

## My Project

Generated by Doxygen 1.9.1



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 sprite_s Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 h	5
3.1.2.2 w	5
3.1.2.3 x	6
3.1.2.4 y	6
3.2 textures_s Struct Reference	6
3.2.1 Detailed Description	6
3.2.2 Member Data Documentation	6
3.2.2.1 background	6
3.2.2.2 ligne_arrive	7
3.2.2.3 meteorite	7
3.2.2.4 vaisseau	7
3.3 world_s Struct Reference	7
3.3.1 Detailed Description	7
3.3.2 Member Data Documentation	7
3.3.2.1 collision_mur	8
3.3.2.2 gameover	8
3.3.2.3 ligne_arrive	8
3.3.2.4 mur	8
3.3.2.5 vaisseau	8
<b>4 File Documentation</b>	<b>9</b>
4.1 definition.h File Reference	9
4.1.1 Detailed Description	9
4.2 graphique.c File Reference	10
4.2.1 Detailed Description	10
4.2.2 Function Documentation	11
4.2.2.1 apply_background()	11
4.2.2.2 apply_meteorite()	11
4.2.2.3 apply_sprite()	11
4.2.2.4 clean()	12
4.2.2.5 clean_textures()	12
4.2.2.6 init()	12
4.2.2.7 init_textures()	13

4.2.2.8 refresh_graphics()	13
4.3 graphique.h File Reference	13
4.3.1 Detailed Description	14
4.3.2 Function Documentation	15
4.3.2.1 apply_background()	15
4.3.2.2 apply_meteorite()	15
4.3.2.3 apply_sprite()	15
4.3.2.4 clean()	16
4.3.2.5 clean_textures()	16
4.3.2.6 init()	16
4.3.2.7 init_textures()	17
4.3.2.8 refresh_graphics()	17
4.4 logique.c File Reference	17
4.4.1 Detailed Description	18
4.4.2 Function Documentation	18
4.4.2.1 abs()	18
4.4.2.2 clean_data()	19
4.4.2.3 depassement_d()	19
4.4.2.4 depassement_g()	19
4.4.2.5 handle_sprites_collision()	20
4.4.2.6 init_data()	20
4.4.2.7 init_sprite()	20
4.4.2.8 is_game_over()	21
4.4.2.9 print_sprite()	21
4.4.2.10 sprites_collide()	21
4.4.2.11 update_data()	22
4.5 logique.h File Reference	22
4.5.1 Detailed Description	23
4.5.2 Function Documentation	23
4.5.2.1 abs()	23
4.5.2.2 clean_data()	24
4.5.2.3 depassement_d()	24
4.5.2.4 depassement_g()	24
4.5.2.5 handle_sprites_collision()	24
4.5.2.6 init_data()	25
4.5.2.7 init_sprite()	25
4.5.2.8 is_game_over()	25
4.5.2.9 print_sprite()	26
4.5.2.10 sprites_collide()	26
4.5.2.11 update_data()	26
4.6 main.c File Reference	27
4.6.1 Detailed Description	27

4.6.2 Function Documentation	27
4.6.2.1 handle_events()	27
4.7 sdl2-light.c File Reference	28
4.7.1 Detailed Description	28
4.7.2 Function Documentation	28
4.7.2.1 apply_texture()	29
4.7.2.2 clean_sdl()	29
4.7.2.3 clean_texture()	29
4.7.2.4 clear_renderer()	30
4.7.2.5 init_sdl()	30
4.7.2.6 load_image()	30
4.7.2.7 pause()	31
4.7.2.8 update_screen()	31
4.8 sdl2-light.h File Reference	31
4.8.1 Detailed Description	32
4.8.2 Function Documentation	32
4.8.2.1 apply_texture()	32
4.8.2.2 clean_sdl()	33
4.8.2.3 clean_texture()	33
4.8.2.4 clear_renderer()	33
4.8.2.5 init_sdl()	33
4.8.2.6 load_image()	34
4.8.2.7 pause()	34
4.8.2.8 update_screen()	35
4.9 tests.c File Reference	35
4.9.1 Detailed Description	35
<b>Index</b>	<b>37</b>



# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">sprite_s</a>	Représentation d'un sprite du jeu . . . . .	<a href="#">5</a>
<a href="#">textures_s</a>	Représentation pour stocker les textures nécessaires à l'affichage graphique . . . . .	<a href="#">6</a>
<a href="#">world_s</a>	Représentation du monde du jeu . . . . .	<a href="#">7</a>





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">definition.h</a>	Header regroupant toutes les constantes utiles au programme . . . . .	9
<a href="#">graphique.c</a>	Module gérant la partie graphique du jeu . . . . .	10
<a href="#">graphique.h</a>	Header du module graphique . . . . .	13
<a href="#">logique.c</a>	Module gérant la partie logique du jeu . . . . .	17
<a href="#">logique.h</a>	Header du module logique . . . . .	22
<a href="#">main.c</a>	Programme principal initial du niveau 2 . . . . .	27
<a href="#">sdl2-light.c</a>	Sur-couche de SDL2 pour simplifier son utilisation pour le projet . . . . .	28
<a href="#">sdl2-light.h</a>	En-tête du module correspondant à une sur-couche de SDL2 pour simplifier son utilisation pour le projet . . . . .	31
<a href="#">tests.c</a>	Programme testant la partie logique du jeu . . . . .	35



## Chapter 3

# Class Documentation

### 3.1 sprite\_s Struct Reference

Représentation d'un sprite du jeu.

```
#include <logique.h>
```

#### Public Attributes

- int [x](#)
- int [y](#)
- int [h](#)
- int [w](#)

#### 3.1.1 Detailed Description

Représentation d'un sprite du jeu.

#### 3.1.2 Member Data Documentation

##### 3.1.2.1 h

```
int sprite_s::h
```

Champ indiquant la hauteur

##### 3.1.2.2 w

```
int sprite_s::w
```

Champ indiquant la largeur

### 3.1.2.3 x

```
int sprite_s::x
```

Champ indiquant l'abscisse de la position

### 3.1.2.4 y

```
int sprite_s::y
```

Champ indiquant l'ordonnée de la position

The documentation for this struct was generated from the following file:

- [logique.h](#)

## 3.2 textures\_s Struct Reference

Représentation pour stocker les textures nécessaires à l'affichage graphique.

```
#include <graphique.h>
```

### Public Attributes

- `SDL_Texture *` [background](#)
- `SDL_Texture *` [vaisseau](#)
- `SDL_Texture *` [ligne\\_arrive](#)
- `SDL_Texture *` [meteorite](#)

### 3.2.1 Detailed Description

Représentation pour stocker les textures nécessaires à l'affichage graphique.

### 3.2.2 Member Data Documentation

#### 3.2.2.1 background

```
SDL_Texture* textures_s::background
```

Texture liée à l'image du fond de l'écran.

### 3.2.2.2 ligne\_arrive

```
SDL_Texture* textures_s::ligne_arrive
```

Texture liée à la ligne d'arrivée

### 3.2.2.3 meteorite

```
SDL_Texture* textures_s::meteorite
```

Texture liée à une météorite

### 3.2.2.4 vaisseau

```
SDL_Texture* textures_s::vaisseau
```

Texture liée à l'image du vaisseau.

The documentation for this struct was generated from the following file:

- [graphique.h](#)

## 3.3 world\_s Struct Reference

Représentation du monde du jeu.

```
#include <logique.h>
```

### Public Attributes

- [sprite\\_t vaisseau](#)
- [sprite\\_t ligne\\_arrive](#)
- [sprite\\_t mur](#)
- int [vy](#)
- int [gameover](#)
- int [collision\\_mur](#)

### 3.3.1 Detailed Description

Représentation du monde du jeu.

### 3.3.2 Member Data Documentation

### 3.3.2.1 collision\_mur

```
int world_s::collision_mur
```

Champ indiquant si le vaisseau est rentré en colision avec un mur

### 3.3.2.2 gameover

```
int world_s::gameover
```

Champ représentant les vitesse du jeu Champ indiquant si l'on est à la fin du jeu

### 3.3.2.3 ligne\_arrive

```
sprite_t world_s::ligne_arrive
```

Champ représentant la ligne d'arrivée

### 3.3.2.4 mur

```
sprite_t world_s::mur
```

Champ représentant un mur de météorites

### 3.3.2.5 vaisseau

```
sprite_t world_s::vaisseau
```

Champ représentant le sprite vaisseau

The documentation for this struct was generated from the following file:

- [logique.h](#)

## Chapter 4

# File Documentation

### 4.1 definition.h File Reference

Header regroupant toutes les constantes utiles au programme.

#### Macros

- #define `SCREEN_WIDTH` 300  
*Largeur de l'écran de jeu.*
- #define `SCREEN_HEIGHT` 480  
*Hauteur de l'écran de jeu.*
- #define `SHIP_SIZE` 32  
*Taille d'un vaisseau.*
- #define `METEORITE_SIZE` 32  
*Taille d'un météorite.*
- #define `FINISH_LINE_HEIGHT` 10  
*Hauteur de la ligne d'arrivée.*
- #define `MOVING_STEP` 10  
*Pas de déplacement horizontal du vaisseau.*
- #define `INITIAL_SPEED` 2  
*Vitesse initiale de déplacement vertical des éléments du jeu.*

#### 4.1.1 Detailed Description

Header regroupant toutes les constantes utiles au programme.

#### Author

LESNIAK Louis & SLIMANI Kamelia

#### Version

2.0

#### Date

14 avril 2021

## 4.2 graphique.c File Reference

Module gérant la partie graphique du jeu.

```
#include "sdl2-light.h"
#include "graphique.h"
#include "logique.h"
#include "definition.h"
#include <stdlib.h>
#include <stdio.h>
```

### Functions

- void [clean\\_textures](#) ([textures\\_t](#) \*textures)  
*La fonction nettoie les textures.*
- void [init\\_textures](#) (SDL\_Renderer \*renderer, [textures\\_t](#) \*textures)  
*La fonction initialise les textures nécessaires à l'affichage graphique du jeu.*
- void [apply\\_sprite](#) (SDL\_Renderer \*renderer, SDL\_Texture \*texture, [sprite\\_t](#) \*sprite, int make\_disappear)  
*La fonction initialise la position du sprite.*
- void [apply\\_background](#) (SDL\_Renderer \*renderer, SDL\_Texture \*texture)  
*La fonction applique la texture du fond sur le renderer lié à l'écran de jeu.*
- void [apply\\_meteorite](#) (SDL\_Renderer \*renderer, [world\\_t](#) \*world, [textures\\_t](#) \*textures)  
*La fonction applique la texture des météorites sur le mur.*
- void [refresh\\_graphics](#) (SDL\_Renderer \*renderer, [world\\_t](#) \*world, [textures\\_t](#) \*textures)  
*La fonction rafraichit l'écran en fonction de l'état des données du monde.*
- void [clean](#) (SDL\_Window \*window, SDL\_Renderer \*renderer, [textures\\_t](#) \*textures, [world\\_t](#) \*world)  
*fonction qui nettoie le jeu: nettoyage de la partie graphique (SDL), nettoyage des textures, nettoyage des données*
- void [init](#) (SDL\_Window \*\*window, SDL\_Renderer \*\*renderer, [textures\\_t](#) \*textures, [world\\_t](#) \*world)  
*fonction qui initialise le jeu: initialisation de la partie graphique (SDL), chargement des textures, initialisation des données*

### 4.2.1 Detailed Description

Module gérant la partie graphique du jeu.

#### Author

LESNIAK Louis & SLIMANI Kamelia

#### Version

2.0

#### Date

14 avril 2021



## 4.2.2 Function Documentation

### 4.2.2.1 apply\_background()

```
void apply_background (
    SDL_Renderer * renderer,
    SDL_Texture * texture )
```

La fonction applique la texture du fond sur le renderer lié à l'écran de jeu.

#### Parameters

<i>renderer</i>	le renderer
<i>texture</i>	la texture liée au fond
<i>sprite</i>	sprite
<i>make_disappear</i>	permet de faire disparaître ou non le sprite

### 4.2.2.2 apply\_meteorite()

```
void apply_meteorite (
    SDL_Renderer * renderer,
    world_t * world,
    textures_t * textures )
```

La fonction applique la texture des météorites sur le mur.

#### Parameters

<i>renderer</i>	le renderer lié à l'écran de jeu
<i>world</i>	les données du monde
<i>textures</i>	les textures

### 4.2.2.3 apply\_sprite()

```
void apply_sprite (
    SDL_Renderer * renderer,
    SDL_Texture * texture,
    sprite_t * sprite,
    int make_disappear )
```

La fonction initialise la position du sprite.

## Parameters

<i>sprite</i>	vaisseau
<i>x</i>	abscisse de la position du sprite
<i>y</i>	ordonnée de la position du sprite
<i>w</i>	largeur de la position du sprite
<i>h</i>	hauteur de la position du sprite

**4.2.2.4 clean()**

```
void clean (
    SDL_Window * window,
    SDL_Renderer * renderer,
    textures_t * textures,
    world_t * world )
```

fonction qui nettoie le jeu: nettoyage de la partie graphique (SDL), nettoyage des textures, nettoyage des données

## Parameters

<i>window</i>	la fenêtre du jeu
<i>renderer</i>	le renderer
<i>textures</i>	les textures
<i>world</i>	le monde

**4.2.2.5 clean\_textures()**

```
void clean_textures (
    textures_t * textures )
```

La fonction nettoie les textures.

## Parameters

<i>textures</i>	les textures
-----------------	--------------

**4.2.2.6 init()**

```
void init (
    SDL_Window ** window,
    SDL_Renderer ** renderer,
```

```
textures_t * textures,  
world_t * world )
```

fonction qui initialise le jeu: initialisation de la partie graphique (SDL), chargement des textures, initialisation des données

#### Parameters

<i>window</i>	la fenêtre du jeu
<i>renderer</i>	le renderer
<i>textures</i>	les textures
<i>world</i>	le monde

#### 4.2.2.7 init\_textures()

```
void init_textures (  
    SDL_Renderer * renderer,  
    textures_t * textures )
```

La fonction initialise les textures nécessaires à l'affichage graphique du jeu.

#### Parameters

<i>screen</i>	la surface correspondant à l'écran de jeu
<i>textures</i>	les textures du jeu

#### 4.2.2.8 refresh\_graphics()

```
void refresh_graphics (  
    SDL_Renderer * renderer,  
    world_t * world,  
    textures_t * textures )
```

La fonction rafraichit l'écran en fonction de l'état des données du monde.

#### Parameters

<i>renderer</i>	le renderer lié à l'écran de jeu
<i>world</i>	les données du monde
<i>textures</i>	les textures

## 4.3 graphique.h File Reference

Header du module graphique.

```
#include "sdl2-light.h"
#include "logique.h"
```

## Classes

- struct [textures\\_s](#)  
*Représentation pour stocker les textures nécessaires à l'affichage graphique.*

## Typedefs

- typedef struct [textures\\_s](#) [textures\\_t](#)  
*Type qui correspond aux textures du jeu.*

## Functions

- void [clean\\_textures](#) ([textures\\_t](#) \*textures)  
*La fonction nettoie les textures.*
- void [init\\_textures](#) (SDL\_Renderer \*renderer, [textures\\_t](#) \*textures)  
*La fonction initialise les textures nécessaires à l'affichage graphique du jeu.*
- void [apply\\_sprite](#) (SDL\_Renderer \*renderer, SDL\_Texture \*texture, [sprite\\_t](#) \*sprite, int make\_disappear)  
*La fonction initialise la position du sprite.*
- void [apply\\_background](#) (SDL\_Renderer \*renderer, SDL\_Texture \*texture)  
*La fonction applique la texture du fond sur le renderer lié à l'écran de jeu.*
- void [apply\\_meteorite](#) (SDL\_Renderer \*renderer, [world\\_t](#) \*world, [textures\\_t](#) \*textures)  
*La fonction applique la texture des météorites sur le mur.*
- void [refresh\\_graphics](#) (SDL\_Renderer \*renderer, [world\\_t](#) \*world, [textures\\_t](#) \*textures)  
*La fonction rafraichit l'écran en fonction de l'état des données du monde.*
- void [clean](#) (SDL\_Window \*window, SDL\_Renderer \*renderer, [textures\\_t](#) \*textures, [world\\_t](#) \*world)  
*fonction qui nettoie le jeu: nettoyage de la partie graphique (SDL), nettoyage des textures, nettoyage des données*
- void [init](#) (SDL\_Window \*\*window, SDL\_Renderer \*\*renderer, [textures\\_t](#) \*textures, [world\\_t](#) \*world)  
*fonction qui initialise le jeu: initialisation de la partie graphique (SDL), chargement des textures, initialisation des données*

### 4.3.1 Detailed Description

Header du module graphique.

#### Author

LESNIAK Louis & SLIMANI Kamelia

#### Version

2.0

#### Date

14 avril 2021

## 4.3.2 Function Documentation

### 4.3.2.1 apply\_background()

```
void apply_background (
    SDL_Renderer * renderer,
    SDL_Texture * texture )
```

La fonction applique la texture du fond sur le renderer lié à l'écran de jeu.

#### Parameters

<i>renderer</i>	le renderer
<i>texture</i>	la texture liée au fond
<i>sprite</i>	sprite
<i>make_disappear</i>	permet de faire disparaître ou non le sprite

### 4.3.2.2 apply\_meteorite()

```
void apply_meteorite (
    SDL_Renderer * renderer,
    world_t * world,
    textures_t * textures )
```

La fonction applique la texture des météorites sur le mur.

#### Parameters

<i>renderer</i>	le renderer lié à l'écran de jeu
<i>world</i>	les données du monde
<i>textures</i>	les textures

### 4.3.2.3 apply\_sprite()

```
void apply_sprite (
    SDL_Renderer * renderer,
    SDL_Texture * texture,
    sprite_t * sprite,
    int make_disappear )
```

La fonction initialise la position du sprite.

## Parameters

<i>sprite</i>	vaisseau
<i>x</i>	abscisse de la position du sprite
<i>y</i>	ordonnée de la position du sprite
<i>w</i>	largeur de la position du sprite
<i>h</i>	hauteur de la position du sprite

**4.3.2.4 clean()**

```
void clean (
    SDL_Window * window,
    SDL_Renderer * renderer,
    textures_t * textures,
    world_t * world )
```

fonction qui nettoie le jeu: nettoyage de la partie graphique (SDL), nettoyage des textures, nettoyage des données

## Parameters

<i>window</i>	la fenêtre du jeu
<i>renderer</i>	le renderer
<i>textures</i>	les textures
<i>world</i>	le monde

**4.3.2.5 clean\_textures()**

```
void clean_textures (
    textures_t * textures )
```

La fonction nettoie les textures.

## Parameters

<i>textures</i>	les textures
-----------------	--------------

**4.3.2.6 init()**

```
void init (
    SDL_Window ** window,
    SDL_Renderer ** renderer,
```

```
textures_t * textures,  
world_t * world )
```

fonction qui initialise le jeu: initialisation de la partie graphique (SDL), chargement des textures, initialisation des données

#### Parameters

<i>window</i>	la fenêtre du jeu
<i>renderer</i>	le renderer
<i>textures</i>	les textures
<i>world</i>	le monde

#### 4.3.2.7 init\_textures()

```
void init_textures (  
    SDL_Renderer * renderer,  
    textures_t * textures )
```

La fonction initialise les textures nécessaires à l'affichage graphique du jeu.

#### Parameters

<i>screen</i>	la surface correspondant à l'écran de jeu
<i>textures</i>	les textures du jeu

#### 4.3.2.8 refresh\_graphics()

```
void refresh_graphics (  
    SDL_Renderer * renderer,  
    world_t * world,  
    textures_t * textures )
```

La fonction rafraichit l'écran en fonction de l'état des données du monde.

#### Parameters

<i>renderer</i>	le renderer lié à l'écran de jeu
<i>world</i>	les données du monde
<i>textures</i>	les textures

## 4.4 logique.c File Reference

Module gérant la partie logique du jeu.

```
#include "logique.h"
#include "definition.h"
#include <stdio.h>
#include <stdlib.h>
```

## Functions

- void `init_sprite` (`sprite_t` \*sprite, int x, int y, int w, int h)  
*La fonction initialise la position du sprite.*
- void `init_data` (`world_t` \*world)  
*La fonction initialise les données du monde du jeu.*
- void `print_sprite` (`sprite_t` sprite)  
*La fonction affiche la position du sprite.*
- void `clean_data` (`world_t` \*world)  
*La fonction nettoie les données du monde.*
- int `is_game_over` (`world_t` \*world)  
*La fonction indique si le jeu est fini en fonction des données du monde.*
- void `update_data` (`world_t` \*world)  
*La fonction met à jour les données en tenant compte de la physique du monde.*
- void `depassement_g` (`sprite_t` \*sprite)  
*fonction qui verifie que le sprite ne depaase pas la limite gauche du jeu*
- void `depassement_d` (`sprite_t` \*sprite)  
*fonction qui verifie que le sprite ne depaase pas la limite droite du jeu*
- int `abs` (int a)  
*fonction qui retourne la valeur absolue d'un entier*
- int `sprites_collide` (`sprite_t` \*sp1, `sprite_t` \*sp2)  
*fonction qui verifie que 2 sprite ne soit pas en collision*
- void `handle_sprites_collision` (`sprite_t` \*sp1, `sprite_t` \*sp2, `world_t` \*world, int make\_disappear)  
*fonction qui verifie que 2 sprite ne soit pas en collision*

### 4.4.1 Detailed Description

Module gérant la partie logique du jeu.

#### Author

LESNIAK Louis & SLIMANI Kamelia

#### Version

2.0

#### Date

14 avril 2021

### 4.4.2 Function Documentation

#### 4.4.2.1 `abs()`

```
int abs (  
    int a )
```

fonction qui retourne la valeur absolue d'un entier



## Parameters

<i>a</i>	int
----------	-----

## Returns

$|a|$  : -a si  $a < 0$  ou a si  $a > 0$

#### 4.4.2.2 clean\_data()

```
void clean_data (
    world_t * world )
```

La fonction nettoie les données du monde.

## Parameters

<i>world</i>	les données du monde
--------------	----------------------

#### 4.4.2.3 depassement\_d()

```
void depassement_d (
    sprite_t * sprite )
```

fonction qui verifie que le sprite ne depaase pas la limite droite du jeu

## Parameters

<i>sprite</i>	vaisseau
---------------	----------

#### 4.4.2.4 depassement\_g()

```
void depassement_g (
    sprite_t * sprite )
```

fonction qui verifie que le sprite ne depaase pas la limite gauche du jeu

## Parameters

<i>sprite</i>	vaisseau
---------------	----------

#### 4.4.2.5 handle\_sprites\_collision()

```
void handle_sprites_collision (
    sprite_t * sp1,
    sprite_t * sp2,
    world_t * world,
    int make_disappear )
```

fonction qui verifie que 2 sprite ne soit pas en collision

##### Parameters

<i>sp1</i>	vaisseau
<i>sp2</i>	sprite (mur ou ligne arrivé)
<i>world</i>	monde
<i>make_disappear</i>	sprite visible/invisible

#### 4.4.2.6 init\_data()

```
void init_data (
    world_t * world )
```

La fonction initialise les données du monde du jeu.

##### Parameters

<i>world</i>	les données du monde
--------------	----------------------

#### 4.4.2.7 init\_sprite()

```
void init_sprite (
    sprite_t * sprite,
    int x,
    int y,
    int w,
    int h )
```

La fonction initialise la position du sprite.

##### Parameters

<i>sprite</i>	vaisseau
<i>x</i>	abscisse de la position du sprite
<i>y</i>	ordonnée de la position du sprite
<i>w</i>	largeur de la position du sprite
<i>h</i>	hauteur de la position du sprite

#### 4.4.2.8 is\_game\_over()

```
int is_game_over (
    world_t * world )
```

La fonction indique si le jeu est fini en fonction des données du monde.

##### Parameters

<i>world</i>	les données du monde
--------------	----------------------

##### Returns

1 si le jeu est fini, 0 sinon

#### 4.4.2.9 print\_sprite()

```
void print_sprite (
    sprite_t sprite )
```

La fonction affiche la position du sprite.

##### Parameters

<i>sprite</i>	vaisseau
---------------	----------

#### 4.4.2.10 sprites\_collide()

```
int sprites_collide (
    sprite_t * sp1,
    sprite_t * sp2 )
```

fonction qui verifie que 2 sprite ne soit pas en collision

##### Parameters

<i>sp1</i>	vaisseau
<i>sp2</i>	sprite (mur ou ligne arrivé)

#### 4.4.2.11 update\_data()

```
void update_data (
    world_t * world )
```

La fonction met à jour les données en tenant compte de la physique du monde.

##### Parameters

<i>les</i>	données du monde
------------	------------------

## 4.5 logique.h File Reference

Header du module logique.

### Classes

- struct [sprite\\_s](#)  
*Représentation d'un sprite du jeu.*
- struct [world\\_s](#)  
*Représentation du monde du jeu.*

### Typedefs

- typedef struct [sprite\\_s](#) [sprite\\_t](#)  
*Type qui correspond aux données du sprite.*
- typedef struct [world\\_s](#) [world\\_t](#)  
*Type qui correspond aux données du monde.*

### Functions

- void [init\\_sprite](#) ([sprite\\_t](#) \*sprite, int x, int y, int w, int h)  
*La fonction initialise la position du sprite.*
- void [init\\_data](#) ([world\\_t](#) \*world)  
*La fonction initialise les données du monde du jeu.*
- void [print\\_sprite](#) ([sprite\\_t](#) sprite)  
*La fonction affiche la position du sprite.*
- void [clean\\_data](#) ([world\\_t](#) \*world)  
*La fonction nettoie les données du monde.*
- int [is\\_game\\_over](#) ([world\\_t](#) \*world)  
*La fonction indique si le jeu est fini en fonction des données du monde.*
- void [update\\_data](#) ([world\\_t](#) \*world)  
*La fonction met à jour les données en tenant compte de la physique du monde.*
- void [depassement\\_g](#) ([sprite\\_t](#) \*sprite)  
*fonction qui verifie que le sprite ne depaase pas la limite gauche du jeu*
- void [depassement\\_d](#) ([sprite\\_t](#) \*sprite)

- fonction qui verifie que le sprite ne depaase pas la limite droite du jeu*
  - int `sprites_collide` (`sprite_t` \*sp1, `sprite_t` \*sp2)
    - fonction qui verifie que 2 sprite ne soit pas en collision*
  - void `handle_sprites_collision` (`sprite_t` \*sp1, `sprite_t` \*sp2, `world_t` \*world, int make\_disappear)
    - fonction qui verifie que 2 sprite ne soit pas en collision*
  - int `abs` (int a)
    - fonction qui retourne la valeur absolue d'un entier*

### 4.5.1 Detailed Description

Header du module logique.

#### Author

LESNIAK Louis & SLIMANI Kamelia

#### Version

2.0

#### Date

14 avril 2021

### 4.5.2 Function Documentation

#### 4.5.2.1 `abs()`

```
int abs (  
    int a )
```

fonction qui retourne la valeur absolue d'un entier

#### Parameters

<i>a</i>	int
----------	-----

#### Returns

$|a|$  : -a si  $a < 0$  ou a si  $a > 0$

#### 4.5.2.2 clean\_data()

```
void clean_data (
    world_t * world )
```

La fonction nettoie les données du monde.

##### Parameters

<i>world</i>	les données du monde
--------------	----------------------

#### 4.5.2.3 depassement\_d()

```
void depassement_d (
    sprite_t * sprite )
```

fonction qui verifie que le sprite ne depaase pas la limite droite du jeu

##### Parameters

<i>sprite</i>	vaisseau
---------------	----------

#### 4.5.2.4 depassement\_g()

```
void depassement_g (
    sprite_t * sprite )
```

fonction qui verifie que le sprite ne depaase pas la limite gauche du jeu

##### Parameters

<i>sprite</i>	vaisseau
---------------	----------

#### 4.5.2.5 handle\_sprites\_collision()

```
void handle_sprites_collision (
    sprite_t * sp1,
    sprite_t * sp2,
    world_t * world,
    int make_disappear )
```

fonction qui verifie que 2 sprite ne soit pas en collision

## Parameters

<i>sp1</i>	vaisseau
<i>sp2</i>	sprite (mur ou ligne arrivé)
<i>world</i>	monde
<i>make_disappear</i>	sprite visible/invisible

#### 4.5.2.6 init\_data()

```
void init_data (  
    world_t * world )
```

La fonction initialise les données du monde du jeu.

## Parameters

<i>world</i>	les données du monde
--------------	----------------------

#### 4.5.2.7 init\_sprite()

```
void init_sprite (  
    sprite_t * sprite,  
    int x,  
    int y,  
    int w,  
    int h )
```

La fonction initialise la position du sprite.

## Parameters

<i>sprite</i>	vaisseau
<i>x</i>	abscisse de la position du sprite
<i>y</i>	ordonnée de la position du sprite
<i>w</i>	largeur de la position du sprite
<i>h</i>	hauteur de la position du sprite

#### 4.5.2.8 is\_game\_over()

```
int is_game_over (  
    world_t * world )
```

La fonction indique si le jeu est fini en fonction des données du monde.

**Parameters**

<i>world</i>	les données du monde
--------------	----------------------

**Returns**

1 si le jeu est fini, 0 sinon

**4.5.2.9 print\_sprite()**

```
void print_sprite (
    sprite_t sprite )
```

La fonction affiche la position du sprite.

**Parameters**

<i>sprite</i>	vaisseau
---------------	----------

**4.5.2.10 sprites\_collide()**

```
int sprites_collide (
    sprite_t * sp1,
    sprite_t * sp2 )
```

fonction qui verifie que 2 sprite ne soit pas en collision

**Parameters**

<i>sp1</i>	vaisseau
<i>sp2</i>	sprite (mur ou ligne arrivé)

**4.5.2.11 update\_data()**

```
void update_data (
    world_t * world )
```

La fonction met à jour les données en tenant compte de la physique du monde.

**Parameters**

<i>les</i>	données du monde
------------	------------------



## 4.6 main.c File Reference

Programme principal initial du niveau 2.

```
#include "sdl2-light.h"
#include "logique.h"
#include "graphique.h"
#include "definition.h"
```

### Functions

- void [handle\\_events](#) (SDL\_Event \*event, [world\\_t](#) \*world)  
*La fonction gère les évènements ayant eu lieu et qui n'ont pas encore été traités.*
- int [main](#) (int argc, char \*args[])  
*programme principal qui implémente la boucle du jeu*

### 4.6.1 Detailed Description

Programme principal initial du niveau 2.

#### Author

LESNIAK Louis & SLIMANI Kamelia

#### Version

2.0

#### Date

14 avril 2021

### 4.6.2 Function Documentation

#### 4.6.2.1 handle\_events()

```
void handle_events (
    SDL_Event * event,
    world_t * world )
```

La fonction gère les évènements ayant eu lieu et qui n'ont pas encore été traités.

#### Parameters

<i>event</i>	paramètre qui contient les événements
<i>world</i>	les données du monde

## 4.7 sdl2-light.c File Reference

sur-couche de SDL2 pour simplifier son utilisation pour le projet

```
#include "sdl2-light.h"
#include <stdio.h>
#include <stdlib.h>
```

### Functions

- int [init\\_sdl](#) (SDL\_Window \*\*window, SDL\_Renderer \*\*renderer, int width, int height)  
*La fonction initialise la SDL. Elle crée la fenêtre du jeu ainsi que le renderer.*
- SDL\_Texture \* [load\\_image](#) (const char path[], SDL\_Renderer \*renderer)  
*La fonction charge une image et renvoie la texture correspondante où la couleur RGB (255, 0, 255) est rendue transparente.*
- void [apply\\_texture](#) (SDL\_Texture \*texture, SDL\_Renderer \*renderer, int x, int y)  
*La fonction permet d'appliquer une texture sur le renderer à une position donnée. La hauteur et la largeur est la même que celle de la texture.*
- void [clean\\_texture](#) (SDL\_Texture \*texture)  
*La fonction nettoie une texture en mémoire.*
- void [clear\\_renderer](#) (SDL\_Renderer \*renderer)  
*La fonction vide le contenu graphique du renderer lié à l'écran de jeu.*
- void [update\\_screen](#) (SDL\_Renderer \*renderer)  
*La fonction met à jour l'écran avec le contenu du renderer.*
- void [pause](#) (int time)  
*La fonction met le programme en pause pendant un laps de temps.*
- void [clean\\_sdl](#) (SDL\_Renderer \*renderer, SDL\_Window \*window)  
*La fonction nettoie le renderer et la fenêtre du jeu en mémoire.*

### 4.7.1 Detailed Description

sur-couche de SDL2 pour simplifier son utilisation pour le projet

Author

Mathieu Constant

Version

0.2

Date

10 mars 2021

### 4.7.2 Function Documentation

#### 4.7.2.1 apply\_texture()

```
void apply_texture (
    SDL_Texture * texture,
    SDL_Renderer * renderer,
    int x,
    int y )
```

La fonction permet d'appliquer une texture sur le renderer à une position donnée. La hauteur et la largeur est la même que celle de la texture.

##### Parameters

<i>texture</i>	la texture que l'on va appliquer
<i>renderer</i>	le renderer qui va recevoir la texture
<i>x</i>	l'abscisse sur le renderer de l'endroit où est appliquée texture (point en haut à gauche de la surface)
<i>y</i>	l'ordonnée sur le renderer de l'endroit où est appliquée texture (point en haut à gauche de la surface)

#### 4.7.2.2 clean\_sdl()

```
void clean_sdl (
    SDL_Renderer * renderer,
    SDL_Window * window )
```

La fonction nettoie le renderer et la fenêtre du jeu en mémoire.

##### Parameters

<i>renderer</i>	le renderer à nettoyer
<i>window</i>	la fenêtre à nettoyer

#### 4.7.2.3 clean\_texture()

```
void clean_texture (
    SDL_Texture * texture )
```

La fonction nettoie une texture en mémoire.

##### Parameters

<i>texture</i>	la texture à nettoyer
----------------	-----------------------

#### 4.7.2.4 clear\_renderer()

```
void clear_renderer (
    SDL_Renderer * renderer )
```

La fonction vide le contenu graphique du renderer lié à l'écran de jeu.

##### Parameters

<i>renderer</i>	le renderer de l'écran
-----------------	------------------------

#### 4.7.2.5 init\_sdl()

```
int init_sdl (
    SDL_Window ** window,
    SDL_Renderer ** renderer,
    int width,
    int height )
```

La fonction initialise la SDL. Elle crée la fenêtre du jeu ainsi que le renderer.

##### Parameters

<i>window</i>	la fenêtre du jeu
<i>renderer</i>	le renderer
<i>width</i>	largeur de l'écran de jeu
<i>height</i>	hauteur de l'écran de jeu

##### Returns

-1 en cas d'erreur, 0 sinon

#### 4.7.2.6 load\_image()

```
SDL_Texture* load_image (
    const char path[],
    SDL_Renderer * renderer )
```

La fonction charge une image et renvoie la texture correspondante où la couleur RGB (255, 0, 255) est rendue transparente.

##### Parameters

<i>path</i>	est le chemin du fichier image. Le fichier doit être obligatoirement du BMP.
<i>renderer</i>	le renderer

**Returns**

la surface SDL contenant l'image avec la couleur RGB (255,0,255) rendue transparente. Elle renvoie NULL si le chargement a échoué (ex. le fichier path n'existe pas)

**4.7.2.7 pause()**

```
void pause (
    int time )
```

La fonction met le programme en pause pendant un laps de temps.

**Parameters**

<i>time</i>	ce laps de temps en milliseconde
-------------	----------------------------------

**4.7.2.8 update\_screen()**

```
void update_screen (
    SDL_Renderer * renderer )
```

La fonction met à jour l'écran avec le contenu du renderer.

**Parameters**

<i>renderer</i>	le renderer de l'écran
-----------------	------------------------

**4.8 sdl2-light.h File Reference**

en-tête du module correspondant à une sur-couche de SDL2 pour simplifier son utilisation pour le projet

```
#include "include/SDL.h"
```

**Functions**

- void [clean\\_sdl](#) (SDL\_Renderer \*renderer, SDL\_Window \*window)  
*La fonction nettoie le renderer et la fenêtre du jeu en mémoire.*
- SDL\_Texture \* [load\\_image](#) (const char path[], SDL\_Renderer \*renderer)  
*La fonction charge une image et renvoie la texture correspondante où la couleur RGB (255, 0, 255) est rendue transparente.*
- int [init\\_sdl](#) (SDL\_Window \*\*window, SDL\_Renderer \*\*renderer, int width, int height)  
*La fonction initialise la SDL. Elle crée la fenêtre du jeu ainsi que le renderer.*

- void `clean_texture` (SDL\_Texture \*texture)  
*La fonction nettoie une texture en mémoire.*
- void `apply_texture` (SDL\_Texture \*texture, SDL\_Renderer \*renderer, int x, int y)  
*La fonction permet d'appliquer une texture sur le renderer à une position donnée. La hauteur et la largeur est la même que celle de la texture.*
- void `clear_renderer` (SDL\_Renderer \*renderer)  
*La fonction vide le contenu graphique du renderer lié à l'écran de jeu.*
- void `update_screen` (SDL\_Renderer \*renderer)  
*La fonction met à jour l'écran avec le contenu du renderer.*
- void `pause` (int time)  
*La fonction met le programme en pause pendant un laps de temps.*

### 4.8.1 Detailed Description

en-tête du module correspondant à une sur-couche de SDL2 pour simplifier son utilisation pour le projet

#### Author

Mathieu Constant

#### Version

0.2

#### Date

10 mars 2021

### 4.8.2 Function Documentation

#### 4.8.2.1 `apply_texture()`

```
void apply_texture (  
    SDL_Texture * texture,  
    SDL_Renderer * renderer,  
    int x,  
    int y )
```

La fonction permet d'appliquer une texture sur le renderer à une position donnée. La hauteur et la largeur est la même que celle de la texture.

#### Parameters

<i>texture</i>	la texture que l'on va appliquer
<i>renderer</i>	le renderer qui va recevoir la texture
<i>x</i>	l'abscisse sur le renderer de l'endroit où est appliquée texture (point en haut à gauche de la surface)
<i>y</i>	l'ordonnée sur le renderer de l'endroit où est appliquée texture (point en haut à gauche de la surface)

#### 4.8.2.2 clean\_sdl()

```
void clean_sdl (
    SDL_Renderer * renderer,
    SDL_Window * window )
```

La fonction nettoie le renderer et la fenêtre du jeu en mémoire.

##### Parameters

<i>renderer</i>	le renderer à nettoyer
<i>window</i>	la fenêtre à nettoyer

#### 4.8.2.3 clean\_texture()

```
void clean_texture (
    SDL_Texture * texture )
```

La fonction nettoie une texture en mémoire.

##### Parameters

<i>texture</i>	la texture à nettoyer
----------------	-----------------------

#### 4.8.2.4 clear\_renderer()

```
void clear_renderer (
    SDL_Renderer * renderer )
```

La fonction vide le contenu graphique du renderer lié à l'écran de jeu.

##### Parameters

<i>renderer</i>	le renderer de l'écran
-----------------	------------------------

#### 4.8.2.5 init\_sdl()

```
int init_sdl (
    SDL_Window ** window,
```

```
SDL_Renderer ** renderer,  
int width,  
int height )
```

La fonction initialise la SDL. Elle crée la fenêtre du jeu ainsi que le renderer.

#### Parameters

<i>window</i>	la fenêtre du jeu
<i>renderer</i>	le renderer
<i>width</i>	largeur de l'écran de jeu
<i>height</i>	hauteur de l'écran de jeu

#### Returns

-1 en cas d'erreur, 0 sinon

#### 4.8.2.6 load\_image()

```
SDL_Texture* load_image (  
    const char path[],  
    SDL_Renderer * renderer )
```

La fonction charge une image et renvoie la texture correspondante où la couleur RGB (255, 0, 255) est rendue transparente.

#### Parameters

<i>path</i>	est le chemin du fichier image. Le fichier doit être obligatoirement du BMP.
<i>renderer</i>	le renderer

#### Returns

la surface SDL contenant l'image avec la couleur RGB (255,0,255) rendue transparente. Elle renvoie NULL si le chargement a échoué (ex. le fichier path n'existe pas)

#### 4.8.2.7 pause()

```
void pause (  
    int time )
```

La fonction met le programme en pause pendant un laps de temps.

#### Parameters

<i>time</i>	ce laps de temps en milliseconde
-------------	----------------------------------



#### 4.8.2.8 update\_screen()

```
void update_screen (
    SDL_Renderer * renderer )
```

La fonction met à jour l'écran avec le contenu du renderer.

##### Parameters

<i>renderer</i>	le renderer de l'écran
-----------------	------------------------

## 4.9 tests.c File Reference

Programme testant la partie logique du jeu.

```
#include "definition.h"
#include "logique.h"
#include <stdio.h>
```

### Functions

- void **test\_init\_sprite\_param** ([sprite\\_t](#) \*sprite, int x, int y, int w, int h)
- void **test\_init\_sprite** ()
- void **test\_depassement\_g\_param** ([sprite\\_t](#) \*sprite)
- void **test\_depassement\_g** ()
- void **test\_depassement\_d\_param** ([sprite\\_t](#) \*sprite)
- void **test\_depassement\_d** ()
- void **test\_sprites\_collide\_param** ([sprite\\_t](#) \*sprite1, [sprite\\_t](#) \*sprite2)
- void **test\_sprites\_collide** ()
- void **test\_handle\_sprites\_collision\_param** ([sprite\\_t](#) \*sp1, [sprite\\_t](#) \*sp2, [world\\_t](#) \*world)
- void **test\_handle\_sprites\_collision** ()
- int **main** ()

#### 4.9.1 Detailed Description

Programme testant la partie logique du jeu.

##### Author

LESNIAK Louis & SLIMANI Kamelia

##### Version

2.0

##### Date

14 avril 2021



# Index

- abs
  - logique.c, 18
  - logique.h, 23
- apply\_background
  - graphique.c, 11
  - graphique.h, 15
- apply\_meteorite
  - graphique.c, 11
  - graphique.h, 15
- apply\_sprite
  - graphique.c, 11
  - graphique.h, 15
- apply\_texture
  - sdl2-light.c, 28
  - sdl2-light.h, 32
- background
  - textures\_s, 6
- clean
  - graphique.c, 12
  - graphique.h, 16
- clean\_data
  - logique.c, 19
  - logique.h, 23
- clean\_sdl
  - sdl2-light.c, 29
  - sdl2-light.h, 33
- clean\_texture
  - sdl2-light.c, 29
  - sdl2-light.h, 33
- clean\_textures
  - graphique.c, 12
  - graphique.h, 16
- clear\_renderer
  - sdl2-light.c, 29
  - sdl2-light.h, 33
- collision\_mur
  - world\_s, 7
- definition.h, 9
- depassement\_d
  - logique.c, 19
  - logique.h, 24
- depassement\_g
  - logique.c, 19
  - logique.h, 24
- gameover
  - world\_s, 8
- graphique.c, 10
  - apply\_background, 11
  - apply\_meteorite, 11
  - apply\_sprite, 11
  - clean, 12
  - clean\_textures, 12
  - init, 12
  - init\_textures, 13
  - refresh\_graphics, 13
- graphique.h, 13
  - apply\_background, 15
  - apply\_meteorite, 15
  - apply\_sprite, 15
  - clean, 16
  - clean\_textures, 16
  - init, 16
  - init\_textures, 17
  - refresh\_graphics, 17
- h
  - sprite\_s, 5
- handle\_events
  - main.c, 27
- handle\_sprites\_collision
  - logique.c, 20
  - logique.h, 24
- init
  - graphique.c, 12
  - graphique.h, 16
- init\_data
  - logique.c, 20
  - logique.h, 25
- init\_sdl
  - sdl2-light.c, 30
  - sdl2-light.h, 33
- init\_sprite
  - logique.c, 20
  - logique.h, 25
- init\_textures
  - graphique.c, 13
  - graphique.h, 17
- is\_game\_over
  - logique.c, 21
  - logique.h, 25
- ligne\_arrive
  - textures\_s, 6
  - world\_s, 8
- load\_image

- sdl2-light.c, 30
  - sdl2-light.h, 34
- logique.c, 17
  - abs, 18
  - clean\_data, 19
  - depassement\_d, 19
  - depassement\_g, 19
  - handle\_sprites\_collision, 20
  - init\_data, 20
  - init\_sprite, 20
  - is\_game\_over, 21
  - print\_sprite, 21
  - sprites\_collide, 21
  - update\_data, 21
- logique.h, 22
  - abs, 23
  - clean\_data, 23
  - depassement\_d, 24
  - depassement\_g, 24
  - handle\_sprites\_collision, 24
  - init\_data, 25
  - init\_sprite, 25
  - is\_game\_over, 25
  - print\_sprite, 26
  - sprites\_collide, 26
  - update\_data, 26
- main.c, 27
  - handle\_events, 27
- meteorite
  - textures\_s, 7
- mur
  - world\_s, 8
- pause
  - sdl2-light.c, 31
  - sdl2-light.h, 34
- print\_sprite
  - logique.c, 21
  - logique.h, 26
- refresh\_graphics
  - graphique.c, 13
  - graphique.h, 17
- sdl2-light.c, 28
  - apply\_texture, 28
  - clean\_sdl, 29
  - clean\_texture, 29
  - clear\_renderer, 29
  - init\_sdl, 30
  - load\_image, 30
  - pause, 31
  - update\_screen, 31
- sdl2-light.h, 31
  - apply\_texture, 32
  - clean\_sdl, 33
  - clean\_texture, 33
  - clear\_renderer, 33
- init\_sdl, 33
  - load\_image, 34
  - pause, 34
  - update\_screen, 35
- sprite\_s, 5
  - h, 5
  - w, 5
  - x, 5
  - y, 6
- sprites\_collide
  - logique.c, 21
  - logique.h, 26
- tests.c, 35
- textures\_s, 6
  - background, 6
  - ligne\_arrive, 6
  - meteorite, 7
  - vaisseau, 7
- update\_data
  - logique.c, 21
  - logique.h, 26
- update\_screen
  - sdl2-light.c, 31
  - sdl2-light.h, 35
- vaisseau
  - textures\_s, 7
  - world\_s, 8
- w
  - sprite\_s, 5
- world\_s, 7
  - collision\_mur, 7
  - gameover, 8
  - ligne\_arrive, 8
  - mur, 8
  - vaisseau, 8
- x
  - sprite\_s, 5
- y
  - sprite\_s, 6