République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ESI ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
Ecole nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

TP BDM

TP RL QLearning

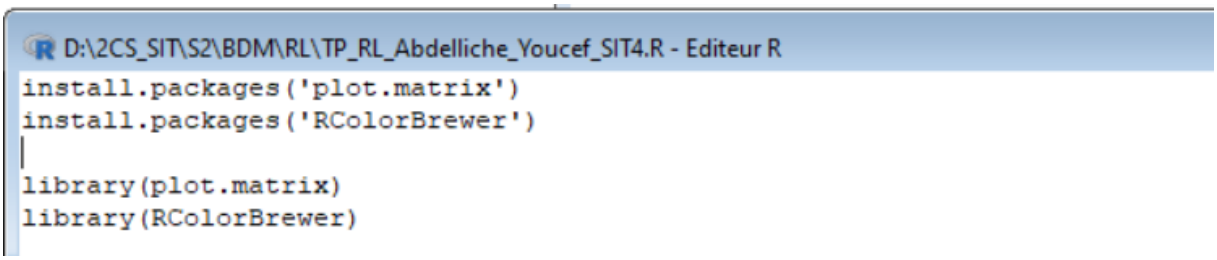
Réalisé par :

- Abdelliche Youcef

Module : BDM

2020/2021

Téléchargement et installation des packages :



La fonction ci-dessous va permette de visualiser sur R n'importe quelle matrice en utilisant le package **plot.matrix :**

```
plot_matrix <- function(mat_pot, digits_arg){
 digits_arg <- ifelse(missing(digits_arg),0,digits_arg)

 plot(mat_pot, cex = 1.2, fmt.cell=paste0('%.',digits_arg,'f'),

 col=brewer.pal(3,"Blues"), breaks=c(-500, 0, 0, 500),

 xlab="States", ylab = "States",main = "")

}
```

Définition de l'environnement :

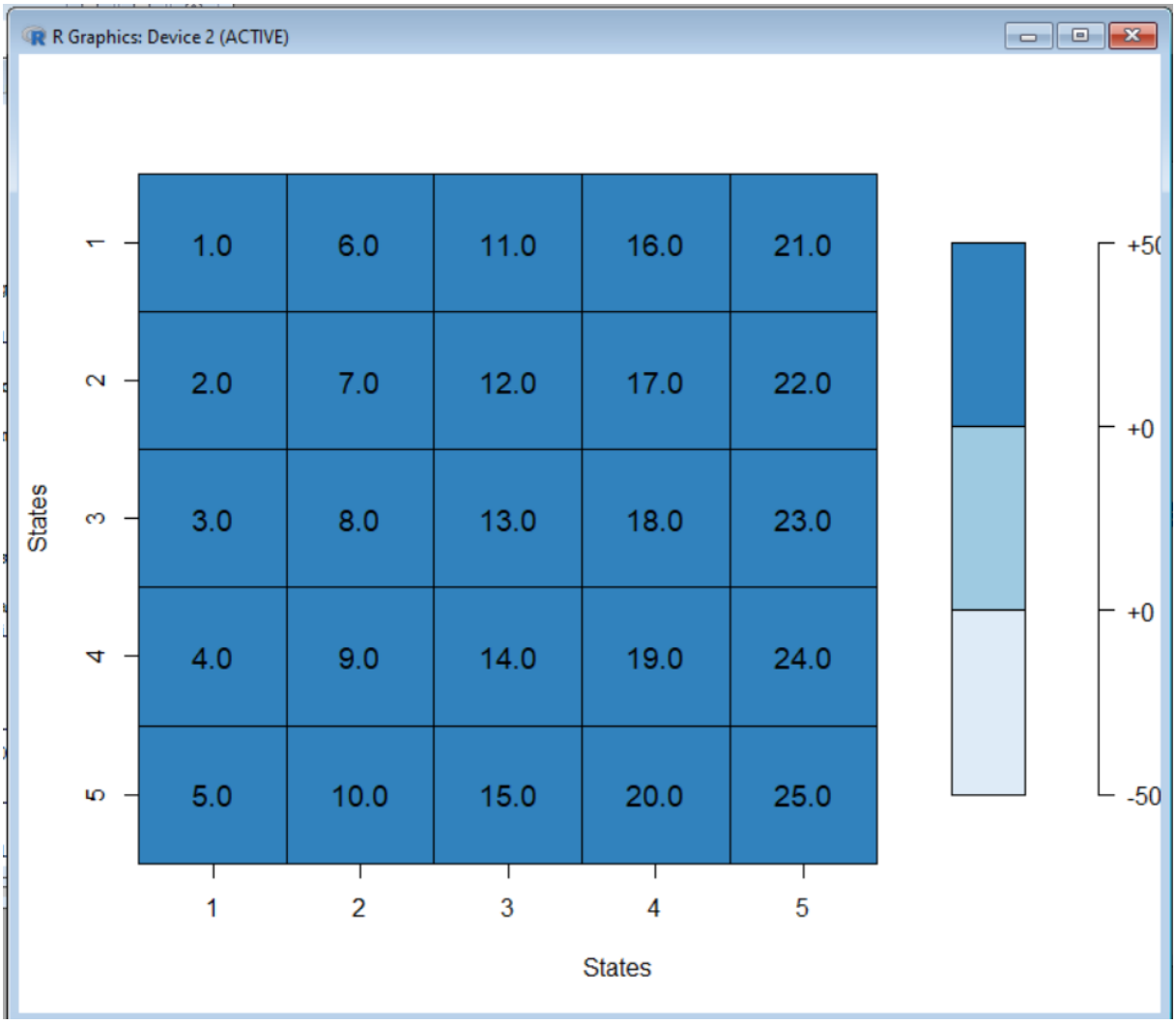
States et actions :

Dans notre exemple, on va prendre 5 états (states) : **1,2,3,4,5** et 4 actions possibles : **gauche, droite, haut, bas.**

Le code ci-dessous va créer une séquence des états (de 1 à 5) et puisqu'on peut passer d'un état à un autre en utilisant les actions, donc on va créer une matrice de 5*5 avec 5 est le nombre des états.

```
states <- seq(1, 5, by = 1)
state_seq <- cbind(merge(states,states), state = seq(1,length(states)*length(states)))

state_mat <- matrix(state_seq$state, nrow = length(states), ncol= length(states))
plot_matrix(state_mat,1) ## matrix, digits
```

Notre environnement est donc la matrice states * states = 5 * 5. On peut la visualiser en exécutant la dernière instruction dans le code ci-dessus :



Rewards et Goal (Récompenses et objectives) :

Une fois que nos états et nos actions sont définis, ensuite on a besoin de la récompense pour chacun des états. On a donné au hasard des récompenses à divers états et créé une matrice :

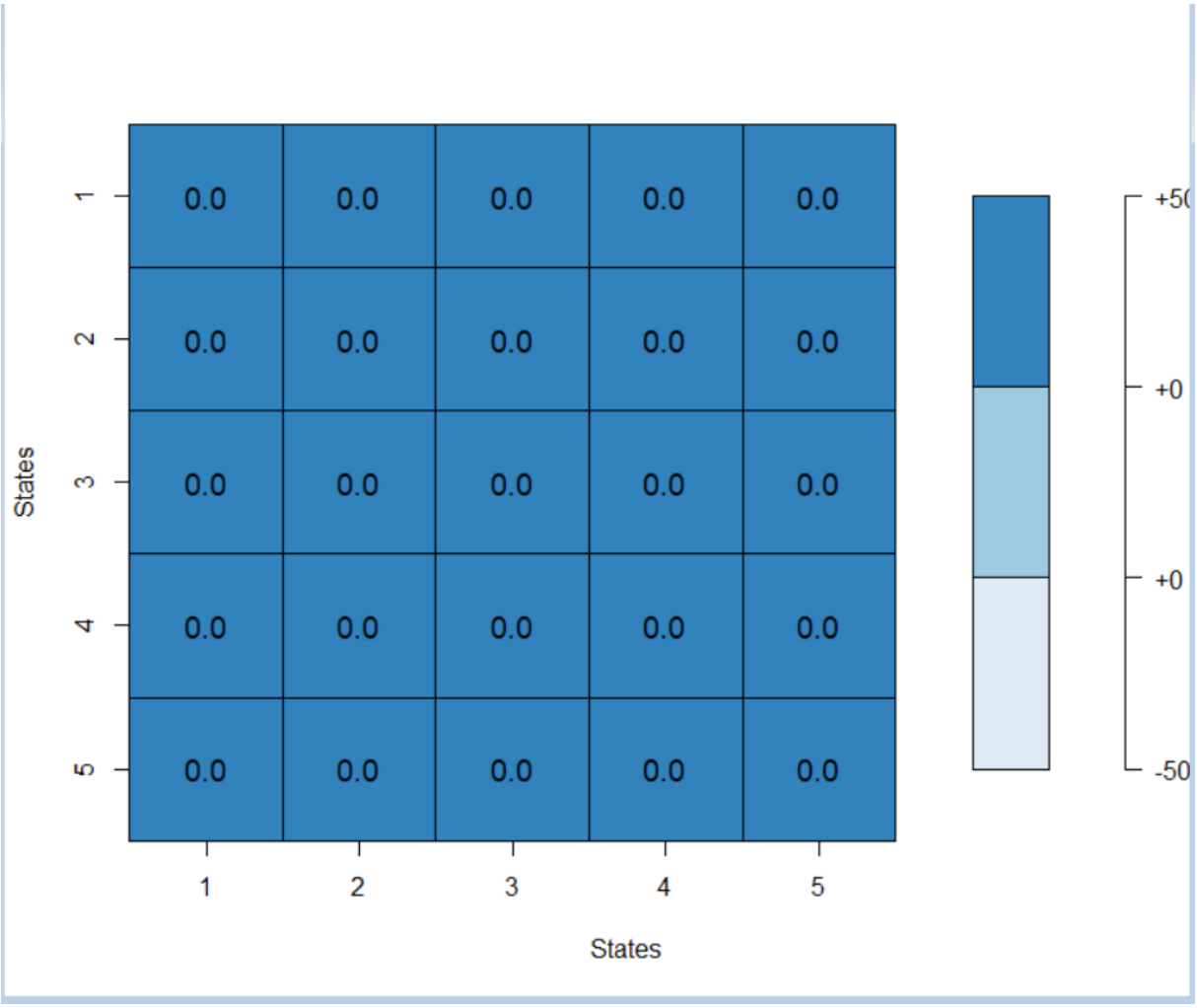
```
rewards <- c(0,-100,-100,0,-100,
                     10,10,10,10,-100,
                     10,10,-100,10,-100,

                     10,10,10,10,-100,

                    -100,-100,10,10,100 )

rewards

rewards_mat <- matrix(rewards, nrow = length(states), ncol= length(states))

plot_matrix(rewards_mat)

max(rewards)

goal <- which(rewards==max(rewards), arr.ind=TRUE)
```

Dans la matrice ci-dessus, si on compare la matrice des rewards (récompenses) avec la matrice des états on voie que les numéros d'état 2, 3, 5, 10, 13, 15, 20, 21 et 22 ont des récompenses de -100 points, ce qui signifie que nous ne voulons pas que notre agent passe par ses zones. Et aussi notre état d'objectif sera l'état 25 qui va récompenser l'agent avec 100 points.

Q-matrice :

Maintenant on va créer la Q-matrice avec les mêmes dimensions de la matrice des états et l'initialiser avec des valeurs de 0 :

```
## Initialisation de la Q-matrice
Q <- matrix(0, nrow = length(states), ncol= length(states))
plot_matrix(Q,1)
```



Next states (prochaines états) :

Nous allons maintenant définir une fonction qui va aider l'agent de savoir tous les états possibles selon l'état courant de l'agent :

```
##### fonction qui va retourner les états possibles pour un état donnée ######
getNextStates <- function(cs) {
 stalen <- length(states);
 NS <- stalen*stalen
 aa <- state_seq[state_seq$state == cs,]
 if (aa$x == max(states)) {
 ns <- c(cs - 1,
 cs - stalen,
 cs + stalen);
 } else if (aa$x == min(states)) {
 ns <- c(cs + 1,
 cs - stalen,
 cs + stalen);
 } else {
 ns <- c(cs + 1,
 cs - 1,
 cs - stalen,
 cs + stalen);
 }
 nss <- sort(ns[ns > 0 & ns <= NS]);
 return(nss);
}
```

Exemple de la fonction getNextStates pour les états 3 et 13 :

```
> getNextStates(3)
[1] 2 4 8
>
> getNextStates(13)
[1]  8 12 14 18
>
```

Pour l'état 3, l'agent peut passer vers les états suivants : 2,4,8 et pour l'état 13 : 8, 12, 14 et 18

Implémentation de l'algorithme Q-learning :

Episodes :

L'épisode est une itération qui commence par l'état initial de l'Agent jusqu'à la dernière état qui l'objectif. Ainsi, un agent commencera à partir d'un état aléatoire, apprendra tout en se déplaçant vers différents états, collectera des récompenses et mettra à jour la valeur Q à chaque étape.

Le code R ci-dessous est écrit pour 10 épisodes (on peut la modifier en changeant la valeur N) :

```
########### Exécution des épisodes ################
N <- 10 # Nbre d'épisode
alpha <- 0.8 # Taux d'apprentissage
gamma <- 0.7 # Facteur de remise
for (i in 1:N) {
 current_episode <- i;
```

```
 cat("\nStart Episode: ", current_episode)
## on chois l'état suivant parmi les actions possibles à l'état actuel
 cs <- sample(state_seq$state, 1)
 cat("\n\tCurrent state: ", cs)
 step_num <- 1;
 while (T) {
 cat("\n\n\tStep no.: ", step_num)
 cat("\n\t\tCurrent State: ", cs)
 reward <- rewards[cs]
 if(reward == 0 | is.na(reward) | length(reward) == 0 ){
 reward <- 0
 }
 cat("\n\t\tReward CS: ", reward)
 next.states <- getNextStates(cs);
 cat("\n\t\tPossible next states: ", next.states)

# prochain.états (next.states)
# Si nous avons des états présents,
 if (length(next.states) == 1) {
 ns <- next.states
# sinon choisissez au hasard.
 } else {
 ns <- sample(next.states, 1)
 }
 cat("\n\t\tNext state: ", ns)

 # Mettre a jour la valeur de  Q  pour les prochains états.
 Q[cs] <- round(Q[cs] + alpha * (reward + gamma * max(Q[getNextStates(ns)])-Q[cs]),1);
 cat("\n\t\tNew Q-Value: ", Q[cs])
# Visualisation de la nouvelle Q-matrice
 plot_matrix(Q,1)
 Sys.sleep(0.2)
```

```
if (cs == goal | step_num > 20) {
break;
}
cs <- ns;
step_num <- step_num + 1;
}
cat("\nEnd Episode: ", current_episode)
}
```

Cela exécutera 10 épisodes mettant à jour la Q-matrice.

Pour comprendre le fonctionnement du code, nous allons maintenant observer chacun des étapes :

(Les résultats sont dans le fichiers « résultats » joints)

La Q-matrice après le dernier épisode :