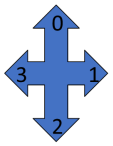
Université de Poitiers
projet 3 : Apprentissage par renforcement

2023–2024

Apprentissage par renforcement

1 Développement d'un jeu

— *Créer une fonction qui permet de simuler le plateau en positionnant notamment les différents élémenst (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);
```

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.

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 créer sa politique par l'algorithme de Q-learning. Pour cela, il faut développer :
- une fonction `action = choose_action(state,epsilon,mat_q)` qui applique la stratégie epsilon-greedy pour choisir l'action à effectuer selon l'état de l'agent (ou sa position).
- une fonction `new_q,new_state = onestep(mat_q,state,epsilon)` qui applique une itération de l'algorithme Q-learning.

La seconde fonction 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 $R = -1, 0, 1$. 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 : $\alpha = 0.81, \gamma = 0.96$.
- 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.

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