TP1 - Puissance 4

À retenir :

- manipulation de tableaux statiques bidimensionnels
- fonction `rand()`
- fonctions `scanf` et `printf`
- lecture et écriture dans un fichier

Le Puissance 4 est un jeu populaire à deux joueurs, qui se joue sur une grille verticale de six lignes et sept colonnes. À tour de rôle, chaque joueur fait tomber un pion de sa couleur dans une colonne de son choix non encore pleine. Le premier joueur qui aligne quatre pions de sa couleur, horizontalement, verticalement ou en diagonal, gagne la partie.

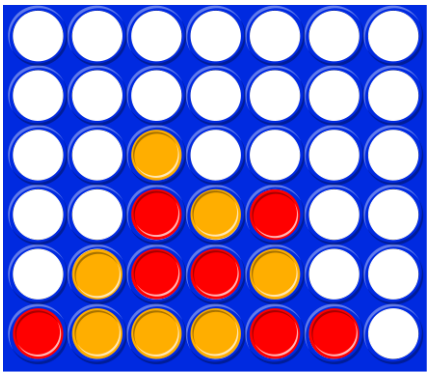

Figure 1: Exemple de partie en cours

I - Simulation de parties

Pour représenter une grille de jeu, on utilise un tableau à deux dimensions de taille 6×7, la première dimension représentant les lignes, de la plus haute à la plus basse, et la seconde les colonnes, de gauche à droite. Un joueur est noté j et prend la valeur 1 ou 2. Dans une grille, la valeur 0 représente une case vide et la valeur 1 ou 2 représente un pion du joueur correspondant.

Ainsi, la grille de l'exemple sera représentée par le tableau suivant :

```
1 int grille_exemple[6][7] = {{0, 0, 0, 0, 0, 0, 0},
2                             {0, 0, 0, 0, 0, 0, 0},
3                             {0, 0, 1, 0, 0, 0, 0},
4                             {0, 0, 2, 1, 2, 0, 0},
5                             {0, 1, 2, 2, 1, 0, 0},
6                             {2, 1, 1, 1, 2, 2, 0}};
```

1. Écrire une fonction `void affiche(int g[6][7])` qui affiche une grille avec le caractère `"*"` pour une case vide, `"X"` pour le joueur 1 et `"O"` pour le joueur 2. Tester votre fonction.

2. Quand un joueur joue dans une colonne, le pion tombe dans la ligne la plus basse possible de cette colonne, de sorte qu'il n'y ait pas de trou sous un pion : autrement dit, un pion ne peut pas flotter au-dessus d'une case vide dans sa colonne. Écrire une fonction `bool grille_valide(int g[6][7])` qui renvoie `true` si la grille respecte cette règle physique et `false` sinon. Tester la fonction.

3. Écrire une fonction `bool coup_possible(int g[6][7], int c)` qui renvoie un booléen indiquant s'il est possible de jouer dans la colonne `c`, c'est à dire si la colonne n'est pas pleine.

4. Écrire une fonction `void jouer(int g[6][7], int j, int c)` qui joue un coup du joueur `j` dans la colonne `c`, en supposant que la colonne `c` n'est pas pleine et que la grille est valide.
Remarque : La fonction doit directement modifier la grille.

5. Tester les fonctions.

6. Écrire quatre fonctions `horiz(int g[6][7], int j, int l, int c)`, `vert(int g[6][7], int j, int l, int c)`, `diag_haut(int g[6][7], int j, int l, int c)` et `diag_bas(int g[6][7], int j, int l, int c)` qui déterminent s'il y a un alignement horizontal vers la droite, vertical vers le bas, suivant une diagonale "montante vers la droite" ou une diagonale "descendante vers la droite" de quatre pions du joueur `j` à partir de la case `(l, c)`. Tester vos fonctions.
Remarque : Certains alignements ne sont trivialement pas possibles pour certaines valeurs du couple `(l, c)`. On pourra tenir compte de ces impossibilités dans les fonctions précédentes ou bien dans la fonction ci-dessous, au choix.

7. Écrire une fonction `bool victoire(int j, int g[6][7])` qui renvoie un booléen indiquant si le joueur `j` a gagné.

8. Écrire une fonction `bool match_nul(int g[6][7])` qui renvoie un booléen indiquant s'il y a match nul lorsque la grille est totalement remplie.

9. Tester vos deux fonctions.

10. Écrire une fonction `coup_aleatoire(int g[6][7], int j)` qui joue un coup aléatoire **autorisé** pour le joueur `j`, en supposant que la grille n'est pas pleine.
Indication : On rappelle qu'on peut générer un entier aléatoire entre 0 et `a` en utilisant `rand()% a`.

11. Écrire un programme `void partie_aleatoire()` qui fait jouer deux adversaires aléatoirement à tour de rôle, depuis une grille vide, en affichant la grille après chaque coup, et qui s'arrête dès qu'un joueur gagne ou que la partie est nulle.

12. Écrire un programme `void jeu_solo()` qui permet à l'utilisateur de jouer contre l'ordinateur jouant aléatoirement, en affichant la grille après chaque coup et le résultat en fin de partie.
Indication : Pour récupérer le coup du joueur, on utilisera la fonction `scanf`.

II - Système de sauvegarde

Afin de pouvoir interrompre une partie et la reprendre plus tard, on souhaite mettre en place un système de sauvegarde. Pour cela, on définit une structure permettant de représenter l'état d'une partie.

13. Proposer une structure `struct partie` contenant la grille, et le joueur dont c'est le tour de jouer. On définira alors

```
1 typedef struct partie partie;
```

14. Écrire une fonction `void init_partie(partie *p)` qui initialise une partie vide (grille remplie de zéros, joueur courant = 1).

15. Écrire une fonction `void sauvegarde(partie *p, char *nom_fichier)` qui enregistre l'état d'une partie dans un fichier texte.

- Chaque ligne du fichier contient les 7 cases d'une ligne de la grille.
- La dernière ligne contient l'entier correspondant au joueur courant.

Exemple de fichier `"partie.txt"` :

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 1 0 0 0 0
0 0 2 1 2 0 0
0 1 2 2 1 0 0
2 1 1 1 2 2 0
1
```

16. Écrire une fonction `void charge(partie *p, char *nom_fichier)` qui lit un fichier de sauvegarde et initialise la structure `partie` correspondante.

17. Modifier la fonction `void jeu_solo()` pour proposer à l'utilisateur, au début du programme, le choix suivant :

- (1) Commencer une nouvelle partie
- (2) Charger une partie existante

Si l'utilisateur choisit (2), la partie est chargée depuis le fichier `"partie.txt"`.

18. Ajouter la possibilité, à tout moment pendant la partie, de taper une commande spéciale (par exemple la colonne `-1`) pour sauvegarder la partie en cours et quitter le programme.

- Le programme écrit alors la partie dans `"partie.txt"`
- Puis s'arrête immédiatement.