MULTI ARM-BANDIT

ASSIGNMENT 1

LUCA SUGAMOSTO , MATRICOLA 0324613

MATTIA QUADRINI , MATRICOLA 0334381

PROF. CORRADO POSSIERI

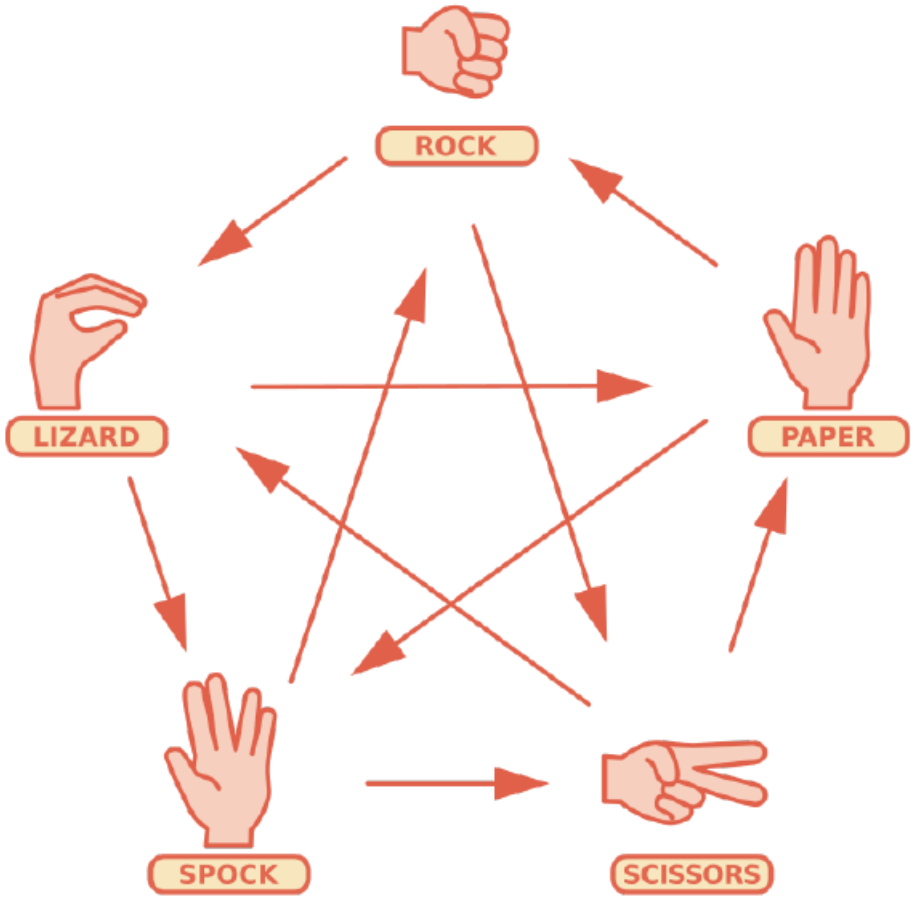
OBIETTIVO DEL PROGETTO

Dato il modello di gioco $Sasso, carta, forbici, Lizard, Spock$, ognuno dei due giocatori deve scegliere una tra le azioni possibili e cercare di ottenere la ricompensa massima; quest'ultima è assegnata ad ogni interazione del gioco nel seguente modo:

- $+1$ in caso di vittoria;
- -1 in caso di sconfitta;
- 0 in caso di pareggio.

Successivamente:

- Analizzare il trend delle ricompense ottenute per ogni azione tenendo conto delle varie politiche usate (per il giocatore 1).
- Comparare i risultati ottenuti tenendo conto dei parametri fissati.



In tutti i casi di studio considerati, il giocatore avversario gioca con distribuzione stocastica stazionaria.

CALCOLO DEL REWARD AD OGNI ITERAZIONE

- La funzione $valutaVincitore$ riceve in ingresso l'azione del giocatore 1 (A_t) e quella casuale dell'avversario (A_{Rand});
  a seconda di queste viene restituito il reward per il giocatore 1.

```
function Rt = valutaVincitore(At,ARand)
    %Funzione che restituisce il reward ottenuto all'istante t dal
    %giocatore 1 valutando l'azione presa da ogni giocatore
    % 1 = Sasso, 2 = Carta, 3 = Forbici, 4 = Spock, 5 = Lizard

    if (At == ARand)
        %Se entrambi scelgono la stessa azione allora il reward ottenuto
        %all'istante t dal giocatore 1 è 0
        Rt = 0;
    elseif (At ~= ARand)
        %Se i due giocatori scelgono azioni diverse tra loro allora si
        %hanno le seguenti conbinazioni ed il reward per il giocatore 1
        %sarà 1 se vince, -1 altrimenti
        if ((At == 1) && (ARand == 5))
            Rt = 1;
        elseif ((At == 5) && (ARand == 4))
            Rt = 1;
        elseif ((At == 4) && (ARand == 3))
            Rt = 1;
        elseif ((At == 3) && (ARand == 2))
            Rt = 1;
        elseif ((At == 2) && (ARand == 1))
            Rt = 1;
        elseif ((At == 1) && (ARand == 3))
            Rt = 1;
        elseif ((At == 2) && (ARand == 4))
            Rt = 1;
        elseif ((At == 3) && (ARand == 5))
            Rt = 1;
        elseif ((At == 4) && (ARand == 1))
            Rt = 1;
        elseif ((At == 5) && (ARand == 2))
            Rt = 1;
        else
            %Tutti i restanti casi provocano la sconfitta del giocatore 1
            Rt = -1;
        end
    end
```

ALGORITMI POSSIBILI PER IL GIOCATORE 1

- ε - greedy
- Upper confidence bound
- Preference updates

POLICY ε - Greedy

CARATTERISTICHE DEL *ε - greedy sample average*

Si definisce $greedy$ quell'azione che massimizza il valore stimato $Q(a)$ rispetto a tutte le altre azioni.
Quest'azione sfrutta le conoscenze del sistema e permette di ottenere ricompense maggiori in tempi brevi.

Contrariamente, l'azione $non - greedy$ può restituire una ricompensa più bassa ma permette di esplorare le altre azioni possibili.
Queste azioni possono produrre una ricompensa maggiore in tempi lunghi.

⇒ Il metodo $\boldsymbol{\varepsilon - greedy}$ sceglie ad ogni istante di tempo l'azione greedy con probabilità $1 - \varepsilon$ (**EXPLOITING**) mentre sceglie un'azione a caso (dall'insieme delle azioni $\mathcal{A}$) con probabilità ε (**EXPLORING**).

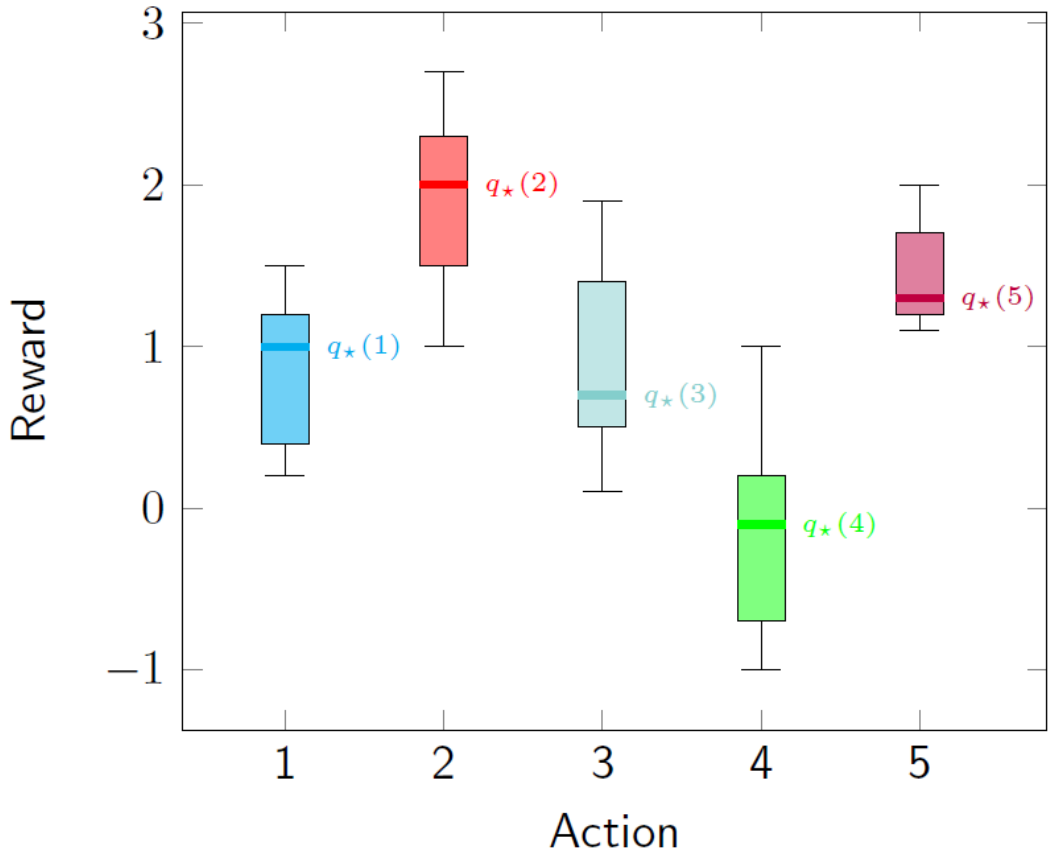
Ad ogni istante di tempo t si calcola la stima : $Q_t(a) \quad \forall a \in \mathcal{A}$, considerando la media delle ricompense ottenute fino ad ora (**sample-average**) : $Q_t(a) = \frac{R_1 + \ldots + R_{N_t}(a)}{N_t(a)}, \quad se\ N_t(a) > 0.$

Per salvare meno dati in memoria, $Q_t(a)$ può essere definita come: $Q_t(a) = Q_{t-1}(a) + \frac{1}{N_t(a)} * \left(R_t - Q_{t-1}(a)\right).$

OSSERVAZIONI PRELIMINARI SU *ε - greedy*

- Se la **varianza** del reward è **larga** (distribuzione di probabilità non centrata nell'intorno del valore atteso) allora è richiesta più esplorazione e quindi un valore più elevato di ε.

- Se la **varianza** del reward è **piccola** (distribuzione di probabilità centrata nell'intorno del valore atteso) allora i metodi greedy conoscono i valori reali di ogni azione e quindi l'azione greedy sarà la migliore da scegliere. In questo caso è preferibile un valore più piccolo di ε.



IMPLEMENTAZIONE DI *ε - greedy sample average*

```
fprintf("Ordine delle azioni nell'insieme A: ");
disp('1: Sasso, 2: Carta, 3: Forbici, 4: Spock, 5: Lizard');

%La seguente funzione è utile per la generazioni di numeri randomici;
%ad ogni valore numerico di input è associata una certa sequenza di numeri
%casuali, quindi per cambiare i valori si varia l'input
rng(2);

% 1 = Sasso, 2 = Carta, 3 = Forbici, 4 = Spock, 5 = Lizard
A = 5;                      %Numero di azioni possibili

%Probabilità (epsilon) con cui si sceglie se prendere l'azione greedy
%(azione che fa ottenere migliore reward) o un'azione casuale in A.
epsilon = 0.70;             %Valore di probabilità

%Inizializzo il vettore delle stime del valore dell'azione che viene poi
%aggiornato effettuando una media aritmetica dei reward
Q = zeros(A,1);
%Inizializzo il vettore che tiene traccia per ogni azione del numero di
%volte che questa viene presa
N = zeros(A,1);

%Definisco un numero di azioni sufficientemente grande così da garantire
%che per istanti di tempi grandi le stime dei valori di azione siano uguale
%ai valori delle azioni q(a).
numIteration = 100000;      %Numero di iterazioni totali
counter = 0;                %Contatore delle iterazioni considerate

finalR = 0;                 %Valore del ritorno alla fine del loop
initialQ = Q;               %Salvo il valore di partenza della variabile Q
andamentoQ = zeros(A,numIteration);         %Vettore che mantiene traccia di Q nel tempo (usato per il plot)
```

Scelta delle azioni →

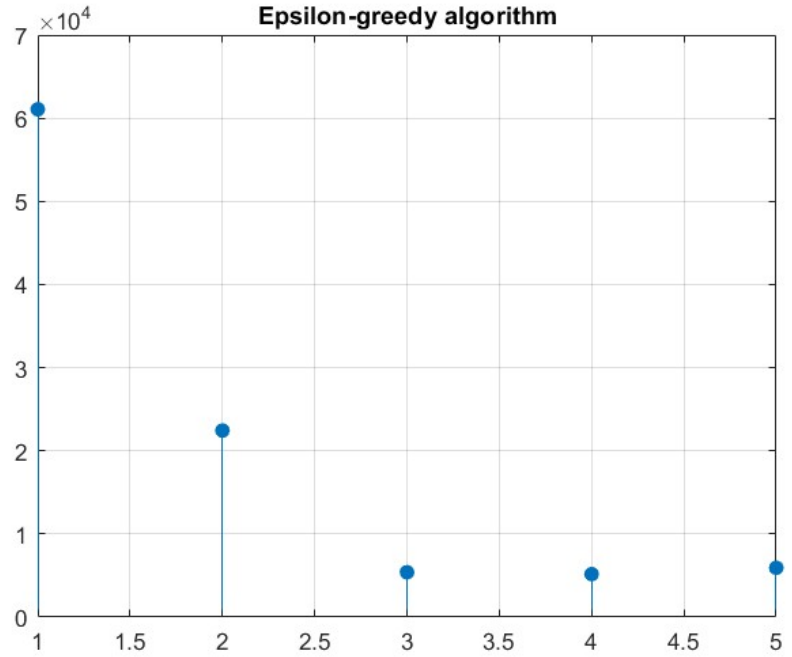
Calcolo del reward per il giocatore 1 →

Aggiornamento dei valori delle azioni →

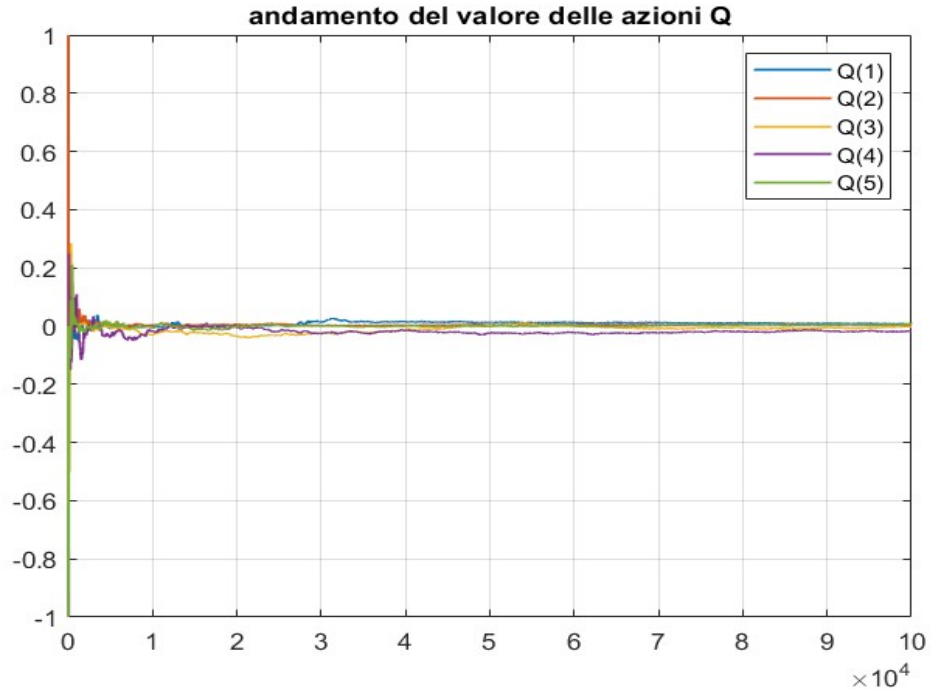
```
while (counter < numIteration)
    %Scelta dell'azione da prendere all'istante t dai due giocatori
    At = epsilonGreedy(Q,epsilon);          %Azione presa all'istante t dal giocatore 1
    ARand = randi(A);                       %Azione presa all'istante t dal giocatore 2

    %Calcolo della ricompensa per il giocatore 1
    Rt = valutaVincitore(At,ARand);         %Ricompensa all'istante t del giocatore 1
    finalR = finalR + Rt;                   %Aggiornamento della ricompensa totale

    %Incremento delle variabili N e Q relative solo all'azione considerata
    %dal giocatore 1, mentre il resto rimane invariato
    N(At,1) = N(At,1) + 1;
    Q(At,1) = Q(At,1) + ((1 / N(At,1)) * (Rt - Q(At,1)));

    counter = counter + 1;                  %Aggiornamento del contatore delle iterazioni
    for i = 1:A
        andamentoQ(i,counter) = Q(i,1);     %Inserimento dei nuovi dati nel vettore
    end
end

figure(1)
stem(N,'filled','-');
grid on
title('Epsilon-greedy algorithm');

time = 0:1:numIteration;
andamentoQ = horzcat(initialQ,andamentoQ);

figure(2)
plot(time, andamentoQ, LineWidth = 1);
grid on
title('andamento del valore delle azioni Q');
legend('Q(1)','Q(2)','Q(3)','Q(4)','Q(5)');

fprintf("Reward finale per il giocatore 1: ");
disp(finalR);
```

TEST SU *ε - greedy sample average* ($\varepsilon = 0.25$)

(Fig.1)



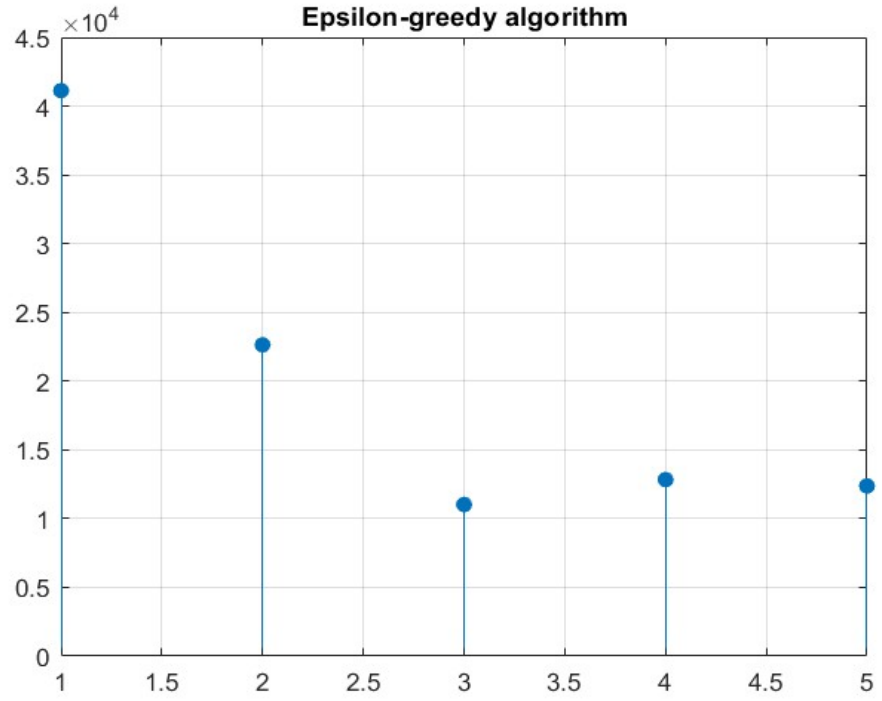
(Fig.2)



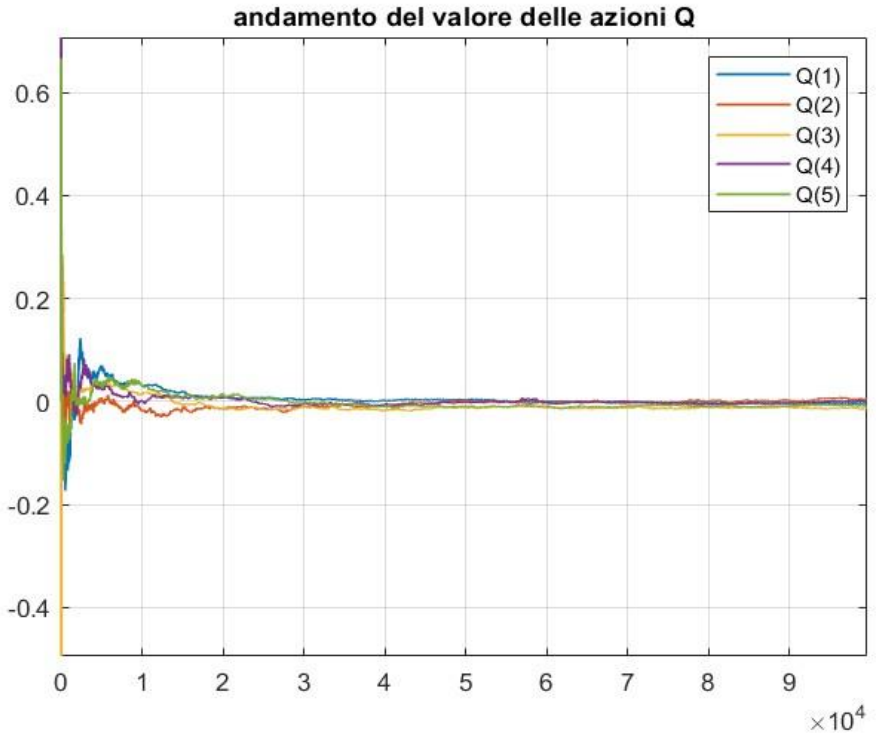
Questo valore di ε garantisce poca esplorazione e maggiore sfruttamento delle informazioni note. Da **(fig.1)** si nota che l'azione 1 è quella greedy e quindi viene presa molto di più rispetto alle altre, che nelle poche occasioni in cui sono considerate, avranno restituito una ricompensa minore. In **(fig.2)** si evidenzia l'andamento di Q e come tende a 0 per ogni azione (0 corrisponde al valore atteso del reward, essendo esso un parametro che può valere : $+1, -1, 0$).

TEST SU *ε - greedy sample average* ($\varepsilon = 0.5$)

(Fig.3)



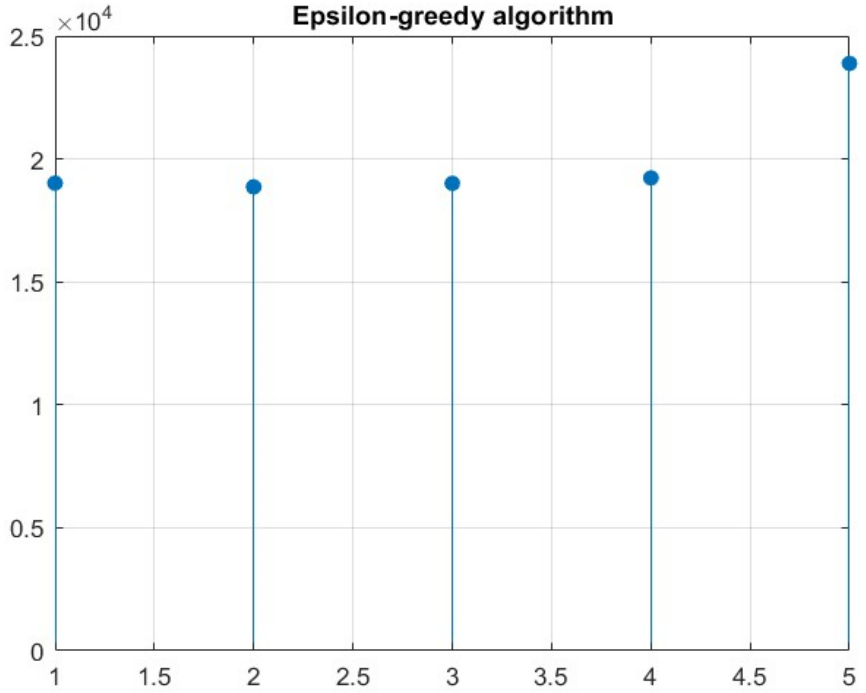
(Fig.4)



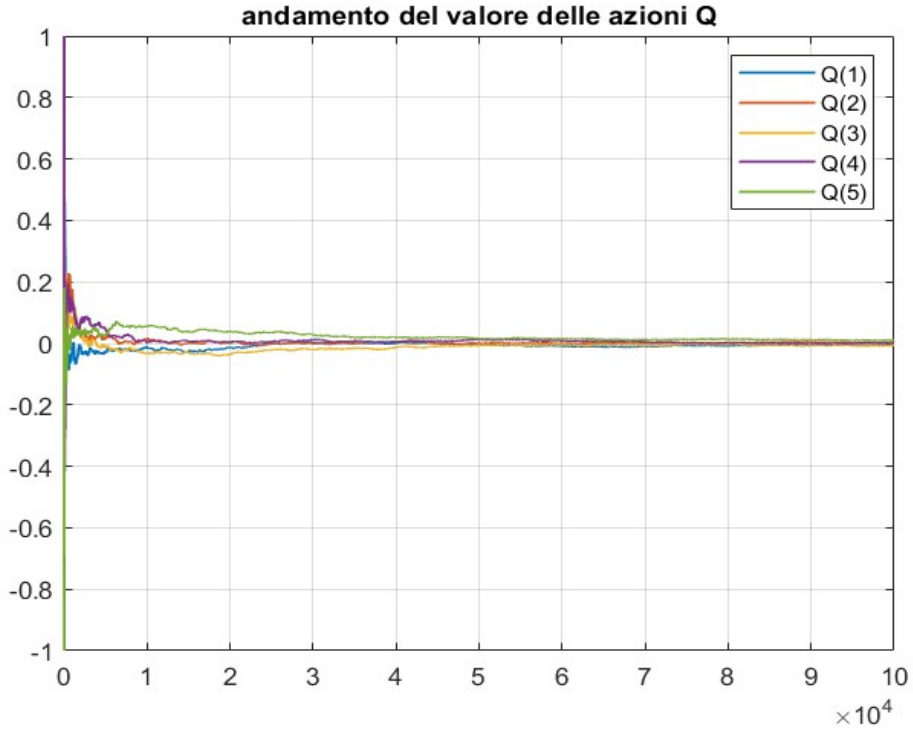
Con questo ε si nota in **(fig.3)** l'aumentare dell'esplorazione, infatti l'azione 1 greedy è presa meno rispetto al caso precedente, favorendo un aumento delle azioni prese a caso. Nella **(fig.4)** si osserva un'oscillazione maggiore delle stime delle azioni in quanto esse sono considerate più volte. Le stime Q tendono a 0 con il passare del tempo.

TEST SU *ε - greedy sample average* ($\varepsilon = 0.95$)

(Fig.5)



(Fig.6)



Da **(fig.5)** si osserva come tutte le azioni tendono ad avere un valore di $N(a)$ simile tra loro; questo perché la probabilità di prendere l'azione casuale aumenta a discapito dell'azione greedy. Anche in questo caso si osserva in **(fig.6)** un aumento delle oscillazioni e come le stime Q tendono a 0 per istanti di tempo molto grandi.

POLICY Upper Confidence Bound

CARATTERISTICHE DEL Upper Confidence Bound - optimistic initialization

Il problema dell' **ε - greedy** è il seguente : le azioni non-greedy vengono scelte casualmente.
Per questo motivo l'algoritmo **upper confidence bound** seleziona tra le azioni non-greedy quella con il potenziale maggiore (cioè l'azione che potrebbe restituire la ricompensa migliore).

Quindi l'azione scelta rispetta la condizione seguente : $A_t = \arg max_a \{Q_t(a) + c * \sqrt{\frac{\ln(t)}{N_t(a)}}\}$

dove :

- Il termine $c > 0$ controlla il **grado di esplorazione** : un valore alto da molto peso alla misura di incertezza.
- Il termine $\sqrt{\frac{\ln(t)}{N_t(a)}}$ indica la **misura dell'incertezza** : all'istante t un'azione fino ad ora presa molte volte ha una misura di incertezza bassa, mentre un'azione presa poche volte ha una misura di incertezza alta; quindi un'azione fino ad ora poco considerata sta aumentando la probabilità di essere scelta successivamente.

La tecnica di **optimistic initialization** consiste nell'assegnare valori alti alle stime $Q_0(a) \ \forall a \in \mathcal{A}$ all'istante iniziale ($t = 0$);
ciò permette di aumentare l'esplorazione (nella sola fase iniziale) in quanto Q_0 è maggiore del reward ottenuto da una qualsiasi azione selezionata e quindi si tendono a provare più volte tutte le azioni, prima che le stime dei valori convergono.

PASSO DI AGGIORNAMENTO IN UCB

Per quanto riguarda il **passo di aggiornamento α**, ne sono stati utilizzati due differenti:

- α costante : Questo passo di aggiornamento permette di trattare problemi non stazionari (cioè problemi in cui i rewards sono presi da una distribuzione non stazionaria) ma al tempo stesso non garantisce la validità delle condizioni di convergenza completa; Tale passo di aggiornamento, a seconda del valore assegnatogli, pesa di più (o di meno) gli ultimi rewards ottenuti rispetto a quelli meno recenti.

  La legge di aggiornamento della stima è : $Q_{k+1} = Q_k + \alpha * (R_k - Q_k) = (1-\alpha)^k * Q_1 + \sum_{i=1}^{k} \alpha * (1-\alpha)^{k-i} * R_i$

- α variabile : Questo passo di aggiornamento è utile per algoritmi stazionari (cioè la cui distribuzione del reward non cambia nel tempo) ed inoltre rispetta le condizioni di convergenza in ogni istante di tempo.

  La legge di aggiornamento della stima è : $Q_{k+1} = Q_k(a) + \frac{1}{N_k(a)} * \big(R_k - Q_k(a)\big), \quad \alpha(k) = \frac{1}{N_k(a)}$

IMPLEMENTAZIONE DI UCB - optimistic initialization – constant step size

```
fprintf("Ordine delle azioni nell'insieme A: ");
disp('1: Sasso, 2: Carta, 3: Forbici, 4: Spock, 5: Lizard');

rng(1);

% 1 = Sasso, 2 = Carta, 3 = Forbici, 4 = Spock, 5 = Lizard
A = 5;                          %Numero di azioni possibili

alpha = 0.01;                   %Inizializzazione del constant step-size
c = 0.75;                       %Inizializzazione del grado di esplorazione

%Inizializzo il vettore delle stime del valore dell'azione che viene poi
%aggiornato effettuando una media aritmetica dei reward
Q = 10*ones(A,1);
%Inizializzo il vettore che tiene traccia per ogni azione del numero di
%volte che questa viene presa
N = zeros(A,1);

%Definisco un numero di azioni sufficientemente grande così da garantire
%che per istanti di tempi grandi le stime dei valori di azione siano uguale
%ai valori delle azioni q(a).
numIteration = 100000;          %Numero di iterazioni totali
counter = 0;                    %Contatore delle iterazioni considerate

finalR = 0;                     %Valore del ritorno alla fine del loop
initialQ = Q;                   %Salvo il valore di partenza della variabile Q
andamentoQ = zeros(A,numIteration);         %Vettore che mantiene traccia di Q nel tempo
```

Scelta delle azioni →

Calcolo del reward per il giocatore 1 →

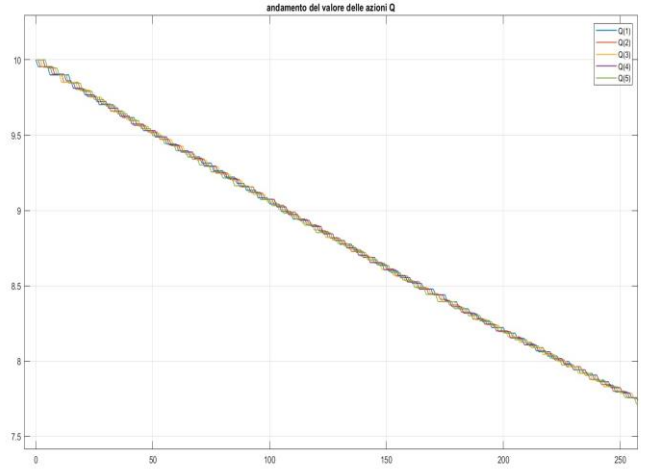
Aggiornamento dei valori delle azioni →

```
while (counter < numIteration)
    counter = counter + 1;                  %Aggiornamento del contatore delle iterazioni

    %Scelta dell'azione da prendere all'istante t dai due giocatori
    At = UCB(Q,N,c,counter);                %Azione presa all'istante t dal giocatore 1
    ARand = randi(A);                       %Azione presa all'istante t dal giocatore 2

    %Calcolo della ricompensa per il giocatore 1
    Rt = valutaVincitore(At,ARand);         %Ricompensa all'istante t del giocatore 1
    finalR = finalR + Rt;                   %Aggiornamento della ricompensa totale

    %Incremento delle variabili N e Q relative solo all'azione considerata
    %dal giocatore 1, mentre il resto rimane invariato
    N(At,1) = N(At,1) + 1;
    Q(At,1) = Q(At,1) + (alpha .* (Rt - Q(At,1)));

    for i = 1:A
        andamentoQ(i,counter) = Q(i,1);     %Inserimento dei nuovi dati nel vettore
    end
end

figure(1)
stem(N,'filled','-');
grid on
title('UCB algorithm with optimistic initialization and costant step-size');

time = 0:1:numIteration;
andamentoQ = horzcat(initialQ,andamentoQ);

figure(2)
plot(time, andamentoQ, LineWidth = 1);
grid on
title('andamento del valore delle azioni Q');
legend('Q(1)','Q(2)','Q(3)','Q(4)','Q(5)');

fprintf("Reward finale per il giocatore 1: ");
disp(finalR);
```

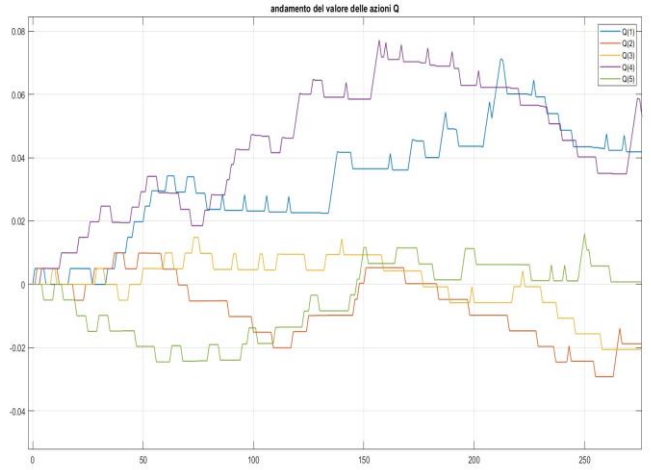
TEST SU UCB - optimistic initialization - constant step size

$\alpha = 0.005$ $rng(5)$	$Q_0 = zeros(A,1)$	$Q_0 = 10 * ones(A,1)$
$c = 0.25$	15299 36011 06944 03683 38063	38421 35391 12293 06595 07300
$c = 0.5$	26976 17978 23711 19677 11658	04303 54217 12143 09021 20316
$c = 0.75$	11333 44948 23220 14036 06463	22582 14145 36417 12930 13926

(Fig.7) Vettore N all'istante finale



(Fig.8) Optimistic initialization con $c = 0.75$



(Fig.9) No optimistic initialization con $c = 0.75$

Come affermato in precedenza, assegnare determinati valori al vettore delle stime Q all'istante iniziale, definisce un determinato livello di esplorazione (solo negli istanti iniziali).

Allo stesso modo, variando la costante c si definisce il grado di esplorazione in quanto il termine di incertezza va ad influenzare molto ($c \to 1$) o poco ($c \to 0$) sulla stima $Q_t(a)$ durante il calcolo del nuovo valore.

TEST SU UCB - optimistic initialization - constant step size

$Q = 10 * ones(A,1)$	$\alpha = 0.005$	$\alpha = 0.05$	$\alpha = 0.5$
$c = 0.25$	-0.1385 -0.0288 -0.1431 -0.1453 -0.1428	-0.5232 -0.5193 0.2206 -0.5058 -0.5160	-0.9967 0.9963 -0.9971 -0.9990 0.1340
$c = 0.5$	-0.1467 -0.0288 -0.1371 0.1389 -0.1334	-0.4982 -0.1268 -0.5136 -0.5147 -0.5163	-0.9976 -0.9974 -0.9967 -0.9988 0.1340
$c = 0.75$	-0.1240 -0.0288 -0.1203 -0.1306 -0.1292	-0.5232 -0.5481 -0.5167 0.5305 -0.0057	-0.9974 -0.9973 -0.8297 -0.9978 -0.9988

(Fig.8) Vettore delle stime Q all'istante finale

Il passo di aggiornamento α costante, non garantisce la validità delle condizioni di convergenza, infatti dalla figura **(fig.8)** è possibile osservare che all'aumentare di α il vettore delle stime $Q(a)$ non converge al valore atteso della ricompensa, ma al valore minimo -1.

Questo perché scegliendo un passo di aggiornamento elevato, si dà poco peso alle ricompense positive e molto peso alle ricompense negative, infatti vale la relazione : $Q_{k+1} = Q_k + \alpha * (R_k - Q_k)$.

Perciò per un periodo di tempo sufficientemente lungo la stima Q tende al valore minimo anche se si otterranno delle ricompense positive.
Tale comportamento si può attenuare riducendo il passo di aggiornamento;

IMPLEMENTAZIONE DI UCB - optimistic initialization – variable step size

```
%A è l'insieme delle azioni possibili
% 1 = ROCK, 2 = PAPER, 3 = SCISSORS, 4 = SPOCK, 5 = LIZARD
A = 5;

%N è il vettore che tiene traccia delle volte che si sceglie un'azione. In
%questo caso viene aggiornato solo la componente associata all'azione
%presa
N = zeros(A,1);

%Definisco un numero di azioni sufficientemente grande così da garantire
%che per istanti di tempi grandi le stime dei valori di azione siano uguale
%ai valori delle azioni q(a).
numIteration = 10000;
counter = 1;     %counter viene aumentato ad ogni iterazione dell'algoritmo

%Definisco la variabile seguente per ottenere alla fine del ciclo il rewa
%totale ottenuto
RtTot = 0;

%Q è il vettore delle stime dei valori delle azioni
Q = 10*ones(A,1);
%Definisco la costante "c" che controlla il grado di esplorazione
%dell'algoritmo. Questa variabile è maggiore di 0 e minore di 1.
c = 0.25;
```

Scelta delle azioni →

```
while counter <= numIteration
    %A questo punto il giocatore 1 sceglie l'azione tra quelle proposte per
    %mezzo dell'algoritmo UCB come segue
    if counter == 1
        At = randi(A);
    else
        At = UpperConfidenceBound(Q,N,c,counter);
    end

    %A questo punto il giocatore 2 (computer) sceglie casualmente l'azione
    %da effettuare tra quelle proposte
    ARand = randi(A);

    %La seguente funzione restituisce il reward all'istante t del giocatore
    %1
    Rt = valutaVincitore(At,ARand);

    %Aggiornamento delle variabili Q ed N associate alla sola azione "a"
    %utilizzata all'istante t
    N(At) = N(At)+1;
    Q(At) = Q(At) + (1/N(At))*(Rt-Q(At));

    %Aggiornamento della variabile che tiene il conto delle iterazioni
    %effettuate
    counter = counter+1;
end
```

Calcolo del reward per il giocatore 1 →

Aggiornamento dei valori delle azioni →

TEST SU UCB - optimistic initialization – variable step size

$rng(5)$	$Q_0 = zeros(A,1)$	$Q_0 = 10 * ones(A,1)$
$c = 0.25$	00001 00001 00020 99977 00001	00001 00001 99996 00001 00001
$c = 0.5$	00002 00150 33137 02574 64137	00002 00002 84736 15258 00002
$c = 0.75$	25385 05891 29548 10575 28601	14239 39922 13068 15159 17612

(Fig.9) Vettore N nell'istante finale

Considerando il passo di aggiornamento $\alpha = {}^{1}/_{N_t(a)}$ variabile , se un'azione viene presa per la prima volta e questa restituisce reward negativo allora la nuova stima $Q_{k+1}(a)$ calerà di molto e l'azione avrà poche possibilità future di essere considerata nuovamente (dipende dal parametro c).

Il peso di una ricompensa negativa e ancor di più di una ricompensa positiva, decresce all'aumentare del numero di volte che l'azione associata è considerata (decremento del termine $\alpha = {}^{1}/_{N_t(a)}$)

Il comportamento descritto è riportato in **(fig.9)**.
Dalla seguente figura si nota come all'aumentare del parametro c si ha una distribuzione maggiore in N; questo perché le azioni prese meno hanno un valore del termine di incertezza maggiore che all'aumentare di c assume più rilevanza.

$$\left(A_t = \arg max_a \left\{Q_t(a) + c * \sqrt{\frac{\ln(t)}{N_t(a)}}\right\}\right)$$

TEST SU UCB - optimistic initialization – variable step size

$rng(5)$	$Q_0 = zeros(A,1)$	$Q_0 = 10 * ones(A,1)$
$c = 0.25$	-1 -1 -0.2000 -0.0006 -1	-1 -1 -0.0022 -1 -1
$c = 0.5$	-1 -0.1333 -0.0047 -0.0287 0.0030	-1 -1 -0.0045 -0.0139 -1
$c = 0.75$	-0.0022 -0.0194 0.0009 -0.0109 -0.0013	-0.0036 0.0050 -0.0023 -0.0029 -0.0014

(Fig.10) Vettore delle stime Q all'istante finale

Consideriamo il passo di aggiornamento $\alpha = {}^{1}/_{N_t(a)}$ variabile, si nota in figura **(fig.10)** come le azioni che vengono prese per la prima volta e che restituiscono reward negativo, non vengano più considerare successivamente se e solo se il parametro c è basso.
Questo perché tali azioni in istanti di tempo successivi non riescono ad assumere una stima maggiore e/o uguale a quella delle altre azioni, che invece possono aver restituito già dall'inizio un reward positivo / nullo.

Un vantaggio del passo di aggiornamento variabile è che la stima $Q_\infty(a)$ converge a 0 (valore medio della ricompensa) per la validità della proprietà di convergenza.

Inoltre, all'aumentare del parametro 'c', si dà più importanza al termine di incertezza, il quale ci permette di considerare anche azioni prese meno nel tempo.

POLICY Preference Updates

CARATTERISTICHE DEL Preference Updates

In questo caso si introduce il concetto di preferenza di un'azione, indicata con il termine $H_t(a)$.
Il valore di preferenza di un'azione è inizialmente lo stesso per tutte le azioni dell'insieme $\mathcal{A}$.

L'azione presa all'istante t non dipende più dal vettore delle stime Q_t, ma è in accordo con la distribuzione soft-max:

$$\Pr[A_t = a] = \frac{e^{H_t(a)}}{\sum_{b=1}^{n} e^{H_t(b)}} = \pi_t(a)$$

Scelta l'azione migliore A_t (cioè quella che massimizza la variabile $\pi_t(a)$) e ricevuto il reward R_t, si aggiorna il vettore delle preferenze $\bar{H}$ come segue :

$$\begin{cases} H_{t+1}(A_t) = H_t(A_t) + \alpha * (R_t - \bar{R}_t) * \big(1 - \pi_t(A_t)\big), & a = A_t \\ H_{t+1}(a) = H_t(a) - \alpha * (R_t - \bar{R}_t) * \pi_t(a), & \forall a \neq A_t \end{cases}$$

(α = passo di aggiornamento costante, $\bar{R}_t$= media di tutti i rewards ottenuti fino all'istante t)

Dalla definizione di aggiornamento del vettore $\bar{H}$, vale che:

- Se $R_t > \bar{R}_t$, allora la probabilità di prendere l'azione A_t all'istante $t+1$ aumenta;
- Se $R_t < \bar{R}_t$, allora la probabilità di prendere l'azione A_t all'istante $t+1$ diminuisce.

(VALE IL VICEVERSA DI QUANTO DETTO PER LE AZIONI NON SELEZIONATE)

IMPLEMENTAZIONE DI Preference Updates

```
%A è l'insieme delle azioni possibili
% 1 = ROCK, 2 = PAPER, 3 = SCISSORS, 4 = SPOCK, 5 = LIZARD
A = 5;

%N è il vettore che tiene traccia delle volte che si sceglie un'azione. In
%questo caso viene aggiornato solo la componente associata all'azione
%presa
N = zeros(A,1);

%Definisco un numero di azioni sufficientemente grande così da garantire
%che per istanti di tempi grandi le stime dei valori di azione siano uguale
%ai valori delle azioni q(a).
numIteration = 10000;
counter = 0;     %counter viene aumentato ad ogni iterazione dell'algoritmo

%Definisco la variabile seguente per ottenere alla fine del ciclo il reward
%totale ottenuto
RtTot = 0;

%Definisco il reward medio rispetto a tutte le azioni fino all'istante t
%incluso
averageRt = 0;

%Definisco il passo di aggiornamento (step-size) costante.
alpha = 0.25;
%Inizializzo il vettore delle preferenze H, utilizzato per assegnare una
%certa preferenza a ciascuna azione di A.
%Inizialmente tutte le preferenze sono uguali.
H = zeros(A,1);
```

Scelta delle azioni →

Calcolo del reward per il giocatore 1 →

Aggiornamento dei valori delle preferenze →

```
while counter < numIteration
    %Aggiornamento del contatore che tiene conto dell'istante di tempo
    %interessato
    counter = counter+1;

    %Il giocatore 1 sceglie l'azione tra quelle proposte per mezzo
    %dell'algoritmo di PREFERENCE UPDATES
    At = preferenceUpdates(H);
    %%A questo punto il giocatore 2 (computer) sceglie casualmente l'azione
    %da effettuare tra quelle proposte
    ARand = randi(A);

    %La seguente funzione restituisce il reward all'istante t
    Rt = valutaVincitore(At,ARand);

    %Aggiorno la media dei reward ottenuti finora
    averageRt = averageRt + (1/counter)*(Rt-averageRt);

    %Calcolo della preferenze associate ad ogni azione tramite la
    %distribuzione soft-max utile all'aggiornamento del vettore H
    pi = softmax(H);

    %Aggiornamento della preferenza H dell'azione presa
    H(At) = H(At) + alpha*(Rt-averageRt)*(1-pi(At));

    for i = 1:A
        %Aggiornamento della preferenza H delle azioni non prese
        if i ~= At
            H(i) = H(i) - alpha*(Rt-averageRt)*pi(i);
        end
    end
end
```

TEST SU Preference Updates

	$\alpha = 0.005$
$rng(1)$	18221 22745 13016 32878 13140
$rng(5)$	19170 29085 12664 18794 20287
$rng(10)$	17519 16590 21162 19244 25485

(Fig.11) Preference updates con $\alpha = 0.005$

	$\alpha = 0.05$
$rng(1)$	1085 7044 9162 3332 79377
$rng(5)$	1085 7044 9162 3332 79377
$rng(10)$	2134 7781 6105 1959 82021

(Fig.12) Preference updates con $\alpha = 0.05$

	$\alpha = 0.5$
$rng(1)$	3175 22 81 96697 25
$rng(5)$	99871 56 23 32 18
$rng(10)$	189 98748 30 11 1022

(Fig.13) Preference updates con $\alpha = 0.5$

All'aumentare di α aumenta il divario tra l'azione con maggior valore di preferenza e tutte le altre, questo perché la quantità sommata ad H_t incide maggiormente sul calcolo di H_{t+1}; questo comportamento lo si può osservare nelle figure **(fig.11)**, **(fig.12)**, **(fig.13)**.

Scelta un'azione A_t e restituito un reward R_t, allora un valore elevato di α fa sì che il reward crei una differenza elevata tra l'azione A_t e tutte le altre azioni a, sia in maniera positiva (cioè ritorna reward positivo e quindi viene preferita anche in istanti di tempo futuri) sia in maniera negativa (cioè ritorna reward negativo e quindi vengono preferite tutte le altre azioni rispetto ad essa per istanti di tempo futuri).

GRAZIE

LUCA SUGAMOSTO (0324613)

MATTIA QUADRINI (0334381)

luca.sugamosto@students.uniroma2.eu

mattia.quadrini.1509@students.uniroma2.eu