EFREI Paris

Projet de C S2

Promotion 2023

Board Games

Auteurs :
Nathan ROUSSELOT

Professeurs :
Kamel CHABCHOUB
Christopher MARSHALL-BRETON

3 avril 2019

Table des matières

1 Introduction et présentation du projet **2**

2 Structure du projet **3**

3 Explication du code **4**

3.1 Le Tic-Tac-Toe 4

3.1.1 print-board() 4

3.1.2 init-board() 4

3.1.3 play() 4

3.1.4 has-win() 4

3.1.5 rand-a-b() 5

3.1.6 easy-AI() 5

3.1.7 Améliorations possibles 5

3.2 Le Puissance 4 6

3.2.1 is-full() 6

3.2.2 has-won() 6

3.2.3 has-won-diff() 6

3.3 Le Tron 7

3.3.1 play() 7

3.3.2 init-player() 7

3.3.3 replay() 8

4 Conclusion **8**

1 Introduction et présentation du projet

Lors de ce premier projet de C, j'ai réaliser trois jeux de plateaux :
- Un Tic Tac Toe
- Un Puissance 4
- Un Tron

Le premier est un simple jeu, très connu, qui consiste à aligner trois 'X' ou trois 'O' pour gagner. Il se joue sur un tableau 3*3. Le Tic-Tac-Toe dans ce projet ne se joue qu'à un joueur, contre un ordinateur (qui joue au hasard).

Le deuxième est un jeu également très connu. Il se déroule sur un board de 6 lignes et 7 colonnes. Ce jeu, en joueur contre joueur, consiste à placer un jeton dans une colonne donnée. Il tombera alors en bas de celle-ci. L'objectif est d'aligner, en diagonal, en ligne ou en colonne 4 de ses jetons (Bleu ou Rouge).

Enfin, le dernier jeu, Tron est un très vieux jeu. Vous vous déplacez dans un board de 21 par 21. Au fur et à mesure que vous vous déplacez, votre trajectoire devient un obstacle. Si vous vous avancez sur une case parmis votre trajectoire ou celle de l'adversaire, vous perdez. Pareillement si vous sortez du Board.

Une fois la partie terminé, un replay accéléré de la game est affiché.

2 Structure du projet

Voici l'arborescence du projet :

```
‘- Board-Games/
   +- tic-tac-toe/
   |  +- tic-tac-toe-main.c
   |  +- tic-tac-toe_game.c
   |  +- tic-tac-toe_game.h
   |  +- terminal_linux.c
   |  +- terminal_linux.h
   +- power-4/
   |  +- power-4.c
   |  +- power-4-game.c
   |  +- power-4-game.h
   |  +- terminal_linux.c
   |  +- terminal_linux.h
   +- tron/
   |  +- tron.c
   |  +- tron-game.c
   |  +- tron-game.h
   |  +- terminal_linux.c
   |  +- terminal_linux.h
   ‘- README
```

Comme vous pouvez le voir, tout a été bien séparé. Les fonctions sont définies dans tous les fichiers [nomdujeu]-game.c et sont appelées pars leurs headers respectifs.

Bien sûr je n'ai pas pu rendu le projet comme tel, mais c'est ainsi qu'il est stocké sur GitHub et sur mon PC.

Enfin, je n'ai pas utilisé conio, mais bien terminal linux, ainsi pour lancer mon projet, privilégiez Linux à Windows / MacOS. Je ne l'ai pas testé sur Plan9 / BSD.

3 Explication du code

3.1 Le Tic-Tac-Toe

3.1.1 print-board()

Le print-board est une fonction très basique, qui prend simplement en paramètre un board, le numéro du round, et le nom du joueur.

```
1    void print_board(struct Board board, int round, char name[]);
```

3.1.2 init-board()

Cette fonction initialise tout simplement les différents champs de la structure Board :

```
1    struct Board
2    {
3        int size;
4        char** cells;
5    };
```

Ainsi cette fonction réalise dynamiquement les cells.

```
1    struct Board init_board(int n);
```

3.1.3 play()

La fonction play ne réalise aucune opération particulière. Il sagit d'un scanf sécurisé un peu amélioré.

```
1    struct Board play(struct Board board);
```

3.1.4 has-win()

Cette fonction est peut être la plus longue du dossier. Mais également la plus simple à comprendre. Il sagit juste d'un enchaînement de test avec des if / else if pour tester toutes les combinaisons possible et conclure si un joueur à gagner ou non.

```
1    int has_win(struct Board board);
```

3.1.5 rand-a-b()

La fonction rand-a-b réalise une pioche aléatoire entre a et b. Comme elle est très courte, nous allons pouvoir l'analyser un peu plus en détail :

```
1     int rand_a_b(int a, int b) {
2         return rand()%(b-a) + a;
3     }
```

Comme vous pouvez le voir, cette fonction est un peu plus poussée que le rand que nous utilisons habituellement. Le modulo de (b-a), avec l'ajout de la borne inférieure a nous permet d'avoir un nombre aléatoire réellement en a et b, et pas entre 0 et a.

Bien évidemment il faut coupler cette fonction avec un SRAND(TIME=NULL) afin d'avoir un véritable random.

3.1.6 easy-AI()

Cette fonction réalise une "AI" très basique : aléatoire ! On utilise donc simplement notre rand-a-b précédemment décrite.

```
1     struct Board easy_AI(struct Board board);
```

3.1.7 Améliorations possibles

Etant donné la simplicité du jeu, nous pourrions réaliser une AI totalement imbattable. Cela prendrait un peu plus de temps mais est tout à fait faisable.

Par ailleurs, la fonction has-win peut être légérement optimisée.

Enfin, il est possible de styliser un peu le menu et le jeu.

3.2 Le Puissance 4

Les fonctions play, init-board, print-board sont très similaires au Tic-Tac-Toe, je ne vais donc pas les réexpliquer ici.

3.2.1 is-full()

La fonction is-full() va vérifier si une colonne particulière est pleine :

```
1 int is_full(struct Board board, int column) {
2     //This function tests whether a column is full or not.
3     if (board.cells[0][column]=='1'||board.cells[0][column]=='2')
      {
4         return 1;
5     }
6     return 0;
7 }
```

Comme vous pouvez le voir, elle regarde si la case en haut de la colonne en paramètre est occupé par le joueur 1 ou 2.

3.2.2 has-won()

La fonction has-won est une version un peu améliorer d'un parcours classique. Elle va simplement regarder autour du dernier jeton placé afin d'économiser le nombre d'itérations.

```
1     int has_won(struct Board board);
```

3.2.3 has-won-diff()

Le fonction has-won-diff est une fonction complémentaire à has-won, qui permet de prendre en compte certains cas, incompatible avec has-won.

```
1     int has_won_diff(struct Board board);
```

3.3 Le Tron

Les fonction print-board et init-board sont assez similaires aux précédentes, ainsi je ne les réexpliquer ici.

3.3.1 play()

La fonction play() est une fonction assez complexe. C'est ici le coeur du programme. Tout d'abord analysons son prototype :

```
1    struct Board play(struct Board board, struct Player * player);
```

Comme pour les ancienne play(), elle return un type Board. Cependant, notion nouvelle, il y a un type Player (que nous détaillerons dans le 3.3.2).

En paramètre il y a donc l'adresse d'un Player, afin de le manipuler comme une donnée modifiée.

La structure globale reste une grosse saisie sécurisée sur Z, Q, S et D, et si jamais le joueur perd, player.alive passe à 0.

3.3.2 init-player()

Voici le type Player :

```
1    struct Player
2    {
3     int id;
4     int last_row;
5     int last_col;
6     int alive;
7    };
```

Ainsi le fonction init-player va simplement créer un Player en remplissant tous ces champs.

```
1    struct Player init_player(int id, int col);
```

3.3.3 replay()

La fonction replay va se charger de rejouer, de manière rapide la partie qui vient de se terminer. Pour son fonctionnement j'ai donc créer un fichier log. Tous les déplacement sont stocké sur une ligne, et de la forme : xx,xx avec xx un entier. Il suffit donc d'utiliser sscanf pour en extraire la valeur, puis de la placer dans le board.

```
1    void replay();
```

4 Conclusion

En conclusion, la difficulté de ce projet était plus du côté optimisation que du côté algorithmique. En effet, une fois le Tic-Tac-Toe terminé, si les fonctions étaient rigouresement définies, alors les autres jeux ne nécessitaient que très peu de modifications.

Merci d'avoir lu mon rapport.