# 

# Programmation SDL

**Antoine Drabble**

Table des matières

[Programmation SDL 1](#_Toc347324711)

[Cahier des charges 3](#_Toc347324712)

[Introduction 3](#_Toc347324713)

[Description de la demande 3](#_Toc347324714)

[Contraintes 3](#_Toc347324715)

[Déroulement du projet 4](#_Toc347324716)

[Journal de travail 6](#_Toc347324717)

[Amélioration possible 10](#_Toc347324718)

[Bug 10](#_Toc347324719)

[Schéma 7](#_Toc347324720)

[Manuel d’utilisation : 8](#_Toc347324721)

[Commandes 8](#_Toc347324722)

[Modifier un niveau 8](#_Toc347324723)

[Ajouter un sprite 9](#_Toc347324724)

[Ajouter un niveau 9](#_Toc347324725)

[Partie Technique 11](#_Toc347324726)

[Introduction 11](#_Toc347324727)

[Les structures 11](#_Toc347324728)

[Les Fonctions 13](#_Toc347324729)

## Cahier des charges

### Introduction

#### Contexte

Pour le stage TPI Introduction, je vais créer un jeu qui me permettra d’améliorer mes connaissances dans le langage C et SDL. Et de pouvoir peut-être continuer sur de la programmation 3D avec OpenGL par exemple.

### Description de la demande

#### Les objectifs

Faire un jeu 2D ressemblant à Mario, codé avec la bibliothèque SDL écrite en C. le jeu reprendra les éléments essentiels du jeu Super Mario Bros Nes.

#### Produit du projet

Il y aura un fichier exécutable .exe pour lancer le jeu et les fichier dont le jeu dépend.

#### Les fonctions du produit

* Plusieurs cartes.
* Gestion des collisions entre le personnage et l’environnement.
* Gestion de la musique.
* On pourra se déplacer vers la gauche la droite et sauter.
* Des blocs que l’on pourra utiliser pour récupérer des bonus.
* Il y aura des obstacles.
* Mario aura 2 niveaux. Le premier sera standard, le deuxième le rendra plus gros et plus resistant.

### 

### Contraintes

#### Contraintes de coûts

Les seules contraintes de coûts sont :

* La place de travail
* Le salaire d’un apprentis 3ème année
* La licence MS Project 2010 Standard

Le programme code::blocks ainsi que la librairie SDL sont libres.

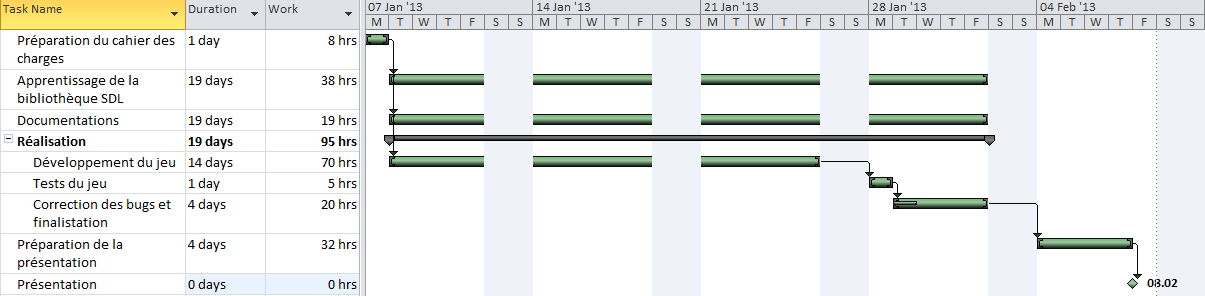
#### Contrainte de délais

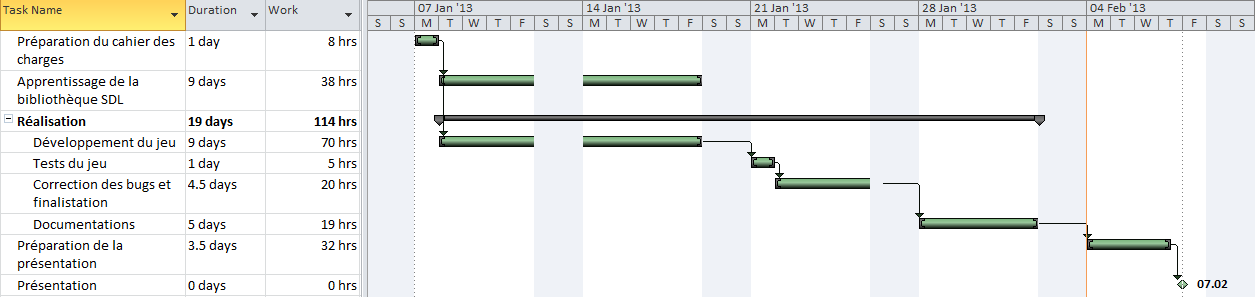
Il faudrait qu’une première version fonctionnel soit disponible à partir du 25.01.13 pour avoir encore 2 semaines pour préparer la présentation ainsi que corriger les bugs.

### Déroulement du projet

#### Planification

La planification a été plutôt bien suivie, ce qui a changé, c’est que le développement a été fait en 3 semaines plutôt que les 4 prévues car je n’ai pas tellement fait de documentation pendant le développement. J’ai pris des notes mais pas rédigé de documentation complète car le code risquait de changer.





#### Ressources

##### Matériel :

* Mon poste de travail
* Ma machine.

##### Logiciel :

* La librairie SDL
* Code::Blocks
* MS Project 2010 standard

##### Sites web**:**

* <http://www.siteduzero.com>

**Mes choix**

J’ai décidé pour mon projet, d’utiliser l’IDE code::block. C’est un environnement de développement libre sous license GNU GPL. Je l’ai choisi car il est libre et gratuit. Mais j’aurais très bien pu utiliser Visual Studio qui lui est payant sauf en version express.

J’ai choisis la bibliothèque SDL ( Simple DirectMedia Layer) principalement car il existe un très bon tutoriel pour celle-ci sur le C pour l’utiliser et je possédais déjà des bases en C. Elle est sous license GNU LGPL, on peut donc faire des programmes propriétaires avec sans devoir partager les codes sources.

J’ai aussi utiliser les librairies SDL\_Images et FMOD. Celles-ci remplacent des fonctions de la SDL qui ne sont pas super. Par exemple la SDL ne gère que les fichiers images .BMP ce qui limite beaucoup, alors que SDL\_Images supporte beaucoup de types.

Et pour FMOD, la SDL a des fonctions complexes et qui ne fonctionnent pas très bien non plus, alors que FMOD gère le son assez facilement sans problèmes.

## Journal de travail

|  |  |
| --- | --- |
| 07.01.13 | Mise en place du cahier des charges.  Rappel de la SDL |
| 08.01.13 | Création des crédits et du menu.  Ajout de la musique  Recherche de comment faire une carte.  Apprentissage du TileMapping |
| 10.01.13 | Apprentissage du C/SDL/TileMapping  Création de la carte et du code qui va lire le fichier de la carte |
| 11.01.13 | Ajout des sprites.  Ajout affichage de la carte |
| 14.0.1.13 | Tutoriel sur l’amélioration des events en sdl.  Ajout du scrolling de la carte. |
| 15.01.13 | Ajout du personnage mario et de ses déplacements.  Début de gestion des collisions. |
| 17.01.13 | Amélioration du code, création de la doc. |
| 18.01.13 | Ajout d’une « gravité »  Ajout des images de mario  Résolution de problèmes de collision. |
| 21.01.13 | Bloquer le matin entier par un bug. Après de longues recherches le problème venait d’un include. Je ne comprends pas pourquoi mais il créait une erreur donc j’ai trouvé une solution alternative. (le debugger ne donnait pas la ligne du problème…)  Ajout du mouvement de mario  Résolutions de bugs |
| 22.0.13 | Ajout des sons  Ajout de l’échec ou réussite du niveau  Améliorations de certain sprites  Tentative de résolution d’un bug qui crash le jeu quand on meurt |
| 24.01.13 | Tentative de résolution d’un bug qui crash le jeu quand on meurt, toujours pas réussi.  Ajout des champignons |
| 25.01.13 | Essaie d’ajout de mouvement au champignon mais il y a un bug. J’essaierai de faire ça si j’ai le temps.  Ajout de niveaux et du menu de choix de niveau. |
| 28.01.13 | Résolution d’un bug d’affichage et de bug de collision avec la limite droite de la carte.  Travail sur la documentation. |
| 30.01.13 | Travail sur la documentation |
| 31.01.13 | Fin de la documentation |
| 01.02.13 | Préparation de la présentation |
| 04.02.13 | Préparation de la présentation |

## Schéma

Ce schéma montre comment sont utilisés les fichiers.

On a, on le verra plus tard, toute la phase d’initialisation des images, sons… en rouge.

Dans le main.c on va d’abord charger en mémoire tous les sons de audio.c et avec game.c on va chercher les images de la carte, des personnages et des objets, on va aussi initialiser les variables.

En bleu c’est le chemin que suit le jeu (c’est expliqué plus précisément plus tard fonction par fonction)

On commence par Main.c on utilise audio.c pour lancer la musique de Mario en boucle, on peut lancer les crédits ou le jeu.

Si on lance le jeu, il va faire les initialisations et suivre la flèche bleu en boucle jusqu’à ce qu’on quitte le jeu.

Game.c

Main.c

Credits.c

Audio.c

Map.c

Event.c

Objectevent.c

Object.c

Char.c

Utilisation

Chargement

## Manuel d’utilisation :

### Commandes

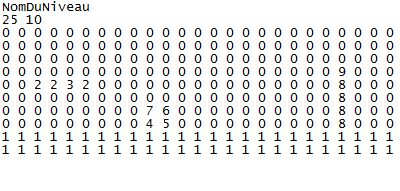
Voici la liste des touches utilisables :

|  |  |
| --- | --- |
| Se déplacer vers la gauche | Flèche directionnel gauche |
| Se déplacer vers la droite | Flèche directionnel droite |
| Sauter | Flèche directionnel haut |
| Choix dans les menus | Touches 1, 2, 3… |
| Retourner au menu | Escape |

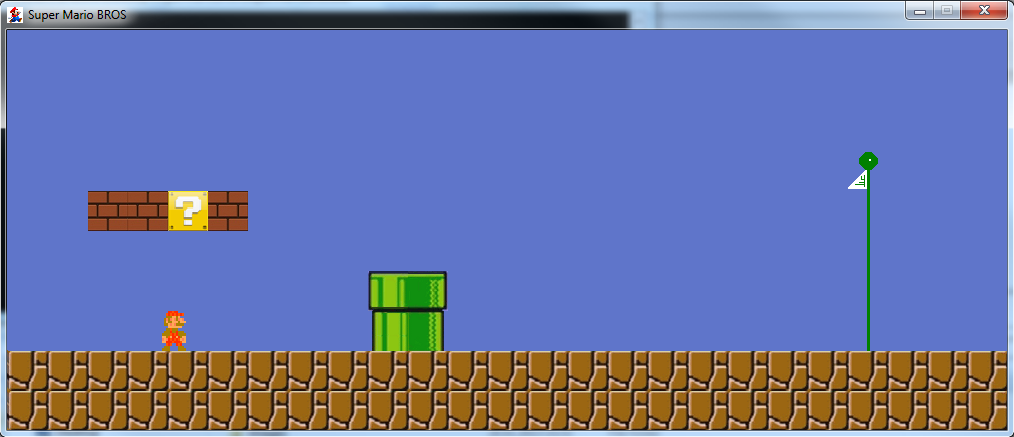
### Modifier un niveau

Les fichiers qui contiennent les niveaux du jeu ont l’extension .lvl. Ils sont écris de la manière suivante. D’abord le nom du niveau, ensuite la largeur et la hauteur du niveau. Et on écrit le niveau, selon le bloc que l’on veut mettre.

Par exemple ce niveau :



Donnera ça :



**Relation bloc-numéro :**

|  |  |
| --- | --- |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\sky.png | 0 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\sol.png | 1 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\block.png | 2 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\boite.png | 3 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\tuyau1.png | 4 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\tuyau2.png | 5 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\tuyau3.png | 6 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\tuyau4.png | 7 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\fin1.png | 8 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\fin2.png | 9 |
| C:\Users\drabble\Documents\SDL\SuperMarioBros\images\pique.png | 10 |

Si l’on veut modifier la taille de la carte, il faut ajouter les numéros sans se tromper dans les espaces et le nombre que l’on en met et modifier en haut le nombre bloc et en largeur/hauteur.

### Ajouter un sprite

Si on veut ajouter un sprite, on est obligé de modifier le code :

* Ajouter l’image du sprite de taille 40x40 dans le dossier images du programme en format PNG de préférence.
* Modifier la variable globale NbSprites dans map.h.
* Ajouter une ligne pour charger l’image du sprite dans ChargerImages (map.c).
* Ajouter une ligne pour définir si on peut traverser notre sprite ou pas dans ChargerImages (map.c).
* Ajouter une ligne pour libérer le sprite dans LibererMap (map.c).

### Ajouter un niveau

Pour ajouter un niveau il faut :

* Créer le niveau en respectant le format, ne pas ajouter d’espaces, mettre le nombre de sprites en largeur en premier et en hauteur en deuxième. Une seule erreur d’écriture ferait crasher le jeu. (voir fichier exemple dans la section « modifier un niveau »
* Modifier l’image du menu des niveaux.
* Ajouter la gestion d’une touche supplémentaire dans le « waitevent » du menu.

## Partie Technique

### Introduction

Le jeu fonctionne de la manière suivante :

On commence par lancer le fichier main.c qui va lancer la musique, et afficher l’image du menu. Le main attend juste qu’on appuie sur les touches 1, 2 ou escape.

La touche 2 lance les crédits, la touche escape quitte le jeu et la touche 1 lance le jeu.

On va donc appuyer sur 1 et lancer le fichier game.c qui va d’abord afficher le menu de sélection des niveaux. On en choisit un avec les touches 1,2 ou 3.

Il va ensuite charger en mémoire le fichier .lvl qui correspond au niveau qu’on a choisis dans un tableau à plusieurs dimensions.

Ensuite il va lancer les fonctions qui affichent la carte, gère les touches, les collisions… en boucle jusqu’à ce qu’on réussisse/perde le niveau ou que l’on clique escape.

AfficherObject

Evolue

MapScroll

EssaiDeplacement

CollisionDecor

AfficherMap

ObjectMove

AfficherPerso

NiveauFini

Deplace

Main

Audio\_Play

Audio\_init

Credit

Game

ChargerImages

ChargerObject

ChargerMap

ChargerChars

ChargerChars

LibererMap

LibererObject

LibererChars

### 

### Les structures

#### Map

La structure Map, définie dans le fichier map.h, contient toutes les informations relatives à la carte.

* Le nombre de bloc en largeur et hauteur (ce qui permet de donner les limites de la carte) qui ont été définis dans le fichier .lvl qui contient le niveau.
* Un tableau à plusieurs dimensions, où est stockée la carte
* Et un deuxième tableau de même taille que celui de la carte qui permet de stocker des informations par rapport aux blocs de la carte. La seule utilisation que j’en fais c’est si on utilise un bloc « ? » qui donne un champignon, dans le tableau « used », à la même position qu’où a été définis le bloc dans le tableau « loadedmap », je mets la valeur à 1 ce qui veut dire qu’il a utilisé. Ensuite dans les collisions, si on touche un bloc « ? », on regarde à la même position que dans le premier tableau s’il vaut 1. S’il ne vaut pas 1 il n’est pas utilisé et un champignon doit donc apparaître.
* L’endroit où doit se trouver la carte par rapport à la fenêtre et à Mario. Pour qu’on affiche l’endroit où se trouve Mario et pas un autre endroit de la carte.

1. **typedef** **struct**
2. {
3. **int** Nb\_Block\_W, Nb\_Block\_H;
4. Uint16\*\* LoadedMap;
5. Uint16\*\* Used;
6. **long** **int** xscroll,yscroll;
7. }Map;

#### Sprites

La structure Sprites contient :

* Une variable SDL\_Surface qui sert à contenir une image. Elle contiendra donc l’image de chaque sprites.
* La deuxième variable dira si on peut traverser le sprite ou pas.
* **typedef** **struct**
* {
* SDL\_Surface\* sprite;
* **int** traverser;
* }Sprites;

#### Char

C’est la structure qui contient toutes les informations de Mario.

* Chaque image, 6 quand il est grand, 6 quand il est petit.
* Sa position.
* Des informations pour voir s’il peut sauter, combien de temps il a sauté, quelle image est utilisée en ce moment… On verra l’utilisation de chaque variable dans les fonctions.
* **typedef** **struct**
* {
* SDL\_Surface\* image;
* SDL\_Rect position;
* **int** jump, jumptime, G, collisionx, selectedmario;
* **int** directionmario, temp, dernieredirection, win, lose, lvl, lvlperdu, invisible;
* }Chars;

#### Object

Cette structure contient les variables des « objets ».

Je n’ai pas pu ajouter d’ennemis, ils auraient aussi été définis ici, en l’occurrence il n’y a que les champignons qui utilisent cette structure.

* Il y a l’image de l’objet
* La position de l’objet, qui est un tableau de position car il faut une position par objet et il peut y avoir plusieurs objets par exemple si on tape plusieurs blocs il y aura plusieurs champignons.
* Le nombre d’objet et si l’objet a été utilisé ou pas. Il faut quand même qu’il disparaisse si on l’utilise !

1. **typedef** **struct**
2. {
3. SDL\_Surface\* image;
4. SDL\_Rect position[50];
5. **int** nb\_objet, used[50];
6. }Object;

### Les Fonctions

#### Main (main.c)

On définit la taille de l’écran et la position de l’image du menu. On la colle sur la surface screen qui contient la taille de l’écran puis on affiche la surface screen.

On lance la fonction audio\_init qui va charger les musiques et le son et on lance la fonction audio\_play avec comme paramètre « 1 » pour qu’il lance la musique de Mario en boucle. Ces fonctions seront décrites plus loin.

Ensuite on attend simplement qu’on appuie une touche. 3 touches du clavier sont gérées, la touche 1 qui lance le jeu (la fonction game), la touche 2 qui lance les crédits (fonction credit) et la touche escape qui va quitter le jeu.

#### Credits (credit.c)

Pour les crédits j’ai juste fait une image sur paint avec mon « titre » et mon nom . Je la fais défiler de haut en bas de la fenêtre et dès qu’elle atteint le bas, on retourne dans le menu. J’ai aussi ajouté la gestion de la touche escape, qui permet de retourner dans le menu plus rapidement.

#### Audio\_init (audio.c)

J’ai définis en variable global les variables qui contiendront les sons pour qu’elles soient accessibles facilement sur chaque fonction, plutôt que de les passer en paramètre.

J’ai initialisé FMOD et assigné les sons au variable pour qu’on n’ait plus qu’à « play » la variable.

#### Audio\_play (audio.c)

J’ai utilisé un moyen très simple pour lire les sons :

1. **if** (song==1)
2. FMOD\_System\_PlaySound(sys, 0, music, 0, NULL); //Joue la musique
3. **if** (song==2)
4. FMOD\_System\_PlaySound(sys, FMOD\_CHANNEL\_FREE, jump, 0, NULL);
5. **if** (song==3)
6. FMOD\_System\_PlaySound(sys, FMOD\_CHANNEL\_FREE, lose, 0, NULL);
7. **if** (song==4)
8. FMOD\_System\_PlaySound(sys, FMOD\_CHANNEL\_FREE, win, 0, NULL);
9. **if** (song==5)
10. FMOD\_System\_PlaySound(sys, FMOD\_CHANNEL\_FREE, shroom, 0, NULL);
11. **if** (song==6)
12. FMOD\_System\_PlaySound(sys, FMOD\_CHANNEL\_FREE, damage, 0, NULL);

Mais si on met beaucoup de sons, il faudrait trouver un autre moyen, par exemple avec un tableau de type FMOD\_SOUND comme ça plutôt que de mettre plein de if, on fait simplement :

1. FMOD\_System\_PlaySound(sys, FMOD\_CHANNEL\_FREE, MonTableauDeSons[ParamètrePassé, 0, NULL);

#### Audio\_stop (audio.c)

J’ai simplement utilisé les fonctions FMOD de libération des variables FMOD.

#### Game (game.c)

La fonction game est l’une des deux qui peuvent être appelée par le main. C’est la fonction de base du fonctionnement du jeu. Celle qui va lancer toutes les fonctions de chargements d’images et autres. Celle qui va définir pour la première fois les structures et celle qui appellera en boucle les fonctions du jeu.

D’abord on définit toutes les variables les plus importantes pour le jeu :

Map\* m;

Sprites\* S;

Chars mario;

Chars\* marioimages;

Object\* shroom;

Input in;

int continuer = 0;

int tempsPrecedent = 0, tempsActuel = 0;

char \*niveau;

Ensuite on affiche le petit menu de choix de niveau, on charge les images et la map et on affiche déjà

S = ChargerImages();

shroom = ChargerObject();

m = ChargerMap(niveau);

ChargerChars(&mario, m, marioimages);

AfficherMap(m, screen, S);

SDL\_Flip(screen);

Ensuite on fait tourner toutes les fonctions du jeu en boucle dans un pollevent:

MapScroll(m, &mario);

Evolue(&in,m,&mario,S, shroom);

AfficherMap(m,screen,S);

objectmove(shroom, &mario, m->xscroll,m->yscroll, m, S);

AfficherObject(screen, shroom, m->xscroll,m->yscroll);

AfficherPerso(&mario,screen,m->xscroll,m->yscroll, marioimages);

SDL\_Flip(screen);

continuer = NiveauFini(screen, &mario, m);

Une fois que la personne a réussi, perd ou quitte la partie, on sort du pollevent, on libère les images et les structures et on retourne dans le waitevent du menu principal.

LibererMap(m,S);

LibererChars(&mario, marioimages);

LibererObject(shroom);

#### ChargerImages (map.c)

Cette fonction charge l’image de chaque sprite avec la fonction IMG\_Load de la librairie SDL\_Image et l’assigne à un chiffre qui sera celui utilisé dans le fichier qui contient le niveau.

Je passe aussi chaque image dans la fonction SDL\_DisplayFormat qui va changer le format de l’image pour l’afficher plus rapidement.

Et finalement je définis pour chaque sprite si on peut le traverser ou non. Ce qui est nécessaire pour la gestion des collisions.

#### ChargerMap (map.c)

Cette fonction lit le fichier qui contient la location des blocs du niveau.

Le fichier doit être écrit sans erreur sinon rien ne fonctionnera, le fichier contient d’abord le nom du niveau (sans espaces) puis un retour à la ligne, ensuite le nombre de blocs de la largeur du niveau et un espace et le nombre de blocs pour la hauteur du niveau puis retour à la ligne.

Ensuite on met le numéro correspondant à notre bloc (définis dans LoadImages), un espace et le numéro du bloc suivant, ainsi de suite jusqu’à ce qu’on ait mis le nombre de bloc en largeur, on fait un retour à la ligne et on écrit la ligne suivante. Il ne faut surtout pas oublier/rajouter d’espaces !

La nombre de blocs pour la hauteurs et la largeur sont enregistré dans leur variable respectives dans la structure Map. La carte sera enregistrée dans le tableau à deux dimensions « LoadedMap », aussi dans la structure Map.

#### AfficherMap(map.c)

Cette fonction va afficher la carte, mais seulement où se trouve Mario. Car on ne veut pas utiliser des ressources en plus pour afficher la carte qui se situe en dehors de la fenêtre.

Il faut commencer par prendre la coordonnée x de la carte par rapport au personnage (on a une variable pour ça, xscroll et yscroll) et on le divise par la taille d’un sprite comme ça on saura le combientième sprite c’est en partant de la gauche. On fait la même chose avec la coordonnée y de la carte et avec les 2 résultats, on aura la coordonnée x et y du tableau qui contient la carte. Donc on a plus qu’à lui dire d’afficher aux coordonnée x et y de notre carte, l’image qui correspond au numéro stocké dans le tableau.

Deux variables globales sont stockées dans map.h, NbSprites et Size\_Sprite. Je les ai définies globalement car elles doivent être accessible facilement, et modifiées facilement aussi car le nombre de sprite ou la taille d’un sprite peut changer. NbSprites sert à l’allocation mémoire de la structure qui va contenir toutes mes sprites. Comme ça si j’ajoute un Sprite, je modifie juste ce nombre et il sera automatiquement modifié quand je fais le malloc de la structure Sprites. Size\_Sprite est souvent utilisé pour des calculs sur la carte. Par exemple pour savoir le combientième bloc c’est depuis la gauche en connaissant sa distance depuis le bord de la carte ou pour savoir chaque combien de pixel il faut afficher un bloc sur la carte.

Pour afficher sur toute la fenêtre il suffit de mettre ce code dans deux for, qui vont de minimum x à maximum x de la fenêtre et même chose pour y. Et on incrémente de sorte qu’on passe d’un sprite à un autre.

WindowW et WindowH sont deux autres variables globales définient dans game.h, elles servent à définir la taille de la fenêtre. Elles sont aussi définies en variable global pour pouvoir les modifier facilement.

1. **void** AfficherMap(Map\* m, SDL\_Surface\* screen, Sprites\* S)
2. {
3. **int** i,j;
4. SDL\_Rect Rect\_dest;
5. **int** numero\_tile;
6. **int** minx, maxx, miny, maxy;
7. minx = m->xscroll / Size\_Sprite-1;
8. miny = m->yscroll / Size\_Sprite-1;
9. maxx = (m->xscroll + WindowW)/Size\_Sprite;
10. maxy = (m->yscroll + WindowH)/Size\_Sprite;
11. **for**(i=minx;i<=maxx;i++)
12. {
13. **for**(j=miny;j<=maxy;j++)
14. {
15. Rect\_dest.y = j\*Size\_Sprite - m->yscroll;
16. Rect\_dest.x = i\*Size\_Sprite - m->xscroll;
17. **if** (i<0 || i>=m->Nb\_Block\_W || j<0 || j>=m->Nb\_Block\_H)
18. {
20. }
21. **else**
22. {
23. numero\_tile = m->LoadedMap[i][j];
24. SDL\_BlitSurface(S[numero\_tile].sprite,NULL,screen,&Rect\_dest);
25. }
26. }
27. }
28. }

#### LibérerMap(map.c)

Après avoir chargé les images les structures en mémoire, il faut les supprimer  donc :

1. SDL\_FreeSurface(S[0].sprite);
2. SDL\_FreeSurface(S[1].sprite);
3. SDL\_FreeSurface(S[2].sprite);
4. SDL\_FreeSurface(S[3].sprite);
5. SDL\_FreeSurface(S[4].sprite);
6. SDL\_FreeSurface(S[5].sprite);
7. SDL\_FreeSurface(S[6].sprite);
8. SDL\_FreeSurface(S[7].sprite);
9. free(S);
10. free(m);

#### ChargerChars (char.c)

J’ai commencé par définir la position de départ de Mario. On peut le mettre où on veut, j’ai juste fait en sorte que sa position y soit deux bloc en-dessus du bas de la carte pour qu’il apparaisse toujours au même endroit si on prend des cartes de tailles différentes.

La variable selectedmario est définie à 1. Cette variable correspond à l’image de Mario affichée, on l’utilise dans la fonction affichermario.

La variable lvl est le niveau de Mario, s’il prend un champignon il passe niveau deux, s’il prend des dégâts, il repasse niveau un. Et s’il est niveau 1 et qu’il prend des dégâts ou qu’il meurt, la variable lvlperdu devient 1 et la prochaine fois qu’on passe dans la fonction CollisionDecor, il verra qu’on a perdu et ça va interrompre la partie.

Pour le reste on charge juste les images avec IMG\_LOAD.

1. Mario->position.w = 30;
2. Mario->position.h = 40;
3. Mario->position.x = 150;
4. mario->position.y = m->Nb\_Block\_H\*Size\_Sprite-2\*Size\_Sprite-mario->position.h;
6. Mario->selectedmario = 1;
7. Mario->lvl = 1;
8. Mario->lvlperdu=0;
10. marioimages[0].image = IMG\_Load("images/mario1.png");
11. marioimages[1].image = IMG\_Load("images/mario2.png");
12. marioimages[2].image = IMG\_Load("images/mario3.png");
13. marioimages[3].image = IMG\_Load("images/mario4.png");
14. marioimages[4].image = IMG\_Load("images/mario5.png");
15. marioimages[5].image = IMG\_Load("images/mario6.png");
17. marioimages[6].image = IMG\_Load("images/mario7.png");
18. marioimages[7].image = IMG\_Load("images/mario8.png");
19. marioimages[8].image = IMG\_Load("images/mario9.png");
20. marioimages[9].image = IMG\_Load("images/mario10.png");
21. marioimages[10].image = IMG\_Load("images/mario11.png");
22. marioimages[11].image = IMG\_Load("images/mario12.png");

#### LibererChars (char.c)

J’ai simplement fait un « SDL\_FreeSurface » pour chaque image de Mario, et un « free » pour les deux objets de structures (Mario et marioimages)

#### AfficherPerso (char.c)

La fonction AffichePerso va afficher Mario. Si on se déplace vers la droite il afficher les 3 images de Mario en direction de la droite en alternant d’image chaque fois qu’on passe 30 fois dans la fonction (sachant qu’on passe une fois dans la fonction toutes les 4 millisecondes environ ça donne un bonne effet de mouvement), même chose pour la gauche. Si on ne bouge pas on affiche l’image de base.

Le tout est mis dans deux if, le premier :

* Si on a pris des dégâts, pendant une certaine durée il affiche Mario une fois sur deux pour montrer qu’il a pris des dégâts. Il est aussi invincible pendant ce temps mais on verra cela dans les fonctions CollisionDecor.

Le deuxième :

* Si Mario est grand, on affiche les images de lui en grand et s’il est petit même chose.

#### ChargerObject(object.c)

On fait un malloc à notre objet shroom de la structure Object, on charge l’image du champignon (notre seul objet pour l’instant) et on définit la variable nb\_objet à 0.

#### AfficherObject (object.c)

D’abord on définit le SDL \_Rect de chaque objet, on fait ça dans cette fonction pas comme pour Mario où on le fait dans la fonction de chargement car il apparait des champignons, il faut définir leur position pendant la partie, pas quand il charge les images et le reste. On fait –xscroll et –yscroll pour les positions x et y car on veut la position par rapport à la fenêtre.

Ensuite pour chaque champignons on va l’afficher à sa position sauf si on a passé dessus et qu’il est déjà utilisé. On vérifie ça dans la fonction ObjectMove.

1. **int** i;
2. SDL\_Rect position[50];
4. **for** (i=1; i<=shroom->nb\_objet; i++){
5. position[i].x = shroom->position[i].x - xscroll;
6. position[i].y = shroom->position[i].y - yscroll;
7. position[i].w = Size\_Sprite;
8. position[i].h = Size\_Sprite;
9. }
11. **for** (i=1; i<=shroom->nb\_objet; i++){
12. **if** (shroom->used[i]==0){
13. SDL\_BlitSurface(shroom->image,NULL,screen,&position[i]);
14. }
15. }

#### LibererObject (object.c)

SDL\_Freesurface de l’image du champignon et free de l’objet créé avec la structure.

#### ObjectMove (ObjectEvent.c)

Cette fonction vérifie si Mario touche le champignon.

On définit les positions de chaque champignon

1. **for** (i=1; i<=shroom->nb\_objet; i++){
2. position[i].x = shroom->position[i].x - xscroll;
3. position[i].y = shroom->position[i].y - yscroll;
4. position[i].w = Size\_Sprite;
5. position[i].h = Size\_Sprite;
6. }

On vérifie s’il est déjà utilisé et s’il ne touche le champignon, s’il touche le champignon, on va d’abord lancer la musique et si notre Mario est au niveau 1 il va grandir, on modifie donc sa taille et la variable qui contient son niveau.

Finalement on met shroom->used[i] à 1 pour dire que le champignon est utilisé.

1. **for** (i=1; i<=shroom->nb\_objet; i++){ //On passe une fois par champignon
2. **if** (shroom->used[i]==0){
3. **if**((position[i].x >= mario->position.x -xscroll + mario->position.w) //Check si mario touche le champignon
4. || (position[i].x + position[i].w <= mario->position.x -xscroll)
5. || (position[i].y >= mario->position.y -yscroll + mario->position.h)
6. || (position[i].y + position[i].h <= mario->position.y - yscroll)
7. ){}
8. **else**{
9. audio\_play(5);
10. **if**(Mario->lvl==1){
11. Mario->position.w = 32;
12. Mario->position.h = 60;
13. Mario->position.y -=20;
14. Mario->lvl=2;
15. }
16. shroom->used[i]=1;
17. }
18. }
19. }

#### Evolue (Event.c)

C’est la première fonction de la série de fonction qui va gérer les collisions.

On prépare deux variables qui contiendront le mouvement qui a été fait, on ne l’applique pas encore à Mario justement parce que s’il y a une collision il faudra annuler le déplacement.

Donc on fait la première variable vx et la deuxième vy et si on se déplace vers la gauche, on fait vx moins le déplacement, si vers la droite on fait vx plus le déplacement.

Si on saute on fait vy – le déplacement mais j’ai dû trouver un moyen pour qu’il redescende à un moment. J’ai donc fait une variable qui s’appelle jumptime, qui s’incrémente de 10 à chaque fois qu’on passe dans la fonction et dès qu’elle atteint 1000 on ne peut plus monter. Donc la « gravité » définie plus tard le fera redescendre.

Ensuite on lance la fonction deplace avec en paramètre nos deux variable de déplacement.

#### MapScroll (Event.c)

Cette fonction sert à gérer la variable xscroll et yscroll qui définissent l’endroit de la fenêtre par rapport à Mario ça fait en sorte que Mario soit toujours au milieu de l’écran, sauf si on est prêt du bord de la carte.

1. m->xscroll = mario->position.x + mario->position.w/2 - WindowW/2;
2. m->yscroll = mario->position.y + mario->position.h/2 - WindowH/2;
3. **if** (m->xscroll<0)
4. m->xscroll=0;
5. **if** (m->yscroll<0)
6. m->yscroll=0;
7. **if** (m->xscroll>m->Nb\_Block\_W\*Size\_Sprite-WindowW-1)
8. m->xscroll=m->Nb\_Block\_W\*Size\_Sprite-WindowW-1;
9. **if** (m->yscroll>m->Nb\_Block\_H\*Size\_Sprite-WindowH-1)
10. m->yscroll=m->Nb\_Block\_H\*Size\_Sprite-WindowH-1;

#### Deplace (Event.c)

Cette fonction est utile seulement si on met des vitesses de déplacement plus grande, car si le déplacement est plus grand qu’un sprite (voir tableau en-dessous) on va le traverser.

Donc on est dans le cas 2, la fonction va se relancer avec cette fois la taille du vecteur de déplacement divisé par deux, et si c’est toujours trop grand, on le divise par trois etc.

Ensuite on lance la fonction EssaiDeplacement avec en paramètre nos deux variable de déplacement.

|  |  |
| --- | --- |
| Cas 1  Si Mario est le carré rouge, on va regarder dans la deuxième position s’il touche un bloc. Si oui, on ne fait rien, sinon on le déplace. | Cas 2  Si le vecteur du déplacement est plus grand qu’un sprite, on va regarder s’il touche un bloc dans la deuxième position et comme il n’en touche pas, si on n’avait pas mis la fonction Deplace pour contrôler, il aurait traversé le bloc. |

#### EssaiDeplacement(event.c)

D’abord on va créer un SDL\_Rect qui sera le deuxième carré dans notre tableau en-dessus. Ça sera Mario dans ces nouvelles coordonnées. Ensuite on lance la fonction CollisionDecor en lui passant le Mario, s’il retourne 1 il y a une collision mais s’il retourne 0 il n’y en a pas et il va donc appliquer au vrai Mario sa nouvelle position.

Entre deux il y a plusieurs contrôles, d’abord si ça nouvelle position est à -100px sous la carte, ça veut dire qu’il est tombé et qu’il a perdu, on met donc la variable Mario->lose à 1.

Ensuite si la gravité est activée on descend de 2px à chaque fois qu’on passe dans la fonction.

Si elle n’est pas activée, ça veut dire qu’on touche le sol (on verra ça dans la fonction suivante) et on peut donc sauter.

Pour les 2 suivants, ligne 10-11-12-13, on vérifie si le perso touche le bord gauche ou droite de la carte, les collisions avec les blocs sont gérées par d’autres if mais ne gère pas les collisions avec le bord de la carte, et quand on se déplace vers la droite ou gauche contre le bord de la carte on reste bloquer contre, j’annule donc le déplacement de Mario s’il est contre le bord gauche ou droite.

Le suivant est pour les collisions avec le haut de la carte, si on le touche, on ne peut plus sauter et on redescend (comme si on c’était taper la tête). Car de toute façon on ne peut pas monter plus haut donc si le saut n’est pas fini, on va juste rester bloquer environ 1 secondes contre le haut.

1. **int** temp;
2. temp = CollisionDecor(m, &test, S, mario, vx, shroom);
4. **if**(test.y+test.h>(Size\_Sprite\*m->Nb\_Block\_H)+100)
5. Mario->lose=1;
6. **if** (mario->G==1) // GravitÃ© activÃ©e si on ne touche pas le sol
7. test.y += 2;
8. **else**
9. Mario->jumptime = 1;
10. **if** (test.x-1<0) //Si on touche le bord gauche de la map
11. test.x-=vx;
12. **if** (test.x + test.w>=m->Nb\_Block\_W\*Size\_Sprite-2) //bord droite
13. test.x-=vx;
14. **if** (test.y<0){ //Si on touche le haut de la map
15. Mario->jumptime = 1001;
16. test.y+=vy;
17. }
18. **if** (temp==0) //Si pas de colision on modifie la position du personnage
19. {
20. Mario->position = test;
21. **return** 1;
22. }
23. **return** 0;

#### CollisionDecor(event.c)

D’abord on va préparer toutes les variables, le but de cette fonction est de voir si Mario touche un bloc dans sa nouvelle position et de gérer plusieurs événements. Sa nouvelle position est la variable perso.

Ces 4 variables donnent les coordonnées des blocs qui touchent Mario.

1. **int** xmin,xmax,ymin,ymax,i,j,indicetile;
2. Mario->G=1;
3. xmin = perso->x / Size\_Sprite;
4. ymin = perso->y / Size\_Sprite;
5. xmax = (perso->x + perso->w -1) / Size\_Sprite;
6. ymax = (perso->y + perso->h -1) / Size\_Sprite;

Ici on regarde si Mario a pris des dégâts il y a peu de temps, on voit plus tard comment on sait si Mario a pris des dégâts. S’il en a pris on fait -1 à la variable comme ça après un certain temps elle vaudra de nouveau 0 et Mario ne sera plus invincible.

Ensuite on le rend invisible une fois sur deux pour montrer qu’il ne peut pas prendre d’autres dégâts pendant un certain temps.

1. **if** (Mario->lvlperdu>0){
2. Mario->lvlperdu--;
3. **if**(Mario->invisible==1)
4. Mario->invisible=0;
5. **else**
6. Mario->invisible=1;
7. }
8. **else**{
9. Mario->invisible=0;
10. }

Et ensuite on fait tous les check pour voir s’il y a un bloc qui touche.

Ligne 19 à 43 on va regarder s’il y a un bloc en-dessous, s’il y a bien un bloc en-dessous et qu’on ne peut pas le traverser, on désactive la gravité. Car si elle restait activée on ferait une collision avec le sol et on ne pourrait plus se déplacer. On met Mario->jump à 1 pour dire qu’il peut sauter. On regarde ligne 27 si le bloc est le bloc des piques, Si on n’a pas perdu de vie dernièrement (avec lvlperdu) et si on le touche en étant petit on meurt. Si on le touche en étant grand, on devient petit, on déplace Mario vers le bas car il change de taille et on met lvlperdu à 400 comme ça il ne peut pas perdre de vie pendant un petit moment et comme on a vu en-dessus, il va clignoter. Finalement si le bloc en-dessous est traversable et que le temps de saut est de plus de 1000 on retombe.

1. j = (perso->y + perso->h +1) / Size\_Sprite;
2. **for**(i=xmin;i<=xmax;i++) //Si il y a un bloc en dessous
3. {
4. indicetile = m->LoadedMap[i][j];
5. **if** (S[indicetile].traverser){
6. Mario->G=0;
7. Mario->jump = 1;
8. }
9. **if** (indicetile==10){
10. **if** (Mario->lvlperdu<1)
11. {
12. **if**(Mario->lvl==1)
13. Mario->lose=1;
14. **if**(Mario->lvl==2)
15. audio\_play(6);
16. Mario->lvl=1;
17. Mario->position.y+=10;
18. Mario->lvlperdu=400;
19. }
20. }
21. **else**{
22. **if**(Mario->jumptime>1000)
23. Mario->jump = 0;
24. }
25. }

Ici on regarde s’il y a un bloc en-dessus, si le bloc en-dessus est non traversable on retombe, c’est la même chose que pour le haut de la carte dans la fonction EssaieDeplacement.

Ensuite si le bloc en-dessus est un bloc à champignon et qu’il n’est pas utilisé, on en ajoute un, on met qu’il n’est pas utilisé et on définit sa position, juste en-dessus du bloc. Et on met dans le tableau User que notre bloc a déjà sorti un champignon, on ne pourra donc pas le réutiliser.

1. j = (perso->y - 1) / Size\_Sprite;
2. **for**(i=xmin;i<=xmax;i++) //Si il y a un bloc en dessus
3. {
4. indicetile = m->LoadedMap[i][j];
5. **if** (S[indicetile].traverser)
6. Mario->jumptime = 1001;
7. **if** (indicetile==3){
8. **if**(S[m->LoadedMap[i][j-1]].traverser==0){
9. **if** (m->Used[i][j]==0){
10. shroom->nb\_objet++; //Creation d'un champignon!
11. shroom->used[shroom->nb\_objet]= 0;
12. shroom->position[shroom->nb\_objet].x = i\*Size\_Sprite;
13. shroom->position[shroom->nb\_objet].y = (j-1)\*Size\_Sprite;
14. m->Used[i][j]=1;
15. }
16. }
17. }
18. }

Là on regarde s’il y a un bloc à gauche ou à droite. S’il y en a un, on annule le déplacement (perso->-=vx ;) et on sort de la boucle ( j=ymax+1).

En dernier on regarde si on touche le bloc 8 ou 9 qui est celui du drapeau. Si on le touche on a gagné. Et on retourne dans EssaiDeplacement.

1. i= (perso->x + perso->w) / Size\_Sprite; //Si on touche un bloc Ã  notre droite,
2. **for**(j=ymin;j<=ymax;j++)
3. {
4. indicetile = m->LoadedMap[i][j];
5. **if** (S[indicetile].traverser){
6. perso->x-=vx;
7. j=ymax+1;
8. }
9. }
10. i = (perso->x -1) / Size\_Sprite; //Si on touche un bloc Ã  note gauche
11. **for**(j=ymin;j<=ymax;j++)
12. {
13. indicetile = m->LoadedMap[i][j];
14. **if** (S[indicetile].traverser){
15. perso->x-=vx;
16. j=ymax+1;
17. }
18. }
19. **for**(i=xmin;i<=xmin;i++){
20. **for**(j=ymin;j<=ymax;j++)
21. {
22. indicetile = m->LoadedMap[i][j];
23. **if** (indicetile==8 || indicetile==9){ //Si on touche le drapeau de la fin
24. Mario->win = 1;
25. }
26. }
28. **return** 0; //Si pas de collision

#### NiveauFini (event.c)

Cette fonction regarde si on a gagné ou perdu ou aucun des deux.

D’abord elle regarde si Mario->win vaut 1 si oui, on a gagné le niveau et il va :

* Lancer la musique de niveau réussi.
* Afficher l’image « victory ».
* Attendre qu’on clique sur une touche pour retourner dans le menu principal.

Ensuite si Mario->lose vaut 1 on a perdu et il va :

* Lancer la musique de niveau perdu.
* Afficher l’image «game over ».
* Attendre qu’on clique sur une touche pour retourner dans le menu principal.

Sinon il ne fait rien.

## Amélioration possible

* Utiliser le menu avec la souris.
* Ajout de la gravité.
* Ajouter des sons/niveau.
* Gestion du son.
* Editeur de carte.
* Ajout d’ennemis.
* Changer la résolution de la fenêtre

Bug

* Quand on meurt, le jeu crash de temps en temps.

## Conclusion

Super Mario Bros était un bon choix, c’est un projet faisable pour un débutant en programmation qui demande tout de même pas mal d’apprentissage et de travail. Ça donne de bonnes bases pour continuer dans les jeux-vidéos mais aussi dans le développement en général (manière de coder, approche, langage C...).

Ce projet m’a beaucoup plu, le développement m’intéresse beaucoup, plus encore dans les jeux vidéo. La SDL est un langage intéressant qui fonctionne bien. Ce projet m’a motivé à continuer dans cette voie, je pense faire d’autres jeux et apprendre d’autres langages/librairies dont l’OpenGL comme j’en ai parlé plus haut. Ce qui manque surtout, ce n’est pas la motivation ni la capacité mais les « bonnes » idées.