Flood it!

Rapport Lot A : Préparation des données

April 24, 2017

Groupe : Les Prez (*Sylia Mehenni, Rémi Jamous, Simon Chollet, Elyne Amsellem*)

Sommaire

1 Introduction

On cherche à réaliser un jeu nommé *Flood it!* qui consiste en une grille de taille nxn de cases colorées aléatoirement en début de partie. Le but est de colorier la grille d'une seule couleur en un nombre de coups limité, fixé par l'application.
Les objectifs de ce premier lot sont de créer les structures de données et les fonctions d'initialisation et de manipulation de ces structures.

2 Organisation

2.1 Répartition des tâches

Référent Sprint 1 : Sylia Mehenni

Sylia et Simon ont réalisé la première partie des fonctions (structure de données, fonction d'affichage, fonction de victoire), Rémi s'est occupé de revoir le code et de vérifier les problèmes de mémoire liés à l'utilisation des *malloc*. Elyne s'est occupée du rapport et de la gestion du *Trello*.

2.2 Mise en commun

Le code est mis en commun sur *https://github.com* afin que chacun puisse apporter des modifications si nécessaire et ajouter des commentaires. On prend soin de commenter efficacement le code avec les paramètres que prend la fonction et ce qu'elle retourne.

On utilise ensuite le générateur de document *Doxygen* pour récupérer un fichier .html contenant l'ensemble du code et des commentaires.

2.3 Tests

On réalise des tests avec *CUnit*, d'après l'exemple donné. Toutes les fonctions autres que les fonctions d'affichage font l'objet d'un test avec des valeurs dont le résultat théorique est connu. Chaque fonction contient également des tests intrinsèques au code pour vérifier qu'elle fait effectivement ce qu'on lui demande, cf Annexes.

2.4 Outils

On ajoute à notre programme un *readme.txt* afin de permettre aux utilisateurs de se familiariser avec le jeu et son installation. Pour l'instant, il est très peu rempli car nous ne sommes qu'à la première étagpe du jeu et de fait, il n'y a pas encore de véritable ébauche de jeu à tester. Cf Annexes.

3 Structures de données et fonctions

On réalise les structures de données suivantes :

- *plateau*, matrice carrée de taille n entier, soit un pointeur de pointeur.
- *pile*, permet de stocker des coordonnées pour savoir si un élément appartient ou non à la matrice plateau de jeu.

On implémente les fonctions suivantes :

- *new*, fonction d'initialisation du plateau de jeu, génère une matrice carrée dont toutes les cases sont initialisées à 0.
- *new_pile*, permet de allouer de la mémoire pour créer une pile, dont on initialise la taille à -1 pour la considérer vide. On a pu créer une fonction qui teste si la pile est vide ou non.
- *aleatoire*, fonction qui crée le plateau et remplit les cases avec des nombres aléatoires, inférieurs à un entier *couleurs* qui définit le nombre de couleurs maximal autorisé pour la partie. De plus, les nombres aléatoirement distribués seront attribués par la suite à une couleur spécifique.
- *changement_couleur*, fonction de type *void*, qui change la couleur (valeur entière contenue dans la case) de la case de coordonnées i,j.
- *affiche*, fonction qui affiche le plateau de jeu.
- *supprime*, fonction qui libère la mémoire allouée au début du programme par les *malloc, calloc* par des *free*.
- *supprime_pi*, de même, désalloue la mémoire pour les piles.
- *ajout*, fonction qui permet d'incrémenter la pile d'un couple de coordonnées. On a une structure de "double" pile, qui contient les coordonnées en x et y sur chaque étage incrémenté.
- *sauvegarder*, fonction qui crée un fichier texte .txt qui conserve la partie de jeu, soit les colorations du plateau.Le fichier est créé s'il n'existe pas, sinon il écrase l'ancien; la première ligne du fichier correspond à la taille de la matrice carrée.
- *reprise*, fonction qui relance le fichier texte de la fonction précédente s'il existe et permet ainsi de retrouver le plateau de jeu en l'état. S'il n'existe pas un tel fichier, alors on génère un plateau rempli de valeurs aléatoires.
- *tache*, fonction qui permet de récupérer la composante connexe, aka la position de la tâche.
- *victoire*, fonction qui teste si le plateau est ou non d'une seule couleur, et renvoie *TRUE* si c'est le cas.
- *colorie_tache*, fonction qui recherche la tâche (cf composante connexe) et la colorie de la couleur donnée en troisième argument.

4 Difficultés rencontrées

L'une des principales difficultés de cette première partie de projet était de travailler avec de nouveaux outils, comme *CUnit* pour réaliser les tests et *Valgrind*.
De plus, la mise en commun de code en C permet d'avancer plus rapidemment et de trouver de nouvelles idées, mais cela peut être compliqué lorsqu'il s'agit de débugguer le code écrit par un autre membre de l'équipe.
Les fuites de mémoire créent facilement des erreurs si on oublie, par exemple, de désallouer de la mémoire.

5 Annexes

Ci-dessous on a écrit un fichier Main pour réaliser des tests de fuites de mémoire avec *Valgrind*.

```
int main(){
  plateau p = new(3);
  affiche(p);
  supprime(&p);

  plateau p2 = aleatoire( 3,6);
  affiche(p2);

  sauvegarder(&p2);
  supprime(&p2);

  plateau p_3 = reprise(1,1);
  affiche(p_3);
  /*supprime(p_3);*/

  plateau p_4 = reprise(1,1);
  if (victoire(p_4)) {
    printf("Gagné !\n");
  }
  else {
    printf("Pas gagné...\n");
  }
  plateau p_5 =aleatoire(1,6);
  /* tableau de 1*1 case, donc sûr de gagner */
  if (victoire(p_5)) {
    printf("Gagné !\n");
  }
  else {
    printf("Pas gagné...\n");
  }
  supprime(&p_4);
  supprime(&p_5);

  plateau t =new(4);
  tache(&p_3,&t);

  supprime(&p_3);
  supprime(&t);
  return 0;}
```

On réalise les tests avec *CUnit* dans un fichier en annexe au dossier, qu'il faut également compiler. Ci-joints les résultats.

```
MacBook-Air-de-SIMON:Flood simon$ ./testunitaire


     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: Suite
  Test: test of changement_couleur() ...passed
  Test: test of colorie_tache() ...passed
  Test: test of victoire() ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      3      3      3      0        0
             asserts      8      8      8      0      n/a

Elapsed time =    0.000 seconds
```