Projet Jeu Snake

Vayaboury Michel, Su Sophie, Li Jinny
Licence Informatique

Année universitaire 2025-2026

Table des matières

1 Introduction **3**
1.1 Problématique 3
1.2 Contexte et cadre du projet 3
1.3 Objectif final du projet 3

2 Diagramme de Gantt **4**

3 Conception **4**
3.1 Résolution du problème 4
3.2 Choix techniques et structures de données 4
3.3 Priorités et hypothèses 5

4 Implémentation **5**
4.1 Langage et outils 5
4.2 Explication de chaque code 6

5 Compilation et exécution **31**

6 Difficultés rencontrées **31**
6.1 Arborescence du projet 31

7 Conclusion **32**
7.1 Limites et perspectives 32

1 Introduction

1.1 Problématique

Le projet consiste à concevoir et implémenter un jeu Snake en langage C. La problématique principale est de gérer le déplacement dynamique d'un serpent sur une grille de jeu, tout en assurant la cohérence de son corps, la génération de nourriture, la détection des collisions et l'interaction avec l'utilisateur en temps réel.

Ce type de jeu, bien que simple en apparence, pose plusieurs défis techniques, notamment la gestion de structures de données, la synchronisation entre la logique du jeu et l'affichage, ainsi que la gestion correcte des entrées utilisateur.

1.2 Contexte et cadre du projet

Ce projet s'inscrit dans le cadre de la licence informatique de deuxième années et a pour objectif de mettre en pratique les notions fondamentales de programmation vues en cours, en particulier en langage C. Il mobilise des compétences liées à la structuration du code, à la gestion de la mémoire, à l'utilisation de structures de données et à la conception d'un programme modulaire.

Le projet a été réalisé en groupe, ce qui a nécessité une organisation du travail en équipe, une répartition des tâches entre nous et une coordination entre les différents personnes. L'environnement de développement du projet utilisé est basé sur un système Linux (Ubuntu).

1.3 Objectif final du projet

L'objectif final du projet est de concevoir un jeu Snake fonctionnel en langage C, exécuté dans un terminal sous environnement Linux.

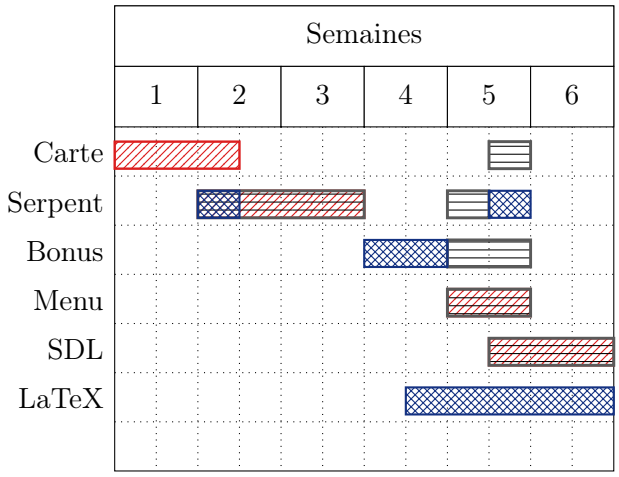
Le jeu doit permettre :

— le déplacement du serpent dans les quatre directions,
— la génération et la consommation de nourriture,
— l'augmentation progressive de la taille du serpent,
— la détection des collisions avec les murs ou le corps du serpent,
— la fin de la partie en cas de collision.

L'affichage du jeu est réalisé dans le terminal et aussi avec une interface graphique externe comme SDL.

2 Diagramme de Gantt

Légende : Michel Sophie Jinny



3 Conception

3.1 Résolution du problème

Pour résoudre le problème posé, le jeu a été conçu autour d'une boucle principale assurant le bon déroulement de la partie. Cette boucle permet de gérer successivement la lecture des entrées utilisateur, la mise à jour de la position du serpent, la vérification des collisions et l'actualisation de l'affichage.

La logique du jeu repose sur une représentation du serpent comme une suite de segments positionnés sur une grille. À chaque déplacement, la tête avance dans une direction donnée et le reste du corps suit ce mouvement.

3.2 Choix techniques et structures de données

Le jeu a été implémenté en langage C en utilisant un affichage en mode terminal. Ce choix permet de se concentrer sur les mécanismes internes du jeu sans dépendre d'une bibliothèque graphique externe.

Le serpent est représenté par une structure de données composée de plusieurs nœuds. Chaque nœud correspond à un segment du corps du serpent et contient ses coordonnées sur la carte de jeu. Cette représentation permet une gestion dynamique de la taille du serpent lors de la consommation de nourriture.

Le projet est organisé de manière modulaire, avec plusieurs fichiers source et fichiers d'en-tête, chacun étant responsable d'une fonctionnalité précise (gestion du serpent, de la carte, du menu, de la génération des éléments).

3.3 Priorités et hypothèses

Les priorités du projet ont été définies afin d'assurer un fonctionnement correct du jeu avant toute amélioration visuelle ou fonctionnelle. La priorité principale a été donnée à la stabilité du jeu, à la gestion correcte des déplacements et à la détection des collisions.

Certaines hypothèses ont été posées pour simplifier la conception, notamment :

— une taille de grille fixe(la carte),
— le déplacement,
— l'absence de niveaux de difficulté dans la version initiale.

4 Implémentation

4.1 Langage et outils

Le jeu a été développé en langage C sous Ubuntu. L'affichage est réalisé en mode terminal

4.2 Explication de chaque code

1 – Fichier `liste.h`

```
1
2   #ifndef LISTE_H_INCLUDED
3   #define LISTE_H_INCLUDED
4
5   #include "noeud.h" // Permet d'utiliser le type Noeud, Elt et les
        fonctions déclarées dans noeud.h
6
7
8   // Structure représentant une liste chaînée avec deux pointeurs "
        sentinelles" :
9   typedef struct _liste{
10    Noeud *sentNext; // - sentNext : pointe vers le début (avant) de la
          liste
11    Noeud *sentBack; // - sentBack : pointe vers la fin (arrière) de la
          liste
12  } Liste;
13
14  Liste* create_Liste(); /* Créer une liste [0, 0] */
15
16  void freeListe(Liste* l); /* Libère toute la mémoire associée à la
        liste (noeuds + structure Liste). */
17
18  Noeud* ithNoeud(Liste* l, int i); /* Renvoie un pointeur vers le noeud
        situé à la position i dans la liste. */
19
20  Elt ithElt(Liste* l, int i); /* Renvoie l'élément (contenu) stocké
        dans le noeud situé à la position i. */
21
22  void insert(Liste *l, int i, Elt e); /* Insère l'élément e à
        laposition i dans la liste */
23
24  void delete(Liste *l, int i); /* Supprime le noeud situé à la position
        i */
25
26  #endif
```

2 – Fichier `map.h`

```
1
2   #ifndef MAP_H
3   #define MAP_H
4   #define LIRE 1024
5
6   #include <stdio.h>
7   #include <stdlib.h>
8   #include <string.h>
9
10  // Structure représentant une carte (la grille):
11  typedef struct _map{
12    char **data; // - data: tableau de lignes (chaînes de caractères),
          chaque ligne représente une rangée de la carte
13    int width; // - width: largeur de la carte (nombre de colonnes)
14    int height; // - height: hauteur de la carte (nombre de lignes)
15  } Map;
16
17  Map *load_map(const char *filename); /* Lire le fichier de carte.txt
       et remplie la grille de la carte et initialise width et height */
18
19  void free_map(Map *map); /* Libère toute la mémoire associée à la
       carte (lignes + structure Map). */
20
21  void print_map(Map *map); /* affichier la carte dans la console */
22
23  #endif
```

3 – Fichier menu.h

```c

#ifndef MENU_H
#define MENU_H

#include "spawn.h"

/* Codes menu principal */
#define MENU_LEAVE 0 // - MENU_LEAVE: quitter le programme
#define MENU_NEW 1 // - MENU_NEW: lancer une nouvelle partie
#define MENU_LOAD 2 // - MENU_LOAD: charger une partie qui est sauvegardée

void purge(void); /* Vider un scanf de '\n', pour éviter des lectures incorrectes.*/

int menu_principal(char *nom_fichier); /* Affiche le menu principal du jeu et met à jour le nom du fichier de carte si nécessaire */

char menu_direc_leave(int position); /* Affiche le menu pendant le jeu */
// position : option actuellement sélectionnée

void save(Snake *snake, Bonus *bonus, char *nom_map); /* Sauvegarder l'état actuel de la partie dans le fichier "save.txt":
snake: - position et état du snake
bonus: - bonus présents
nom_map: -nom de la carte utilisée */

int load_save(Snake **snake, Bonus **bonus, Map **map, char* nom_map);
/* Charge une partie depuis le fichier "save.txt" (initialise: snake, bonus et la carte + met à jour le nom de la carte)*/

#endif
```

4 – Fichier noeud.h

```
1
2   #ifndef NOEUD_H_INCLUDED
3   #define NOEUD_H_INCLUDED
4
5   #include <stdio.h>
6   #include <stdlib.h>
7
8   // Structure de Elt qui contient: un caractère(c) et un entier(i).
9   typedef struct _elt{
10    char c;
11    int i;
12  }Elt;
13
14  typedef struct _noeud Noeud; // Renommé _noeud à Noeud
15
16  // Structure de noeud qui contient :
17  struct _noeud{
18    Elt cont; // - cont : donnée stockée dans le noeud
19    int sent; // - sent : indique si le noeud est une sentinelle
20    Noeud *suiv; // - suiv : pointeur vers le noeud suivant
21  };
22
23  Noeud* create_Noeud(Elt e, Noeud* n); /* Créer un noeud */
24
25  void freeNoeud(Noeud* n);/* Libèrer la mémoire d'un noeud*/
26
27  Noeud* create_Sent();/* Créer un noued sentinelle */
28
29  Noeud* next(Noeud* n);/* Renvoie le pointeur vers le noeud suivant */
30
31  Elt content(Noeud *n);/* Renvoie l'élément (la donnée) dans le noeud.
        */
32
33  void changeNext(Noeud * n, Noeud * p);/* Relie deux noeuds ensemble */
34  // Le noeud n pointera désormais vers p comme noeud suivant.
35
36  #endif
```

5 – Fichier snake.h

```
1
2   #ifndef SNAKE_H
3   #define SNAKE_H
4
5   #include <stdbool.h>
6   #include "liste.h" // Permet d'utiliser le type Liste et les fonctions
        associées
7   #include "map.h" // Permet d'utiliser le type Map et les fonctions
        associées
8
9   #define NB_VOWEL 6 //nombre de voyelle
10
11  //structure de snake: taille, corps, les coordonnées x et y du corps,
        et le score
12  typedef struct _snake{
```

```
13      int size; // - size  : taille du serpent
14      Liste *body; // - body  : liste représentant le corps
15      Liste *x; // Cordonnées x
16      Liste *y; // Cordonnées y
17      int score; // - score: score actuel du joueur
18    } Snake;
19
20    // Structure de bonus:
21    typedef struct _bonus{
22      char letter; // Les lettres
23      int x; // Cordonnés x de la lettre
24      int y; // Cordonnés y de la lettre
25      int footsteps; // Le nombre de pas du serpent
26    } Bonus;
27
28
29    Snake* create_Snake(); /*Créer le serpent (vide + réserver des places)
          */
30
31    void freeSnake(Snake *snake); /* Libèrer la mémoire du serpent */
32
33    void init_snake(Snake *snake); /* Initialiser le serpent */
34
35    void write_snake(Map *map, Snake *snake); /* Ecrire le serpent dans la
          carte */
36
37    void delete_queue(Snake *snake); /* Supprimer que les coordonnées de
          la queue */
38
39    bool is_Bonus(Bonus *bonus, int x, int y); /* Booléen pour dire si la
          case du coordonnée x et y de la carte est un bonus ou non */
40
41    bool belongs_vowel(Bonus *bonus); /* Booléen pour dire si le bonus est
          une voyelle ou pas */
42
43    void eat_insert(Snake *snake, Bonus* bonus, Map *map); /* Manger le
          bonus */
44
45    bool belongs_snake(Snake* snake, int x, intl y); /* Booléen pour dire
          si la case du coordonnée x et y de la carte est le corps du serpent
          ou non */
46
47    int case_next(Map *map, int x, int y, Bonus *bonus, Snake *snake); /*
          Voir la case du coordonnée x et y*/
48
49    int mouvement_snake(Snake *snake, char button, Map *map, Bonus *bonus)
          ; /* Définir le mouvement du serpent selon le touche entrée(boutton
          )*/
50
51
52    #endif
```

6 – Fichier spawn.h

```c

#ifndef SPAWN_H
#define SPAWN_H


#include "snake.h"


Bonus* create_Bonus(Map *map, Snake * snake); /* Créer le bonus alé
    atoire */

Bonus * init_Bonus(Snake * snake, Map *map); /* Initialise le bonus */

void freeBonus(Bonus *bonus); /* Libèrer la mémoire du bonus */

void write_bonus(Map *map, Bonus * bonus); /* Ecrire le bonus sur la
    carte */

Bonus* delete_bonus(Map *map, Bonus *bonus, Snake * snake); /*
    Supprimer le bonus */


/*Supplément pour le jeu snake en SDL*/
bool belongs_lettrer(char variable);/*chercher si variable appartient
    à l'alphabet*/

#endif
```

7 – Fichier map.c

```c

#include "map.h"

// Lire le fichier de "carte.txt" et remplie la matrice de la carte et
     ses tailles
Map *load_map(const char *filename) {
  char fullpath[50]; // Tableau pour mettre le chemin des "fichiers .
     txt"
  snprintf(fullpath, sizeof(fullpath),"sauvegarde/%s", filename); //
     Mettre dans le tableau le dossier, et le nom du fichier donnée
  FILE *file = fopen(fullpath, "r"); // Ouverir le fichier en seule
     lecture

  // Teste d'erreur d'ouverture
  if (file == NULL) {
    perror("Erreur d'ouverture de carte.txt");
  }
  // Réserver(alloue) une place de taille de carte
  Map *map = malloc(sizeof(Map));
  // Initialisation des données de la carte
  map->data = NULL;
  map->width = 0;
  map->height = 0;

  // Tableau pour mettre les caractères lue
```

```
22      char buffer[1024];
23
24      // tant qu'il peut lire, il lit la ligne entier
25      while (fgets(buffer, sizeof(buffer), file)) {
26        int len = strlen(buffer); // La taille des caractères lue d'une
              ligne, donc c'est la longueur
27
28        // Si le dernier caractère lue est à la ligne, change en \0 (fin
              sans saute de ligne)
29        if (buffer[len - 1] == '\n')
30        buffer[len - 1] = '\0';
31
32        // Mettre à jour la hauteur de la carte
33        if (map->width == 0)
34        map->width = strlen(buffer);
35
36        // Prendre une place déjà créer pour une matrice
37        map->data = realloc(map->data, sizeof(char *) * (map->height + 1))
              ;
38        // Réserver de la mémoire de la taille de la matrice
39        map->data[map->height] = malloc(map->width + 1);
40        // Stocker les caractères de la carte
41        strcpy(map->data[map->height], buffer);
42
43        map->height++; // Augmenter la hauteur si on peut toujour lire
44      }
45
46      // Fermer le fichier ouvert
47      fclose(file);
48      return map;
49    }
50
51    // Libérer la mémoire de la carte
52    void free_map(Map *map) {
53      // Parcourir selon la hauteur
54      for (int i = 0; i < map->height; i++) {
55        free(map->data[i]); // Libérer le tableau de hauteur i
56      }
57      free(map->data); // Libérer le tableau de hauteur
58      free(map); // Libérer le pointeur de la carte
59    }
60
61    // Affichier la carte
62    void print_map(Map *map) {
63      // Parcourir le tableau par hauteur
64      for (int i = 0; i < map->height; i++) {
65        printf("%s\n", map->data[i]); // Afficher le tableau de hauteur i
66      }
67    }
```

8 – Fichier `menu.c`

```
1
2   #include "menu.h"
3
4   /* Vider le scanf tantque il y a '\n' ou EOF avec getchar() */
```

```c
void purge(void){
  int c = 0;
  while ((c=getchar())!='\n' && c!= EOF);
}

/* Afficher le Menu principal */
int menu_principal(char *name_fichier) {
  int choix, choix2;
  printf("\n");
  printf(" ==================== SNAKE ==================== \n");
  printf("1. Nouvelle partie\n");
  printf("2. Choisir la carte\n");
  printf("3. Partie sauvegardée\n");
  printf("4. Quitter\n");
  printf("Choix : ");
  scanf("%d", &choix);
  purge();

  // Pour dire lancer le jeu: MENU_NEW
  if (choix == 1) {
    return MENU_NEW;
  }
  // Pour changer le nom fichier du carte, ensuite dire lancer le jeu: MENU_NEW
  if (choix == 2) {
    printf("\n");
    printf(" ------------- Taille de la carte ------------- \n");
    printf("1. carte%dx%d\n",16,9);
    printf("2. carte%dx%d\n",27,14);
    printf("3. carte%dx%d\n",49,18);
    scanf("%d", &choix2);
    purge();

    // Changer le nom du fichier selon le choix du jouer
    if(choix2 == 1){
      strcpy(name_fichier, "carte.txt");
    }
    if(choix2 == 2){
      strcpy(name_fichier, "carte2.txt");
    }
    if(choix2 == 3){
      strcpy(name_fichier, "carte3.txt");
    }
    return MENU_NEW;
  }
  // Pour dire lancer le sauvegarde: MENU_LOAD
  if (choix == 3) {
    return MENU_LOAD;
  }
  return MENU_LEAVE;
}

/* Menu pendant le jeu */
char menu_direc_leave(int position) {
  char c;
  // Si c'est 1, il demande la direction, sinon il affiche le menu de quitte
  if(position == 1){
```

```
61        printf("Direction (k o l m), quitter(q): ");
62      }else{
63        printf("\n");
64        printf(" ******** Vous voulez vraiment quitté? ********** \n");
65        printf("x. sauvegarder\n");
66        printf("r. recommencer\n");
67        printf("c. Choisir la carte\n");
68        printf("q. Quitter\n");
69      }
70      scanf(" %c", &c);
71      purge();
72      return c;
73    }
74
75    /* Sauvegarder la partie dans "save.txt" */
76    void save(Snake *snake, Bonus *bonus, char *name_map) {
77      // Ouvre le fichier de sauvegarde et lis les données du fichier en é
          criture seule
78      FILE *f = fopen("sauvegarde/save.txt", "w");
79      if (!f) { printf("Erreur ouverture save.txt\n"); return; } // teste
          l'erreur d'ouverture
80
81      // Ecrire les données dans le fichier "save.txt" (sauvegarde dans
          une ordre précise pour la suite de relire)
82
83      fprintf(f, "%s\n", name_map); // Nom du fichier de niveau
84      fprintf(f, "%d\n", snake->score); // Le score
85      fprintf(f, "%d\n", snake->size);// La taille du serpent
86
87      // Sauvegarde des segments
88      // Serpent
89      Noeud *nb = snake->body->sentNext->suiv;//
90      while (nb->cont.c != 0) {
91        fprintf(f, " %c ", nb->cont.c);
92        nb = nb->suiv;
93      }
94      fprintf(f, "\n"); // Pour la beauté de lecture dans save.txt
95
96      // Coordonnée x
97      Noeud *nx = snake->x->sentNext->suiv;
98      while (nx->cont.i != 0) {
99        fprintf(f, "%d ", nx->cont.i);
100       nx = nx->suiv;
101     }
102     fprintf(f, "\n");
103
104     // Coordonnée y
105     Noeud *ny = snake->y->sentNext->suiv;
106     while (ny->cont.i != 0) {
107       fprintf(f, "%d ", ny->cont.i);
108       ny = ny->suiv;
109     }
110     fprintf(f, "\n");
111
112     // Sauvegarde du bonus
113     fprintf(f, "%d %d %d\n", bonus->x, bonus->y, bonus->footsteps);
114
115     fclose(f);// Fermer le fichier save
```

```
116     printf(" ------------Sauvegarde effectuée--------------- \n");
117   }
118
119   /* Charger une partie depuis save.txt */
120   int load_save(Snake **snake, Bonus **bonus, Map **map, char *name_map)
          {
121     //ouvert le fichier save pour lire les sauvegardes
122     FILE *f = fopen("sauvegarde/save.txt", "r");
123     //teste erreur d'ouverture
124     if (!f) return 0;
125
126     // Récupérer les données(il lit dans une ordre précise)
127     // Renommer le nom du fichier par la carte de sauvegarde
128     fscanf(f, "%s", name_map);
129     // Charger la carte
130     *map = load_map(name_map);
131
132     // Prendre le score et taille
133     int score, size;
134     fscanf(f, "%d", &score);
135     fscanf(f, "%d", &size);
136
137     // Créer le serpent et charger les infordmations
138     *snake = create_Snake();
139     (*snake)->score = score;
140     (*snake)->size = size;
141
142     // Ajouter au fur et à mesure les lettres du corps et ces coordonné
          es
143     // Prendre puis inserte
144     Elt b, x, y;
145     for (int i = 1; i <= size; i++) {
146       fscanf(f, " %c ", &b.c); // Un espace précise devant pour qu'il
            ignore les espaces à la suite
147       insert((*snake)->body, i, b);
148     }
149
150     for (int k = 1; k <= size; k++) {
151       fscanf(f, "%d", &x.i);
152       insert((*snake)->x ,k, x);
153     }
154
155     for (int j = 1; j <= size; j++) {
156       fscanf(f, "%d", &y.i);
157       insert((*snake)->y, j, y);
158     }
159
160     // Récupérer les informations du bonus
161     int bx, by, footsteps;
162     fscanf(f, "%d %d %d", &bx, &by, }&footsteps);
163     // Créer et charger le bonus
164     *bonus = create_Bonus(*map, *snake);
165     (*bonus)->x = bx;
166     (*bonus)->y = by;
167     (*bonus)->footsteps = footsteps;
168
169     fclose(f); // Fermeture du fichier save
170     return 1;
```

```
171 }
```

9 – Fichier noeud.c

```
1
2  #include "noeud.h"
3
4  /* Créer un noeud e */
5  Noeud* create_Noeud(Elt e, Noeud* s){
6    // Réserver un place de type Noeud
7    Noeud *n = malloc (sizeof(Noeud));
8    // Remplir les données du noeud
9    n->cont = e;
10   n->sent = 0;
11   n->suiv = s;
12   return n;
13 }
14
15 /* Libèrer la mémoire de noeud */
16 void freeNoeud(Noeud* n){
17   free(n);
18 }
19
20 /* Créer le sentinelle */
21 Noeud* create_Sent(){
22   //definir l'entier 0
23   Elt e = {.i = 0};
24   //créer le noeud
25   Noeud *n = create_Noeud(e, NULL);
26   //noeud -> sentinelle
27   n->sent = 1;
28   return n;
29 }
30
31 /* Chercher le noeud suivant */
32 Noeud* next(Noeud* n){
33   return n->suiv;
34 }
35
36 /* Le contenu du noeud */
37 Elt content(Noeud *n){
38   return n->cont;
39 }
40
41 /* Relier 2 noeuds ensemble, l'un après l'autre */
42 void changeNext(Noeud * n, Noeud * p){
43   n->suiv = p;
44 }
```

10 – Fichier snake.c

```c

#include "snake.h"

/* Tableau de voyelle */
char vowel[] = {'a', 'e', 'i', 'o', 'u', 'y'};


/* Créer le serpent tout vide (réserver des places) */
Snake* create_Snake(){
  Snake *snake = malloc(sizeof(Snake));

  snake->size = 0;
  snake->body = create_Liste();
  snake->x = create_Liste();
  snake->y = create_Liste();
  snake->score = 0;
  }
  return snake;
}


/*Libèrer la mémoire du serpent, d'abord ses pointeurs des 'Liste',
puis le pointeur de 'Snake'*/
void freeSnake(Snake *snake){
  freeListe(snake->body);
  freeListe(snake->x);
  freeListe(snake->y);
  free(snake);
}


/* Initialiser le serpent à une taille 1, d'un corps 'z', un score 0
et d'un coordonnée(1, 1). Il sera toujours sur le coordonnée(1, 1) */
void init_snake(Snake *snake){
  Elt v = {'z', 1};

  snake->size = 1;}
  snake->score = 0;
  insert(snake->body, 0, v);
  insert(snake->x, 0, v);
  insert(snake->y, 0, v);

}

/* Ecrire le serpent dans la carte */
void write_snake(Map *map, Snake *snake){
  // Teste des entrées d'une erreur: NULL (vide)
    if (map == NULL || map->data == NULL || snake == NULL) return;
  // Des noeuds pour faire le parcours de la liste
    Noeud *actuel_body = snake->body->sentNext->suiv;
    Noeud *actuel_x = snake->x->sentNext->suiv;
    Noeud *actuel_y = snake->y->sentNext->suiv;

  // Parcourir une liste (x, y et body ont tous la même taille)
    while (actuel_body->cont.c != 0){
    // Ecrire le serpent
      map->data[actuel_y->cont.i][actuel_x->cont.i] = actuel_body->
```

```
            cont.c;
58          // Mettre à jour(le suivant)
59            actuel_x = actuel_x->suiv;
60            actuel_y = actuel_y->suiv;
61            actuel_body = actuel_body->suiv;
62        }
63    }
64
65
66    /* Supprimer le coordonnée de la queue quand le serpend se déplace
          sans manger */
67    void delete_queue(Snake *snake){
68      delete(snake->x, snake->size+1);
69      delete(snake->y, snake->size+1);
70    }
71
72    /* Booléen pour dire si la case du coordonnée x et y de la carte est
          un bonus ou pas */
73    bool is_Bonus(Bonus *bonus, int x, int y){
74      if(bonus->x == x && bonus->y == y){
75        return true;
76      }else{
77        return false;
78      }
79    }
80
81    /* Booléen pour dire si le bonus est une voyelle ou pas */
82    bool belongs_vowel(Bonus *bonus){
83      int i = 0, v = 0;
84
85      /* Parcourir le tableau de voyelle*/
86      while(i < NB_VOWEL && v == 0){
87        /* Si c'est une voyelle, on sorte directement (v = 1), sinon on
              continue */
88        if(bonus->letter == vowel[i]){
89          v = 1;
90        }else{
91          i++;
92        }
93      }
94      /* Si i = 6 donc faux (car les indice du tableau de voyelles sont de
            0 à 5), donc si i est plus petit que 6, c'est qu'il à trouver:
           vraie */
95
96      return i < 6;
97    }
98
99    /* Manger le bonus */
100   void eat_insert(Snake *snake, Bonus* bonus, Map *map){
101     // Mettre en Elt pour le fonction insert
102     Elt b = {.c = bonus->letter};
103     Elt x = {.i = bonus->x};
104     Elt y = {.i = bonus->y};
105
106     // Augmenté la taille du serpent
107     snake->size += 1;
108     // Si c'est une voyelle, ajouté 10 points, sinon 5 points)
109     if(belongs_vowel(bonus)){}
```

```
110       snake->score += 10;
111     }else{
112       snake->score += 5;
113     }
114
115     // Insérer dans la tête le bonus mangé
116     insert(snake->body, 0, b);
117     insert(snake->x, 0, x);
118     insert(snake->y, 0, y);
119   }
120
121   /* Booléen pour dire si la case du coordonnée x et y de la carte est
        le corps du serpent ou pas */
122   bool belongs_snake(Snake* snake, int x, int y){
123     // Des noeuds pour faire le parcours de la liste
124     // ->suiv, pour ignorer le 0 du début; liste vide = [0, 0]
125     Noeud *actuel_x = snake->x->sentNext->suiv;
126     Noeud *actuel_y = snake->y->sentNext->suiv;
127
128     // Parcourir une liste (x, y ont tous la même taille)
129     // Actuel_body->cont.c != 0, pour ignorer le 0 de la fin
130     while (actuel_y->cont.c != 0){
131       // Si la case du coordonnée x et y est égale à ieme corps du
            serpent, on sorte (return)
132       if(actuel_x->cont.i == x && actuel_y->cont.i == y){
133         return true;
134       }
135       // Mettre à jour(le suivant)
136       actuel_x = actuel_x->suiv;
137       actuel_y = actuel_y->suiv;
138     }
139     // Sinon c'est faux
140     return false;
141   }
142
143   /* Voir la case du coordonnée x et y est-il un bonus, une partie du
        corps du serpent, un mur ou vide
144   si il retourne 2, il mange: 0, il perdre: 1, vide */
145   int case_next(Map *map, int x, int y, Bonus *bonus, Snake *snake){
146     //la case du coordonnée x et y
147     char valeur = map->data[y][x];
148
149     // Si c'est un bonus
150     if(is_Bonus(bonus, x, y)){
151       eat_insert(snake, bonus, map);
152       return 2;
153       // Si c'est une partie du corps du serpent ou un mur
154     }else if(belongs_snake(snake, x, y) || valeur == '#'){
155       return 0;
156     }else{
157       return 1;
158     }
159   }
160
161   /* Deplacer/manger/perdre le serpent selon la touche d'entrée(boutton)
        */
162   int mouvement_snake(Snake *snake, char button, Map *map, Bonus *bonus)
        {
```

```
163     // Resultat pour definir si le jeu continue(0) ou le jeu est perdu
            (1)
164     int resultat = 0, mouvement;
165     // Le coordonnée x et y de la tête du serpent
166     Elt x_elt = ithElt(snake->x, 1);
167     Elt y_elt = ithElt(snake->y, 1);
168
169     // Selon les touches
170     switch(button){
171       // Avancer
172       case 'o':
173       // Regarder la case en haut (1 pour avancer, 2 pour manger, 1 pour
              perdre)
174       mouvement = case_next(map, x_elt.i, y_elt.i-1, bonus, snake);
175
176       if(mouvement == 1){
177         /* Inserte x et y la case suivant au serpent pour qu'il déplace,
178         et supprime le coordonée de la queue, de même pour les autres
              touches */
179         y_elt.i--;
180         insert(snake->x, 0, x_elt);
181         insert(snake->y, 0, y_elt);
182         delete_queue(snake);
183       }else if(mouvement == 2){
184         bonus->footsteps = 10; // Pour que le jeu crée un nouveau bonus
              après l'avoir mangé
185       }else{
186         resultat = 1;
187       }
188       break; // fini définitivement
189
190       // à gauche
191       case 'k':
192       // Regarder la case à gauche (1 pour avancer, 2 pour manger, 1
              pour perdre)
193       mouvement = case_next(map, x_elt.i-1, y_elt.i, bonus, snake);
194
195       if(mouvement == 1){
196         x_elt.i--;
197         insert(snake->x, 0, x_elt);
198         insert(snake->y, 0, y_elt);
199         delete_queue(snake);
200       }else if(mouvement == 2){
201         bonus->footsteps = 10;
202       }else{
203         resultat = 1;
204       }
205       break;
206
207       // à droite
208       case 'm':
209       // Regarder la case à droite (1 pour avancer, 2 pour manger, 1
              pour perdre)
210       mouvement = case_next(map, x_elt.i+1, y_elt.i, bonus, snake);
211
212       if(mouvement == 1){
213         x_elt.i++;
214         insert(snake->x, 0, x_elt);
```

```
215         insert(snake->y, 0,y_elt);
216         delete_queue(snake);
217       }else if(mouvement == 2){
218         bonus->footsteps = 10;
219       }else{
220         resultat = 1;
221       }
222       break;
223
224       // Reculer
225       case 'l':
226       // Regarder la case en bas (1 pour avancer, 2 pour manger, 1 pour
              perdre)
227       mouvement = case_next(map, x_elt.i, y_elt.i+1, bonus, snake);
228
229       if(mouvement == 1){
230         y_elt.i++;
231         insert(snake->x, 0, x_elt);
232         insert(snake->y, 0, y_elt);
233         delete_queue(snake);
234       }else if(mouvement == 2){
235         bonus->footsteps = 10;
236       }else{
237         resultat = 1;
238       }
239       break;
240     }
241     return resultat;
242   }
```

11 – Fichier spawn.c

```
1
2   #include "snake.h" // Pour utilisé les fonctions de serpent
3   #include "spawn.h" // et bonus et autre include dans ces fichiers
4
5
6   // Tableau d'alphabet
7   char tab[] = {'a', 'b', 'c', 'd', 'e',
8     'f', 'g', 'h', 'i', 'j',
9     'k', 'l', 'm', 'n', 'o',
10    'p', 'q', 'r', 's', 't',\newpage
11    'u', 'v', 'w', 'x', 'y',
12    'z'};
13
14  /* Créer le bonus aléatoire */
15  Bonus* create_Bonus(Map *map, Snake * snake){
16    // Crée un espace
17    Bonus *bonus = malloc(sizeof(Bonus));
18    bonus->letter = tab[rand() % 26]; // Aléatoire entre 0 et 25
19    bonus->x = 1 + rand() % (map->width-2); // Aléatoire entre 1 et
          longeur-2
20    bonus->y = 1 + rand() % (map->height-2); // Aléatoire entre 1 et
          largeur-2
21    // (-2 pour pas dépacer la map et pas mette sur le bord de la map)
22
```

```
23      bonus->footsteps = 0; // Initialisation de pas
24      return bonus;
25    }
26
27    /* Initialiser le bonus, pour qu'il n'est pas sur le mur, ni sur le
          serpent */
28    Bonus * init_Bonus(Snake * snake, Map *map){
29      Bonus * bonus = create_Bonus(map, snake);
30      int x = bonus->x, y = bonus->y;
31      // tantque la case du bonus appartient au serpent ou au mur, recréer
            une autre
32      while(belongs_snake(snake, x, y) || map->data[y][x] == '#') {
33        freeBonus(bonus);
34        bonus = create_Bonus(map, snake);
35        x = bonus->x, y = bonus->y; // Met à jour x et y du bonus
36      }
37      return bonus;
38    }
39
40    /* Libèrer la mémoire de bonus */
41    void freeBonus(Bonus *bonus){
42      free(bonus);
43    }
44
45    /* Ecrire le bonus sur la carte */
46    void write_bonus(Map *map, Bonus * bonus){
47      map->data[bonus->y][bonus->x] = bonus->letter;
48    }
49
50    /* Supprimer le bonus */
51    Bonus* delete_bonus(Map *map, Bonus *bonus, Snake * snake){
52      // Supprimer seulement si le nombre de pas égale à 10 pas, et crée
53      if(bonus->footsteps >= 10){
54        freeBonus(bonus);
55        bonus = init_Bonus(snake, map);
56      }
57      return bonus;
58    }
59
60    /*Supplément pour le jeu snake en SDL*/
61    /*chercher si variable appartient à l'alphabet*/
62    bool belongs_lettrer(char variable){
63        int i=0, v = 0;
64        while(i< SOMME_LETTER && v == 0){
65            if(variable == tab[i]){
66                v = 1;
67          }else{
68            i++;
69          }
70        }
71        return i < SOMME_LETTER;
72    }
```

12 – Fichier `liste.c`

```c

#include "liste.h"

/* Créer une liste [0, 0] */
Liste* create_Liste(){
  Liste *l = malloc(sizeof(Liste)); // Alloue de la mémoire pour liste

  /* Créer les sentinelles */
  l->sentNext = create_Sent(); // Sentinelle avant
  l->sentBack = create_Sent(); // Sentinelle arrière

  changeNext(l->sentNext, l->sentBack); // Relier le sentinelle avant et le arrière
  return l;
}

// Libère la mémoire de la liste
/* Libére la mémoire de chaque noeud de la liste, puis de la liste pour ne pas perdre les noeuds avant */
void freeListe(Liste* l){
  // Première noeud
  Noeud *n = l->sentNext;
  // Tantque il y a un noeud, libérer et passe aux suivant
  while(n != NULL){
    l->sentNext = n->suiv; // Pour pas perdre le noeud suivant
    freeNoeud( n ); // Libérer le noeud
    n = l->sentNext;
  }
  free(l);
}

/* Chercher l'ième noeud */
Noeud* ithNoeud(Liste* l, int i){
  int j = 0; // Initialise j
  Noeud* res;
  res = l->sentNext;

  // Chercher tant que il y a un noeud et au ième
  while(res != NULL && j < i){
    res = res->suiv;
    j++;
  }
  return res;
}

/* Chercher le contenu l'ième noeud */
Elt ithElt(Liste* l, int i){
  return (content(ithNoeud(l, i)));
}

/* Inserer dans l'ième place un noeud */
void insert(Liste *l, int i, Elt e){
  Noeud* prec = ithNoeud(l,i-1); // Noeud précédent
  // Créer et pointer le noeud aux précédent
  Noeud *n = create_Noeud(e, next(prec));
  changeNext(prec, n);
}
```

```
56
57  /* Supprimer l'ième noeud */
58  void delete(Liste *l, int i){
59    if (l == NULL) {exit(EXIT_FAILURE);} // Si c'est vide, il sort et
        affiche une erreur
60
61    Noeud* prec = ithNoeud(l,i-1); // Noeud précédent
62    Noeud *n = next(prec); // Noeud suivant
63    changeNext(prec,next(n)); // Pointer le noeud suivant au précédent
64    freeNoeud( n ); // Libérer
65  }
```

13 – Fichier `main.c`

```
1
2   /* Fichier principal du jeu de serpent */
3
4   #include <stdio.h> // Pour écrire et lire
5   #include <stdlib.h> // Pour les réserves et libéres d'espace et alé
        atoire
6   #include <string.h> // Pour les fonctions str: strlen(), strcpy()
7   #include <time.h> // Pour utiliser time()
8
9   // Pour utiliser tous les fonctions qu'on à créer pour le jeu: map,
        serpent, bonus et menu
10  #include "map.h"
11  #include "snake.h"
12  #include "spawn.h"
13  #include "menu.h"
14
15
16  int main(void) {
17    // Initialisation des données
18    srand(time(NULL)); // Initialise le nombre de secondes écoulées
        depuis l'époque
19    Map *map = NULL;
20    Snake *snake = NULL;
21    Bonus *bonus = NULL;
22    char name_map[100]; // Tableau pour le nom du fichier de terrain (
        maximum 100 caractères)
23    strcpy(name_map, "carte.txt"); // Initialise la carte.txt
24
25    int choix = menu_principal(name_map); // Faire le choix
26    // Et selon le choix: partir, jouer, jouer et le sauvegarde
27    if (choix == MENU_LEAVE) {
28      printf("Au revoir !\n");
29      return 0;
30    }
31
32    if (choix == MENU_NEW) {
33      map = load_map(name_map); // Charger la carte
34      if (!map) { printf("Erreur map\n"); return 1; } // Tester si vide
35
36      snake = create_Snake(); // Créer le sepent
37      init_snake(snake); // Initialiser à 'z'
38      bonus = init_Bonus(snake, map); // Créer le bonus
39    }
40
```

```
41      if (choix == MENU_LOAD) {
42        // Charger le sauvegarde, si il rentre, c'est qu'il y a un erreur
43        if (!load_save(&snake, &bonus, &map, name_map)) {
44          printf("Erreur chargement\n");
45          return 1;
46        }
47      }
48
49      // Tantque le jeu roule, il mettre à jour
50      int run = 0, choix2;
51      while (!run) {
52        // Charger le bonus, le serpent et affiche la carte
53        write_bonus(map, bonus);
54        write_snake(map, snake);
55        print_map(map);
56
57        printf("Score : %d\n", snake->score); // Afficher le score
58
59        char c = menu_direc_leave(1); // Affiche la demande du sens
60
61        // Quitter
62        if (c == 'q') {
63          c = menu_direc_leave(2); // Demande d'affirmation de quitter
64
65          // Sauvegarde et relance la demande d'affirmation de quitter
66          while(c == 'x'){
67            save(snake, bonus, name_map);
68            c = menu_direc_leave(2);
69          }
70
71          // Choisir la carte ou rejouer le jeu
72          if (c == 'c' || c == 'r') {
73            if(c == 'c'){
74              printf("\n");
75              printf(" ------------- Taille de la carte ------------- \n")
                    ;
76              printf("1. carte%dx%d\n",16,9);
77              printf("2. carte%dx%d\n",27,14);
78              printf("3. carte%dx%d\n",49,18);
79              scanf("%d", &choix2);
80              getchar();
81
82              // Change le nom de fichier selon le choix
83              if(choix2 == 1){
84                strcpy(name_map, "carte.txt");
85              }
86              if(choix2 == 2){
87                strcpy(name_map, "carte2.txt");
88              }
89              if(choix2 == 3){
90                strcpy(name_map, "carte3.txt");
91              }
92            }
93
94            // Lance le nouveau jeu
95            // Libèrer la mémoire des données: serpent, bonus et carte
96            freeSnake(snake);
97            freeBonus(bonus);
```

```
98            free_map(map);
99
100           map = load_map(name_map); // Recharge la carte
101           // Recréer les données, et les initialisations
102           snake = create_Snake();
103           init_snake(snake);
104           bonus = init_Bonus(snake, map);
105         }
106
107         // si l'utilisateur veux ou non quitter la partie
108         if (c == 'q') {
109           printf("\n");
110           printf("Fin de la partie\n");
111           break; // Sortir imédiatement
112         }
113       }
114       // Les touches de direction
115       else if(c == 'o' || c == 'l' ||c == 'k' ||c == 'm'){
116         // Le serpent mange(0)/avance(0)/perdre(1)
117         run = mouvement_snake(snake, c, map, bonus);
118
119         bonus = delete_bonus(map, bonus, snake); // Vérifier si le
              nombre de pas >= 10 et recréer
120         if (run == 0) bonus->footsteps++; // Ajouter un pas quand le
              serpent bouge
121
122         free_map(map); // Nettoyage de la carte après utilisation
123
124
125         map = load_map(name_map); // Initialiser la map
126         if (!map) { printf("Erreur map\n"); break; } // Teste d'erreur
              de vide
127       }else{
128
129       }
130
131     }
132
133     // Affichier le phrase de fin
134     printf("Perdu ! Score final : %d\n", snake->score);
135
136     // Libèrer toutes les données de la mémoire à libèrer
137     freeSnake(snake);
138     freeBonus(bonus);
139     free_map(map);
140
141     return 0;
142   }
```

14 – Fichier `main.c` de SDL

```
1   #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <SDL2/SDL.h>
5 #include "map.h"
6 #include "snake.h"
7 #include "spawn.h"
```

```c

#define TILE_SIZE 32


int main(void) {
    Map *map = NULL;
    Snake *snake = NULL;
    Bonus *bonus = NULL;
    char nom_map[100] = "carte.txt";

    Uint32 last_input_time = 0;     // tempo de la manipulation
    Uint32 last_snake_time = 0;     // tempo de la mise a jour de serpent
    char next_direction = '\0';     // prochain touche
    char current_direction = '\0';  // touche executer

    //initialisation
    map = load_map(nom_map);
    snake = create_Snake();
    init_snake(snake);
    bonus = init_Bonus(snake, map);

    //initialisation sdl
    SDL_Init(SDL_INIT_VIDEO);

    //initialisation de la taille du jeu
    SDL_Window *window = SDL_CreateWindow(
        "Map SDL",
        SDL_WINDOWPOS_CENTERED,
        SDL_WINDOWPOS_CENTERED,
        map->width * TILE_SIZE,
        map->height * TILE_SIZE,
        0
    );

    //créer la fenêtre du jeu
    SDL_Renderer *renderer = SDL_CreateRenderer(window, -1,
       SDL_RENDERER_ACCELERATED);

    SDL_Event event; //evenement sdl (ce que le joueur fait)

    int fin = 0;

    while (!fin) {
        //il affiche sur terminal, et charge aussi( si on enlève, il n'y
            a plus rien sur SDL)
        write_bonus(map, bonus);
        write_snake(map, snake);
        print_map(map);


        printf("Score : %d\n", snake->score);

        //les touches et le croix pour sortir de la page(SDL_QUIT)
        if (SDL_PollEvent(&event)){
            switch (event.type){
            case SDL_QUIT:
                fin = 1;
                break;
```

```
64
65              case SDL_KEYDOWN:
66                  switch (event.key.keysym.sym){
67                      case SDLK_q:
68                          fin = 1;
69                          break;
70
71                      case SDLK_o:
72                          next_direction = 'o';
73                          break;
74
75                      case SDLK_l:
76                          next_direction = 'l';
77                          break;
78
79                      case SDLK_k:
80                          next_direction = 'k';
81                          break;
82
83                      case SDLK_m:
84                          next_direction = 'm';
85                          break;
86                  }
87              }
88          }
89
90          // chaque 50ms charge le jeu
91          Uint32 current_time = SDL_GetTicks();
92          if (current_time - last_input_time >= 50) {
93              last_input_time = current_time;
94
95              // mettre a jour la touche
96              if (next_direction != '\0') {
97                  current_direction = next_direction;
98                  next_direction = '\0';  // initialisation next_direction
99              }
100         }
101
102         SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255); //initialiser
                font noir
103         SDL_RenderClear(renderer); //coloré
104
105         //colorer la map
106         for (int y = 0; y < map->height; y++){
107                 for (int x = 0; x < map->width; x++){
108                     SDL_Rect tile = {
109                         x * TILE_SIZE,
110                         y * TILE_SIZE,
111                         TILE_SIZE,
112                         TILE_SIZE
113                     };
114
115                     if (map->data[y][x] == '#'){
116                             SDL_SetRenderDrawColor(renderer, 150, 150,
                                150, 255); // mur
117                     }else if (belongs_lettrer(map->data[y][x])){
118                         if(bonus->y ==y && bonus->x == x){
119                             SDL_SetRenderDrawColor(renderer, 200, 50,
```

```
                                50, 255); // bonus
120                         }else{
121                             SDL_SetRenderDrawColor(renderer, 0, 120, 0,
                                    255); // snake
122                         }
123                     }else{
124                         SDL_SetRenderDrawColor(renderer, 30, 30, 30,
                                255); // vide
125                     }
126                     SDL_RenderFillRect(renderer, &tile);
127                 }
128             }
129             SDL_RenderPresent(renderer);
130 
131             //chaque 300ms charger la map
132             if (current_time - last_snake_time >= 300) {
133                 last_snake_time = current_time;
134 
135 
136                 fin = mouvement_snake(snake, current_direction, map, bonus);
137 
138                 bonus = delete_bonus(map, bonus, snake);
139                 if (!fin) bonus->footsteps++;
140 
141 
142                 // initialisation de la map
143                 free_map(map);
144                 map = load_map(nom_map);
145             }
146 
147         }
148 
149         printf("Perdu ! Score final : %d\n", snake->score);
150 
151         //liberer tout
152         freeSnake(snake);
153         freeBonus(bonus);
154         free_map(map);
155         SDL_DestroyRenderer(renderer);
156         SDL_DestroyWindow(window);
157         SDL_Quit();
158         return 0;
159 }
```

15 – Fichier `Makefile`

```
1
2  #compiler pour chaque fichier puis all pour compiler tout dans jeu,
       clean il supprime tout les fichier.o et le jeu dans bin si il
       existe
3  #-I... pour chercher les bibliotèques .h
4  #pour le code sdl, il n'y a pas de fichier menu, donc un make sans
       menu.o .h .c ...
5
6  fichier_o = obj/map.o obj/main.o obj/noeud.o obj/liste.o obj/snake.o
       obj/spawn.o obj/menu.o
7
8  all: map.o main.o noeud.o liste.o snake.o spawn.o menu.o
9  gcc $(fichier_o) -o bin/jeu
10
11 map.o: src/map.c
12 gcc -c src/map.c -Wall -Iheader -o obj/map.o
13
14 noeud.o: src/noeud.c
15 gcc -c src/noeud.c -Wall -Iheader -o obj/noeud.o
16
17 liste.o: src/liste.c
18 gcc -c src/liste.c -Wall -Iheader -o obj/liste.o
19
20 snake.o: src/snake.c
21 gcc -c src/snake.c -Wall -Iheader -o obj/snake.o
22
23 spawn.o: src/spawn.c
24 gcc -c src/spawn.c -Wall -Iheader -o obj/spawn.o
25
26 menu.o: src/menu.c
27 gcc -c src/menu.c -Wall -Iheader -o obj/menu.o
28
29 main.o: src/main.c
30 gcc -c src/main.c -Wall -Iheader -o obj/main.o
31
32 clean:
33 rm obj/*.o ; rm bin/jeu
```

5 Compilation et exécution

D'abord accéder au dossier souhaiter(terminal ou sdl) :

```
/code_terminal
```

```
/code_sdl
```

La compilation du programme se fait à l'aide de la commande suivante :

```
make
```

Le jeu est ensuite lancé avec la commande :

```
./bin/jeu
```

6 Difficultés rencontrées

L'insertion des variables dans les listes du serpent a été compliquée. On oubliait souvent les sentinelles avant et arrière, ainsi que les valeurs à 0 qui ne font pas partie du serpent. Ces oublis provoquaient des erreurs dans le fonctionnement du jeu.

La gestion du bonus a aussi posé problème. Au lieu que le serpent arrive sur la case du bonus puis le mange, les coordonnées devenaient difficiles à gérer. La tête du serpent et le bonus avaient les mêmes coordonnées, ce qui obligeait à modifier toute la liste des positions en x et en y. En réfléchissant autrement, il suffisait de regarder la case suivante et de se demander ce que le serpent devait faire. Cela a permis d'éviter de modifier toute la liste lorsque le bonus est mangé.

Concernant la SDL, la mise à jour du jeu était difficile à régler. Lorsque le jeu était trop rapide, le serpent se déplaçait trop vite. Mais lorsque la mise à jour était ralentie, les touches du clavier ne répondaient pas immédiatement et le changement de direction du serpent se faisait avec du retard. Il a donc fallu séparer la gestion des touches et la mise à jour du jeu. De plus, le menu en SDL était trop compliqué à programmer et, par manque de temps, il n'a pas pu être terminé.

6.1 Arborescence du projet

Le projet est structuré autour de plusieurs fichiers sources et fichiers d'en-tête afin de séparer les responsabilités :

— `main.c` : point d'entrée du programme,
— `snake.c / snake.h` : gestion du serpent,
— `liste.c / liste.h` et `noeud.c / noeud.h` : structures de données,
— `map.c / map.h` : gestion de la carte,
— `menu.c / menu.h` : interface utilisateur en terminal,
— `spawn.c / spawn.h` : génération des éléments du jeu.

Un `Makefile` est également utilisé afin de faciliter la compilation du projet.

7 Conclusion

7.1 Limites et perspectives

Dans ce projet, nous avons réalisé le jeu Snake en deux versions : une version fonctionnant dans le terminal et une version avec une interface graphique à l'aide de la bibliothèque Simple DirectMedia Layer (SDL). La version terminal nous a permis de nous concentrer principalement sur le fonctionnement du jeu, comme les déplacements du serpent, la gestion des collisions et l'apparition de la nourriture.

La version graphique avec SDL a ensuite permis d'améliorer l'affichage et de rendre le jeu plus agréable à utiliser. Le fait de pouvoir réutiliser une grande partie du code déjà écrit montre que la logique du jeu était bien séparée de l'affichage, ce qui a facilité l'ajout de cette interface graphique.

Les objectifs du projet ont donc été atteints, car le jeu est jouable, fonctionnel et existe sous deux formes différentes(sdl et terminal).

Même si le jeu fonctionne correctement, certaines améliorations pourraient encore être apportées. Par exemple, le jeu ne propose qu'un seul niveau de difficulté et l'interface graphique reste assez simple. Il n'y a pas non plus de système de score avancé ni de sauvegarde des résultats.

Pour aller plus loin, il serait possible d'ajouter :

- plusieurs niveaux de difficulté,
- un système de score plus complet,
- des améliorations visuelles comme des animations ou des sons,
- de nouvelles règles ou mécaniques de jeu.

Ces améliorations permettraient de rendre le jeu plus complet et plus intéressant, tout en approfondissant les connaissances en programmation et en développement de projet.