université de BORDEAUX

DEPARTEMENT DE LICENCE

LICENCE 3 PARCOURS INFORMATIQUE
RAPPORT

PROJET DE PROGRAMMATION SYSTÈME

Brian DIETRICH
Abdoul DIALLO

1. LEXIQUE

Map: carte de jeu, ensemble d'éléments constituant l'univers du jeu.
util: utilitaires pour les manipulations de la carte de jeu.
timer: Compte-à-rebours à l'échéance duquel il s'agit de réaliser une action prédéfinie.
temporisateurs: voir timer

2. ORGANISATION ET PARTAGE DU TRAVAIL

le projet, découpé de façon générale, en trois parties distinctes, avec des charges de travail plus ou moins importantes, nous a conduit à partager le travail comme suit:

-Pour chaque étape on divise le travail en 2 parties
-Le premier à finir essaye au maximum de seconder l'autre

Afin de faciliter la mise en commun du travail réalisé, le logiciel de gestion de version Git a été utilisé.Le dépôt distant et commun à tous les participants est hébergé sur la plateforme github et est accessible via ce lien : https://github.com/dicowenza/MARIO17.git

3. PROCÉDURE DE QUALITÉ

Pour fournir un travail de qualité et éviter le maximum d'erreurs un certains nombre de mesures ont été prises :
-Etablissement de standards concernant la nomenclature des variables et des fonctions,l'indentation, le principe de sauvegarde d'une carte,….
-relecture de code
-ajout de commentaires exhaustifs

4. SPÉCIFICATION DU STANDARD DE SAUVEGARDE D'UNE CARTE

L'établissement de ce standard vise à minimiser les risques d'erreurs lors de l'utilisation des fichiers de sauvegarde sur différentes plateformes tout en permettant la mise en place d'une ligne directrice pour le(s) développeur(s) chargé(s) du code en lien avec la sauvegarde.

La taille des types courants (int, unsigned...) n'est pas définit par le standard C : cela peut poser des problèmes de portabilité lorsqu'on les utilise pour écrire dans un fichier.Pour la lecture/écriture du fichier, on aurait pu donc utiliser les types de taille fixe définit par le langage C int_8 et int_32 respectivement pour char (les caractères et tableaux de caractères) et unsigned int pour les entiers.

La valeur d'une case correspond à -1 pour MAP_OBJECT_NONE, ou à l'indice de l'objet (débutant à 0) dans l'ordre dans lequel ils apparaissent dans le fichier (et donc dans l'ordre dans lequel il faut les ajouter avec map_object_add).

5. FORMAT DE SAUVEGARDE DANS L'ORDRE

Les dimensions de la carte:
unsigned int width: largeur ou hauteur de la carte
unsigned int height : longueur de la carte
Le nombre d'objets:
unsigned int nbObjects: le nombre d'objets de la carte
Les cases de la carte (ordre : colonne par colonne)
unsigned int colonne 0, ligne 0, cellule 0 ...
unsigned int colonne [largeur - 1], ligne [hauteur - 1], cellule [hauteur - 1]
Les bjets(ici on considère un seul objet, mais pour la sauvegarde on concatène plusieurs objets)
Pour se faciliter la compréhension on peut représenter un objet comme la structure C suivante :

```
typedef struct{
        unsigned int filenameSize;
        char *filename;
        unsigned int frame;
        unsigned int solidity;
        unsigned int collectible;
        unsigned int destructible;
        unsigned int generator;
}object;
```

avec :
filenameSize : taille de la chaine de caractères (nom de l'objet)
filename : nom de l'objet
frames : nombres de frames
solidity : 0 si MAP_OBJECT_AIR, 1 si MAP_OBJECT_SEMI_SOLID, 2 si MAP_OBJECT_SOLID
destructible : 1 si destructible, 0 sinon
collectible : 1 si collectible, 0 sinon
generator:1 si generator,0 sinon

on enregistre donc ces dernières informations pour chaque objet de la carte pour pouvoir les restituer lors d'un éventuel chargement.

6. DÉTAILS RELATIFS AU DÉVELOPPEMENT DE LA SAUVEGARDE ET DU CHARGEMENT DES CARTES

Les fonctions de sauvegarde et de chargement servant d'appui au reste du projet, il nous a fallu définir un format de fichier afin de pouvoir développer l'utilitaire de carte en parallèle. Il a donc été choisi de sauvegarder le fichier selon le format cité plus haut. Vu le nombre de fois que l'utilisation d'un message d'erreur est nécessaire on a implémenté la fonction suivante pour palier le problème: cette fonction affiche donc le message qu'elle reçoit en paramètre et quitte le programme.

```c
//affichage un message d'erreur
void bigError(char * message)
{
    fputs("ERROR: ", stderr);
    fputs(message, stderr);
    fputs("\n", stderr);
    exit(EXIT_FAILURE);
}
```

a. LA SAUVEGARDE

Le principe de la fonction de sauvegarde consiste à « lire » une carte et à l'enregistrer dans un fichier. L'utilisation de writer (fonction implémentée par nos soins) permet un affichage d'erreur en cas de problème d'écriture dans un fichier. l'enregistrement du fichier est une succession d'acquisition par les fonctions de map et d'écriture dans le fichier. Pour l'ensemble des objets présents sur la carte on enregistre le nom, le nombre de frames, la solidité et l'ensemble des caractéristiques de l'objet.

```
void writer(int load, void * buf, int count)
{
   int r = write(load, buf, count);
   if (r != count)
      bigError("writing to map file failed");
}

void map_save (char *filename)
{
// On commence par créer un fichier de sauvegarde
  int save=open(filename,O_WRONLY|O_CREAT|O_TRUNC,0664);
  if(save==-1)
    bigError("probleme d'ouverture de filename");
//on recupere les dimensions de la carte
  unsigned int width=map_width();
  unsigned int height=map_height();
  unsigned int nb_objet=map_objects();
//on commence la sauvegarde des dimensions de la carte
  writer(save,&width,sizeof(unsigned int));
  writer(save,&height,sizeof(unsigned int));
  writer(save,&nb_objet,sizeof(unsigned int));
//on recupere les différent types d'objet selon les coordonnées de la carte
  for(int x=0;x<width;++x){
    for(int y=0;y<height;++y){
      unsigned int pixeltype=map_get(x,y);
      writer(save,&pixeltype,sizeof(unsigned int));
    }
  }
//pour les objets
  for(int i=0;i<nb_objet;++i){

    char * filename = map_get_name(i);
    unsigned int filenameSize = strlen(filename);
    writer(save, &filenameSize, sizeof(unsigned int));

    for (int j = 0; j < filenameSize; j++)
    {
       char c = filename[j];
       writer(save, &c, sizeof(char));
    }
    unsigned int frames=map_get_frames(i);
    unsigned int solidity=map_get_solidity(i);
    unsigned int destructible =map_is_destructible(i);
    unsigned int collectible=map_is_collectible(i);
    unsigned int generator=map_is_generator(i);

    writer(save,&frames,sizeof(unsigned int));
    writer(save,&solidity,sizeof(unsigned int));
    writer(save,&destructible,sizeof(unsigned int));
    writer(save,&collectible,sizeof(unsigned int));
    writer(save,&generator,sizeof(unsigned int));
  }
  close(save);
  printf("sauvegarde reussi !\n");
}
```

b. LE CHARGEMENT

La fonction de chargement permet de charger une carte depuis un fichier pouvant être modifié par l'utilitaire, il était donc nécessaire d'y charger les objets. L'allocation de la carte faite, il n'est pas utile de charger une case vide, c'est le test effectué avec cette instruction (if pixel !=MAP_object_none). Les cases non-vides sont alors chargées.L'utilisation de la fonction reader avec le même principe de fonctionnement que writer affiche donc un message d'erreur et arrete le programme en cas de problème de lecture dans le fichier sauvegardé précédement.Le chargement des objets s'effectue par la suite,de la même manière que l'enregistrement. On effectue une lecture sur le fichier de l'ensemble des caractéristiques de l'objet qu'on ajoute par la suite avec la fonction map_object_add. L'ensemble des tests permettent d'arrêter la lecture du fichier dans un cas d'erreur.

```
void reader(int load, void * buf, int count)
{
   int r = read(load, buf, count);
   if (r != count)
      invalidMap();
}
void map_load (char *filename)
{//On commence par ouvrir le fichier ou la carte a été sauvegardée
  int load = open(filename, O_RDONLY);
     if (load == -1)
        bigError("unable to open map file for reading");
  /*On lit dans le fichier les dimensions de la carte puis on vérifie que les dimensions soient inférieures
        aux dimensiosn maximales définies dans le .h*/
     unsigned int width     = 0;
     unsigned int height    = 0;
     unsigned int nbObjects = 0;
     reader(load, &width    , sizeof(unsigned int));
     reader(load, &height   , sizeof(unsigned int));
     reader(load, &nbObjects, sizeof(unsigned int));

     if ( width < MAP_MIN_WIDTH || width > MAP_MAX_WIDTH || height < MAP_MIN_HEIGHT
        || height > MAP_MAX_HEIGHT|| nbObjects < 0 || nbObjects > MAP_MAX_OBJECTS )
      invalidMap();

    /* on crée la carte avec les dimensions récupérées*/
     map_allocate(width, height);

  /* on récupére ensuite les différents type d'objet de la carte et on vérifie que si le type est différent
   de MAP_OBJECT_NONEet que le type d'objet n'est pas compris entre 0 et ke nombre d'objets de la carte on
   renvoi une erreur sinon on fait un map_set(x,y,pixel)*/
     for (int x = 0; x < width; x++){
        for (int y = 0; y < height; y++)
        {
           unsigned int pixel = MAP_OBJECT_NONE;
           reader(load, &pixel, sizeof(unsigned int));

           if (pixel != MAP_OBJECT_NONE && (pixel < 0 || pixel > nbObjects - 1))
              invalidMap();

           map_set(x, y, pixel);
        }
      }

  /* Il ne reste plus qu'à récupérer les objets de la carte ( briques , fleurs , pièces ... ) avec leurs
  différentes caractéristiques (chemin , frames , solidité , destructible , générateur ) */
     map_object_begin(nbObjects);

     for (int i = 0; i < nbObjects; i++)
     {
/*On récupére le chemin de l'image de l'objet , si on arrive à lire la taille de la chaine de caractère
 on réalise un malloc sinon on renvoi une erreur */
           unsigned int filenameSize = 0;
           reader(load, &filenameSize, sizeof(unsigned int));
```

```
            if (filenameSize < 1 || filenameSize > MAP_OBJECT_MAX_FILENAME_SIZE)
               invalidMap();

            char * filename = malloc(filenameSize + 1);

            for (int j = 0; j < filenameSize; j++)
            {
               char c = 0;
               reader(load, &c, sizeof(char));
               filename[j] = c;
            }

            filename[filenameSize] = '\0';

            /* puis on récupére chaque caractèristiques solidité , destructible ... */
            unsigned int frames       = 0;
            unsigned int  solidity     = 0;
            unsigned int  destructible = 0;
            unsigned int  collectible  = 0;
            unsigned int  generator    = 0;
            reader(load, &frames      , sizeof(unsigned int));
            reader(load, &solidity    , sizeof(unsigned int));
            reader(load, &destructible, sizeof(unsigned int));
            reader(load, &collectible , sizeof(unsigned int));
            reader(load, &generator   , sizeof(unsigned int));

            if( frames < 1 || frames > MAP_OBJECT_MAX_FRAMES
                || (solidity != MAP_OBJECT_AIR && solidity != MAP_OBJECT_SEMI_SOLID && solidity != MAP_OBJECT_SOLID)
                || (destructible != 0 && destructible != 1)
                || (collectible  != 0 && collectible  != 1)
                || (generator    != 0 && generator    != 1)
              )
              invalidMap();
           /* on crée ensuite une variable flags afin de stocker les caractéristiques de l'objet */
            unsigned int flags = 0;
            flags = solidity;
            if (destructible)
                flags = MAP_OBJECT_DESTRUCTIBLE;
            if (collectible)
                flags = MAP_OBJECT_COLLECTIBLE;
            if (generator)
                flags = MAP_OBJECT_GENERATOR;
  /*puis on fait un map_object_add pour ajouter l'objet avec ses caractériqtiques */
            map_object_add(filename, frames, flags);
  /*on libére ensuite la mémoire du tableau filename */
            free(filename);
   }

   map_object_end();
   /* on ferme le fichier de sauvegarde */
   close(load);
}
```

7. DÉTAILS RELATIFS AU DÉVELOPPEMENT DE L'UTILITAIRE DE MANIPULATION DE LA CARTE

a. RÉCUPÉRATION DES INFORMATIONS

Connaissant parfaitement le format de saved.map on a donc facilement accès aux informations qui y sont enregistrées en utilisant la fonction C de manipulation de position dans les fichiers **lseek(int** *fd*, **off_t** *offset*, **int** *whence*) ,on récupère la longueur, la largeur et le nombre d'objet de la carte.

b. MODIFICATION DES DIMENSIONS DE LA CARTE

La modification des dimensions de la carte (longueur et largeur) entraine la gestion de deux phénomènes :

- si la carte s'agrandit il n'y a aucun problème on recopie le contenue précédent de la carte et on ajoute des MAP_OBJECT_NONE pour la partie à redimensionner donc il n'y a là aucune perte d'informations (les mêmes objets aux mêmes positions …)

- si la carte rétrécit par contre, il faut accepter la perte d'informations situées dans la zone à redimensionner donc par conséquent on assiste à une perte d'informations sur la carte sauvegardée précédemment. On veille par la même occasion à ce que les valeurs de redimensionnement de la carte respectent les intervalles définis dans map.h.

c. SUPPRESSION DES OBJETS NON UTILISÉS

la suppression des objets non utilisés entraîne une modification du nombre d'objets de la carte.Le principe utilisé est très simple, on parcourt la carte sauvegardée on marque les objets qui sont utilisés on les sauvegarde(on les recopie sur la carte) et on supprime tous ceux qui ne sont pas marqués(on ne les recopie pas).Puis on remet à jour le nombre d'objets.

d. AJOUT DE NOUVEAUX OBJETS

Pour ajouter un objet sur une carte contenue dans un fichier on récupère dans un premier temps le nombre d'objets de la carte puis on se positionne à la fin du fichier pour ajouter l'objet reçu en paramètre de la ligne de commande ./maputil <file> --setobjects …….
Une fois l'ajout terminé on incrémente donc le nombre d'objets en conséquence.

Dans cette deuxième partie voici les quelques fonctions utilisées pour faciliter les opérations de copie d'un fichier à l'autre:

```
void copyWrite(int src,int dst,int count,void * buf){

  read(src,&buf,count);

  write(dst,&buf,count);
}


void copyEndFile(int src,int dst,unsigned int frames ,unsigned int solidity,unsigned int
                 destructible,unsigned int collectible , unsigned int generator ){

  copyWrite(src,dst,sizeof(unsigned int),&frames);

  copyWrite(src,dst,sizeof(unsigned int),&solidity);

  copyWrite(src,dst,sizeof(unsigned int),&destructible);

  copyWrite(src,dst,sizeof(unsigned int),&collectible);

  copyWrite(src,dst,sizeof(unsigned int),&generator);
}


unsigned int getWidth(int file){
  unsigned int largeur;
  lseek(file,0,SEEK_SET);
  read(file,&largeur,sizeof(unsigned int ));
  return largeur;
}


unsigned int getHeight(int file){
  unsigned int hauteur;
  lseek(file,sizeof(unsigned int),SEEK_SET);//on se positionne à la deuxieme valeur du fichier
  read(file,&hauteur,sizeof(unsigned int));
  return hauteur;
}


unsigned int getNbObject(int file){

  unsigned int nbObjet;
  lseek(file,2*sizeof(unsigned int),SEEK_SET);//on se positionne à la troisieme valeur du fichier
  read(file,&nbObjet,sizeof(unsigned int));
  return nbObjet;
}
```

8. DÉTAILS RELATIFS AU DÉVELOPPEMENT DES TEMPORISATEURS

Il s'agit dans cette partie d'implémenter un système de gestion de compte-à-rebours (appelés ici temporisateurs ou timer). La principale difficultée résidant dans le fait que ce système entraîne et nécessite l'apparition d'un principe d'asynchronisme. En effet, le jeu règle un timer et ne se préoccupe plus de son déroulement jusqu'à son échéance, le code est donc ici non-bloquant, c'est-à-dire que le jeu n'attend pas la résolution d'un timer pour continuer son exécution.

9. LES ACQUIS DURANT LE PROJET

La réalisation de ce projet nous a permit d'acquérir et de consolider plusieurs connaissances. Ce sont entre autres :

-la collaboration virtuelle dans la production de codes (utilisation du dépôt Git) : l'utilisation d'un répertoire en commun nous a confronter aux problèmes de collaboration et nous a permis d'avoir de nouvelles méthodes de communications.

-la mise en pratique de plusieurs notions abordées en cours : création et manipulation de fichiers avec utilisation des bibliothèques et fonctions disponibles, création et utilisation de processus(suite à des bugs après implémentation nous avons dû renoncer à l'utilisation des processus au cours de la deuxième partie du projet).

-la dernière partie du projet sur les temporisateurs a permis globalement de mettre en oeuvre et d'utiliser des threads.

-le débogage de code pas à pas et l'élimination de fuites mémoire en cas d'allocation de celle-ci. A cela on peut ajouter l'importance des tests et leurs mises en oeuvre.