Université Jean-Monnet

Faculté des Sciences et Techniques

Rapport du projet de programmation impérative S3

Licence 2 Informatique

2015-2016

Etudiants

HEGAZI Mohamed - TOUAIBIA Mohamed - CITAK Bünyamin

Département d'Informatique

Table des matières

1 Introduction

Le sujet de notre projet consiste à implémenter **le Jeu de Yam's**, dont voici le lien *http://www.leyams.fr* qui nous a permis de connaître tout ce que l'on devait savoir sur le jeu. Pour cela nous devions utilisées nos connaissances acquise durant le semestre 3 en Language C. Donc nous avons commencé dans un premier temps à reflechir aux fonctions qui pourront nous servir pour la suite du projet .Puis nous avons mis en évidence certaines règles comme bloquer aucun, un ou plusieurs dés, le fait que le jeu ne se déploie pas identiquement pour un ou plusieurs joueurs, notamment avec l'ajout d'un "objectif" quand on est en mode solo. Une fois l'idée en tête, nous avons réfléchis chacun de notre côté, pour tout remettre en commun et partager les nouveaux problèmes que l'on a constaté durant l'écriture des fonctions.

Dans la suite de notre rapport nous allons expliquer plus en détail notre démarche, les fonctions principales et les choix d'implémentations.

2 Notre démarche

Une fois les fonctions de base terminées (**lance.c,menu.c tourde.c, main.c,decision.c**), nous avons eu la nécessité de traiter les cas d'erreurs, puis les problèmes qui sont venus s'ajouter aux contraintes de départ. Pour cela, nous avons créé plusieurs fonctions qui ont étés mis en liens entre-elles grâce au module ***gamefun.h***, ce dernier nous permet (pas de s) d'avoir nos prototypes de fonctions dans tous nos fichiers (.c). Et surtout il contient nos structures (**Joueur** et **meilleurscore**) qui nous ont aidées, la première fois dans la sauvegarde et le chargement mais aussi dans l'écriture dans les fichiers (**topscore.dat** et **topscoresingle.dat)** car nous avons voulus que lorsque l'on rentre dans le menu dans "Afficher meilleur score" qu'on puisse voir deux tableaux différents l'un quand on joue en groupe et l'autre quand on joue en mode solo ! Ce qui nous semblait plus équitable, même si la problématique de départ était un fichier avec les 10 meilleurs mais vu que ce n'était pas précisé, on a voulu rajouter ce deuxième tableau. Cependant, plus le jeu avançait, et plus nous avions des fichiers, car nous devions aussi avoir des fonctions courtes, c'est pour cela que pour chaque fonction nous avons créé un fichier, mais le problème était de bien compiler sans tout taper à la main, donc nous avons créé un fichier **"Makefile"**. Celui-ci prend en paramètre tous nos fichiers (.c + notre module) et les associe entre-elles. Ce qui permet de compiler juste en tapant "**make**" dans le terminal, et cela nous permet de compiler le jeu !

Notre démarche d'implémenter notre jeu de cette manière-là nous a permis d'avoir un fichier ***main.c*** compréhensible ! Car l'inverse aurait été incompréhensible malgré les nombreux commentaires qu'on aurait pu apporter !

3 Les fonctions principales

Nous allons expliquer les fonctions (***"Sauvegarder une partie"*** et ***"Charger une partie"***) qui sont les plus importantes car elles nous permettent d'avoir un jeu plus aboutie, c'est aussi les parties où nous avons traité de manière plus elaborée, car nous pouvons quitter une partie en cours de jeu, puis la reprendre à une date ultérieure et nous pouvons choisir de recommencer une nouvelle partie !

La fonction **sauvegarde** prend en paramètre un tableau de type joueur, le numéro du joueur et celui qui a joué en dernier. Par la suite, elle prend le nom donné à la partie que l'on veut sauvegarder, puis elle l'écrit dans le fichier "filenamedat.dat"(filenamedat, sera remplacer par le nom de notre sauvergarde), ceci sera utile dans la fonction suivante pour charger la partie.

La fonction (**load**) prend en paramètre un tableau d'entier (les numéros des joueurs, c'est-à-dire le nombre de joueurs existant avant la sauvegarde effectuée dans le fichier "filenamedat.dat") et un tableau de type joueur, puis regarde quel joueur à le nombre de tour le moins élévé , et lui ajoute +1, pour que sa soit le tour du joueur suivant à jouer (**tourde**, est la fonction qui réalise ceci). Mais dans un cas de tour identique entre plusieurs joueurs, c'est celui qui a le numéro joueur le plus petit qui va débuté. Puis, la fonction lit dans le fichier "filenamedat.dat" toutes les données qui nous seront utiles par la suite. Autrement dit, pour chaque joueur, on a une structure (**Joueur**), et en relançant le chargement de cette partie, pour chaque joueur, on a exactement les mêmes données. Ceci est possible grâce aux lectures de fichiers, et à l'inclusion de structure dedans ! C'est avec cela que nous avons évité le problème d'une structure pour chaque partie définissant le jeu ! Avec une seule structure liée à une personne et en y mettant toutes les informations que nous aurons besoin sur un joueur, nous pouvons être "plus efficace". Donc, on récupère la définition que l'on donne à un joueur dès le début de la partie. Puis durant toute la partie, il va le garder avec lui, c'est propre à lui ! Donc son tableau n'est pas lié à ceux des autres. Enfin, seulement à la fin de la partie nous comparerons la somme des tableaux et celui qui aura le meilleur score (i.e. somme de son tableau supérieur aux autres).

Finalement, si nous résumons ces deux fonctions selon nos connaissances de l'an dernier ça consiste à prendre en photo à un instant T le jeu ; puis en le rechargeant on remet en place cette image et on continue sans aucunes modifications, comme si de rien n'était !

4 Les choix d'implémentation

Maintenant, nous pouvons expliquer certains choix de structures, des fonctions de bases. Comment elles s'exécutent, et surtout expliquer, pourquoi nous avons choisi d'écrire notre programme de cette manière-là et pas une autre !

Tout d'abord, on nous précisé qu'une personne pouvait jouer sans même connaître les règles du jeu. Mais on nous a également dit qu'il n'y avait aucune limite d'âge du ou des joueur(s). Ce qui explique nos fonctions d'affichages, qui ont pour but d'informer le(s) joueur(s). C'est à dire que le jeu est "guidé" ; ainsi nous avons pu détecter plus simplement nos erreurs en début de projet, puisque dans les boucles il peut y avoir des "segmentation fault" étant donné qu'on arrive à une boucle infinie, et ce problème est souvent le plus dur à résoudre. C'est pour cela que nous avons mis beaucoup de conditions qui emmène le joueur à jouer le plus correctement possible afin d'éviter les bugs en plein milieu d'une partie. Ce qui aurait été, également, désagréable durant nos tests réalisés en y jouant le jeu. Puis nous avions dès le début le problème des dés bloqués ou non, pour cela nous avons voulus au début de l'écriture mettre un tableau temporaire. Cependant, nous avons eu quelques problèmes c'est pour cette raison là qu'on s'est tourné vers un tableau de "flag". Ce tableau nous l'avons aussi inclus dans la structure **Joueur**. Il fonctionne avec des 0 et 1 ce qui était plus simple pour nous. Pour décrire une situation prenons l'exemple du cas où nous bloquons le dé numéro 3 et seulement lui, alors le tableau de flag associé au joueur va être le suivant :

Dé 1	**Dé 2**	**Dé 3**	**Dé 4**	**Dé 5**
0	0	**1**	0	0

Ce qui veut dire que seul le dé numéro 3 est bloqué et au bout du troisième lancé on récupère notre tableau de dés, pour décider de la règle que nous voulons utilisée !

De plus, grâce aux nombreux tests réalisés, nous avons constaté que nous oublions les combinaisons déjà effectuées, c'est pour cela que nous avons insérer le tableau de chaque joueur du tour précédent, pour qu'il puisse jouer en fonction de cela. C'est-à-dire qu'on n'essaye pas d'avoir un full alors qu'on l'a choisi au deuxième tour.

Et il y a le choix d'implémenter un menu "bis", celui-ci nous servira pour la sauvegarde après un tour et non entre deux lancés. Ce qui nous a permis de nous faciliter le problème de la récupération pour rejouer le jeu ultérieurement. Ainsi, une fois sauvegarder avec un nom la partie qui était en cours, on peut recommencer une nouvelle partie.

Puis, nous avons mis des testeurs pour la pluspart des scanf, par example :

```
printf("Quelle est votre decision?\n");
      testInt=scanf("%d",&n);
```

Ceci test si le nombre tapé par le joueur est un entier. Car nous avons besoin de récupérer, un type particulier, donc on "oblige" à avoir un entier et pas une lettre par exemple ici.

5 Conclusion

En guise de conclusion, nous pouvons dire que notre démarche d'implémentation et nos parties répétées, nous ont permis de créer un jeu très guidé, car on est parti du principe qu'on devait déjà nous-même réussir à jouer correctement. Puisque le jeu était destiné à une population large n'ayant pas forcément de connaissances sur le jeu de Yam's.

Puis, nous avons assez vite observé qu'il y avait des idées d'implémentations qui dépassaient nos connaissances. Mais surtout qu'un problème détecté déclenchait d'autres problèmes à la chaine.

Ce qui prouve l'importance de toutes ses règles et de ses délimitations que l'on s'impose avant de commencer à écrire le jeu. Puis la pratique nous montre les problèmes qu'on ne s'était pas spécialement posé au départ. Donc nous avons appris à réfléchir sur un brouillon avant de se lancer directement dans l'écriture de fonction, sans vraiment savoir ce que l'on faisait.

Finalement, nous pouvons dire que selon nous c'est la meilleure façon pour être efficace dans un temps donné.