Rapport de projet « Programmation Impérative »
Le Morpion

C. DEFRETIERE

11 mai 2019

Résumé

Dans le cadre de l'unité d'enseignement « Programmation Impérative », ce rapport traitera du jeu nommé « Morpion »et surtout de sa conception. Le but est d'expliciter l'implémentation de ce jeu réalisée en C, et de commenter la progression de ce projet. Il ne s'agit pas, dans ce rapport, d'expliquer ligne par ligne le code, mais de saisir la philosophie du projet, afin de comprendre ensuite l'implémentation du jeu. Une explication détaillée d'une fonction par exemple, aura lieu si il y a besoin.

Bonne lecture

Table des matières

1 Mise en place

Ce programme est écrit en C respectant la norme ANSI, il est fortement recommandé d'utiliser un système linux.

1.1 Compilation

Un fichier "makefile" est fourni avec le projet,

La commande `$ make` permet de tout compiler,
et `$ make clean` permet d'effacer les fichiers binaires.

1.2 Utilisation

Pour jouer il suffit de lancer le fichier main créé à la suite de la compilation, entrez les coordonnées dans l'ordre **ligne puis colonne**, ligne étant un chiffre et colonne un caractère minuscule.
Ce qui donne par exemple : `5b`

Si le coup n'est pas valide, le programme redemande une saisie, malheureusement il y a parfois plantage. (problème avec l'utilisation de scanf ?)

Pour changer les paramètre du jeu, ouvrez votre éditeur de texte favori et modifiez la fonction "newMorpion" au début du fichier `game.c`.

2 Structure

2.1 main

Le fichier `main.c` n'a besoin que du module `game`, se dernier utilise des fonctions très simple telles que :

newMorpion Permet de creer un nouveau jeu

waitPlayer Fait jouer le joueur

play Fait jouer l'ordinateur

checkWinner Qui renvoie le gagnant

Il y a également, mis à la disposition par `game`, des contantes : PLAYER, COMPUTER et EMPTY.
Ainsi, `main.c` n'a besoin que de faire une boucle de jeu très simple. En effet, toutes les fonctions utilisées dans `main.c` n'ont pour seul paramètre l'objet "Morpion" initialisé au tout début par "newMorpion". Cette manière de programmer qui peut faire penser à l'orienté objet est très pratique.

2.2 game

Le module `game` est le cœur de ce projet, il implémente toutes les fonctions et structures précédemment citées. Les fonctions auxiliaires sont signalées par un caractère '_' a.k.a. "underscore".

La structure "Morpion" est l'équivalent d'un objet. Il permet de transporter toutes les informations relatives à un jeu très facilement.

```
typedef struct Morpion{
    int size, winCondition;
    int* board;
    int depthLimit;
} Morpion;
```

"Size" correspond à la taille du coté du plateau, "winCondition" au nombre de symboles alignés nécessaires pour gagner, "board" au stockage du plateau et "depthLimit" à la profondeur maximum de l'arbre.
Comme vous pouvez le constater, le jeu est très modulable, des directives précisent également d'autres paramètres comme PLAYER, COMPUTER et EMPTY qui sont les caractères ASCII utilisés pour l'affichage.

N.B. Si "depthLimit" est mis à zero, le programme considère que l'ordinateur doit jouer en mode aléatoire.

3 Plus en détail

3.1 La vérification du gagnant

La vérification du gagnant est plutôt simple :
Le programme parcours chacune des cases du tableau, si il tombe sur une joueur **j** quelconque il regarde dans chacune des directions le nombre de symboles de **j** consécutifs. Si un de ces nombres est supérieur ou égal à "winCondition" précédemment cité, **j** est gagnant.

Pour "regarder" dans chaque direction, on utilise un peu de trigonométrie. Une variable angle fait un tour de cercle, par pas de PI/4. On récupère le cosinus et sinus de cet angle et en l'arrondissant on obtient le déplacement horizontal et vertical à effectuer pour chaque case autour.

3 * PI/4	2 * PI/4	PI/4
4 * PI/4	X	0
5 * PI/4	6 * PI/4	7 * PI/4

3.2 Évaluation du plateau

Afin de pouvoir arrêter l'arbre à une certaine profondeur, il est nécessaire de créer une fonction d'évaluation.
Pour ma part j'ai choisis de faire le même parcours que pour la fonction de vérification du gagnant, mais ici, à chaque fois que des symboles sont alignés, on ajoute ou enlève des points selon le joueur. Plus le nombre de symboles alignés est grand, plus il faut attribuer de points, l'attribution des points se fait donc suivant une fonction linéaire de coefficient SEQ_FACTOR définit dans `game.h`.

Cette fonction pose un problème, en effet imaginons un ligne de 3 croix, le programme commence par lire la première, trouve un alignement à sa droite de longueur 3 et attribue le nombre de points suivant :

$$1 * SEQ\_FACTOR + 2 * SEQ\_FACTOR + 3 * SEQ\_FACTOR$$

Il passe ensuite sur la deuxième croix, trouve un alignement à gauche et un alignement à droite de longueur 2 :

Gauche : $1 * SEQ\_FACTOR + 2 * SEQ\_FACTOR$

Droite : $1 * SEQ\_FACTOR + 2 * SEQ\_FACTOR$

Finalement, il continue sur la troisième croix, et trouve un alignement de longueur 3 à sa gauche :

$$1 * SEQ\_FACTOR + 2 * SEQ\_FACTOR + 3 * SEQ\_FACTOR$$

Au final, le programme comptabilise plus de ligne qu'il ne devrait, mais cette erreur nous arrange un peu. En effet cela augmente la valeur des alignements de manière considérable, l'I.A. essaiera donc d'aligner le plus de symboles et trouvera dangereux ce genre de placement chez l'adversaire.

3.3 Arbre de jeu

L'arbre de jeu permet de modéliser l'ensemble des coups possibles, voici la structure :

```
typedef struct Tree{
    int value;
    struct Tree** children;
    int nChildren;
    int boardIndex;
} Tree;
```

"value" permet de stocker la valeur retenue par un nœud, le minimum ou maximum des enfants stockés dans "children", ou l'évaluation du plateau si il s'agit d'une feuille. "nChildren"correspond au nombre d'enfants, et "boardIndex" s'agit de l'index du symbole changé par le nœud courant.

Une fonction récursive remplie cet arbre ; pour chaque profondeur, un seul plateau est alloué en mémoire, chaque nœud de la même profondeur se le faisant passer l'un après l'autre. Une fois les feuilles de l'arbre construites, tous les plateaux potentiels se détruisent, il ne reste dans l'arbre que "boardIndex", qui enregistre le changement d'un parent à son fils, cette stratégie est inspiré du versioning Git.

Une fois l'arbre construit, il suffit de prendre le maximum parmi les enfants de la racine et de jouer la case "boardIndex".