Flood it!

Rapport Lot B

May 2, 2017

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

Sommaire

Objectif: Réalisation d'un jeu avec affichage en mode texte dans le terminal.

1 Introduction

Remarque : ce rapport traite uniquement du travail réalisé pour le Lot B, en tenant compte des fonctions codées pour le Lot A. Un rapport d'ensemble sera réalisé lors de la dernière étape du projet.

Dans cette seconde étape, le but est de permettre à un utilisateur de lancer le programme de *Flood it!* dans un terminal et de pouvoir jouer entièrement une partie. La partie se termine lorsque la condition de victoire est remplie, avec affichage du nombre de coups. Par rapport à l'idée globale du jeu, on affiche une grille dont les cases contiennent un caractère en référence à une couleur (B pour bleu, G pour gris, R pour rouge, V pour vert, J pour jaune, M pour marron). L'utilisateur doit saisir au clavier la couleur dont il souhaite colorier la tâche.

Les objectifs de ce second lot sont d'utiliser les structures de données et les fonctions de manipulation de ces structures codées dans le lot A pour réaliser une première version du jeu.

2 Organisation

2.1 Répartition des tâches

Référent Sprint 1 : Simon Chollet

Sylia et Elyne ont codé la fonction d'affichage du jeu ainsi qu'une fonction de type *bool* qui permet de progresser dans le jeu. Elles se sont également occupées des commentaires (cf *Doxygen*) et du rapport. Simon et Rémi ont mis ensemble les scripts du lot A et du lot B pour pouvoir réaliser le *main* permettant de compiler et de lancer le jeu. Ils ont également réalisé des tests pour vérifier que le code compile et fait ce que l'on souhaite.

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. Il a fallu pour cela commenter le code selon les conventions reconnues par *Doxygen* et télécharger l'application qui permet de réaliser le fichier .html.

2.3 Tests

On réalise des tests en compilant le programme dans un terminal, avec le nouveau *Makefile* que l'on a codé. Le but est de réussir à jouer une partie et de la terminer, et de vérifier la présence d'éventuelles erreurs lorsqu'on tape aléatoirement des caractères au clavier.
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. Il contient à présent des instructions un peu plus détaillées sur comment lancer le jeu et terminer une partie (Cf Annexes).

3 Fonctions

On code les fonctions suivantes, en plus des fonctions du Lot A :

- *initialisation*, fonction qui prend en arguments 2 pointeurs vers un entier qui représentent respectivement la taille du plateau de jeu (matrice carrée) et le nombre de coups autorisés pour terminer la partie. On souhaite récupérer ces 2 nombres, or le langage C ne permet pas de les récupérer sous forme de couple de valeurs, par exemple. Donc le but est d'utiliser des pointeurs pour pouvoir accéder à ces valeurs en dehors de la fonction, une fois qu'ils sont définis dans le *main*. La fonction est donc de type *void*, elle se contente de demander à l'utilisateur d'entrer au clavier la taille souhaitée et le nombre de coups, de tester si ces nombres sont effectivement entiers et s'ils sont entre 1 et 100.

- *barre*, fonction qui permet d'afficher une grille dans le terminal lorsqu'on lance le jeu, en affichant simplement des traits - (x3) suivis d'un (+) pour indiquer l'intersection des cases. Cette fonction est utile ensuite pour la fonction qui gèrera l'affichage complet de la grille. Elle prend en argument la taille du plateau pour ne pas la dépasser et ne renvoie rien.

- *aff*, fonction qui prend en argument un plateau de jeu et réalise l'affichage complet du jeu. En effet, dans la fonction on récupère la taille du plateau, stockée en attribut et on affiche à l'écran la grille en appelant la fonction *barre*. On affiche le caractère représentatif de la couleur de la case entouré de 2 symboles "|" pour réalisé les parois de la grille. Cf Annexes.

- *etape*, fonction qui retourne un booléen. Elle demande à l'utilisateur de choisir un couleur pour colorier la tâche principale et vérifie que le carctère est autorisé. On fait en sorte que la fonction accepte les caractères minuscules et majuscules, dans la mesure des couleurs disponibles. On utilise un *switch* pour rendre le code plus lisible. On stocke un entier correspondant à une couleur et si c'est le cas en sortant du *switch*, on appelle *colorie_tache* et on renvoie $TRUE$, sinon on renvoie $FALSE$.

4 Difficultés rencontrées, remarques sur le code

Dans cette seconde partie, nous n'avons pas rencontrés trop de difficultés à coder ce qui était demandé.

Nous avons pu avancer un peu sur ce qui est demandé dans les lots suivants et essayer d'implémenter une interface graphique. Le résultat obtenu (Cf Tests en Annexes) est intéressant pour la suite de notre travail.Les outils utilisés sont ceux qui sont demandés dans les consignes.

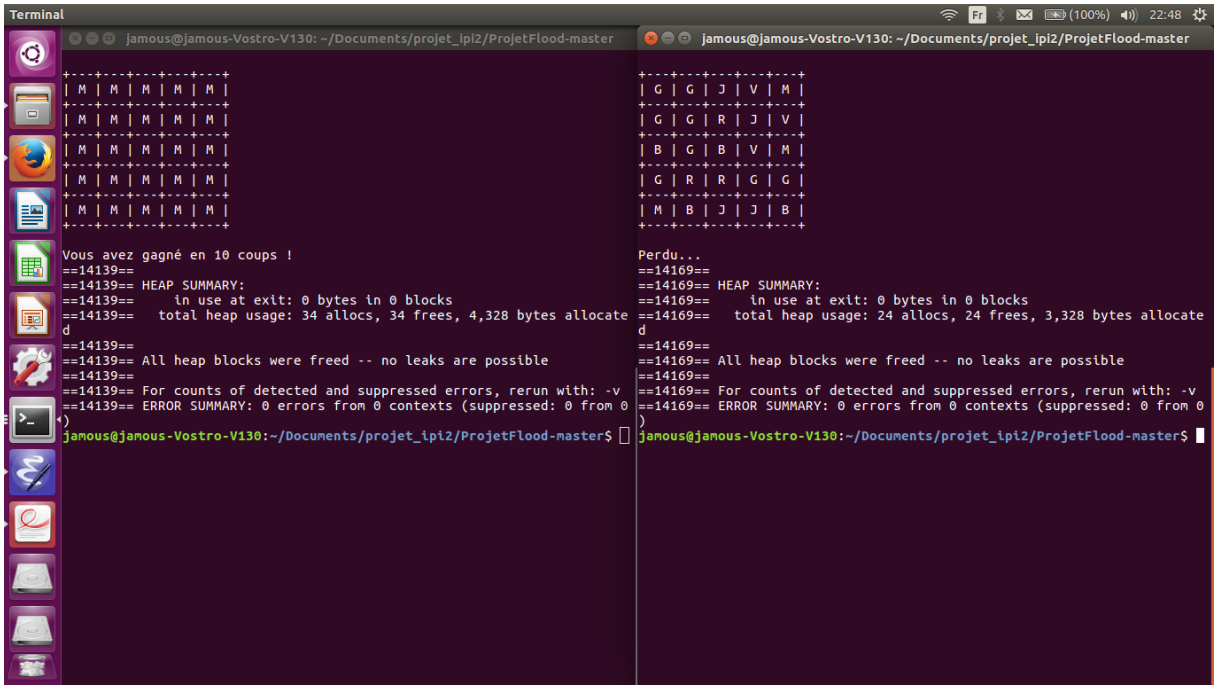
On a pu remarqué en faisant les tests que d'après notre code, il est possible de gagner le jeu en tapant une unique ligne de code, composé de coups ie une suite de caractères parmi ceux autorisés (Cf Annexes). De plus, le code s'adapte bien à cette particularité car si on tape des caractères non reconnus, ils ne sont pas comptabilisés et seuls les caractères connus sont pris en compte.

5 Annexes

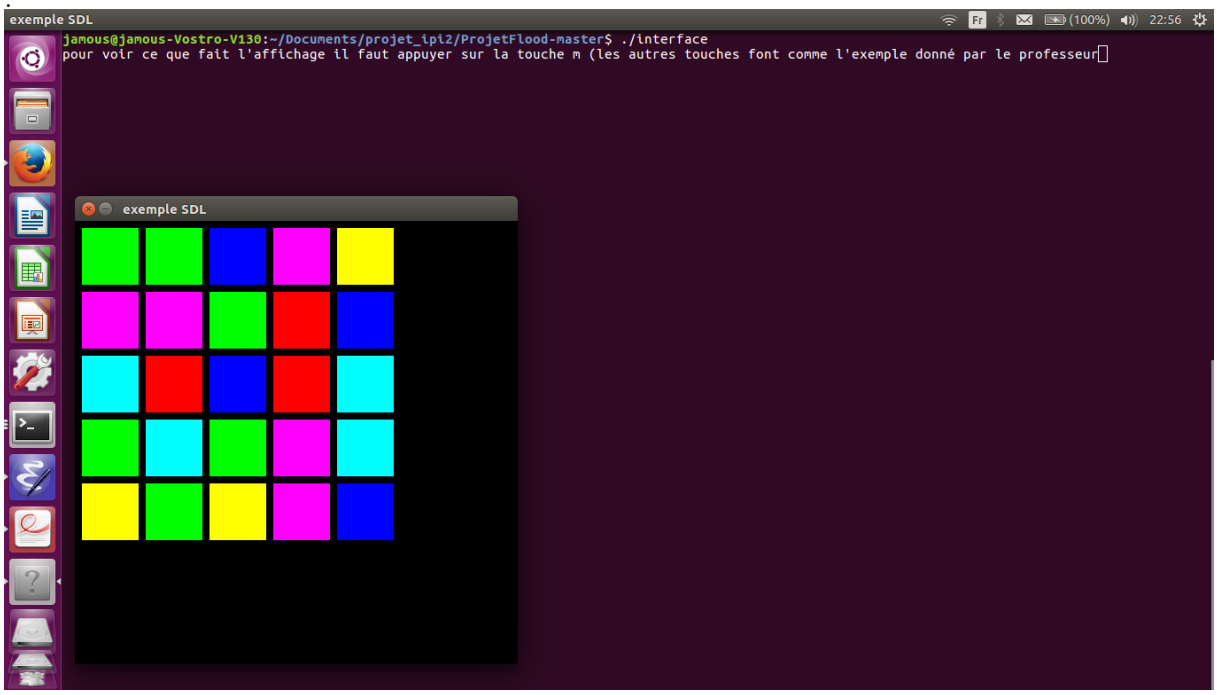
Ci-dessous, le nouveau fichier **Main** pour lancer le jeu en mode texte.

```
int main(){
int n;
int tour_max;
initialisation(&n, &tour_max);
plateau jeu = aleatoire(n, 6);
plateau tache = new(n);
int tour = 0;
int vict = victoire(jeu);
while (tour < tour_max && !vict) {
system("clear");
aff(jeu);
printf("\nTour : %d sur %d\n\n", tour, tour_max);
if (etape(&jeu, &tache)) {
tour++;
}
vict = victoire(jeu);
}
if (vict) {
system("clear");
aff(jeu);
printf("Vous avez gagné en %d coups !\n", tour);
}
else {
system("clear");
aff(jeu);
printf("Perdu...\n");
}
supprime(&jeu);
supprime(&tache);
return(0);
};
```

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

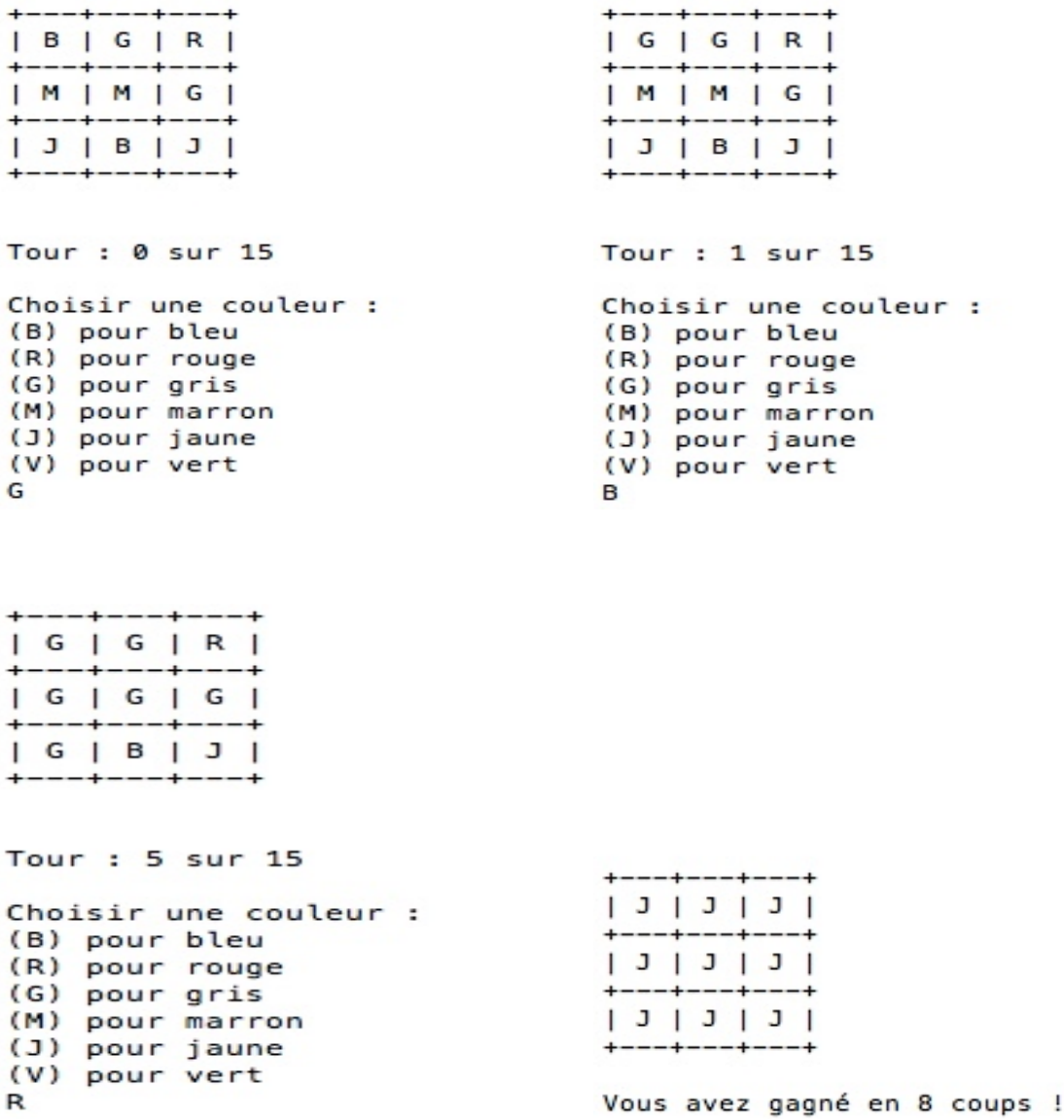


Voici également le résultat des tentatives d'implémentation de l'interface graphique, un bonus! :



On peut à présent faire des tests du jeu en lui-même, avec les résultats suivants:

- Jeu avec un plateau de taille 3 avec une marge de 15 coups:



On met en évidence les résultat d'un test qui consiste à entrer une suite de caractères avec certains reconnus et d'autres non, on constate que cela fonctionne, seuls les caractères reconnus sont pris en compte.