Code source du groupe: LEONI Laëtitia, AHMAD BOISSETRI Binzagr et GHIBERT Lucas.

tempo.c

```
#define _XOPEN_SOURCE 600

#include <SDL.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sys/time.h>
#include <signal.h>
#include <pthread.h>
#include <stdbool.h>
#include "timer.h"

// Return number of elapsed µsec since... a long time ago
static unsigned long get_time (void)
{
  struct timeval tv;

  gettimeofday (&tv ,NULL);

  // Only count seconds since beginning of 2016 (not jan 1st, 1970)
  tv.tv_sec -= 3600UL * 24 * 365 * 46;

  return tv.tv_sec * 1000000UL + tv.tv_usec;
}

#ifdef PADAWAN

/*timer values */
typedef struct echeancier {
    long t_initial;
    struct itimerval timer;
    void* parametre;
    bool arme;
    } echeancier;

echeancier tab_echeancier[100];

/*traitant alarm*/
void hand(int sig)
```

```
{
 long t_actuel = get_time();
 int i = 0;
 while((t_actuel - (tab_echeancier[i].t_initial + tab_echeancier[i].timer.it_value.tv_sec *
1000000UL + tab_echeancier[i].timer.it_value.tv_usec)) > 100 || tab_echeancier[i].arme ==
false) {
        i++;
  }
  sdl_push_event(tab_echeancier[i].parametre);
  tab_echeancier[i].arme = false;
}

/*Demon fonction threads */
void *Demon(void *p)
{
  struct sigaction action;
  action.sa_handler = hand;
  sigset_t my_set;
  sigfillset(&my_set);
  sigdelset(&my_set,SIGALRM);
  sigprocmask(SIG_BLOCK, &my_set,NULL);
  sigaction (SIGALRM, &action, NULL);
  while(1){
        sigsuspend(&(my_set));
  }
}


// timer_init returns 1 if timers are fully implemented, 0 otherwise
int timer_init (void)
{
  for(int i = 0; i < 100; i++){
        //initialiser echeance[i]
        tab_echeancier[i].t_initial = 0;
        tab_echeancier[i].timer.it_interval.tv_sec = 0;
        tab_echeancier[i].timer.it_interval.tv_usec = 0;
        tab_echeancier[i].timer.it_value.tv_sec = 0;
        tab_echeancier[i].timer.it_value.tv_usec = 0;
        tab_echeancier[i].parametre = NULL;
        tab_echeancier[i].arme = false;
  }
  pthread_t pid;
  sigset_t pere;
  sigfillset(&pere);
  sigdelset(&pere,SIGALRM);
  sigprocmask(SIG_BLOCK, &pere,NULL);
  pthread_create(&pid,NULL,Demon,NULL);
  return 1;

}

void timer_set (Uint32 delay, void *param)
{ long sec = delay / 1000;
  long usec = (delay - sec*1000) * 1000;
  int i = 0;
  while(tab_echeancier[i].arme != false)
  i++;
  tab_echeancier[i].parametre= param;
  tab_echeancier[i].timer.it_interval.tv_sec=0;
  tab_echeancier[i].timer.it_interval.tv_usec=0;
  tab_echeancier[i].timer.it_value.tv_sec=sec;
  tab_echeancier[i].timer.it_value.tv_usec=usec;
  tab_echeancier[i].parametre = param;
  tab_echeancier[i].arme = true;
  tab_echeancier[i].t_initial = get_time();
  setitimer(ITIMER_REAL, &(tab_echeancier[i].timer), NULL);
}

#endif
```

mapio.c

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#include "map.h"
#include "error.h"

#ifdef PADAWAN

void map_new (unsigned width, unsigned height){
  map_allocate (width, height);

  for (int x = 0; x < width; x++)
        map_set (x, height - 1, 0); // Ground

  for (int y = 0; y < height - 1; y++) {
          map_set (0, y, 1); // Wall
          map_set (width - 1, y, 1); // Wall
  }

  map_object_begin (6);

  // Texture pour le sol
  map_object_add ("images/ground.png", 1, MAP_OBJECT_SOLID);
  // Mur
  map_object_add ("images/wall.png", 1, MAP_OBJECT_SOLID);
  // Gazon
  map_object_add ("images/grass.png", 1, MAP_OBJECT_SEMI_SOLID);
  // Marbre
  map_object_add ("images/marble.png", 1, MAP_OBJECT_SOLID |
MAP_OBJECT_DESTRUCTIBLE);
  // Fleurs
  map_object_add ("images/flower.png", 1, MAP_OBJECT_AIR);
  // Pièce
  map_object_add ("images/coin.png", 20, MAP_OBJECT_AIR |
MAP_OBJECT_COLLECTIBLE);

  map_object_end ();
}

void map_save (char *filename){
    int mapSave = open(filename, O_WRONLY|O_CREAT, 0666);

    int width = map_width();
    int height = map_height();
    int nbObjects = map_objects();

    write(mapSave, &width, sizeof(int));
    write(mapSave, &height, sizeof(int));
    write(mapSave, &nbObjects, sizeof(int));

    int object, endOfMap = -1;
    for(int i=0 ; i<width ; i++)
            for(int j=0 ; j<height ; j++){
                    object = map_get(i,j);
                    if(object!=MAP_OBJECT_NONE){
                            write(mapSave, &i, sizeof(int));
                            write(mapSave, &j, sizeof(int));
                            write(mapSave, &object, sizeof(int));
                    }
            }
    write(mapSave, &endOfMap, sizeof(int));
    int length, framesObj, solidityObj, destructibleObj, collectibleObj, generatorObj;
    for(int i=0 ; i<nbObjects ; i++){
            length = strlen(map_get_name(i));
            framesObj = map_get_frames(i);
            solidityObj = map_get_solidity(i);
            destructibleObj = map_is_destructible(i);
            collectibleObj = map_is_collectible(i);
            generatorObj = map_is_generator(i);

            write(mapSave, &length, sizeof(int));
            write(mapSave, map_get_name(i), strlen(map_get_name(i))*sizeof(char));
            write(mapSave, &framesObj, sizeof(int));
            write(mapSave, &solidityObj, sizeof(int));
            write(mapSave, &destructibleObj, sizeof(int));
            write(mapSave, &collectibleObj, sizeof(int));
            write(mapSave, &generatorObj, sizeof(int));
    }

    printf("Map save performed to the end\n");
}

void map_load (char *filename){
    int mapLoad = open(filename, O_RDONLY, 0666);

    int width, height, nbObjects;
    read(mapLoad, &width, sizeof(int));
    read(mapLoad, &height, sizeof(int));
    read(mapLoad, &nbObjects, sizeof(int));
    map_allocate (width, height);
    int x = 0, y, nameObj;
    while(x!=-1){
            read(mapLoad, &x, sizeof(int));
            if(x!=-1){
                    read(mapLoad, &y, sizeof(int));
                    read(mapLoad, &nameObj, sizeof(int));
                    map_set(x, y, nameObj);
            }
    }
    map_object_begin (nbObjects);
    int length, frame, solidity, destructible, collectible, generator;

    char * name = NULL;
    for(int i=0 ; i<nbObjects ; i++){
            read(mapLoad, &length, sizeof(int));
            name = realloc(name, (length+1)*sizeof(char));
            name[length]='\0';
            read(mapLoad, name, length*sizeof(char));

            read(mapLoad, &frame, sizeof(int));
            read(mapLoad, &solidity, sizeof(int));
            read(mapLoad, &destructible, sizeof(int));
            read(mapLoad, &collectible, sizeof(int));
            read(mapLoad, &generator, sizeof(int));
            map_object_add(name, frame, solidity |
((destructible)?MAP_OBJECT_DESTRUCTIBLE:solidity) |
((collectible)?MAP_OBJECT_COLLECTIBLE:solidity) |
((generator)?MAP_OBJECT_GENERATOR:solidity));
    }
    free(name);
    map_object_end ();
}

#endif
```

maputil.c

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#include "map.h"

void usage2(char * commande);

int setCommande(char * commande){
    int argI = -1;
    if(!strcmp(commande, "--getwidth"))
            argI = 0;
    if(!strcmp(commande, "--getheight"))
            argI = 1;
    if(!strcmp(commande, "--getobjects"))
            argI = 2;
    if(!strcmp(commande, "--getinfo"))
            argI = 3;
    if(!strcmp(commande, "--setwidth"))
            argI = 4;
    if(!strcmp(commande, "--setheight"))
            argI = 5;
    if(!strcmp(commande, "--setobjects"))
            argI = 6;
    if(!strcmp(commande, "--pruneobjects"))
            argI = 7;
    return argI;
}

void getInfos(int argI, char * file){
    int mapFile = open(file, O_RDONLY, 0666);
    int x;
    if(mapFile==-1)
            usage2("Le fichier ne s'est pas ouvert correctement.");
    int k = 1;
    if(argI==1)
            lseek(mapFile, sizeof(int), SEEK_SET);
    if(argI==2)
            lseek(mapFile, 2*sizeof(int), SEEK_SET);
    if(argI==3)
            k = 3;
    for(int i=0 ; i<k ; i++){
            read(mapFile, &x, sizeof(int));
            printf("%d\n", x);
    }
}

void setWidthHeight(char * argument, int mapFile, int fdTmp, int argI){
    int arg = atoi(argument);
    int width_old, height_old;
    int whn, x = 0, y, n, changeOfMap, nbbytes = 1;
    int wall = 1, floor = 0, width, height, x_wall_left = 0, x_wall_right;
    if(!arg)
            usage2("Quatrieme argument invalide (requiert un entier).");
    if(argI==4 && (arg<MIN_WIDTH||arg>MAX_WIDTH))
            usage2("Largeur non valide.");
    if(argI==5 && (arg<MIN_HEIGHT||arg>MAX_HEIGHT))
            usage2("Hauteur non valide.");
    for(int i=0 ; i<3 ; i++){
            read(mapFile, &whn, sizeof(int));
            if(i==0)
                    width_old = whn;
            if(i==1)
                    height_old = whn;
            if((argI==4 && i==0) || (argI==5 && i==1)){
                    write(fdTmp, &arg, sizeof(int));
				if(argI==5)
					changeOfMap = arg-whn;
		}else
				write(fdTmp, &whn, sizeof(int));
	}
	if((argI==4 && arg==width_old) || (argI==5 && arg==height_old))
		usage2("Valeur similaire. Aucun changement effectue.");

	width = arg-1;
	height = height_old-1;
	while(x!=-1){
		read(mapFile, &x, sizeof(int));
		if(x!=-1){
				read(mapFile, &y, sizeof(int));
				read(mapFile, &n, sizeof(int));
				/*retrecissement de la map en largeur*/
				if(argI==4 && arg<width_old){
					if(x<width || (x==width && y==height_old-1)){
						write(fdTmp, &x, sizeof(int));
						write(fdTmp, &y, sizeof(int));
						write(fdTmp, &n, sizeof(int));
					}
					for(int j=0 ; j<height_old-1 ; ++j){
						write(fdTmp, &width, sizeof(int));
						write(fdTmp, &j, sizeof(int));
						write(fdTmp, &wall, sizeof(int));
					}
				}
				/*agrandissement de la map en largeur*/
				else if(argI==4 && arg>width_old){
					if(x<width_old-1){
						write(fdTmp, &x, sizeof(int));
						write(fdTmp, &y, sizeof(int));
						write(fdTmp, &n, sizeof(int));
					}
					for(int i=width_old-1 ; i<arg ; ++i){
						write(fdTmp, &i, sizeof(int));
						write(fdTmp, &height, sizeof(int));
						write(fdTmp, &floor, sizeof(int));
					}
					for(int j=0 ; j<height_old-1 ; ++j){
						write(fdTmp, &width, sizeof(int));
						write(fdTmp, &j, sizeof(int));
						write(fdTmp, &wall, sizeof(int));
					}
				}
```

```
                else if(argI==5){
                        y += changeOfMap;
                        /*retrecissement de la map en hauteur*/
                        if(arg<height_old){
                                if(y<arg && y>=0){
                                        write(fdTmp, &x, sizeof(int));
                                        write(fdTmp, &y, sizeof(int));
                                        write(fdTmp, &n, sizeof(int));
                                }
                        }
                        /*agrandissement de la map en hauteur*/
                        else if(arg>height_old){
                                x_wall_right = width_old-1;;
                                write(fdTmp, &x, sizeof(int));
                                write(fdTmp, &y, sizeof(int));
                                write(fdTmp, &n, sizeof(int));

                                for(int j=0 ; j<(arg-height_old) ; ++j){
                                        write(fdTmp, &x_wall_left, sizeof(int));
                                        write(fdTmp, &j, sizeof(int));
                                        write(fdTmp, &wall, sizeof(int));
                                        write(fdTmp, &x_wall_right, sizeof(int));
                                        write(fdTmp, &j, sizeof(int));
                                        write(fdTmp, &wall, sizeof(int));
                                }
                        }
                }
        }
   }
   write(fdTmp, &x, sizeof(int));
   while(nbbytes!=0){
        nbbytes = read(mapFile, &x, sizeof(int));
        write(fdTmp, &x, nbbytes);
   }
}

void setObjects(int nbArg, char ** arguments, int mapFile, int fdTmp){
   if(((nbArg-3)%6))
        usage2("Nombre d'arguments de la liste invalide (6 demandes).");
   int x = 0, nbObj, length, frame, solidity, destructible, collectible, generator;
   char * name = NULL;
   char * objectArg = NULL;

   while(x!=-1){
        read(mapFile, &x, sizeof(int));
        write(fdTmp, &x, sizeof(int));
```

```
}

for(int i=3 ; i<nbArg ; i+=6){
        length = strlen(arguments[i]);
        name = realloc(name, length*sizeof(char));
        strcpy(name, arguments[i]);
        if(atoi(arguments[i+1])<1)
                usage2("Nombre de frames invalides.");
        else
                frame = atoi(arguments[i+1]);

        objectArg = realloc(objectArg, strlen(arguments[i+2])*sizeof(char));
        strcpy(objectArg, arguments[i+2]);
        if(!strcmp(objectArg,"solid"))
                solidity = MAP_OBJECT_SOLID;
        else if(!strcmp(objectArg,"semi-solid"))
                solidity = MAP_OBJECT_SEMI_SOLID;
        else if(!strcmp(objectArg,"air"))
                solidity = MAP_OBJECT_AIR;
        else
                usage2("Valeur de solidite : solid | semi-solid | air.");

        objectArg = realloc(objectArg, strlen(arguments[i+3])*sizeof(char));
        strcpy(objectArg, arguments[i+3]);
        if(!strcmp(objectArg,"destructible"))
                destructible = 1;
        else if(!strcmp(objectArg,"not-destructible"))
                destructible = 0;
        else
                usage2("Valeur de destructible : destructible | not-destructible.");

        objectArg = realloc(objectArg, strlen(arguments[i+4])*sizeof(char));
        strcpy(objectArg, arguments[i+4]);
        if(!strcmp(objectArg,"collectible"))
                collectible = 1;
        else if(!strcmp(objectArg,"not-collectible"))
                collectible = 0;
        else
                usage2("Valeur de collectible : collectible | not-collectible.");

        objectArg = realloc(objectArg, strlen(arguments[i+5])*sizeof(char));
        strcpy(objectArg, arguments[i+5]);
        if(!strcmp(objectArg,"generator"))
                generator = 1;
        else if(!strcmp(objectArg,"not-generator"))
                generator = 0;
```

```
		else
			usage2("Valeur de generator : generator | not-generator.");
		write(fdTmp, &length, sizeof(int));
		write(fdTmp, name, length*sizeof(char));
		write(fdTmp, &frame, sizeof(int));
		write(fdTmp, &solidity, sizeof(int));
		write(fdTmp, &destructible, sizeof(int));
		write(fdTmp, &collectible, sizeof(int));
		write(fdTmp, &generator, sizeof(int));
	}
	free(name);
	free(objectArg);

	lseek(fdTmp, 2*sizeof(int), SEEK_SET);
	nbObj = (nbArg-3)/6;
	write(fdTmp, &nbObj, sizeof(int));

	lseek(mapFile, 2*sizeof(int), SEEK_SET);
	read(mapFile, &x, sizeof(int));
	if(x>nbObj){
		nbObj--;
		while(x!=-1){
			read(mapFile, &x, sizeof(int));
			if(x!=-1){
				lseek(mapFile, sizeof(int), SEEK_CUR);
				read(mapFile, &x, sizeof(int));
				lseek(fdTmp, 2*sizeof(int), SEEK_CUR);
				if(x>nbObj)
					write(fdTmp, &nbObj, sizeof(int));
				else
					lseek(fdTmp, sizeof(int), SEEK_CUR);
			}
		}
	}
}

void pruneObjects(int mapFile, int fdTmp){
	int x = 0, nbObj, changeOfObj = 0, newNbObj = 0, new_n;
	char * name = NULL;
	lseek(mapFile, 2*sizeof(int), SEEK_SET);
	read(mapFile, &nbObj, sizeof(int));
	int * used = malloc(nbObj*sizeof(int));
	for(int i=0 ; i<nbObj ; i++)
		used[i] = 0;
	while(x!=-1){
		read(mapFile, &x, sizeof(int));
		if(x!=-1){
			lseek(mapFile, sizeof(int), SEEK_CUR);
			read(mapFile, &x, sizeof(int));
			if(used[x]==0)
				used[x] = 1;
		}
	}
	x = 0;
	for(int i=0 ; i<nbObj ; i++){
		if(used[i]==0)
			changeOfObj = 1;
		else
			newNbObj++;
	}
	if(changeOfObj==1){
		lseek(mapFile, 0, SEEK_SET);
		while(x!=-1){
			read(mapFile, &x, sizeof(int));
			write(fdTmp, &x, sizeof(int));
		}
		for(int i=0 ; i<nbObj ; i++){
			read(mapFile, &x, sizeof(int));
			name = realloc(name, (x+1)*sizeof(char));
			name[x] = '\0';
			read(mapFile, name, x*sizeof(char));
			if(used[i]==1){
				write(fdTmp, &x, sizeof(int));
				write(fdTmp, name, x*sizeof(char));
				for(int j=0 ; j<5 ; j++){
					read(mapFile, &x, sizeof(int));
					write(fdTmp, &x, sizeof(int));
				}
			}else
				lseek(mapFile, 5*sizeof(int), SEEK_CUR);
		}
		lseek(fdTmp, 2*sizeof(int), SEEK_SET);
		write(fdTmp, &newNbObj, sizeof(int));
		lseek(mapFile, 3*sizeof(int), SEEK_SET);
		while(x!=-1){
			read(mapFile, &x, sizeof(int));
			if(x!=-1){
				for(int i=0 ; i<2 ; i++)
					read(mapFile, &x, sizeof(int));
				new_n = x;
				for(int i=0 ; i<x ; i++){
					if(used[i]==0)
```

```
                                                new_n--;
                                        }
                                        lseek(fdTmp, 2*sizeof(int), SEEK_CUR);
                                        if(new_n!=x)
                                                write(fdTmp, &new_n, sizeof(int));
                                        else
                                                lseek(fdTmp, sizeof(int), SEEK_CUR);
                                }
                        }
        }else
                usage2("Nombre d'objets inchange.");
        free(name);
}

void usage(char * commande){
        fprintf(stderr, "%s fichier commande (entier/listeCaracteristiquesObjets)\n", commande);
        exit(EXIT_FAILURE);
}

void usage2(char * commande){
        fprintf(stderr, "%s\n", commande);
        execlp("/bin/sh", "sh", "-c", "rm tmp.map", NULL);
        exit(EXIT_FAILURE);
}

int main(int argc, char ** argv){
        if(argc<3)
                usage(argv[0]);
        char * commande = argv[2];

        int argI = setCommande(commande);

        if(argI>=0 && argI<=3){
                getInfos(argI, argv[1]);
        }else if(argI>=4){
                int mapFile = open(argv[1], O_RDONLY, 0666);
                int fdTmp = open("tmp.map", O_WRONLY|O_CREAT, 0666);
                if(mapFile==-1 || fdTmp==-1)
                        usage2("Le fichier ne s'est pas ouvert correctement.");
                if(argI!=7){
                        if(argc<4)
                                usage2("Quatrieme argument requis.");
                        if(argI==4 || argI==5)
                                setWidthHeight(argv[3], mapFile, fdTmp, argI);
                        else
                                setObjects(argc, argv, mapFile, fdTmp);
        }else
                pruneObjects(mapFile, fdTmp);
        close(mapFile);
        close(fdTmp);
        char * commandeSys = malloc(sizeof(char)*(strlen("mv tmp.map ")+strlen(argv[1])));
        strcat(commandeSys, "mv tmp.map ");
        strcat(commandeSys, argv[1]);
        execlp("/bin/sh" ,"sh", "-c", commandeSys, NULL);
        free(commandeSys);
    }else
        usage2("Commande invalide.");
    return EXIT_SUCCESS;
}
```