Rapport de projet : Groupe 9

Gaëlle BRAUD - Arthur LONGUEFOSSE - Gabriel LORRAIN
Université de Bordeaux
L3 Informatique
gaelle.braud@etu.u-bordeaux.fr
arthur.longuefosse@etu.u-bordeaux.fr
gabriel.lorrain@etu.u-bordeaux.fr

7 décembre 2017

Table des matières

1 Rendu intermédiaire n°1 (17 novembre)

1.1 Objectifs à atteindre

L'objectif du premier rendu était de se familiariser avec le jeu (commandes de base, édition de la carte..), puis d'include deux nouveaux objets (*flower* et *coin*) et leurs propriétés.
Ensuite nous devions implémenter la sauvegarde et le chargement des cartes, en développant les fonctions `map_save` et `map_load`.

Fichier à rendre :

```
mapio.c
├── map_save
└── map_load
```

1.2 Travail réalisé et résultats obtenus

Pour include les deux nouveaux objets, nous avons ajouter 2 au nombre d'objets total (précédemment 6) : `map_objet_begin(8)`, puis nous avons appeler un `map_object_add` pour chaque objet, en spécifiant en paramètre l'image `.png`, le nombre de sprites et les propriétés.
Les deux objets peuvent désormais être sélectionné et affiché dans le jeu.

Pour implémenter la sauvegarde d'une carte, nous avons compléter la fonction `map_save`. Pour ce faire, nous ouvrons un fichier vide `save` avec la fonction `open` puis nous stockons chaque caractéristiques de la carte dans des variables grâce aux fonctions fournies :

- Largeur : `int width = map_width();`
- Hauteur : `int height = map_height();`
- etc...

Ensuite nous stockons de la même manière les caractéristiques des objets (en les parcourant un par un), et le contenu de chaque case en parcourant la hauteur et la largeur de la carte.
A chaque définition de variable, un appel à la fonction `write` est effectué, pour placer dans le fichier `save` les caractéristiques de la carte et des objets. Cet appel est placé dans une condition pour pouvoir gérer les erreurs :

```
int width = map_width();
if(write(save,&width,sizeof(int))!=sizeof(int)){
  exit_with_error("erreur write width");
}
```

Pour implémenter le chargement d'une carte, nous avons compléter la fonction `map_load`. Pour ce faire, nous ouvrons le fichier contenant la sauvegarde avec `open`, puis nous stockons les caractéristiques contenues dans le fichier dans des variables grâce à la fonction `read`. Chaque appel est également placé dans une condition pour pouvoir gérer les erreurs :

```
int width;
if(read(fd,&width,sizeof(int)) == -1){
   exit_with_error("erreur lecture width");
}
```

Ensuite, nous utilisons les fonctions fournies pour créer la carte :
`map_allocate(width,height)` et `map_object_begin(nb_objets)`
De la même manière que le `map_save`, nous stockons les caractéristiques des objets dans des variables en les parcourant un par un, et à chaque fin de boucle on appelle `map_object_add(nom, frame, prop)`.
Une fois tous les objets ajoutés, on termine par `map_object_end()`.
Enfin, le contenu de chaque case est stocké en parcourant la hauteur (`i de 0 à height`) et la largeur (`i de 0 à width`) de la carte, et à chaque case on appelle `map_set(j,i,contenu)`.

La sauvegarde et le chargement sont désormais fonctionnels :

- Pour sauvegarder une map, on appuie simplement sur la touche **s** dans le jeu, ce qui va créer ou écraser le fichier save.map contenu dans le dossier /maps.
- Pour charger une carte, il suffit d'appeler le fichier de sauvegarde du dossier /maps : `./game -l maps/saved.map`

1.3 Difficultés rencontrées

- SAVE : Propriété get solidity en binaire

La principale difficulté rencontrée pour le `map_load` était de récupérer le nom des objets (pour afficher l'image) ; nous avions pensé à utiliser un `malloc` pour initialiser le tableau contenant les caractères du nom, mais le tableau de caractère ne se remplissait pas correctement. La solution était d'utiliser un `calloc` pour bien initialiser les éléments du tableau à 0.

2 Rendu intermédiaire n°2 (30 novembre)

2.1 Objectifs à atteindre

L'objectif du deuxième rendu était de développer des utilitaires de manipulation de carte via le programme `maputil`.
Celui-ci doit permettre :

- l'affichage des informations du fichier (`getinfo`) ;
- la modification de la taille de la carte (`setwidth, setheight`) ;
- le remplacement des objets d'une carte (`setobjects`) ;
- la suppression des objets inutilisés (`pruneobjects`).

Fichiers à rendre :

```
util
└── maputil.c
```

2.2 Travail réalisé et résultats obtenus

La fonction `getInfo` correspond simplement à l'appel de trois fonctions :

- `getWidth` (largeur) ;
- `getHeight` (hauteur) ;
- `getObjects` (nombre d'objets).

Ces trois fonctions sont simplement un `read` d'un `int` sur le fichier de sauvegarde. Le choix de la valeur(`int`) à lire se fait grâce à `lseek`, qui va permettre de déplacer la tête de lecture sur le fichier de sauvegarde :

int	**déplacement à effectuer**
`width`	0
`height`	`lseek(fd,sizeof(int),SEEK_SET)`
`nb_objects`	`lseek(fd,2*sizeof(int),SEEK_SET)`

Pour réaliser le reste des fonctions de `maputil`, nous avons fait le choix de ré-écrire entièrement une nouvelle carte (`new.map`) avec les attributs à changer (`width, height`, objets à remplacer ou suppression des objets inutilisés) et de remplacer l'ancienne carte par la nouvelle, définie de la manière suivante :

Pour `setWidth` et `setHeight`, on récupère chaque caractéristique de la carte avec un `read`, et on les place dans la nouvelle carte avec un `write` avec le nouvel attribut (`width` ou `height`). La boucle implémentant le contenu de chaque cases va également s'adapter avec le nouvel attribut, en supprimant le contenu des cases si il diminue, ou en ajoutant des cases vides si il augmente.

Pour `setObject`, ce sont les objets qui vont désormais être modifiés. On récupère donc les caractéristiques (inchangées) de la carte dans la nouvelle carte, puis à partir des arguments on récupère les propriétés des objets. Pour cela, nous avons implémenter une boucle `for` qui va s'incrémenter de 6 à chaque fin de boucle, pour permettre de récupérer les 6 caractéristiques de chaque objet dans la boucle. (`nom, frame, solidity, is_destructible, is_collectible ,is_generator`).

Enfin, pour `pruneObject`, on récupère les caractéristiques (inchangées) de la carte dans la nouvelle carte, puis on supprime les objets non présents sur la carte. Pour cela, on initialise un tableau `occ[nb_objet]`, puis on parcourt toutes les cases de la carte : dès qu'objet est trouvé, on incrémente la case du tableau correspondant à l'objet. Les objets non présents sur la carte (`i.e. occ[ob] = 0`) ne seront pas défini sur la nouvelle carte, grâce à un `lseek` sur leurs propriétés.

L'utilitaire de manipulation de carte `maputil` est désormais fonctionnel, depuis le dossier `util` :

— Obtenir des informations sur la carte sauvegardée :
```
./maputil ../maps/saved.map --getwidth
./maputil ../maps/saved.map --getheight
./maputil ../maps/saved.map --getobjects
./maputil ../maps/saved.map --getinfo
```

— Changer la taille de la carte :
```
./maputil ../maps/saved.map --setwidth <w>
./maputil ../maps/saved.map --setheight <h>
```

— Remplacer des objets de la carte :
```
xargs ./util/maputil maps/saved.map --setobjects < util/objects.txt
```

— Supprimer les objets qui n'apparaissent pas sur la carte :
```
./maputil ../maps/saved.map --pruneobjects
```

2.3 Difficultés rencontrées

La principale difficulté rencontrée pour réaliser `maputil` était de savoir quelle méthode utiliser pour modifier les propriétés de la carte et des objets. Nous avons opté pour une redéfinition entière d'une nouvelle carte, car nous pouvons modifier toutes les propriétés à la création. Il nous suffisait ensuite de remplacer l'ancienne carte par la nouvelle.

3 Rendu final (10 décembre)

3.1 Objectifs à atteindre

L'objectif du rendu final était d'implémenter un gestionnaire de temporisateurs, qui permettra au jeu de planifier des événements.
Pour cela, nous devions compléter les fonctions :

- `timer_init`, qui permet l'initialisation des variables et la mise en place des traitants de signaux ;
- `timer_set`, qui permet *d'armer* un temporisateur, grâce au paramètre `delay` qui spécifie la durée avant qu'un événement ne se déclenche ;

- **Bonus** : `timer_cancel`, qui permet d'annuler un temporisateur précédemment armé avec `timer_set`.

Fichiers à rendre :

- `tempo.c`
  - `map_save`
  - `map_load`
- `rapport PDF`

3.2 Travail réalisé et résultats obtenus

3.3 Difficultés rencontrées