ENSIIE

Initiation à la Programmation Impérative, Projet 2016

Morpion 3D

Vianney Touchard

Pour le 3 janvier 2017

Table des matières

1 Sujet **1**

2 Outils de développement **1**

3 Choix techniques **1**
3.1 Langue 1
3.2 Modules 2
3.3 La structure s_jeu 2
3.4 Les préfixes 2
3.5 Les booléens 2

4 Solutions apportées **2**
4.1 Le module console 2
4.2 Les flags 2
4.3 Les tests de victoire 2
4.4 L'aléatoire 3
4.5 Le tableau events 3

5 Manuel d'utilisation **3**

1 Sujet

Le projet de programmation du premier semestre à l'ENSIIE consiste à implémenter un morpion 3D (plutôt similaire à un puissance 4 d'ailleurs).

Le jeu se joue au tour par tour, en empilant des pions dans des piles sur un plateau carré de taille n*n choisie au début de partie. Le but du jeu est d'être le premier joueur à aligner n pièces.

Deux variantes sont jouables : la variante vue du dessus où le but est de former des lignes avec les dessus des piles, et la variante 3D où les pièces doivent être alignées dans l'espace.

2 Outils de développement

Par habitude, j'ai choisi de développer sous Windows avec l'IDE Code : :Blocks et l'implémentation de gcc pour Windows MinGW. J'ai également eu recours à l'outil de gestion de versions Git et à l'hébergement gratuit github. Il est ainsi facile de tester le code de façon transparente sous Linux et Windows en utilisant les mêmes commandes (`git`, `make`, `gcc`).

3 Choix techniques

3.1 Langue

La langue utilisée dans le code est l'anglais (pour avoir une homogénéité avec l'utilisation des fonctions standard qui sont nommées en anglais et en règle plus générale avec le monde de la programmation qui utilise plus communément cette langue).

Les commentaires sont en revanche écrits en Français (exception faite des `@requires`, `@assigns`, etc. imposés)

3.2 Modules

Dans le projet, les différentes déclarations et implémentations sont séparées dans des fichiers différents, qu'un makefile permet de compiler en une commande `make`.

Le module `pile` avait été implémenté lors d'un TP durant le semestre et a été réusilisé tel quel.

3.3 La structure s_jeu

Pour regrouper les variables de partie tout en permettant leur utilisation facilement au travers de plusieurs fonctions, j'ai décidé d'utiliser une structure `s_jeu` qui ne sera instanciée qu'une seule fois pour une partie. Les fonctions associées requièrent systématiquement un pointeur vers une instance de cette structure.

3.4 Les préfixes

Par souci de lisibilité, ces fonctions sont préfixées (par le préfixe `j_`). L'implémentation de la structure de pile suit également cette logique.

3.5 Les booléens

En C, le type `bool` est différemment implémenté (ou pas) selon les standards (voire Wikipedia). Ainsi, la solution choisie a été d'utiliser les entier pour cette utilisation (sachant que les tests `if(a!=0)` et `if(a)` sont équivalents)

4 Solutions apportées

4.1 Le module console

La gestion de la partie et l'affichage sont gérés indépendamment. Pour simplifier ce dernier, j'ai eu recours à un tableau de symboles ASCII qui est modifié à volonté et ensuite affiché ligne par ligne en une seule fois. Cela permet de séparer les différentes fonctions de dessin et d'appeler un affichage à n'importe quel moment. Le module `console` apporte ces fonctions.

Il apporte également les fonctions de saisie sécurisées pour demander à l'utilisateur un caractère ou un nombre. Celles-ci sont basées sur la description de l'utilisation de scanf avec des expressions régulières donnée ici.

4.2 Les flags

4.3 Les tests de victoire

Les tests de victoire sont très similaires entre les variantes.

Ils reposent sur une recherche dans toutes les directions possibles des pièces alignées à partir de certaines cases. D'abord à partir de toutes les cases, on peut ensuite optimiser.

En effet,

4.4 L'aléatoire

L'autre problème rencontré était que lors de l'activation de l'option séisme, le premier tour voyait une pile s'écrouler à chaque test. La probabilité donnée par la formule ne correspondait pas à une fréquence si haute.

Il s'est avéré que l'appel à `rand()`, même après qu'une graine ait été fournie par `srand(time(NULL))`. Après affichage des séquences de `rand()`, il s'est avéré que les premiers résultats étaient similaires. L'ajout de quelques appels non utilisés au début du programme semble avoir réglé le problème.

4.5 Le tableau events

L'affichage des éboulements aurait pu se faire lors de leur génération mais cette solution n'était pas satisfaisante parce que l'affichage devait se poursuivre ensuite durant le tour du joueur suivant.

Pour garder en mémoire quelles piles se sont éboulées, j'ai créé un tableau `events` dans la structure `s_jeu` qui marquait les piles touchées.

Par la suite, comme les fonctions de détection demandaient de connaître où un pion avait été joué, j'ai rajouté cette information dans le tableau grâce à un flag supplémentaire (cf précédemment). Un dernier flag a été rajouté pour déboguer `j_checkUp`, qui codait les piles marquées commes alignées (le bogue était finalement un oubli du test de vacuité des piles observées)

5 Manuel d'utilisation