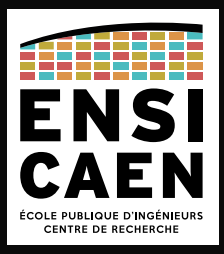

Outils de développement logiciel

TP n° 9 : Puissance 4

Alain Lebret

2023-2024

Objectif

Créer et utiliser des bibliothèques statiques et dynamiques.
Pré-requis : chapitre 9
Durée estimée : 1 séance

1 Introduction

Puissance 4 (en anglais ***Connect Four***) est un jeu de stratégie dans la grande tradition du morpion. Il se présente sous la forme d'une grille de 7 colonnes sur 6 lignes et de deux jeux de pions de couleurs différentes. La grille doit être posée verticalement de sorte que les pions s'introduisent par le haut de la grille dans une des 7 fentes. Le pion ainsi introduit tombe soit sur le fond de la grille (ligne=6), si la colonne est vide, soit sur un autre pion précédemment introduit (ligne<6) comme le montre la figure 1.

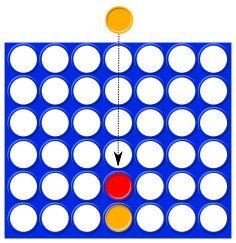

Figure 1: Puissance 4

Les règles sont les suivantes :

1. À tour de rôle, chaque joueur fait tomber un de ses pions dans une des fentes du sommet des colonnes (non déjà pleines) de la grille.
2. Le jeu continue jusqu'à ce que l'un des joueurs ait une ligne continue de 4 pions de sa couleur. L'alignement peut être vertical, horizontal ou diagonal (figure 2).
3. Le premier joueur qui a fait une ligne continue de 4 pions gagne la partie.

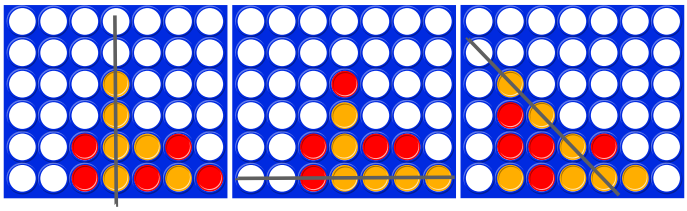

Figure 2: Puissance 4 : exemples de coups gagnants.

Le programme permet de visualiser le déroulement du jeu entre 2 joueurs humain. La machine se contente de saisir le coup de chaque joueur, de vérifier qu'il est valide, de mettre le pion à sa place dans la grille, et enfin de reconnaître quand un joueur a gagné.

2 Organisation du projet

Le projet est découpé en deux parties :

1. La construction d'une bibliothèque de gestion de matrice de caractères.
2. La construction du programme de puissance 4 qui utilise la bibliothèque précédente.

Le projet total est construit dans un dossier organisé comme suit :

```
├─ odl
│  |
│  ├─ tp09
│  |   ├─ README.md
│  |   ├─ Makefile
│  |   |
│  |   ├─ bin/
|  |   |   ├─ connect4
|  |   |   ├─ test/
|  |   |   |    ├─ test_matrix
|  |   |   |    ├─ test_game
│  |   ├─ doc/
|  |   |   ├─ Doxyfile
│  |   ├─ include/
|  |   |   ├─ matrix.h
|  |   |   ├─ game.h
│  |   ├─ lib/
|  |   |   ├─ libmatrix.a
|  |   |   ├─ libmatrix.so
│  |   ├─ src/
|  |   |   ├─ matrix/
|  |   |   |    ├─ Makefile
|  |   |   |    ├─ matrix.c
|  |   |   ├─ connect4/
|  |   |   |    ├─ Makefile
|  |   |   |    ├─ game.c
|  |   |   |    ├─ main.c
│  |   ├─ tests/
```

```
|   |   |   ├─ matrix/
|   |   |   |    ├─ Makefile
|   |   |   |    ├─ test_matrix.c
|   |   |   ├─ connect4/
|   |   |   |    ├─ Makefile
|   |   |   |    ├─ test_game.c
|   |   |   |
```

Construction d'une bibliothèque de gestion de matrices

Le fichier "include/matrix.h" détient la définition du type matrice :

```c
/**
 * @brief A 2D matrix.
 */
struct matrix_t {
    char **data;      /*!< The concrete matrix data. */
    unsigned rows;    /*!< The number of rows.       */
    unsigned columns; /*!< The number of columns.    */
};
```

Selon cette définition, la déclaration d'une matrice dans une fonction se fait par :

```c
struct matrix_t m;
```

Le fichier contient aussi la déclaration de tous les prototypes des fonctions de gestion des matrices.

Le dossier "src/matrix" contient le fichier C définissant les trois fonctions suivantes :

- Une fonction qui alloue dynamiquement une matrice à partir du nombre de lignes et de colonnes :

```c
struct matrix_t alloc_matrix(unsigned rows, unsigned columns);
```

- Une fonction qui initialise une matrice avec une valeur :

```c
void init_matrix(struct matrix_t *m, char value);
```

- Une fonction qui efface le terminal puis affiche la matrice :

```c
void display_matrix(struct matrix_t *m);
```

Le fichier "tests/matrix/Makefile" contient la cible "test_matrix" et produit un programme de test à partir du fichier "tests/matrix/test_matrix.c". La production de la cible se fera de cette manière :

```
$ make test_matrix # pour les tests avec la bibliothèque
```

Le fichier "src/matrix/Makefile" contient les cibles "staticlib" et "dynamiclib" qui permettront de produire les bibliothèques "lib/libmatrix.a" et "lib/libmatrix.so", respectivement statique et dynamique, contenant chacune les fonctions de gestion des matrices. La production des bibliothèques se fera de cette manière :

```
$ make staticlib  # construction de la bibliothèque statique
$ make dynamiclib # construction de la bibliothèque dynamique
```

Construction du programme du jeu puissance 4

Pour le jeu, on choisit trois caractères : "`.`" (`DOT_KEY`) pour une case vide, "`*`" (`TIMES_KEY`) pour le joueur n° 1 et "x" (`X_KEY`) pour le joueur n° 2.

Le dossier "src/connect4" contient les fichiers ".c" qui définissent les fonctions suivantes :

- Une fonction qui teste si à partir d'une case (***row***, ***column***), il y a quatre pions du même joueur alignés, à droite, en haut, en bas, à gauche ou sur les deux diagonales dont la case est un élément de l'alignement.

```c
int check_winner(struct matrix_t m, unsigned row, unsigned column);
```

Cette fonction appelle 4 fonctions spécialisées. Par exemple, la fonction suivante est appelée par `check_winner()` pour tester s'il y a 4 pions alignés sur la ligne ***row***. Elle fonctionne en calculant les coordonnées du pion le plus à gauche du pion donné et qui possède la même couleur (ici le même caractère), ainsi que le pion le plus à droite :

```
int check_horizontal(struct matrix_t board, unsigned row, unsigned column, char
↪  pawn);
```

Il suffit d'appliquer le même principe pour la colonne et les deux diagonales autour de la case (***row***, ***column***).

- Une fonction qui effectue un tour de jeu pour un des deux joueurs.

```
void play(struct matrix_t m, int player);
```

- Un programme principal qui construit la grille de jeu et lance une partie.

Chaque fonction écrite sera testée individuellement à partir du fichier "test_game.c" qui vous est fourni.

Le fichier "tests/connect4/Makefile" contient la cible suivante :

- `test_connect4` : pour les tests.

Le fichier "src/connect4/Makefile" contient la cible suivante :

- `connect4` : pour construire le jeu.

Le fichier global "Makefile" appelle les fichiers "Makefile" des dossiers "tests" et "src" et contient les cibles principales suivantes :

- `tests` : construire les tests de la bibliothèque et du jeu.
- `all` : construire tout le projet (bibliothèques et programme final).

3 Travail à réaliser

1. Placez-vous dans votre dossier "odl" et créez le sous-dossier "tp09", puis placez-vous dans ce dernier. "tp09" sera votre dossier de travail pour cette séance et vous y effectuerez toutes vos actions.
2. Copiez l'ensemble des fichiers ".c" et ".h" du dossier "ressources-odl-fisa/tp09/" dans votre dossier de travail en respectant la structure arborescente.
3. Complétez si nécessaire au fur et à mesure les Makefiles et les différentes fonctions en les validant. Une fois l'ensemble des tests validés, vous pourrez compléter le jeu.

4 Livrable

Déposez sur la plateforme Moodle l'archive compressée "tp09-nom1-nom2.tgz" qui contiendra l'ensemble du dossier "tp09", excepté les fichiers objets, les bibliothèques et les exécutables, ou encore la documentation générée par ***Doxygen***.

Vous penserez à inclure dans le dossier "tp09" un fichier "README.md" qui :

- identifie l'auteur du TP ;
- décrit brièvement le contenu du dossier (objectif et composants) ;
- donne les instructions pour compiler à l'aide de la commande ***make*** les programmes de test ainsi que les bibliothèques et le programme final.

5 Résumé

Dans ce TP vous avez réalisé et mis en oeuvre une bibliothèque statique et sa version dynamique. Vous avez aussi écrit des fichiers "Makefile" plus complexes et mis au point vos fonctions sous contrainte de validation par des tests unitaires.