Projet le Roi des Roses©

Partie modélisation et fonctions obligatoires

Guillaume Collet

Version 1.0

1 Prérequis et consignes

Ce sujet comporte 3 fichiers pdf :
— `roi_des_roses.pdf` contient les règles du jeu ;
— `modelisation.pdf` contient la modélisation du jeu et les spécifications des différentes fonctions obligatoires ;
— `ia.pdf` contient les spécifications de la fonction `play` qui implémente votre pseudo-IA.

Organisation : Ce projet est à réaliser par groupes de deux personnes. Une personne de la promo aura la responsabilité de me renvoyer la liste des groupes et d'organiser le tableau du tournoi (car il y aura un tournoi).

Rendu : Vous avez 3 fichiers à rendre avant le 30 novembre à 23h59 (autrement dit, le 1 décembre 0h00, c'est trop tard). **Attention** : Le sujet pourra être modifié pendant la période de réalisation (corrections/améliorations/modifications) principalement pour lever des ambiguïtés ou pour préciser des notions trop floues.

Le rendu se fera sur votre compte gitlab via un "push". Tous les étudiants doivent avoir les 3 fichiers sur leur répertoire gitlab (organisez-vous). Les fichiers s'appellent `roi_des_roses.py`, `test_roi_des_roses.py` et `ia.py`. Le détail de ces fichiers est expliqué dans les fichiers pdf dédiés.

Tests : Le projet sera en partie corrigé par des tests automatisés mais vous n'y aurez pas accès. **Vous devez écrire vos propres tests** dans le fichier `test_roi_des_roses.py`. Je vous encourage vivement à partager vos tests avec les autres groupes afin de voir si vous répondez bien aux exigences demandées sur chaque fonction. Vous pouvez même vous constituer une base de tests commune pour toute la promo. Dans tous les cas, un projet rendu sans test perdra des points.

Note : La note du projet est séparée en deux parties :
- une partie "modélisation" sur **14 pts** où vous devez rédiger des fonctions qui répondent à des fonctionnalités précises ;
- une partie "IA" sur **6 pts** où vous devez créer une pseudo intelligence artificielle qui joue au jeu le Roi des Roses©.

Dans les deux parties, vous devez respecter les spécifications de fonction à la lettre. Les deux équipes arrivant en finale du tournoi auront droit à un bonus d'un point sur leur note de projet.

2 Modélisation

Dans cette partie, vous devez écrire des fonctions qui vont vous aider à modéliser le jeu le Roi des Roses©. La finalité étant d'avoir un programme python que vous exécuterez en ligne de commande, qui affichera le jeu dans le terminal et qui interagira avec les joueurs (humains).

En développement d'application, une méthode classique consiste à utiliser un système "Model-View-Controler" ou MVC. Ce système sépare le modèle (les données et leur manipulation) de la vue (l'affichage). Le modèle et la vue communiquant à travers des contrôleurs.

Dans notre cas, nous laisserons de côté les contrôleurs. La vue sera simplement un affichage dans le terminal (via des print). Le modèle se fera à travers des fonctions et des structures de données simples de Python (principalement des listes).

2.1 Le plateau

Le plateau de jeu est une matrice de 9 cases par 9 cases. Elle sera modélisée par une variable appelée `plateau` qui contiendra une liste de listes de `str`. La première dimension contiendra la ligne, la deuxième dimension la colonne. Autrement dit, `plateau[2][5]` correspond à la case ligne 2, colonne 5 (sachant que la case `plateau[0][0]` est en haut à gauche, voir figure 1).

Chaque case peut être dans un des trois états suivants :
- "est vide" modélisé par le caractère `"."` ;
- "contient un pion rouge" modélisé par le caractère `"R"` ;
- "contient un pion blanc" modélisé par le caractère `"B"`.

2.2 Les joueurs

Les joueurs sont désignés par leur couleur. Dans la suite du document, lorsqu'il y a un paramètre `couleur`, les deux seules valeurs possibles sont `Rouge` et `Blanc` (sauf pour l'affichage des cases où ce sera `R` et `B`).

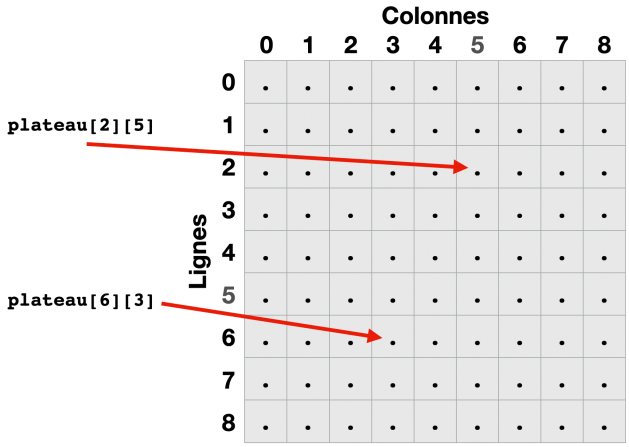

FIGURE 1 – Modélisation du plateau vide : chaque case contient le caractère `"."`.

2.3 Le roi

Le roi est un pion particulier qui se promène sur le plateau, c'est pourquoi nous ne le représentons pas directement dans le plateau avec les pions rouges et blancs. Le roi est modélisé par sa position, c'est-à-dire sa ligne `l_roi` et sa colonne `c_roi`.

2.4 Les cartes de déplacement

Les cartes de déplacement sont modélisées par des `str` contenant la direction et le nombre de cases (dans cet ordre). Ainsi, la carte `"N2"` signifie que l'on va vers le Nord de 2 cases. Nous avons donc les huit directions : `"N"`, `"NE"`, `"E"`, `"SE"`, `"S"`, `"SO"`, `"O"` et `"NO"`. Que l'on compose avec les nombres `"1"`, `"2"`, et `"3"`.

Attention : Le dernier caractère d'une carte est **toujours** le nombre de cases.

La pioche : Au début du jeu, la pioche contient les 24 cartes mélangées. On la modélise sous la forme d'une liste nommée `pioche` qui contient les 24 chaînes de caractères représentants les cartes.

Les mains des joueurs : Chaque joueur peut avoir jusqu'à 5 cartes en main, c'est ce qu'on appelle sa "main". Un joueur peut donc avoir une main vide. La main sera représentée par une liste contenant les cartes qu'il a en main. La longueur de cette liste est donc au maximum de 5 (je sais, je me répète mais ce n'est pas inutile). Les mains des joueurs seront modélisées par les variables `main_r` pour le joueur rouge et et `main_b` pour le joueur blanc.

La défausse : Lorsqu'une carte est jouée par un joueur, elle passe dans la défausse. Vous modéliserez la défausse par une liste nommée `defausse`.

Listing 1 – Exemple de modélisation de la pioche, de la défausse et des mains au début du jeu.

```
pioche = ["SO3", "E3", "O3", "NO1", "N2", "NE2", "NO2", "N1",
    "SE1", "SE3", "O1", "E2", "S3", "NE1"]
main_r = ["SE2", "SO2", "N3", "S2", "NE3"]
main_b = ["S1", "E1", "SO1", "NO3", "O2"]
defausse = []
```

Listing 2 – Exemple de modélisation de la pioche, de la défausse et des mains au cours du jeu.

```
pioche = ["SO3", "E3", "NO1", "NE2", "NO2", "SE1", "E2", "S3"]
main_r = ["SE2", "SO2", "NE3"]
main_b = ["S1", "SO1", "NO3", "O2", "NE1"]
defausse = ["S2", "N3", "SE3", "O1", "E1", "N1", "N2", "O3"]
```

Attention : Quand la pioche est vide, on tranfère le contenu de la défausse dans la pioche et on mélange.

3 Fonctions obligatoires

Dans un fichier nommé `roi_des_roses.py`, vous devez écrire les fonctions définies dans les sous-sections suivantes. Vous pouvez évidemment définir des fonctions supplémentaires dans ce fichier si besoin. Vous devrez "pusher" ce fichier dans votre répertoire de travail géré sur la plateforme gitlab (pour tous les membres du groupe).

Attention : le nom de fichier sera différent pour votre pseudo-IA. Vous ne devez donc **pas** intégrer le code de votre pseudo-IA dans ce fichier.

Vous devez également écrire vos tests dans un fichier nommé `test_roi_des_roses.py` afin de conserver le code de `roi_des_roses.py` le plus propre possible. Ce fichier est à rendre sur votre compte gitlab. Les tests seront pris en compte dans la notation.

3.1 Initialisation du jeu

Afin de lancer le jeu, nous avons besoin d'initialiser les différentes structures de données : le plateau, les mains des joueurs, la pioche et la défausse.

Consigne : Vous devez écrire la fonction `init_jeu` :

```
    def init_jeu(nb_lignes, nb_colonnes):
```

En sortie, la fonction doit renvoyer un tuple constitué, dans l'ordre, des variables :
— `plateau` : une liste contenant `nb_lignes` listes contenant `nb_colonnes "."`
— `l_roi` : la ligne où est situé le roi (au milieu du plateau)
— `c_roi` : la colonne où est situé le roi(au milieu du plateau)
— `main_r` : la liste des 5 cartes du joueur rouge (tirées au hasard)
— `main_b` : la liste des 5 cartes du joueur blanc (tirées au hasard)
— `pioche` : la liste des cartes restantes mélangées
— `defausse` : une liste vide

La fonction `init_jeu` s'utilisera comme dans la ligne suivante :

```
plt, lr, cr, mr, mb, pioche, defausse = init_jeu(9, 9)
```

3.2 Affichage de l'état du jeu

Dans tout jeu vidéo, il faut afficher "l'état du jeu" après chaque coup. Comme dans le jeu de société, les cartes des joueurs sont visibles, ainsi que le plateau et le roi (mais pas la pioche). Dans votre affichage, vous allez présenter ces 4 informations. Tout l'affichage sera textuel dans le terminal, à l'aide de la fonction `print`. Attention, vous avez deux fonctions à écrire dans cette section.

Vous commencerez par afficher les mains des joueurs **exactement** comme dans les lignes suivantes.

Listing 3 – Exemple d'affichage des mains des joueurs

```
Rouge : SE2 SO2 NE3
Blanc : S1 SO1 NO3 O2 NE1
```

Consigne : Vous devez écrire la fonction `afficher_main` :

```
    def afficher_main(couleur, main):
```

Cette fonction affiche **une seule main à la fois**. Dans l'exemple précédent, nous avons donc appelé deux fois la fonction `afficher_main` (ce qu'il faudra également faire dans votre fonction `afficher_jeu`).

Listing 4 – Exemple d'appels à la fonction `afficher_main`

```
>>> afficher_main("Rouge", ["SE2", "SO2", "NE3"])
Rouge : SE2 SO2 NE3
>>> afficher_main("Blanc", ["S1", "SO1", "NO3", "O2", "NE1"])
Blanc : S1 SO1 NO3 O2 NE1
```

Consigne : Vous devez écrire la fonction `afficher_jeu` :

```
def afficher_jeu(plateau, l_roi, c_roi, main_r, main_b):
```

À la suite des mains des joueurs, vous afficherez les cases du plateau comme dans l'exemple suivant.

Listing 5 – Exemple d'affichage complet en début de partie

```
Rouge : SE2 SO2 N3 S2 NE3
Blanc : S1 E1 SO1 NO3 O2 N3
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    | X  |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
```

Le contenu de chaque case est représenté à l'écran par une chaîne de caractères contenant exactement deux caractères.

- Le premier caractère pour la présence du roi est au choix :
  - `"X"`, le roi est sur la case ;
  - `" "`, le roi n'est pas sur la case.
- Le deuxième caractère pour la présence d'un pion est au choix :
  - `"B"`, un pion blanc est sur la case ;
  - `"R"`, un pion rouge est sur la case ;
  - `" "`, la case est vide.

Pour chaque case, vous ajouterez un espace à gauche et à droite (il y a donc 4 caractères par case : un espace, le roi ou un espace, un pion ou un espace, un espace). Chaque case est séparée des autres par le caractère `"|"` horizontalement et par des caractères `"-"` verticalement. Votre affichage doit être **exactement** comme celui des exemples.

Listing 6 – Exemple d'affichage complet en cours de partie

```
Rouge : SE2 SO2 NE3
Blanc : S1 NO3
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |  B |    |  R |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |  R |    |    |    |    |    |    |
-------------------------------------------------
|  R | XB |  B |    |  R |  B |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
```

3.3 Mouvement du roi possible

Lorsqu'un joueur souhaite bouger le roi, il faut vérifier que le mouvement imposé par la carte jouée est possible. Un mouvement est impossible si le roi arrive en dehors du plateau ou sur un pion rouge ou blanc déjà joué.

Consigne : Vous devez écrire la fonction `mouvement_possible` :

```
    def mouvement_possible(plateau, l_roi, c_roi, carte):
```

La fonction `mouvement_possible` doit renvoyer `True` si le mouvement de la carte est possible, `False` sinon.

3.4 Main jouable

Une autre fonction très utile est de savoir, parmi les cartes d'une main, lesquelles sont jouables, c-à-d celles pour lesquelles le mouvement du roi est possible.

Consigne : Vous devez écrire la fonction `main_jouable` :

```
    def main_jouable(plateau, l_roi, c_roi, main):
```

La fonction `main_jouable` doit renvoyer la liste des cartes de la `main` pour lesquelles le mouvement du roi est possible. La liste est vide si aucun mouvement est possible.

3.5 Demande une action à un joueur

Quand le tour d'un joueur est venu, votre jeu doit lui demander quelle action il souhaite faire :

```
Vous êtes le joueur Rouge, que souhaitez-vous faire ?
```

Consigne : Vous devez écrire la fonction `demande_action` :

```
    def demande_action(couleur, plateau, l_roi, c_roi, main):
```

La fonction `demande_action` doit afficher le texte suivant :

```
Vous êtes le joueur [couleur], que souhaitez-vous faire ? "
```

En remplaçant `[couleur]` par la couleur donnée en paramètre.

Les réponses possibles du joueur sont :
- une carte de sa main ;
- `"pioche"` s'il souhaite piocher une carte ;
- `"passe"` s'il ne peut pas faire les deux actions précédentes.

Attention : votre fonction doit vérifier que la réponse est correcte, c'est-à-dire vérifier que :
- si carte jouée, elle existe dans la main du joueur et le mouvement du roi est possible ;
- si pioche, la main contient moins de 5 cartes ;
- si passe, les deux autres actions sont impossibles.

Tant que la réponse n'est pas correcte, votre fonction doit afficher le message `"Action impossible, que souhaitez-vous faire ? "` et attendre la réponse du joueur. On ne s'embête pas à expliquer au joueur ce qu'il doit faire.

Listing 7 – Exemple de demandes d'action

```
Rouge : SE2 SO2 NE3
Blanc : S1 NO3
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |    |  B |    |  R |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |  R |    |    |    |    |    |    |
-----------------------------------------------
|  R | XB |  B |    |  R |  B |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
Vous êtes le joueur Rouge, que souhaitez-vous faire ? N1
Action impossible, que souhaitez-vous faire ? SO2
Action impossible, que souhaitez-vous faire ? passe
Action impossible, que souhaitez-vous faire ? NE3
Rouge : SE2 SO2
Blanc : S1 NO3
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |    |  B | XR |  R |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |  R |    |    |    |    |    |    |
-----------------------------------------------
|  R |  B |  B |    |  R |  B |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
|    |    |    |    |    |    |    |    |    |
-----------------------------------------------
Vous êtes le joueur Blanc, que souhaitez-vous faire ?
```

3.6 Bouge le roi

Si un joueur joue une carte, nous sommes maintenant sûr que le mouvement du roi est possible. Il faut donc faire bouger le roi et changer l'état du jeu.

Consigne : Vous devez écrire la fonction `bouge_le_roi` :

```
def bouge_le_roi(plateau, l_roi, c_roi, main_r, main_b,
    defausse, carte, couleur):
```

Dans cette fonction, il y a beaucoup de choses à faire :
— calculer les nouvelles valeurs de `l_roi` et `c_roi` en fonction de la `carte` jouée ;
— placer un pion de la bonne couleur et au bon endroit sur le plateau ;
— retirer la carte jouée de la main du joueur ;
— ajouter la carte jouée dans la défausse.

La fonction `bouge_le_roi` doit renvoyer un tuple contenant, dans l'ordre, les versions mises à jour (si nécessaire) des variables `plateau`, `l_roi`, `c_roi`, `main_r`, `main_b`, et `defausse`.

3.7 Définition des territoires

C'est la fin de partie, il faut maintenant définir les territoires des joueurs afin de compter les points. Pour cela nous avons besoin d'une fonction qui, étant donnée une case de départ contenant un pion d'une couleur, nous renvoie la liste des cases faisant partie du même territoire.

Consigne : Vous devez écrire la fonction `territoire` :

```
def territoire(plateau, ligne, colonne, couleur):
```

Nous partons de la case à la position (`ligne`, `colonne`).
— si cette case ne contient pas de pion de la bonne `couleur`, la fonction renvoie une liste vide.
— si cette case ne contient un pion de la bonne `couleur`, la fonction renvoie une liste contenant au moins la position de la case de départ (`[(ligne, colonne)]`) ainsi que les positions des cases faisant partie du même territoire (l'ordre des cases n'est pas important).

Listing 8 – Plateau de fin de partie

```
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |  B | XR |  R |  R |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |  R |    |    |    |    |    |    |
-------------------------------------------------
|  R |  B |  B |    |  R |  B |  R |    |    |
-------------------------------------------------
|    |    |  B |  B |  B |    |    |    |    |
-------------------------------------------------
|    |    |  B |    |    |    |    |    |    |
-------------------------------------------------
```

Voici des exemples d'appel à la fonction `territoire`, avec la variable `plateau` contenant le plateau précédent (listing 8).

```
>>> territoire(plateau, 3, 4, "Rouge")
[(3, 4), (3, 5), (3, 6)]
>>> territoire(plateau, 3, 3, "Rouge")
[]
>>> territoire(plateau, 3, 3, "Blanc")
[(3,3)]
>>> territoire(plateau, 7, 2, "Rouge")
[]
>>> territoire(plateau, 7, 2, "Blanc")
[(7, 2), (6, 1), (6, 2), (8, 2), (7, 3), (7, 4)]
```

3.8 Scores

Puisque nous pouvons déterminer les territoires, il est maintenant temps de calculer les scores des joueurs.

Consigne : Vous devez écrire la fonction `score` :

```
    def score(plateau, couleur):
```

La fonction doit renvoyer un entier correspondant à la somme des scores des territoires du joueur de la couleur donnée. Sur le plateau 8, voici un exemple d'appel à la fonction `score`.

Listing 9 – Exemple d'appels à la fonction `score`

```
>>> score(plateau, "Rouge")
13
>>> score(plateau, "Blanc")
38
```

3.9 Boucle de jeu principale

Maintenant que vous êtes équipés de toutes ces fonctions, vous pouvez définir une boucle de jeu principale. Cette boucle de jeu doit être dans une fonction main qui sera appelée que si on lance votre programme en ligne de commande. Vous devez donc intégrer les deux lignes suivantes à la fin de votre fichier `roi_des_roses.py`.

```
if __name__ == "__main__":
    main()
```

Consigne : Vous devez écrire la fonction `main` :

```
    def main():
```

Dans cette fonction, vous devez (entre autres choses) :
- initialiser les structures de données (plateau, mains, etc) ;
- lancer la boucle de jeu dans laquelle :
  - on affiche l'état du jeu
  - on demande l'action du joueur actif
  - on applique l'action sur l'état du jeu
  - on vérifie si le jeu est terminé ou pas
  - si oui, on arrête, si non on change de joueur actif et on boucle
- si la partie est finie, on calcule le score et on les affiche.

Lorsque la partie est terminée, vous devez afficher les scores des joueurs de la façon suivante.

```
Rouge 83
Blanc 54
Rouge a gagné la partie
```

Si les deux joueurs sont à égalité, vous afficherez :

```
Rouge 64
Blanc 64
Égalité
```