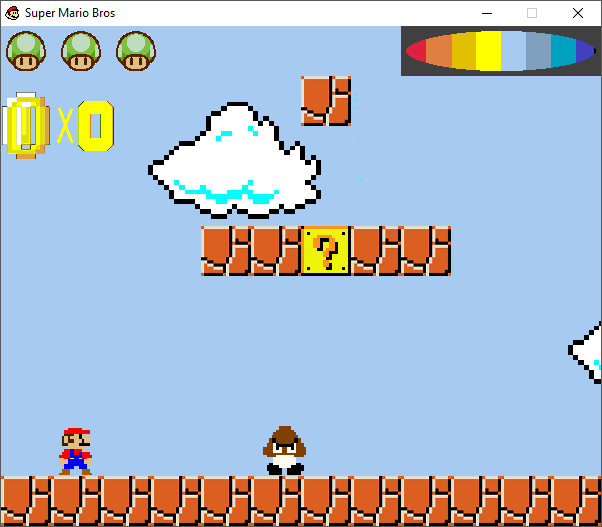
PROJET ISN

Réaliser un jeu de plateforme avec la SDL

Léo LIRZIN, Aymerick MICHELET et Anthony QUERE

Pour de plus amples détails sur notre projet, vous pouvez nous contacter aux adresses mail suivantes :

* [anthony.quere78@orange.fr](mailto:anthony.quere78@orange.fr)
* [leolirzin@gmail.com](mailto:leolirzin@gmail.com)
* [micheletaymerick@gmail.com](mailto:micheletaymerick@gmail.com)

SOMMAIRE

Organisation

* main.c
* **game :**
* conditions.c
* GAME.c
* Init\_game.c
* **joueur :**
* animation.c
* deplacement\_joueur.c
* saut\_joueur.c
* niveau.c
* music.c
* **mechant :**
* contact\_mechant.c
* deplacement\_mechant.c
* refresh\_mechant.c
* bowser.c
* **menu :**
* init\_menu.c
* menu.c

Cahier des charges

Organisation :

Répartition des taches :

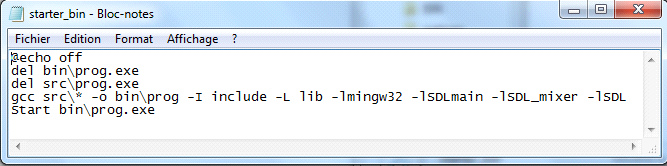
|  |  |  |
| --- | --- | --- |
| LIRZIN Léo | QUERE Anthony | MICHELET Aymerick |
| Graphismes  Timer  Animations  Condition de victoire  Conditions de défaite  Affichage des vies | Interactions  Conditions  Ennemis | Saut  Déplacement  Organisation du programme  Gravité |

|  |
| --- |
| Gadget (fait à deux ou à trois) |
| Vies  Fenêtre (icône, nom, …)  Musique  Et le reste |

Mis en commun du travail :

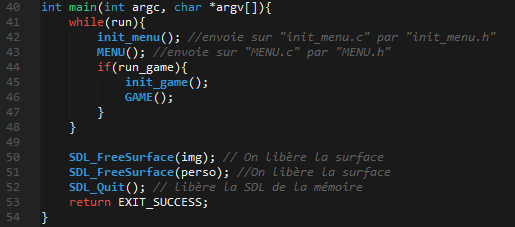
Nous avons créé un dossier partagé sur BOX pour que nous puissions tous accéder au projet depuis chez nous.

Nous avons opté pour la compilation via l’invite de commande que nous avons simplifié avec un fichier codé en BATCH pour simplifier cette étape.



Nos bibliothèques : la SDL (1.2) et la librairie standard

* **main.c :**



Fonction principale de notre programme.

Les SDL\_FreeSurface() servent à libérer les pointeur de surface que nous utilisons.

**Dossier game**

* conditions.c

void conditions()

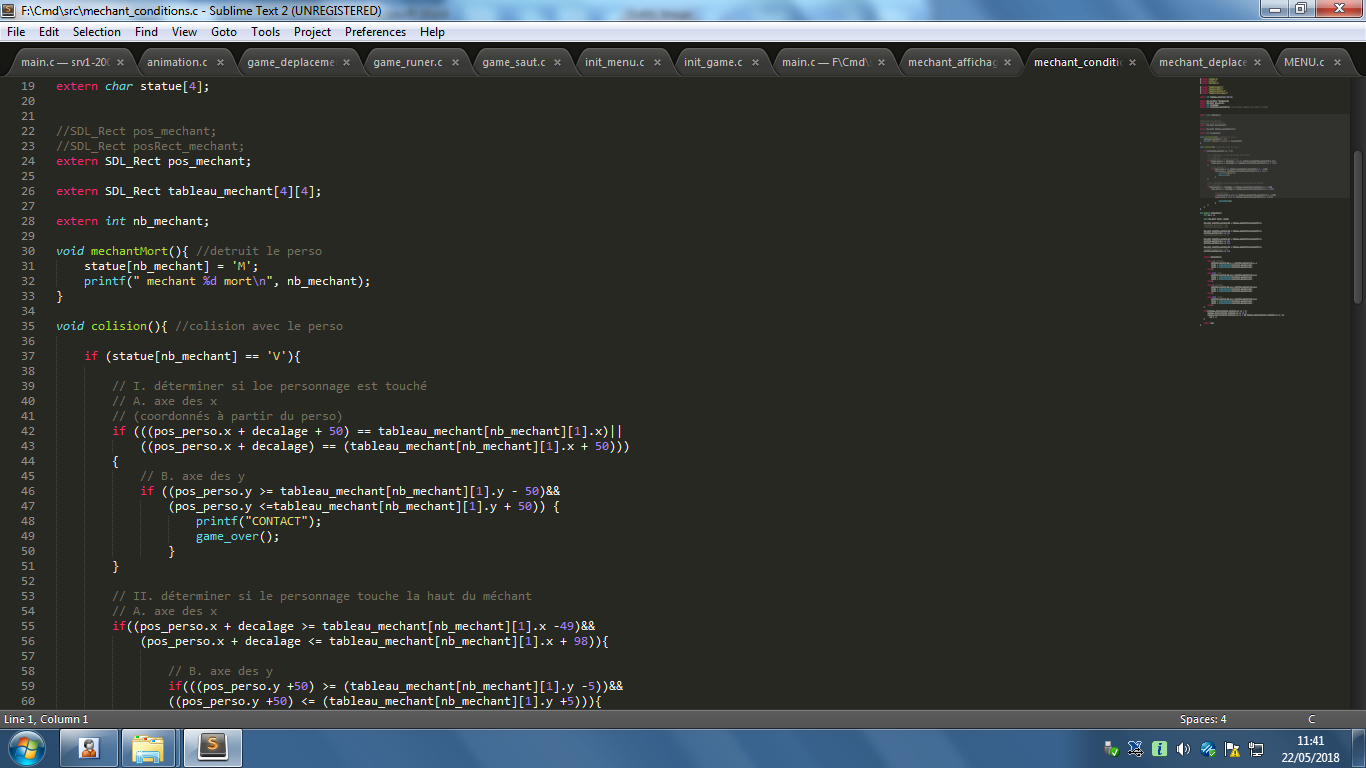
SI le personnage tombe dans un trou, le personnage perd une vie. Si le personnage touche le drapeau, il gagne la partie.

Cette fonction ne prend pas de paramètre.

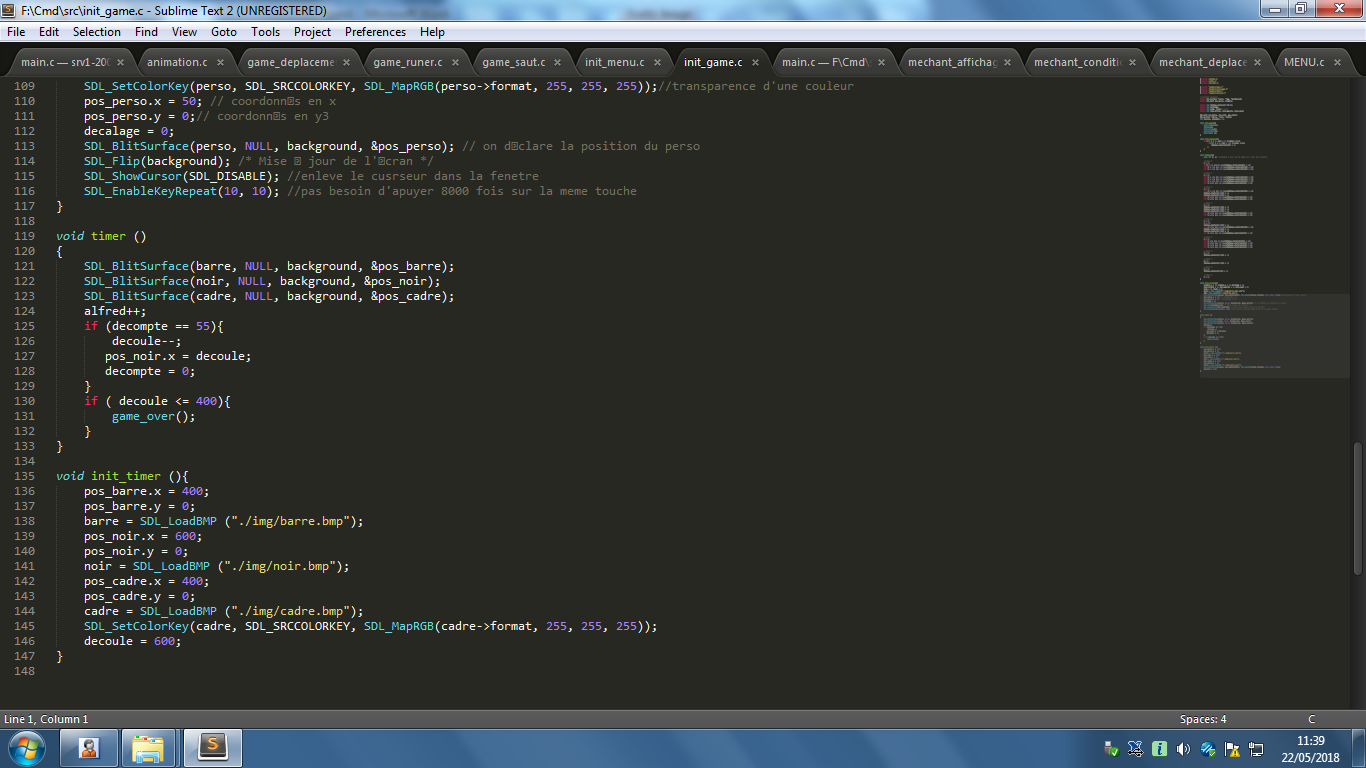
void game\_over()

Il y a 3 façons de mourir :

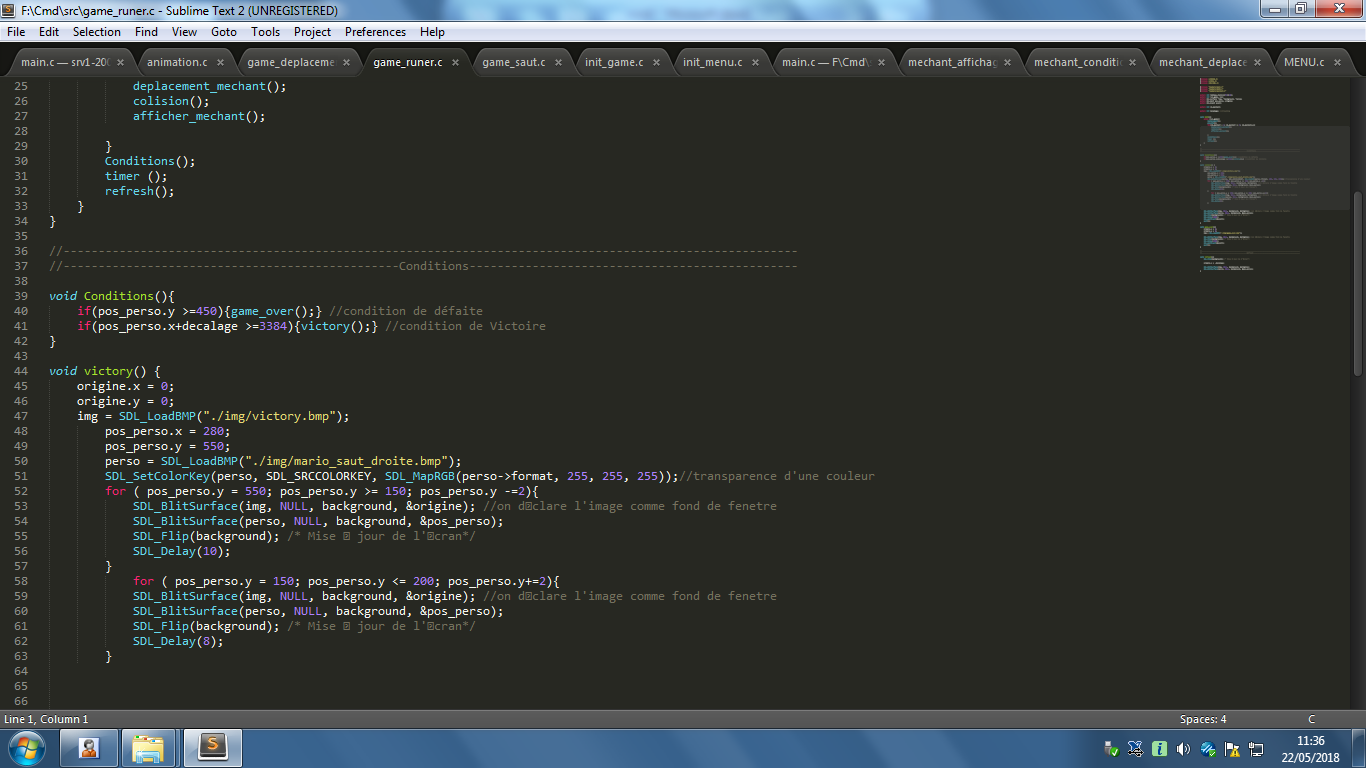
mourir à cause d’un ennemi en cas de collision avec ce dernier Contact\_mechant.C :



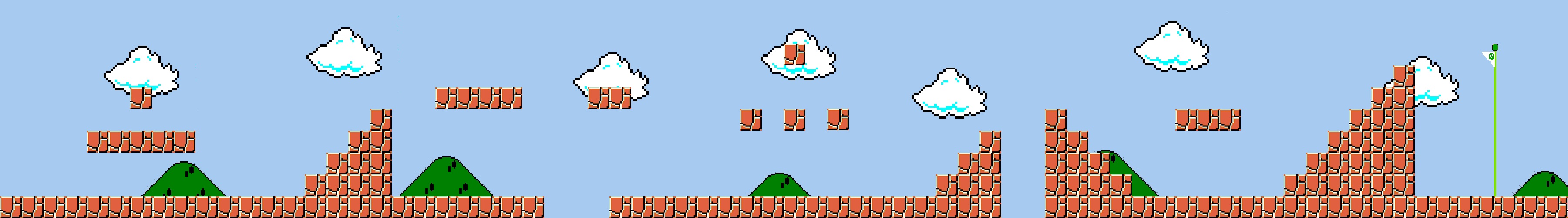
mourir à cause du timer quand ce dernier s’est écoulé (decoule <= 400) GAME.C :



Mourir en tombant dans un trou repartit de part et d’autre du niveau (pos\_perso.y >= 450) Conditions.C :

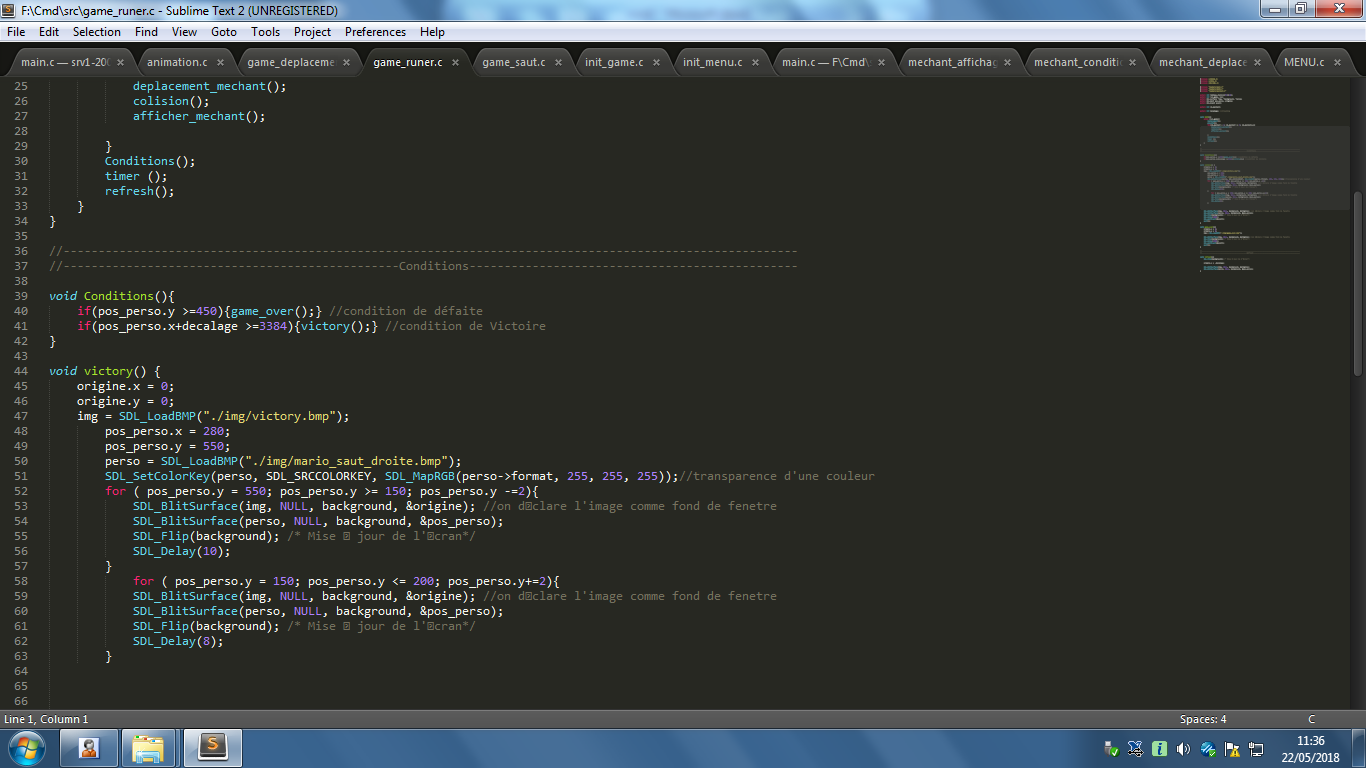


Mourir fera apparaitre l’écran de mort et recommencera le niveau avec une vie en moins. Le jeu, la première fois initialisé, offre trois vies au joueur. Une fois ses vies épuisées, c’est-à-dire être mort trois fois, l’écran de game over s’affichera et le programme cessera de fonctionner.

****Cette fonction ne prend pas de paramètre.

void victory()

Il y a une seule façon de gagner : atteindre les coordonnées du drapeau dans le niveau (pos\_perso.x + decalage >= 3384)

****

****

****Dès lors, l’écran de victoire s’affiche et Mario saute vers le centre de l’écran tout droit en position de saut vers la droite et s’arrête au centre. Le programme cessera ensuite de fonctionner.

Cette fonction ne prend pas de paramètre.

* GAME.c

Init\_coin():

La fonction Init\_coin() va positionner d’une part le compteur de pièce qui s’appelle «coinnb» et d’une autre part l’image «coinx.BMP» indiquant que le compteur affiché est bien celui des pièces. Ces deux derniers sont affichés à coté en bas des vies, montrant leur corrélation. Toutes les 10 pièces, une vie est ajoutée. On enlève ensuite le noir de coinx.BMP pour créer de la transparence.



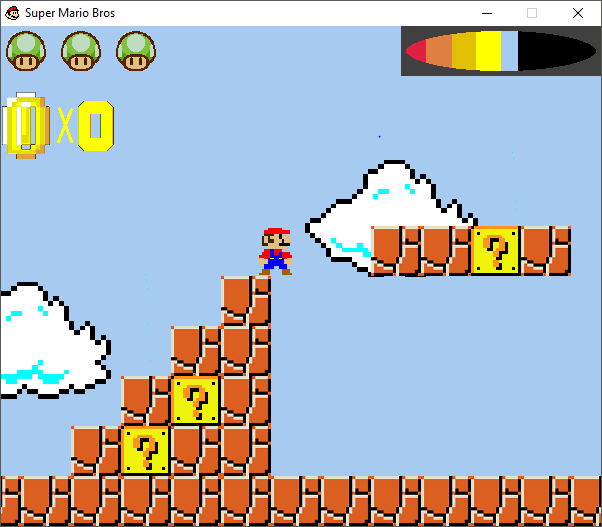
CoinAff():

La fonction CoinAff() va définir l’image affiché au compteur et va aussi attribuer une vie supplémentaire à chaque 10 pièces. La réinitialisation des pièces à partir de 10 pièces est présente autre part que dans cette fonction mais est présente néanmoins. Le «switch» va lire la variable coin, qui contient le nombre de pièce appartenant au joueur, et va attribuer une image d’un chiffre correspondant au nombre de pièce. Le noir de cette image va être ensuite enlever pour créer de la transparence. 10 images représentant les chiffres de 0 à 9 sont attribuées à «coinnb» selon le nombre de pièce. A 10 pièces, coin est remis à 0 et donc le chiffre affiché sera 0.

C:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\9AB4C943.tmpC:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D9BCBDAD.tmpC:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\256E5749.tmpC:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\937CB707.tmpC:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\1CDA33D1.tmpC:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2D734E8B.tmpC:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\697F5DB5.tmpC:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\ED6E93CF.tmpC:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\1DB09F59.tmpC:\Users\Megaport\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\42CA4AD3.tmp

void timer() et init\_timer()

Le timer apparait sous forme de jauge. La forme de jauge est plus intuitive qu’un timer sous forme de chiffre du fait qu’on as pas besoin de regarder le timer pour savoir le temps restant.

Il est constitué d’une barre colorée, d’une barre noire et d’un cadre. Dans le cadre se trouve la barre verte et au fur et à mesure du temps la barre noire va se déplacer vers la gauche en couvrant la barre verte.

Une fois que la barre noire recouvre entièrement la barre verte, le temps est écoulé et la condition de game over est activée.

Le timer dure environ 45 secondes en sachant que le niveau se termine le plus rapidement possible en 20 secondes.

Cette fonction ne prend pas de paramètre.

void systeme\_vie()

Système\_vie() affiche le nombre de vie restante. Cette affichage se fait avec une boucle. La condition est la suivante :

I = 0

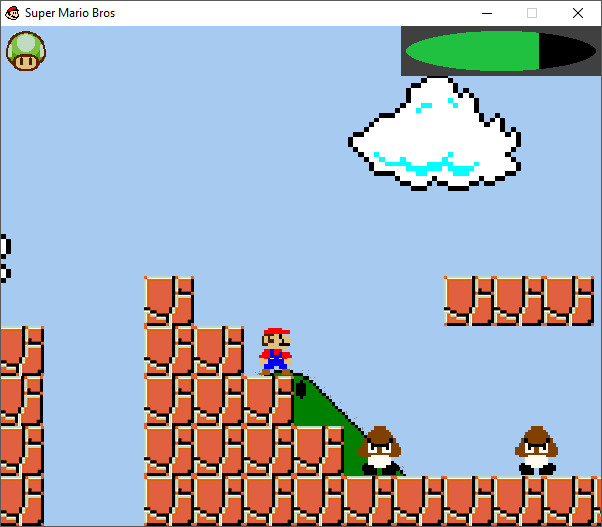
Tant que I <= vie – 1

Position du champignon 1 up prend la valeur de I multiplié par 55 en axe des x

On applique l’image du champignon 1 up

I = I + 1

Fin Tant que

A 3 vies, il y a un champignon 1 up à 0 ; 55 et 110 en axe des x.

A 2 vies, il y aura un champignon 1 up à 0 et 55 en axe des x.

A 1 vie, il y aura seulement un champignon 1 up à 0 en axe des x.

A 0 vie, la condition de défaite est vérifiée et aucun champi n’est donc affiché.

Cette fonction ne prend pas de paramètre.

void refresh()

On affiche dans l’ordre :

* Le fond du niveau
* La barre du temps
* Les méchants
* Le personnage
* Les vies

Puis on utilise « SDL\_Flip(background); » Pour afficher le tout à l’écran.

Cette fonction ne prend pas de paramètre.

void mechant()

Gère le déplacement des méchants un per un en vérifiant pour chacun s’il est en contact avec le joueur.

Cette fonction ne prend pas de paramètre.

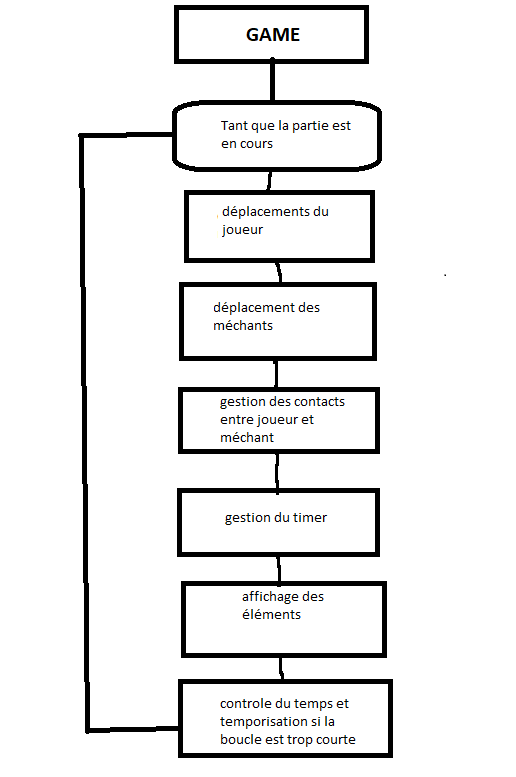
void joueur()

Gère le déplacement du joueur et active ou non la gravité.

Cette fonction ne prend pas de paramètre.

void GAME()

boucle du programme pendant le jeu



Cette fonction ne prend pas de paramètre.

* Niveau.c

Dans ce fichier, on va adapter le tableau lvl à chaque niveau, positionner les méchant et le personnage en début de niveau. Pour ce qui est du tableau du niveau, il est lu sur le « niveau.lvl » contenant le plan de chaque niveau. Ainsi, sans même accéder au code source, il est possible de modifier les niveaux, chaque caractère du tableau correspondant à un bloc.

* Init\_game.c

void init\_mechant()

* On créé un tableau à double entrée de type SDL\_Rect pour stocker les points relatifs aux ennemies :

|  |  |  |  |
| --- | --- | --- | --- |
| N° du méchant | Point : position stricte du méchant | Point : position du méchant sur la grille du niveau | Point : position du méchant avec le décalage compris |
| Méchant n°1 | tbMechant[1][1] | tbMechant[1][2] | tbMechant[1][3] |
| Méchant n°2 | tbMechant[2][1] | tbMechant[2][2] | tbMechant[2][3] |
| Méchant n°3 | tbMechant[3][1] | tbMechant[3][2] | tbMechant[3][3] |

POURQUOI prendre 3 points différents ?

Le premier point correspond à la position absolue du méchant, c'est-à-dire sa position sur le niveau.

Le deuxième point correspond à la position du méchant sur la grille du niveau, de la même manière que le personnage, le niveau est découpé en une grille dans laquelle chaque case correspond à un carré de 50 pixels de côté et à laquelle on attribue une valeur, soit « 1 » si il y a un bloc, soit « 0 » si il y a un vide.

Le troisième point prend en compte le décalage du niveau, sans quoi les méchants trop éloignés ne seraient pas visibles à l’écran.

Les deuxièmes et troisièmes points dépendent du premier.

* De la même manière, on créé un tableau à une entrée de type char donnant le statue du méchant (vivant ou mort)

Cette fonction ne prend pas de paramètre.

void init\_game()

Cette fonction ne prend pas de paramètre.

**Sous dossier joueur**

* animation.c

void animation(void)

On peut créer l’illusion de déplacement et d’animation en appelant au fur et à mesure certaines images à la place du personnage :

F:\Cmd\img\mario_idle.bmpSans déplacement : Mario a dernièrement regardé vers la droite (emp = 0, bool\_droite = 0 ; bool\_gauche = 0 ; test\_bas = 1), il prend cette position :

F:\Cmd\img\mario_idle_gauche.bmpMario a dernièrement regardé vers la gauche (emp = 1, bool\_droite = 0 ; bool\_gauche = 0 ; ), il prend cette position :

F:\Cmd\img\mario_saut_droite.bmpDéplacements en l’air  : Mario regarde ou (bool\_droite =1 et bool\_gauche = 0) /et a dernièrement regardé vers la droite ( emp = 0) tout en ne touchant pas le sol (test\_bas = 0) , il prend cette position :

Mario regarde (bool\_droite =0 et bool\_gauche = 0) ou/et a dernièrement regardé vers la gauche (emp = 1) tout en ne touchant pas le sol (test\_bas = 0) , il prend cette position :

F:\Cmd\img\mario_saut_gauche.bmp

Déplacements au sol : Mario avance vers la droite ( bool\_droite = 1 ; bool\_gauche = 0 ; test\_bas = 1) ; , on change d’image toutes les 0.1 secondes de droite vers la gauche :

F:\Cmd\img\frame_3_droite.bmpF:\Cmd\img\frame_2_droite.bmpF:\Cmd\img\frame_1_droite.bmp

Mario avance vers la gauche( bool\_droite = 0 ; bool\_gauche = 1; test\_bas = 1) , on change d’image toutes les 0.1 secondes de droite vers la gauche :

F:\Cmd\img\frame_3_gauche.bmpF:\Cmd\img\frame_2_gauche.bmpF:\Cmd\img\frame_1_gauche.bmp

Cette fonction ne prend pas de paramètre.

deplacement\_joueur.c

SDL\_Rect set\_position(SDL\_Rect Point)

On divise les coordonnés du point pas 50 (taille d'un bloc) et on les renvois sous la forme d'un autre point, ce qui permet de placer le méchant sur la grille du niveau (voire fonction go/goM).

Cette fonction ne prend pas de paramètre.

void droite()

SI il n'y a pas d'obstacle à droite :

SI le personnage a atteint le milieu de l'écran :

on incrémente la position en x du strawling

SINON

on incrémente la position en x du personnage

Cette fonction ne prend pas de paramètre.

void gauche()

SI il n'y a pas d'obstacle à gauche :

SI le personnage a atteint le premier quart de l'écran de l'écran :

on décrémente la position en x du personnage

SINON

on décrémente la position en x du personnage

Cette fonction ne prend pas de paramètre.

void deplacement()

permet de recueillir les commandes de l’utilisateur et d’agir en conséquence. Il va par exemple rediriger vers droite(), gauche(), ou statue\_saut().

Cette fonction ne prend pas de paramètre.

void deplacement\_joueur()

a un rôle de main local. C’est-à-dire qu’il structure les différentes fonctions liées au déplacement en suivant un ordre d’exécution et de condition spécifique.

Cette fonction ne prend pas de paramètre.

int go(char direction)

Le niveau est fait de tel sorte que l’on peut le découper en une grille de 71 par 10. Chaque case est soit égale à 1 sur il y a un bloc, soit à 0 s’il n’y a pas de bloc. Il ne peut pas y avoir à la fois un bloc et un vide sur la même case.

Comment savoir si le personnage est en contact avec un bloc ?

Prenons l’exemple du vide : Au début du niveau, le personnage se trouve sur un bloc qui l’empêche de tomber.

On va tester la nature du bloc en dessous à gauche et à droite du personnage, si au moins un des deux est à l'état 1, la fonction renvoit 1.

On utilise deux fonctions distinctes, go() et goM() car la premiere se base sur la position du personnage et la deuxième sur les positions des méchants.

Les déplacements du personnage :

On récupère la direction du personnage choisie par l’utilisateur via le clavier puis on teste pour savoir si il y a un obstacle dans cette direction et si il n’y en a pas, le personnage avance.

* saut\_joueur.c

double fonction(double i)

Retourne le résultat à l’équation suivante : -i\*i+22\*i. Elle est utilisée pour donner les valeurs que prend le perso lors d’un saut (excepter saut « sur méchant ») ou d’une chute (excepter chute après saut « sur méchant »).

double fonction2(double i)

Retourne le résultat à l’équation suivante : -i\*i+15\*i. Elle est utilisée pour donner les valeurs que prend le perso lors d’un saut (saut « sur méchant ») ou d’une chute (chute après « saut sur méchant »).

void statue\_saut() et void saut()

Permet de déterminer l’aptitude du joueur à sauter et si oui, d’initialiser les paramètres nécessaire selon la nature du saut (saut « normal » ou saut « sur méchant »). Permet également d’éviter le spam de la touche espace.

Ces fonctions ne prennent pas de paramètre.

void gravite()

Permet de faire varier la hauteur de perso par rapport à l’incrémentation de la variable double x à la fonction mathématique utilisé ( fonction() ou fonction2() ), pour que sa chute réponde au mieux aux lois de la physique. Réinitialise dès que l’incrémentation arrive à sa fin.

Cette fonction ne prend pas de paramètre.

**Sous dossier mechant**

* contact\_mechant.c

int goM(int direction)

voir fonction go().

void mechantMort()

Change le statut du méchant de 'V' à 'M' pour que le méchant s'arrête de bouger et que les contacts avec le personnage s'arrêtent.

void contact\_lateral()

On veut que la position en x du bord droit du personnage soit égale à celle du bord gauche du méchant et vis versa. Pareil pour l'axe des y. Puis le personnage perd une vie.

Cette fonction ne prend pas de paramètre.

void contact\_vertical()

On veut que le bord en bas du personnage rentre en contact avec le bord supérieur du méchant. Ensuite, le méchant meurt et ne pourra plus enlever de vie au personnage.

Cette fonction ne prend pas de paramètre.

void contact()

Appel les fonctions contact\_lateral() et contact\_vertical() pour gérer les collisions personnage-joueur

Les conditions se font à partir des coordonnés du méchant et du personnage et non pas à partir des positions sur la grille car le personnage et le méchant peuvent être sur la même case sans entrer en contact.

Cette fonction ne prend pas de paramètre.

* deplacement\_mechant.c

SDL\_Rect setRectMechant(SDL\_Rect Point)

On divise les coordonnés du point pas 50 (taille d'un bloc) et on les renvois sous la forme d'un autre point, ce qui permet de placer le méchant sur la grille du niveau (voire fonction go/goM).

void deplacement\_mechant()

La direction du mechant est soit de 1 (droite) soit de -1 (gauche)

S’il y a un obstacle

Le méchant change de direction (direction = - direction)

SINON

Position du méchant en x += direction

Cette fonction ne prend pas de paramètre.

* refresh\_mechant.c

void afficher\_mechant()

On affiche le méchant en se basant sur le troisième point.

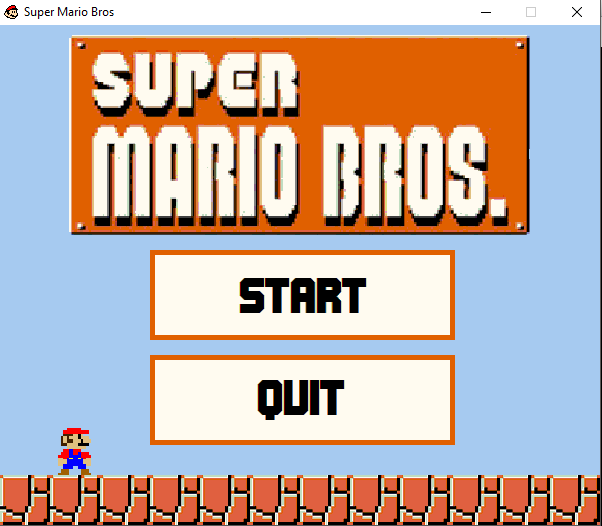
Cette fonction ne prend pas de paramètre.

* Le bowser.c

Ce dossier amène un nouveau type d’ennemie plus gros sur lequel il faudra sauter plusieurs fois afin de le battre. Ce fichier est donc une deuxième version des autres fonctions pour les « goombas » mais cette fois ci adapté à ce nouveau personnage présent au niveau 4

**Dossier menu**

* init\_menu.c



On créé un fenêtre appelée « Super Mario Bros », on lui donne une icone, une surface de fond appelée « background »

* MENU.c

int start\_game()

On attend un événement de la part du joueur puis on en test la nature, soit c’est un clic gauche, dans ce cas on regarde la position de ce clic, s’il est sur le bouton START, on lance le niveau, si il est sur le bouton QUIT, on ferme la fenêtre. De même si le joueur appui sur la croix en haut à droite, la fenêtre se ferme aussi. Il a fallu rajouter ce cas au code car la SDL permet de donner d’autres utilisations à ce bouton.

* La musique

Le contenue lié à la musique est placé dans le fichier « music.c ». Ce n’est là qu’une simplification de la librairie SDL\_mixer 1.2. Ce fichier nous permet de jouer de la musique et des sons dans notre projet simplement en appelant les fonctions playSon() et playMus() prenant comme paramètre le numéro du son ou de la musique voulu. Les sons sont répétés 1 fois chacun et les musiques le sont en boucle à l’exception de celles pour les game over et la victoire.

**Objectif du projet**

Ecrire un programme qui simule le jeu MARIO

**Travail demandé**

Vous devez vous partager les tâches au sein de l’équipe pour élaborer un programme qui :

* Affiche une page d’accueil ( Start , Quit..)
* Affiche le décor
* Affiche les obstacles (blocs et pièges)
* Affiche le personnage en position de départ
* Permet au joueur de déplacer MARIO dans le décor (défilement horizontal du décor + sauts etc).
* Affiche un chronomètre
* Retour au début du niveau et suppression d’une vie si MARIO tombe sur un piège ou si Chronomètre=0
* Affiche Game Over si plus de vie puis retour au menu

Option : 2 ou 3 niveaux, pièces à collecter pour obtenir des vies, sons

**Production élève attendue**

Le programme du jeu MARIO ( ne pas oublier les commentaires).

Chaque élève doit élaborer un dossier personnel comprenant :

* Un rappel du projet
* Les règles du jeu que vous avez choisies
* Un exposé rapide des travaux effectués par les autres membres de l’équipe
* Un exposé des ses travaux personnels :
  + Un algorithme ( ou algorigramme) associé au programme principal
  + Le rôle des fonctions élaborées par l’élève et des variables utilisées.
  + Les algorithmes (ou algorigrammes) associés à ces fonctions
* Une conclusion contenant par exemple les améliorations envisageables.

Chaque élève doit préparer une présentation (type powerpoint) pour l’oral.

.