BELLIK Nassim
NASS Michaël

Rapport de projet

Tetris

BELLIK Nassim
NASS Michaël

Sommaire

BELLIK Nassim
NASS Michaël

Introduction

Lors de ce 2ème semestre, nous avons appris le langage C. Ce langage est la base d'autres langage dérivant directement de sa structure (Java, C++, awk et Perl) dû à sa grande stabilité. Il rentre dans la programmation structuré (procédurale) de la famille de la programmation impérative (décrit de façon précise les séquences de commandes à exécuter). Il est généralement utilisé dans les systèmes embarqués.

Le problème donné à ce premier projet C est de programmer un jeu TETRIS intégralement. Il nous a été donné diverses fonctions pour mieux avancer lors de notre recherche via nos professeurs.

TETRIS est un des premiers jeux d'arcade créé. Il se joue seul et consiste à placer des blocs de sorte à faire que lorsqu'ils se combinent, le score du joueur augmente.

Il nous est demandé de programmer dans 3 fichiers : **main().c** qui représente le corps du programme en exécutant les fonctions, **projet1c.h** qui listes les fonctions existante du programme et **projet1c.c** qui contient toute les fonctions du programme.

De plus, il nous a été donné diverses fonctions pour mieux avancer lors de notre recherche via nos professeurs.

Nous allons à travers ce rapport vous dévoiler les parties les plus importantes de notre programme,

I. Le code

Nous avons commencé par établir la façon dont nous allions gérer les données du projet. Vu que nous étions entrain d'étudier les structures, on a estimé que c'était le meilleur outil pour pouvoir stocker les blocs des différents terrains. Nous avons donc créé une structure bloc de la forme suivante :

```
14
15  typedef struct Block {
16      int appartenance;
17      int** content;
18      int size;
19  }Block;
```

L'appartenance (0, 1, 2 ou 3) indique à quel terrain le bloc appartient. 0 est l'appartenance globale, 1 est l'appartenance au cercle, 2 au losange et 3 au triangle. Size indique la taille du bloc au format x*x. Content est le contenu du bloc au format tableau 2D d'entiers contenant des 0 pour du vide et des 1 pour les blocs.

Pour stocker nos blocs, on a décidé de créer une autre structure Blocks contenant un tableau de Block et sa taille.

```
21  typedef struct Blocks {
22      Block* blocks;
23      int size;
24  }Blocks;
```

Ensuite, nous avons inséré les fonctions essentielles données sur Moodle dans un fichier distinct, et nous avons apporté quelques modifications afin de mieux les faire fonctionner avec nos structures Block.

Pour obtenir la liste des blocs jouables, on appelle la fonction getStandardizedBlocks() avec comme argument la forme du plateau de jeu.

La fonction getStandardizedBlocks() appelle elle-même la fonction getBlocks() qui crée dynamiquement un tableau de Block dans une structure Blocks dont le contenu de chaque bloc est formaté en [5][5] car les blocs sont générés à la main dans le fichier blocs.c.

```
} else if (i == 19) {
    int content[5][5] = {
        {0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0},
        {1, 1, 0, 0, 0},
        {0, 1, 0, 0, 0},
        {0, 1, 1, 0, 0}
    };
    fillBlock(&s1, content);
```

Une fonction convertBlock() est alors chargée de convertir le format statique [5][5] au format désiré dynamiquement.

```
98  Block convertBlock(Block block) {
99      Block s1;
100     s1.size = block.size;
101     s1.appartenance = block.appartenance;
102     s1.content = create_2D_Array(block.size, block.size);
103     //printf("Size : %d", block.size);
104     int i, j, k;
105     int size = block.size;
106     int startpoint = 5 - size;
107     int endpoint = 5;
108
109     for (i = startpoint, k = 0; i < endpoint ; i++, k++) {
110         for (j=0; j < size; j++) {
111             s1.content[i-startpoint][j] = block.content[i][j];
112         }
113     }
114
115     return s1;
116
117 }
```

Pour imprimer un bloc à l'écran, on utilise la fonction printBlocker(), qui va encadrer le bloc de tirets pour mieux les afficher à la suite lorsque l'on l'appelle plusieurs fois de suite.

Une partie du code utilise des macro contenus dans le fichier global.h, mais nous ne les avons pas appliqué sur l'ensemble de notre programme :

```
13 #define INJOUABLE -1
14 #define LIBRE 0
15 #define OCCUPE 1
16
17 #define LIGNE 0
18 #define COLONNE 1
```

II. Les difficultés et problèmes

Nous avons été fortement ralentis par des bugs de compilation de notre programme. En effet, durant le développement du Tetris, nous avons découvert que le projet ne voulait pas se compiler sur le PC de Nassim, pour une raison que nous n'avons à ce jour toujours pas compris. Nous avons dû installer le sous-système Linux pour Windows afin de compiler et exécuter le programme avec gcc dans Ubuntu, faute de pouvoir le faire dans Visual Studio directement.

De plus, nous avions commencé une fonction qui permettait d'écraser le carré du plateau de jeu avec des -1 pour former le losange et le triangle, mais ces fichiers ont disparu de nos ordinateurs suite à une mise à jour, alors qu'elle était non-finalisée et donc pas push sur notre repo GitHub. Nous avons essayé de les reproduire à la dernière minute, mais sans succès, donc nous les avons retiré du programme.

Dans notre projet, il nous manque donc la génération des formes personnalisées bien que le code du programme soit prêt à fonctionner avec, il nous faudrait simplement insérer les -1 au bon endroit à la fin de la fonction initGame(). Les fonctions de calculs de score, de vérifications et d'annulations de lignes et de colonnes sont également implémentés, mais la fonction de décalage de lignes présentait des problèmes car nous ne pouvions pas la tester avec l'environnement final (pour empêcher les sorties des blocs hors du plateau).

Conclusion

Ainsi, malgré les difficultés rencontrés, nous avons réussi dans l'ensemble devoir demandé dans . Nous avons mis à profit nos connaissances sur les structures et les fonctions. en C.