

Machine learning autonome : apprendre à sortir d'un labyrinthe grâce au Q-learning

Par OZDEMIR Serdar

Sommaire

Introduction : Intérêt , problématiques

I. Le coeur de l'apprentissage: théorie et formule maîtresse

- A. Théorie : agent , état , action et récompense
- B. L'algorithme de Q-learning : Formule maîtresse

II. Expérience pratique : le labyrinthe en Python

- A. Présentation du labyrinthe
- B. Transcription en python : Matrice Q , récompenses et malus
- C. Outil de lecture visuelle : interface graphique et carte “thermique”

III. Expériences et résultats

- A. Influence des coefficients sur l'interaction et sa vitesse
- B. Influencer le trajet de l'agent : résoudre le voyageur de commerce?
- C. Aspect apprentissage : Réutilisation de la même matrice Q

Conclusion

Le coeur de l'apprentissage: théorie et formule maîtresse

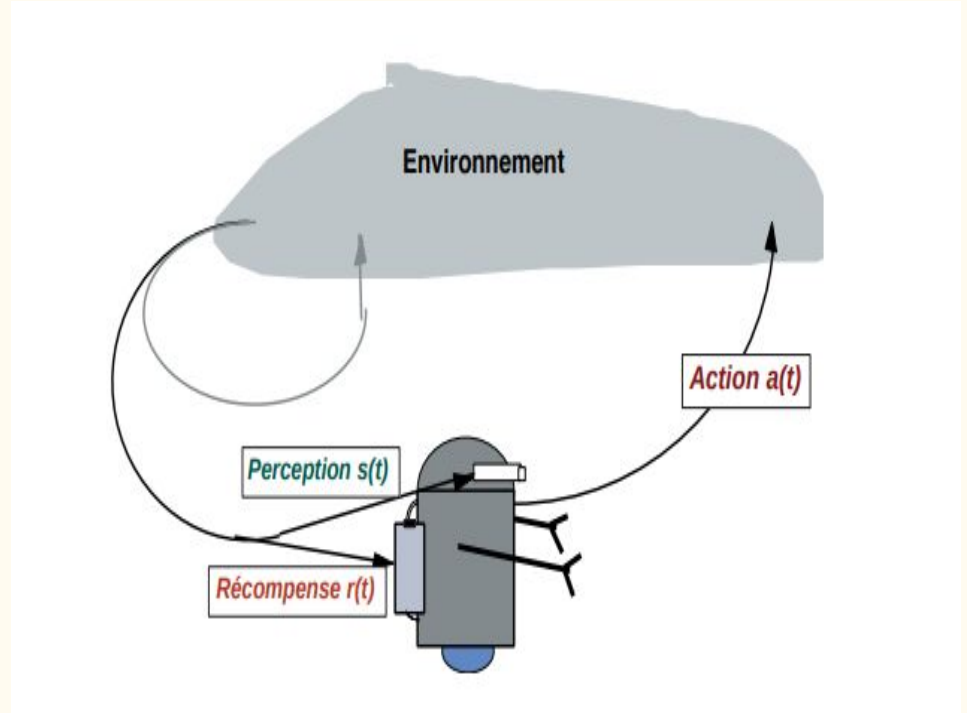


Principe

Un agent évoluant dans un milieu caractérisé par:

- Un état discret
- Une action

“Récompense” en fonction de l'utilité de l'action pour se rapprocher d'un but



Quantifier l'utilité? Formule maîtresse du Q-learning

Fonction ou matrice Q définie par itération :

d'après [4] Démontré comme convergeant
vers une succession d'actions Q^* optimale

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)]$$

\mathcal{E} L'ensemble des états

\mathcal{A} L'ensemble des actions

s Un état

a Une action

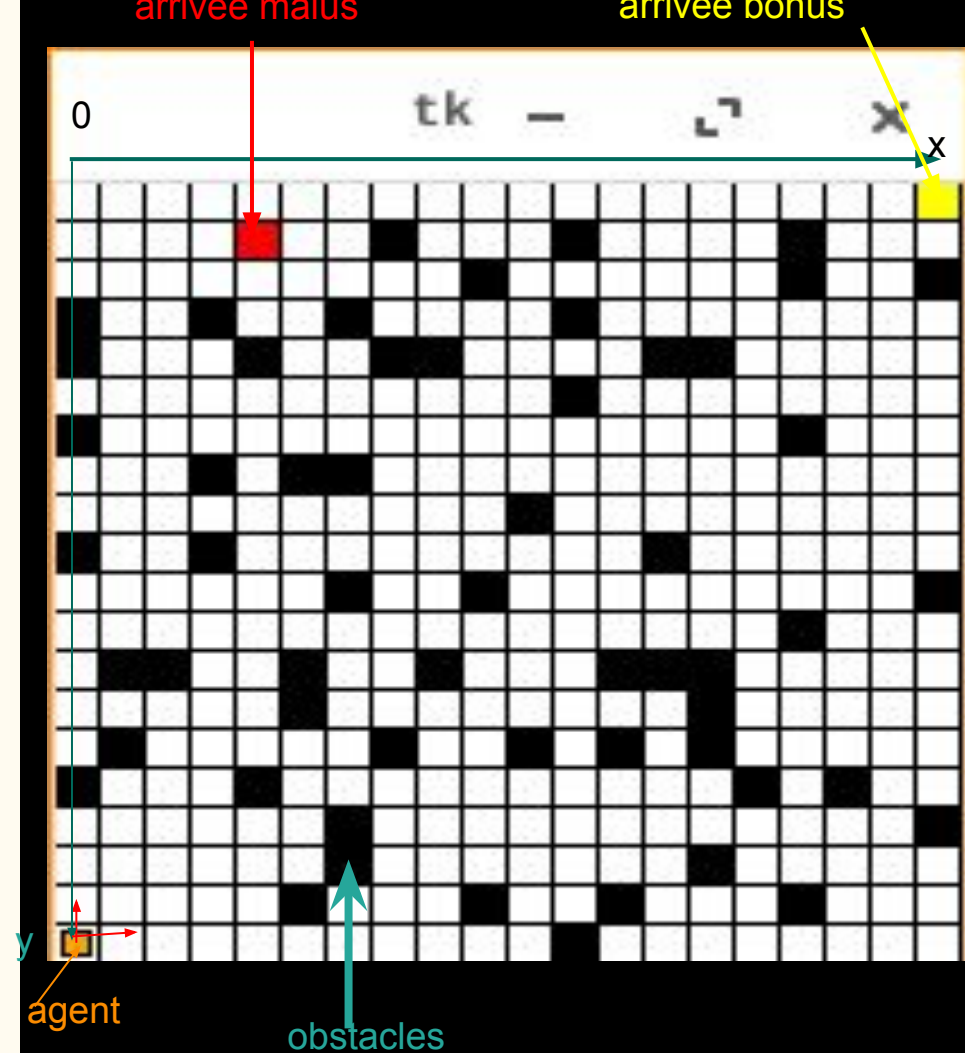
$0 \leq \gamma \leq 1$ Le taux de diminution des renforcements

Q[s][a]	action 1	action 2	action 3	action 4	...	action n	
état 1	0,2840122869	0,294580213	0,9293091949	0,2375495187	...	0,3089669029	
état 2	0,423317852	0,9370502509	0,4366532197	0,3178643791	...	0,3935787244	
état 3	0,6688511702	0,07705257775	0,4321472519	0,9057358908	...	0,8057912989	
état 4	0,07296739802	0,9314849402	0,7807701583	0,6717572414	...	0,7068750649	
...	
...	
...	
...	
...	
...	
état n-2	0,9749142464	0,2425538003	0,006806069254	0,4505776883	0,6856231027	0,2900142387	
état n-1	0,3992379726	0,5353277435	0,8625905616	0,5218028734	0,7136122717	0,3896530229	
état n	0,9000002751	0,1975965812	0,2337589261	0,8397484591	0,7743333472	0,4762497543	

Expérience pratique : le labyrinthe en Python

Présentation du labyrinthe

Parcours d'obstacle : but?
Parvenir à l'arrivée jaune en un
minimum de pas
Grille 20*20



Transcription en python

Matrice Q : utilisée comme matrice

États: tuples de position (x , y)

Actions : Déplacements possibles (haut, bas, gauche, droite)

Max Q : plus grande valeur associée à une action pour un état donné

r : récompense de l'action effectuée

ici c'est :

- un walk reward de **-0.05** pour une arrivée sur un bloc “normal”
- **+1** ou **-1** sur un bloc terminal jaune ou rouge

$$Q[s][a] = (1 - \alpha) * Q(s, a) + \alpha (r + \gamma * \max_{a' \in A} (Q(s, a')))$$

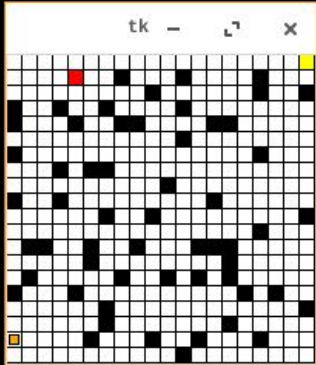
Q[s][a]	haut	bas	gauche	droite
(0 ;0)	0,8887586574	0,2318675143	0,291397744	0,2530896776
(0 ;1)	0,3159628038	0,5096924007	0,7021593453	0,3244752051
(0 ;2)	0,02252814672	0,009478397598	0,1081616977	0,4140526635
(0 ;3)	0,08914194099	0,4238189238	0,02721855479	0,9993835742
...
...
...
...
(19 ;17)	0,9647931298	0,3892247012	0,6084894526	0,7331544589
(19 ;18)	0,8670433842	0,1460729469	0,8346330623	0,1193675221
(19 ;19)	0,2679125273	0,3604113458	0,06685312541	0,9984351647

MaxQ pour un état

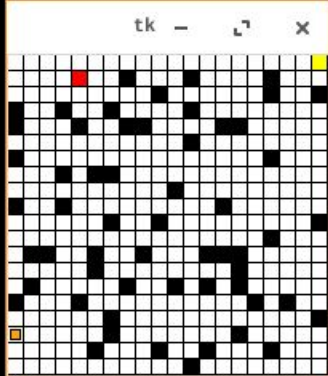
Voir annexe Apprentissage.py (lignes 60 à 106) pour l'implémentation de cette itération

Transcription en python : exemple de déplacement

```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
Q((0, 19)) = {'up': 0.1, 'down': 0.1, 'left': 0.1, 'right': 0.1}
```



```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
Q((0, 19)) = {'up': 0.1, 'down': 0.1, 'left': 0.1, 'right': 0.1}
Q((0, 18)) = {'up': 0.1, 'down': 0.1, 'left': 0.1, 'right': 0.1}
```



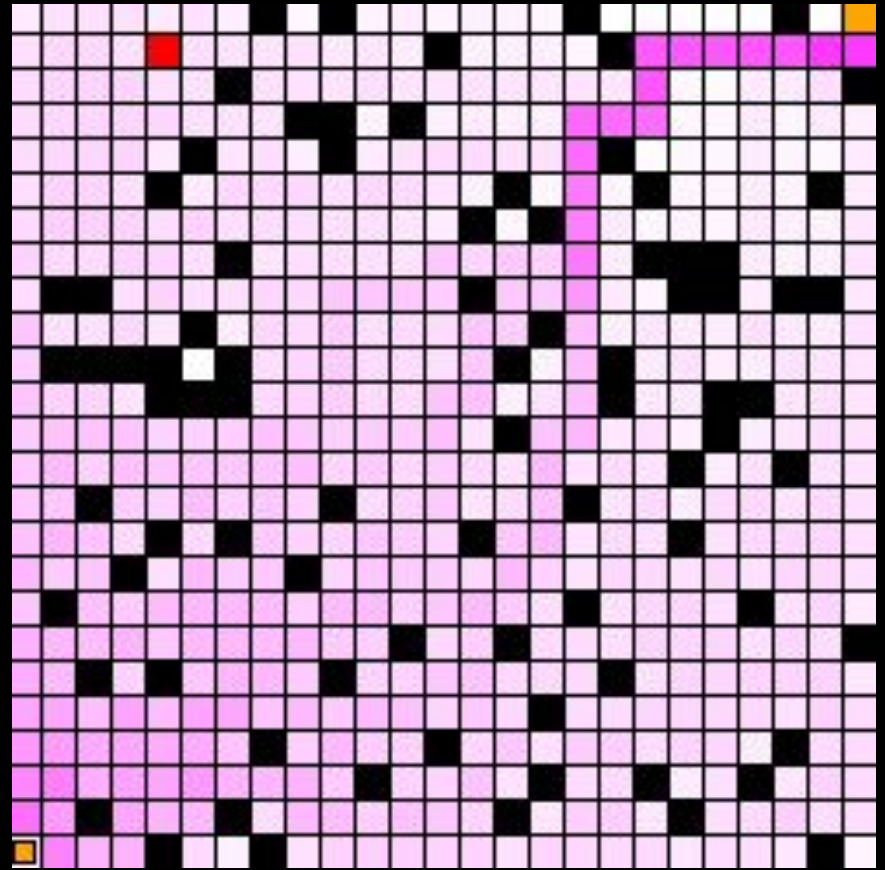
Initialisation : tous les évènements sont équiprobables

Outil de lecture visuelle : carte “thermique”

chaque passage par une case la
rosit un peu plus : visualisation
du chemin optimal trouvé

(Voir annexe Milieu.py lignes 75 à 102 + ligne 115)

L’algorithme ne s’arrête que
lorsque l’agent effectue le meme
score un certain nombre de fois
d’affilée (Voir annexe apprentissage.py fonction
signal_arret ligne 45)



Expériences et résultats

Formule et influence des coefficients

$$Q[s][a] = (1 - \alpha) * Q(s, a) + \alpha (r + \gamma * \max_{a' \in A} (Q(s, a')))$$

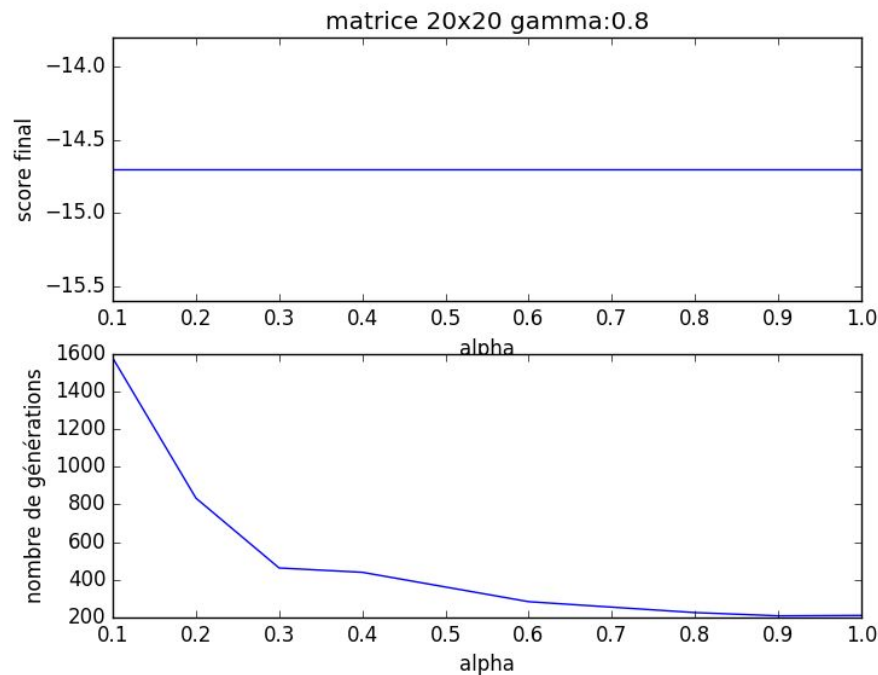
milieu déterministe : meilleur $\alpha = 1$ d'après

L'apprentissage de réflexes par renforcement :

Apprentissage artificiel: Concepts et algorithmes [4]

effectivement

*Méthodologie : voir annexes milieu_nogui.py et
apprentissage_nogui.py*



Formule et influence des coefficients

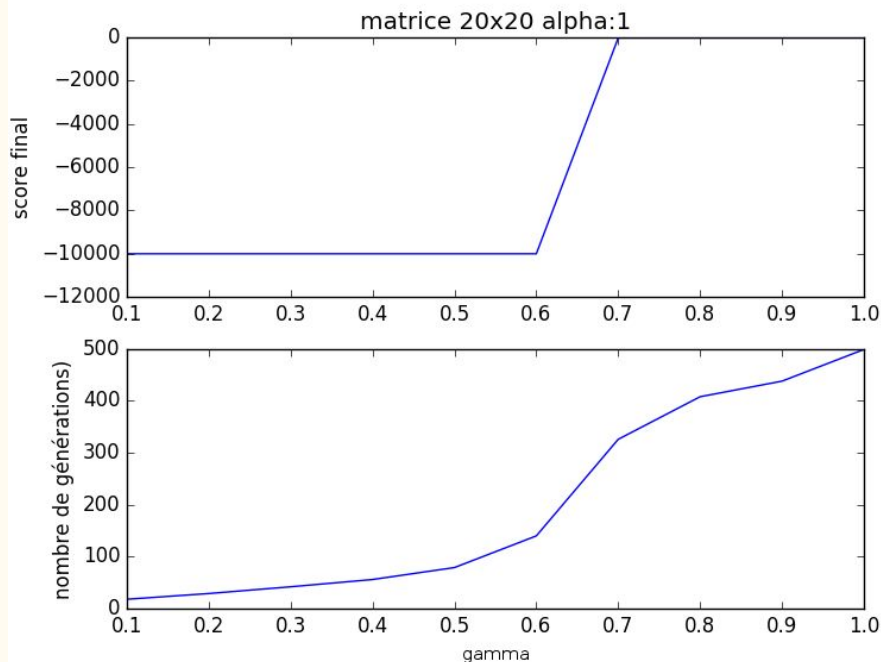
$$Q[s][a] = (1 - \alpha) * Q(s, a) + \alpha (r + \gamma * \max_{a' \in A} (Q(s, a')))$$

alpha fixé à 1 dans ce cas , influence de gamma?

Valeur minimale de gamma ! Sinon impossible de converger vers un chemin optimal ! Modifie l'interaction avec le milieu

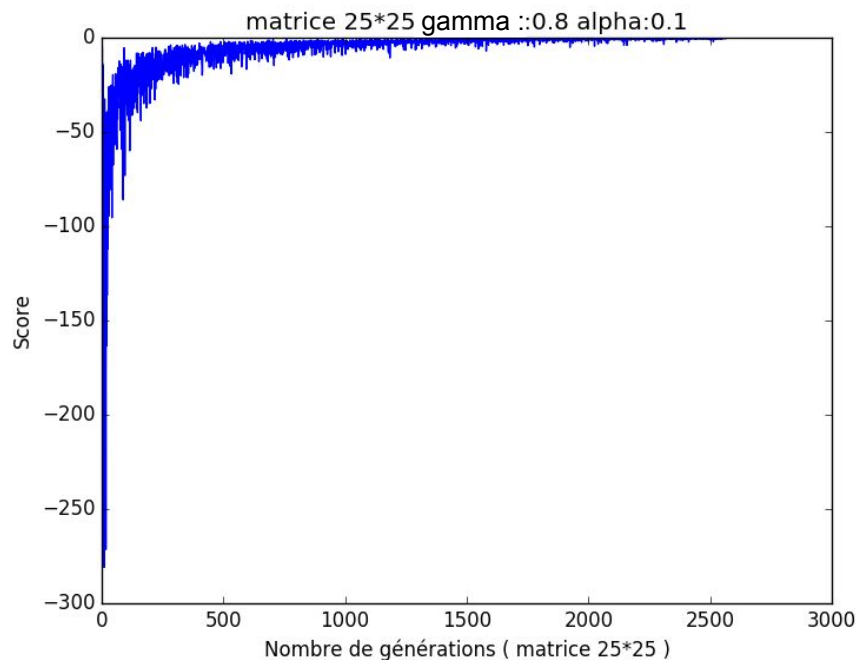
score limite en cas de divergence à -10000

Voir annexe apprentissage.py ligne 110



Formule et influence des coefficients : aspect du score

Convergence vers un chemin “presque optimal”
rapide mais chemin optimal long à obtenir

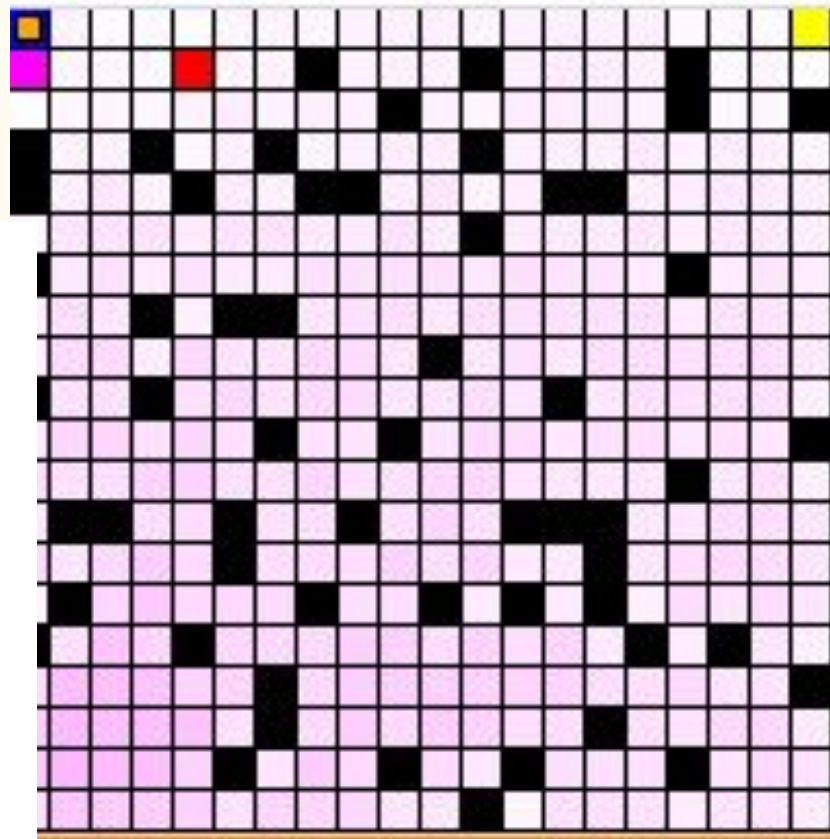
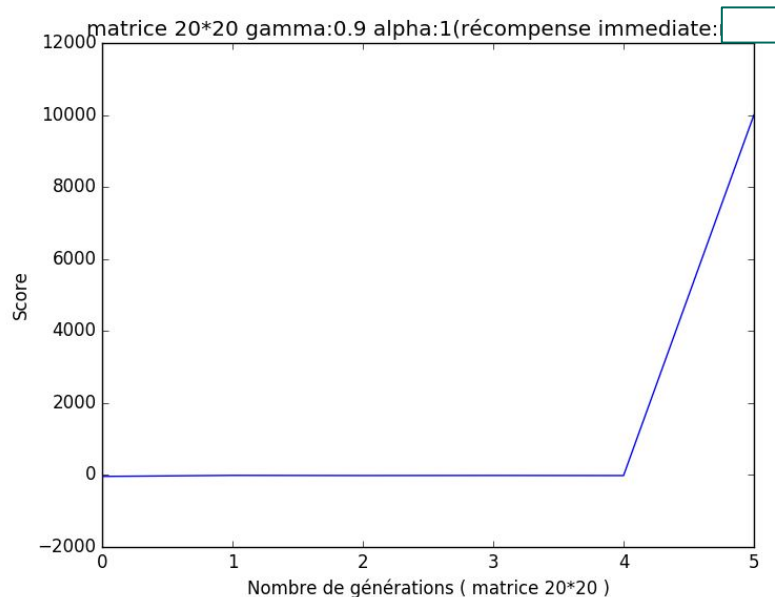


Pourquoi pas essayer d'influencer l'agent?

Expérience : bloc de “récompense” (en bleu) pour inciter à passer par ce bloc

Approche naïve : récompense immédiate

Échec



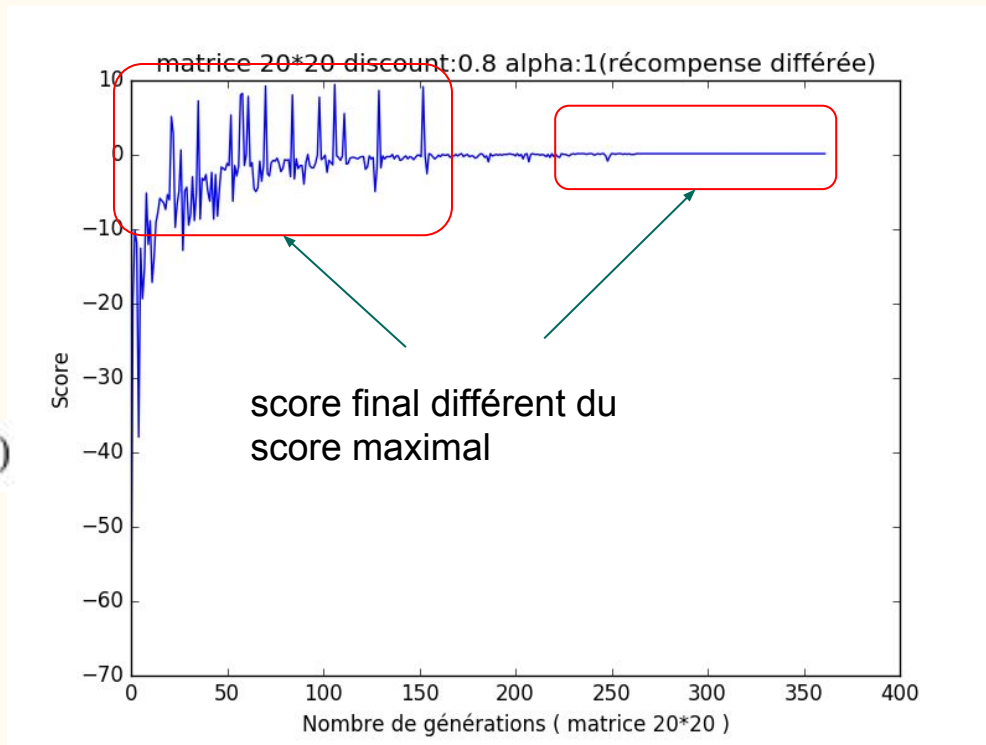
Pourquoi pas essayer d'influencer l'agent?

Hypothèse : il faut différer la récompense pour “masquer” la position du bonus à l'agent :
instauration d'un coffre (*Voir annexe Milieu.py lignes 119 à 138*) récompense seulement à l'arrivée

Échec 2 mais prévisible par la théorie :

$$Q[s][a] = (1 - \alpha) * Q(s, a) + \alpha (r + \gamma * \max_{a' \in A} (Q(s, a')))$$

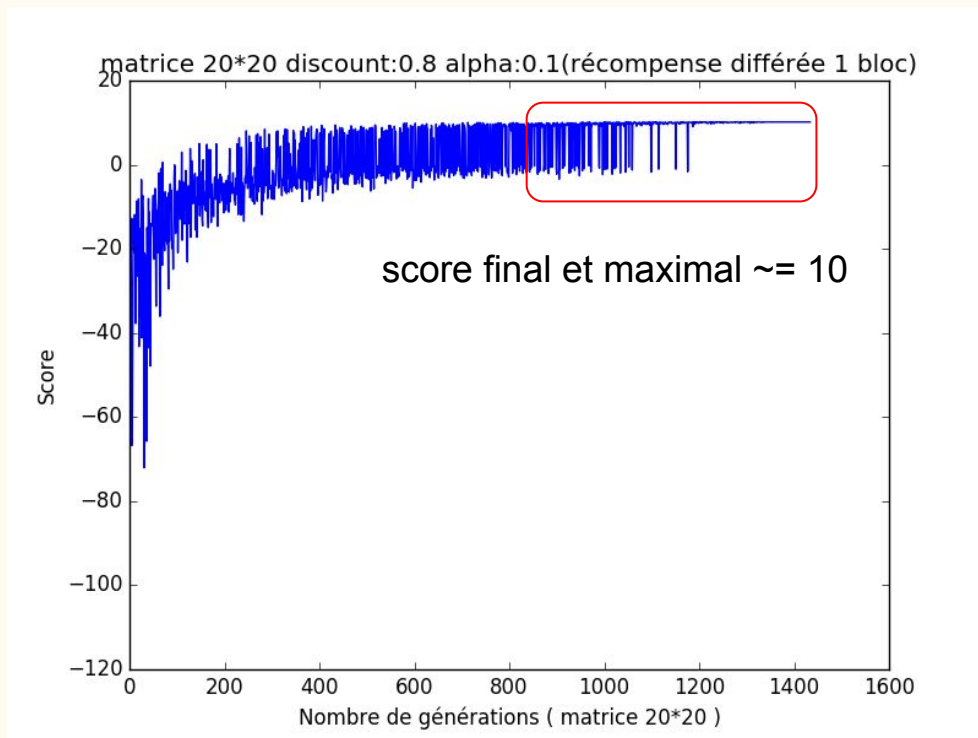
OR ICI ALPHA = 1 !



Pourquoi pas essayer d'influencer l'agent?

Avec $\alpha = 0.1$ (prend en compte état précédent) cela fonctionne!

Bonus ici initialisé à 10

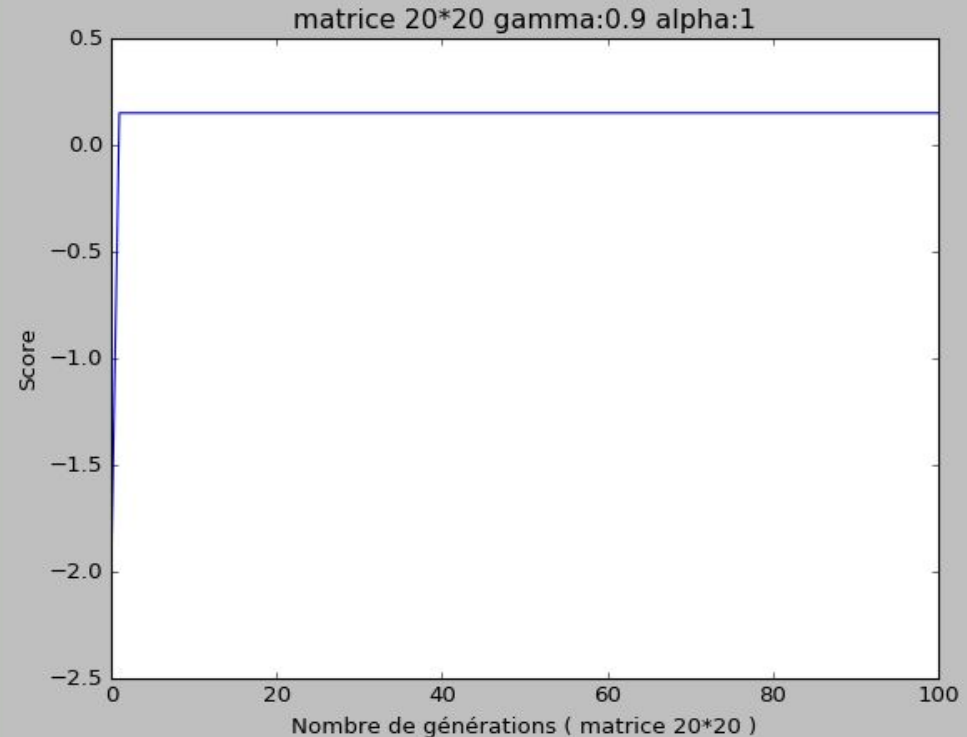
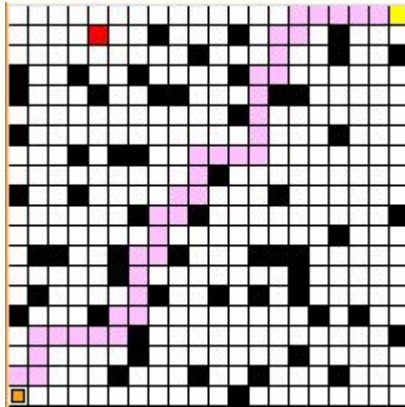


Aspect apprentissage : atout majeur du Q-learning

Enregistrement/lecture de la matrice Q
pré-optimale pour une grille donnée dans un
fichier et lancement avec cette matrice

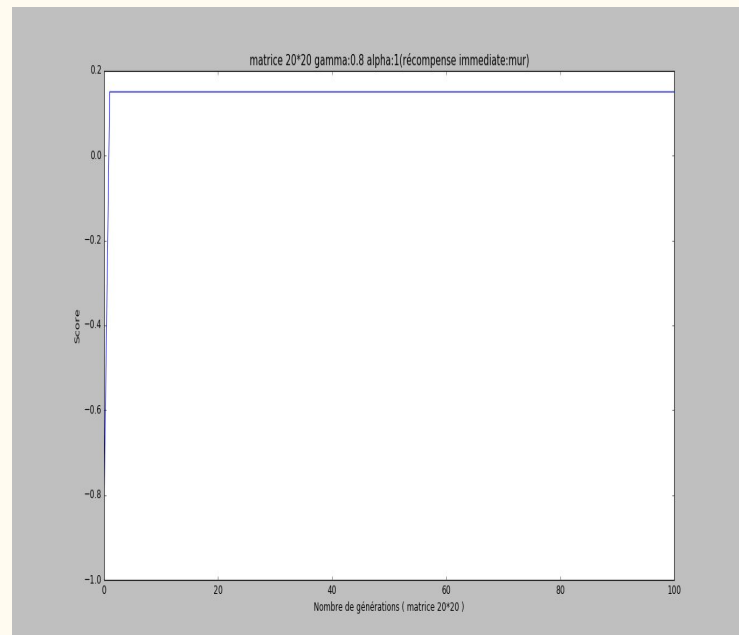
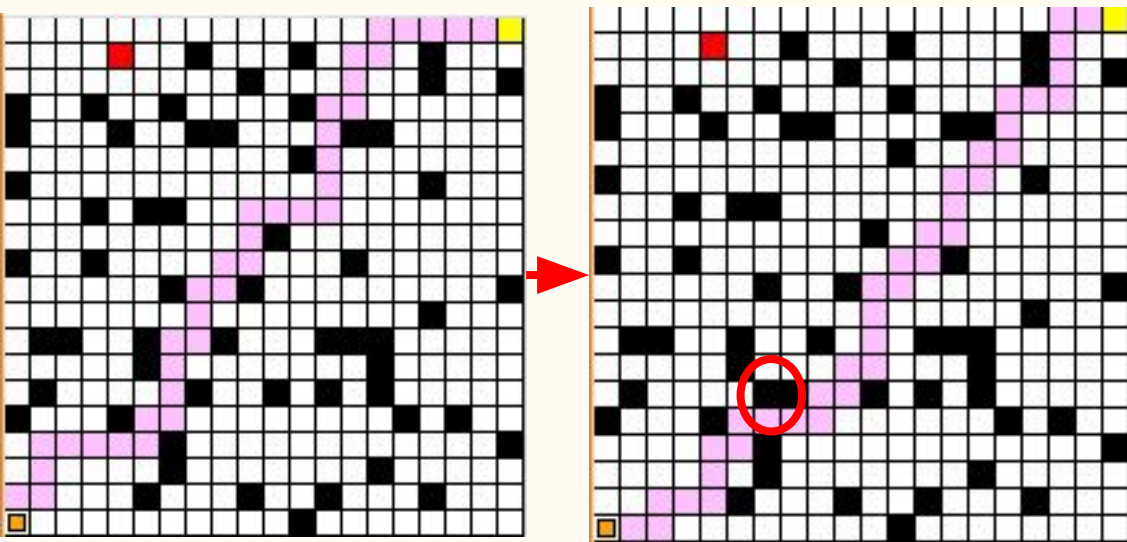
(Voir annexe *apprentissage.py* ligne 22 et
apprentissage_nogui.py ligne 185)

Chemin trouvé trivialement



Aspect apprentissage : atout majeur du Q-learning

Tolérance aux modifications? Meilleur chemin toujours instantané
(Voir *Milieu.py* ligne 47 pour la modification du chemin)



Conclusion

Intérêts :

- permet l'analyse de l'interaction entre un agent autonome et un milieu
- évolution en toute autonomie de l'agent sans aucune information sur le milieu
- apprentissage et adaptation de l'agent à des milieux similaires
- pourrait éventuellement permettre une résolution du problème du voyageur de commerce?

Inconvénients :

- lourd en calculs pour des environnements très grands
- solution : Considérer des chemins “presque idéaux”?