mapio.c :

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

#include <errno.h>

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

#define NB_CARACTERISTIQUES 5
#define MAX_OBJETS 10

#ifdef PADAWAN



int generer_flags(int solid, int collectible, int destructible, int generator){
    int flags = 0;
    //Ajout dans tous les case de la solidité
    flags |= solid;

    //Ajout des flags à 1 uniquement.
    if (collectible)
        flags |= MAP_OBJECT_COLLECTIBLE;
    if (destructible)
        flags |= MAP_OBJECT_DESTRUCTIBLE;
    if (generator)
        flags |= MAP_OBJECT_GENERATOR;

    return flags;
}


void map_new (unsigned width, unsigned height)
{
  map_allocate (16, 16);

  for (int x = 0; x < 16; x++){
    map_set (x, 16 - 1, 0); // Sol
  }
    map_set (5,5, 2);//un carré de marble
    map_set(6,5,3); //Carré de grass


  for (int y = 0; y < 16 - 1; y++) {
    map_set (0, y, 1); // Wall
    map_set (16 -1, y, 1); // Wall
  }

  map_object_begin (4);

  map_object_add ("images/ground.png", 1, MAP_OBJECT_SOLID); // 0

  map_object_add ("images/wall.png", 1, MAP_OBJECT_SOLID); // 1

  map_object_add ("images/marble.png", 1, MAP_OBJECT_SOLID | MAP_OBJECT_DESTRUCTIBLE); // 2

  map_object_add ("images/grass.png", 1, MAP_OBJECT_SEMI_SOLID); // 3

  map_object_end ();

}
```

```c
void map_save (char *filename)
{
        //Numéro d'erreur.
        errno = 0;

        //Ouverture du fichier.
        filename = "saved.map";
    int map = open(filename, O_WRONLY | O_CREAT, 0666);
        if (map == -1){
            fprintf(stderr, "Error creating the save file. %s", strerror(errno));
            exit(errno);
        }

    //Récupération de largeur, hauteur et nb objets suivi de leur écriture dans le fichier.
    int hauteur = (int)map_height();
    int largeur = (int)map_width();
    int nb_objets = (int)map_objects();

    write(map, &largeur, sizeof(int));
    write(map, &hauteur, sizeof(int));
        write(map, &nb_objets, sizeof(int));

    //Parcours de l'ensemble de la carte et récupération des objets aux coordonées
    int taille_carte = largeur * hauteur;
    int tab_carte[taille_carte];

        int index = 0;
    for (int y = 0; y < hauteur; y++){
            for (int x = 0; x < largeur; x++){
                tab_carte[index] = map_get (x,y);
                index++;
            }
        }
    write(map, &tab_carte, taille_carte * sizeof(int));


    //Ecriture pour chaque objet des informations le concernant
    int tab_cara[NB_CARACTERISTIQUES];
    for (int i = 0; i < nb_objets; i++){
            //Récupération de la taille du nom et du nom de l'objet.
            char *nom_objet = map_get_name(i);
            int taille_nom = strlen(nom_objet);

            write(map, &taille_nom, sizeof(int));
            write(map, nom_objet, taille_nom * sizeof(char));

            tab_cara[0] = map_get_frames(i);
            tab_cara[1] = map_get_solidity(i);
            tab_cara[2] = map_is_destructible(i);
            tab_cara[3] = map_is_collectible(i);
            tab_cara[4] = map_is_generator(i);
            printf(" test de merde: %d et %d", tab_cara[4], tab_cara[2]);

            write(map, &tab_cara, NB_CARACTERISTIQUES * sizeof(int));
        }
    close(map);
        fprintf(stdout, "Game saved successfully !\n");
}



void map_load (char *filename){

        //Numéro d'erreur
        errno = 0;
```

```
    int largeur = 0;
    int hauteur = 0;
    int nb_objets = 0;

    int objet = 0;

        //Ouverture du fichier
    int map = open(filename, O_RDONLY);
        if (map == -1){
            fprintf(stderr, "Error creating the save file. %s", strerror(errno));
            exit(errno);
        }

    //Lecture des informations de base : largeur, hauteur et nb_objets
    read(map, &largeur, sizeof(int));
    read(map, &hauteur, sizeof(int));
    read(map, &nb_objets, sizeof(int));

    map_allocate(largeur, hauteur);

    //Parcours de la matrice et placement des objets sur la carte
    for (int y = 0; y < hauteur; y++){
        for (int x = 0; x < largeur; x++){
            read(map, &objet, sizeof(int));
                        if (objet != -1)
                            map_set(x, y, objet);
                }
    }

    int taille_nom = 0;
    int nb_sprites = 0;
    int collectible = 0;
    int destructible = 0;
    int generator = 0;
    int solid = 0;

    map_object_begin(nb_objets);

        //récupération dans le fichier de chaque objet.
    for (int i = 0; i < nb_objets; i++){
        read(map, &taille_nom, sizeof(int));

                char nom_objet[taille_nom];
                char caractere_nom;

                //Récupération du nom de l'objet caractère par caractère.
                for (int i = 0; i< taille_nom; i++){
                  read(map, &caractere_nom, sizeof(char));
                  nom_objet[i] = caractere_nom;
                }
                //Lecture des caractéritiques dans le fichier
        read(map, &nb_sprites, sizeof(int));
        read(map, &solid, sizeof(int));
        read(map, &destructible, sizeof(int));
        read(map, &collectible, sizeof(int));
        read(map, &generator, sizeof(int));

                //récupération des flags uniquement à 1
                int flags = generer_flags(solid, collectible, destructible, generator);

                map_object_add (nom_objet, nb_sprites, flags);
    }
        map_object_end();
}
#endif
```

tempo.c :

```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"

pthread_mutex_t mux;

struct file {
    struct elem_file *premier;
};

struct elem_file{

    void *param_event;
    struct itimerval it;
    struct elem_file *suivant;
    struct elem_file *pre;
    unsigned long temps;
};


struct file *File;
// 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

// Routine de traitement
void routine(int sig){
   // fprintf (stderr, "sdl_push_event(%p) appelée au temps %ld\n", File->premier->param_event, get_time ());
}

void* demon (void* arg){

    struct itimerval it;
    getitimer(ITIMER_REAL, &it);

    //Strucure obligatoire car SIGALRM fait quitter le programme si il n'est pas traité par un handler.
    struct sigaction s;
    s.sa_flags = 0;
    sigemptyset(&s.sa_mask);
    s.sa_handler = routine;

    sigaction(SIGALRM, &s, NULL);
```

```c
    sigset_t mask;
    //Remplissage du masque de signaux pour bloquer tous les signaux.
    sigfillset(&mask);
    //Suppression de SIGALRM du masque pour qu'il puisse être délivré. Il est le seul signal non bloqué.
    sigdelset(&mask, SIGALRM);

    //Boucle pour traiter les events en continu.
    while(1){
        //Attente de SIGALRM
        sigsuspend(&mask);

        //traitement de l'event
        sdl_push_event(File->premier->param_event);
        pthread_mutex_lock(&mux);
        supprimer_premier_element_file();
        triFile();
        pthread_mutex_unlock(&mux);
        setitimer(ITIMER_REAL, &File->premier->it, NULL);
    }
}

// timer_init returns 1 if timers are fully implemented, 0 otherwise
int timer_init (void)
{
    pthread_mutex_init(&mux, NULL);

    //Creation d'un masque pour bloquer la transmission du SIGALRM dans le processus
    pthread_t thread;
    sigset_t block_mask;

    sigemptyset(&block_mask);
    sigaddset(&block_mask, SIGALRM);
    //Blocage du signal SIGALRM
    sigprocmask(SIG_BLOCK, &block_mask, NULL);

    pthread_create(&thread, NULL, demon, NULL);
    //Allocation de la Strucure de données
    File = malloc(sizeof(struct file));
    File->premier = NULL;
    return 1; // Implementation not ready
}

void triFile()
{
    struct itimerval it;
    getitimer(ITIMER_REAL, &it);
    //Calcul du delai restant
    long delai_restant_premier_elem = it.it_value.tv_sec*1000 + it.it_value.tv_usec/1000;

    struct elem_file *e;
    if (File->premier != NULL && File->premier->suivant != NULL)
    {
        e = File->premier->suivant;
        while (e->suivant != NULL)
        {
            //Récupération du temps de l'élément suivant
            long temps = e->suivant->it.it_value.tv_usec/1000 + e->suivant->it.it_value.tv_sec*1000;

            //Tri de la file via décalage du premier élément.
            if (delai_restant_premier_elem < temps - delai_restant_premier_elem)
            {
                printf("Dans if tri\n");
                if (e == File->premier->suivant)
                {
                    printf("premier\n");
                    e->suivant->pre = File->premier->suivant;
```

```
                    File->premier->pre = e;
                    File->premier->suivant = e->suivant;
                    File->premier = e;
                    File->premier->pre = NULL;
                }
                else
                {
                    printf("deuxieme\n");
                    e->suivant->pre = e->pre;
                    e->pre->suivant = e->suivant;
                    e->pre = e->suivant;
                    e->suivant = e->suivant->suivant;
                    e->pre->suivant = e;
                }
            }
            e = e->suivant;
        }
    }
}
void timer_set (Uint32 delay, void *param)
{
    long int sec = delay / 1000;
    // Recuperation du reste de la division par mille et conversion en micro-secondes
    long int micro = (delay % 1000) * 1000;

    struct itimerval it;

    struct timeval tv_interval;
    struct timeval tv_value;

     // les deux valeurs sont importantes. pour 2,2 secondes, sec = 2 (en secondes) et micro = 200000 (0,2s
converties en micro secondes)
    tv_value.tv_sec = sec;
    tv_value.tv_usec = micro;

    it.it_interval = tv_interval;
    it.it_value = tv_value;

    // Allocation de la Strucure et affectation des paramètres.
    struct elem_file *elem = malloc(sizeof(struct elem_file));

    elem->suivant = elem->pre = NULL;
    elem->param_event = param;
    elem->it = it;
    elem->temps = get_time();

    pthread_mutex_lock(&mux);
    ajouter_element_file(elem);
    pthread_mutex_unlock(&mux);

    setitimer(ITIMER_REAL, &File->premier->it, NULL);

    triFile();


}

void ajouter_element_file(struct elem_file *e)
{
    //Ajout de l'élément en premier si la file ne contient pas d'élément.
    if (File->premier == NULL)
    {
        File->premier = e;
        File->premier->suivant = File->premier->pre = NULL;
    }
    //Ajout de l'élément à la suite des autres.
```

```
    else{
        struct elem_file *el;
        el = File->premier;
        while (el->suivant != NULL){
            el = el->suivant;
        }
        e->pre = el;
        e->suivant = NULL;
        el->suivant = e;
    }

}

void supprimer_premier_element_file()
{
    //Déréférencement et free du premier élément.
    if (File->premier != NULL)
    {
        struct elem_file *e = File->premier;
        File->premier = File->premier->suivant;
        free(e->pre);
    }
}

void afficherFile()
{
    struct elem_file *e;
    e = File->premier;
    do{
        printf("%p: temps %d et %d adresse --> ", e, e->it.it_value.tv_sec, e->it.it_value.tv_usec);
        e = e->suivant;
    }while(e != NULL);
    printf("\n");
}
#endif
```

maputil.c :

```
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <errno.h>
#include <ctype.h>

#define MAP_OBJECT_AIR           0
#define MAP_OBJECT_SEMI_SOLID    1
#define MAP_OBJECT_SOLID         2

#define NB_CARACTERISTIQUES 9
#define MAP_OBJECT_NONE -1
#define NB_OBJECT_MAX 10

int lireEntierPositif (int fichierMap);
int getWidth(int fichierMap);
int getHeight(int fichierMap);
int getObjects(int fichierMap);
void printWidth(int fichierMap);
void printHeight(int fichierMap);
void printObjects(int fichierMap);
int supprimerListeObjets (int fichierMap);
int verificationArgumentsSetObjects (char *argv[], int n, int fichierMap);
void setWidthHeight(int longueur, char type, int fichierMap);
int setObjects(char **argv, int nombre_args, int fichierMap);
```

```c
void pruneObjects(int fichierMap);


enum ErrArgsSetObjects {ERRNOM, ERRTAILLE};
char* caracteristiques[] = {"solid", "semi-solid", "air", "collectible", "destructible", "generator",
"not-destructible", "not-collectible", "not-generator"};


int main (int argc, char** argv)
{
    if (argc > 2)   //doit avoir pour argument au moins le nom du fichier et un argument
    {
        int fichierMap = open(argv[1],O_RDWR, 0666);
        if (fichierMap != -1)
        {
            if (strcmp(argv[2], "--getinfos") == 0)
            {
                char* e = "largeur: ";
                write(1, e, strlen(e));
                printWidth(fichierMap);
                e = "hauteur: ";
                write(1, e, strlen(e));
                printHeight(fichierMap);
                e = "nombre d'objets: ";
                write(1, e, strlen(e));
                printObjects(fichierMap);
            }
            else if (strcmp(argv[2], "--getwidth") == 0)
                printWidth(fichierMap);
            else if (strcmp(argv[2], "--getheight") == 0)
                printHeight(fichierMap);
            else if (strcmp(argv[2], "--getobjects") == 0)
                printObjects(fichierMap);
            else if (strcmp(argv[2], "--setobjects") == 0)
            {
                int repVerificationArguments = verificationArgumentsSetObjects(&argv[3], argc-3, fichierMap);
                if (repVerificationArguments == ERRNOM)
                {
                    char *e = "erreur noms arguments\n";
                    write(2, e, strlen(e));
                }
                else if (repVerificationArguments == ERRTAILLE)
                {
                    char *e = "erreur taille des arguments\n";
                    write(2, e, strlen(e));
                }
                else
                {
                    printf("c'est bon \n");
                    setObjects(&argv[3], argc-3, fichierMap);
                }
            }

            else if (strcmp(argv[2], "--setwidth") ==  0)
              setWidthHeight(atoi(argv[3]), 'w', fichierMap);
            else if (strcmp(argv[2], "--setheight") ==  0)
              setWidthHeight(atoi(argv[3]), 'h', fichierMap);
            else if (strcmp(argv[2], "--pruneobjects") ==  0)
              pruneObjects(fichierMap);

            else
            {
                char *e = "erreur arguments\n";
                write(2, e, strlen(e));
            }
```

```
            close(fichierMap);
        }
        else
        {
            char *e = "erreur chargement fichier\n";
            write(2, e, strlen(e));
        }
    }
    return 0;
}
int verificationArgumentsSetObjects (char *argv[], int n, int fichierMap)
{
    if (n % 6 != 0 || n / 6 < getObjects(fichierMap))
        return ERRTAILLE;

    for (int i = 0; i < n; i++)
    {
        //Si c'est une image
        if (i % 6 == 0)
        {
            i++;
            if (atoi(argv[i]) == 0)
                return ERRNOM;
            i++;
        }
        //Vérification que les caractéristiques sont correctes.
        int index_cara = 0;
        while (index_cara < NB_CARACTERISTIQUES)
        {
            if (strcmp(argv[i], caracteristiques[index_cara]) == 0)
                break;
            index_cara++;
        }
        //Si argument non reconnu
        if (index_cara == NB_CARACTERISTIQUES)
            return ERRNOM;
    }
}
int setObjects(char **argv, int nombre_args, int fichierMap)
{
    int hauteurMap = getHeight(fichierMap);
    int largeurMap = getWidth(fichierMap);
    int saveMap[hauteurMap][largeurMap];
    lseek(fichierMap, 3*sizeof(int), SEEK_SET);
    //Sauvegarde de la matrice
    for (int i = 0; i < hauteurMap; i++)
          read(fichierMap, &saveMap[i], largeurMap*sizeof(int));

    //Suppression de tout sauf largeur et hauteur
    ftruncate(fichierMap, 2*sizeof(int));
    printf("%s\n", strerror(errno));
    lseek(fichierMap, 0, SEEK_END);

    int tmp = nombre_args/6;
    write(fichierMap, &tmp, sizeof(int));//on écrit la nouvelle valeur du nombre d'objet
    for (int i = 0; i < hauteurMap; i++)        //on récrit la matrice
          write(fichierMap, &saveMap[i], largeurMap*sizeof(int));

    //Ecriture de la nouvelle liste
    for (int i = 0; i < nombre_args; i++)
    {
        //Ecriture du nom des fichiers d'images
          if (i % 6 == 0)
          {
              int tailleChaine = strlen(argv[i]);
                  write(fichierMap, &tailleChaine, sizeof(int));
```

```
            write(fichierMap, argv[i], tailleChaine*sizeof(char));
                i ++;
                    int frame = atoi(argv[i]);
                    write(fichierMap, &frame, sizeof(int));
        }
        //Ecriture des caractéristiques.
        else
        {
            int index_cara = 0;
            while (index_cara < NB_CARACTERISTIQUES)
            {
                if (strcmp(argv[i], caracteristiques[index_cara]) == 0)
                {
                    //Solidité de l'objet
                    if (index_cara >= 0 && index_cara < 3)
                        write(fichierMap, &index_cara, sizeof(int));
                    else
                    {
                        int flags = 0;
                        if (index_cara >= 3 && index_cara < 6)
                             flags = 1;

                        write(fichierMap, &flags, sizeof(int));
                    }
                    break;
                }
                index_cara ++;
            }
        }
    }


}
int lireEntierPositif (int fichierMap)   //positif pour avoir un message d'erreur qui est -1
{
    int nb;
    int rep = read(fichierMap, &nb, sizeof(int));
    if (rep == - 1 || rep == 0)
        return -1;
    return nb;
}
int getWidth(int fichierMap)
{
    lseek(fichierMap, 0, SEEK_SET);
    return lireEntierPositif(fichierMap);
}
void printWidth(int fichierMap)
{
    int nb = getWidth(fichierMap);
    if (nb == -1)
    {
        char* e = "erreur chargement largeur\n";
        write(2, e, strlen(e));
    }
    else
        printf("%d\n", nb);
}
int getHeight(int fichierMap)
{
    lseek(fichierMap, sizeof(int), SEEK_SET);
    return lireEntierPositif(fichierMap);
}
void printHeight(int fichierMap)
{
    int nb = getHeight(fichierMap);
    if (nb == -1)
```

```
    {
        char* e = "erreur chargement hauteur\n";
        write(2, e, strlen(e));
    }
    else
        printf("%d\n", nb);
}
int getObjects(int fichierMap)
{
    lseek(fichierMap, 2*sizeof(int), SEEK_SET);
    return lireEntierPositif(fichierMap);
}
void printObjects(int fichierMap)
{

    int nb = getObjects(fichierMap);
    if (nb == -1)
    {
        char* e = "erreur chargement nombre d\'objet\n";
        write(2, e, strlen(e));
    }
    else
        printf("%d\n", nb);
}

void setWidthHeight(int longueur, char type, int fichierMap){

        int hauteurMap = getHeight(fichierMap); //on sauvegarde tous les paramètres du fichier map dans des
variables.
    int largeurMap = getWidth(fichierMap);
    int nb_objects = getObjects(fichierMap);

    int saveMap[hauteurMap][largeurMap];
    lseek(fichierMap, 3*sizeof(int),SEEK_SET);
    for (int i = 0; i < hauteurMap; i++){   //sauvegarde de la matrice.
      read(fichierMap, &saveMap[i], largeurMap*sizeof(int));
    }

    int taillechaine[nb_objects];
    char *chaine[nb_objects];
    int nb_caract=5;
    int caract[nb_objects][nb_caract];

    for(int i = 0; i<nb_objects; i++){ //sauvegarde de la taille de la chaine de caractère, du nom de l'objet et
de ses caractéristiques.
      read(fichierMap,&taillechaine[i],sizeof(int));
      chaine[i]=malloc(taillechaine[i]*sizeof(char));
      for(int j=0; j<taillechaine[i]; j++){
        read(fichierMap, &chaine[i][j], sizeof(char));
      }
      for(int k=0; k<nb_caract;k++){
        read(fichierMap, &caract[i][k], sizeof(int));
      }
    }

    int old_largeur=largeurMap;
    int old_hauteur=hauteurMap;

    if(type=='w')
      largeurMap = longueur;
    else
      hauteurMap = longueur;

    int newMap[hauteurMap][largeurMap]; //création nouvelle matrice avec la nouvelle valeur de largeur ou de
hauteur.
    memset(newMap, MAP_OBJECT_NONE, sizeof(newMap[hauteurMap][largeurMap])*hauteurMap*largeurMap);
```

```
    if(largeurMap>old_largeur)
      largeurMap=old_largeur;
    if(hauteurMap>old_hauteur)
      hauteurMap=old_hauteur;

    for(int i=0; i<hauteurMap; i++){
      for(int j=0; j<largeurMap; j++){
        newMap[i][j]=saveMap[i][j]; //remplit de la matrice avec les éléments de l'ancienne.
      }
    }

    if(type=='w')
      largeurMap = longueur;
    else
      hauteurMap = longueur;

    ftruncate(fichierMap, 0); //on efface le fichier entier.
        lseek(fichierMap, 0, SEEK_SET); //on se positionne au début du fichier.
        write(fichierMap, &largeurMap, sizeof(int)); //on réécrit tout.
        write(fichierMap, &hauteurMap, sizeof(int));
        write(fichierMap, &nb_objects, sizeof(int));

    for(int i=0; i<hauteurMap; i++){ //réécriture de la nouvelle matrice.
      for(int j=0; j<largeurMap; j++){
        write(fichierMap, &newMap[i][j], sizeof(int));
      }
    }

    for(int i=0; i<nb_objects; i++){
      write(fichierMap, &taillechaine[i], sizeof(int));
      for(int j=0; j<taillechaine[i]; j++){
        write(fichierMap, &chaine[i][j], sizeof(char));
      }
      for(int k=0; k<nb_caract; k++){
        write(fichierMap, &caract[i][k], sizeof(int));
      }
        }

    for(int i=0; i<nb_objects; i++){ //on free les mallocs du tableau contenant les noms des éléments.
      free(chaine[i]);
    }
}

void pruneObjects(int fichierMap){

        int hauteurMap = getHeight(fichierMap);
    int largeurMap = getWidth(fichierMap);
    int nb_objects = getObjects(fichierMap);

    int saveMap[hauteurMap][largeurMap];
    lseek(fichierMap, 3*sizeof(int),SEEK_SET);
    for (int i = 0; i < hauteurMap; i++){
      read(fichierMap, &saveMap[i], largeurMap*sizeof(int));
    }

    int taillechaine[nb_objects];
    char *chaine[nb_objects];
    int nb_caract=5;
    int caract[nb_objects][nb_caract];

    for(int i = 0; i<nb_objects; i++){
      read(fichierMap,&taillechaine[i],sizeof(int));
      chaine[i]=malloc(taillechaine[i]*sizeof(char));
      for(int j=0; j<taillechaine[i]; j++){
        read(fichierMap, &chaine[i][j], sizeof(char));
      }
```

```
      for(int k=0; k<nb_caract;k++){
        read(fichierMap, &caract[i][k], sizeof(int));
      }
    }

    int compteur[NB_OBJECT_MAX];
    memset(compteur, 0, sizeof(int)*NB_OBJECT_MAX);
    int id[10]={0,1,2,3,4,5,6,7,8,9};
    for(int i=0; i<hauteurMap; i++){ //pour chaque type d'éléments on compte son nombre d'occurences dans le
fichier de la carte.
      for(int j=0; j<largeurMap; j++){
        for(int k=0; k<9; k++){
          if(saveMap[i][j]==id[k]){
        compteur[k]++;
          }
        }
      }
    }

    ftruncate(fichierMap, 0);
    lseek(fichierMap, 0, SEEK_SET);
    write(fichierMap, &largeurMap, sizeof(int));
    write(fichierMap, &hauteurMap, sizeof(int));

    int true_nb_objects=0;
    for(int i=0; i<9; i++){ //on détermine le nombre réel d'élements présent sur la carte.
      if(compteur[i]!=0){
        true_nb_objects++;
      }
    }

    write(fichierMap, &true_nb_objects, sizeof(int));

    int curseur=0;
    int object_not_erase[true_nb_objects];
    for(int i=0; i<9; i++){ //on place dans un tableau, les id des élements présent sur la carte.
      if(compteur[i]!=0){
        object_not_erase[curseur]=i;
        curseur++;
      }
    }

    for(int i=0; i<hauteurMap; i++){
      write(fichierMap, &saveMap[i], largeurMap*sizeof(int));
    }

        for(int i=0; i<nb_objects; i++){
      if(i==object_not_erase[i]){ //si l'id de l'élément est bien présent alors on l'écrit dans le fichier.
         write(fichierMap, &taillechaine[i], sizeof(int));
         for(int j=0; j<taillechaine[i]; j++){
           write(fichierMap, &chaine[i][j], sizeof(char));
         }
         for(int k=0; k<nb_caract; k++){
           write(fichierMap, &caract[i][k], sizeof(int));
         }
        }
    }

    for(int i=0; i<nb_objects; i++){
      free(chaine[i]);
    }
}
```