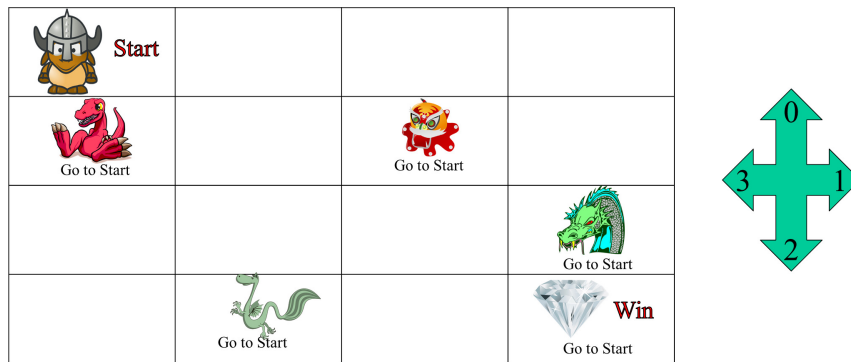


Apprentissage par renforcement

1 Développement d'un jeu

- Créer une fonction qui permet de simuler le plateau en positionnant notamment les différents éléments (case départ, fin, dragons).
 - Créer une fonction qui permet de simuler l'interaction entre l'agent et son environnement.
- Nous proposons de simuler le plateau suivant :



En complément, lorsque le joueur sort du plateau il revient dans la case où il était.

```
[position,reward,fin]=application.action(action,position,space,r);
```

Ce programme doit évaluer le retour de l'environnement en fonction d'une action de l'agent.

Les paramètres :

- La variable **action** contient l'action effectuée (0,3).
- La variable **position** contient la position de l'agent.
- La variable **space** contient l'organisation du plateau.
- La variable **r** contient les récompenses que l'on a prévue.

Le résultat :

- La variable **position** est la nouvelle position.
- La variable **Reward** est la récompense obtenue (un réel).
- La variable **fin** est un boolean indiquant si la partie est finie.

2 Développement du Q-learning

Vous allez développer la procédure permettant à un agent de mettre à jour sa politique par l'algorithme de Q-learning. Pour cela, il faut développer :

- une fonction **action = choose_action(position,epsilon,Q,Space)** qui applique la stratégie epsilon-greedy pour choisir l'action à effectuer selon la position de l'agent.

Une itération de l'algorithme reçoit la table en cours stockant le cumul des récompenses pour une action choisie pour un état particulier. Et donc pour l'état en cours contenue dans la variable **state**, elle met à jour la case associée à l'action choisie par stratégie epsilon-greedy de paramètre **epsilon**. Elle retourne alors la table mise à jour, ainsi que le nouvel état de l'agent.

- Tester votre algorithme avec des récompenses $R = 2, -2, 0, -1$ (-2 quand je tombe sur un dragon, 0 quand je tombe sur une case vide, -1 quand je sors du jeu, 2 quand je tombe le diamant) Etudier la table Q . Jouer une partie avec la politique optimale liée à la table et commenter le parcours. Nous proposons d'étudier le paramétrage suivant : $\alpha = 0.9, \gamma = 0.5$.
- Faites évoluer votre algorithme en modifiant les récompenses pour gagner en efficacité. Etudier la table Q . Jouer une partie avec la politique optimale liée à la table et commenter le parcours. Comparer avec la première politique. Nous pourrions utiliser $R = 30, -10, -1, -2$

3 Deep Q-learning

Nous souhaitons, dans le même contexte, mettre en place une stratégie Deep Q-Learning. Vous pouvez réutiliser les fonctions d'interaction avec l'environnement.

Pour simplifier, nous proposons dans un premier temps de tester l'algorithme à travers une structure simple : 2 couches denses ayant 16 entrées (nombre de cases) et 4 sorties (4 actions).

Il vous faut alors :

- modifier la fonction `action = choose_action(state,epsilon,modele)` qui va choisir dans certains cas l'action à appliquer avec `Sortie_Q = model.predict(np.array([vec_etat]))`,
- la procédure permettant de mettre à jour les poids du réseau grâce à la différentiation automatique `tf.GradientTape()`.

Pour réaliser cela, vous devez vous reporter à l'annexe vue durant le cours.

- Tester votre algorithme avec des récompenses $R = -20, -1, 100, -3$. Jouer une partie avec la politique optimale liée à la table et commenter le parcours.
- Faites évoluer votre algorithme en introduisant un second réseau.