Rapport du projet de programmation système

Aza, Willem, Tremor Sullyvan, Delar Emmanoe

3 décembre 2017

Table des matières

1 Introduction

En petit groupe de 3 étudiants, nous avons réalisé un projet qui faisait appel au cours de programation système que nous avons suivi tout au long du semestre. Le but de ce projet était de développer quelques mécanismes de base destinés à servir dans le développement d'un petit jeu de plateforme 2D. Le projet s'est déroulé en plusieurs parties distinctes. La première partie portait sur la mise en place d'un mécanisme de sauvegarde et de chargement de la carte utilisée dans le jeu. La deuxième partie portait sur la gestion des temporisateurs permettant de planifier les différents événements du jeu.

2 Manipulation de fichiers.

Nous faisons quelques manipulations de fichiers afin de rendre le jeu un peu plus réactif, c'est-à-dire que tout joueur lorsqu'il commence un jeu devrait pouvoir mémoriser une partie en cours, ou faire quelques opérations sur une sauvegarde de jeu tel qu'un changement de pseudo sans avoir à tout recommencer. Le jeu étant dépourvu de ces options, nous implémenterons des fonctions nécessaires à la sauvegarde, au chargement de sauvegarde, et aux modifications de données enregistrées provenant d'une partie.

2.1 map-save

C'est une fonction qui consiste à mémoriser toutes les infos du jeu à un instant 'T', ce moment est déclenché quand le joueur tape 's' lors d'une partie. Pour réaliser cette fonction nous faisons des appels systèmes plutôt qu'utiliser les fonctions existantes en faisant attention que le retour d'appel s'effectue correctement grâce à notre fonction **verification()**. On débute par l'ouverture d'un fichier maps/saved.save grâce à l'appel **open()** qui nous renverra un descripteur de fichier sur lequel nous ferons l'inscription de données.

On procède à l'ajout de données dans un ordre bien précis. Cette ordre est important car nous aurons à lire ces données plus tard. Parmis les données enregistrées, on retouve des éléments essentiels tels que la taille et la largeur de la fenêtre, le nombre d'objets du jeu... A partir de ces données on mémorise plus facilement la position de chacun des éléments existants, leur nom et toutes autres informations concernant cet objet, afin de les restituer convenablement à l'aide de boucles.

2.2 map-load

C'est une fonction qui consiste à charger une partie qui aurait été sauvergardée au préable par notre joueur. Le chargement a pour but de récupérer toutes les informations dans un fichier par défaut : maps/saved.save ou tout autre fichier à condition que celui-ci respect l'ordre de sauvegarde d'informations. Pour réaliser cette fonction nous faisons des appels systèmes plutôt qu'uti-

liser les fonctions existantes en faisant attention que le retour d'appel s'effectue correctement grâce à notre fonction **verification()**.

On procède à la lecture de données dans l'ordre d'écriture fait sur fichier. Parmis les données, on retrouve des éléments essentiels tels que les dimensions de la fenêtre, du nombre d'objets possibles de cette partie ... A partir de ce relevé d'informations nous ferons le stockage dans des variables prévues à cet effet. Une fois ces variables instanciées nous procédons à la création d'une partie grâce aux fonctions prédéfinies telles que **map allocate()**, **map_set**... Il suffit maintenant de taper 'ı./game -l maps/saved.map' pour appliquer notre fonction.

2.3 Utilitaire de manipulation de carte

C'est un exécutable qui consiste à faire des changements sur une sauvegarde, il permet d'avoir les informations d'une partie, ou mieux encore changer des données enregistrées comme les dimensions de la fenêtre ou les objets contenus dans la partie, s'il y avait un pseudo on aurait pu le changer aussi grâce à cet exécutable.

Pour réaliser cet exécutable nous faisons des appels systèmes plutôt qu'utiliser les fonctions existantes en faisant attention que le retour d'appel s'effectue correctement grâce à notre fonction **verification()**. En fonction du nombre d'arguments en ligne de commande nous pouvons déterminer s'il sagit d'une modification ou d'un relevé d'informations.

Les relevés d'informations se font grâce à **get()** qui lit notre fichier et qui instanciera une variable, puis la renverra afin de restituer l'information voulue par l'utilisateur selon l'option (- -getwidth, ...) choisie. **get()** fait appel à **lseek()** pour se deplacer à la bonne position, nous utilisons un pointeur pour l'instanciation de la valeur voulue, l'instanciation est realisée grâce à **verif_ES()** qui dans le cas d'un relevé d'informations fera appel à **read()**. Une fois que l'opération voulue est faite, **get()** renvoie la position courante (utile selon l'opération que l'on souhaite faire).

Les modifications des données ont presque le même procédé sauf que **verif_ES()** fera appel à **write()**. Pour l'option - -pruneobjects qui efface des données inutilisées, on effectue une recherche d'objets inutilisés, on classifie les objets selon leur existance ou non dans la partie, ce classement est réalisé par **exchange()**, les objets inutilisés finissent en fin de liste par échange, on utilise **ftruncate()** pour nettoyer la fin du fichier et une mise à jour est faite.

3 Gestion des temporisateurs.

A continuer

4 Difficultés rencontrées.

A continuer