Mini-Projet [Sujet]

Ouvert le : vendredi 4 octobre 2024, 16:50
À rendre : dimanche 27 octobre 2024, 23:59

Cet examen est individuel et sans soutenance.

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 deux petits jeux de stratégie combinatoire abstraits.

La qualité du code sera pris en compte de façon significative lors de la notation. Les variables globales sont par exemple interdites.

Pour chacun des deux jeux vous joindrez une petite documentation expliquant vos algorithmes.

La structure du code est imposée, ne pas la respecter entraînera un ajournement direct à ce projet.

1 - Snort

1.1 Les règles du jeu

Ce jeu a été créé en 1970 par Simon Norton.

Deux joueurs s'affrontent sur un plateau bi-dimensionnel carré de n cases sur n cases initialement vide.

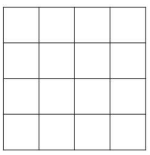
Le premier joueur possède des pions blancs et le second des pions noirs.

À tour de rôle, chaque joueur place un pion sur une case vide non adjacente orthogonalement à une case contenant un pion de l'adversaire.

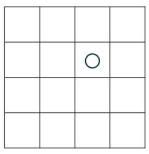
Le gagnant est le dernier joueur à pouvoir placer l'un de ses pions. Ou de façon équivalente, un joueur perd la partie dès qu'il ne peut plus placer l'un de ses pions sur le plateau.

Voici un exemple de partie sur un plateau de 4 cases sur 4 cases.

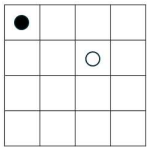
Plateau initialement vide :



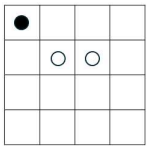
Tour du joueur 1 :



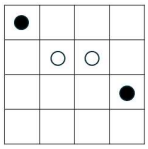
Tour du joueur 2 :



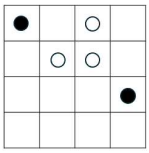
Tour du joueur 1 :



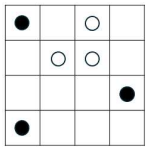
Tour du joueur 2 :



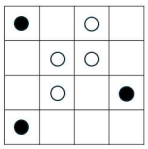
Tour du joueur 1 :



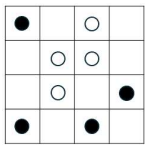
Tour du joueur 2 :



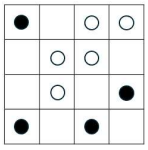
Tour du joueur 1 :



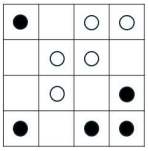
Tour du joueur 2 :



Tour du joueur 1 :



Tour du joueur 2 :



Le vainqueur est donc le joueur 2 car le joueur 1 ne peut plus poser de pions.

1.2 - 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.
- 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", "j" : entiers quelconques.

Implémenter les sous-programmes suivants :

- Une fonction "newBoard(n)" qui retourne une liste à deux dimensions représentant l'état initial d'un plateau de jeu de **n** cases sur **n** cases.
- Une procédure "displayBoard(board, n)" qui réalise l'affichage du plateau sur la console. On représentera une case vide par un '.', un pion blanc par un 'x' et un pion noir par un 'o'. On numérotera les lignes et les colonnes (à partir de 1) pour que les joueurs puissent repérer facilement les coordonnées d'une case. On aura donc un affichage ressemblant à celui-ci (après quelques tours de jeu) :

```
1 | x . . . . . . .
2 | . x x . . . . .
3 | . . . . . . . .
4 | . . . o . . . .
5 | x . o . . . . o
6 | . . . o . . . .
7 | . . . . . . . .
8 | . . . . . . . .
  ---------------
    1 2 3 4 5 6 7 8
```

- Une fonction "possibleSquare(board, n, player, i, j)" qui retourne **True** si **i** et **j** sont les coordonnées d'une case où le joueur **player** peut poser un pion, et **False** sinon.
- Une fonction "selectSquare(board, n, player)" qui fait saisir au joueur **player** les coordonnées 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 ces coordonnées ne seront pas valides en regard des règles du jeu (et des dimensions du plateau), on lui demandera de nouveau de les saisir. Finalement, la fonction retournera ces coordonnées.
- Une procédure "updateBoard(board, player, i, j)" où l'on suppose ici que **i** et **j** sont les coordonnées d'une case où le joueur **player** peut poser un pion. Cette procédure réalise cette pose.
- Une fonction "again(board, n, player)" qui retourne **True** si le joueur **player** peut encore poser un pion sur le plateau et **False** sinon.
- Un programme principal "snort(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 de jeu de **n** cases sur **n** cases.

Voici de nouveau l'exemple de la sous-partie 1.1, mais cette fois en utilisant notre programme :

```
 1 | . . . .
 2 | . . . .
 3 | . . . .
 4 | . . . .
   ---------

    1 2 3 4
Au joueur 1 de jouer
Choisir un numéro de ligne : 2
Choisir un numéro de colonne : 3

 1 | . . . .
 2 | . . x .
 3 | . . . .
 4 | . . . .
   ---------

    1 2 3 4
Au joueur 2 de jouer
Choisir un numéro de ligne : 2
Choisir un numéro de colonne : 2
Choisir un numéro de ligne : 12
Choisir un numéro de colonne : 111
Choisir un numéro de ligne : 1
Choisir un numéro de colonne : 1

 1 | o . . .
 2 | . . x .
 3 | . . . .
 4 | . . . .
   ---------

    1 2 3 4
Au joueur 1 de jouer
Choisir un numéro de ligne : 2
Choisir un numéro de colonne : 2

 1 | o . . .
 2 | . x x .
 3 | . . . .
 4 | . . . .
   ---------

    1 2 3 4
Au joueur 2 de jouer
Choisir un numéro de ligne : 3
Choisir un numéro de colonne : 4

 1 | o . . .
 2 | . x x .
 3 | . . . o
 4 | . . . .
   ---------

    1 2 3 4
Au joueur 1 de jouer
Choisir un numéro de ligne : 1
Choisir un numéro de colonne : 2
Choisir un numéro de ligne : 1
Choisir un numéro de colonne : 3

 1 | o . x .
 2 | . x x .
 3 | . . . o
 4 | . . . .
   ---------

    1 2 3 4
Au joueur 2 de jouer
Choisir un numéro de ligne : 4
Choisir un numéro de colonne : 1

 1 | o . x .
 2 | . x x .
 3 | . . . o
 4 | o . . .
   ---------

    1 2 3 4
Au joueur 1 de jouer
```

```
Choisir un numéro de ligne : 3
Choisir un numéro de colonne : 2

 1 | o . x .
 2 | . x x .
 3 | . x . o
 4 | o . . .
   ---------

     1 2 3 4
Au joueur 2 de jouer
Choisir un numéro de ligne : 4
Choisir un numéro de colonne : 3

 1 | o . x .
 2 | . x x .
 3 | . x . o
 4 | o . o .
   ---------

     1 2 3 4
Au joueur 1 de jouer
Choisir un numéro de ligne : 1
Chosiir un numéro de colonne : 4

 1 | o . x x
 2 | . x x .
 3 | . x . o
 4 | o . o .
   ---------

     1 2 3 4
Au joueur 2 de jouer
Choisir un numéro de ligne : 4
Choisir un numéro de colonne : 4

 1 | o . x x
 2 | . x x .
 3 | . x . o
 4 | o . o o
   ---------

     1 2 3 4
Vainqueur : 2
```

2 - Dodgem

2.1 - Les règles du jeu

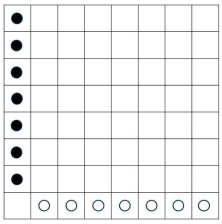
Ce jeu a été créé en 1972 par Colin Vout.

Deux joueurs s'affrontent sur un plateau bi-dimensionnel carré de n cases sur n cases.

Le premier joueur possède des pions blancs et le second des pions noirs.

Initialement le premier joueur dispose $n-1$ pions sur la dernière ligne en excluant la première case de celle-ci, et le second joueur dispose $n-1$ pions sur la première colonne en excluant la dernière case de celle-ci.

Par exemple pour $n=8$ on a :



À tour de rôle, chaque joueur déplace l'un de ses pions vers une case vide orthogonalement adjacente.

Le premier joueur ne peut déplacer un pion que vers le haut, vers la gauche ou vers la droite.

Le second joueur ne peut déplacer un pion que vers la droite, vers le haut ou vers le bas.

Si le premier joueur possède un pion sur la ligne du haut et qu'il décide de le déplacer vers le haut, ce pion sort définitivement du plateau.

Si le second joueur possède un pion sur la colonne de droite et qu'il décide de le déplacer vers la droite, ce pion sort définitivement du plateau.

Un joueur gagne la partie s'il réussit à faire sortir tous ses pions du plateau ou si tous ses pions sont bloqués par ceux de son adversaire.

Voici un exemple de partie sur un plateau de 3 cases sur 3 cases.

Configuration initiale du plateau :

Tour du joueur 1 :

Tour du joueur 2 :

Tour du joueur 1 :

Tour du joueur 2 :

Tour du joueur 1 :

Tour du joueur 2 :

Tour du joueur 1 :

Tour du joueur 2 :

Tour du joueur 1 :

Tour du joueur 2 :

Tour du joueur 1 :

Tour du joueur 2 :

Le joueur 2 a gagné car il a réussi à faire sortir tous ses pions du plateau.

2.2 - 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 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", "j" : entiers quelconques.
- "directions" : un t-uple de 4 couples constituant les 4 directions orthogonales possibles `directions = ((-1, 0), (0, 1), (1, 0), (0, -1))`.
- "m" : entier compris entre 1 et 4 qui repère l'une des 4 directions.

Implémenter les sous-programmes suivants :

- Une fonction "newBoard(n)" qui retourne une liste à deux dimensions représentant l'état initial d'un plateau de jeu de **n** cases sur **n** cases.
- Une procédure "displayBoard(board, n)" qui réalise l'affichage du plateau sur la console. On représentera une case vide par un '.', un pion blanc par un 'x' et un pion noir par un 'o'. On numérotera les lignes et les colonnes (à partir de 1) pour que les joueurs puissent repérer facilement les coordonnées d'une case. La configuration initiale du plateau de 5 lignes et 5 colonnes sera donc affichée comme cela :

```
1 | o . . . . . .
2 | o . . . . . .
3 | o . . . . . .
4 | o . . . . . .
5 | o . . . . . .
6 | o . . . . . .
7 | o . . . . . .
8 | . x x x x x x x
  ---------------
    1 2 3 4 5 6 7 8
```

- Une fonction "possiblePawn(board, n, directions, 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, directions, 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 des dimensions du plateau, on lui demandera de nouveau de les saisir. Finalement, la fonction retournera ces coordonnées.
- Une fonction "possibleMove(board, n, directions, player, i, j, m)" où l'on suppose ici que **i** et **j** sont les coordonnées du pion que le joueur **player** souhaite déplacer. Cette fonction retourne **True** si le joueur **player** peut déplacer ce pion dans la direction **m** et **False** sinon.
- Une fonction "selectMove(board, n, directions, player, i, j)" où l'on suppose ici que **i** et **j** sont les coordonnées du pion que le joueur **player** souhaite déplacer. Cette fonction fait saisir au joueur **player** une direction dans laquelle il souhaite déplacer ce pion. Tant que cette direction ne sera pas valide en regard des règles du jeu et des dimensions du plateau, on lui demandera de nouveau de la saisir. Finalement, la fonction retournera cette direction.

- Une procédure "move(board, n, directions, player, i, j, m)" où l'on suppose ici que **i** et **j** sont les coordonnées du pion que le joueur **player** souhaite déplacer dans la direction **m**. Cette procédure réalise ce déplacement.
- Une fonction "win(board, n, directions, player)" qui retourne **True** si le joueur **player** a gagné la partie et **False** sinon.
- Un programme principal "dodgem(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 de jeu de **n** cases sur **n** cases.

Voici une utilisation de notre programme pour présenter un exemple de partie avec une condition de victoire par blocage :

```
 1 | o . .
 2 | o . .
 3 | . x x
   ------

     1 2 3
Au joueur 1 de jouer
Choisir la ligne d'un pion : 3
Choisir la colonne d'un pion : 2
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 2
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 3
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 4

 1 | o . .
 2 | o . .
 3 | x . x
   ------

     1 2 3
Au joueur 2 de jouer
Choisir la ligne d'un pion : 12
Choisir la colonne d'un pion : 222
Choisir la ligne d'un pion : 3
Choisir la colonne d'un pion : 3
Choisir la ligne d'un pion : 1
Choisir la colonne d'un pion : 1
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 2

 1 | . o .
 2 | o . .
 3 | x . x
   ------

     1 2 3
Au joueur 1 de jouer
Choisir la ligne d'un pion : 3
Choisir la colonne d'un pion : 3
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 4

 1 | . o .
 2 | o . .
 3 | x x .
   ------

     1 2 3
Au joueur 2 de jouer
Choisir la ligne d'un pion : 2
Choisir la colonne d'un pion : 1
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 1

 1 | o o .
 2 | . . .
 3 | x x .
   ------

     1 2 3
Au joueur 1 de jouer
Choisir la ligne d'un pion : 3
Choisir la colonne d'un pion : 1
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 1

 1 | o o .
 2 | x . .
 3 | . x .
   ------

     1 2 3
Au joueur 2 de jouer
Choisir la ligne d'un pion : 1
Choisir la colonne d'un pion : 2
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 2

 1 | o . o
 2 | x . .
 3 | . x .
   ------

     1 2 3
Au joueur 1 de jouer
Choisir la ligne d'un pion : 3
```

```
Choisir la colonne d'un pion : 2
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 1

 1 | o . o
 2 | x x .
 3 | . . .
   ---------

     1 2 3
Au joueur 2 de jouer
Choisir la ligne d'un pion : 1
Choisir la colonne d'un pion : 3
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 2

 1 | o . .
 2 | x x .
 3 | . . .
   ---------

     1 2 3
Au joueur 1 de jouer
Choisir la ligne d'un pion : 2
Choisir la colonne d'un pion : 2
Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : 1

 1 | o x .
 2 | x . .
 3 | . . .
   ---------

     1 2 3
Vainqueur : 2
```

Ajouter un travail

Statut de remise

Statut des travaux remis	Aucun devoir n'a encore été remis
Statut de l'évaluation	Non évalué
Temps restant	21 jours 6 heures restants
Dernière modification	-
Commentaires	▸ Commentaires (0)