Cette tâche a été verrouillée le 5 Déc 2021 à 23:59.

Cet examen est individuel.

Votre rendu s'effectuera sous la forme d'une archive au format .zip contenant vos codes sources.

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.

Le but de ce mini-projet est de programmer en Python un petit jeu de stratégie combinatoire abstrait nommé Alak. Cette version du jeu a été conçue en 2001 par Alan Baljeu (Connexions vers un site externe.).

1 - Les règles du jeu

Deux joueurs s'affrontent sur un plateau unidimensionnel de ***n*** cases. Le premier joueur possède des pions blancs et le second des pions noirs. Au début de la partie toutes les cases sont vides.

Les blancs commencent, puis les joueurs vont à tour de rôle poser un de leurs pions en respectant les différentes contraintes suivantes :

1. Un joueur ne peut placer un pion que sur une case vide, et cette case ne doit pas avoir été occupée par l'un de ses propres pions au tour précédent (voir règle de capture à suivre). Si un joueur ne peut plus placer de pion la partie est terminée.
2. Si après qu'un joueur ait placé un pion, un groupe de pions adjacents de son adversaire n'a plus de cases adjacentes vides ceux-ci sont capturés et retirés du plateau. D'après la première règle, son adversaire ne pourra donc pas poser un pion sur l'une de ces cases au prochain tour. Par « groupe de pions » on entend « au moins un pion ».
3. Un joueur peut en revanche placer un pion et constituer un groupe de pions adjacents sans cases adjacentes vides sans que ce groupe de pions soit capturé.

Quand la partie est terminée, le vainqueur est le joueur ayant le plus de pions. Un match nul est possible.

Voici un exemple de début de partie pour fixer les idées. Le plateau possède ici 9 cases :

Le joueur 1 place un pion sur la case n°2 :

Le joueur 2 place un pion sur la case n° 3 :

Le joueur 1 place un pion sur la case n°7 :

Le joueur 2 place un pion sur la case n° 5 :

Le joueur 1 place un pion sur la case n°9 :

Le joueur 2 place un pion sur la case n° 6 :

Le joueur 1 place un pion sur la case n°4. Deux groupes de pions adjacents blancs n'ont plus alors de cases adjacentes vides, ils sont donc capturés :

Le joueur 2 ne peut pas placer sur les cases n°3,5 et 6 (car elles étaient occupées par un de ses pions au tour précédent). Il choisit de placer un pion sur la case n° 8. Il capture alors le pion du joueur 1 situé sur la case n°9 car celui-ci n'a pas de case adjacente vide :

Le joueur 1 ne peut pas placer un pion sur la case n°9 (mêmes raisons que précédemment). Il choisit de placer un pion sur la case n° 5 :

Le joueur 2 place un pion sur la case n° 3, ce qu'il peut faire d'après la troisième règle :

Et ainsi de suite.

2 - Implémentation 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.
- La grille sera naturellement une liste à une dimension 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.
- Pour mémoriser les numéros des cases des pions capturés lors du tour de jeu précédent le tour actuel, on utilisera également une liste.
- 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 égaux à 0, 1 ou 2 représentant le plateau de jeu.
- "n" : entier strictement positif égal au nombre d'éléments de "board".
- "player" : entier représentant le joueur dont c'est le tour (par exemple 1 pour le premier joueur et 2 pour le second).
- "removed" : liste mémorisant les numéros des cases des pions capturés lors du tour de jeu précédent le tour actuel.
- "i" : entier quelconque.

Implémenter les sous-programmes suivants :

- Une fonction "newBoard(n)" qui retourne une liste à une dimension représentant l'état initial d'un plateau de jeu de **n** cases.
- 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'. Après quelques tours de jeu on aura donc un affichage du style :

```
. xo. ouais. X . . . . . .
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

- Une fonction "possible(board, n, player, removed, i)" qui retourne **True** si **i** est l'indice d'une case où le joueur **player** peut poser un pion, et **False** sinon.
- Une fonction "select(board, n, player, removed)" qui fait saisir au joueur **player** le numéro d'une case où il peut poser un pion. 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, on lui demandera de nouveau de le saisir. Finalement, la fonction retournera ce numéro.
- Une procédure "put(board, n, player, removed, i)" où l'on suppose ici que **i** est le numéro d'une case où le joueur **player** peut poser un pion. Cette

procédure réalise cette pose et ce qui en découle (captures, mise à jour de **removed**, etc.).
- Une fonction "again(board, n, player, removed)" qui retourne **True** si le joueur **player** peut poser un pion sur le plateau et **False** sinon.
- Une fonction "win(board, n)" qui retourne une chaîne de caractères indiquant l'issue de la partie.
- Un programme principal "alak(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 comportant **n** cases.

Voici de nouveau l'exemple de la partie 1, mais cette fois en utilisant notre programme et en finissant la partie :

```
. . . . . .
1 2 3 4 5 6 7 8 9


Joueur 1
Choisissez un nombre licite de carré : 2


. X . . . . . .
1 2 3 4 5 6 7 8 9


Joueur 2
Choisissez un nombre licite de carré : 3


. xo. . . . . .
1 2 3 4 5 6 7 8 9


Joueur 1
Choisissez un nombre licite de carré : 2
Choisissez un nombre licite de carré : 10
Choisissez un nombre licite de carré : 7


. xo. . . X . .
1 2 3 4 5 6 7 8 9
```

```
Joueur 2
Choisissez un nombre licite de carré : 5

. xo. o. X . .
1 2 3 4 5 6 7 8 9

Joueur 1
Choisissez un nombre licite de carré : 9

. xo. o. X . X
1 2 3 4 5 6 7 8 9

Joueur 2
Choisissez un nombre licite de carré : 6

. xo. ouais. X
1 2 3 4 5 6 7 8 9

Joueur 1
Choisissez un nombre licite de carré : 4

. X . X . . X . X
1 2 3 4 5 6 7 8 9

Joueur 2
Choisissez un nombre licite de carré : 3
Choisissez un nombre licite de carré : 5
Choisissez un nombre licite de carré : 6
Choisissez un nombre licite de carré : 8
```

```
. X . X . . xo.
1 2 3 4 5 6 7 8 9

Joueur 1
Choisissez un nombre licite de carré : 9
Choisissez un nombre licite de carré : 5

. X . xx. xo.
1 2 3 4 5 6 7 8 9

Joueur 2
Choisissez un nombre licite de carré : 2
Choisissez un nombre licite de carré : 3

. xoxx. xo.
1 2 3 4 5 6 7 8 9

Joueur 1
Choisissez un nombre licite de carré : 1

xxxxx . xo.
1 2 3 4 5 6 7 8 9

Joueur 2
Choisissez un nombre licite de carré : 6

xxx. . o. o.
1 2 3 4 5 6 7 8 9
```

```
Joueur 1
Choisissez un nombre licite de carré : 5
Choisissez un nombre licite de carré : 9

xxx. . o. bœuf
1 2 3 4 5 6 7 8 9

Joueur 2
Choisissez un nombre licite de carré : 5

xxx. oui bœuf
1 2 3 4 5 6 7 8 9

Joueur 1
Choisissez un nombre licite de carré : 4

xx. xoo. bœuf
1 2 3 4 5 6 7 8 9

Joueur 2
Choisissez un nombre licite de carré : 7

xx. xoooox
1 2 3 4 5 6 7 8 9

Joueur 1
Choisissez un nombre licite de carré : 3

xxxxoooox
1 2 3 4 5 6 7 8 9
```

```
Gagnant : 1
```

3 - Bonus

Les deux bonus suivants sont facultatifs et ne rentrent donc pas dans le barème de base. Ils apporteront des points supplémentaires.

3.1 - Une version torique

Implémenter une autre version de ce jeu (on conservera bien sûr la précédente fonctionnelle), où la première case du plateau est maintenant adjacente avec la dernière.

3.2 - Une autre règle de prises

Implémenter une autre version de ce jeu (on conservera bien sûr la précédente fonctionnelle), où l'on supprime la règle 3, c'est-à-dire que si un joueur place un pion constituant un groupe de pions adjacents sans cases adjacentes vides alors ce groupe de pions est capturé.