RELAZIONE TAXI DRIVER 2019/2020

19 SETTEMBRE

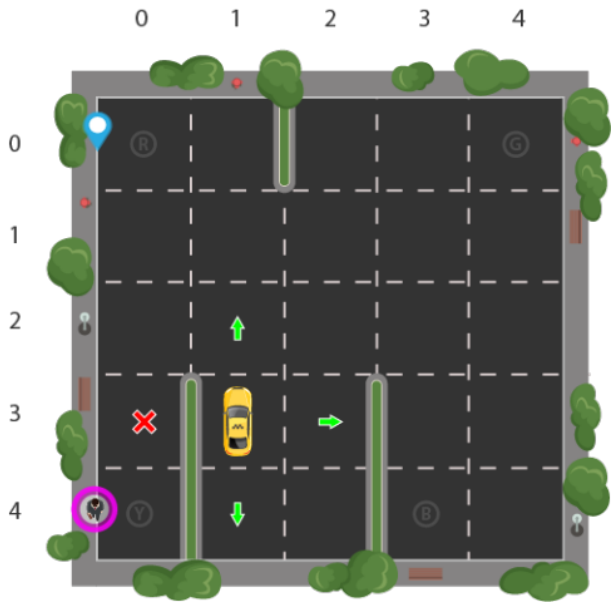
INGEGNERIA DELLA CONOSCENZA

GRUPPO: Luigi Francesco Pio Camporeale, 677267
Christian Vincenzo Madera, 676704
Giuseppe Pio Pracella, 677712

EMAIL: luigicamporeale99@gmail.com
chris.madera99@gmail.com
g.pra.giu@gmail.com

1. INTRODUZIONE

In Taxi driver, l'agente è il taxi che si muove in una griglia di dimensione $5 * 5$. All'interno della griglia sono presenti 4 lettere R, G, B, Y , le quali rappresentano 4 destinazioni differenti. Un passeggero può essere in una delle 4 destinazioni e può raggiungere solo le 3 rimanenti (questo significa che nessun passeggero potrà mai tornare nel punto di partenza).



1.1 GESTIONE PUNTI

- **Guadagna 20 punti se porta a destinazione un passeggero**
- **Perde 10 punti se prende un passeggero in un punto diverso da R, G, B, Y**
- **Perde 10 punti se lascia un passeggero in una destinazione diversa da R, G, B, Y**
- **Perde 1 punto ad ogni azione**

1.2 COMPORTAMENTO ATTESO

- **Il Taxi deve trovare il passeggero percorrendo il percorso più breve**
- **Il Taxi deve prendere il passeggero**
- **Il Taxi deve trovare il percorso più breve per portare il passeggero a destinazione**

1.3 SPAZIO DEGLI STATI

Nel Reinforcement Learning, l'agente incontra uno stato e quindi agisce in base allo stato in cui si trova. Lo spazio degli stati è l'insieme di tutte le possibili situazioni che il nostro taxi potrebbe vivere. Lo stato dovrebbe contenere informazioni utili di cui l'agente ha bisogno per compiere l'azione corretta.
Supponiamo che il Taxi sia l'unico veicolo in questo parcheggio. Possiamo suddividere il parcheggio in una griglia $5 * 5$, che ci dà 25 possibili posizioni del taxi. Queste 25 locations sono una parte del nostro spazio degli stati.

Notiamo anche che ci sono quattro posizioni dove possiamo prendere e lasciare un passeggero: R, G, Y, B o $[(0, 0), (0, 4), (4, 0), (4, 3)]$ nelle coordinate (riga, colonna).

In questo ambiente ci sono $25 * 5 * 4 = 500$ stati possibili totali, dove:

- **25 stati rappresentano le posizioni in cui si può trovare il taxi**
- **$4 + 1$, dove:**
  - **4 rappresentano le posizioni in cui si può trovare il passeggero**
  - **1 la possibilità che il passeggero si trovi nel taxi**
- **4 rappresenta le destinazioni possibili.**

1.4 SPAZIO DELLE AZIONI

Quando l'agente si trova in uno dei 500 stati possibili, deve scegliere l'azione da svolgere. Nel nostro caso, il taxi può svolgere due tipologie di azione, ovvero:

- **Muoversi in una direzione**
- **Decidere di far salire il passeggero o di farlo scendere**

In poche parole, abbiamo 6 possibili azioni:

- **South**
- **North**
- **West**
- **Est**
- **Pickup**
- **Dropoff**

Queste 6 azioni rappresentano lo spazio delle azioni, ovvero l'insieme delle azioni tra cui l'agente può scegliere quando si trova in uno stato.

2. REINFORCEMENT LEARNING

Il Reinforcement Learning (RL) è un'area del machine learning che si concentra su come un agente può agire in un ambiente per massimizzare le ricompense che riceve. Un agente di RL deve determinare quali azioni svolgere in base alle sue percezioni e alle ricompense che riceve. Utilizzando un gioco come esempio di ambiente, il RL si concentra su come il giocatore può scegliere di agire (per esempio muoversi in una certa direzione) per massimizzare una ricompensa, che in questo caso potrebbe essere il punteggio.

2.1 PROCESSO DECISIONALE DI MARKOV

I processi decisionali di Markov (MDP) formalizzano il processo decisionale sequenziale che è alla base del RL.
In un MDP, l'agente interagisce con l'ambiente in cui si trova. Le interazioni sono sequenziali nel tempo. Ad ogni istante T, l'agente ottiene una rappresentazione dello stato dell'ambiente. Data questa rappresentazione, l'agente decide l'azione da compiere. L'ambiente quindi entra in un nuovo stato e l'agente riceve una ricompensa che è una conseguenza della sua azione precedente.
In un MDP abbiamo:

- **Un agente**
- **Un ambiente**
- **Gli stati**
- **Le azioni possibili**
- **Le ricompense**

Il processo di selezionare un'azione da un dato stato, transitare verso un nuovo stato e ricevere una ricompensa, avviene ciclicamente, creando una sequenza di stati, azioni e ricompense. L'obiettivo dell'agente è massimizzare la somma delle ricompense che riceve scegliendo delle azioni in degli stati precisi. Ciò significa che l'agente dovrebbe massimizzare non solo la ricompensa immediata, ma le ricompense accumulate nel tempo.
In un MDP abbiamo:

- **S un set di stati**
- **A un set di azioni**
- **R un set di ricompense**

Ad ogni istante $t = 0, 1, 2, 3, \ldots$, l'agente si trova in uno stato $S_t \in S$ e seleziona una azione $A_t \in A$ che formano una coppia stato-azione (S_t, A_t). Possiamo pensare alla ricezione di una ricompensa come una arbitraria funzione f che mappa coppie stato-azione su ricompense. In un dato istante t abbiamo che:

$$f(St, At) = R_t + 1$$

Si formerà dunque una sequenza del tipo:

$$S0, A0, R1, S1, A1, R2, S2, A2, R3, \ldots$$

Gli insiemi S e R sono finiti, le variabili aleatorie R_t e S_t hanno delle distribuzioni di probabilità definite. Tutti i possibili valori che possono essere assegnati a R_t e S_t hanno delle probabilità associate che dipendono da S_{t-1} e da A_{t-1}, stato precedente e azione effettuata al tempo $t-1$. Per esempio, supponiamo che $s' \in S$ e $r \in R$. C'è qualche probabilità che $S_t = s'$ e $R_t = r$. Questa probabilità è determinata dallo stato precedente $s \in S$ e l'azione $a \in A(s)$, dove $A(s)$ è il set di azioni che si possono effettuare nello stato s.

3. RENDIMENTO ATTESO (o EXPECTED RETURN)

L'obiettivo di un agente in un MDP è quello di massimizzare le sue ricompense cumulative. Abbiamo bisogno di un modo per aggregare e formalizzare queste ricompense. Per questo, si introduce il concetto di rendimento atteso. Si può pensare al rendimento come la somma delle future ricompense. Quindi possiamo definire il rendimento G al tempo t come:

$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$, dove T è il passo di tempo finale.

L'obiettivo dell'agente è cioè quello di massimizzare il rendimento atteso delle ricompense. Il rendimento atteso è ciò che guida l'agente nel prendere le decisioni. Questa definizione va bene per task "episodici", cioè in quei task che hanno un tempo finito T. Esistono anche dei task continui che non hanno limite di tempo.

4. IPERPARAMETRI

4.1 TASSO DI APPRENDIMENTO α

Il tasso di apprendimento determina con quale estensione le nuove informazioni acquisite sovrascriveranno le vecchie informazioni. Un fattore 0 impedirebbe all'agente di apprendere, al contrario un fattore pari ad 1 farebbe sì che l'agente si interessi solo delle informazioni recenti.

4.2 FATTORE DI SCONTO γ

Il fattore di sconto determina l'importanza delle ricompense future. Un valore di gamma uguale a 0 rende l'agente "opportunista", in quanto considera solo la ricompensa attuale r; al contrario, un valore di gamma prossimo a 1 permette di cercare ricompense anche a lungo termine. Per valori maggiori di 1 i valori di Q possono divergere.

5. POLICIES E VALUE FUNCTIONS

5.1 POLICIES

Una policy o politica è una funzione che ha in input uno stato e restituisce la probabilità di selezionare ogni azione possibile da quello stato. Indichiamo una policy con π. Se un agente segue una policy π allora nell'istante t, $\pi\ (a \mid s)$ è la probabilità di scegliere l'azione a sé ci si trova nello stato s. La policy risponde alla domanda "Quanto è probabile per un agente selezionare una data azione da un dato stato?".

5.2 VALUE FUNCTIONS

Una Value Function è una funzione degli stati o di coppie stato-azione che stima quanto è buono per un agente essere in un dato stato o quanto è buono scegliere una specifica azione partendo da uno stato specifico. Le Value Functions sono definite rispetto alla policy che un agente segue.

5.3 STATE-VALUE FUNCTION

La funzione State-Value per la policy π, denotata come v_π, ci dice quanto è buono essere in uno stato s per un agente che segue la policy π. Il valore di uno stato s per la policy π è il rendimento atteso che si ha partendo dallo stato s al tempo t e poi seguendo la policy π.

$$v_\pi = E_\pi[G_t|S_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right]$$

dove γ è il discount rate, compreso nell'intervallo $[0, 1]$ che serve a considerare maggiormente i reward a breve termine rispetto a quelli a lungo termine

5.4 ACTION-VALUE FUNCTION

La funzione Action-Value per la policy π, denotata come q_π, ci dice quanto è buono per un agente scegliere una specifica azione a parendo da un dato stato s seguendo la policy π.

$$q_\pi = E_\pi[G_t|S_t = s, A_t = a] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]$$

La Action-Value Function q_π è chiamata Q-Function e il suo output per una coppia stato-azione è chiamato Q-value, dove la lettera Q sta per qualità di una coppia stato-azione.

5.5 POLICY OTTIMALE

L'obiettivo degli algoritmi di RL è trovare la policy che porterà l'agente a guadagnare molte ricompense se l'agente la segue. Una policy ottimale porta l'agente ad ottenere più ricompense di qualunque altra policy. Una policy π è considerata migliore o uguale alla policy π' se l'expected return di π è maggiore o uguale a quello di π' per tutti gli stati.

$$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v_{\pi'}(s) \; \forall s \in S$$

5.6 FUNZIONE STATE-VALUE OTTIMALE

La funzione State-Value ottimale, denotata con v_* è così definita:

$$v_*(s) = \max_\pi v_\pi(s) \; \forall s \in S$$

v_* in output ha l'expected return più alto che si può ottenere seguendo una policy per ogni stato.

5.7 FUNZIONE ACTION-VALUE OTTIMALE

La funziona Action-Value ottimale, denotata con q_* ed è così definita:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \; \forall s \in S \wedge a \in A(s)$$

q_* in output ha l'expected return più alto che si può ottenere seguendo la policy che lo massimizza per ogni coppia stato-azione.

5.8 EQUAZIONE DI OTTIMALITA' DI BELLMAN per q_*

Una proprietà fondamentale di q_* è quella che deve soddisfare questa equazione:

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

Questa equazione è chiamata Equazione di ottimalità di Bellman. Data una coppia stato-azione (s,a), al tempo t, l'expected return partendo dallo stato s, selezionando l'azione a, e poi seguendo la policy ottimale (cioè il Q-value della coppia) è la ricompensa che si ottiene selezionando l'azione a nello stato s, cioè R_{t+1}, più il massimo expected return che può essere raggiunto da una qualsiasi tra le prossime possibili coppie stato azione (s',a') scontato di γ. Siccome l'agente sta seguendo una policy ottimale, lo stato successivo s' sarà lo stato da cui la migliore prossima azione a' può essere scelta al tempo t + 1, dove per migliore azione si intende quella che massimizzerà l'expected return.

6. Q-LEARNING

Il Q-Learning è una tecnica in grado di risolvere il problema di trovare una policy ottimale per un MDP. L'obiettivo del Q-Learning è trovare una policy ottimale in modo che il valore atteso della somma delle ricompense di tutti gli step successivi sia il massimo possibile. Il Q-Learning cerca una policy ottimale imparando i Q-values ottimali per ogni coppia stato-azione.

6.1 VALUE ITERATION

L'algoritmo Q-Learning aggiorna iterativamente i Q-values per ogni coppia stato-azione utilizzando l'Equazione di Bellman fino a quando la Q-function converge verso la Q-function ottimale, q_*.

6.2 Q-TABLE

La Q-table è una tabella azioni stati in cui l'agente memorizza i Q-values per ogni coppia stato-azione. Quando la Q-table viene aggiornata l'agente può utilizzarla per scegliere l'azione con il Q-value più alto a partire dallo stato in cui si trova.

6.3 EPISODI

L'agente per imparare dovrà giocare al gioco più volte. Ogni partita che l'agente gioca si chiama episodio. L'agente potrebbe scegliere le azioni basandosi sui valori memorizzati nella Q-table, ma durante il primo episodio tutti i Q-values sono settati a 0. L'agente dovrebbe avere un modo per esplorare l'ambiente prima di poter sfruttare la Q-table. Per questo vengono introdotti i concetti di Exploration ed Exploitation.

6.4 EXPLORATION VS. EXPLOITATION

In ogni episodio, in base allo stato in cui si trova, l'agente seleziona l'azione con il Q-value più alto stimato nella Q-table. Dal momento che, durante il primo episodio, nello stato iniziale tutti i Q-values sono inizializzati a zero, non c'è nessun modo per l'agente di differenziarli e quindi non sa quale azione eseguire per prima. Inoltre, negli stati successivi, selezionare l'azione con il più alto Q-value per lo stato corrente non è sempre la scelta migliore. Perciò abbiamo bisogno di un trade-off tra exploration ed exploitation per scegliere le nostre azioni. Questo trade-off è ottenuto utilizzando la epsilon greedy strategy.

6.5 EPSILON GREEDY STRATEGY

Per avere il bilanciamento giusto tra exploration ed exploitation, usiamo la cosiddetta epsilon greedy strategy. Con questa strategia, definiamo un exploration rate e che inizialmente impostiamo ad 1. Questo exploration rate è la probabilità che il nostro agente esplorerà l'ambiente piuttosto che sfruttare la Q-table scegliendo sempre l'azione migliore. Con e = 1 , è sicuro al 100% che l'agente inizierà esplorando dell'ambiente.
All'inizio di ciascun nuovo episodio, l'exploration rate e diminuisce di una percentuale fissata. In questo modo, nel momento in cui l'agente acquisisce più informazioni riguardo l'ambiente, inizia a sfruttare più frequentemente la Q-table creata. L'agente diventerà "greedy" in termini di sfruttamento dell'ambiente una volta che avrà avuto l'opportunità di esplorarlo e imparare quanto più possibile da esso.
Per determinare se l'agente sceglierà l'exploration o l'exploitation ad ogni tempo t, generiamo un numero random tra 0 ed 1. Se questo numero è più grande di e, allora l'agente sceglierà l'azione successiva mediante exploitation ovvero sceglierà l'azione in base al più alto Q-value per lo stato corrente dalla Q-table. Altrimenti, l'azione successiva sarà scelta mediante exploration, ovvero l'azione sarà scelta in modo casuale per esplorare l'ambiente.

```
1    if random_num > e } :
2    #choose action via exploitation
3    else :
4    #choose action via exploration
```

6.6 SCEGLIERE UN'AZIONE

A questo punto, la prima azione sarà scelta stocasticamente attraverso l'esplorazione dal momento che il nostro exploration rate è impostato ad 1. Questo sta a significare che, con il 100% di probabilità, l'agente esplorerà l'ambiente durante il primo episodio del gioco piuttosto che sfruttarlo. Dopo che l'agente compie un'azione, osserva il prossimo stato, osserva il reward ottenuto dall'azione corrente e aggiorna il Q-value nella Q-table per l'azione che ha intrapreso dallo stato precedente.

6.7 AGGIORNARE IL Q-VALUE

Per aggiornare il Q-value per la coppia stato-azione usiamo l'equazione di Bellman accennata precedentemente:

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

Data la coppia stato-azione (s, a), l'obiettivo è rendere il Q-value $q(s, a)$ il più possibile vicino alla parte destra dell'equazione così che il Q-value eventualmente possa convergere al valore di $q_*(s, a)$. Per avere la convergenza, ogni volta che incontriamo una coppia (s, a):

- **Si calcola la perdita tra il Q-value e il Q-value ottimale per (s, a)**
- **Si aggiorna il Q-value al fine di diminuire la perdita.**

$$q_*(s, a) - q(s, a) = loss$$

$$E\left[R_{t+1} + \gamma \max_{a'} q_*(s', a')\right] - E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] = loss$$

Per vedere come aggiornare il Q-value, bisogna prima introdurre l'idea di learning rate.

6.8 LEARNING RATE

Il learning rate è un numero tra 0 e 1 che può essere pensato come quanto velocemente l'agente abbandona il Q-value precedente nella Q-table (data una coppia stato-azione) per un nuovo Q- value. Supponiamo di avere un Q-value nella Q-table per alcune coppie stato-azione arbitrarie che l'agente ha avuto modo di apprendere allo step precedente. Se l'agente apprendesse le stesse coppie stato-azione al passo successivo quando ha imparato di più circa l'ambiente, il Q-value avrà bisogno di essere aggiornato per riflettere il cambiamento nelle aspettative che l'agente possiede in quel momento per il futuro.
Non vogliamo solo sovrascrivere il vecchio Q-value ma, piuttosto, vogliamo usare il learning rate come uno strumento per determinare quanta informazione mantenere circa il precedente Q-value data la coppia stato-azione, contro il nuovo Q-value calcolato per la stessa coppia stato-azione al passo successivo. Denotiamo il learning rate con il simbolo α e assegneremo arbitrariamente α = 0.7.
Più è alto il valore di learning rate, più velocemente l'agente adotterà il nuovo Q-value. Per esempio, se il learning rate è 1, la stima per il Q-value, data la coppia stato-azione, sarebbe direttamente il nuovo Q-value calcolato e non verrebbe considerato il precedente Q-value che è stato calcolato, data la coppia stato-azione al passo precedente.

6.9 CALCOLARE IL NUOVO Q-VALUE

La formula per calcolare il nuovo Q-value per una coppia stato-azione (s, a) al tempo t è:

$$q^{new}(s, a) = (1 - \alpha) q(s, a) + \alpha\left(R_{t+1} + \gamma \max_{a'} q\,(s', a')\right)$$
$$= (1 - 0.7)(0) + 0.7\left(-1 + 0.99(\max_{a'} q\,(s', a'))\right)$$

Così il nuovo Q-value è uguale alla somma pesata del vecchio Q-value e il Q-value ottimo in base al learning value. Il vecchio valore nel nostro caso è 0 perché è la prima volta che l'agente sta incontrando questa particolare coppia stato-azione e moltiplichiamo questo vecchio valore per $(1-\alpha)$. Il nostro valore appreso è il reward che l'agente riceve muovendosi dallo stato iniziale più la stima scontata dell'ottimo futuro Q-value per la prossima coppia stato-azione (s', a') al tempo t + 1. Questo valore appreso è poi moltiplicato per il nostro learning rate. Supponendo che il discount rate γ = 0.99 avremmo:

$$q^{new}(s,a) = (1-\alpha)q(s,a) + \alpha\left(R_{t+1} + \gamma \max_{a'} q\,(s', a')\right)$$
$$= (1-0.7)(0) + 0.7\left(-1 + 0.99(\max_{a'} q\,(s', a'))\right)$$

Focalizziamoci sul termine $\max_{a'} q(s', a')$. Dal momento che tutti i Q-values sono inizializzati a 0 nella Q-table abbiamo:

$$\max_{a'} q(s', a') = max\big(q(empty6, left), q(empty6, right), q(empty6, up), q(empty6, down)\big)$$
$$= max(0,0,0,0) = 0$$

Adesso, possiamo sostituire il valore 0 in $\max_{a'} q(s', a')$ nella precedente equazione per risolvere $q^{new}(s,a)$.

$$q^{new}(s,a) = (1-\alpha)\, q(s,a) + \alpha(R_{t+1} + \gamma \max_{a'} q(s', a')) = (1-0.7)(0) + 0.7(-1 +$$
$$0.99(\max_{a'} q(s', a'))) = (1-0.7)(0) + 0.7(-1 + 0.99(0)) = 0 + 0.7(-1) = -0.7$$

Adesso prenderemo questo nuovo Q-value appena calcolato e lo memorizzeremo nella nostra Q-table per questa particolare coppia stato-azione. Questo è tutto ciò che serve per un singolo step. Lo stesso processo avverrà ogni step successivo finché l'episodio non termina. Quando la Q-function converge alla Q-function ottimale, avremo la nostra ottima policy.

6.10 NUMERO MASSIMO DI STEP

Possiamo anche specificare un numero massimo di step che il nostro agente esegue prima che l'episodio termini anche se non raggiunge il goal. Per esempio, si potrebbe definire una condizione per cui se l'agente non ha raggiunto la terminazione entro 100 step allora il gioco terminerà al 100° step.

7. SARSA

Il SARSA è un metodo TD on-policy. Una policy è una coppia state-action e serve a mappare le azioni che possono essere intraprese in ciascuno stato. Un metodo di controllo on-policy sceglie l'azione per ogni stato durante l'apprendimento seguendo una certa policy (una delle più usate è quella di valutare sé stessa, come nella policy iteration). Il nostro obiettivo è quello di stimare $Q_\pi(s, a)$ per la policy corrente e tutti le coppie state-action *(s-a).* Per far ciò possiamo utilizzare la regola di aggiornamento TD applicata ad ogni timestep, facendo transitare l'agente da una coppia state-action ad un'altra (diversamente dalle altre tecniche RL model dependent dove l'agente transita da uno stato all'altro).

7.1 SARSA VS Q-LEARNING

La principale differenza tra il Q-Learning e il SARSA è che il Q-Learning è off-policy, e il SARSA è on-policy. Le equazioni riportate di seguito mostrano le equazioni di aggiornamento per il Q-Learning e il SARSA.

Q-LEARNING:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

A prima vista sembrano quasi uguali, però nel Q-Learning noi aggiorniamo la nostra Q-function prendendo in considerazione l'azione che massimizza la Q-function $Q(s_t + 1, a)$ per lo stato successivo.

Nel SARSA, usiamo la stessa policy (epsilon-greedy) che ha generato l'azione precedente a_t per generare l'azione successiva a_{t+1}. Intuitivamente, il SARSA è on-policy perché usiamo la stessa policy per generare l'azione corrente e l'azione successive. Infine, valutiamo la selezione dell'azione da parte della nostra policy, e la miglioriamo andando a migliorare le stime della Q-function.

Per il Q-learning, non abbiamo alcun vincolo su come viene selezionata l'azione successiva, solo che abbiamo questa visione "ottimistica" che tutte le selezioni di azioni d'ora in avanti da ogni stato dovrebbero essere ottimali, quindi scegliamo l'azione *a* che massimizza $Q(s_t + 1, a)$. Ciò significa che con il Q-learning possiamo fornire dati generati da una qualsiasi politica di comportamento (politica esperta, casuale, persino cattiva) e dovrebbe apprendere comunque i valori Q ottimali.

8. IMPLEMENTAZIONE

Per risolvere l'ambiente abbiamo utilizzato diversi approcci:

- **Random Agent**
- **Q-Learning**
- **Q-Learning Genetico**
- **SARSA**
- **SARSA Genetico**

Di seguito andiamo ad illustrare e ad analizzare i risultati ottenuti con ciascun approccio.

Figura 1: Tempo di esecuzione (in secondi) sulle ordinate

8.1 RANDOM AGENT

Il primo approccio che abbiamo adottato è quello di cercare di risolvere l'ambiente senza l'utilizzo del Reinforcement Learning. Perciò abbiamo tentato un approccio brute-force per risolvere il problema. Analizzando il grafico e i dati raccolti ci rendiamo subito conto che questo tipo di soluzione non è molto valida, anche dopo migliaia di episodi di training il numero di penalties rimane molto elevato. Questo accade perché l'agente non sta imparando dalla propria esperienza passata. Infatti, anche aumentando il numero di episodi non otterremmo miglioramenti. In questo caso l'agente non ha memoria della migliore azione da svolgere per ciascuno stato, cosa che invece è alla base del Reinforcement Learning. A questo punto non ci resta che analizzare il problema sotto l'ottica di quest'ultimo.

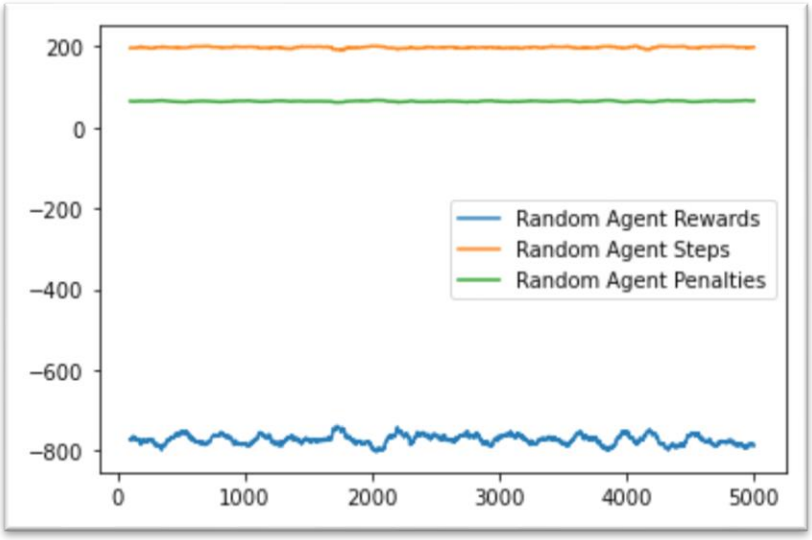


Figura 2: Numero di Training Episodes sulle ascisse e Rewards, Steps, Penalties sulle ordinate

8.2 APPROCCIO BASE

8.2.1 Q-LEARNING

Dopo aver osservato che l'approccio Random Agent risulta inefficiente al fine di risolvere l'ambiente, abbiamo deciso di applicare il Q-Learning. Nel caso di quest'ultimo abbiamo notato che dopo un certo numero di episodi il numero di penalties e di steps comincia a diminuire drasticamente, ciò significa che l'agente sta imparando dalla propria esperienza passata (proprio ciò che volevamo). I parametri utilizzati nelle figure 3 e 4 sono i seguenti:

1 $total\_episodes = 5000$
2 $total\_test\_episodes = 1000$
3 $learning\_rate = 0.1$
4 $discount\_rate\ (gamma) = 0.99$
5 $exploration\_rate\ (epsilon) = 1.0$
6 $max\_exploration\_rate\ (max\_epsilon) = 1.0$
7 $min\_exploration\_rate\ (min\_epsilon) = 0.01$
8 $exploration\_decay\_rate = 0.001$

8.2.2 SARSA

Un ulteriore approccio che abbiamo deciso di applicare è quello del SARSA. La motivazione di questa scelta è dovuta al fatto che volevamo mettere a confronto due metodi di RL, però uno on-policy e l'altro off-policy. Per permettere un confronto ottimale tra i due metodi, abbiamo utilizzato i medesimi parametri del Q-Learning.

1 $total\_episodes = 5000$
2 $total\_test\_episodes = 1000$
3 $learning\_rate = 0.1$

4 $discount\_rate\ (gamma)\ =\ 0.99$
5 $exploration\_rate\ (epsilon)\ =\ 1.0$
6 $max\_exploration\_rate\ (max\_epsilon)\ =\ 1.0$
7 $min\_exploration\_rate\ (min\_epsilon)\ =\ 0.01$
8 $exploration\_decay\_rate\ =\ 0.001$

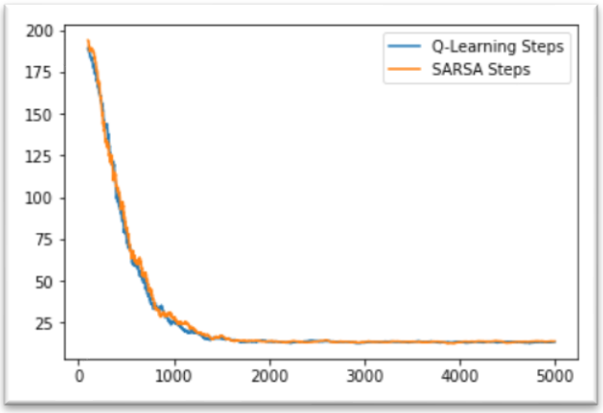


Figura 3: Numero di Training Episodes sulle ascisse e Numero di Steps sulle ordinate

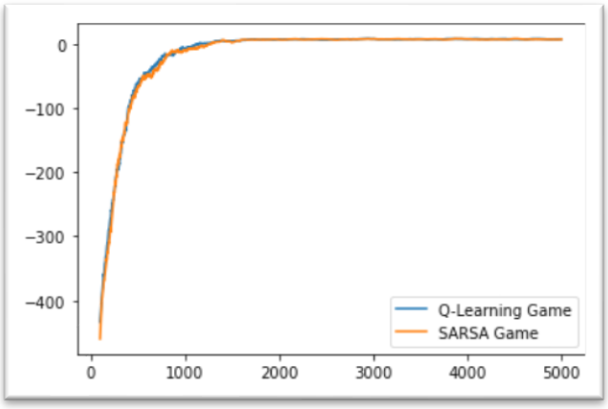


Figura 4: Numero di Rewards sulle ordinate

8.3 Q-LEARNING GENETICO

Questo approccio prevede che ci siano 10 agenti, ognuno con una sua Q-table. Le Q-table vengono inizializzate casualmente e ogni agente effettua 5.000 episodi di Q-Learning base. Vengono quindi valutate le Q-table su 1.000 episodi di test. Le Q-table che raggiungono un maggior tasso di successo avranno più probabilità di essere selezionate come genitori per il crossover (Stochastic Beam Search). Le 2 Q-table migliori vengono utilizzate anche nella generazione successiva. Il crossover avviene selezionando casualmente un indice di riga della Q-table. Sia i il numero casuale generato nell'intervallo [1, n], dove n è il numero di righe della Q-table. Il primo figlio avrà le righe da 1 a i del primo genitore e le righe da i ad n del secondo genitore.

1 $num\_training\_episodes\ =\ 5000$
2 $num\_test\_episodes\ =\ 1000$
3 $population\_size\ =\ 10$
4 $num\_generation\ =\ 10$
5 $max\_steps\ =\ 1000$
6 $learning\_rate\ =\ 0.1$
7 $discount\_rate\ (gamma)\ =\ 0.99$
8 $exploration\_rate\ (epsilon)\ =\ 1.0$
9 $max\_exploration\_rate\ (max\_epsilon)\ =\ 1.0$
10 $min\_exploration\_rate\ (min\_epsilon)\ =\ 0.01$
11 $exploration\_decay\_rate\ =\ 0.001$

Con i seguenti parametri si raggiunge un tasso di successo del 68%. I risultati sono dipendenti dalla configurazione iniziale ma generalmente l'algoritmo riesce comunque a trovare (più o meno velocemente) delle Q-Table con questo tasso di successo.

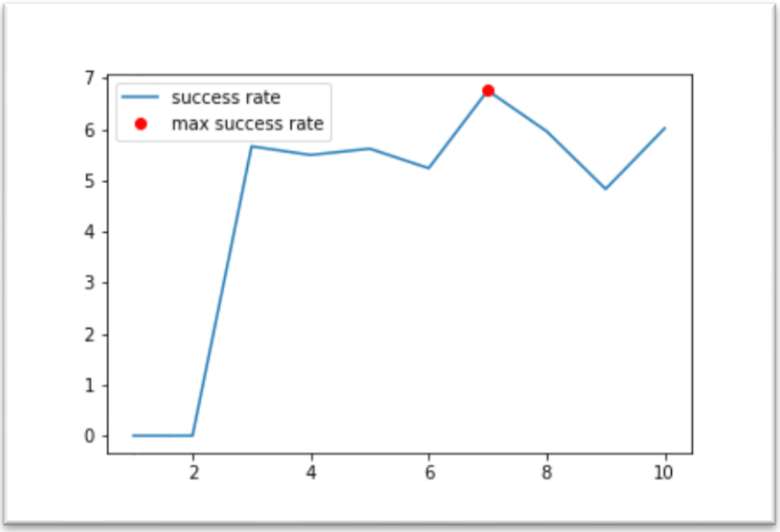


Figura 5: Tasso di successo sulle ordinate, generazione sulle ascisse

8.4 SARSA GENETICO

In questo caso, abbiamo utilizzato lo stesso approccio usato per il Q-Learning genetico, solo che ogni agente effettua 5.000 episodi di SARSA base.

$num\_training\_episodes = 5000$
$num\_test\_episodes = 1000$
$population\_size = 10$
$num\_generation = 10$
$max\_steps = 1000$
$learning\_rate = 0.1$
$discount\_rate\ (gamma) = 0.99$
$exploration\_rate\ (epsilon) = 1.0$
$max\_exploration\_rate\ (max\_epsilon) = 1.0$
$min\_exploration\_rate\ (min\_epsilon) = 0.01$
$exploration\_decay\_rate = 0.001$

I risultati in questo caso oscillano molto più del caso precedente, ma si registra un picco intorno al 65% di tasso di successo.

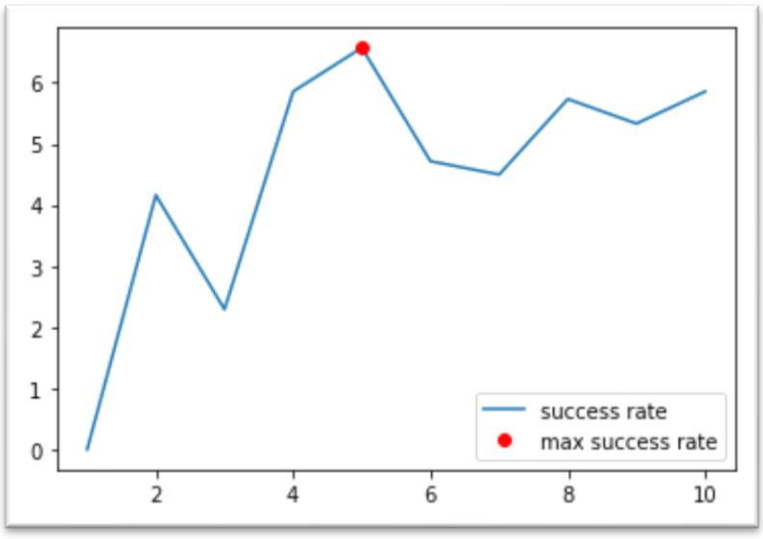


Figura 6: Tasso di successo sulle ordinate, generazione sulle ascisse