Projet Programmation – USALP1 DUT 1 – Puissance 4

P.Courtieu

Résumé

Programmation du jeu de puissance 4 en utilisant un tableau à double entrée pour représenter en mémoire l'état du jeu. Plus d'information sur le projet sur la page du projet.

1 Règles du jeu

Les règles du jeu sont simples et expliquées ici, cité partiellement ci-dessous.

> « Le but du jeu est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur dispose de 21 pions d'une couleur (par convention, en général jaune ou rouge). Tour à tour, les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans la dite colonne à la suite de quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle. »

Fonctionnalités minimales Pour avoir la moyenne, vous devez implanter parfaitement les fonctionnalités suivantes :

- affichage de la grille à chaque tour,
- alternance des joueurs (`X` et `O`),
- pions placés correctement dans la grille,
- détection de la grille pleine (partie nulle).

Voir également la section 4 pour des contraintes sur la forme des entrées.

Points bonus Des bonus seront accordés pour les points suivants (plus ou mois par ordre de difficulté) :

- sortie du jeu : si un joueur tape `0` au lieu d'un numéro de colonne, le jeu imprime une dernière fois l'état du jeu et s'arrête ;
- condition de victoire correcte pour `NBC`=6 et `NBL`=7 ;
- undo : si le joueur tape `-1` le jeu revient en arrière d'un coup dans la partie (suggestion : garder en mémoire la suite des coups joués (max 42 coups), soit pour « déjouer » le dernier coup, soit pour rejouer la partie depuis le début sauf le dernier) ;
- condition de victoire correcte pour toute valeur de `NBC` et `NBL`.

Dates de rendus Un rendu intermédiaire et un rendu final sont demandés :

- 10 décembre 2020 : Une première version effectuant l'affichage correct du damier initial (vide) et demandant les numéros de colonne successivement aux deux joueurs. L'insertion des pions n'a pas à être correcte à ce stade, ni les points bonus.
- 15 janvier : Version finale.

2 Détails d'implantation

Le déroulement d'une partie est très comparable à celui du morpion, sauf que les joueurs tapent un numéro de *colonne* (au lieu d'un numéro de case) dans laquelle il désire insérer son pion. Le jeu calcule ensuite dans quelle case le pion « tombe ». Le test de victoire est plus compliqué que celui du morpion, vous pouvez néanmoins vous inspirer de la méthode utilisée dans le morpion fourni dans le squelette (voir plus bas).

Pour représenter en mémoire l'état du jeu pendant l'exécution du programme, vous utiliserez un tableau de **`char`** à double entrée (voir également le morpion).

3 S'inspirer du morpion

Remarque : cette section et les suivantes font également l'objet d'explications en vidéo sur la page du projet.

Vous disposez dans le squelette d'une version du morpion programmée avec un tableau à double entrée. Vous pouvez vous inspirer de ce code.

4 Format d'entrées-sorties

Ceci concerne entre autre la façon dont vous serez évalués.

Le fonctionnement de votre programme doit strictement respecter certaines règles. Un manquement à ces règles donnera automatiquement la note de zéro.

Des scripts de tests seront bientôt fournis afin de vous permettre de vérifier que vous êtes conformes.

4.1 Les entrées

Voir également la vidéo Testabilité à ce sujet sur la page du projet.

Les seules entrées lues au clavier sont les coups successifs des joueurs. Rien d'autre. Il est interdit de demander autre chose à l'utilisateur. Par exemple votre programme ne doit **pas** demander les noms des joueurs ou autre fioriture.

4.2 Les sorties

Le format de sortie (l'affichage de la grille et les messages) est contraint aussi :

— Ne changez absolument pas la fonction d'affichage de la grille.

— pour signaler une victoire vous devez afficher un message contenant exactement la chaîne `"X gagne"` ou bien `"O gagne"`. Cette chaîne peut faire partie d'un message plus long (ex : `"Le joueur O gagne!"`).

— pour signaler une partie nulle vous devez afficher un message contenant exactement `"partie nulle"`. Cette chaîne peut faire partie d'un message plus long (ex : `"Fin: partie nulle!"`).

4.3 Tests automatiques

Grâce à ces contraintes vous pourrez tous utiliser les fichiers de tests que nous vous fournirons (ainsi que ceux que vous écrirez vous même). Voici comment procéder :

Pour tester votre programme vous pouvez lui faire « jouer » des parties en lisant les coups des joueurs dans un fichier plutôt qu'au clavier. Pour cela il faut rediriger l'entrée standard du programme vers le fichier en utilisant l'opérateur shell de redirection : <. Par exemple :

```
./puissance4 < partiePuissance4.txt
```

En lançant le programme de cette façon les coups sont lus dans le fichier `partiePuissance4.txt` plutôt qu'au clavier.

C'est très utile pour « rejouer » une partie rapidement. Vous pouvez (c'est même fortement recommandé) écrire d'autres jeux de test pour vous même. En particulier lorsque vous modifiez votre programme cela permet de vérifier rapidement que les fonctionnalités qui marchaient avant les modifications ne sont pas « cassées ».

Remarque importante : L'évaluation de votre projet se fera avec cette technique de test avec un nombre important de tests. Il faut donc que votre programme respecte rigoureusement le protocole. **Un projet qui ne respecte pas les contraintes d'entrées/sorties décrites plus haut aura la note zéro.**

5 Makefile

Pour vous permettre de compiler facilement, vous disposez dans le squelette d'un fichier `Makefile` permettant de compiler le programme principal (`puissance4`) en tapant dans un terminal :

```
make
```

et le morpion en tapant :

```
make morpion
```

Il faut avoir la commande `make` installée, c'est en général le cas par défaut dans tous les systèmes linux et macos et dans tous les environnements pseudo-linux (`mingw`, `cygwin`, sous-système windows pour linux etc) sous windows également.

6 `#define`

Dans ce projet nous utiliserons la *commande de pré-processeur* **`#define`** qui permet de définir des macros. De la même façon que nous écrivons (depuis le début du cours) `TRUE` plutôt que 1 lorsqu'il s'agit d'un résultat de test, nous écrirons `NBC` et `NBL` pour désigner les nombres de colonnes et de lignes de la grille. En effet recopier des constantes partout dans les programmes est considéré comme une mauvaise pratique : elle deviennent très difficiles à changer. Ceci est également expliqué dans la vidéo sur le morpion.

On utilisera aussi des macros pour les 3 symboles pouvant apparaître dans la grille.

L'en-tête du fichier `puissance4.c` est donc celui-ci :

```
#include"inout.h"
#include<stdlib.h>
#define BOOL int
#define TRUE 1
#define FALSE 0
// Alias à utiliser partout pour éviter de mettre 6 et 7 partout dans le code
#define NBL 6 // Nombre de ligne du puissance 4
#define NBC 7 // Nombre de colonnes du puissance 4
#define COCHEVIDE ' '
#define COCHEX 'X'
#define COCHEO 'O'
```

Rappel sur `BOOL` : Attention ne faites ***jamais***

```
if (xxx == TRUE){ // équivalent à if (xxx == 1)
...
}
```

mais plutôt :

```
if (XXX){
...
}
```

En effet un test positif (par exemple `(x>0)`) retourne une valeur *quelconque* non nulle, donc pas forcément 1 `TRUE`). Symétriquement ne faites pas **`if`** `(xxx==FALSE)` mais **`if`** `(!XXX)`.