo molti episodi.

Policy Pursuit

Abbiamo provato questa policy con parametri leggermente diversi.

Parametri: $\alpha = 0.8$, $\gamma = 0.2$, $\lambda = 0.8$

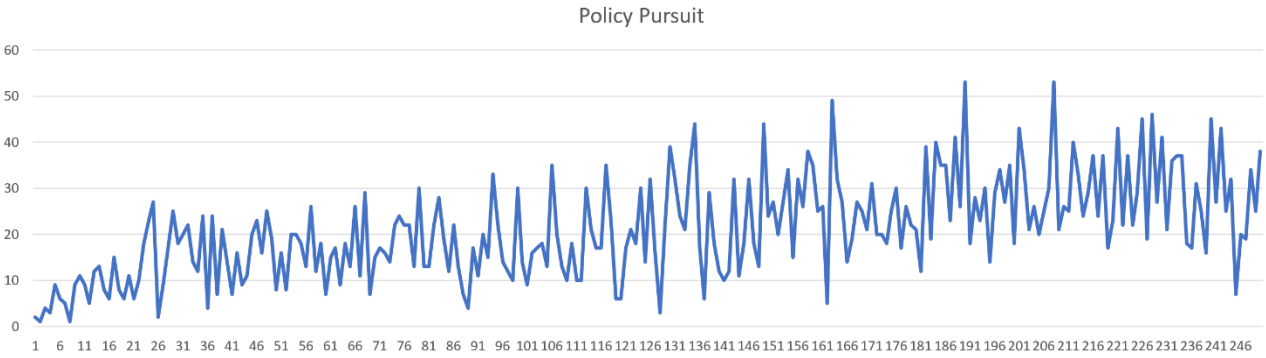


Grafico del punteggio di un'agente con policy Pursuit e parametri $\alpha = 0.8$, $\gamma = 0.2$, $\lambda = 0.8$

Parametri: $\alpha = 0.9$, $\gamma = 0.2$, $\lambda = 0.9$

Con questa policy abbiamo ottenuto il miglior rendimento medio e il miglior risultato, 72 punti dopo 230 episodi, ci riferiremo a questa con pursuit2 per distinguerla dalla precedente.

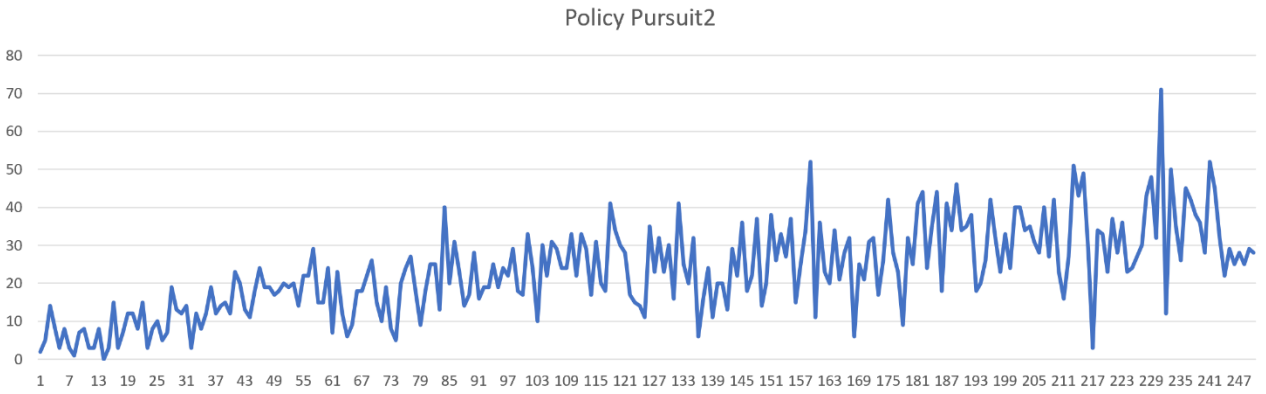


Grafico del punteggio di un'agente con policy Pursuit e parametri $\alpha = 0.9$, $\gamma = 0.2$, $\lambda = 0.9$

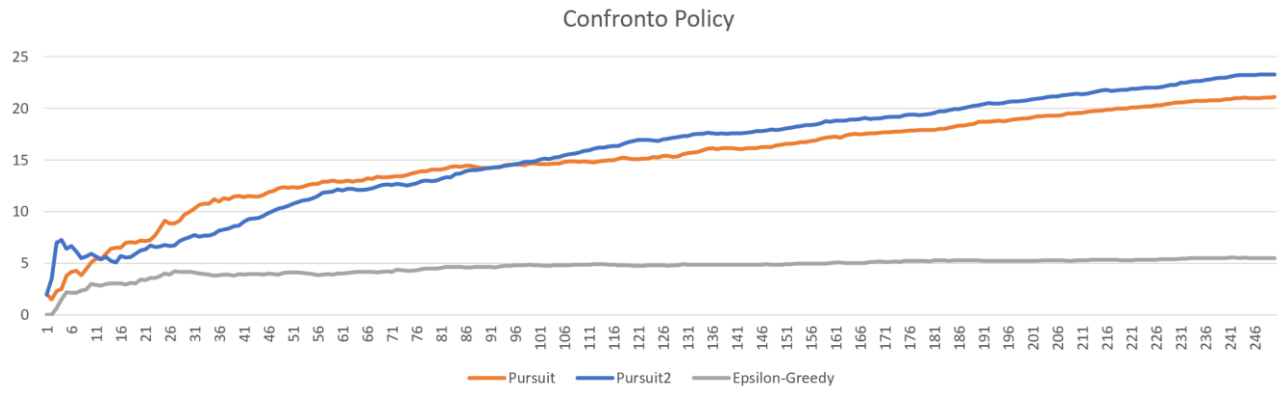


Grafico di confronto tra policy

È chiaro come con la policy Pursuit l'agente apprenda molto più rapidamente rispetto alla ε-greedy, e vediamo come dare un valore maggiore al tasso di apprendimento e alla traccia migliori l'abilità dell'agente che utilizza questa policy. Inizialmente l'agente, non conoscendo l'ambiente, è molto più sensibile all'eventualità di compiere mosse sbagliate che portano l'agente ad avere punteggi bassi; infatti, dopo una decina di episodi la policy pursuit ha una media maggiore di punteggio, ma dopo poco meno di 100 episodi la media della policy pursuit2 incrementa più velocemente.

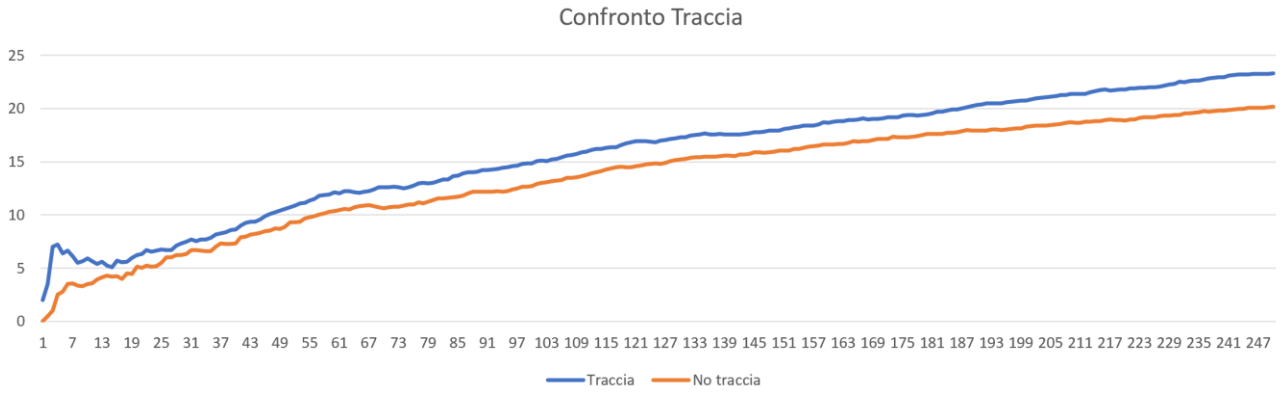


Grafico di confronto tra policy pursuit con e senza traccia

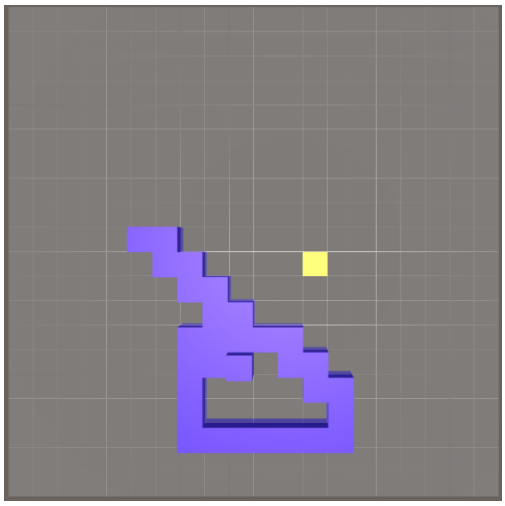
In quest'ultimo grafico abbiamo testato l'algoritmo con la policy pursuit2 con e senza traccia, è possibile vedere come la traccia migliori in generale le prestazioni dell'agente, velocizzando il tempo per trovare la strategia migliore e in generale per ottenere punteggi più alti quando la lunghezza della coda cresce.

Considerazioni finali

Dopo 250 episodi si nota come l'agente arrivi alla capacità di evitare la propria coda e di finire fuori dalla griglia. La strategia sembra essere sempre quella di allinearsi con il frutto per poi procedere dritto verso di esso.

Questo ci porta anche alla considerazione del fatto che superata una certa lunghezza questa tecnica diventi inefficace portando quindi l'agente a scontrarsi appunto con sé stesso, essendo la sua visione limitata a sole quattro celle, chiudendosi intorno alla sua stessa coda senza possibilità di uscirne, questo è estremamente dipendente dalla posizione in cui viene generato il frutto ogni qual volta venga mangiato ed è quindi causa di fluttuazioni nei punteggi.

Nell'esempio sottostante dopo aver mangiato un frutto, quello successivo viene generato in una posizione che costringe l'agente, dopo aver effettuato due mosse che lo portavano verso la sua direzione, a finire chiuso dalla sua coda.



Situazione principale di fine episodio dopo circa 250 episodi

Per poter ottenere dei risultati migliori la soluzione potrebbe essere quella di ampliare il concetto di stato dell'agente, permettendogli di avere una visione più ampia, questo però porterebbe rapidamente l'aumento della complessità generale dell'algoritmo.