Jeu du puissance 4

Le *puissance 4* est un jeu à deux joueurs, le joueur rouge (1) et le joueur jaune (2), se jouant sur une grille de 6 lignes et 7 colonnes. Chacun à son tour, les joueurs laissent tomber un jeton, en haut d'une des 7 colonnes, le jeton vient alors s'empiler en haut des jetons déjà présents dans la colonne. Le joueur qui parvient à créer un alignement de 4 jetons de sa couleur gagne la partie.

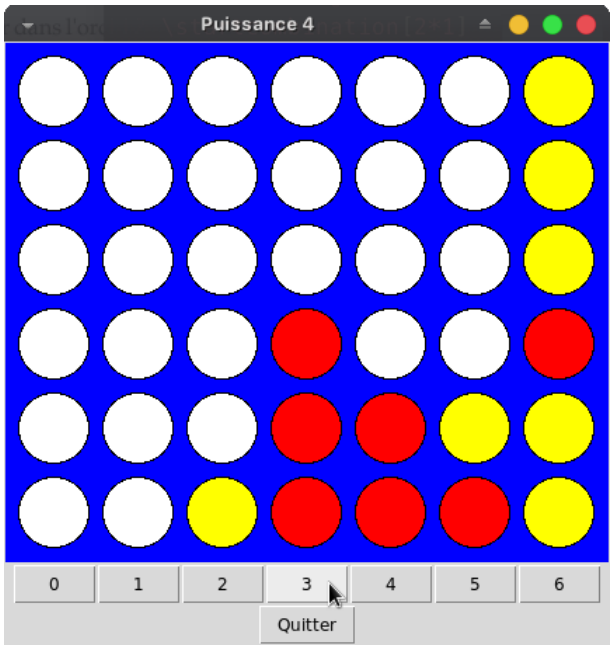

Figure 1 Jeu du puissance 4

1 Programmation du jeu

On modélise la grille de jeu par une matrice d'entiers de 6 lignes et 7 colonnes. La ligne d'indice 0 sera la ligne située tout en haut de la grille de puissance 4; ainsi la ligne d'indice 5 est la ligne du bas.

Les entiers de la matrice représentent le contenu des cases selon le code suivant :

```
const int VIDE = 0
const int A = 1 // rouge
const int B = 2 // jaune
```

Un *coup* pour un joueur est un entier entre 0 et 6 représentant le numéro de colonne choisi.

Question 1

Écrire une fonction `void print_grille(int grille[6][7])` affichant sur la sortie standard l'état du jeu.

Question 2

Écrire une fonction `bool est_libre(int grille[6][7], int j)` retournant un booléen indiquant s'il est possible de jouer dans la colonne `j` avec la `grille` donnée.

Question 3

Écrire une fonction `int coups_possibles(int grille[6][7], int res[7])` inscrivant au début du tableau `res` la liste des coups qu'il est possible de jouer avec la `grille` donnée. La valeur de retour sera le nombre de coups qu'il est possible de jouer, c'est-à-dire le nombre de valeurs utiles enregistrées dans `res`.

Question 4

Écrire une fonction `bool est_final(int grille[6][7])` testant si la grille correspond à une position finale du jeu, dans le sens où il n'y a plus de place pour jouer. (La condition de victoire sera traitée plus tard avec la fonction `gagne`).

Question 5

Écrire une fonction `void jouer(int grille[6][7], int joueur, int j)` qui modifie la `grille` donnée en entrée, en faisant jouer le joueur `joueur` en colonne `j`. À l'aide d'une assertion, on pourra annoter en pré-condition que le coup proposé est jouable.

On donne la fonction suivante :

```
bool diagonale1(int grille[6][7], int joueur) {
    for (int i=0; i < 3; i++) {
        for (int j=0; j < 4; j++) {
            if (grille[i][j] == joueur
                && grille[i+1][j+1] == joueur
                && grille[i+2][j+2] == joueur
                && grille[i+3][j+3] == joueur
                )
            return true;
        }
    }
    return false;
}
```

Question 6

Quel est le rôle de la fonction `diagonale1` ? Écrire de même les fonctions :
a. `ligne`
b. `colonne`
c. `diagonale2`

Question 7

En déduire une fonction `bool gagne(int grille[6][7], int joueur)` qui retourne `true` si et seulement si dans la position `grille` le joueur `joueur` a gagné la partie.

Question 8

Écrire une fonction principale `main` qui exécute une partie de puissance 4 en humain contre humain. À chaque tour il faudra :
1. Afficher la grille
2. Afficher le joueur qui doit jouer
3. Attendre que le joueur tape un numéro de colonne
4. Si le coup est valide jouer le coup et réagir en conséquence

2 Intelligence artificielle : algorithme min-max

On souhaite maintenant avoir la possibilité de jouer au puissance 4 contre un *bot*. On programmera ce *bot* à l'aide de l'algorithme min-max vu en cours. On rappelle que celui-ci repose sur l'utilisation

d'une fonction *heuristique* qui permet d'estimer les chances de gain de chaque joueur. On considérera qu'une heuristique positive signifie que le joueur A a l'avantage. L'heuristique vaudra $+\infty$ (resp. $-\infty$) lorsque la position est gagnée pour le joueur A (resp. le joueur B).

On donne la fonction heuristique basique suivante :

```
const int INF = 1000;
int heuristique(int grille[6][7]):
    if (gagne(grille, A)) return +INF;
    if (gagne(grille, B)) return -INF;
    return 0;
}
```

Question 9

Écrire une fonction

`int eval(int grille[6][7], int joueur, int profondeur)`

qui retourne une évaluation de la `grille`, dans la situation où c'est au tour du joueur `joueur`. Cette évaluation reposera sur l'algorithme *min-max* utilisé avec la profondeur de recherche `profondeur`. Elle utilisera la fonction `heuristique`.

Pour tester, modifier le main pour qu'il affiche à chaque tour son évaluation de partie.

On introduit le type :

```
struct evaluation_s {
    int valeur; // Valeur de l'évaluation
    int coup; // Meilleur coup à jouer
};
typedef struct evaluation_s evaluation;
```

Question 10

Écrire une fonction

`evaluation minmax(int grille[6][7], int joueur, int profondeur)`

basée sur le code de la fonction `eval` mais qui retourne cette fois-ci la valeur de l'évaluation ainsi que le meilleur coup à jouer.

Question 11

Modifier la fonction `main` de la section précédente pour que l'humain puisse jouer contre la machine.

3 Améliorations

Question 12

Améliorer la fonction heuristique pour qu'elle estime la position selon le principe suivant. On considère le score d'un joueur en considérant les alignements encore réalisables pour lui, et en les pondérant ainsi :

- 1pt : s'il a déjà placé 1 jeton sur cet alignement
- 3pt : s'il a déjà placé 2 jetons sur cet alignement
- 6pt : s'il a déjà placé 3 jetons sur cet alignement

L'heuristique est égale à la différence de score des deux joueurs, ou $\pm\infty$ si l'un des joueurs a gagné.

Question 13

Implémenter l'élagage $\alpha - \beta$. On pourra afficher le nombre de nœuds explorés dans l'arbre avec et sans élagage.