

Code source projet de programmation système 2016/2017

Groupe IN501A2  
Gautier DELACOUR  
Bérangère SUBERVIE  
Maxime VALLERON

6 janvier 2017

# Table des matières

<b>1</b>	<b>mapio.c</b>	<b>2</b>
<b>2</b>	<b>maputil.c</b>	<b>6</b>
<b>3</b>	<b>tempo.c</b>	<b>20</b>

# Chapitre 1

## mapio.c

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include "map.h"
7 #include "error.h"
8
9 #ifdef PADAWAN
10
11 void map_new(unsigned width, unsigned height){
12     map_allocate(width, height);
13
14     // Sol
15
16     for(int x = 0; x < width; x++){
17         map_set(x, height - 1, 0);
18
19         // Mur
20
21         for(int y = 0; y < height - 1; y++){
22             map_set(0, y, 1);
23             map_set(width - 1, y, 1);
24         }
25
26         map_object_begin(6);
27
28         // Ajout des differents objets utilises
29
30         map_object_add("images/ground.png", 1, MAP_OBJECT_SOLID);
31         map_object_add("images/wall.png", 1, MAP_OBJECT_SOLID);
32         map_object_add("images/grass.png", 1, MAP_OBJECT_SEMI_SOLID);
33         map_object_add("images/marble.png", 1, MAP_OBJECT_SOLID | MAP_OBJECT_DESTRUCTIBLE);
34         map_object_add("images/flower.png", 1, MAP_OBJECT_AIR);
35         map_object_add("images/coin.png", 20, MAP_OBJECT_AIR | MAP_OBJECT_COLLECTIBLE);
36
37         map_object_end();
38     }
39
40
41 void map_save(char *filename){
42     int valeur, err, tmp;
43     int height = map_height();
44     int width = map_width();
45     int frame, type1, type2, type3, type4;
46     char name[100];
47     int nb_object = map_objects();
48
49 }
```

```

50 // Ouverture du fichier de sauvgarde
51
52 int output = open("maps/saved.map", O_TRUNC | O_WRONLY | O_CREAT, 0666);
53
54 if(output == -1){
55     fprintf(stderr, "Probleme_dans_maps/saved.map: %s\n", filename);
56     exit(1);
57 }
58
59 lseek(output, 0, SEEK_SET);
60
61 // Sauvgarde de la taille
62
63 write(output, &width, sizeof(int));
64 write(output, &height, sizeof(int));
65
66 // Sauvgarde des objets
67
68 write(output, &nb_object, sizeof(int));
69
70 for(int i = 0; i < nb_object; i++){
71     frame = map_get_frames(i);
72     strcpy(name, map_get_name(i));
73     tmp = strlen(name);
74     write(output, &tmp, sizeof(int));
75     for(int j = 0; j < tmp; j++){
76         write(output, &name[j], sizeof(int));
77     }
78     type1 = map_get_solidity(i);
79     type2 = map_is_destructible(i);
80     type3 = map_is_collectible(i);
81     type4 = map_is_generator(i);
82     write(output, &frame, sizeof(int));
83     write(output, &type1, sizeof(int));
84     write(output, &type2, sizeof(int));
85     write(output, &type3, sizeof(int));
86     write(output, &type4, sizeof(int));
87 }
88
89 // Lecture de map
90
91 for(int y = 0; y < height; y++){
92     for(int x = 0; x < width; x++){
93         valeur = map_get(x, y);
94         err = write(output, &valeur, sizeof(int));
95         if(err == -1){
96             fprintf(stderr, "Probleme_de_sauvegarde: %s\n", filename);
97             exit(1);
98         }
99     }
100 }
101
102 close(output);
103 }
104
105
106 void map_load(char *filename){
107     int err, fd;
108     char n, type1, type2, type3, type4;
109     char *adress = malloc(sizeof(char));
110
111     // Ouverture du fichier de sauvgarde
112
113     fd = open(filename, O_RDONLY, 0666);
114
115     if(fd == -1){
116         fprintf(stderr, "Desole: %s_n'existe_pas\n", filename);
117     }

```

```

118     exit(1);
119 }
120
121 err = lseek(fd, 0, SEEK_SET);
122
123 if(err == -1){
124     fprintf(stderr, "Probleme_de_format:_%s\n", filename);
125     exit(1);
126 }
127
128 // Chargement de la taille
129
130 err = read(fd, &n, sizeof(int));
131
132 if(err == -1){
133     fprintf(stderr, "Probleme_de_format:_%s\n", filename);
134     exit(1);
135 }
136
137 unsigned width = n;
138 err = read(fd, &n, sizeof(int));
139
140 if(err == -1){
141     fprintf(stderr, "Probleme_de_format:_%s\n", filename);
142     exit(1);
143 }
144
145 unsigned height = n;
146 map_allocate(width, height);
147 err = read(fd, &n, sizeof(int));
148
149 if(err == -1){
150     fprintf(stderr, "Probleme_de_format:_%s\n", filename);
151     exit(1);
152 }
153
154 int nb_object = n;
155
156 // Chargement des objets
157
158 map_object_begin(nb_object);
159 char *tmp1 = malloc(sizeof(char) * 20);
160
161 for(int i = 0; i < nb_object; i++){
162     err = read(fd, &adress[0], sizeof(int));
163     if(err == -1){
164         fprintf(stderr, "Probleme_de_format:_%s\n", filename);
165         exit(1);
166     }
167     tmp1 = realloc(tmp1, ((int)adress[0]) * sizeof(char));
168     for(int j = 0; j < (int)adress[0]; j++){
169         err = read(fd, &tmp1[j], sizeof(int));
170         if(err == -1){
171             fprintf(stderr, "Probleme_de_format:_%s\n", filename);
172             exit(1);
173         }
174     }
175     err = read(fd, &n, sizeof(int));
176     if(err == -1){
177         fprintf(stderr, "Probleme_de_format:_%s\n", filename);
178         exit(1);
179     }
180     err = read(fd, &type1, sizeof(int));
181     if(err == -1){
182         fprintf(stderr, "Probleme_de_format:_%s\n", filename);
183         exit(1);
184     }
185     err = read(fd, &type2, sizeof(int));

```

```

186     if(err == -1){
187         fprintf(stderr, "Probleme_de_format:%s\n", filename);
188         exit(1);
189     }
190     err = read(fd, &type3, sizeof(int));
191     if(err == -1){
192         fprintf(stderr, "Probleme_de_format:%s\n", filename);
193         exit(1);
194     }
195     err = read(fd, &type4, sizeof(int));
196     if(err == -1){
197         fprintf(stderr, "Probleme_de_format:%s\n", filename);
198         exit(1);
199     }
200     if(type2){
201         map_object_add(tmp1, n, type1 | MAP_OBJECT_DESTRUCTIBLE);
202     }
203     else if(type3){
204         map_object_add(tmp1, n, type1 | MAP_OBJECT_COLLECTIBLE);
205     }
206     else if(type4){
207         map_object_add(tmp1, n, type1 | MAP_OBJECT_GENERATOR);
208     }
209     else{
210         map_object_add(tmp1, n, type1);
211     }
212     for(int k = 0; k < adress[0]; k++){
213         strcpy(tmp1 + k, "");
214     }
215 }
216
217 map_object_end ();
218 free(tmp1);
219 free(adress);
220
221 // Ecriture de map
222
223 for (int y = 0; y < height; y++){
224     for (int x = 0; x < width; x++){
225         err = read(fd, &n, sizeof(int));
226         if(err == -1){
227             fprintf(stderr, "Probleme_de_format:%s\n", filename);
228             exit(1);
229         }
230         map_set(x, y, n);
231     }
232 }
233
234 close(fd);
235 }
236
237 #endif

```

# Chapitre 2

## maputil.c

```
1 #define _XOPEN_SOURCE 500
2
3 #include <fcntl.h>
4 #include <stdio.h>
5 #include <unistd.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <sys/types.h>
9 #include "map.h"
10 #include "error.h"
11
12 #ifdef PADAWAN
13
14 #define NB_OPTIONS 8
15
16 static void usage(char *arg){
17     fprintf(stderr, "%s_<file>_<option>\n", arg);
18     exit(EXIT_FAILURE);
19 }
20
21
22
23 // Alloue un tableau contenant toutes les options possibles de maputil
24
25 void optionsAlloc(char *t[]){
26     t[0] = malloc(10 * sizeof(char));
27     strcpy(t[0], "—getwidth");
28     t[1] = malloc(11 * sizeof(char));
29     strcpy(t[1], "—getheight");
30     t[2] = malloc(12 * sizeof(char));
31     strcpy(t[2], "—getobjects");
32     t[3] = malloc(9 * sizeof(char));
33     strcpy(t[3], "—getinfo");
34     t[4] = malloc(10 * sizeof(char));
35     strcpy(t[4], "—setwidth");
36     t[5] = malloc(11 * sizeof(char));
37     strcpy(t[5], "—setheight");
38     t[6] = malloc(12 * sizeof(char));
39     strcpy(t[6], "—setobjects");
40     t[7] = malloc(14 * sizeof(char));
41     strcpy(t[7], "—pruneobjects");
42 }
43
44
45
46 // Libere le tableau contenant toutes les options de maputil
47
48 void optionsFree(char *t[]){
49     for(int i = 0; i < NB_OPTIONS; i++)
```

```

50     free(t[i]);
51 }
52
53
54
55 // Renvoie la largeur d'une carte
56
57 int getWidth(int Fd){
58     int i;
59     int e = lseek(Fd, 0, SEEK_SET);
60
61     if(e == -1){
62         perror("lseek");
63         exit(EXIT_FAILURE);
64     }
65
66     e = read(Fd, &i, sizeof(int));
67     if(e == -1){
68         perror("read");
69         exit(EXIT_FAILURE);
70     }
71
72     return i;
73 }
74
75
76
77 // Renvoie la hauteur d'une carte
78
79 int getHeight(int Fd){
80     int i;
81     int e = lseek(Fd, sizeof(int), SEEK_SET);
82
83     if(e == -1){
84         perror("lseek");
85         exit(EXIT_FAILURE);
86     }
87
88     e = read(Fd, &i, sizeof(int));
89
90     if(e == -1){
91         perror("read");
92         exit(EXIT_FAILURE);
93     }
94
95     return i;
96 }
97
98
99
100 // Renvoie le nombre d'objets d'une carte
101
102 int getObjects(int Fd){
103     int i;
104     int e = lseek(Fd, 2 * sizeof(int), SEEK_SET);
105
106     if(e == -1){
107         perror("lseek");
108         exit(EXIT_FAILURE);
109     }
110
111     e = read(Fd, &i, sizeof(int));
112
113     if(e == -1){
114         perror("read");
115         exit(EXIT_FAILURE);
116     }
117

```



```

118     return i;
119 }
120
121
122
123 // Change l'ancienne largeur par la nouvelle w
124
125 void setWidth(int Fd, int w){
126     if(16 <= w && w <= 1024){
127         int oldW = getWidth(Fd);
128         if(oldW == w)
129             return;
130         int h = getHeight(Fd);
131         int j = 0;
132         int lenName, k;
133         int t[h * w];
134         int nbObjects = getObjects(Fd);
135         int nbCaracObj = 5;
136         int e = lseek(Fd, 0, SEEK_SET);
137         if(e == -1){
138             perror("lseek");
139             exit(EXIT_FAILURE);
140         }
141         e = write(Fd, &w, sizeof(int));
142         if(e == -1){
143             perror("write");
144             exit(EXIT_FAILURE);
145         }
146
147         // Place le curseur au debut de la liste des objets
148
149         e = lseek(Fd, 3 * sizeof(int), SEEK_SET);
150         if(e == -1){
151             perror("lseek");
152             exit(EXIT_FAILURE);
153         }
154
155         // Recupere la taille du nom de chaque fichier dans lenName, et decale de cette taille
156         // plus le nombre de caracteristiques des objets
157
158         for(int i = 0; i < nbObjects; i++){
159             e = read(Fd, &lenName, sizeof(int));
160             if(e == -1){
161                 perror("read");
162                 exit(EXIT_FAILURE);
163             }
164             e = lseek(Fd, (lenName + nbCaracObj) * sizeof(int), SEEK_CUR);
165             if(e == -1){
166                 perror("lseek");
167                 exit(EXIT_FAILURE);
168             }
169         }
170
171         // Recopie des elements communs aux nouvelles et anciennes tailles
172
173         for(int y = 0; y < h; y++){
174             for(int x=0;x<oldW;x++){
175                 if(x < w){
176                     e = read(Fd, &(t[j]), sizeof(int));
177                     if(e == -1){
178                         perror("read");
179                         exit(EXIT_FAILURE);
180                     }
181                     j++;
182                 }
183             }
184             // Si la taille est retrecie

```

```

185
186     if(oldW > w)
187         lseek(Fd, (oldW - w) * sizeof(int), SEEK_CUR);
188
189     // Si la taille est augmentee
190
191     if(oldW < w){
192         for(k = j; k < j + (w - oldW); k++){
193             t[k] = MAP_OBJECT_NONE;
194         }
195         j = k;
196     }
197 }
198
199 // Place le curseur au debut de la liste des objets
200
201 e = lseek(Fd, 3 * sizeof(int), SEEK_SET);
202 if(e == -1){
203     perror("lseek");
204     exit(EXIT_FAILURE);
205 }
206
207 // Recupere la taille du nom de chaque fichier dans lenName, et decale de cette taille
208 // plus le nombre de caracteristiques des objets
209
210 for(int i = 0; i < nbObjects; i++){
211     e = read(Fd, &lenName, sizeof(int));
212     if(e == -1){
213         perror("read");
214         exit(EXIT_FAILURE);
215     }
216     e = lseek(Fd, (lenName + nbCaracObj) * sizeof(int), SEEK_CUR);
217     if(e == -1){
218         perror("lseek");
219         exit(EXIT_FAILURE);
220     }
221 }
222
223 // Ecrit les elements du tableau dans le fichier Fd
224
225 for(int y = 0; y < h; y++){
226     for(int x = 0; x < w; x++){
227         e = write(Fd, t + (y * w + x), sizeof(int));
228         if(e == -1){
229             perror("write");
230             exit(EXIT_FAILURE);
231         }
232     }
233 }
234
235 // Tronque le fichier s'il est plus petit
236
237 if(oldW > w){
238     int offset = lseek(Fd, oldW - w, SEEK_END);
239     ftruncate(Fd, offset);
240 }
241 else
242     printf("Nouvelle_largeur_non_autorisee!\n");
243 }
244
245
246 // Change l'ancienne hauteur par la nouvelle h
247
248 void setHeight(int Fd, int h){
249     if(12 <= h && h <= 20){
250         int oldH = getHeight(Fd);

```

```

252     if(oldH == h)
253         return;
254     int w = getWidth(Fd);
255     int j = 0;
256     int lenName, k;
257     int t[h * w];
258     int nbObjects = getObjects(Fd);
259     int nbCaracObj = 5;
260     int e = lseek(Fd, sizeof(int), SEEK_SET);
261     if(e == -1){
262         perror("lseek");
263         exit(EXIT_FAILURE);
264     }
265     e = write(Fd, &h, sizeof(int));
266     if(e == -1){
267         perror("write");
268         exit(EXIT_FAILURE);
269     }
270
271     // Place le curseur au debut de la liste des objets
272
273     e = lseek(Fd, 3 * sizeof(int), SEEK_SET);
274     if(e == -1){
275         perror("lseek");
276         exit(EXIT_FAILURE);
277     }
278
279     // Recupere la taille du nom de chaque fichier dans lenName, et decale de cette taille
280     // plus le nombre de caracteristiques des objets
281
282     for(int i = 0; i < nbObjects; i++){
283         e = read(Fd, &lenName, sizeof(int));
284         if(e == -1){
285             perror("read");
286             exit(EXIT_FAILURE);
287         }
288         e = lseek(Fd, (lenName + nbCaracObj) * sizeof(int), SEEK_CUR);
289         if(e == -1){
290             perror("lseek");
291             exit(EXIT_FAILURE);
292         }
293     }
294
295     // Recopie des elements communs aux nouvelles et anciennes tailles
296
297     int tmpH = 0;
298     if( oldH > h)
299         tmpH = oldH + 1;
300     else
301         tmpH = h;
302     for(int y = 0; y < tmpH; y++){
303
304         // Si la taille est augmentee
305
306         if(y < h - oldH){
307             for(k = 0; k < w; k++){
308                 t[k + y * w] = MAP_OBJECT_NONE;
309             }
310             j = k + y * w;
311         }
312         else if( y > oldH - h){
313             for(int x = 0; x < w; x++){
314                 e = read(Fd, &(t[j]), sizeof(int));
315                 if(e == -1){
316                     perror("read");
317                     exit(EXIT_FAILURE);
318                 }
319                 j++;

```

```

319     }
320 }
321
322 // Si la taille est retrecie
323
324 else if(y < oldH - h){
325     lseek(Fd, w * sizeof(int), SEEK_CUR);
326 }
327 }
328
329 // Place le curseur au debut de la liste des objets
330
331 e = lseek(Fd, 3 * sizeof(int), SEEK_SET);
332 if(e == -1){
333     perror("lseek");
334     exit(EXIT_FAILURE);
335 }
336
337 // Recupere la taille du nom de chaque fichier dans lenName, et decale de cette taille
338 // plus le nombre de caracteristiques des objets
339
340 for(int i = 0; i < nbObjects; i++){
341     e = read(Fd, &lenName, sizeof(int));
342     if(e == -1){
343         perror("read");
344         exit(EXIT_FAILURE);
345     }
346     e = lseek(Fd, (lenName + nbCaracObj) * sizeof(int), SEEK_CUR);
347     if(e == -1){
348         perror("lseek");
349         exit(EXIT_FAILURE);
350     }
351 }
352
353 // Ecrit les elements du tableau dans le fichier Fd
354
355 for(int y = 0; y < h; y++){
356     for(int x = 0; x < w; x++){
357         e = write(Fd, t + (y * w + x), sizeof(int));
358         if(e == -1){
359             perror("write");
360             exit(EXIT_FAILURE);
361         }
362     }
363 }
364
365 // Tronque le fichier s'il est plus petit
366
367 if(oldH > h){
368     int offset = lseek(Fd, (oldH - h), SEEK_END);
369     ftruncate(Fd, offset);
370 }
371 else
372     printf("Nouvelle_hauteur_non_autorisee!\n");
373 }
374
375
376
377 void setObjects(int Fd, char *name, int frame, int solid, int destructible, int collectible,
378               int generator){
379     int nbObjects = getObjects(Fd);
380     int w = getWidth(Fd);
381     int h = getHeight(Fd);
382     int t[h * w];
383
384     // Place le curseur au debut de la liste des objets

```

```

385 int e = lseek(Fd, 3 * sizeof(int), SEEK_SET);
386
387 if(e== -1){
388     perror("lseek");
389     exit(EXIT_FAILURE);
390 }
391
392 char *adress = malloc(sizeof(char));
393 char *tmp1 = malloc(sizeof(char) * 20);
394
395 for(int i = 0; i < nbObjects; i++){
396     e = read(Fd, &adress[0], sizeof(int));
397     if(e == -1){
398         fprintf(stderr, "Probleme_de_format_du_fichier_de_sauvegarde\n");
399         exit(1);
400     }
401     tmp1 = realloc(tmp1, ((int)adress[0]) * sizeof(char));
402     for(int j = 0; j < ((int)adress[0]); j++){
403         e = read(Fd, &tmp1[j], sizeof(int));
404         if(e == -1){
405             fprintf(stderr, "Probleme_de_format_du_fichier_de_sauvegarde\n");
406             exit(1);
407         }
408     }
409     if(strcmp(tmp1, name) == 0){
410         e = write(Fd, &frame, sizeof(int));
411         if(e == -1){
412             perror("write");
413             exit(1);
414         }
415         e = write(Fd, &solid, sizeof(int));
416         if(e == -1){
417             perror("write");
418             exit(1);
419         }
420         e = write(Fd, &destructible, sizeof(int));
421         if(e == -1){
422             perror("write");
423             exit(1);
424         }
425         e = write(Fd, &collectible, sizeof(int));
426         if(e == -1){
427             perror("write");
428             exit(1);
429         }
430         e = write(Fd, &generator, sizeof(int));
431         if(e == -1){
432             perror("write");
433             exit(1);
434         }
435         free(tmp1);
436         free(adress);
437         return;
438     }
439     else{
440         e = lseek(Fd, 5 * sizeof(int), SEEK_CUR);
441         if(e == -1){
442             perror("lseek");
443             exit(EXIT_FAILURE);
444         }
445     }
446     for(int k = 0; k < adress[0]; k++){
447         strcpy(tmp1 + k, "");
448     }
449
450     for(int y = 0; y < h; y++){
451         for(int x = 0; x < w; x++){
452             e = read(Fd, t + (y * w + x), sizeof(int));

```

```

453     if(e == -1){
454         perror("read");
455         exit(EXIT_FAILURE);
456     }
457 }
458 }
459
460 e = lseek (Fd, 2 * sizeof(int), SEEK_SET);
461
462 if(e == -1){
463     perror("lseek");
464     exit(EXIT_FAILURE);
465 }
466
467 nbObjects++;
468 e = write(Fd, &nbObjects, sizeof(int));
469
470 if(e == -1){
471     perror("write");
472     exit(1);
473 }
474
475 int lenName;
476 e = lseek(Fd, 3 * sizeof(int), SEEK_SET);
477
478 if(e == -1){
479     perror("lseek");
480     exit(EXIT_FAILURE);
481 }
482
483 for(int i = 0; i < nbObjects - 1; i++){
484     e = read(Fd, &lenName, sizeof(int));
485     if(e == -1){
486         perror("read");
487         exit(EXIT_FAILURE);
488     }
489     e = lseek(Fd, (lenName + 5) * sizeof(int), SEEK_CUR);
490     if(e == -1){
491         perror("lseek");
492         exit(EXIT_FAILURE);
493     }
494 }
495
496 lenName = strlen(name);
497 e = write(Fd, &lenName, sizeof(int));
498
499 if(e == -1){
500     perror("write");
501     exit(1);
502 }
503
504 for(int j = 0; j < lenName; j++){
505     e = write(Fd, &name[j], sizeof(int));
506     if(e == -1){
507         perror("write");
508         exit(1);
509     }
510 }
511
512 if(e == -1){
513     perror("write");
514     exit(1);
515 }
516
517 e = write(Fd, &frame, sizeof(int));
518
519 if(e == -1){
520     perror("write");

```

```

521     exit(1);
522 }
523
524 e = write(Fd, &solid , sizeof(int));
525
526 if(e == -1){
527     perror("write");
528     exit(1);
529 }
530
531 e = write(Fd, &destructible , sizeof(int));
532
533 if(e == -1){
534     perror("write");
535     exit(1);
536 }
537
538 e = write(Fd, &collectible , sizeof(int));
539
540 if(e == -1){
541     perror("write");
542     exit(1);
543 }
544
545 e = write(Fd, &generator , sizeof(int));
546
547 if(e == -1){
548     perror("write");
549     exit(1);
550 }
551
552 for(int y = 0; y < h; y++){
553     for(int x = 0; x < w; x++){
554         e = write(Fd, t + (y * w + x) , sizeof(int));
555         if(e == -1){
556             perror("write");
557             exit(EXIT_FAILURE);
558         }
559     }
560 }
561
562 free(tmp1);
563 free(adress);
564 }
565
566
567
568 void pruneOjects(int Fd){
569     int nbObjects = getObjects(Fd);
570     int nbObjectsTmp = 0;
571     int t[nbObjects];
572     int tlenName[nbObjects];
573     char **tname = malloc(sizeof(char *) * nbObjects);
574
575     for(int k = 0; k < nbObjects; k++){
576         tname[k] = malloc(sizeof(char) * 20);
577
578         int tframe[nbObjects];
579         int tsolid[nbObjects];
580         int tdestructible[nbObjects];
581         int tcollectible[nbObjects];
582         int tgenerator[nbObjects];
583
584         for(int i = 0; i < nbObjects; i++){
585             t[i] = 0;
586
587             int w = getWidth(Fd);
588             int h = getHeight(Fd);

```

```

589 int tab[w * h];
590 int object;
591 int e = lseek(Fd, 3 * sizeof(int), SEEK_SET);
592
593 if(e== -1){
594     perror("lseek");
595     exit(EXIT_FAILURE);
596 }
597
598 for(int i = 0; i < nbObjects; i++){
599     e = read(Fd, &tlenName[i], sizeof(int));
600     if(e == -1){
601         perror("read1");
602         exit(EXIT_FAILURE);
603     }
604     for(int j = 0; j < tlenName[i]; j++){
605         e = read(Fd, tname[i] + j, sizeof(int));
606         if(e == -1){
607             perror("read2");
608             exit(EXIT_FAILURE);
609         }
610     }
611     e = read(Fd, &tframe[i], sizeof(int));
612     if(e == -1){
613         perror("read3");
614         exit(EXIT_FAILURE);
615     }
616     e = read(Fd, &tsolid[i], sizeof(int));
617     if(e == -1){
618         perror("read4");
619         exit(EXIT_FAILURE);
620     }
621     e = read(Fd, &tdestructible[i], sizeof(int));
622     if(e == -1){
623         perror("read5");
624         exit(EXIT_FAILURE);
625     }
626     e = read(Fd, &tcollectible[i], sizeof(int));
627     if(e == -1){
628         perror("read6");
629         exit(EXIT_FAILURE);
630     }
631     e = read(Fd, &tgenerator[i], sizeof(int));
632     if(e == -1){
633         perror("read7");
634         exit(EXIT_FAILURE);
635     }
636 }
637
638 for(int y = 0; y < h; y++){
639     for(int x = 0; x < w; x++){
640         e = read(Fd, &object, sizeof(int));
641         if(e == -1){
642             perror("write");
643             exit(EXIT_FAILURE);
644         }
645         tab[x + y * w] = object;
646         if(object != -1 && t[object] != 1){
647             t[object] = 1;
648             nbObjectsTmp++;
649         }
650     }
651 }
652
653 e = lseek(Fd, 2 * sizeof(int), SEEK_SET);
654
655 if(e == -1){
656     perror("lseek");

```



```

657     exit(EXIT_FAILURE);
658 }
659
660 e = write(Fd, &nbObjectsTmp, sizeof(int));
661
662 if(e == -1){
663     perror("write");
664     exit(EXIT_FAILURE);
665 }
666
667 int tmp = 0;
668
669 for(int i = 0; i < nbObjects; i++){
670     if(t[i]){
671         e = write(Fd, &tlenName[i], sizeof(int));
672         if(e == -1){
673             perror("write");
674             exit(EXIT_FAILURE);
675         }
676         for(int j = 0; j < tlenName[i]; j++){
677             e = write(Fd, &tname[i][j], sizeof(int));
678             if(e == -1){
679                 perror("write");
680                 exit(EXIT_FAILURE);
681             }
682         }
683         e = write(Fd, &tframe[i], sizeof(int));
684         if(e == -1){
685             perror("write");
686             exit(EXIT_FAILURE);
687         }
688         e = write(Fd, &tsolid[i], sizeof(int));
689         if(e == -1){
690             perror("write");
691             exit(EXIT_FAILURE);
692         }
693         e = write(Fd, &tdestructible[i], sizeof(int));
694         if(e == -1){
695             perror("write");
696             exit(EXIT_FAILURE);
697         }
698         e = write(Fd, &tcollectible[i], sizeof(int));
699         if(e == -1){
700             perror("write");
701             exit(EXIT_FAILURE);
702         }
703         e = write(Fd, &tgenerator[i], sizeof(int));
704         if(e == -1){
705             perror("write");
706             exit(EXIT_FAILURE);
707         }
708         t[i] = tmp;
709         tmp++;
710     }
711 }
712
713 for(int y = 0; y < h; y++){
714     for(int x = 0; x < w; x++){
715         if(tab[y * w + x] == -1)
716             e = write(Fd, &tab[y * w + x], sizeof(int));
717         else
718             e = write(Fd, &t[tab[y * w + x]], sizeof(int));
719         if(e == -1){
720             perror("write");
721             exit(EXIT_FAILURE);
722         }
723     }
724     printf("\n");

```

```

725 }
726 }
727
728
729
730 // Teste la correspondance entre l'option demandee et les options existantes, et appelle une
    fonction correspondante si elle existe
731
732 int traitementOption(char *optTab[], int Fd, char *argv[], int k, int argc){
733     char *option = argv[k];
734     char *arg;
735
736     if(k < argc)
737         arg = argv[k + 1];
738
739     int n = 0;
740     printf("\nOption choisie : \t");
741
742     // getwidth
743
744     if(!strcmp(option, optTab[0])){
745         printf("%s\t", optTab[0]);
746         printf("%d\n", getWidth(Fd));
747         n = 1;
748     }
749
750     // getheight
751
752     else if(!strcmp(option, optTab[1])){
753         printf("%s\t", optTab[1]);
754         printf("%d\n", getHeight(Fd));
755         n = 1;
756     }
757
758     // getobjects
759
760     else if(!strcmp(option, optTab[2])){
761         printf("%s\t", optTab[2]);
762         printf("%d\n", getObjects(Fd));
763         n = 1;
764     }
765
766     // getinfo
767
768     else if(!strcmp(option, optTab[3])){
769         printf("%s\t", optTab[3]);
770         printf("Largeur : %d\tHauteur : %d\tNombre d'objets : %d\n", getWidth(Fd), getHeight(Fd),
771             , getObjects(Fd));
772         n = 1;
773     }
774
775     // setwidth
776
777     else if(!strcmp(option, optTab[4])){
778         printf("%s\t", optTab[4]);
779         int w = atoi(arg);
780         printf("%d\n", w);
781         setWidth(Fd, w);
782         n = 2;
783     }
784
785     // setheight
786
787     else if(!strcmp(option, optTab[5])){
788         printf("%s\t", optTab[5]);
789         int h = atoi(arg);
790         printf("%d\n", h);
791         setHeight(Fd, h);

```

```

791     n = 2;
792 }
793
794 // setobjects
795
796 else if(!strcmp(option, optTab[6])){
797     printf("%s\n", optTab[6]);
798     if((argc - 3) % 6 != 0){
799         fprintf(stderr, "Erreur, _nombre_d'arguments_non_valide\n");
800         exit(1);
801     }
802     char *name = argv[k + 1];
803     arg = argv[k + 2];
804     int frame = atoi(arg);
805     arg = argv[k + 3];
806     int solid, destructible, collectible, generator;
807     if(!strcmp(argv[k + 3], "solid"))
808         solid = 2;
809     else if(!strcmp(argv[k + 3], "semi_solid"))
810         solid = 1;
811     else if(!strcmp(argv[k + 3], "air"))
812         solid = 0;
813     else{
814         fprintf(stderr, "Erreur, _arguments_non_valide:\t solid/semi_solid/air\n\n");
815         exit(1);
816     }
817     if(!strcmp(argv[k + 4], "destructible"))
818         destructible = 1;
819     else if(!strcmp(argv[k+4], "not-destructible"))
820         destructible = 0;
821     else{
822         fprintf(stderr, "Erreur, _arguments_non_valide:\t destructible/not-destructible\n\n");
823         exit(1);
824     }
825     if(!strcmp(argv[k + 5], "collectible"))
826         collectible = 1;
827     else if(!strcmp(argv[k + 5], "not-collectible"))
828         collectible = 0;
829     else{
830         fprintf(stderr, "Erreur, _arguments_non_valide:\t collectible/not-collectible\n\n");
831         exit(1);
832     }
833     if(!strcmp(argv[k + 6], "generator"))
834         generator = 1;
835     else if(!strcmp(argv[k + 6], "not-generator"))
836         generator = 0;
837     else{
838         fprintf(stderr, "Erreur, _arguments_non_valide:\t generator/not-generator\n\n");
839         exit(1);
840     }
841     setObjects(Fd, name, frame, solid, destructible, collectible, generator);
842     n = argc - 2;
843 }
844
845 // pruneobjects
846
847 else if(!strcmp(option, optTab[7])){
848     printf("%s\n", optTab[7]);
849     pruneObjects(Fd);
850     n = 1;
851 }
852 else{
853     printf("\nOption_inconnue!\n");
854     n = 1;
855 }
856
857 printf("\n");
858 return n;

```

```

859 }
860
861
862
863 int main(int argc, char *argv[]) {
864     char *optTab[NB_OPTIONS];
865     int k = 2;
866     int n, Fd;
867
868     if (argc < 3)
869         usage(argv[0]);
870
871     Fd = open(argv[1], O_RDWR);
872
873     if (Fd == -1) {
874         perror("open");
875         exit(EXIT_FAILURE);
876     }
877
878     optionsAlloc(optTab);
879
880     for (int i = 0; i < NB_OPTIONS; i++)
881         printf("%d: %s\n", i, optTab[i]);
882
883     // k est la position de l'option traitée dans argv
884
885     while (k < argc) {
886
887         // n est le nombre d'arguments utilisés
888
889         n = traitementOption(optTab, Fd, argv, k, argc);
890         k += n;
891         if (k >= 2) {
892             printf("maputil ne gère actuellement qu'une seule option à la fois!\n");
893             break;
894         }
895     }
896
897     optionsFree(optTab);
898     close(Fd);
899     return EXIT_SUCCESS;
900 }
901
902 #endif

```

# Chapitre 3

## tempo.c

```
1 #define _XOPEN_SOURCE 700
2
3 #include <SDL.h>
4 #include <unistd.h>
5 #include <stdlib.h>
6 #include <stdio.h>
7 #include <time.h>
8 #include <sys/time.h>
9 #include <signal.h>
10 #include <pthread.h>
11 #include "timer.h"
12
13
14
15 static unsigned long get_time (void){
16     struct timeval tv;
17     gettimeofday (&tv ,NULL);
18
19     // Compte seulement les secondes a partir de 2016
20
21     tv.tv_sec -= 3600UL * 24 * 365 * 46;
22     return tv.tv_sec * 1000000UL + tv.tv_usec;
23 }
24
25
26
27 #ifndef PADAWAN
28
29
30
31 struct evenement{
32     void *parametre;
33     unsigned long temps;
34 };
35
36 struct evenement t[100];
37 int compteur = 0;
38
39
40
41 void trie(struct evenement t[], int compteur){
42     void* tmpparametre = 0;
43     unsigned long tmptemps = 0;
44     for(int i = 0; i < compteur; i++){
45         for(int j = i + 1; j < compteur; j++){
46             if(t[j].temps < t[i].temps){
47                 tmpparametre = t[i].parametre;
48                 tmptemps = t[i].temps;
49                 t[i].parametre = t[j].parametre;
```

```

50         t[i].temps = t[j].temps;
51         t[j].parametre = tmpparametre;
52         t[j].temps = tmptemps;
53     }
54 }
55 }
56 }
57
58
59
60 void traitant(int s){
61     printf ("sdl_push_event(%p) appelee au temps %ld\n", t[0].parametre, get_time ());
62     sdl_push_event(t[0].parametre);
63
64     for(int i = 0; i < compteur + 1; i++){
65         t[i].temps = t[i + 1].temps;
66         t[i].parametre = t[i + 1].parametre;
67     }
68
69     struct itimerval timer;
70     timer.it_interval.tv_sec = 0;
71     timer.it_interval.tv_usec = 0;
72     timer.it_value.tv_sec = (t[0].temps-get_time()) / 1000000;
73     timer.it_value.tv_usec = (t[0].temps-get_time()) % 1000000;
74     int err = setitimer(ITIMER_REAL, &timer, 0);
75
76     if(err){
77         perror("setitimer");
78         exit(1);
79     }
80
81     compteur--;
82 }
83
84
85
86 void *f(void *i){
87     sigset_t mask, empty_mask;
88     sigemptyset(&mask);
89     sigemptyset(&empty_mask);
90     sigaddset(&mask, SIGALRM);
91     sigprocmask(SIG_BLOCK, &mask, NULL);
92     struct sigaction s;
93     s.sa_handler = traitant;
94     sigemptyset(&s.sa_mask);
95     s.sa_flags = 0;
96     sigaction(SIGALRM, &s, NULL);
97
98     while(1){
99         sigsuspend(&empty_mask);
100     }
101 }
102 }
103
104
105
106 // timer_init retourne 1 si les temporisateurs sont totalement implements, sinon retourne 0
107
108 int timer_init (void){
109     pthread_t pid = (pthread_t)NULL;
110     pthread_create(&pid, NULL, f, (void *)pid);
111
112     // L'implementation est prete
113
114     return 1;
115 }
116
117

```

```

118
119 void timer_set (Uint32 delay, void *param){
120     unsigned long time = (unsigned long)(delay * 1000) + get_time();
121     t[compteur].temps = time;
122     t[compteur].parametre = param;
123     compteur++;
124     trie(t, compteur);
125     struct itimerval timer;
126     timer.it_interval.tv_sec = 0;
127     timer.it_interval.tv_usec = 0;
128     timer.it_value.tv_sec = (t[0].temps-get_time()) / 1000000;
129     timer.it_value.tv_usec = (t[0].temps-get_time()) % 1000000;
130     int err = setitimer(ITIMER_REAL, &timer, 0);
131
132     if(err){
133         perror("setitimer");
134         exit(1);
135     }
136 }
137
138
139
140 #endif
141
142 // timerset :enfiler parametre, trie, timer init
143
144 // traitant :trie, timer init

```