Assignment 3

Introduzione al problema

- Guidare una macchina da corsa per fare una curva
- L’obiettivo è andare più veloci possibili senza uscire dal tracciato
- L'auto si trova in una delle posizioni discrete della griglia.
- La velocità è discreta: numero di celle della griglia spostate orizzontalmente e verticalmente per ogni intervallo di tempo.
- Le azioni sono incrementi delle componenti di velocità.
- Le velocità sono limitate a un massimo di 5.
- Gli episodi iniziano in modo casuale sulla linea di partenza.
- Gli episodi terminano se si taglia il traguardo.
- Le ricompense sono -1 per ogni intervallo.
- Se si raggiunge il limite, si torna casualmente sulla linea di partenza.

Creazione del tracciato

- Viene identificato tramite una matrice t.c. fuori=-1, start=1, tracciato= 0, fine=2
- Alla funzione si passa le dimensioni del tracciato
- La linea di partenza è composta di 5 celle
- Per ogni riga successiva, tramite vettori di probabilità, si decide se il tracciato si restringe, allarga o rimane lo stesso
- Vincolo di lasciare il percorso di almeno 4 celle
- Il traguardo è nelle ultime 5 righe a destra

Creazione del tracciato

```
def create_track(N1,N2):  3 usages  gronz
    track = np.zeros( shape: (N1,N2),dtype=int) #fuori =-1, start=
    s1=randint( a: 0,N2-10)
    s2=randint(s1+5,N2)
    for i in range(N2):  #crea linea di start
        if i in range(s1,s2):
            track[0,i]=1
        else:
            track[0,i]=-1
    y=s1
    for i in range(1,N1):   #lato sinistro del percorso
        x = np.random.choice( a: [-1, 0, 1], p=[0.3, 0.3, 0.4]) #c
        y = y + x if 0 <= y + x <= N2 - 5  else y
        for j in range(y):
            track[i,j]=-1
    y=s2
    for i in range(1,N1-5): #lato destro lascio le ultime 5 che l
        x = np.random.choice( a: [-1, 0, 1], p=[0.4, 0.3, 0.3]) #c
        if 0 <= y + x < N2 and not track[i,y-5]==-1:  ##lascia se
            y = y + x
        if track[i,y-5]==-1:
            y+=1
        for j in range(y,N2):
            track[i,j]=-1
        #if track[i,y-2]==-1:
         #   track[i,y]=track[i,y-1]=0
    for i in range(1,6): #fine
        track[N1-i,N2-1]=2
    for riga in track:
        print(riga)
    return track
```

On-policy every-visit Monte Carlo control

- Si inizializza la policy casualmente
- Si utilizza una policy ε-greedy, in cui l'epsilon decade a ogni episodio fino a un valore minimo
- Si genera l'episodio
- Si calcolano i ritorni

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G - Q(S_t, A_t))$$

$$A^* \leftarrow \arg\max_a Q(S_t, a)$$

for all $a \in \mathcal{A}(S_t)$ **do**

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(S_t)|}, & \text{if } a = A^*, \\ \frac{\varepsilon}{|\mathcal{A}(S_t)|}, & \text{otherwise} \end{cases}$$

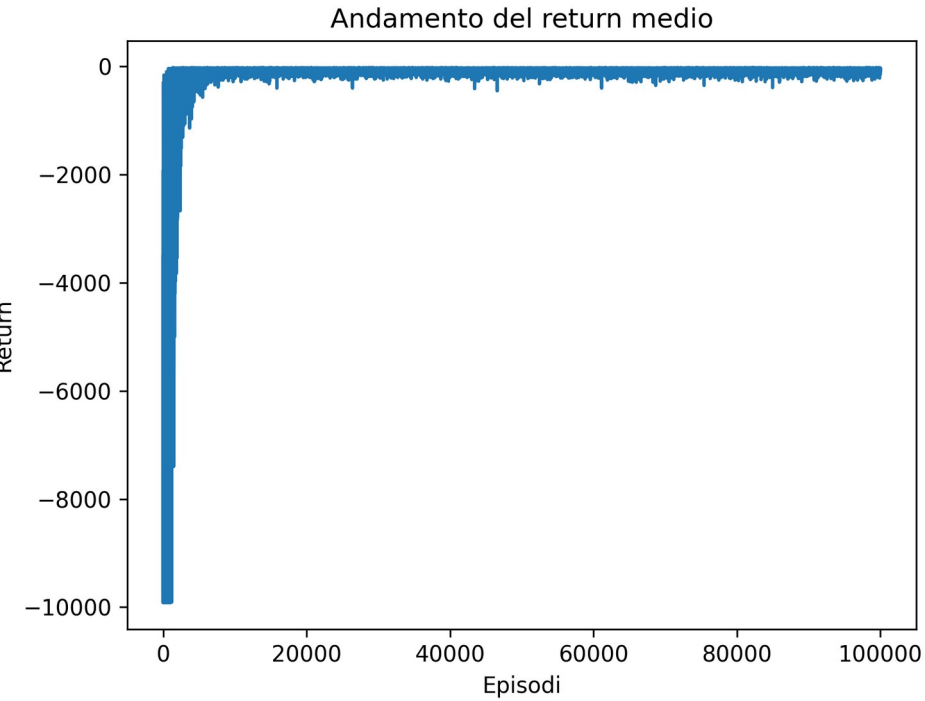
On-policy every-visit Monte Carlo control

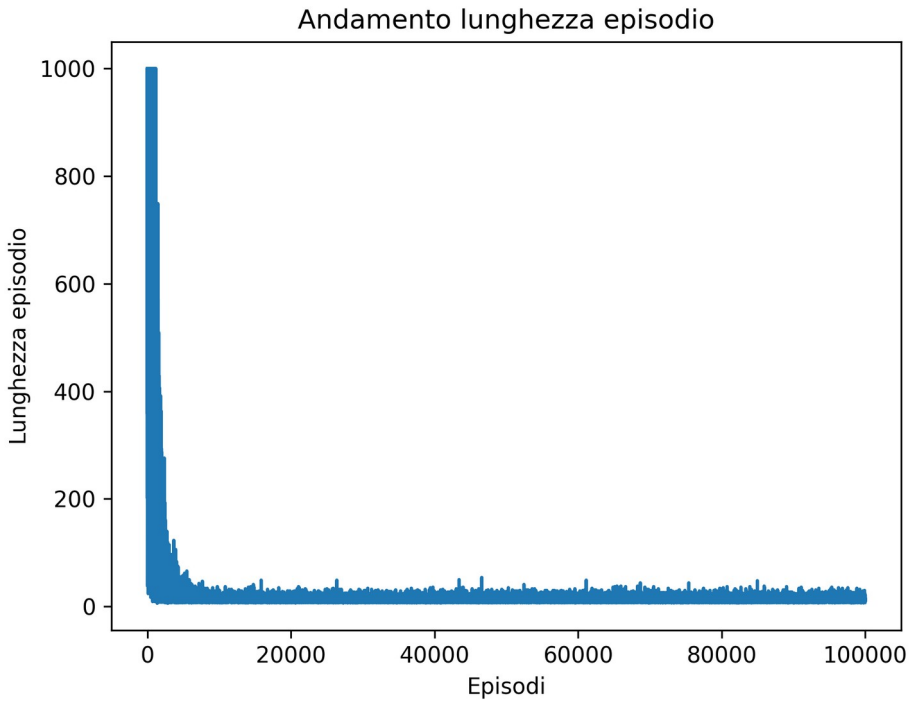
```python
def MC(track):  1 usage  gronz *
    Q = np.zeros((N1,N2,A))   #(N1,N2) griglia, azioni
    N = np.zeros((N1,N2,A))
    pi = np.zeros( shape: (N1, N2), dtype=int)
    actions = [(ao, av) for ao in [-1, 0, 1] for av in [-1, 0, 1]]
    for i in range(N1): #random policy
        for j in range(N2):
            if not track[i,j]==-1:  #non ha senso inizializzare la policy fuori dal tracciato
                pi[i,j] = randint( a: 0,A-1)
    # monitoraggio
    avg_returns = []
    avg_lengths = []
    eps = 1
    eps_end = 0.01
    decay_rate = 0.9995
    for el in range(100_000):#numero episodi
        episode = generate_episode(track,pi,eps)
        G=0
        total_return = 0
        for t, (stato, azione , reward) in enumerate(reversed(episode)):
            s1, s2 = stato
            a_idx = actions.index(azione)
            G = 0.9 * G + reward  # ritorno con gamma=0.9
            total_return += G
            N[s1, s2, a_idx] += 1
            alpha = 1 / N[s1, s2, a_idx]
            Q[s1, s2, a_idx] += alpha * (G - Q[s1, s2, a_idx])
            best_a = np.argmax(Q[s1,s2,:])
            pi[s1, s2] = best_a
        avg_returns.append(total_return)
        avg_lengths.append(len(episode))
        eps = max(eps_end, eps * decay_rate)
        if (el+1) % 1000 == 0:
            print(f"[Episodio {el+1}] "
                  f"Lunghezza media: {np.mean(avg_lengths[-50:]):.1f}, "
                  f"Return medio: {np.mean(avg_returns[-50:]):.2f}")
    return pi, Q, avg_returns, avg_lengths
```

Generazione episodio

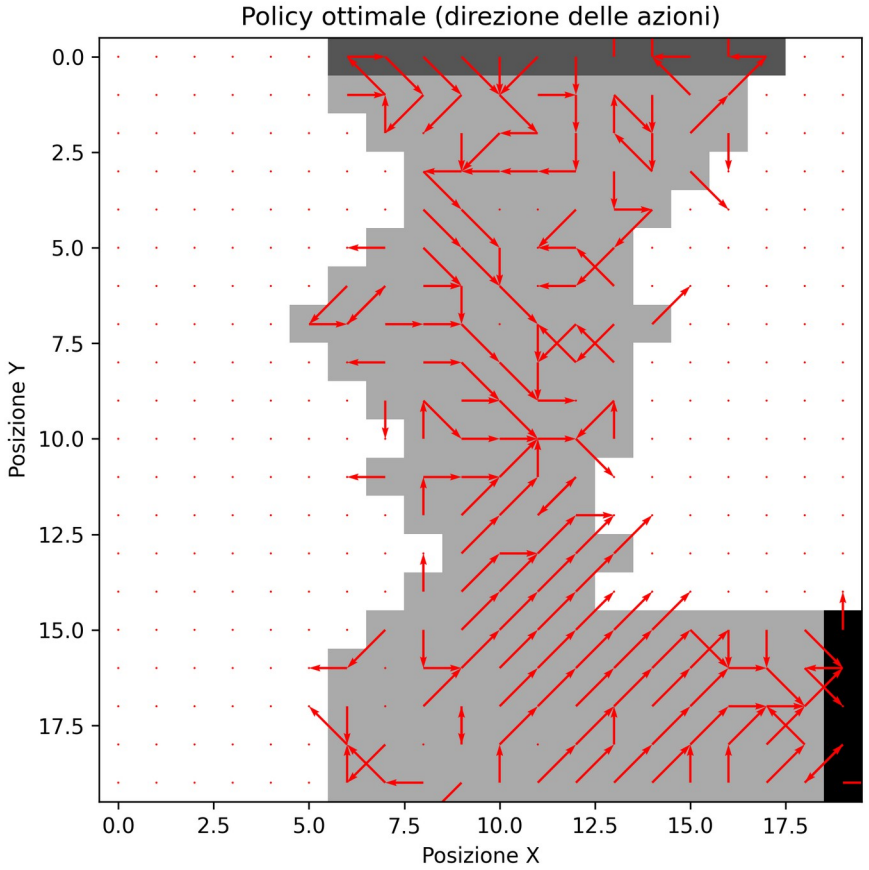
- Si parte dalla linea di partenza con velocità 0
- Limite numero di step poiché le prime policy casuali potrebbero non trovare soluzione
- Si sceglie l’azione secondo la policy ε-greedy
- Si calcola la nuova velocità limitandola a 5
- La macchina si muove prima verticalmente e poi orizzontalmente
- Se esce fuori riparte dalla linea di partenza con velocità 0
- 2 tipologie di reward

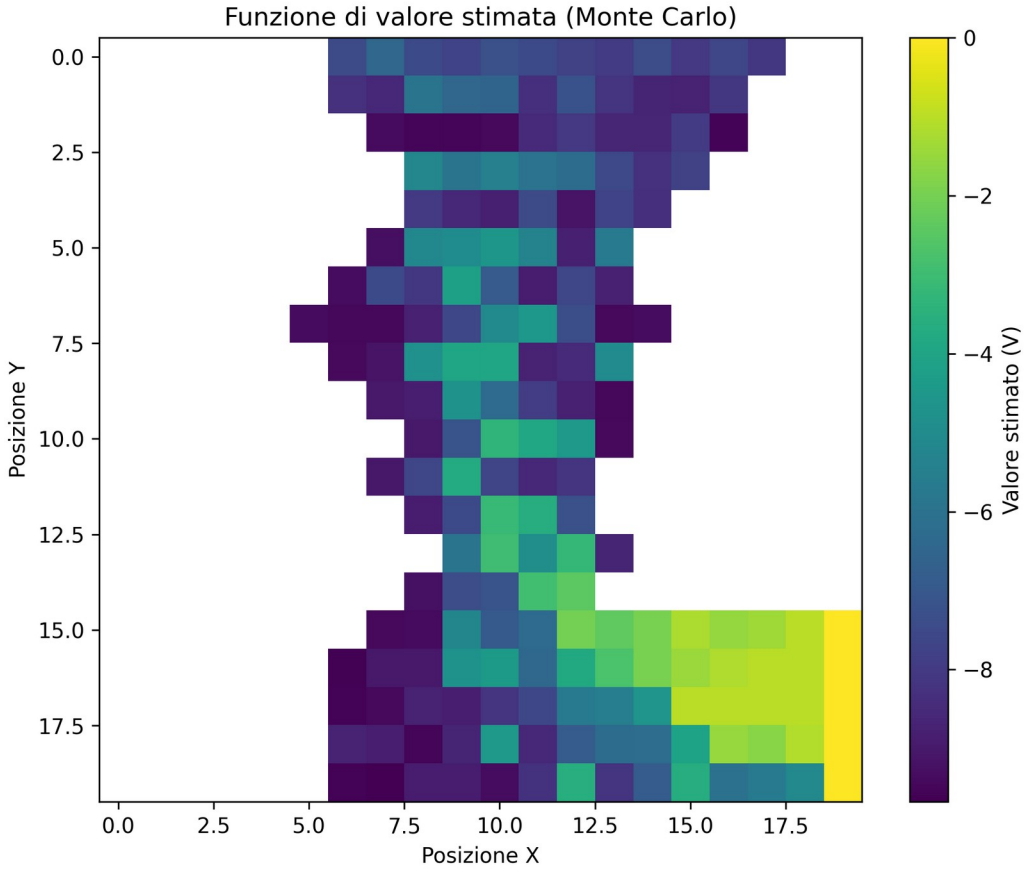
Andamento training



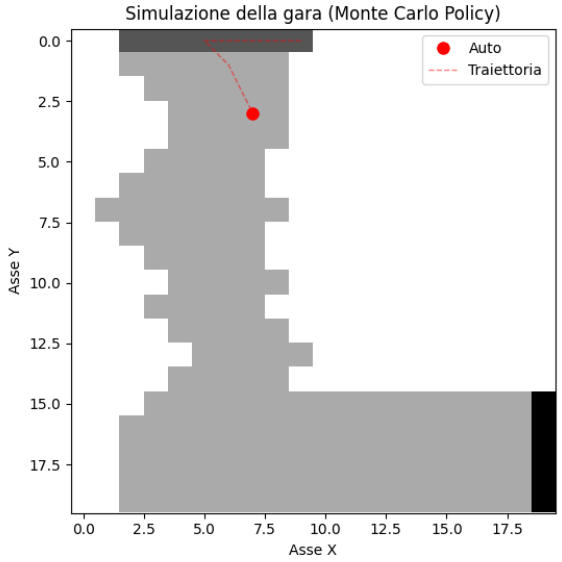


Soluzione


Policy ottimale (direzione delle azioni)
Posizione Y
Posizione X
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Funzione di valore stimata (Monte Carlo)
Posizione Y
Posizione X
Valore stimato (V)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
−2
−4
−6
−8

Corsa


Simulazione della gara (Monte Carlo Policy)
Auto
Traiettoria
Asse Y
Asse X
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

FINE