AP1 TP

9 Semaine 9 : Matrices 1/1

Exercice 1 : Morpion

Traditionnellement, un morpion se joue à deux joueurs sur une grille 3×3. Le plateau de jeu peut être modélisé par un tableau d'entiers à deux dimensions :

		X
	X	O
X		O

0	0	1
0	1	2
1	0	2

En C++, un tableau à deux dimensions d'entiers peut s'écrire :

```
int mon_tableau[NB_LIGNES][NB_COLONNES];
```

Par convention, le premier indice entre crochet désigne la ligne (entre 0 et `NB_LIGNES-1`) et le deuxième indice désigne la colonne (entre 0 et `NB_COLONNES-1`). Dans notre cas, on pourra donc définir une grille de morpion de la manière suivante :

```
int grille[3][3];
```

Dans notre exemple, `grille[0][2]` vaut 1, `grille[2][2]` vaut 2, `grille[1][0]` vaut 0. On convient que si un élément (une case) de la grille vaut 0, alors personne n'a joué sur cette case. Si une case vaut 1, alors le joueur 1 a déposé un jeton dessus. Si une case vaut 2, alors le joueur 2 a déposé un jeton dessus. Il est donc très facile de faire un morpion avec un nombre arbitraire de joueurs, par exemple numéroté de 1 à `NB_JOUEURS`.

Ceux qui veulent faire un morpion encore plus général (rien ne l'empêche) pourront par exemple définir une constante `TAILLE` et définir une grille de taille `TAILLE` fois `TAILLE`.

On va donc maintenant écrire un morpion (le nombre de joueurs différent de 2 ou la taille variable de la grille sont optionnels). Chaque joueur va jouer alternativement (d'abord le 1, puis le 2, etc.). A chaque fois qu'un jeton sera déposé, il faudra vérifier si le joueur a gagné la partie. Un joueur gagne si il a réussi à former une colonne de jeton à lui, une ligne de jeton à lui, ou une diagonale de jeton à lui.

1. **Affichage de la grille**.
   Écrire une fonction qui prend une grille en paramètre et l'affiche comme une grille de morpion. Par exemple :

```
   |   | X
---+---+---
   | X | O
---+---+---
 X |   | O
```

Tests	Résultat(s) attendu(s)	Résultat(s) observé(s)
0 ;0 ;1 2 ;0 ;0 1 ;2 ;0		

2. **Case vide**. Écrire une fonction qui retourne `true` ou `false` selon que la case spécifiée est libre ou non.

Tests	Résultat(s) attendu(s)	Résultat(s) observé(s)
0 ;0 ;1 2 ;0 ;0 1 ;2 ;0		
tester cases [0][0] [0][3] [0][2] [1][0]		

3. **Déposer un jeton**. Écrire une fonction `LirePos` qui prend deux entiers i, j en paramètres. Cette action permet de saisir les indices valides d'une case libre. Écrire une fonction permettant à un joueur de déposer un jeton sur une case libre.

Tests	Résultat(s) attendu(s)	Résultat(s) observé(s)
0 ;0 ;1 2 ;0 ;0 1 ;2 ;0		
tester dépôt cases [0][0] [0][3] [1][2] [1][0] [1][1]		

4. **Ligne complète**. Écrire la fonction qui renvoie `true` si une ligne donnée est entièrement remplie par des jetons d'un joueur.

Tests	Résultat(s) attendu(s)	Résultat(s) observé(s)
1 ;1 ;1 0 ;0 ;0 2 ;2 ;0		
tester lignes 1 2 3		

5. **Colonne complète**. Même question que précédemment avec cette fois-ci une colonne.

Tests	Résultat(s) attendu(s)	Résultat(s) observé(s)
1 ;0 ;2 1 ;0 ;2 1 ;0 ;0		
tester lignes 1 2 3		

6. **Diagonale complète**. Écrire la fonction qui renvoie `true` si une de deux diagonales est entièrement remplie par des jetons d'un joueur.

Tests	Résultat(s) attendu(s)	Résultat(s) observé(s)
1 ;0 ;0 1 ;1 ;2 0 ;2 ;1		
tester les 2 diagonales		
1 ;0 ;2 0 ;2 ;1 2 ;0 ;0		
tester les 2 diagonales		

7. **Morpion**. A l'aide des fonctions précédentes, vous pouvez écrire simplement un jeu de morpion complet (vous pouvez écrire au besoin des fonctions supplémentaires).