SUPINFO Academic Dept.

SUPINFO International University

INSTITUTE OF INFORMATION TECHNOLOGY

Algorithmique et programmation en Python

Premier Mini - Projet

Deux petits jeux de stratégie

Version 1.0
Last update: 24/10/2018
Use: Students/Staff
Author: Laurent GODEFROY

SOMMAIRE

1 PREAMBULE

Cet examen est **individuel**.

Toute forme de plagiat ou utilisation de codes disponibles sur internet ou tout autre support, même de manière partielle, est strictement interdite et se verra sanctionnée d'un 0, d'une mention « cheater », et le cas échéant d'un conseil de discipline.

Vous devrez effectuer votre rendu sous la forme d'une archive au format « .zip » contenant les codes sources de votre projet.

Ce mini-projet ne donnera pas lieu à des soutenances.

Un barème **indicatif** vous est donné dans la dernière partie de ce sujet.

Le but de ce mini-projet est de programmer en Python deux petits jeux de stratégie combinatoire abstrait.

2 NIMBLE

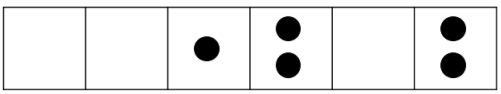
2.1 LES REGLES DE CE JEU

Remarque importante : aucun code n'est demandé dans cette sous-partie où l'on se contente d'expliquer les règles.

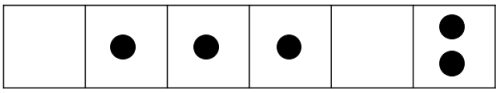
Deux joueurs s'affrontent sur un plateau unidimensionnel de n cases. Initialement chaque case contient un nombre aléatoire (pouvant être nul) de pions. A tour de rôle les joueurs choisissent une case contenant au moins un pion, et déplace l'un des pions y figurant vers une case située plus à gauche, *i.e.* une case d'indice strictement inférieur. Cette case peut être vide ou déjà contenir un ou plusieurs pions.

Le gagnant est le dernier joueur à pouvoir déplacer un pion. Autrement dit, dès qu'un joueur ne peut plus déplacer de pion il a perdu. Cela correspond bien sûr à une situation où tous les pions du plateau sont sur la première case.

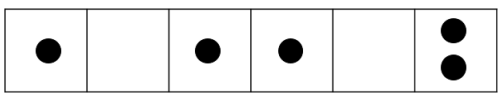
Donnons un exemple de partie sur un plateau de 6 cases :



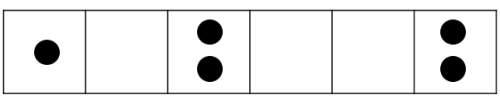
Le premier joueur bouge un pion de la case n°4 vers la case n°2 :



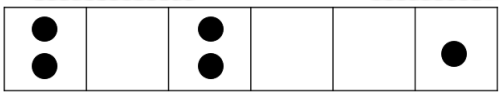
Le second joueur bouge un pion de la case n°2 vers la case n°1 :



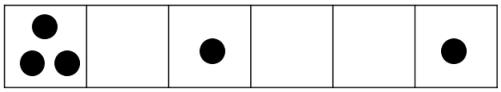
Le premier joueur bouge un pion de la case n°4 vers la case n°3 :



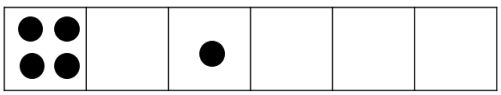
Le second joueur bouge un pion de la case n°6 vers la case n°1 :



Le premier joueur bouge un pion de la case n°3 vers la case n°1 :



Le second joueur bouge un pion de la case n°6 vers la case n°1 :



Le premier joueur bouge un pion de la case n°3 vers la case n°1 :

Le second joueur ne peut plus déplacer de pion, il a perdu.

2.2 PROGRAMMATION DE CE JEU EN PYTHON

Il vous est fortement recommandé de lire l'intégralité de cette partie avant de commencer à coder.

Remarques importantes :

- On pourra éventuellement implémenter des sous-programmes en plus de ceux demandés.
- Le plateau sera naturellement une liste à une dimension d'entiers positifs ou nuls, chacun d'eux indiquant le nombre de pions présents sur la case correspondante.
- Plutôt que de recalculer régulièrement la dimension de cette liste on préfèrera la passer en paramètre de nos sous-programmes.
- L'exemple de rendu visuel du programme n'est qu'indicatif, vous pouvez l'améliorer.

Notations des paramètres des sous-programmes que l'on va implémenter :

- « board » : liste à une dimension d'entiers positifs ou nul représentant le plateau de jeu.
- « n » : entier strictement positif égal au nombre d'éléments de « board ».

- « p » : entier strictement positif égal au nombre maximal de pions par case lors de l'initialisation du plateau.
- « i » : entier quelconque.
- « j » : entier quelconque.

Implémenter les sous-programmes suivants dans un fichier "nimble.py" :

- Une fonction « newBoard(n,p) » qui retourne une liste à une dimension représentant l'état initial d'un plateau de jeu possédant **n** cases. Sur chaque case sera positionné un nombre aléatoire de pions compris (au sens large) entre **0** et **p**.
- Une procédure « display(board,n) » qui réalise l'affichage sur la console du plateau. On représentera chaque case par le nombre de pions qu'elle contient, et on indiquera en-dessous son numéro. Exemple :

```
 0 |  3 |  0 |  2 |  3 |  1 |
---------------------------
 1 |  2 |  3 |  4 |  5 |  6 |
```

- Une fonction « possibleSquare(board,n,i) » qui retourne **True** si **i** est le numéro d'une case contenant un pion déplaçable, et **False** sinon.
- Une fonction « selectSquare(board,n) » qui fait saisir le numéro d'une case contenant un pion déplaçable. On supposera qu'il existe une telle case, on ne testera pas ce fait ici. Tant que ce numéro ne sera pas valide en regard des règles du jeu et de la dimension du plateau, on demandera de nouveau de le saisir. Finalement, la fonction retournera ce numéro.
- Une fonction « possibleDestination(board,n,i,j) » où l'on suppose ici que **i** est le numéro d'une case contenant un pion déplaçable. Cette fonction retourne **True** si **j** est le numéro d'une case vers laquelle le pion de numéro **i** peut se déplacer, et **False** sinon.
- Une fonction « selectDestination(board,n,i) » où l'on suppose ici que **i** est le numéro d'une case contenant un pion déplaçable. Cette fonction fait saisir le numéro d'une case vers laquelle le pion de coordonnées **i** peut se déplacer. On supposera qu'il existe un telle case, on ne testera pas ce fait ici. Tant que ce numéro ne sera pas valide en regard des règles du jeu et de la dimension du plateau, on demandera de nouveau de le saisir. Finalement, la fonction retournera ce numéro.
- Une procédure « move(board,n,i,j) » où l'on suppose ici que **i** est le numéro d'une case contenant un pion déplaçable, et que **j** est le numéro de sa destination. Cette procédure réalise ce déplacement.
- Une fonction « lose(board,n) » qui retourne **False** s'il existe au moins un pion déplaçable sur le plateau et **True** sinon.
- Une procédure « nimble(n,p) » qui utilisera les sous-programmes précédents (et d'autres si besoin est) afin de permettre à deux joueurs de disputer une partie complète sur un plateau de **n** cases, chacune d'elle contenant au plus **p** pions.

Voici de nouveau l'exemple de la sous-partie 2.1, mais cette fois en utilisant notre programme :

```
 0 |  0 |  1 |  2 |  0 |  2 |
----------------------------
 1 |  2 |  3 |  4 |  5 |  6 |

Player  1
Choose a square : 2
Choose a square : 7
Choose a square : 4
Choose a destination : 2

 0 |  1 |  1 |  1 |  0 |  2 |
----------------------------
 1 |  2 |  3 |  4 |  5 |  6 |

Player  2
Choose a square : 2
Choose a destination : 1

 1 |  0 |  1 |  1 |  0 |  2 |
----------------------------
 1 |  2 |  3 |  4 |  5 |  6 |

Player  1
Choose a square : 4
Choose a destination : 3

 1 |  0 |  2 |  0 |  0 |  2 |
----------------------------
 1 |  2 |  3 |  4 |  5 |  6 |

Player  2
Choose a square : 6
Choose a destination : 1
```

```
Player  2
Choose a square : 6
Choose a destination : 1

 2 |  0 |  2 |  0 |  0 |  1 |
----------------------------
 1 |  2 |  3 |  4 |  5 |  6 |

Player  1
Choose a square : 3
Choose a destination : 1

 3 |  0 |  1 |  0 |  0 |  1 |
----------------------------
 1 |  2 |  3 |  4 |  5 |  6 |

Player  2
Choose a square : 6
Choose a destination : 1

 4 |  0 |  1 |  0 |  0 |  0 |
----------------------------
 1 |  2 |  3 |  4 |  5 |  6 |

Player  1
Choose a square : 3
Choose a destination : 1

 5 |  0 |  0 |  0 |  0 |  0 |
----------------------------
 1 |  2 |  3 |  4 |  5 |  6 |

Winner : 1
```

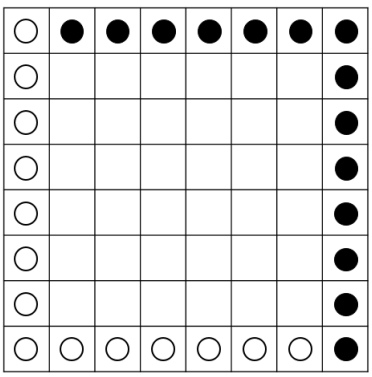
3 MING MANG

3.1 LES REGLES DE CE JEU

Remarque importante : aucun code n'est demandé dans cette sous-partie où l'on se contente d'expliquer les règles.

Il s'agit d'un jeu ancestral tibétain. Deux joueurs s'affrontent sur un plateau carré de dimension n, *i.e.* comportant n lignes et n colonnes. Le premier joueur possède des pions blancs et le second des pions noirs.

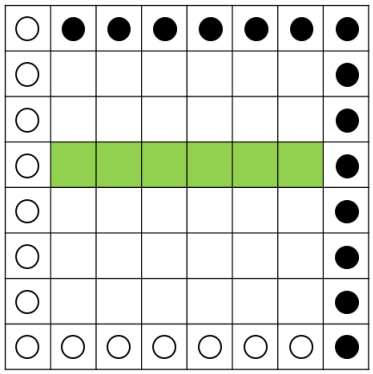
Au début de la partie, les pions sont positionnés comme suit (exemple avec un plateau de 8 lignes et 8 colonnes) :



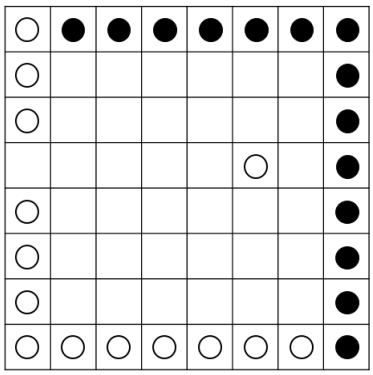
Il y a donc des pions blancs (*resp.* noirs) sur toute la première (*resp.* dernière) colonne et sur la dernière (*resp.* première) ligne à l'exception de la dernière (*resp.* première) case.

Les blancs commencent, puis à tour de rôle les joueurs déplacent l'un de leurs pions vers une case vide située sur la même colonne ou la même ligne que celui-ci, avec la contrainte qu'il n'y ait que des cases vides entre la position de départ et la position d'arrivée. On ne peut donc pas sauter au-dessus d'un autre pion, qu'il soit de la même couleur ou non.

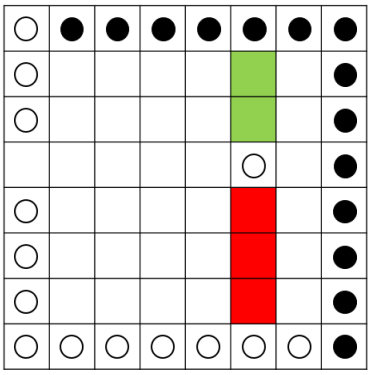
Par exemple à partir de la configuration initiale ci-dessus, si le premier joueur décide de bouger son pion situé sur la quatrième ligne et première colonne il ne pourra le faire que vers les cases colorées en vert :



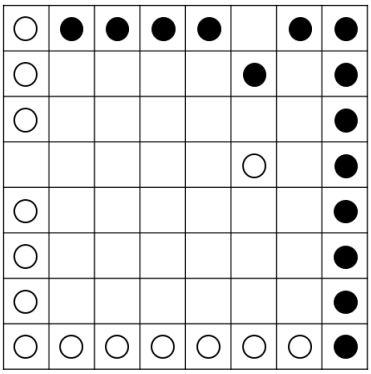
Par exemple, la case située sur la sixième colonne :



Par suite, si le second joueur décide de déplacer son pion située sur la première ligne et sixième colonne, il ne pourra le faire que vers les cases colorées en vert et pas vers celles en rouge :



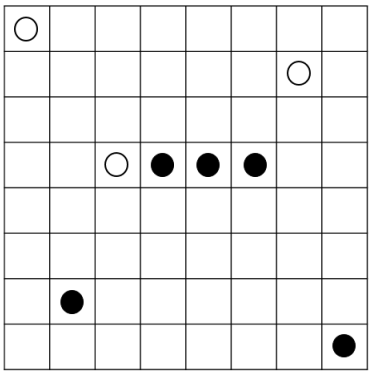
Par exemple, la case située sur la deuxième ligne :



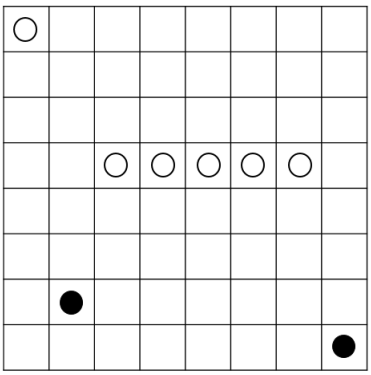
Et ainsi de suite.

Si après le déplacement d'un pion blanc (*resp.* noir), un ou plusieurs pions noirs (*resp.* blancs) se retrouvent bloqués entre deux pions blancs (*resp.* noirs), ces pions noirs (*resp.* blancs) sont capturés et changent de couleur. Dans le cas de plusieurs pions ainsi bloqués, ceux-ci doivent être sur des cases consécutives, et situés sur une même ligne ou une même colonne.

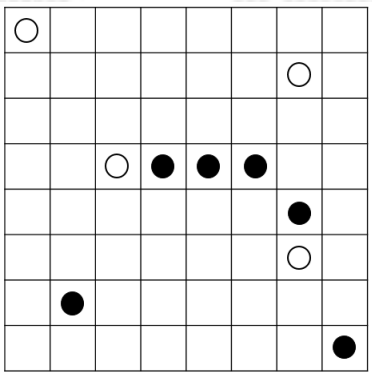
Considérons par exemple cette configuration :



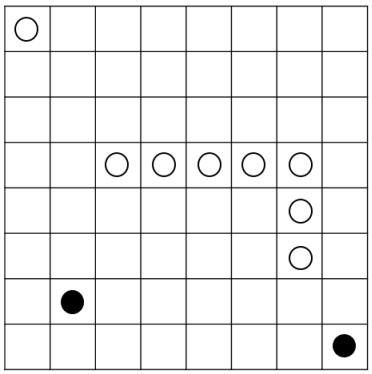
Si le joueur ayant les blancs déplace le pion situé sur la seconde ligne et septième colonne vers la case située sur la quatrième ligne et septième colonne, il capture trois pions noirs :



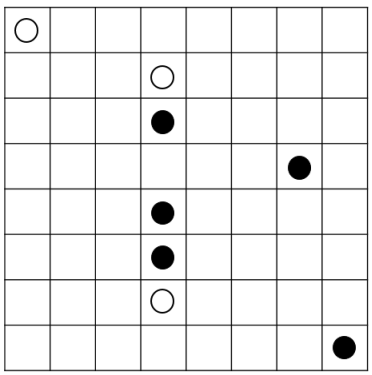
A noter que les prises peuvent se faire dans plusieurs directions simultanément. Considérons par exemple cette configuration :



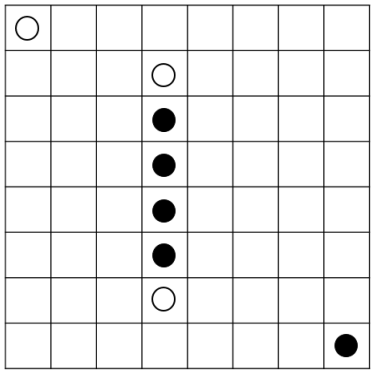
Si le joueur ayant les blancs déplace le pion situé sur la seconde ligne et septième colonne vers la case située sur la quatrième ligne et septième colonne, il capture quatre pions noirs, et ce dans deux directions :



Par contre un si lors d'un déplacement un pion forme un groupe de pions amis bloqués entre deux pions ennemis il n'y a pas de capture. Considérons par exemple cette configuration :



Si le joueur ayant les noirs déplace le pion situé sur la quatrième ligne et septième colonne vers la case située sur la quatrième ligne et quatrième colonne, il n'y a pas de prises :



Le gagnant est le dernier joueur à pouvoir déplacer un pion. Autrement dit, dès qu'un joueur n'a plus de pions ou ne peut plus déplacer de pion il a perdu.

3.2 PROGRAMMATION DE CE JEU EN PYTHON

Il vous est fortement recommandé de lire l'intégralité de cette partie avant de commencer à coder.

Remarques importantes :

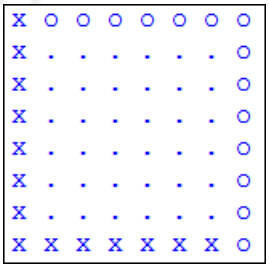
- On pourra éventuellement implémenter des sous-programmes en plus de ceux demandés.
- Le plateau sera naturellement une liste à deux dimensions d'entiers égaux à 0, 1 ou 2. Une case vide sera représentée par 0, un pion du premier joueur par 1 et un pion du second par 2.
- Plutôt que de recalculer régulièrement la dimension de cette liste on préfèrera la passer en paramètre de nos sous-programmes.
- L'exemple de rendu visuel du programme n'est qu'indicatif, vous pouvez l'améliorer.

Notations des paramètres des sous-programmes que l'on va implémenter :

- « board » : liste à deux dimensions d'entiers égaux à 0, 1 ou 2 représentant le plateau de jeu.
- « n » : entier strictement positif égal au nombre de lignes et de colonnes de « board ».
- « player » : entier représentant le joueur dont c'est le tour (par exemple 1 pour le premier joueur et 2 pour le second).
- « i » : entier quelconque.
- « j » : entier quelconque.
- « k » : entier quelconque.
- « l » : entier quelconque.

Implémenter les sous-programmes suivants dans un fichier "mingMang.py" :

- Une fonction « newBoard(n) » qui retourne une liste à deux dimensions représentant l'état initial d'un plateau de jeu à **n** lignes et **n** colonnes.
- Une procédure « display(board,n) » qui réalise l'affichage sur la console du plateau. On représentera une case vide par un '.', un pion blanc par un 'x' et un pion noir par un 'o'. Initialement, un plateau de 8 lignes et 8 colonnes sera donc affiché comme cela :



- Une fonction « possiblePawn(board,n,player,i,j) » qui retourne **True** si **i** et **j** sont les coordonnées d'un pion que le joueur **player** peut déplacer, et **False** sinon.
- Une fonction « selectPawn(board,n,player) » qui fait saisir au joueur **player** les coordonnées d'un pion pouvant se déplacer. On supposera qu'il existe un tel pion, on ne testera pas ce fait ici. Tant que ces coordonnées ne seront pas valides en regard des règles du jeu et de la dimension du plateau, on lui demandera de nouveau de les saisir. Finalement, la fonction retournera ces coordonnées.
- Une fonction « possibleDestination(board,n,i,j,k,l) » où l'on suppose ici que **i** et **j** sont les coordonnées d'un pion pouvant se déplacer. Cette fonction retourne **True** si **k** et **l** sont les coordonnées d'une case vers laquelle le pion de coordonnées **i** et **j** peut se déplacer, et **False** sinon.
- Une fonction « selectDestination(board,n,i,j) » où l'on suppose ici que **i** et **j** sont les coordonnées d'un pion pouvant se déplacer. Cette fonction fait saisir les coordonnées d'une case vers laquelle le pion de coordonnées **i** et **j** peut se déplacer. On supposera qu'il existe un telle case, on ne testera pas ce fait ici. Tant que ces coordonnées ne seront pas valides en regard des règles du jeu et de la dimension du plateau, on lui demandera de nouveau de les saisir. Finalement, la fonction retournera ces coordonnées.
- Une procédure « move(board,n,player,i,j,k,l) » où l'on suppose ici que **i** et **j** sont les coordonnées d'un pion que le joueur **player** peut déplacer, et que **k** et **l** sont les coordonnées de sa destination. Cette procédure réalise cette pose et les éventuelles captures qui en découlent.
- Une fonction « lose(board,n,player) » qui retourne **False** si le joueur **player** peut déplacer l'un de ses pions et **True** sinon.
- Une procédure « mingMang(n) » qui utilisera les sous-programmes précédents (et d'autres si besoin est) afin de permettre à deux joueurs de disputer une partie complète sur un plateau à **n** lignes et **n** colonnes.

Voici de nouveau l'exemple de début de partie de la sous-partie 3.1, mais cette fois en utilisant notre programme :

```
x o o o o o o o
x . . . . . . o
x . . . . . . o
x . . . . . . o
x . . . . . . o
x . . . . . . o
x . . . . . . o
x x x x x x x o

Player 1 :
Select a pawn, row : 1
Select a pawn, column : 1
Select a pawn, row : 4
Select a pawn, column : 1
Select a destination, row : 4
Select a destination, column : 8
Select a destination, row : 4
Select a destination, column : 6

x o o o o o o o
x . . . . . . o
x . . . . . . o
. . . . . x . o
x . . . . . . o
x . . . . . . o
x . . . . . . o
x x x x x x x o

Player 2 :
Select a pawn, row : 1
Select a pawn, column : 6
Select a destination, row : 4
Select a destination, column : 6
Select a destination, row : 5
Select a destination, column : 6
Select a destination, row : 2
Select a destination, column : 6

x o o o o . o o
x . . . . o . o
x . . . . . . o
. . . . . x . o
x . . . . . . o
x . . . . . . o
x . . . . . . o
x x x x x x x o
```

3.3 BONUS

Implémenter une autre version de ce jeu (on conservera bien sûr la précédente fonctionnelle), où un joueur n'a pas le droit d'effectuer un déplacement amenant à une configuration de la partie déjà rencontrée.

4 BAREME INDICATIF

Ce barème peut-être amener à évoluer, il n'est donc qu'**indicatif**.

- Partie 2 : 10 points
- Partie 3 : 30 points

Ce qui fait un total de 40 points ramenés sur 20 points par proportionnalité.