PROJET
REINFORCEMENT LEARNING

LACHAUSSEE QUENTIN

14-01-2022

mesea
MAINTENANCE SEA TOURS · BORDEAUX

SOMMAIRE

I – DÉVELOPPEMENT DU 1ER JEU - ENVIRONNEMENT

Le plateau de jeu étant configuré de cette manière :

Start			
Dragon Go to Start		Dragon Go to Start	
		Reward = 0	Dragon Reward = -1 Go to Start
	Dragon Go to Start		**Jail** Reward = 1 Go to Start

Je l'ai recréé sous forme de liste où 1 valeur correspond à 1 ligne :

```
env = ["SRRR",
       "DRDR",
       "RRRD",
       "RDRE"]
```

- S : correspond à « Start » (Début de partie)
- D : correspond à « Dragon » (Obstacle)
- R : correspond à « Rien » (Case vide)
- E : correspond à « End » (Fin de partie)

I – DÉVELOPPEMENT DU 1ER JEU - RÉCOMPENSE

Avec l'environnement de jeu sous forme de liste de 4 valeurs pour les 4 lignes :

```
env = ["SRRR",
       "DRDR",
       "RRRD",
       "RDRE"]
```

On va tout séparer pour avoir un plateau de jeu ou chaque valeur correspond à 1 case :

```
plateau = [char for char in "".join(env)]
```

```
['S', 'R', 'R', 'R', 'D', 'R', 'D', 'R', 'R', 'R', 'R', 'D', 'R', 'D', 'R', 'E']
```

Ensuite on va attribuer à chaque case une récompense selon sa valeur :

```
plateau = [0 if x=="S" else x for x in plateau]
plateau = [0 if x=="R" else x for x in plateau]
plateau = [-1 if x=="D" else x for x in plateau]
plateau = [1 if x=="E" else x for x in plateau]
```

```
[0, 0, 0, 0, -1, 0, -1, 0, 0, 0, 0, -1, 0, -1, 0, 1]
```

Pour rappel :

- S : correspond à « Start » (Début de partie)
- D : correspond à « Dragon » (Obstacle)
- R : correspond à « Rien » (Case vide)
- E : correspond à « End » (Fin de partie)

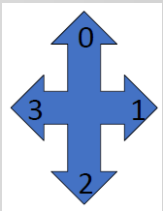
I – DÉVELOPPEMENT DU 1ER JEU – POLITIQUE D'ACTION

Pour la politique d'action, j'ai utilisé la librairie « pandas » pour me faire un DataFrame :

```python
def instanciate_Q(env: list):
    """
    Instancier Q
    va initialiser le DataFrame Q en fonction de l'environnement de jeux passé en paramètres

    :param env: environnement de jeux case par case, sous form de matrice transformé en liste
    :type env: liste
    :return Q: la politique du jeux selon la case et l'action (1 case = 1 ligne | 1 action = 1 colonne)
    :rtype Q: DataFrame
    """

    # on définit l'espace de l'environnement de jeux
    state_space = [x for x in range(len(env[0])*len(env))]
    # on lui ajoute une clé de correspondance
    state_key = [1 for x in range(len(env[0])*len(env))]
    # on crée un premier DataFrame correspondant à chaque case du jeu (son état)
    state_space_pd = pd.DataFrame({'key': state_key,'state':state_space})

    # on définit les actions possibles dans le jeux
    action_space = [x for x in range(4)]
    # on lui ajoute une clé de correspondance
    action_key = [1 for x in range(4)]
    # on crée un deuxième DataFrame correspondant à chaque action du jeu
    action_space_pd = pd.DataFrame({'key':action_key, 'action':action_space})

    # on merge les 2 DataFrame en faisant un produit cartésien entre les états et les actions
    Q = pd.merge(state_space_pd, action_space_pd, on='key')
    # on initialise toutes les valeurs à 0
    Q["q_value"] = 0

    # on retourne la politique par défaut
    return Q
```

	key	state	action	q_value
0	1	0	0	0
1	1	0	1	0
2	1	0	2	0
3	1	0	3	0
4	1	1	0	0
...	...	...	...	...

- « State » correspond à la case de l'environnement
- « Action » correspond au mouvement avec :
  - ✓ 0 : le joueur se déplace en haut
  - ✓ 1 : le joueur se déplace à droite
  - ✓ 2 : le joueur se déplace en bas
  - ✓ 3 : le joueur se déplace à gauche
- « Q_value » correspond à la politique d'action (dépendra des récompenses obtenues)



II – Q-LEARNING – CHOIX D'UNE ACTION

Pour que l'algorithme fasse le choix d'une action, j'ai utilisé la librairie « numpy » pour faire un « random.choice » (« choix aléatoire ») entre :

- **Exploration,** selon la probabilité « Epsilon » -> un choix aléatoire entre les 4 actions
- **Exploitation,** selon la probabilité « 1 - Epsilon » -> utiliser la meilleur politique d'action

```
# on choisit une action pour trouver le plus court chemin selon :
    # Au hasard avec une probabilité "epsilon"
    # La meilleure avec une probabilité "1-epsilon"
random_action = np.random.choice((0, 1), p=[epsilon, (1 - epsilon)])

# soit simplement avec une fonction random
if random_action == 0:
    chosen_action = randint(0,3)

# soit en prenant la meilleure action qui fut faite dans le passé
else:
    # on prépare le filtre de sélection en fonction de l'état actuel
    select_current_state = (Q["state"] == state)

    # on récupère la valeur max
    value_max = np.max(Q[select_current_state]["q_value"])

    # on prépare le filtre de sélection en fonction de la valeur max
    select_max_reward = (Q["q_value"] == value_max)

    # si plusieurs action donnent la valeur max, alors on en choisit une au hasard
    chosen_action = np.random.choice(Q[select_current_state & select_max_reward]["action"])
```

Avec Epsilon égale à :

$$\varepsilon = \frac{Nb\ de\ partie\ total}{Nb\ de\ partie\ total + Num\ de\ la\ partie\ en\ cours}$$

II – Q-LEARNING – RÉSULTAT DE LA NOUVELLE POLITIQUE

Pour obtenir le résultat de la nouvelle politique, j'ai utilisé cette formule :

$$Q(s_n, a_n) \leftarrow Q(s_n, a_n) + \alpha[r_n + \gamma \max_a Q(s_{n+1}, a) - Q(s_n, a_n)]$$

```
# on prépare les filtres de sélection en fonction de l'état actuel, l'état suivant et l'action choisie
select_current_state = (Q["state"] == state)
select_next_state = (Q["state"] == next_state)
select_chosen_action = (Q["action"] == chosen_action)

# on va chercher la valeur de la politique de l'etat actuel et de l'action choisie
current_Q_value = Q.loc[select_current_state & select_chosen_action, "q_value"]

# on calcul l'estimation de la politique
estimate_Q_value = reward + lambda_ * np.max(Q.loc[select_next_state, "q_value"])

# on met à jour la politique actuelle par l'estimation
new_Q_value = current_Q_value + alpha * (estimate_Q_value - current_Q_value)
Q.loc[select_current_state & select_chosen_action, "q_value"] = new_Q_value
```

La politique est désormais modifiée vers le + ou le - selon si la partie était bonne ou mauvaise.

II – Q-LEARNING – LANCEMENT DE L'APPRENTISSAGE

Pour un 1er test, j'ai initialisé les paramètres comme suit et j'ai appelé ma fonction « play » qui va lancer l'apprentissage de l'algorithme et me retourner 2 DataFrames :

```
# on spécifie les paramètres
alpha = 0.19
lambda_ = 0.96
nb_game = 1000

# on spécifie l'environnement
env = ["SRRR",
       "DRDR",
       "RRRD",
       "RDRE"]

# on lance le traitement
Q, resultats = play(env, alpha, lambda_, nb_game)
```

- **Q** : Ma politique d'action finale
- **Resultats** : Le détails de mes parties

III – AFFICHAGE DES RÉSULTATS – POLITIQUE D'ACTION

Dans « Q » on y retrouve la politique d'action finale :

	key	state	action	q_value
0	1	0	0	81.154192
1	1	0	1	85.917176
2	1	0	2	-5.000000
3	1	0	3	80.981329
4	1	1	0	81.689096
5	1	1	1	78.962773
6	1	1	2	88.686053
7	1	1	3	83.207886
8	1	2	0	74.315061
9	1	2	1	69.496643
10	1	2	2	-5.000000
11	1	2	3	80.395634
12	1	3	0	55.171184

Avec les valeurs dans l'encadré rouge, on comprends que dans la case 0, il ne faut surtout pas descendre (action 2) pour ne pas avoir une récompense négative (signifiant une défaite)

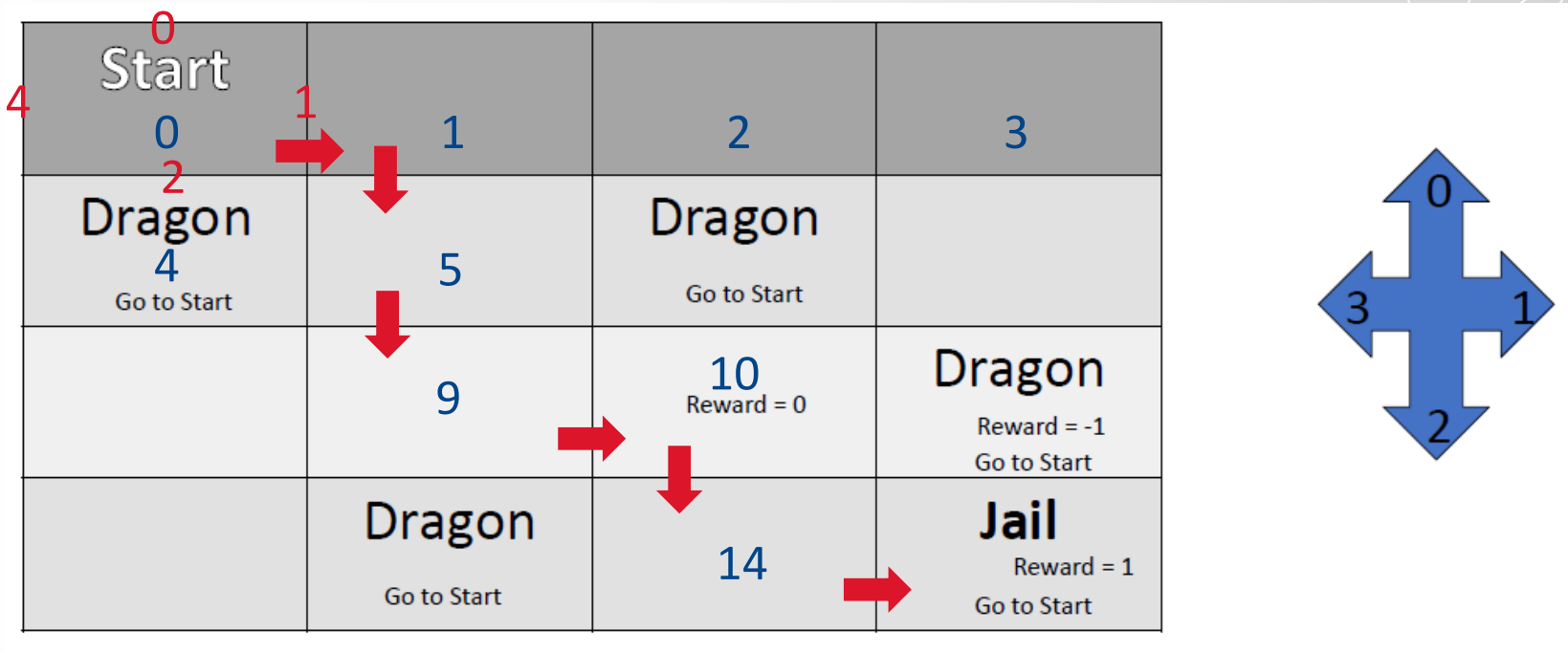
III – AFFICHAGE DES RÉSULTATS – POLITIQUE D'ACTION

On peut exécuter cette ligne de code pour avoir les meilleures action à faire selon la case :

```
In [23]: Q.iloc[Q.groupby("state").q_value.idxmax(),:]
```

Out[23]:

	key	state	action	q_value
1	1	0	1	85.917176
6	1	1	2	88.686053
11	1	2	3	80.395634
15	1	3	3	72.892735
16	1	4	0	0.000000
22	1	5	2	91.350421
24	1	6	0	0.000000
28	1	7	0	51.160797
33	1	8	1	86.923729
37	1	9	1	94.119796
42	1	10	2	96.999943
44	1	11	0	0.000000
48	1	12	0	70.670491
52	1	13	0	0.000000
57	1	14	1	100.000000
60	1	15	0	0.000000



Avec les valeurs dans les encadrés rouge, on comprends que :

Dans la **case 0**, il faut aller à **droite** -> Puis -> Dans la **case 1**, il faut aller en **bas**

Et ainsi de suite jusqu'à se retrouver dans la **case 14** et aller à **droite** pour arriver à la case finale sans encombre

III – AFFICHAGE DES RÉSULTATS – DETAILS DES PARTIES

Dans « résultats » on y retrouve le détails des déplacements des 1000 parties :

	game	step	done	state	action	next_state	reward
0	0	0	False	0	0	0	-1
1	0	1	False	0	0	0	-1
2	0	2	False	0	0	0	-1
3	0	3	True	0	2	4	-5
4	1	0	False	0	3	0	-1
...	...	...	...	...	...	...	...
5657	998	1	False	1	2	5	1
5658	998	2	False	5	2	9	1
5659	998	3	False	9	1	10	1
5660	998	4	True	10	1	11	-5
5661	999	0	True	0	2	4	-5

Avec en description de chaque déplacement :

- « state » : son état
- « action » : l'action qu'il a prévu de faire

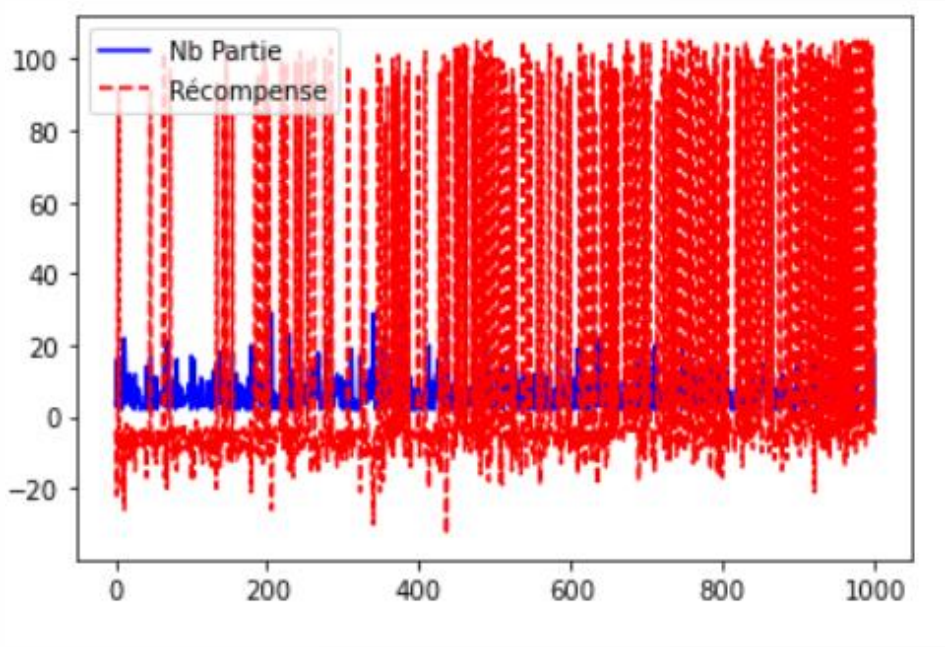
Et donc :

- « next_state » : son prochain état
- « reward » : la récompense qui en découle

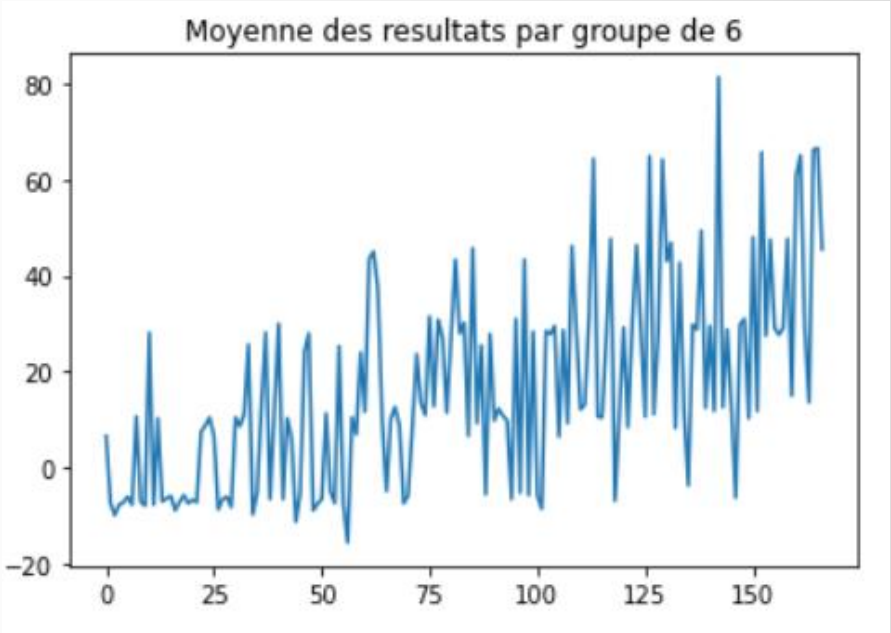
PS : « done » expose si son prochain coup amènera à une fin de partie (tomber sur un dragon ou trouver le joyau) ou non respectivement par « True » ou « False »

III – AFFICHAGE DES RÉSULTATS – COURBE

A l'aide des résultats on peut afficher des courbes pour suivre le nombre de partie réussi selon leurs récompenses :



Plus le rouge est dominant,
Plus on peut avoir confiance en la politique de décision



Plus la courbe monte,
Plus l'algorithme exploite la politique de décision qu'il considère comme efficace

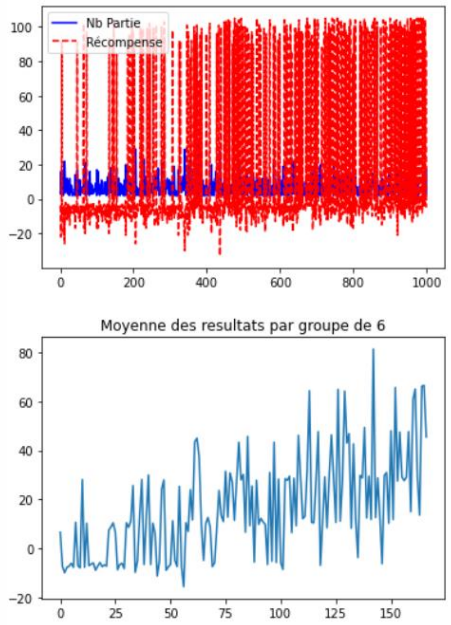
mesea
MAINTENANCE SEA TOURS · BORDEAUX

IV – GAGNER EN EFFICACITÉ – MODIFIER LES RÉCOMPENSES

Après modification des récompenses, les résultats n'étaient pas significativement différents :

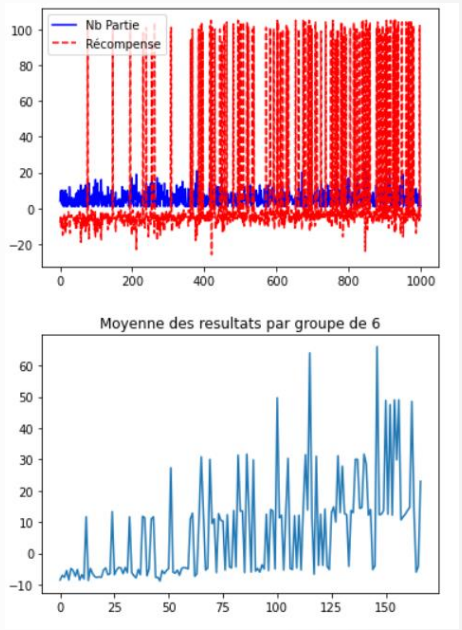
Test 1 :

- D = -1
- R = 0
- E = 100



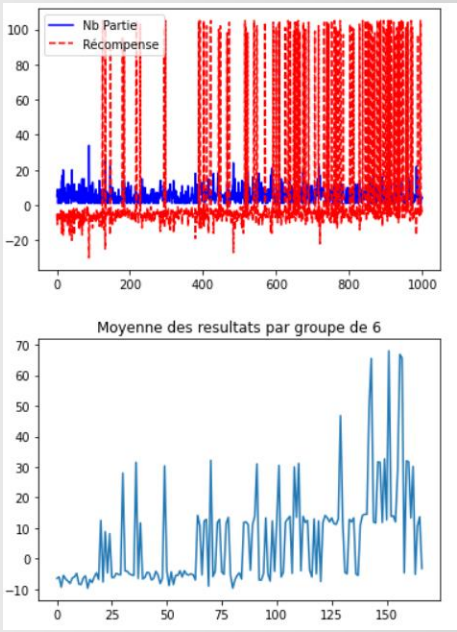
Test 2 :

- D = -5
- R = 1
- E = 100



Test 3 :

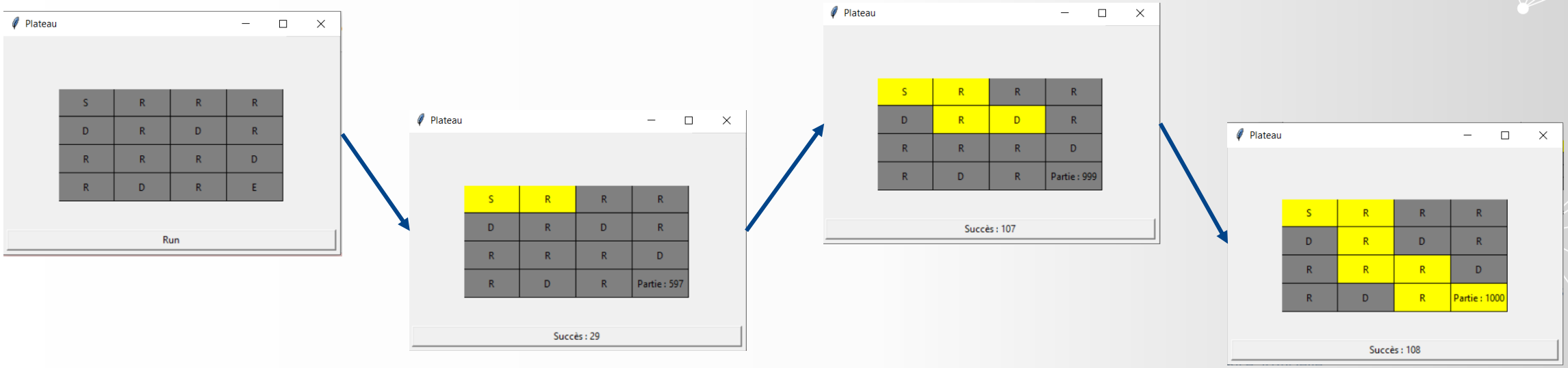
- D = -1
- R = 1
- E = 100



Les résultats me semblaient très liés à la chance lors du choix aléatoire des actions au début des parties.

IV – GAGNER EN EFFICACITÉ – SUIVRE L'APPRENTISSAGE

Pour suivre l'apprentissage, j'ai utilisé la librairie « tkinter » pour me faire une fenêtre interactive qui pourrait m'afficher toutes les parties effectuées ainsi que tous leurs déplacements :



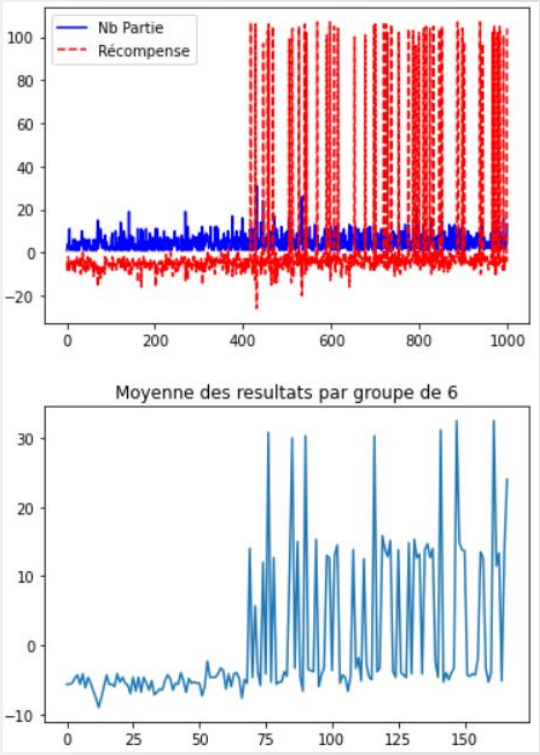
Et en suivant les parties, cela m'a **conforté** dans l'idée que les résultats me semblaient très liés à la chance lors du choix aléatoire des actions au début des parties.

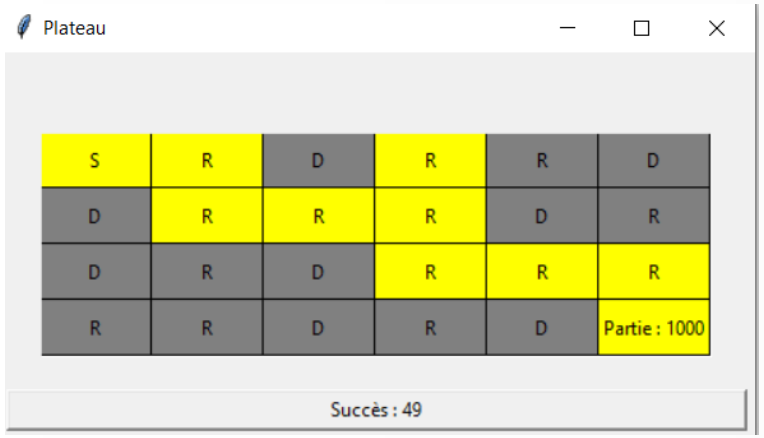
C'est pourquoi j'ai eu l'idée **d'augmenter la taille du plateau** pour mieux comprendre le comportement de notre intelligence artificielle.

V – DÉVELOPPEMENT DU 2ÈME JEU – ENVIRONNEMENT 6 PAR 4

J'ai commencer par faire un plateau de 6 colonnes pour 4 lignes :

```
# Deuxième test : 6 par 4
alpha = 0.19
lambda_ = 0.96
nb_game = 1000

env = ["SRDRRD",
       "DRRRDR",
       "DRDRRR",
       "RRDRDE"]

# on lance le traitement
Q, resultats = play(env, alpha, lambda_, nb_game, first_win=False)
resultats_by_group = show_resultats(resultats)
show_game(env, resultats)
```





Les résultats étaient **plutôt bon dès le 1er test**,
Mais avant de changer les récompenses ou les paramètres,
Je me suis dis : « pourquoi ne pas directement viser plus haut ? »…

V – DÉVELOPPEMENT DU 2ÈME JEU – ENVIRONNEMENT ALÉATOIRE

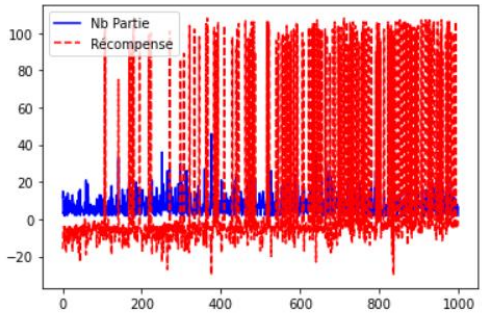
J'ai donc instauré un nouveau paramètre « nb_case » qui va me permettre de générer des plateaux de jeux carrés avec « nb_case » lignes et « nb_case » colonnes :

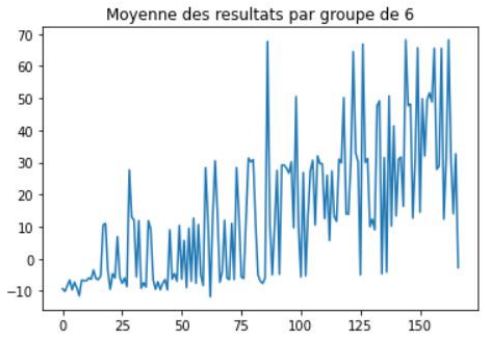
```
alpha = 0.19
lambda_ = 0.96
nb_game = 10000
nb_case = 5

env = "".join(["S"]+[np.random.choice(("R","D"), p=[0.8, 0.2]) for i in range(nb_case*nb_case-2)]+["E"])
env = [env[i:i+nb_case] for i in range(0, len(env), nb_case)]
for n in range(nb_case):
    print(env[n])
# on lance le traitement si l'environnement convient
Q, resultats = play(env, alpha, lambda_, nb_game, first_win=False)
resultats_by_group = show_resultats(resultats)
show_game(env, resultats)
```

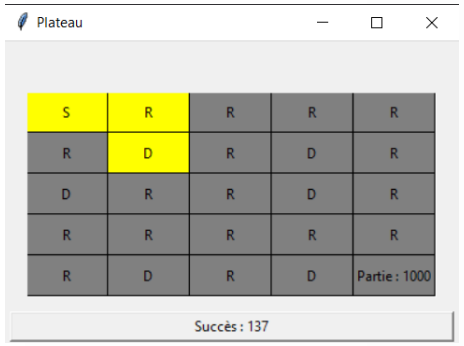
Les cases « Start » et « End » seront forcément respectivement en haut à gauche et en bas à droite.
Les autres cases seront remplies aléatoirement avec une probabilité de 80% d'être vides, et 20% d'avoir un dragon.
L'environnement est affiché avant de lancer l'apprentissage pour que l'utilisateur s'assure que le niveau est possible (le joyau est atteignable).

V – DÉVELOPPEMENT DU 2ÈME JEU – RÉSULTATS

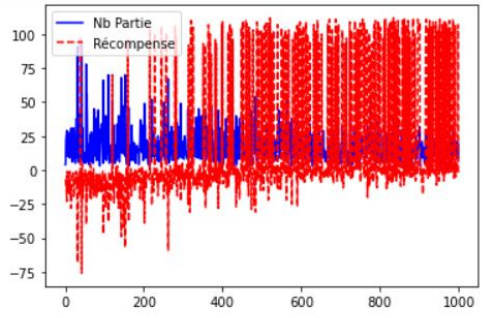
J'ai commencé par du 5x5 :

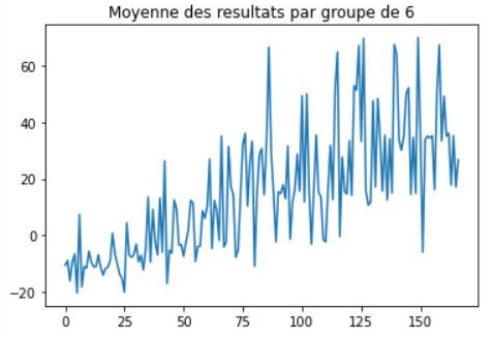


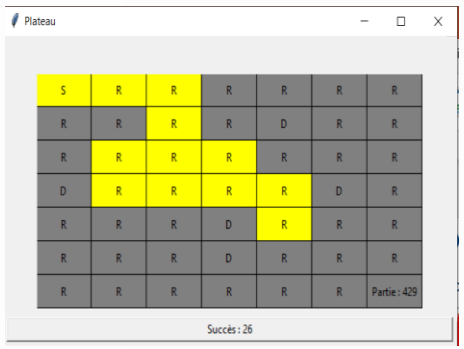




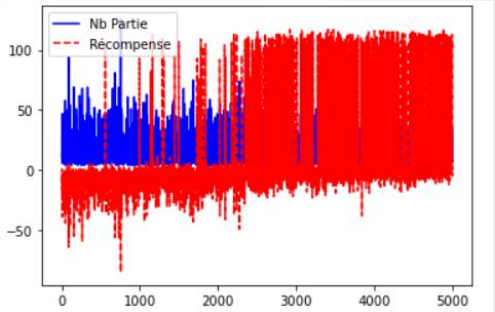
Puis du 7x7 :

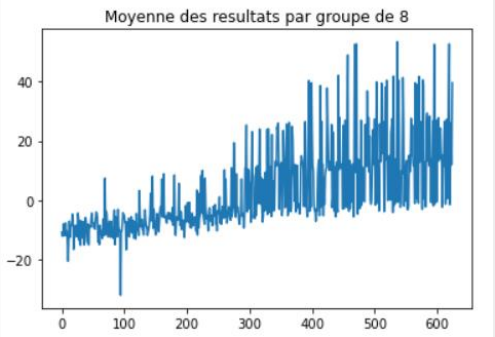


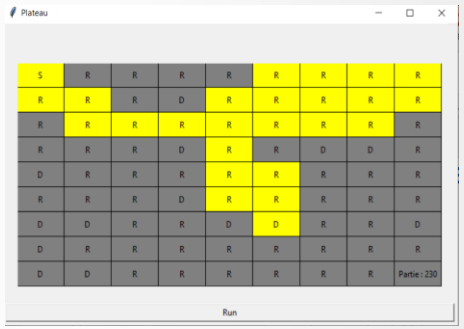




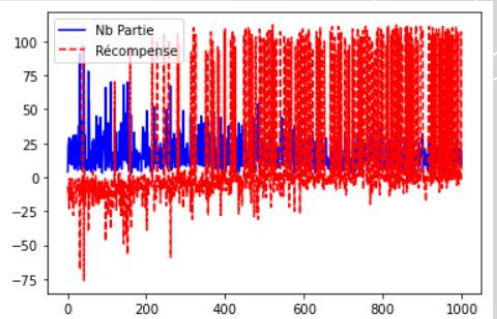
Puis du 9x9 :

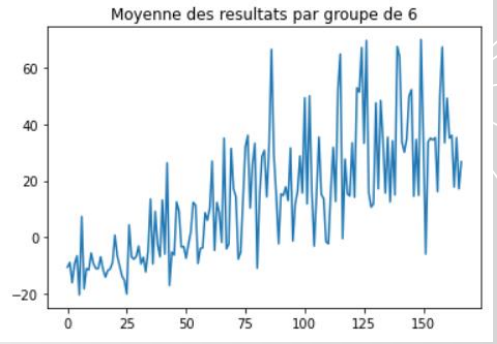


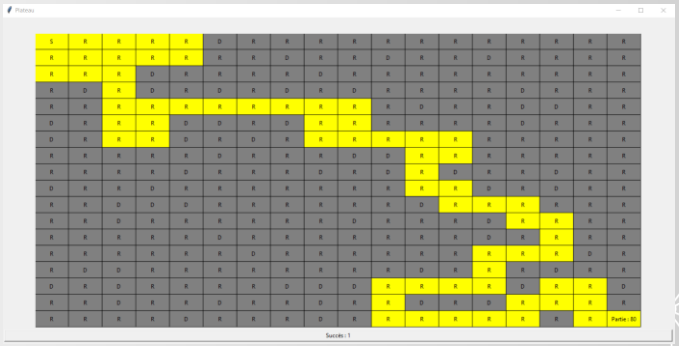




Jusqu'à du 18x18
(limite affichable par mon écran) :







mesea
MAINTENANCE SEA TOURS · BORDEAUX

VI – CONCLUSION

Pour conclure, en augmentant la surface du plateau de jeux j'ai remarqué que les meilleurs résultats apparaissaient selon 3 critères :

- **Plus la récompense de la case finale est élevée**, plus l'apprentissage va suivre le 1er chemin qui lui mène à elle quitte à abandonner l'idée d'identifier le meilleur chemin
- **Plus la récompense des cases neutres est élevé**, plus l'apprentissage va avoir tendance à se balader quitte à oublier le chemin de la victoire
- **La récompense du dragon ne doit pas être négativement plus forte que la récompense de victoire**, pour ne pas annihiler le chemin de la victoire par les nombreux dragons qui l'avoisinent

Mon choix final c'est donc porté sur ces récompenses :

```
wall_reward = -4 # sortir du plateau
action_reward = 1 # faire un déplacement neutre
dragon_reward = -5 # rencontrer un dragon
win_reward = 100 # atteindre le joyau
undo_reward = -1 # revenir sur ses pas
```

Et elles m'apportent des résultats pleinement satisfaisants

VII – DEEP Q-LEARNING

Pour être honnête je n’ai pas du tout réussis à comprendre cette partie Deep Q-Learning.
Le concept est trop compliqué pour moi.

D’autant plus que je n’ai pas réussi à trouver d’aide précises sur Internet qui aurait pu m’aiguiller sur cette partie.

Donc j’espère qu’après avoir corrigé nos devoirs, vous pourriez nous transmettre un corrigé pour que je puisse comprendre cette partie au moins sur la forme, ce serait top !

MERCI DE VOTRE ATTENTION

mesea
MAINTENANCE SEA TOURS · BORDEAUX