MONTE CARLO CONTROL

ASSIGNMENT 3

LUCA SUGAMOSTO , MATRICOLA 0324613

MATTIA QUADRINI , MATRICOLA 0334381

PROF. CORRADO POSSIERI

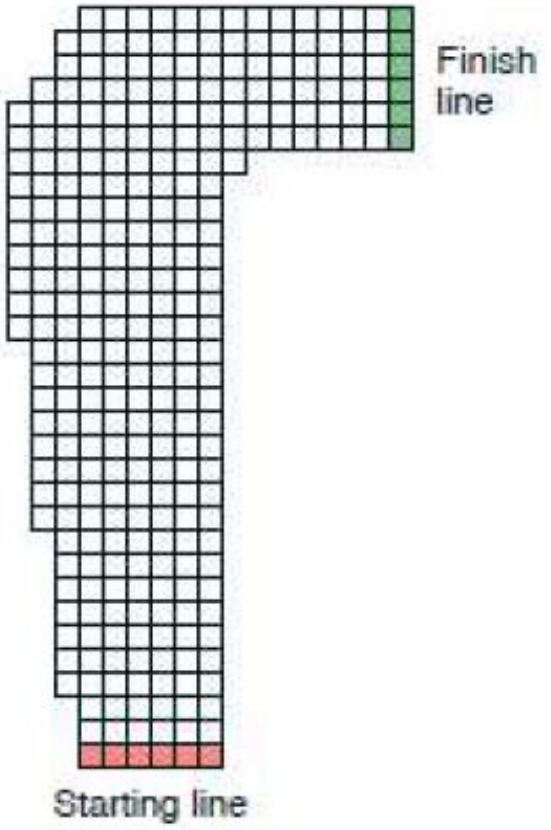
OBIETTIVO DEL PROGETTO

Si consideri di guidare un'auto lungo un percorso prefissato. L'auto si trova in una serie discreta di posizioni del percorso. Anche la velocità è discreta, cioè un numero di celle sono percorse orizzontalmente e verticalmente ad ogni passo temporale.

Le azioni permettono di variare le componenti di velocità, ognuna delle due può essere variata di $-1, 0, 1$ in ogni passo.
Entrambe le componenti di velocità sono ristrette ad essere non negative, minori di 5 e non possono essere entrambe nulle, eccetto sulla linea di partenza.

Ogni episodio inizia in uno degli stati iniziali selezionato casualmente, con entrambe le componenti di velocità nulle e finisce quando l'auto attraversa uno degli stati terminali.
Se l'auto colpisce il bordo della pista, viene riportata nella linea di partenza e viene azzerata la velocità.

Il reward è -1 per ogni passo temporale trascorso finché l'auto non attraversa la linea di arrivo.



CONSIDERAZIONI PRELIMINARI

- Il metodo MONTE CARLO è **model - free**, cioè non c'è bisogno di alcuna conoscenza riguardo l'ambiente del processo decisionale Markoviano, quindi non è necessario conoscere le probabilità di transizione di stato $p(s', r\,|s, a)$. Esso richiede solamente l'**esperienza** passata per poter apprendere.
  Inoltre, tale metodo è definito solo per task episodici.

- Data la policy π e dato un insieme di episodi ottenuti seguendo tale policy, si può passare una o più volte attraverso uno stato s. Ogni occorrenza dello stato s nell'episodio è chiamato **visita** di s.
  Nell'implementazione dell'algoritmo si è utilizzato il metodo : **Every - Visit MC method**, quindi per la stima di $v_\pi(s)$ *o* $q_\pi(s)$ si considerano tutte le visite (occorrenze) dello stato s nello stesso episodio.

- Molte coppie stato - azione possono non essere visitate; per mantenere un certo livello di esplorazione, gli episodi iniziano da uno stato che viene scelto casualmente dall'insieme degli stati di partenza.
  Questo metodo di selezione iniziale prende il nome di **exploring starts**.

ALGORITMI POSSIBILI PER IL GIOCATORE

- Monte Carlo Exploring Start Algorithm

Every - Visit MC Control with Exploring start

Function RACETRACK_DYNAMICS

Questa funzione permette la creazione della matrice $track$ rappresentante la mappa del tracciato, differenziando tra pixel appartenenti alla pista e quelli al di fuori.

Inoltre, si definiscono l'insieme degli stati iniziali e quello degli stati finali del tracciato per mezzo di due vettori, rispettivamente $initialState$ e $finalState$.

Tutte queste variabili sono salvate in un file $.mat$, il quale viene chiamato dalle altre funzioni.

```
%Numero di stati del processo
S = 17*32;        %17 colonne e 32 righe

%Inizializzazione della matrice che rappresenta la pista da percorrere
track = zeros(32,17);

%Le caselle con valore 1 rappresentano quelle percorribili mentre quelle
%con valore 0 rappresentano zone non percorribili.
%Usando i seguenti indici si considera lo stato i-esimo e non le coordinate
%dello stato i-esimo
for s = 5:14
    track(s) = 1;
end
for s = 36:54
    track(s) = 1;
end
for s = 66:93
    track(s) = 1;
end
for s = 97:295
    track(s) = 1;
end
%Usando i seguenti indici si considerano le coordinate dello stato i-esimo
for j = 11:17
    for i = 1:6
        track(i,j) = 1;
    end
end

%Inizializzazione degli stati di partenza e degli stati di arrivo
initialState = [128; 160; 192; 224; 256; 288];
finalState = [513; 514; 515; 516; 517; 518];

save raceTrack.mat  S track initialState finalState
```

$(Fig. 1)$

Function SPEEDCONTROL

La funzione in figura $(Fig.2)$ riceve in ingresso le componenti di velocità $currentV$ (rispettivamente lungo x, y) assunte fino ad ora dall'auto e le probabili nuove componenti di velocità $newV$, calcolate come segue : $V_{new} = V_{current} + A_{new}$.

Successivamente si effettuano dei controlli sulle componenti del vettore V_{new} per vedere se sono rispettate le condizioni di minimo e/o di massimo valore assumibile.

Infine viene restituito il vettore delle nuove velocità da applicare all'auto nel passo temporale corrente.

```
function finalV = speedControl(currentV, newV)
    %funzione che riceve in ingresso le componenti di velocità lungo x ed y
    %e controlla se sono rispettate le condizioni seguenti:
    % - velocità non negative
    % - velocità minore di 5
    % - al massimo una componente di velocità nulla

    %Controllo del valore minimo ammissibile per la velocità
    if (newV(1) < 0 && newV(2) > 0)         %Se componente lungo x è negativa
        newV(1) = 0;
    elseif (newV(1) > 0 && newV(2) < 0)     %Se componente lungo y è negativa
        newV(2) = 0;
    elseif (newV(1) <= 0 && newV(2) <= 0)   %Se entrambe le componenti sono negative/nulle
        newV = currentV;                    %La nuova velocità rimane uguale alla precedente
    end

    %Controllo del valore massimo ammissibile per la velocità
    if (newV(1) > 4)                        %Se componente lungo x è maggiore o uguale a 5
        newV(1) = 4;
    elseif (newV(2) > 4)                    %Se componente lunngo y è maggiore o uguale a 5
        newV(2) = 4;
    end

    %Velocità finale dopo i controlli sulle singole componenti
    finalV = newV;
end
```

$(Fig.2)$

Function UPDATESTATE

La seguente funzione $(Fig.3)$ riceve in ingresso:

- Lo stato dell'auto all'istante di tempo $t-1$;
- La velocità con cui si muove l'auto all'istante di tempo t;
- Le informazioni sul tracciato da percorrere.

Con queste informazioni sono calcolate le nuove coordinate dell'auto dopo lo spostamento. Valutando quest'ultime si può ottenere l'uscita della funzione :

- Se le nuove coordinate sono al di fuori delle dimensioni della matrice o sono all'interno della matrice, ma al di fuori del tracciato → si riposiziona l'auto in uno stato iniziale, si attribuisce reward negativo e viene azzerata la velocità;
- Se le nuove coordinate sono all'interno del tracciato → l'auto vi si posiziona, si attribuisce reward negativo se il nuovo stato è di 'passaggio' mentre si ha reward nullo se è uno stato terminale. (La velocità deve essere diversa da zero).

```
function [nextState, reward, v] = updateState(state, v, track, initialState, finalState)
    %Funzione che riceve in ingresso:
        %Lo stato corrente in cui si trova l'auto (state);
        %Il vettore delle componenti di velocità lungo x ed y (v);
        %Le informazioni sul tracciato.
    %In uscita viene restituito il nuovo stato in cui si posiziona l'auto,
    %la ricompensa ottenuta passando al nuovo stato e la velocità dell'auto

    initialS = length(initialState);            %Numero degli stati di partenza
    finalS = length(finalState);                %Numero degli stati di arrivo

    %Ricavare gli indici di riga e di colonna dello stato corrente
    [numRow, numCol] = ind2sub([32 17], state);

    %Calcolo dello stato successivo e successivo controllo sul fatto che
    %questo sia o meno uno stato percorribile
    next_numRow = numRow - v(1);                %Il segno "-" è dovuto al fatto che l'indice di riga cresce verso il basso
    next_numCol = numCol + v(2);                %Il segno "+" è dovuto al fatto che l'indice di colonna cresce verso destra

    if (next_numRow < 1 || next_numCol < 1 || next_numRow >32 || next_numCol > 17)
        %Condizione in cui la transizione porta l'auto fuori dalla matrice,
        %quindi non sono operazioni possibili

        %Riposizionamento casuale dell'auto in uno degli stati di partenza,
        %assegnazione negativa della ricompensa e reset della velocità
        nextState = initialState(randi(initialS));
        % nextState = initialState(1);
        reward = -1;
        v = [0,0];
    else
        %L'auto si sposta all'interno della matrice degli stati
        nextState = sub2ind([32 17], next_numRow, next_numCol);

        %Controllare che il nuovo stato in cui si trova l'auto faccia parte
        %della pista (valore associato è 1) oppure si trovi fuori dalla
        %pista (valore associato è 0)
        if (track(nextState) == 0)              %Auto fuori dalla pista
            nextState = initialState(randi(initialS));
            % nextState = initialState(1);
            reward = -1;
            v = [0,0];
        else                                    %Auto all'interno della pista
            %Controllare se il nuovo stato sia uno stato finale oppure
            %transitorio
            if (nextState >= finalState(1) && nextState <= finalState(finalS))
                %Si è raggiunta la linea di arrivo
                reward = 0;
                v = [0,0];
            else
                %Si trasla su uno stato transitorio
                reward = -1;
            end
        end
    end
```

$(Fig.3)$

Function CONTROLMC (Fase di inizializzazione)

Nella fase di inizializzazione si eseguono le seguenti procedure:

- Caricamento del tracciato;
- Inizializzazione del numero di episodi;
- Inizializzazione dei parametri fondamentali $\alpha, \gamma, \varepsilon$;
- Inizializzazione della lista con tutte le azioni possibili;
- Inizializzazione della stima della funzione qualità, del contatore delle coppie stato - azione e di una policy generata casualmente;
- Modifica della policy per gli stati terminali affinché l'azione presa in essi sia nulla.

```
load raceTrack.mat                      %Caricamento delle variabili  necessarie all'algoritmo

numEpisodes = 5*(10^5);                 %Numeri di episodi totali
finalS = length(finalState);            %Numero di stati terminali totali
initialS = length(initialState);        %Numeto di stati iniziali totali

%Parametro che garantisce l'esplorazione delle azioni in ogni iterazione.
%All'aumentare delle iterazioni tale parametro deve decrescere per
%garantire che sia usata la policy ottimale e sempre meno azioni vengano
%prese casualmente
epsilon = 0.8;
decrease = 0.00015;                     %Valore sottratto ad epsilon

gamma = 0.9;                            %Fattore di scarto (indica a quale reward dare maggiore importanza)
alpha = 0.1;                            %Passo di aggiornamento costante

%Inizializzazione della matrice che contiene tutte le azione possibili
actionList = [[0,1];[1,0];[1,1];[0,0];[-1,0];[0,-1];[-1,-1];[-1,1];[1,-1]];

A = length(actionList);                 %Numero di azioni totali

%FASE DI INIZIALIZZAZIONE DELL'ALGORITMO
Q = zeros(S,A);                         %Stima della funzione qualità
N = zeros(S,A);                         %Occorrenze per ogni coppia stato-azione
pi = randi(A,[S 1]);                    %Policy iniziale randomica

%L'azione associata agli stati terminali è quella di non fare nulla quindi
%sia la componente x sia la componente y deve essere nulla
for i = 1:finalS
    pi(finalState(i)) = 4;              %"4" è l'indice dell'azione [0, 0]
end

counters = [];                          %Contiene tutti i contatori associati ad ogni singolo episodio
bestPath = [];                          %Contiene tutti gli stati percorsi dall'auto usando la policy ottimale
bestActions = [];                       %Vettore delle azioni prese nel percorso migliore
valBestPath = inf;                      %Indica il numero di stati percorsi dall'auto usando la policy ottimale
```

$(Fig. 4)$

Function CONTROLMC (Fase di aggiornamento)

```
%FASE DI AGGIORNAMENTO DELL'ALGORITMO
for i = 1:numEpisodes
    fprintf("n° episodio esaminato:");
    disp(i);
    fprintf("Epsilon utilizzato:");
    disp(epsilon);
    %Scelta casuale dello stato da cui partire
    S0 = initialState(randi(initialS));          %Scelta casuale dello stato di partenza
    %Scelta fissa dello stato da cui partire (per analizzare se partendo da
    %ogni stato iniziale si arriva comunque a quello finale)
    % S0 = initialState(1);

    firstAction = 1;                             %Variabile posta ad 1 ogni volta che inizia un nuovo episodio
    v = [0, 0];                                  %Componenti della velocità inizialmente nulle ad ogni nuovo episodio

    %Inizializzazione dei vettori che tengono traccia degli stati, delle
    %azioni e dei rewards relativi all'episodio corrente
    states = S0;                                 %Contiene già il primo stato che si visita
    currentState = S0;                           %Stato in cui si trova l'auto
    actions = [];
    rewards = [];

    reward = -1;                                 %Valore della ricompensa istantanea
    counter = 1;                                 %Numero di iterazioni necessarie per arrivare nello stato terminale
```

Parte 1 dell'algoritmo :
Inizializzazione delle variabili utili nella fase di aggiornamento dei dati

```
%Generazione dell'episodio a partire dalla coppia (S0, A0)
while (reward == -1)
    if (firstAction == 1)
        %Scelta dell'azione di partenza A(0)
        a = randi(A);                            %Indice dell'azione da prendere
        action = actionList(a,:);                %Azione vera e propria da prendere

        %Si assegna alla variabile "firstAction" valore 0 così le
        %successive azioni sono prese seguendo la policy calcolata
        firstAction = 0;
    else
        %Scelta delle azioni A(t) successive (algoritmo Eps-Greedy)
        if (rand(1) < epsilon)
            %Azione scelta casualmente
            a = randi(A);                        %Indice dell'azione da prendere
            action = actionList(a,:);            %Azione vera e propria da prendere
        else
            %Azione scelta seguendo la policy
            a = pi(currentState);                %indice dell'azione da prendere
            action = actionList(a,:);            %Azione vera e propria da prendere
        end

    end

    %Controllo dei valori di velocità da passare in ingresso alla
    %funzione "updateState", questi devono essere: non negativi, minori
    %di 5 e non possono essere entrambi nulli (negli stati intermedi)
    newV = v + action;                           %Nuovo vettore delle velocità lungo x ed y

    %Velocità da passare in ingresso alla funzione "updateState"
    v = speedControl(v, newV);

    %Calcolo del nuovo stato in cui si colloca l'auto e della nuova
    %ricompensa ottenuta
    [nextState, reward, v] = updateState(currentState, v, track, initialState, finalState);

    %Aggiunta dei nuovi valori nelle liste associate agli stati,
    %azioni, rewards
    states = horzcat(states, nextState);
    rewards = horzcat(rewards, reward);
    actions = horzcat(actions, a);

    %Aggiornamento delle variabili per l'iterazione del loop successiva
    currentState = nextState;
    counter = counter + 1;
end
```

Parte 2 dell'algoritmo : Scelta delle azioni e aggiornamento degli stati attraversati

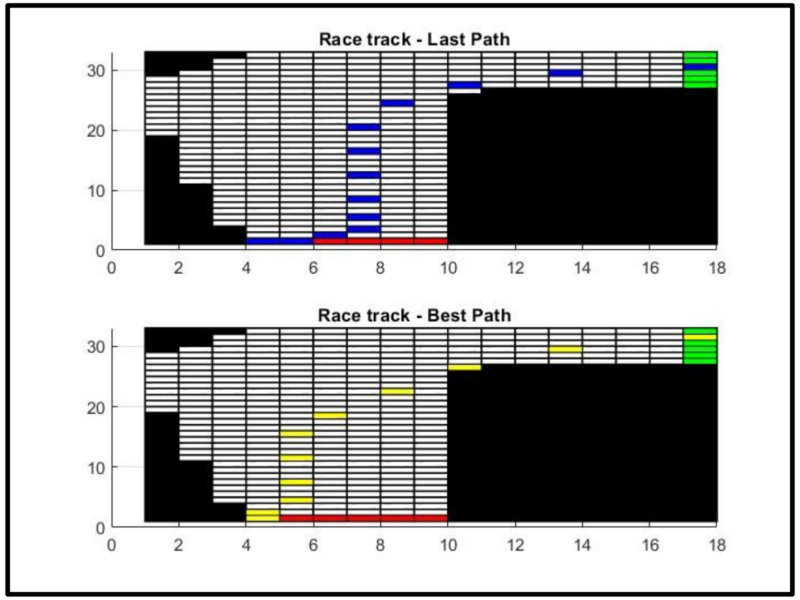
```
    %GENERAZIONE DELL'EPISODIO i-ESIMO TERMINATO

    %Aggiunta dei nuovi valori nella lista associata ai contatori delle
    %iterazioni per singolo episodio
    counters = horzcat(counters,counter);

    if (mod(i,100) == 0 && epsilon > 0.05)
        epsilon = epsilon - decrease;            %Aggiornamento del parametro Epsilon (minore esplorazione, maggiore sfruttamento)
    end

    %Se l'episodio generato in questa iterazione è migliore di tutti gli
    %altri generati fino ad ora allora lo salvo come "miglior percorso"
    if (states(end) >= finalState(1) && states(end) <= finalState(finalS) && counter < valBestPath)
        bestPath = states;                       %Aggiornamento del percorso migliore
        valBestPath = counter;                   %Aggiornamento del numero di stati del percorso migliore
        bestActions = actions;                   %Aggiornamento delle azioni prese nel percorso migliore
    end

    %AGGIORNAMENTO DELLA POLICY
    %Una volta generato l'episodio si va a calcolare a retroso il ritorno
    %atteso G, il quale permette di calcolare la funzione qualità e
    %successivamente migliorare la policy
    G = 0;                                       %Ritorno atteso
    for t = length(rewards):-1:1                 %Scorrimento dell'indice all'indietro
        G = rewards(t) + (gamma * G);
        %Il passo di aggiornamento nella seguente formula è COSTANTE
        Q(states(t), actions(t)) = Q(states(t), actions(t)) + (alpha * (G - Q(states(t), actions(t))));

        %Il passo di aggiornamento nella seguente formula è VARIABILE
        %N(states(t), actions(t)) = N(states(t), actions(t)) + 1;
        %Q(states(t), actions(t)) = Q(states(t), actions(t)) + (1 / N(states(t), actions(t))) * (G - Q(states(t), actions(t)))

        %La policy viene aggiornata ad ogni loop
        pi(states(t)) = find(Q(states(t), :) == max(Q(states(t), :)), 1, "first");
    end
end
```

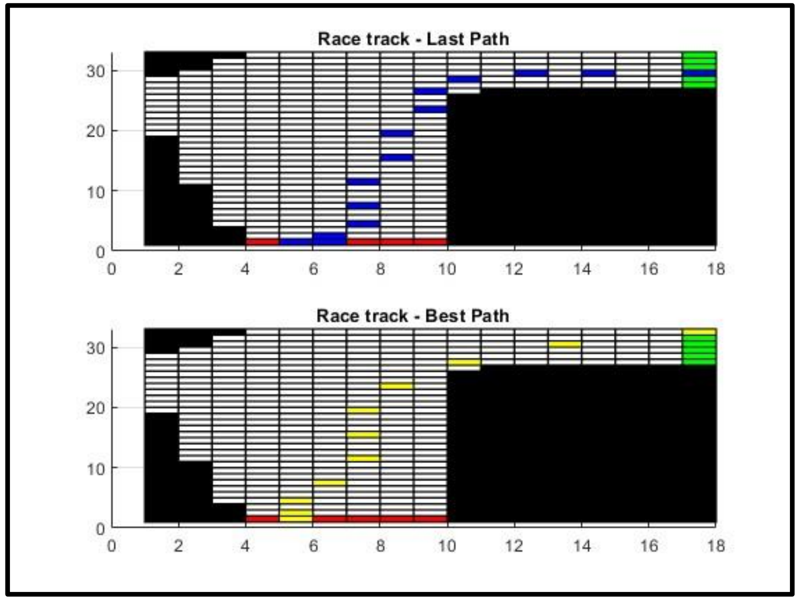
Parte 3 dell'algoritmo :
Terminazione dell'episodio i-esimo e generazione di una nuova policy migliore

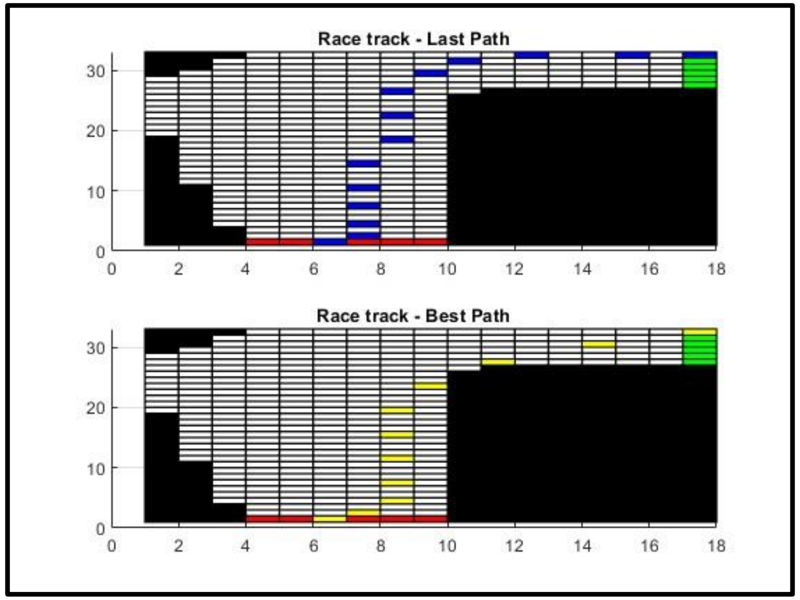
ANALISI DEI RISULTATI OTTENUTI

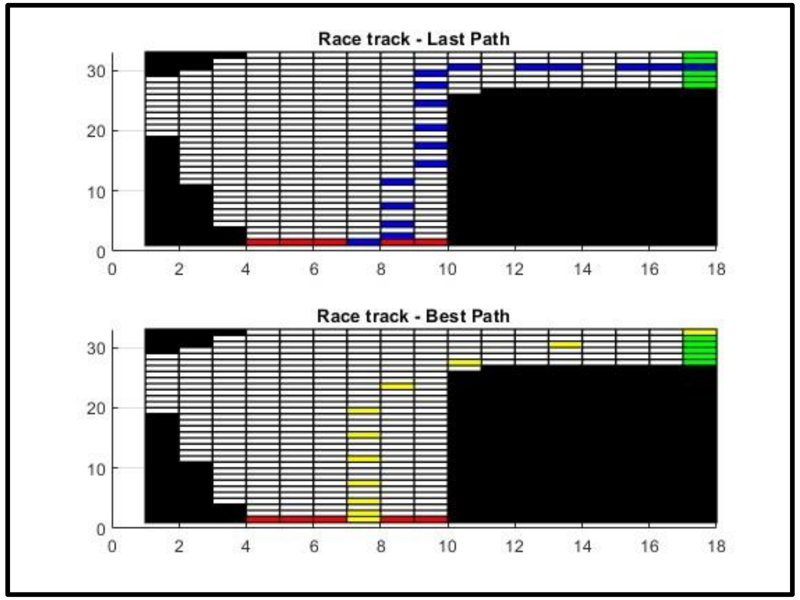
Sono riportate le immagini del percorso effettuato dall'auto a partire da ognuno degli stati iniziali, con i seguenti parametri:

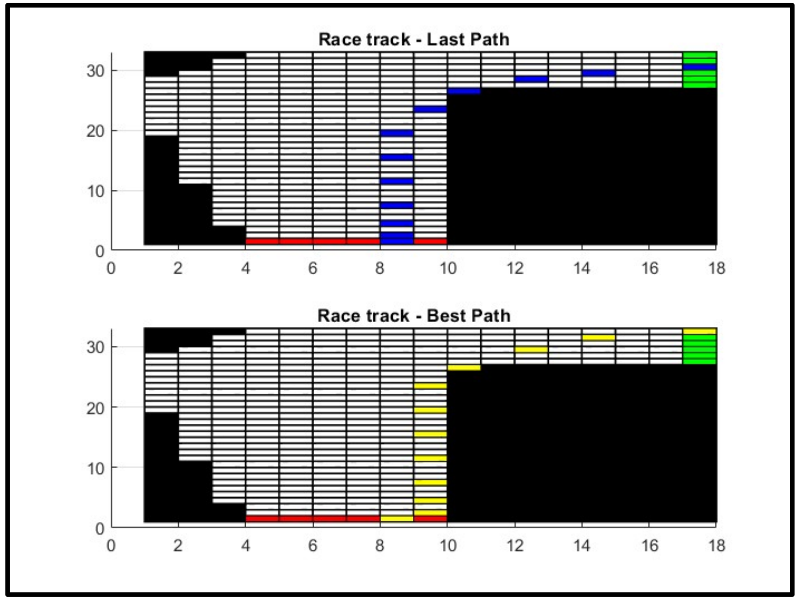
$\alpha = 0.1, \ \ \gamma = 0.9, \ \ \varepsilon = 0.8 \ \rightarrow 0.05 \ (decrescente \ nel \ tempo)$

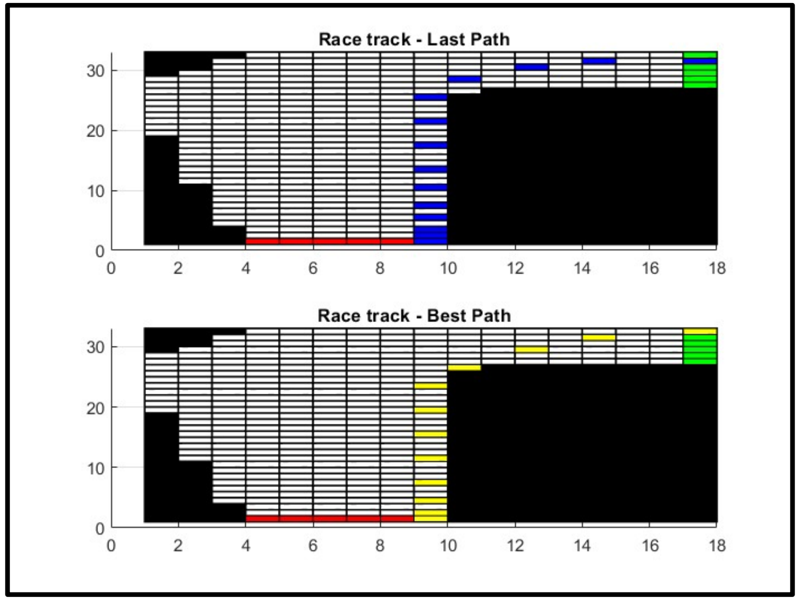












CASO DI STUDIO ASSOCIATO AD UN SINGOLO PUNTO DI PARTENZA

Si è considerato di partire dallo stato S_o più a sinistra della linea di partenza ($S_0 = 128$).

Si osserva come in tutti i casi esaminati si ottiene lo stesso reward per quanto riguarda il percorso migliore. Questo corrisponde al percorrere $|R_{BP}|$ stati per arrivare a destinazione (un qualsiasi stato terminale).

Inoltre, si può notare che all'aumentare del parametro $\gamma \in [0,1]$ diminuisce il tempo necessario per analizzare tutti gli episodi (numero di episodi totali $= 5 * 10^5$)

Si è usato lo stesso seme $rng(1)$ così da considerare la stessa policy casuale all'inizio di ogni test.

α	γ	ε	R_{LP}	R_{BP}	Elapsed Time [sec]	rng
0,001	0,1	0,8 → → 0,05	-14	-10	$2{,}68 * 10^3$	1
0,001	0,5	0,8 → → 0,05	-10	-10	$1{,}98 * 10^3$	1
0,001	0,9	0,8 → → 0,05	-10	-10	$1{,}81 * 10^3$	1
0,01	0,1	0,8 → → 0,05	-15	-10	$1{,}98 * 10^3$	1
0,01	0,5	0,8 → → 0,05	-11	-10	$1{,}75 * 10^3$	1
0,01	0,9	0,8 → → 0,05	-10	-10	$1{,}18 * 10^3$	1
0,1	0,1	0,8 → → 0,05	-10	-10	$1{,}54 * 10^3$	1
0,1	0,5	0,8 → → 0,05	-11	-10	$1{,}49 * 10^3$	1
0,1	0,9	0,8 → → 0,05	-12	-10	$1{,}66 * 10^3$	1

GRAZIE

LUCA SUGAMOSTO (0324613)

MATTIA QUADRINI (0334381)

luca.sugamosto@students.uniroma2.eu

mattia.quadrini.1509@students.uniroma2.eu