Rapport final

UE Programmation Système

LIAO Zuzhi -
OUSSENY Irfaane -
MAHAZOASY Heritiana Allan

Contenu

I- Sauvegarde et chargement des cartes

a) Map_save :

Après les premiers exercices permettant la compréhension du code de mapio.c et la gestion des objets de la map, la première tâche fut de créer les fonctions map_save et map_load, mettant en avant les fonctions write, read et open étudiés en cours.

La première chose à faire a été le protocole de création d'un fichier map : l'ordre des valeurs et le contenu étant à nos soins, nous avions une grande marge de liberté vis-à-vis de la forme finale. Si au tout début cette partie a été légèrement hasardeuse, nous en sommes arrivés à ce protocole-ci :

Map_width|Map_height|nombre d'Objets| [taille_nom_objet|nom|nombre de frames|solidity|is_destructible|iscollectible|is_generator] *nombre d'objets|matrice width*height

Vous trouvez ci-dessus la structure d'un fichier map sans espaces ni les « | » et accolades qui nous permettent ici de vous donner un aperçu lisible du fichier de sortie.

```
57    write(file,&width,sizeof(int));
58    write(file,&height,sizeof(int));
59    write(file,&nbObjet,sizeof(int));
60
61    int frame, solidity, isDestructible, isGenerator, isCollectible;
62    char* fichierPNG;
63    int taille;
64
65   //on envoie tous les objets différents de la map
66    for (int i = 0; i < nbObjet; i++) {
67      fichierPNG = map_get_name(i);
68      taille = strlen(fichierPNG);
69      //on envoie dans le fichier la longueur du nom de l'objet
70      write(file,&taille,sizeof(int));
71      //tant qu'on arrive pas à la fin du nom de l'objet, on envoie caractère par caractère dans le file
72      for(int indice=0;indice < taille;indice++) {
73        write(file,&fichierPNG[indice], sizeof(char));
74      }
75      /* On ecrit dans le fichier le nom,le nombre de sprites du fichier ainsi que les
76        les propriétés qu'il peut avoir
77      */
78      frame = map_get_frames(i);
79      write(file,&frame,sizeof(int));
80
81
82      solidity = map_get_solidity(i);
83      write(file,&solidity,sizeof(int));
84
85
86      isDestructible = map_is_destructible(i);
87      write(file,&isDestructible,sizeof(int));
88
89
90      isCollectible = map_is_collectible(i);
91      write(file,&isCollectible,sizeof(int));
92
93
94      isGenerator = map_is_generator(i);
95      write(file,&isGenerator,sizeof(int));
96
97    }
```

Ce n'est que plus tard que nous nous étions rendu compte du fonctionnement des objets et de la fonction map_get. En effet, nous pensions devoir rajouter pour chaque élément de la matrice ses paramètres ainsi que son nom pour chaque case, ce qui aurait impliqué un nombre assez grand

de read et de write à faire lors de la sauvegarde et du chargement. Au final, le fait de savoir qu'il suffit de renvoyer l'index de l'objet dans le tableau nous a fait beaucoup de bien.

```
98
99      int contenuCase;
100     for (int i = 0; i < width; i++) {
101       for (int j = 0; j < height; j++) {
102         contenuCase = map_get(i,j); //numéro de l'objet dans la liste des objets de la map
103         write(file,&contenuCase,sizeof(int));
104       }
105     }
106
```

Nous avons notamment beaucoup hésité sur la forme du write et l'écriture d'un entier dans un fichier. En effet, nous ne voulions pas qu'il renvoie un entier en hexadécimal alors nous avions commencé par créer une variable string temporaire dans lequel nous écrivions l'entier via une fonction fprintf, string que nous écrivions dans le fichier afin d'être sûr que le fichier récupérerait les bonnes informations. C'est après plusieurs essaies que nous comprirent correctement l'utilisation du sizeof(int) en troisième paramètre et que cela nous aiderait d'ailleurs énormément dans la suite du devoir avec le map_load.

b) Map load :

C'est en s'inspirant du Map_new que nous avions défini le plan de base de Map_save et de Map_load, cette dernière n'étant que la suite logique de la fonction de sauvegarde, sa réalisation n'a pas été la plus difficile. C'est au moment où il fallait récupérer le nom des objets de la map que nous avions eu notre première réelle difficulté du devoir. En effet, à la base nous utilisions un système de flag entre les différentes informations pour être sûr que nous récupérions un mot entier. Chaque fin de nom était ponctuée d'un point d'exclamation. Si au début cela nous semblait astucieux le résultat final n'était pas celui que nous attendions et nous avions pris beaucoup de temps à le débugger sans succès. C'est à ce moment-là que l'on a décidé de restructurer map_save afin que l'on communique directement dans le fichier la taille des mots, évitant ainsi les surprises. Les deux solutions étaient bonnes, mais l'une a été plus facile pour nous à implémenter que l'autre.

```
132     for (int i = 0; i < nbObject; i++) {
133       read(file,&buffer,sizeof(int)); //compte le nombre de caractère dans le nom du des objets
134       if (buffer> taille) {
135         fichierPNG = realloc(fichierPNG, buffer*sizeof(char));
136         taille = buffer;
137       }
138       for (int j = 0; j < buffer; j++) {
139         read(file,&fichierPNG[j], sizeof(char));
140       }
```

L'autre difficulté de la partie fût la bonne utilisation de la fonction open(). En effet, après avoir implémenté une fonction qui nous semblait plus que correct, ce que nous lisions dans le fichier n'était pas le résultat attendu. Pourtant tout semblait parfait du côté du map_save. Le problème devait forcément venir de notre map_load. A la lecture, nous recevions une erreur d'allocation à cause d'une trop grosse valeur attribuée à width et height qui étaient semblable à une adresse mémoire. Après avoir passé plusieurs demi-journées dessus, on en avait déduit que le problème

devait venir de l'open qui était défini en O_WRONLY|O_RDONLY. C'est en enlevant la permission d'écriture et en laissant seulement la lecture que nous l'on a résolu notre problème.

```
111  void map_load (char *filename)
112  {
113    int file = open(filename,O_RDONLY);
```

Il a ensuite suffit de lire la matrice et utiliser la fonction map_set prédéfini pour chaque point de la map pour nous retrouver avec un load fonctionnel.

```
168    for (int i = 0; i < width; i++) {
169      for (int j = 0; j < height; j++) {
170        read(file,&buffer,sizeof(int));
171        if (buffer != MAP_OBJECT_NONE) {
172          map_set(i,j,buffer);
173        }
174      }
175    }
176    close(file);
177  }
178
```

II- Utilitaire de manipulation de la carte

a) Get_width, get_height, get_objects et get_info:

Il est clair que nous ne passerons pas beaucoup de temps dans cette section, voilà pourquoi nous regroupons les get ensemble.

En simple, nous avons d'abord récupéré toutes les informations de base, qu'importe la demande de l'utilisateur puis nous les redistribuons. La seule difficulté a été encore une fois l'appel à open que nous avions défini sur WR_ONLY|RD_ONLY qui donnait des valeurs hasardeuses aux reads, que nous avons ensuite transformée en O_RDWR et qui cette fois-ci apportait les bonnes valeurs.

b) Set_width et set_height :

On sait que le début d'un fichier de sauvegarde est composé de plusieurs entiers. Afin de remplacer une donnée, il suffit de faire un certain nombre de saut en fonction de la donnée recherchée à l'aide de lseek.

```
350    else if (strcmp(argv[2],"--setheight") == 0 ) {
351      // on lit la valeur de map_width
352      //read(file, &width,sizeof(int));
353      printf("\nentrée dans la condition setheight\n");
354
355      setmap(file,width,height,width,atoi(argv[3]),nbObject);
356      height = atoi(argv[3]);
357      lseek(file,sizeof(int), SEEK_SET);
358      write(file,&height, sizeof(int));
359      printf("\nheight  : %d\n", height);
360    }
```

Ici, SetHeight implique de faire d'abord un saut d'entier avant d'être à la position du height et ensuite le remplacer par la valeur désirée. Il va de même pour la fonction combinant set_width et set_height.

Setmap est la fonction que nous avons créé afin de répondre aux besoins des différents set. En effet, outre le fait de remplacer la valeur de height ou de width, il fallait aussi réarranger la taille de la matrice du terrain. Elle prend en paramètre les nouvelles tailles et les anciennes ainsi que le fichier.

Tout d'abord, on se met au niveau du début de la matrice avec un lseek démarrant de la fin afin de revenir [largeur*hauteur] en arrière.

```
48    off_t position1=lseek(file,-((oldWidth*oldHeight)*(sizeof(int))),SEEK_END);
```

On récupère les anciennes tailles que l'on va comparer avec les nouvelles afin de savoir si l'on réduit ou agrandit la map. Grâce à ça, on a une idée de ce qui doit être fait au niveau de la réorganisation des éléments.

On a créé une structure Element qui prend des coordonnées et une valeur représentant le contenu (l'objet) qui lui corresond et on en fait un tableau de la taille de la matrice.

```
24  typedef struct element{
25    int x;
26    int y;
27    int value;
28  }element;
```

On parcourt ensuite toute la matrice et on récupère chacun des élements que l'on ajoute dans le tableau.

```
81    lseek(file,-((oldWidth*oldHeight)*(sizeof(int))),SEEK_END);
82    int none=-1;
83      for(int i=0;i<newWidth;i++){
84        for(int j=0;j<newHeight;j++){
85          for(int indice=0;indice<(nbelmnt);indice++)
86           if((tabElmnt[indice].x==i) && (tabElmnt[indice].y)==(j)){
87              }
88          }
89        }
90      }
```

Cependant, cette fonction ne marche pas correctement pour deux raisons. La première est que les éléments perdus restent dans le fichier et l'on n'a pas réussi à les enlever, même avec un truncate. Du coup, certains éléments perdus reviennent et créent conflits avec la nouvelle map et rien ne semblait pouvoir les effacer. Le second problème vient du réajustement en hauteur qui crée une déformation du terrain malgré les réajustements réalisés.

Il a été pensé qu'en passant par un fichier temporaire puis une concaténation en remplaçant toute la matrice au lieu de la réécrire par-dessus, la fonction aurait été fonctionnelle. C'est une leçon pour la prochaine fois.

c) Setobject :

Tout d'abord, on crée un tableau qui prendra le nom de tous les objets de la map.

```
216       char ** names = (char **)malloc(nbObject * sizeof(char *));
217       int len;
218       for (int i = 0; i < nbObject; i++) {
219         read(file,&len, sizeof(int));
220         names[i] = (char *)malloc(len * sizeof(char) + 1);
221         for (int j = 0; j < len; j++) {
222           read(file, &names[i][j], sizeof(char));
223         }
224         names[i][len] = '\0';
225         lseek(file,5 * sizeof(int),SEEK_CUR);
226       }
```

On vérifie que l'objet donné en paramètre n'existe pas déjà, auquel cas on ne fait rien.

```
236       int * res = (int *)malloc(num * sizeof(int));
237       for (int i = 0; i < num; i++) {
238         res[i] = 1;
239         const char * name = argv[3 + 6 * i];
240         for (int j = 0; j < nbObject; j++) {
241           if (strcmp(name, names[j]) == 0) {
242             res[i] = 0;
243             numFound++;
244             break;
245           }
246         }
247       }
```

On crée un fichier temporaire dans lequel on va écrire la taille, le nom et toutes les informations de l'objet que l'on concatène ensuite au fichier principal.

d) Pruneobjects :

On crée un fichier temporaire. Une fois arrivé dans la partie map du fichier, on compte les différents éléments de map et on compte leurs récurrences.

```
121       int *res = (int *)malloc(nbObject * sizeof(int));
122       memset(res, 0, nbObject * sizeof(int));
123       while(read(file,&value,sizeof(int)) != 0){
124         if (value < 0) {
125           continue;
126         }
127         res[value] = 1;
128       }
129       int num = 0;
130       for (int i = 0; i < nbObject; i++){
131         if (res[i] == 0){
132           num++;
133         }
134       }
```

Si la récurrence de l'objet est à 0 on ne le rajoute pas dans le fichier temporaire :

```
195       while(read(file,&val,sizeof(int)) != 0){
196         if (val == -1) {
197           write(fileTMP, &val, sizeof(int));
198         }
199         else {
200           write(fileTMP, &res[val], sizeof(int));
201         }
202       }
```

Et on concatene au fichier principal.

```
203       char order[50] = {'\0'};
204       strcat(order, "mv tmp.map ");
205       strcat(order, argv[1]);
206       system(order);
207       free(res);
208       return 0;
209     }
```

III- Gestion des temporisateurs

La gestion des temporisateurs a été un moment assez compliquée car en dehors des parties très copier-collées tel que les initialisations des handlers et des threads, du timer et du masque, nous nous sommes retrouvé face à un bug dont nous n'avons toujours pas trouvé la source.

Nous avons créé une structure LISTE :

```
37  LISTE* new_event(long timer,void* event){
38    LISTE* lBis = (LISTE*) malloc(sizeof(LISTE));
39    lBis->evenement=event;
40    lBis->tps=timer;
41    lBis->next_event=NULL;
42    return lBis;
43  }
```

C'est grâce à cette structure que nous gérons les différents événements sur la map. Nous avons ensuite implémenté une fonction new_event qui crée un nouvel évenement. La fonction untilNext permet, elle, d'atteindre le dernier élément de la liste sans déranger à l'ordre chaîné. A l'arrivé d'un nouvel élément, si la liste d'événement est vide on ajoute un event, sinon on utilise UntilNext pour l'ajouter à la fin.

Nous en arrivons à l'initialisation du timer et l'appel à la fonction setitimer qui a été facile d'utilisation grâce aux différentes ressources sur le net.

```
107     struct itimerval timer;
108     timer.it_interval.tv_sec = 0;
109     timer.it_interval.tv_usec = 0;
110
111     timer.it_value.tv_sec = delay/1000;
112     timer.it_value.tv_usec = (delay%1000)*1000;
113
114     pthread_mutex_lock(&mutex);
115     setitimer(ITIMER_REAL, &timer, NULL);
116     pthread_mutex_unlock(&mutex);
117     return (timer_id_t) NULL;
```

Le lock du mutex vient de l'idée que peut-être que sur plusieurs threads, il était possible que le compte à rebours d'un timer puisse être influencé par un autre thread. Cette idée nous est venue à cause du bug auquel on a fait face. En effet, lorsque plusieurs événements s'enchaînent sur la liste chaînée alors le dernier événement ne se déclenche pas. Etonnamment, le SIGALARM qu'il devrait envoyé est interrompu ou récupéré par les autres éléments, laissant une mine ou une bombe en l'attente d'un signal qui ne vient jamais ou seulement si l'on prépare une nouvelle bombe. De ce fait, le timer_set de notre projet est non fonctionnel malgré tout ce que l'on a essayé pour débugger ce problème.

De l'autre côté, le handler envoie sdl_push_event à chaque SIGALARM et fait avancer la liste chaînée.

IV- Conclusion

Le projet était une belle application de tout ce qui a été appris durant ce semestre. Pour qui est de la situation finale globale, il est clair que nous ne sommes pas fiers du résultat. Le projet a été pris au sérieux mais nous avons mal structuré notre manière de travailler, attaquant trop tardivement le projet car nous n'avions pas su balancer avec les autres matières. Par ailleurs, on dénote un certain manque de pratique de chacun dans la compréhension des différents chapitres. Néanmoins, ce projet nous a permis de mettre en avant des lacunes importantes de chacun, par rapport à la quantité de travail à fournir dans la formation.

Merci à vous pour l'encadrement et l'attention que vous avez apporté à notre travail.