# Autres jeux

marc-michel dot corsini at u-bordeaux dot fr

19 février 2021

Pour tester vos joueurs, d'autres jeux sont fournis que le « Puissance 4 ». Leur intérêt réside dans une résolution plus rapide et vont pouvoir servir de banc d'essai aux différents joueurs mis en place au cours du jalon 02. Chaque jeu est disponible dans un fichier annexe, nous allons, dans un premier temps voir comment ils peuvent être utilisés, avant de les étudier succinctement, un par un.
**Remarque** La classe `Human` peut ne pas foncionner avec ces jeux, si vous avez fait une version spécifique au « Puissance 4 » ce n'est pas grave. Par contre **tous** les joueurs artificiels doivent fonctionner.

## 1 Tester vos joueurs à l'aide d'un jeu quelconque

Dans la suite de cette section nous supposerons que le jeu auquel nous souhaitons jouer se trouve dans le fichier `game.py` – bien entendu, ce fichier est fictif, il faudra utiliser l'un des fichiers décrits ultérieurement.
Ce fichier contient la classe `Game` – là encore c'est un nom fictif.
Il suffit de lancer un shell sur `main_parties` qui demande le nom du fichier du jeu

```
quel est le fichier de description du jeu ?
```

Il suffit de donner en réponse `game.py`. Une fois le fichier lu vous allez dans un premier temps créer le jeu en appelant le constructeur de la classe. Puis créer les joueurs que vous souhaitez tester par exemple un joueur `Randy` et un joueur `MinMax`. Puis vous les opposer soit sur une manche, soit sur une partie de plusieurs manches.

```
>>> jeu = c4.Game(...) # il faut mettre les bons paramètres
>>> a = Randy('aleatoire', jeu)
>>> b = MinMax('max', jeu, pf=3)
>>> manche(a, b, jeu)
# or
>>> s = partie(a, b, jeu, 4)
>>> s.statistics
```

## 2 Les différents jeux

Plusieurs fichiers de jeux sont disponibles dans cette version du projet

1. allumettes.py
2. dice.py
3. divide_left.py
4. divide.py
5. marienbad.py

Nous allons décrire, pour chaque fichier, l'objectif du jeu, le nom de la classe et les paramètres à spécifier, l'ordre de présentation n'est pas l'ordre alphabétique.

### 2.1 allumettes.py

Il s'agit donc d'un jeu à 2 joueurs, comportant un certain nombre d'allumettes. À son tour chaque joueur peut retirer de 1 à 3 allumettes. L'objectif peut être, au choix, pour gagner de « ne pas prendre » la dernière allumette, ou bien « prendre » la dernière allumette.
La classe s'appelle `Matches`, le constructeur prend deux paramètres

1. Le nombre d'allumettes au départ (au moins 1, au plus 25)
2. Un booléen, la valeur `True` signifie que pour gagner, l'objectif est de « ne pas prendre » la dernière allumette.

## 2.2 marienbad.py

Ce jeu à deux joueurs est une extension du jeu précédent. La première différence réside dans le fait qu'il y a 3 groupes d'allumettes. La seconde est dans la règle de prise, deux choix sont offerts.

1. La prise, sur une ligne, peut être de 1 à 3 allumettes
2. La prise, sur une ligne, peut-être de 1 à toutes les allumettes

Comme pour le jeu des allumettes, on a à disposition deux objectifs – prendre ou ne pas prendre la dernière allumette.
La classe s'appelle `Marienbad` le constructeur prend quatre paramètres, les deux derniers sont optionnels – ils ont une valeur par défaut.

1. Le nombre d'allumettes 'N' de référence compris entre 5 et 13
2. Un entier qui est l'une des variantes proposées pour la répartition
   - 0 : la répartition sera $N, 2N, 3N$
   - 1 : la répartition sera $N, N < k \leq 2N, 2N < k' \leq 3N$
   - 2 : la répartition sera $\frac{N}{4}, \frac{N}{2}, N$
3. Un booléen `prise=True` pour prendre à chaque tour de 1 à 3 allumettes sur une ligne.
4. Un booléen `prendre=True` prendre la dernière allumette pour gagner.

## 2.3 dice.py

Il s'agit ici, dun jeu constitué d'un compteur et d'un dé à 6 faces classiques dont la face supérieure est visible. À chaque tour de jeu, le joueur qui a le trait fait pivoter le dé d'un quart de tour pour amener une autre face du dé visible. Se faisant, le compteur est décrémenté des points de la face visible. Le perdant est celui qui, par son action obligée, amène le compteur à une valeur strictement négative.
La classe s'appelle `Dice` et prend 2 paramètres dont un est optionnel

1. La valeur initiale du compteur entre 7 et 100
2. `random_state=42` qui permet d'initialiser aléatoirement la première face visible.

## 2.4 divide.py, divide_left.py

Pour ce jeu, deux versions ont été écrites `divide.py` est la version « classique », `divide_left.py` est une version qui minimise le nombre de coups jouables. La description suivante porte sur la version classique.
Deux boîtes contiennent des jetons en nombre quelconque. À son tour de jeu, un joueur choisit d'écarter une boîte et de répartir les jetons de l'autre boîte dans les 2 boîtes de telle sorte que chaque boîte contienne au moins 1 jeton. Si, à son tour de jeu, un joueur ne peut pas respecter la règle (vider, répartir), il est déclaré perdant.
La classe s'appelle `Divide` et prend 2 paramètres qui sont les pions contenus dans chaque boîte au départ du jeu.

# 3 Mise en application

Voici les résultats obtenus pour une partie de 4 manches entre un joueur `Randy` et un joueur `MinMax` qui anticipe 5 coups en avance

```
>>> jeu = c4.Matches(7, True)
matches
>>> a = Randy('alea', jeu)
matches
>>> b = MinMax('mm', jeu, pf=5)
matches
>>> s = partie(a, b, jeu, 4)
...
final Board

Il y a 00 allumettes. Retirez de 1 à 3 allumettes.
Pour gagner vous ne devez pas prendre la dernière allumette

Coup(s) joué(s) = 4, trait au joueur 1

waiting ... Done
```

```
>>> s.subkeys
('alea_01', 'mm_02', 0, 1)
>>> s.specific_statistic('mm_02')
{'pv': 4, 'sigma': 14, 'avg_victories': 1.0, 'avg_stones': 3.5}

>>> jeu = c4.Matches(7, False)
matches
>>> a = Randy('alea', jeu)
matches
>>> b = MinMax('mm', jeu, pf=5)
matches
>>> s = partie(a, b, jeu, 4)
...
final Board

Il y a 00 allumettes. Retirez de 1 à 3 allumettes.
Pour gagner vous devez prendre la dernière allumette

Coup(s) joué(s) = 3, trait au joueur 2

waiting ... Done
>>> s.subkeys
('alea_03', 'mm_04', 0, 1)
>>> s.specific_statistic('mm_04')
{'pv': 4, 'sigma': 14, 'avg_victories': 1.0, 'avg_stones': 3.5}

>>> jeu = c4.Marienbad(7,2)
marienbad
>>> print(jeu)

Vous avez devant vous 3 groupes d'allumettes
A votre tour de jeu vous devez choisir 1 groupe
et, dans ce groupe, vous enlevez 1 à 3 allumettes

Pour gagner vous devez prendre la dernière allumette de la dernière boîte

< 1, 3, 7 >
Coup(s) joué(s) = 0, trait au joueur 1

>>> a = Randy('x', jeu)
marienbad
>>> b = MinMax('y', jeu, pf=3)
marienbad
>>> s = partie(a, b, jeu, 4)
...
final Board

Vous avez devant vous 3 groupes d'allumettes
A votre tour de jeu vous devez choisir 1 groupe
et, dans ce groupe, vous enlevez 1 à 3 allumettes

Pour gagner vous devez prendre la dernière allumette de la dernière boîte

< 0, 0, 0 >
Coup(s) joué(s) = 7, trait au joueur 2

waiting ... Done
>>> s.subkeys
('x_02', 'y_03', 0, 1)
>>> s.specific_statistic('y_03')
```

```
{'pv': 4, 'sigma': 26, 'avg_victories': 1.0, 'avg_stones': 6.5}

>>> jeu = c4.Dice(13)
dice
>>> a = Randy('alea', jeu)
dice
>>> b = MinMax('mm', jeu, pf=3)
dice
>>> s = partie(a, b, jeu, 4)
A: le compteur à ne pas dépasser
B: la face visible du dé
Tournez le dé d'1/4 de tour: si après coup, A - B < 0 vous avez perdu

< 13, 6 >
Coup(s) joué(s) = 0, trait au joueur 1
...
final Board

A: le compteur à ne pas dépasser
B: la face visible du dé
Tournez le dé d'1/4 de tour: si après coup, A - B < 0 vous avez perdu

< -3, 3 >
Coup(s) joué(s) = 8, trait au joueur 1

waiting ... Done
>>> s.subkeys
('alea_01', 'mm_02', 0, 1)
>>> s.specific_statistic('mm_02')
{'pv': 4, 'sigma': 30, 'avg_victories': 1.0, 'avg_stones': 7.5}

>>> jeu = c4.Divide(9, 4)
divide
>>> a = Randy('randy', jeu)
divide
>>> b = MinMax('mm', jeu, pf=3)
divide
>>> s = partie(a, b, jeu, 4)
...

A & B : 2 boites contenant des jetons

Choisissez une boite qui sera vidée
Répartissez les jetons de l'autre boite dans A & B
- il faut au moins 1 jeton par boite

Le perdant est celui qui ne peut pas respecter les règles

< 4, 9 >
Coup(s) joué(s) = 0, trait au joueur 1

best is ('B', 1), with value 100
best is ('A', 1), with value 100
final Board

A & B : 2 boites contenant des jetons

Choisissez une boite qui sera vidée
Répartissez les jetons de l'autre boite dans A & B
- il faut au moins 1 jeton par boite
```

```
Le perdant est celui qui ne peut pas respecter les règles

< 1, 1 >
Coup(s) joué(s) = 3, trait au joueur 2

waiting ... Done
>>> s.subkeys
('randy_01', 'mm_02', 0, 1)
>>> s.specific_statistic('mm_02')
{'pv': 4, 'sigma': 14, 'avg_victories': 1.0, 'avg_stones': 3.5}
```