~\Downloads\Puissance4(2).c

```c
/*jeu de puissance 4 en C*/
/*Emma CLUGERY*/
/*GROUPE 1A1*/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>

#define NBLIG 6
#define NBCOL 7
#define  PION_A 'X'
#define PION_B 'O'
#define VIDE '_'

const char INCONNU = ' ';
const int COLONNE_DEBUT = NBLIG/2;

typedef char grille[NBLIG][NBCOL];

void initGrille (grille);
void afficher (grille, char, int);
bool grillePleine (grille);
void jouer (grille, char, int*, int*);
int choisirColonne (grille, char, int);
int trouverLigne (grille, int);
bool estVainqueur (grille, int, int);
void finDePartie (char);

int main (){
    char vainqueur ;
    int ligne, colonne;
    grille g;

    initGrille (g);
    vainqueur = INCONNU;
    afficher (g, PION_A, COLONNE_DEBUT);

   while ((vainqueur == INCONNU) && !(grillePleine(g)))
    {
        jouer(g, PION_A, &ligne, &colonne);
        afficher(g, PION_B, COLONNE_DEBUT);
        if (estVainqueur(g, ligne, colonne))
        {
            vainqueur = PION_A;
        }
        else if (! grillePleine(g))
        {
            jouer(g, PION_B, &ligne, &colonne);
```

```c
            afficher(g, PION_A, COLONNE_DEBUT);
                if (estVainqueur(g, ligne, colonne))
                {
                    vainqueur = PION_B;
                }

        }


    }
    finDePartie(vainqueur);
return EXIT_SUCCESS;
}

// Fonctions et Procédures

void initGrille (grille g){
    int i;
    int j;
    for (i=0; i < NBLIG; i++){
        for(j=0; j < NBCOL; j++){
            g[i][j] = VIDE;
        }
    }
}

void afficher (grille g, char pion, int colonne ){
    system("clear");
    int i;
    int j;


    printf("Joueur %c a vous de jouer\n", pion);
    printf("\n");
    printf(" 1 2 3 4 5 6 7 \n");
    printf("                \n");
    for (int i = 0; i < colonne; i++){
        printf("  ");
    }
    printf("%2c\n", pion);

    for (i = 0; i < NBLIG; i++)
    {
        printf("|");

        for (j= 0; j < NBCOL; j++)
    {
        printf("%c", g[i][j]);
        printf("|");
    }
    printf("\n");
    }


}
```

```c

bool grillePleine(grille g){
    int i, j;
    bool plein;
    plein = true;
    for (i = 0; i < NBLIG; i++)
    {
        for (j = 0; j < NBCOL; j++)
        {
            if (g[i][j]== VIDE)
            {
                plein = false;
            }

        }

    }
    return plein;
}

void jouer(grille g, char pion, int* ligne, int* colonne){
   *colonne = choisirColonne(g, pion, COLONNE_DEBUT);
   *ligne = trouverLigne(g, *colonne);
    while ((*ligne) == -1) {
        *ligne = trouverLigne(g, *colonne);
    }
    g[*ligne][*colonne] = pion;
}

int choisirColonne(grille g, char pion, int colonne){
    char act, rc;
    char espace;
    char message[150];
    espace = ' ';
    int indice = colonne;

    scanf("%c%c", &act, &rc);
    while (act != espace || trouverLigne(g, indice) == -1){
        strcpy(message, "");
        switch (act){
        case 'q':
            if (indice > 0){
                indice = indice - 1;
            } else {
                strcpy(message, "Erreur ! Vous ne pouvez pas aller au delà, Veuillez choisir entre les 7 colonnes. \n");
            }
            break;
        case 'd':
            if (indice < NBCOL - 1){
                indice = indice + 1;
            } else {
                strcpy(message, "Erreur ! Vous ne pouvez pas aller au delà, Veuillez choisir entre les 7 colonnes. \n");
            }
```

```
160             break;
161         case ' ':
162             if (trouverLigne(g, indice) == -1){
163                 strcpy(message, "Erreur ! Colonne Pleine .\n");
164             }
165             break;
166         default:
167            strcpy(message, "Entrez q pour aller à gauche, Entrez d pour aller à droite, Et
    faites espace pour valider votre choix.\n");
168             break;
169         }
170         afficher(g, pion, indice);
171         printf("%s\n", message);
172         scanf("%c%c", &act, &rc);
173     }
174     return indice;
175 }
176 
177 int trouverLigne(grille g, int colonne){
178     int ligne = -1;
179 
180     while (ligne < NBLIG && g[ligne+1][colonne] == VIDE){
181         ligne = ligne + 1;
182     }
183     return ligne;
184 }
185 
186 bool estVainqueur(grille g, int ligne, int colonne){
187     bool vainqueur;
188     int i, aligne, alignMax;
189     alignMax = 1;
190     aligne = 1;
191     i = 1;
192     vainqueur = false;
193 
194     // Vérification d'une ligne verticale
195     while (ligne+i < NBLIG && aligne < 4 && g[ligne+i][colonne] == g[ligne][colonne]){
196         i++;
197         aligne++;
198     }
199      if (aligne > alignMax){
200         alignMax = aligne;
201     }
202 
203     // Vérification d'une ligne horizontale
204     aligne = 1;
205     i = 1;
206     while (colonne+i < NBCOL && aligne < 4 && g[ligne][colonne+i] == g[ligne][colonne]){
207         i++;
208         aligne++;
209     }
210 
211     i = 1;
212     while (colonne-i >= 0 && aligne < 4 && g[ligne][colonne-i] == g[ligne][colonne]){
213         i++;
```

```
214         aligne++;
215     }
216      if (aligne > alignMax){
217         alignMax = aligne;
218     }
219 
220     // Vérification d'une ligne en diagonale en bas à droite vers le haut à gauche
221     aligne = 1;
222     i = 1;
223     while (colonne+i < NBCOL && ligne-i >= 0 && aligne < 4 && g[ligne-i][colonne+i] ==
g[ligne][colonne]){
224         i++;
225         aligne++;
226     }
227 
228     i = 1;
229     while (colonne-i >= 0 && ligne+i < NBLIG && aligne < 4 && g[ligne+i][colonne-i] ==
g[ligne][colonne]){
230         i++;
231         aligne++;
232     }
233      if (aligne > alignMax){
234         alignMax = aligne;
235     }
236 
237     // Vérification d'une ligne en diagonale en bas à gauche vers le haut à droite
238     aligne = 1;
239     i = 1;
240     while (colonne-i >= 0 && ligne-i >= 0 && aligne < 4 && g[ligne-i][colonne-i] ==
g[ligne][colonne]){
241         i++;
242         aligne++;
243     }
244 
245     i = 1;
246     while (colonne+i < NBCOL && ligne+i < NBLIG && aligne < 4 && g[ligne+i][colonne+i] ==
g[ligne][colonne]){
247         i++;
248         aligne++;
249     }
250     if (aligne > alignMax){
251         alignMax = aligne;
252     }
253 
254 
255     if (alignMax >= 4) {
256         vainqueur = true;
257         return vainqueur;
258     } else {
259         vainqueur = false;
260         return vainqueur;
261     }
262 }
263 
264 void finDePartie(char pion) {   //procédure de fin de partie
265     printf("\n");
```

```
266     switch (pion) {
267         case 'X':
268             printf("Le joueur %c a gagné !! \n", pion);
269             break;
270         case 'O':
271             printf("Le joueur %c a gagné !! \n", pion);
272             break;
273         default:
274             printf("Match nul ! \n");
275             break;
276     }
277 }
278
279
```