Star Seeker

Généré par Doxygen 1.9.2

Chapitre 1

Index des classes

1.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Chapitre 2

Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

Chapitre 3

Documentation des classes

3.1 Référence de la structure element

Attributs publics

- t_entite * **valeur**
- struct element * **pred**
- struct element * **succ**

La documentation de cette structure a été générée à partir du fichier suivant :
- include/liste.h

3.2 Référence de la structure niveau_base_t

Toutes les informations relatives au stockage d'un niveau.

3.2.1 Description détaillée

Toutes les informations relatives au stockage d'un niveau.

La documentation de cette structure a été générée à partir du fichier suivant :
- include/generation_niveau.h

3.3 Référence de la structure niveau_informations_t

Attributs publics

- int **hauteur**
- int **longueur**
- int **rouge**
- int **vert**
- int **bleu**
- int **matrice** [LONGUEUR_NIVEAU_MAX][HAUTEUR_NIVEAU_MAX]
- int **i_dep**
- int **j_dep**
- int **i_fin**
- int **j_fin**

La documentation de cette structure a été générée à partir du fichier suivant :
- include/generation_niveau.h

3.4 Référence de la structure s_animation

Structure contenant les données nécéssaires à la réalisation d'une animation.

`#include <animation.h>`

Attributs publics

- int vitesse
- int **nb_textures**
- int indice_texture
- int id_max
- unsigned int dernier_update

3.4.1 Description détaillée

Structure contenant les données nécéssaires à la réalisation d'une animation.

Les champs de cette structure ne sont pas destinés à être changés après l'appel de la fonction de création.

3.4.2 Documentation des données membres

3.4.2.1 dernier_update

`unsigned int s_animation::dernier_update`

Nombre de lignes du tileset (pour des mesures du prévention) Temps qu'il était au moment du changement vers la l'image courante

3.4.2.2 id_max

`int s_animation::id_max`

Indice de l'image active de l'animation courante

3.4.2.3 indice_texture

`int s_animation::indice_texture`

Nombre d'images sur une ligne du tileset

3.4.2.4 vitesse

```
int s_animation::vitesse
```

Temps en milisecondes avant le passage à l'image suivante

La documentation de cette structure a été générée à partir du fichier suivant :
— include/animation.h

3.5 Référence de la structure s_entite

La documentation de cette structure a été générée à partir du fichier suivant :
— include/entite.h

3.6 Référence de la structure s_monstre

Attributs publics

— void(∗ **deplacement**)(t_monstre ∗, float, float)
— int **cpt_deplacement**

La documentation de cette structure a été générée à partir du fichier suivant :
— include/monstre.h

3.7 Référence de la structure s_moteur

Attributs publics

— SDL_Window ∗ **window**
— SDL_Renderer ∗ **renderer**
— t_textures ∗ **textures**
— unsigned int temps
— unsigned int **temps_precedent**
— t_camera ∗ camera
— t_niveau ∗ niveau_charge
— int echelle
— int window_width
— int window_height

3.7.1 Documentation des données membres

3.7.1.1 camera

```
t_camera* s_moteur::camera
```

Temps au début de la frame précédente

3.7.1.2 echelle

```
int s_moteur::echelle
```

Niveau actuellement chargé Echelle du jeu, c'est à dire la taille des éléments

3.7.1.3 niveau_charge

```
t_niveau* s_moteur::niveau_charge
```

Caméra du jeu

3.7.1.4 temps

```
unsigned int s_moteur::temps
```

Temps au début d'une frame

3.7.1.5 window_height

```
int s_moteur::window_height
```

Hauteur de la fenêtre

3.7.1.6 window_width

```
int s_moteur::window_width
```

Largeur de la fenêtre

La documentation de cette structure a été générée à partir du fichier suivant :

— include/moteur.h

3.8 Référence de la structure s_niveau

Attributs publics

— t_salle ** **salles**
— int h
— int l
— t_salle * salle_chargee
— int **i_charge**
— int j_charge
— t_liste * liste_entites
— SDL_Rect * collisions
— int taille_collisions

3.8.1 Documentation des données membres

3.8.1.1 collisions

```
SDL_Rect* s_niveau::collisions
```

Tableau des zones non accessibles au joueur

3.8.1.2 h

```
int s_niveau::h
```

Hauteur du niveau (dimensions de la matrice en i)

3.8.1.3 j_charge

```
int s_niveau::j_charge
```

Position en i dans la matrice de salle de la salle chargée (utile à updateNiveau)

3.8.1.4 l

```
int s_niveau::l
```

Largeur du niveau (dimensions de la matrice en j)

3.8.1.5 liste_entites

```
t_liste* s_niveau::liste_entites
```

Position en j dans la matrice de salle de la salle chargée (utile à updateNiveau) Liste des entités rendues "vivantes"

3.8.1.6 salle_chargee

```
t_salle* s_niveau::salle_chargee
```

Salle ou sous salle où se situe le joueur

3.8.1.7 taille_collisions

```
int s_niveau::taille_collisions
```

Taille du tableau de collisions

La documentation de cette structure a été générée à partir du fichier suivant :
— include/niveau.h

3.9 Référence de la structure s_personnage

La documentation de cette structure a été générée à partir du fichier suivant :
— include/personnage.h

3.10 Référence de la structure s_projectile

Attributs publics

— e_type_entite cible
— int dommages
— int duree_de_vie

3.10.1 Documentation des données membres

3.10.1.1 cible

```
e_type_entite s_projectile::cible
```

Type d'entité prenant les degats si touché

3.10.1.2 dommages

```
int s_projectile::dommages
```

Points de dégats infligés

3.10.1.3 duree_de_vie

```
int s_projectile::duree_de_vie
```

Temps en milisecondes avant l'auto-destruction du projectile

La documentation de cette structure a été générée à partir du fichier suivant :
— include/projectiles.h

3.11 Référence de la structure s_salle

Attributs publics

- struct s_salle * portes [NOMBRE_DE_PORTES]
- int id_salle
- t_dimensions_salle * dimensions
- t_entite ** entites
- int nb_entite

3.11.1 Documentation des données membres

3.11.1.1 dimensions

```
t_dimensions_salle* s_salle::dimensions
```

Informations sur le groupe de salle de notre salle

3.11.1.2 entites

```
t_entite** s_salle::entites
```

Entités générés avec la salle

3.11.1.3 id_salle

```
int s_salle::id_salle
```

Manière d'identifier si plusieurs salles forment une même salle

3.11.1.4 nb_entite

```
int s_salle::nb_entite
```

Nombre des entités générés avec la salle

3.11.1.5 portes

```
struct s_salle* s_salle::portes[NOMBRE_DE_PORTES]
```

Salles reliées à notre salle

La documentation de cette structure a été générée à partir du fichier suivant :

- include/niveau.h

3.12 Référence de la structure t_attaque_corps

Modélise une arme de corps à corps.

`#include <attaque.h>`

Attributs publics

- int **cooldown**
- int **cooldown_restant**

3.12.1 Description détaillée

Modélise une arme de corps à corps.

La documentation de cette structure a été générée à partir du fichier suivant :
- include/attaque.h

3.13 Référence de la structure t_attaque_tir

Modélise une arme générant des projectiles.

`#include <attaque.h>`

Attributs publics

- e_type_projectile **type_projectile**
- int **cooldown**
- int nb_salves
- int nb_proj_salve
- int tir_interval
- int nb_salves_restantes
- int temps_debut_attaque

3.13.1 Description détaillée

Modélise une arme générant des projectiles.

3.13.2 Documentation des données membres

3.13.2.1 nb_proj_salve

`int t_attaque_tir::nb_proj_salve`

Nombre de salves

3.13.2.2 nb_salves

```
int t_attaque_tir::nb_salves
```

Temps avant de pouvoir réutiliser l'attaque

3.13.2.3 nb_salves_restantes

```
int t_attaque_tir::nb_salves_restantes
```

Temps entre le tir de chaque salve

3.13.2.4 temps_debut_attaque

```
int t_attaque_tir::temps_debut_attaque
```

Nombre de salves restantes

3.13.2.5 tir_interval

```
int t_attaque_tir::tir_interval
```

Nombre de projectiles par salves

La documentation de cette structure a été générée à partir du fichier suivant :

— include/attaque.h

3.14 Référence de la structure t_camera

Modélise une caméra virtuelle qu'il suffit de bouger pour déplacer tout l'affichage des éléments sur la fenêtre.

```
#include <camera.h>
```

Attributs publics

— float x
— float y
— float futur_x
— float futur_y

3.14.1 Description détaillée

Modélise une caméra virtuelle qu'il suffit de bouger pour déplacer tout l'affichage des éléments sur la fenêtre.

3.14.2 Documentation des données membres

3.14.2.1 futur_x

```
float t_camera::futur_x
```

Stockage d'une future position de la caméra en x (pour l'animation de transition)

3.14.2.2 futur_y

```
float t_camera::futur_y
```

Stockage d'une future position de la caméra en y (pour l'animation de transition)

3.14.2.3 x

```
float t_camera::x
```

Position en x du centre de la caméra par rapport au niveau

3.14.2.4 y

```
float t_camera::y
```

Position en y du centre de la caméra par rapport au niveau

La documentation de cette structure a été générée à partir du fichier suivant :

— include/camera.h

3.15 Référence de la structure t_couleurRVB

Structure qui stocke une couleur décomposée en RVB. Chaque int est compris entre 0 et 255 inclus.

```
#include <generation_niveau.h>
```

Attributs publics

— int **rouge**
— int **vert**
— int **bleu**

3.15.1 Description détaillée

Structure qui stocke une couleur décomposée en RVB. Chaque int est compris entre 0 et 255 inclus.

La documentation de cette structure a été générée à partir du fichier suivant :

— include/generation_niveau.h

3.16 Référence de la structure t_dimensions_salle

Informations sur le nombre de sous-salles formant la salle.

```
#include <niveau.h>
```

Attributs publics

— int i
— int j
— int largeur
— int hauteur
— int nombre

3.16.1 Description détaillée

Informations sur le nombre de sous-salles formant la salle.

La salle d'origine désigne la salle la plus en haut à gauche du groupe.

3.16.2 Documentation des données membres

3.16.2.1 hauteur

```
int t_dimensions_salle::hauteur
```

Hauteur de la salle

3.16.2.2 i

```
int t_dimensions_salle::i
```

Coordonnée en hauteur, de la salle d'origine dans la matrice niveau

3.16.2.3 j

```
int t_dimensions_salle::j
```

Coordonnée en largeur, de la salle d'origine dans la matrice niveau

3.16.2.4 largeur

```
int t_dimensions_salle::largeur
```

Largeur de la salle

3.16.2.5 nombre

```
int t_dimensions_salle::nombre
```

Nombre de sous-salles (utile pour la libération de la mémoire)

La documentation de cette structure a été générée à partir du fichier suivant :

— include/niveau.h

3.17 Référence de la structure t_entite

Modélise une entité.

```
#include <entite.h>
```

3.17.1 Description détaillée

Modélise une entité.

La documentation de cette structure a été générée à partir du fichier suivant :

— include/entite.h

3.18 Référence de la structure t_joueur

Structure modélisant le joueur, hérite des attributs d'entité et de personnage.

```
#include <joueur.h>
```

Attributs publics

— t_joueur_flags * **flags**
— t_attaque_tir **attaque_tir_equipee**

3.18.1 Description détaillée

Structure modélisant le joueur, hérite des attributs d'entité et de personnage.

La documentation de cette structure a été générée à partir du fichier suivant :

— include/joueur.h

3.19 Référence de la structure t_joueur_flags

Structure contenant les entrées clavier/souris actives du joueur.

`#include <joueur.h>`

Attributs publics

— int **to_up**
— int **to_down**
— int **to_left**
— int **to_right**
— int **shooting**

3.19.1 Description détaillée

Structure contenant les entrées clavier/souris actives du joueur.

La documentation de cette structure a été générée à partir du fichier suivant :

— include/joueur.h

3.20 Référence de la structure t_liste

Attributs publics

— t_element * **drapeau**
— t_element * **ec**

La documentation de cette structure a été générée à partir du fichier suivant :

— include/liste.h

3.21 Référence de la structure t_moteur

Objet contenant les données nécéssaires au rendu du jeu, aussi pour la gestion des collisions.

`#include <moteur.h>`

3.21.1 Description détaillée

Objet contenant les données nécéssaires au rendu du jeu, aussi pour la gestion des collisions.

La documentation de cette structure a été générée à partir du fichier suivant :
— include/moteur.h

3.22 Référence de la structure t_niveau

Structure représentant une matrice de salles, c'est à dire un niveau.

`#include <niveau.h>`

3.22.1 Description détaillée

Structure représentant une matrice de salles, c'est à dire un niveau.

La documentation de cette structure a été générée à partir du fichier suivant :
— include/niveau.h

3.23 Référence de la structure t_personnage

Modélise un personnage (joueur ou ennemi).

`#include <personnage.h>`

3.23.1 Description détaillée

Modélise un personnage (joueur ou ennemi).

La documentation de cette structure a été générée à partir du fichier suivant :
— include/personnage.h

3.24 Référence de la structure t_projectile

Modélise une entité.

`#include <projectiles.h>`

3.24.1 Description détaillée

Modélise une entité.

La documentation de cette structure a été générée à partir du fichier suivant :
— include/projectiles.h

3.25 Référence de la structure t_salle

Structure représentant une salle et les salles qui lui sont liées.

`#include <niveau.h>`

3.25.1 Description détaillée

Structure représentant une salle et les salles qui lui sont liées.

Une salle est un rectangle, mais peut être associée à d'autres salles pour former des salles aux formes plus complexes. Plusieurs sous-salles formant une même salle complexe portent le même id_salle et pointent sur la même structure dimensions.

La documentation de cette structure a été générée à partir du fichier suivant :

— include/niveau.h

3.26 Référence de la structure t_textures

Contient toutes les textures du jeu.

`#include <textures.h>`

Attributs publics

— SDL_Texture ∗ **map**
— SDL_Texture ∗ **player**
— SDL_Texture ∗ **projectiles**

3.26.1 Description détaillée

Contient toutes les textures du jeu.

La documentation de cette structure a été générée à partir du fichier suivant :

— include/textures.h

Chapitre 4

Documentation des fichiers

4.1 Référence du fichier include/animation.h

Module facilitant la gestion des animations.

Classes

— struct s_animation
Structure contenant les données nécéssaires à la réalisation d'une animation.

Définitions de type

— typedef struct s_animation t_animation
Structure contenant les données nécéssaires à la réalisation d'une animation.

Fonctions

— void updateAnimation (t_animation ∗animation, unsigned int temps)
Actualise l'animation si nécéssaire selon la configuration de la structure.
— t_animation ∗ creerAnimation (int vitesse, int nb_textures, int id_max)
Génère une structure permettant d'animer un objet.
— void detruireAnimation (t_animation ∗∗animation)
Détruit une structure animation et mets son pointeur à NULL.

4.1.1 Description détaillée

Module facilitant la gestion des animations.

Ce module utilise les configurations données dans la structure t_animation pour se charger d'animer une texture avec updateAnimation.

Auteur

Julien Rouaux

4.1.2 Documentation des définitions de type

4.1.2.1 t_animation

```
typedef struct s_animation t_animation
```

Structure contenant les données nécéssaires à la réalisation d'une animation.

Les champs de cette structure ne sont pas destinés à être changés après l'appel de la fonction de création.

4.1.3 Documentation des fonctions

4.1.3.1 creerAnimation()

```
t_animation* creerAnimation (
             int vitesse,
             int nb_textures,
             int id_max )
```

Génère une structure permettant d'animer un objet.

Il est possible qu'une entité ne possède pas d'animation (pointeur NULL). Dans ce cas le champ id_animation permet de sélectionner la bonne partie de la texture.

Paramètres

vitesse	Temps en milisecondes avant le passage à la frame suivante de l'animation
nb_textures	Nombre de frames différentes d'une animation
id_max	Identifiant d'animation le plus élevé

Renvoie

La structure animation ou NULL si échec.

4.1.3.2 detruireAnimation()

```
void detruireAnimation (
             t_animation ** animation )
```

Détruit une structure animation et mets son pointeur à NULL.

Paramètres

animation	L'adresse du pointeur animation

4.1.3.3 updateAnimation()

```
void updateAnimation (
            t_animation * animation,
            unsigned int temps )
```

Actualise l'animation si nécéssaire selon la configuration de la structure.

Paramètres

animation	L'animation à actualiser
temps	Le temps actuel du jeu

4.2 Référence du fichier include/attaque.h

Module gérant les attaques des personnages.

```
#include <projectiles.h>
```

Classes

- struct t_attaque_tir
  Modélise une arme générant des projectiles.
- struct t_attaque_corps
  Modélise une arme de corps à corps.

Énumérations

- enum **e_nom_attaque** { **DEMO** }

Fonctions

- void **chargerAttaqueTir** (t_attaque_tir *attaque, e_nom_attaque nouvelle_attaque)

4.2.1 Description détaillée

Module gérant les attaques des personnages.

Auteur

Julien Rouaux

4.3 Référence du fichier include/camera.h

Module de calcul de position de caméra pour savoir où dessiner les éléments sur la fenêtre.

```
#include <SDL2/SDL.h>
```

Classes

— struct t_camera
Modélise une caméra virtuelle qu'il suffit de bouger pour déplacer tout l'affichage des éléments sur la fenêtre.

Définitions de type

— typedef struct s_moteur **t_moteur**

Fonctions

— t_camera * creerCamera (float x, float y)
Alloue la mémoire pour une caméra.
— void detruireCamera (t_camera **camera)
Libère la mémoire allouée à une caméra et mets son pointeur à NULL.
— void updateCamera (t_moteur *moteur, int largeur, int hauteur, int orig_x, int orig_y, float j_x, float j_y)
Place la caméra au bon endroit selon la configuration de la salle et de la position du joueur. Le sujet de la caméra sera placé au centre.
— void updateFutureCamera (t_moteur *moteur, int largeur, int hauteur, int orig_x, int orig_y, float j_x, float j_y)
Place la caméra au bon endroit selon la configuration de la salle et de la position du joueur. Le sujet de la caméra sera placé au centre.

4.3.1 Description détaillée

Module de calcul de position de caméra pour savoir où dessiner les éléments sur la fenêtre.

Auteur

Julien Rouaux

4.3.2 Documentation des fonctions

4.3.2.1 creerCamera()

```
t_camera* creerCamera (
            float x,
            float y )
```

Alloue la mémoire pour une caméra.

Paramètres

x	Position de la caméra en x
y	Position de la caméra en y

Renvoie

Le pointeur de la caméra, NULL si échec.

4.3.2.2 detruireCamera()

```
void detruireCamera (
            t_camera ** camera )
```

Libère la mémoire allouée à une caméra et mets son pointeur à NULL.

Paramètres

camera	L'adresse du pointeur de la caméra

4.3.2.3 updateCamera()

```
void updateCamera (
            t_moteur * moteur,
            int largeur,
            int hauteur,
            int orig_x,
            int orig_y,
            float j_x,
            float j_y )
```

Place la caméra au bon endroit selon la configuration de la salle et de la position du joueur. Le sujet de la caméra sera placé au centre.

Les coordonnées de la caméra sont mises à l'échelle du jeu car elles prennent en compte la taille de la fenêtre.

Paramètres

moteur	Moteur du jeu
camera	Camera à actualiser
largeur	Nombre de sous-salles contenues dans la salle en largeur
hauteur	Nombre de sous-salles contenues dans la salle en hauteur
orig_x	Coordonnées d'origine de la salle en x
orig_y	Coordonnées d'origine de la salle en y
j_x	Position en x du joueur relative au niveau
j_y	Position en y du joueur relative au niveau

4.3.2.4 updateFutureCamera()

```
void updateFutureCamera (
            t_moteur * moteur,
            int largeur,
            int hauteur,
            int orig_x,
            int orig_y,
            float j_x,
            float j_y )
```

Place la caméra au bon endroit selon la configuration de la salle et de la position du joueur. Le sujet de la caméra sera placé au centre.

Les coordonnées de la caméra sont mises à l'échelle du jeu car elles prennent en compte la taille de la fenêtre.

Paramètres

moteur	Moteur du jeu
camera	Camera à actualiser
largeur	Nombre de sous-salles contenues dans la salle en largeur
hauteur	Nombre de sous-salles contenues dans la salle en hauteur
orig_x	Coordonnées d'origine de la salle en x
orig_y	Coordonnées d'origine de la salle en y
j_x	Position en x du joueur relative au niveau
j_y	Position en y du joueur relative au niveau

4.4 Référence du fichier include/entite.h

Module de manipulation des entites.

```
#include <SDL2/SDL.h>
#include <moteur.h>
#include <animation.h>
#include <attributs_entites.h>
```

Classes

— struct s_entite

Définitions de type

— typedef struct s_entite **t_entite**

Énumérations

— enum e_type_entite { **E_AUCUN** , **E_JOUEUR** , **E_MONSTRE** , **E_PROJECTILE** }
Identifiant permettant de reconnaitre la nature d'une entité.

Fonctions

— int dessinerEntite (t_moteur ∗moteur, t_entite ∗entite)
Dessine l'entité à ses coordonnées, tout en gérant son animation s'il en possède une.
— int deplacerEntite (const t_moteur ∗moteur, t_entite ∗entite)
Déplace l'entité selon sa direction et sa vitesse. L'entité n'est pas avancé si sa future position entraine une collision avec un mur.
— t_entite ∗ creerEntite (float x, float y, SDL_Texture ∗texture)
Génère une entité générique.
— void detruireEntite (t_entite ∗∗entite)
Détruit une entité générique et mets sont pointeur à NULL.

4.4.1 Description détaillée

Module de manipulation des entites.

Pour faciliter le développement, ce module permet l'affichage et le déplacement automatique d'une entité selon ses paramètres.

Auteur

Julien Rouaux

4.4.2 Documentation des fonctions

4.4.2.1 creerEntite()

```
t_entite* creerEntite (
            float x,
            float y,
            SDL_Texture * texture )
```

Génère une entité générique.

Cette fonction est principalement utile pour créer une entité qui hérite des attributs de cette structure.

Paramètres

x	Position en x de l'entité
y	Position en y de l'entité
texture	Apparence de l'entité

Renvoie

L'adresse de l'entité créée.

4.4.2.2 deplacerEntite()

```
int deplacerEntite (
            const t_moteur * moteur,
            t_entite * entite )
```

Déplace l'entité selon sa direction et sa vitesse. L'entité n'est pas avancé si sa future position entraine une collision avec un mur.

La fonction teste si une collision survient en traitant les deux axes de déplacement séparément afin que l'entité puisse glisser contre le mur s'il se dirige en diagonale au lieu de se voir annuler son déplacement.

Paramètres

moteur	Structure moteur du jeu
entite	L'entité à afficher

Renvoie

-1 si collision, sinon 0.

4.4.2.3 dessinerEntite()

```
int dessinerEntite (
            t_moteur * moteur,
            t_entite * entite )
```

Dessine l'entité à ses coordonnées, tout en gérant son animation s'il en possède une.

Paramètres

moteur	Structure moteur du jeu
entite	L'entité à afficher

Renvoie

0 si succès, sinon une valeur négative (SDL_Error() pour connaitre l'erreur).

4.4.2.4 detruireEntite()

```
void detruireEntite (
            t_entite ** entite )
```

Détruit une entité générique et mets sont pointeur à NULL.

Cette fonction détruit aussi l'animation si l'entité en a une.

Paramètres

entite	L'adresse du pointeur sur l'entité

4.5 Référence du fichier include/generation_niveau.h

Librairie de generation.c.

Classes

- struct t_couleurRVB
  Structure qui stocke une couleur décomposée en RVB. Chaque int est compris entre 0 et 255 inclus.
- struct niveau_informations_t

Macros

- #define **LONGUEUR_NIVEAU_MAX** 25
- #define **HAUTEUR_NIVEAU_MAX** 25
- #define **VIDE** 0
- #define **SALLE** -1
- #define **POURCENTAGE_DE_SALLES_GLOBAL** 20
- #define **CHANCE_GEN_SALLE_8_VOISINES_LIBRES** 100
- #define **CHANCE_GEN_SALLE_7_VOISINES_LIBRES** 70
- #define **CHANCE_GEN_SALLE_6_VOISINES_LIBRES** 40
- #define **CHANCE_GEN_SALLE_5_VOISINES_LIBRES** 20
- #define **CHANCE_GEN_SALLE_4_VOISINES_LIBRES** 10
- #define **CHANCE_GEN_SALLE_3_VOISINES_LIBRES** 5
- #define **CHANCE_GEN_SALLE_2_VOISINES_LIBRES** 1
- #define **CHANCE_GEN_SALLE_1_VOISINE_LIBRE** 1
- #define **CHANCE_GEN_SALLE_0_VOISINE_LIBRE** 0
- #define **NOMBRE_VOISINES_DISPO_NOUVELLE_SALLE_MIN** 4
- #define **CHANCE_DE_GENERER_EXTENSION_DE_ID_DE_SALLE** 15

Fonctions

- niveau_informations_t ∗ creer_niveau_info (const char ∗nom_planete)
  Fonction principale : crée le niveau et l'écrit dans une structure.
- void **detruire_niveau_info** (niveau_informations_t ∗∗niveau)

4.5.1 Description détaillée

Librairie de generation.c.

Auteur

Camille REMOUÉ

4.5.2 Documentation des fonctions

4.5.2.1 creer_niveau_info()

```
niveau_informations_t* creer_niveau_info (
            const char * nom_planete )
```

Fonction principale : crée le niveau et l'écrit dans une structure.

Paramètres

nom_planete	Nom associé à un niveau unique : il génère la seed

4.6 Référence du fichier include/joueur.h

Module de gestion du joueur.

```
#include <SDL2/SDL.h>
#include <entite.h>
#include <personnage.h>
#include <attaque.h>
#include <attributs_personnages.h>
```

Classes

- struct t_joueur_flags
  Structure contenant les entrées clavier/souris actives du joueur.
- struct t_joueur
  Structure modélisant le joueur, hérite des attributs d'entité et de personnage.

Macros

- #define PROPORTION_JOUEUR 1.6
- #define VITESSE_JOUEUR_DEFAULT 6.5

Fonctions

- t_joueur * creerJoueur (float x, float y, SDL_Texture *apparence)
  Génère une structure joueur.
- void detruireJoueur (t_joueur **joueur)
  Libère la mémoire allouée au joueur et mets son pointeur à NULL.

4.6.1 Description détaillée

Module de gestion du joueur.

Auteur

Julien Rouaux

4.6.2 Documentation des macros

4.6.2.1 PROPORTION_JOUEUR

```
#define PROPORTION_JOUEUR 1.6
```

Taille du joueur par rapport à la taille d'une tile

4.6.2.2 VITESSE_JOUEUR_DEFAULT

```
#define VITESSE_JOUEUR_DEFAULT 6.5
```

La vitesse du joueur par défaut

4.6.3 Documentation des fonctions

4.6.3.1 creerJoueur()

```
t_joueur* creerJoueur (
            float x,
            float y,
            SDL_Texture * apparence )
```

Génère une structure joueur.

Paramètres

x	Position du joueur en x
y	Position du joueur en y
apparence	Texture du joueur

Renvoie

Le pointeur joueur, NULL si echec.

4.6.3.2 detruireJoueur()

```
void detruireJoueur (
            t_joueur ** joueur )
```

Libère la mémoire allouée au joueur et mets son pointeur à NULL.

Paramètres

joueur	L'adresse du pointeur joueur

4.7 Référence du fichier include/liste.h

Module de gestion d'une liste.

```
#include <stdio.h>
#include <stdlib.h>
#include <entite.h>
```

Classes

- struct element
- struct t_liste

Définitions de type

- typedef struct element **t_element**

Fonctions

- void **init_liste** (t_liste *l)
- int liste_vide (t_liste *l)
  Booleen, vrai si la liste est vide.
- int hors_liste (t_liste *l)
  Booleen, l'element courant est hors liste.
- void en_tete (t_liste *l)
  Met l'element courant au premier element de la liste.
- void en_queue (t_liste *l)
  Met l'element courant au dernier element de la liste.
- void precedent (t_liste *l)
  Met l'element courant a l'element juste avant l'element courant.
- void suivant (t_liste *l)
  Met l'element courant a l'element juste apres l'element courant.
- void valeur_elt (t_liste *l, t_entite **v)
  *Recupere la valeur de l'element de la liste et _entite *.*
- void modif_elt (t_liste *l, t_entite **v)
  Recupere la valeur de l'element de la liste et l'assigne a v.
- void oter_elt (t_liste *l)
  Supprime l'element de la liste a l'endroit ou est l'ec.
- void ajout_droit (t_liste *l, t_entite *v)
  Ajoute a droite de l'element courant une valeur v.
- void ajout_gauche (t_liste *l, t_entite *v)
  Ajoute a gauche de l'element courant une valeur v.

4.7.1 Description détaillée

Module de gestion d'une liste.

Auteur

Guillaume Richard

4.7.2 Documentation des fonctions

4.7.2.1 ajout_droit()

```
void ajout_droit (
            t_liste * l,
            t_entite * v )
```

Ajoute a droite de l'element courant une valeur v.

Paramètres

l	une liste
v	une valeur

4.7.2.2 ajout_gauche()

```
void ajout_gauche (
            t_liste * l,
            t_entite * v )
```

Ajoute a gauche de l'element courant une valeur v.

Paramètres

l	une liste
v	une valeur

4.7.2.3 en_queue()

```
void en_queue (
            t_liste * l )
```

Met l'element courant au dernier element de la liste.

Paramètres

l	une liste

4.7.2.4 en_tete()

```
void en_tete (
            t_liste * l )
```

Met l'element courant au premier element de la liste.

Paramètres

l	une liste

4.7.2.5 hors_liste()

```
int hors_liste (
            t_liste * l )
```

Booleen, l'element courant est hors liste.

Paramètres

l	une liste

Renvoie

int booleen 1 : vrai, 0 : faux

4.7.2.6 liste_vide()

```
int liste_vide (
            t_liste * l )
```

Booleen, vrai si la liste est vide.

Paramètres

l	une liste

Renvoie

int booleen 1 : vrai, 0 : faux

4.7.2.7 modif_elt()

```
void modif_elt (
            t_liste * l,
            t_entite ** v )
```

Recupere la valeur de l'element de la liste et l'assigne a v.

Paramètres

l	une liste
v	une valeur

4.7.2.8 oter_elt()

```
void oter_elt (
            t_liste * l )
```

Supprime l'element de la liste a l'endroit ou est l'ec.

Paramètres

l	une liste

4.7.2.9 precedent()

```
void precedent (
            t_liste * l )
```

Met l'element courant a l'element juste avant l'element courant.

Paramètres

l	une liste

4.7.2.10 suivant()

```
void suivant (
            t_liste * l )
```

Met l'element courant a l'element juste apres l'element courant.

Paramètres

l	une liste

4.8 Référence du fichier include/monstre.h

Module de gestion des monstres.

```
#include <SDL2/SDL.h>
#include <entite.h>
#include <attributs_personnages.h>
```

Classes

— struct s_monstre

Définitions de type

— typedef struct s_monstre **t_monstre**

4.8.1 Description détaillée

Module de gestion des monstres.

Auteur

Julien Rouaux

4.9 Référence du fichier include/moteur.h

Module de chargement d'une fenêtre, du renderer, des textures, d'une caméra et les retournes dans une structure.

```
#include <SDL2/SDL.h>
#include <textures.h>
#include <camera.h>
```

Classes

— struct s_moteur

Macros

— #define NB_FPS 60
— #define **TEMPS_POUR_CHAQUE_SECONDE** ((float)1000/NB_FPS)
— #define VITESSE_TRANSITION 10
— #define NOMBRE_DE_PORTES 4
— #define NB_TILE_LARGEUR 13
— #define NB_TILE_HAUTEUR 7

Définitions de type

— typedef struct s_niveau **t_niveau**
— typedef struct s_moteur **t_moteur**

Fonctions

— t_moteur * chargerMoteur (unsigned int temps)
Charge une fenêtre, un rendu, les textures, et une caméra.
— void detruireMoteur (t_moteur **moteur)
Libère la mémoire allouée pour la structure moteur et mets son pointeur à NULL.
— void updateEchelle (t_moteur *moteur)
Calcule la taille en pixel d'un bloc selon la taille de l'écran et actualise les champs du moteur.

4.9.1 Description détaillée

Module de chargement d'une fenêtre, du renderer, des textures, d'une caméra et les retournes dans une structure.

Le moteur contient aussi les collisions

Auteur

Julien Rouaux

4.9.2 Documentation des macros

4.9.2.1 NB_FPS

```
#define NB_FPS 60
```

Nombre de frames par secondes du jeu

4.9.2.2 NB_TILE_HAUTEUR

```
#define NB_TILE_HAUTEUR 7
```

Surface au sol en hauteur

4.9.2.3 NB_TILE_LARGEUR

```
#define NB_TILE_LARGEUR 13
```

Surface au sol en largeur

4.9.2.4 NOMBRE_DE_PORTES

```
#define NOMBRE_DE_PORTES 4
```

Nombre de portes d'une salle (une par mur)

4.9.2.5 VITESSE_TRANSITION

```
#define VITESSE_TRANSITION 10
```

Vitesse d'une transition exprimée en tile par secondes

4.9.3 Documentation des fonctions

4.9.3.1 chargerMoteur()

```
t_moteur* chargerMoteur (
            unsigned int temps )
```

Charge une fenêtre, un rendu, les textures, et une caméra.

Renvoie

Structure moteur, NULL si échec.

4.9.3.2 detruireMoteur()

```
void detruireMoteur (
            t_moteur ** moteur )
```

Libère la mémoire allouée pour la structure moteur et mets son pointeur à NULL.

Paramètres

moteur	L'adresse du pointeur du moteur.

4.9.3.3 updateEchelle()

```
void updateEchelle (
            t_moteur * moteur )
```

Calcule la taille en pixel d'un bloc selon la taille de l'écran et actualise les champs du moteur.

Paramètres

moteur	La structure moteur du jeu

4.10 Référence du fichier include/niveau.h

Module de chargement d'un niveau en structure interprétable pour le jeu.

```
#include <stdio.h>
#include <SDL2/SDL.h>
#include <moteur.h>
#include <entite.h>
#include <liste.h>
#include <generation_niveau.h>
```

Classes

- struct t_dimensions_salle
  Informations sur le nombre de sous-salles formant la salle.
- struct s_salle
- struct s_niveau

Définitions de type

- typedef struct s_salle **t_salle**
- typedef struct s_niveau **t_niveau**

Énumérations

- enum e_porte { **UP** , **RIGHT** , **DOWN** , **LEFT** }
  Position d'une porte dans une salle.

Fonctions

- void detruireNiveau (t_niveau **niveau)
  Libère la mémoire allouée pour un niveau et mets son pointeur à NULL.
- int lancerNiveau (t_moteur *moteur, niveau_informations_t *info)
  Lance un niveau.
- void arreterNiveau (t_niveau **niveau)
  *Libère la mémoire allouée pour le niveau. Actuellement cette fonction est identique à void detruireNiveau(t_niveau **niveau).*
- void updateNiveau (t_niveau *niveau, float j_x, float j_y, int echelle)
  Actualise la salle chargée du niveau selon l'activité du joueur (lorsqu'il change de salle par exemple).

4.10.1 Description détaillée

Module de chargement d'un niveau en structure interprétable pour le jeu.

Auteur

Julien Rouaux

4.10.2 Documentation des fonctions

4.10.2.1 arreterNiveau()

```
void arreterNiveau (
            t_niveau ** niveau )
```

Libère la mémoire allouée pour le niveau. Actuellement cette fonction est identique à void detruireNiveau(t_niveau **niveau).

Opération à réaliser lorsque l'on quitte un niveau. Des opérations supplémentaires (notamment animations) peuvent être réalisées avant la destruction.

Paramètres

niveau	Le niveau à fermer

4.10.2.2 detruireNiveau()

```
void detruireNiveau (
            t_niveau ** niveau )
```

Libère la mémoire allouée pour un niveau et mets son pointeur à NULL.

Paramètres

niveau	L'adresse du pointeur du niveau

4.10.2.3 lancerNiveau()

```
int lancerNiveau (
            t_moteur * moteur,
            niveau_informations_t * info )
```

Lance un niveau.

Les couleurs du niveau sont aussi chargées

4.10.2.4 updateNiveau()

```
void updateNiveau (
            t_niveau * niveau,
            float j_x,
            float j_y,
            int echelle )
```

Actualise la salle chargée du niveau selon l'activité du joueur (lorsqu'il change de salle par exemple).

Calcule les coordonnées des bords de la salle et change la salle courante du niveau si le joueur dépasse ces limites.

Paramètres

niveau	Le niveau chargé
joueur	Le joueur
echelle	L'échelle du rendu

4.11 Référence du fichier include/partie.h

A FAIRE.

```
#include <moteur.h>
```

Fonctions

— int **chargerPartie** (t_moteur *moteur, int nouvelle_partie)

4.11.1 Description détaillée

A FAIRE.

Auteur

Julien Rouaux

4.12 Référence du fichier include/personnage.h

Module.

```
#include <entite.h>
#include <attributs_personnages.h>
```

Classes

— struct s_personnage

Définitions de type

— typedef struct s_personnage **t_personnage**

4.12.1 Description détaillée

Module.

Auteur

Julien Rouaux

4.13 Référence du fichier include/projectiles.h

Module de définition de projectiles.

```
#include <SDL2/SDL.h>
#include <entite.h>
#include <attributs_entites.h>
```

Classes

— struct s_projectile

Définitions de type

— typedef struct s_projectile **t_projectile**

Énumérations

— enum **e_type_projectile** { **BALLE** , **BOULE_FEU** }

Fonctions

— t_projectile * creerProjectile (e_type_projectile type, float x, float y, float direction_vx, float direction_vy, e_type_entite cible, SDL_Texture *texture)
Création d'un projectile.
— void detruireProjectile (t_projectile **projectile)
Libère la mémoire allouée à une structure projectile.

4.13.1 Description détaillée

Module de définition de projectiles.

Auteur

Julien Rouaux

4.13.2 Documentation des fonctions

4.13.2.1 creerProjectile()

```
t_projectile* creerProjectile (
            e_type_projectile type,
            float x,
            float y,
            float direction_vx,
            float direction_vy,
            e_type_entite cible,
            SDL_Texture * texture )
```

Création d'un projectile.

Paramètres

type	La nature du projectile désiré
x	La position de départ en x
y	La position de départ en y
direction_vx	Sa direction en x
direction_vy	Sa direction en y
cible	Le type de l'entité qui doit subir des dégats s'il est touché
texture	Tileset des projectiles

Renvoie

Le pointeur projectile (NULL si echec).

4.13.2.2 detruireProjectile()

```
void detruireProjectile (
            t_projectile ** projectile )
```

Libère la mémoire allouée à une structure projectile.

Paramètres

projectile	L'adresse du pointeur du projectile

4.14 Référence du fichier include/rendu_niveau.h

Module d'affichage d'un niveau.

```
#include <moteur.h>
#include <niveau.h>
```

Fonctions

— int **afficherNiveau** (t_moteur ∗moteur, float j_x, float j_y)

4.14.1 Description détaillée

Module d'affichage d'un niveau.

Auteur

Julien Rouaux

4.15 Référence du fichier include/sauvegarde.h

Sauvegarde l'etat actuel de certains objets dans un fichier. Recupere la sauvegarde precedente si elle existe et la charge dans les objets appropriés.

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL2/SDL.h>
#include <joueur.h>
#include <entite.h>
#include <animation.h>
#include <generation_niveau.h>
```

Macros

- #define **filename_joueur** "./save/save_joueur.save"
- #define **filename_niveau** "./save/save_niveau.save"

Fonctions

- int chargerSauvegarde (t_joueur *joueur, niveau_informations_t **niveau)
  Fonction de chargement du niveau et du joueur.
- int **sauvegarder** (t_joueur *joueur, niveau_informations_t *niveau)

4.15.1 Description détaillée

Sauvegarde l'etat actuel de certains objets dans un fichier. Recupere la sauvegarde precedente si elle existe et la charge dans les objets appropriés.

Auteur

Guillaume

4.15.2 Documentation des fonctions

4.15.2.1 chargerSauvegarde()

```
int chargerSauvegarde (
            t_joueur * joueur,
            niveau_informations_t ** niveau )
```

Fonction de chargement du niveau et du joueur.

Paramètres

joueur	joueur
niveau	niveau (NULL si l'on ne souhaite pas charger le niveau)

Renvoie

0 si succès, valeur négative si echec.

4.16 Référence du fichier include/textures.h

Module de chargement de textures. Propose aussi quelques outils relatifs aux terxtures.

```
#include <SDL2/SDL.h>
```

Classes

— struct t_textures
Contient toutes les textures du jeu.

Macros

— #define **TAILLE_TILE** 16

Énumérations

— enum t_tile_type {
AUCUN , **SOL** , **MUR** , **PORTE_BAS** ,
PORTE_DROITE , **PORTE_HAUT** , **PORTE_GAUCHE** }
Identifiants des tiles d'un niveau.

Fonctions

— t_textures * chargerTextures (SDL_Renderer *renderer)
Charge les textures du jeu.
— void detruireTextures (t_textures **textures)
Libère la mémoire allouée pour les textures et mets son pointeur à NULL.
— void splitTexture (SDL_Rect *rectangle, int x, int y)
Outil permettant de découper une tileset et récupérer une partie de la texture.
— void tileNiveau (SDL_Rect *rectangle, t_tile_type type)
Permet de récupérer la bonne partie du tileset de niveau en fonction du type de tile désiré.

4.16.1 Description détaillée

Module de chargement de textures. Propose aussi quelques outils relatifs aux terxtures.

Auteur

Julien Rouaux

4.16.2 Documentation des fonctions

4.16.2.1 chargerTextures()

```
t_textures* chargerTextures (
            SDL_Renderer * renderer )
```

Charge les textures du jeu.

Paramètres

renderer	Pointeur du renderer de la fenêtre.

Renvoie

Le pointeur de la structure contenant les références à toutes les textures, NULL si echec.

4.16.2.2 detruireTextures()

```
void detruireTextures (
            t_textures ** textures )
```

Libère la mémoire allouée pour les textures et mets son pointeur à NULL.

Paramètres

textures	La structure des textures du jeu.

4.16.2.3 splitTexture()

```
void splitTexture (
            SDL_Rect * rectangle,
            int x,
            int y )
```

Outil permettant de découper une tileset et récupérer une partie de la texture.

Paramètres

rectangle	Rectangle où stocker le résultat de calcul
x	Position en x de la tile désirée
y	Position en y de la tile désirée

4.16.2.4 tileNiveau()

```
void tileNiveau (
            SDL_Rect * rectangle,
            t_tile_type type )
```

Permet de récupérer la bonne partie du tileset de niveau en fonction du type de tile désiré.

Paramètres

rectangle	Rectangle où stocker le résultat de calcul
type	Type de tile désiré

4.17 Référence du fichier include/window.h

Module de création de la fenêtre du jeu et son rendu.

```
#include <SDL2/SDL.h>
```

Fonctions

- int creerFenetreEtRendu (SDL_Window **window, SDL_Renderer **renderer)
  Génère une fenêtre et son rendu, pour les mettre dans les pointeurs donnés en paramètre.
- void detruireFenetreEtRendu (SDL_Window **window, SDL_Renderer **renderer)
  Libère la mémoire allouée pour la fenêtre et son rendu, et met les pointeurs à NULL.

4.17.1 Description détaillée

Module de création de la fenêtre du jeu et son rendu.

Auteur

Julien Rouaux

4.17.2 Documentation des fonctions

4.17.2.1 creerFenetreEtRendu()

```
int creerFenetreEtRendu (
            SDL_Window ** window,
            SDL_Renderer ** renderer )
```

Génère une fenêtre et son rendu, pour les mettre dans les pointeurs donnés en paramètre.

Paramètres

window	L'adresse du pointeur de fenêtre
renderer	L'adresse du pointeur de rendu

Renvoie

0 si tout s'est bien passé, sinon une valeur négative.

4.17.2.2 detruireFenetreEtRendu()

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

Libère la mémoire allouée pour la fenêtre et son rendu, et met les pointeurs à NULL.

Paramètres

window	L'adresse du pointeur de fenêtre
renderer	L'adresse du pointeur de rendu

4.18 Référence du fichier src/animation.c

Module facilitant la gestion des animations.

```
#include <stdio.h>
#include <stdlib.h>
#include <animation.h>
```

Fonctions

- void updateAnimation (t_animation *animation, unsigned int temps)
  Actualise l'animation si nécéssaire selon la configuration de la structure.
- t_animation * creerAnimation (int vitesse, int nb_textures, int id_max)
  Génère une structure permettant d'animer un objet.
- void detruireAnimation (t_animation **animation)
  Détruit une structure animation et mets son pointeur à NULL.

4.18.1 Description détaillée

Module facilitant la gestion des animations.

Ce module utilise les configurations données dans la structure t_animation pour se charger d'animer une texture avec updateAnimation.

Auteur

Julien Rouaux

4.18.2 Documentation des fonctions

4.18.2.1 creerAnimation()

```
t_animation* creerAnimation (
            int vitesse,
            int nb_textures,
            int id_max )
```

Génère une structure permettant d'animer un objet.

Il est possible qu'une entité ne possède pas d'animation (pointeur NULL). Dans ce cas le champ id_animation permet de sélectionner la bonne partie de la texture.

Paramètres

vitesse	Temps en milisecondes avant le passage à la frame suivante de l'animation
nb_textures	Nombre de frames différentes d'une animation
id_max	Identifiant d'animation le plus élevé

Renvoie

La structure animation ou NULL si échec.

4.18.2.2 detruireAnimation()

```
void detruireAnimation (
            t_animation ** animation )
```

Détruit une structure animation et mets son pointeur à NULL.

Paramètres

animation	L'adresse du pointeur animation

4.18.2.3 updateAnimation()

```
void updateAnimation (
            t_animation * animation,
            unsigned int temps )
```

Actualise l'animation si nécéssaire selon la configuration de la structure.

Paramètres

animation	L'animation à actualiser
temps	Le temps actuel du jeu

4.19 Référence du fichier src/camera.c

Module de calcul de position de caméra pour savoir où dessiner les éléments sur la fenêtre.

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL2/SDL.h>
#include <moteur.h>
#include <camera.h>
```

Fonctions

- t_camera * creerCamera (float x, float y)
  Alloue la mémoire pour une caméra.
- void detruireCamera (t_camera **camera)
  Libère la mémoire allouée à une caméra et mets son pointeur à NULL.
- void updateCamera (t_moteur *moteur, int largeur, int hauteur, int orig_x, int orig_y, float j_x, float j_y)
  Place la caméra au bon endroit selon la configuration de la salle et de la position du joueur. Le sujet de la caméra sera placé au centre.
- void updateFutureCamera (t_moteur *moteur, int largeur, int hauteur, int orig_x, int orig_y, float j_x, float j_y)
  Place la caméra au bon endroit selon la configuration de la salle et de la position du joueur. Le sujet de la caméra sera placé au centre.

4.19.1 Description détaillée

Module de calcul de position de caméra pour savoir où dessiner les éléments sur la fenêtre.

Auteur

Julien Rouaux

4.19.2 Documentation des fonctions

4.19.2.1 creerCamera()

```
t_camera* creerCamera (
            float x,
            float y )
```

Alloue la mémoire pour une caméra.

Paramètres

x	Position de la caméra en x
y	Position de la caméra en y

Renvoie

Le pointeur de la caméra, NULL si échec.

4.19.2.2 detruireCamera()

```
void detruireCamera (
            t_camera ** camera )
```

Libère la mémoire allouée à une caméra et mets son pointeur à NULL.

Paramètres

camera	L'adresse du pointeur de la caméra

4.19.2.3 updateCamera()

```
void updateCamera (
            t_moteur * moteur,
            int largeur,
            int hauteur,
            int orig_x,
            int orig_y,
            float j_x,
            float j_y )
```

Place la caméra au bon endroit selon la configuration de la salle et de la position du joueur. Le sujet de la caméra sera placé au centre.

Les coordonnées de la caméra sont mises à l'échelle du jeu car elles prennent en compte la taille de la fenêtre.

Paramètres

moteur	Moteur du jeu
camera	Camera à actualiser
largeur	Nombre de sous-salles contenues dans la salle en largeur
hauteur	Nombre de sous-salles contenues dans la salle en hauteur
orig_x	Coordonnées d'origine de la salle en x
orig_y	Coordonnées d'origine de la salle en y
j_x	Position en x du joueur relative au niveau
j_y	Position en y du joueur relative au niveau

4.19.2.4 updateFutureCamera()

```
void updateFutureCamera (
            t_moteur * moteur,
            int largeur,
            int hauteur,
            int orig_x,
            int orig_y,
            float j_x,
            float j_y )
```

Place la caméra au bon endroit selon la configuration de la salle et de la position du joueur. Le sujet de la caméra sera placé au centre.

Les coordonnées de la caméra sont mises à l'échelle du jeu car elles prennent en compte la taille de la fenêtre.

Paramètres

moteur	Moteur du jeu
camera	Camera à actualiser
largeur	Nombre de sous-salles contenues dans la salle en largeur
hauteur	Nombre de sous-salles contenues dans la salle en hauteur
orig_x	Coordonnées d'origine de la salle en x
orig_y	Coordonnées d'origine de la salle en y
j_x	Position en x du joueur relative au niveau
j_y	Position en y du joueur relative au niveau

4.20 Référence du fichier src/entite.c

Module de manipulation des entites.

```
#include <math.h>
#include <SDL2/SDL.h>
#include <moteur.h>
#include <textures.h>
#include <animation.h>
#include <entite.h>
#include <liste.h>
#include <niveau.h>
```

Fonctions

- int dessinerEntite (t_moteur ∗moteur, t_entite ∗entite)
  Dessine l'entité à ses coordonnées, tout en gérant son animation s'il en possède une.
- int deplacerEntite (const t_moteur ∗moteur, t_entite ∗entite)
  Déplace l'entité selon sa direction et sa vitesse. L'entité n'est pas avancé si sa future position entraine une collision avec un mur.

— void detruireEntite (t_entite **entite)
Détruit une entité générique et mets sont pointeur à NULL.
— t_entite * creerEntite (float x, float y, SDL_Texture *texture)
Génère une entité générique.

4.20.1 Description détaillée

Module de manipulation des entites.

Pour faciliter le développement, ce module permet l'affichage, l'animation et le déplacement automatique d'une entité selon ses paramètres.

Auteur

Julien Rouaux

4.20.2 Documentation des fonctions

4.20.2.1 creerEntite()

```
t_entite* creerEntite (
            float x,
            float y,
            SDL_Texture * texture )
```

Génère une entité générique.

Cette fonction est principalement utile pour créer une entité qui hérite des attributs de cette structure.

Paramètres

x	Position en x de l'entité
y	Position en y de l'entité
texture	Apparence de l'entité

Renvoie

L'adresse de l'entité créée.

4.20.2.2 deplacerEntite()

```
int deplacerEntite (
            const t_moteur * moteur,
            t_entite * entite )
```

Déplace l'entité selon sa direction et sa vitesse. L'entité n'est pas avancé si sa future position entraine une collision avec un mur.

La fonction teste si une collision survient en traitant les deux axes de déplacement séparément afin que l'entité puisse glisser contre le mur s'il se dirige en diagonale au lieu de se voir annuler son déplacement.

Paramètres

moteur	Structure moteur du jeu
entite	L'entité à afficher

Renvoie

-1 si collision, sinon 0.

4.20.2.3 dessinerEntite()

```
int dessinerEntite (
            t_moteur * moteur,
            t_entite * entite )
```

Dessine l'entité à ses coordonnées, tout en gérant son animation s'il en possède une.

Paramètres

moteur	Structure moteur du jeu
entite	L'entité à afficher

Renvoie

0 si succès, sinon une valeur négative (SDL_Error() pour connaitre l'erreur).

4.20.2.4 detruireEntite()

```
void detruireEntite (
            t_entite ** entite )
```

Détruit une entité générique et mets sont pointeur à NULL.

Cette fonction détruit aussi l'animation si l'entité en a une.

Paramètres

entite	L'adresse du pointeur sur l'entité

4.21 Référence du fichier src/events.c

Module de gestion des evenements (souris/clavier/fenetre)

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL2/SDL.h>
#include <joueur.h>
```

Fonctions

— int handleEvents (t_joueur *joueur)
Gere tout les evenements liés a la souris, au clavier et a la fenetre.

4.21.1 Description détaillée

Module de gestion des evenements (souris/clavier/fenetre)

Auteur

Guillaume

4.21.2 Documentation des fonctions

4.21.2.1 handleEvents()

```
int handleEvents (
            t_joueur * joueur )
```

Gere tout les evenements liés a la souris, au clavier et a la fenetre.

Renvoie

Boleen, vrai si l'utilisateur ferme la fenetre avec la croix, par defaut 0

Boléen, 1 si on scrolle vers le haut, 0 si on scrolle vers le bas

gestion de la souris

Si un des boutons de la souris est appuyé

Bouton gauche

Molette

Bouton droit

Cas d'erreur

Nombre de clics

Clic simple

Double clic

Si plus qu'un double clic

Si un des boutons de la souris est relaché

Bouton gauche

Molette

Bouton droit

Recupere la valeur de la molette, 1 si on scrolle vers le haut, 0 si on scrolle vers le bas

touche relachée

Fenetre montrée

Fenetre cachée

Fenetre maximisée

Fenetre minimisée

4.22 Référence du fichier src/generation_niveau.c

Génération d'un niveau : l'agencement des salles et leurs ids.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <string.h>
#include <ctype.h>
#include <generation_niveau.h>
#include <outils.h>
```

Fonctions

- void **detruire_niveau_info** (niveau_informations_t **niveau)
- niveau_informations_t * creer_niveau_info (const char *nom_planete)
  Fonction principale : crée le niveau et l'écrit dans une structure.

4.22.1 Description détaillée

Génération d'un niveau : l'agencement des salles et leurs ids.

Auteur

Camille REMOUÉ

4.22.2 Documentation des fonctions

4.22.2.1 creer_niveau_info()

```
niveau_informations_t* creer_niveau_info (
            const char * nom_planete )
```

Fonction principale : crée le niveau et l'écrit dans une structure.

Paramètres

nom_planete	Nom associé à un niveau unique : il génère la seed

4.23 Référence du fichier src/joueur.c

Module de gestion du joueur.

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL2/SDL.h>
#include <moteur.h>
#include <niveau.h>
#include <entite.h>
#include <animation.h>
#include <liste.h>
#include <projectiles.h>
#include <joueur.h>
#include <attaque.h>
```

Fonctions

- void detruireJoueur (t_joueur **joueur)
  Libère la mémoire allouée au joueur et mets son pointeur à NULL.
- t_joueur * creerJoueur (float x, float y, SDL_Texture *apparence)
  Génère une structure joueur.

4.23.1 Description détaillée

Module de gestion du joueur.

Auteur

Julien Rouaux

4.23.2 Documentation des fonctions

4.23.2.1 creerJoueur()

```
t_joueur* creerJoueur (
            float x,
            float y,
            SDL_Texture * apparence )
```

Génère une structure joueur.

Paramètres

x	Position du joueur en x
y	Position du joueur en y
apparence	Texture du joueur

Renvoie

Le pointeur joueur, NULL si echec.

4.23.2.2 detruireJoueur()

```
void detruireJoueur (
            t_joueur ** joueur )
```

Libère la mémoire allouée au joueur et mets son pointeur à NULL.

Paramètres

joueur	L'adresse du pointeur joueur

4.24 Référence du fichier src/liste.c

Module de gestion de liste en t_entite.

```
#include <entite.h>
#include <liste.h>
```

Fonctions

- void **init_liste** (t_liste *l)
- int liste_vide (t_liste *l)
  Booleen, vrai si la liste est vide.
- int hors_liste (t_liste *l)
  Booleen, l'element courant est hors liste.
- void en_tete (t_liste *l)
  Met l'element courant au premier element de la liste.
- void en_queue (t_liste *l)
  Met l'element courant au dernier element de la liste.
- void precedent (t_liste *l)
  Met l'element courant a l'element juste avant l'element courant.
- void suivant (t_liste *l)
  Met l'element courant a l'element juste apres l'element courant.
- void valeur_elt (t_liste *l, t_entite **v)
  *Recupere la valeur de l'element de la liste et _entite *.*
- void modif_elt (t_liste *l, t_entite **v)

Recupere la valeur de l'element de la liste et l'assigne a v.

— void oter_elt (t_liste *l)

Supprime l'element de la liste a l'endroit ou est l'ec.

— void ajout_droit (t_liste *l, t_entite *v)

Ajoute a droite de l'element courant une valeur v.

— void ajout_gauche (t_liste *l, t_entite *v)

Ajoute a gauche de l'element courant une valeur v.

4.24.1 Description détaillée

Module de gestion de liste en t_entite.

Auteur

Guillaume Richard

4.24.2 Documentation des fonctions

4.24.2.1 ajout_droit()

```
void ajout_droit (
            t_liste * l,
            t_entite * v )
```

Ajoute a droite de l'element courant une valeur v.

Paramètres

l	une liste
v	une valeur

4.24.2.2 ajout_gauche()

```
void ajout_gauche (
            t_liste * l,
            t_entite * v )
```

Ajoute a gauche de l'element courant une valeur v.

Paramètres

l	une liste
v	une valeur

4.24.2.3 en_queue()

```
void en_queue (
            t_liste * l )
```

Met l'element courant au dernier element de la liste.

Paramètres

l	une liste

4.24.2.4 en_tete()

```
void en_tete (
            t_liste * l )
```

Met l'element courant au premier element de la liste.

Paramètres

l	une liste

4.24.2.5 hors_liste()

```
int hors_liste (
            t_liste * l )
```

Booleen, l'element courant est hors liste.

Paramètres

l	une liste

Renvoie

int booleen 1 : vrai, 0 : faux

4.24.2.6 liste_vide()

```
int liste_vide (
            t_liste * l )
```

Booleen, vrai si la liste est vide.

Paramètres

l	une liste

Renvoie

int booleen 1 : vrai, 0 : faux

4.24.2.7 modif_elt()

```
void modif_elt (
            t_liste * l,
            t_entite ** v )
```

Recupere la valeur de l'element de la liste et l'assigne a v.

Paramètres

l	une liste
v	une valeur

4.24.2.8 oter_elt()

```
void oter_elt (
            t_liste * l )
```

Supprime l'element de la liste a l'endroit ou est l'ec.

Paramètres

l	une liste

4.24.2.9 precedent()

```
void precedent (
            t_liste * l )
```

Met l'element courant a l'element juste avant l'element courant.

Paramètres

l	une liste

4.24.2.10 suivant()

```
void suivant (
            t_liste * l )
```

Met l'element courant a l'element juste apres l'element courant.

Paramètres

l	une liste

4.25 Référence du fichier src/moteur.c

Module de chargement d'une fenêtre, du renderer, des textures, d'une caméra et les retournes dans une structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <moteur.h>
#include <window.h>
#include <textures.h>
#include <camera.h>
```

Fonctions

- t_moteur * chargerMoteur (unsigned int temps)
  Charge une fenêtre, un rendu, les textures, et une caméra.
- void detruireMoteur (t_moteur **moteur)
  Libère la mémoire allouée pour la structure moteur et mets son pointeur à NULL.
- void updateEchelle (t_moteur *moteur)
  Calcule la taille en pixel d'un bloc selon la taille de l'écran et actualise les champs du moteur.

4.25.1 Description détaillée

Module de chargement d'une fenêtre, du renderer, des textures, d'une caméra et les retournes dans une structure.

Le moteur contient aussi les collisions

Auteur

Julien Rouaux

4.25.2 Documentation des fonctions

4.25.2.1 chargerMoteur()

```
t_moteur* chargerMoteur (
            unsigned int temps )
```

Charge une fenêtre, un rendu, les textures, et une caméra.

Renvoie

Structure moteur, NULL si échec.

4.25.2.2 detruireMoteur()

```
void detruireMoteur (
            t_moteur ** moteur )
```

Libère la mémoire allouée pour la structure moteur et mets son pointeur à NULL.

Paramètres

moteur	L'adresse du pointeur du moteur.

4.25.2.3 updateEchelle()

```
void updateEchelle (
            t_moteur * moteur )
```

Calcule la taille en pixel d'un bloc selon la taille de l'écran et actualise les champs du moteur.

Paramètres

moteur	La structure moteur du jeu

4.26 Référence du fichier src/niveau.c

Module de chargement d'un niveau en structure interprétable pour le jeu.

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <moteur.h>
#include <entite.h>
#include <liste.h>
#include <generation_niveau.h>
#include <niveau.h>
```

Fonctions

- void detruireNiveau (t_niveau **niveau)
  Libère la mémoire allouée pour un niveau et mets son pointeur à NULL.
- int lancerNiveau (t_moteur *moteur, niveau_informations_t *info)
  Lance un niveau.
- void arreterNiveau (t_niveau **niveau)
  *Libère la mémoire allouée pour le niveau. Actuellement cette fonction est identique à void detruireNiveau(t_niveau **niveau).*
- void updateNiveau (t_niveau *niveau, float j_x, float j_y, int echelle)
  Actualise la salle chargée du niveau selon l'activité du joueur (lorsqu'il change de salle par exemple).

4.26.1 Description détaillée

Module de chargement d'un niveau en structure interprétable pour le jeu.

Auteur

Julien Rouaux

4.26.2 Documentation des fonctions

4.26.2.1 arreterNiveau()

```
void arreterNiveau (
            t_niveau ** niveau )
```

Libère la mémoire allouée pour le niveau. Actuellement cette fonction est identique à void detruireNiveau(t_niveau **niveau).

Opération à réaliser lorsque l'on quitte un niveau. Des opérations supplémentaires (notamment animations) peuvent être réalisées avant la destruction.

Paramètres

niveau	Le niveau à fermer

4.26.2.2 detruireNiveau()

```
void detruireNiveau (
            t_niveau ** niveau )
```

Libère la mémoire allouée pour un niveau et mets son pointeur à NULL.

Paramètres

niveau	L'adresse du pointeur du niveau

4.26.2.3 lancerNiveau()

```
int lancerNiveau (
            t_moteur * moteur,
            niveau_informations_t * info )
```

Lance un niveau.

Les couleurs du niveau sont aussi chargées

4.26.2.4 updateNiveau()

```
void updateNiveau (
            t_niveau * niveau,
            float j_x,
            float j_y,
            int echelle )
```

Actualise la salle chargée du niveau selon l'activité du joueur (lorsqu'il change de salle par exemple).

Calcule les coordonnées des bords de la salle et change la salle courante du niveau si le joueur dépasse ces limites.

Paramètres

niveau	Le niveau chargé
joueur	Le joueur
echelle	L'échelle du rendu

4.27 Référence du fichier src/outils.c

Bibliothèque de petites fonctions pratiques dans de nombreux cas.

```
#include <stdlib.h>
#include <outils.h>
```

Fonctions

— int de (const int nbFaces)

4.27.1 Description détaillée

Bibliothèque de petites fonctions pratiques dans de nombreux cas.

Auteur

Camille

4.27.2 Documentation des fonctions

4.27.2.1 de()

```
int de (
            const int nbFaces )
```

Lancer de dé avec un nombre de faces personnalisé .

Paramètres

nbFaces	Nombre de faces du dé à lancer

4.28 Référence du fichier src/rendu_niveau.c

Module d'affichage d'un niveau.

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL2/SDL.h>
#include <moteur.h>
#include <textures.h>
#include <niveau.h>
#include <rendu_niveau.h>
```

Fonctions

— int **afficherNiveau** (t_moteur ∗moteur, float j_x, float j_y)

4.28.1 Description détaillée

Module d'affichage d'un niveau.

Auteur

Julien Rouaux

4.29 Référence du fichier src/sauvegarde.c

Sauvegarde l'etat actuel de certains objets dans un fichier. Recupere la sauvegarde precedente si elle existe et la charge dans les objets appropriés.

```
#include <sauvegarde.h>
```

Fonctions

- int chargerSauvegarde (t_joueur *joueur, niveau_informations_t **niveau)
  Fonction de chargement du niveau et du joueur.
- int **sauvegarder** (t_joueur *joueur, niveau_informations_t *niveau)

4.29.1 Description détaillée

Sauvegarde l'etat actuel de certains objets dans un fichier. Recupere la sauvegarde precedente si elle existe et la charge dans les objets appropriés.

Auteur

Guillaume

4.29.2 Documentation des fonctions

4.29.2.1 chargerSauvegarde()

```
int chargerSauvegarde (
            t_joueur * joueur,
            niveau_informations_t ** niveau )
```

Fonction de chargement du niveau et du joueur.

Paramètres

joueur	joueur
niveau	niveau (NULL si l'on ne souhaite pas charger le niveau)

Renvoie

0 si succès, valeur négative si echec.

4.30 Référence du fichier src/textures.c

Module de chargement de textures. Propose aussi quelques outils relatifs aux terxtures.

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL2/SDL.h>
#include <textures.h>
```

Fonctions

- t_textures * chargerTextures (SDL_Renderer *renderer)
  Charge les textures du jeu.
- void detruireTextures (t_textures **textures)
  Libère la mémoire allouée pour les textures et mets son pointeur à NULL.
- void splitTexture (SDL_Rect *rectangle, int x, int y)
  Outil permettant de découper une tileset et récupérer une partie de la texture.
- void tileNiveau (SDL_Rect *rectangle, t_tile_type type)
  Permet de récupérer la bonne partie du tileset de niveau en fonction du type de tile désiré.

4.30.1 Description détaillée

Module de chargement de textures. Propose aussi quelques outils relatifs aux terxtures.

Auteur

Julien Rouaux

4.30.2 Documentation des fonctions

4.30.2.1 chargerTextures()

```
t_textures* chargerTextures (
            SDL_Renderer * renderer )
```

Charge les textures du jeu.

Paramètres

renderer	Pointeur du renderer de la fenêtre.

Renvoie

Le pointeur de la structure contenant les références à toutes les textures, NULL si echec.

4.30.2.2 detruireTextures()

```
void detruireTextures (
            t_textures ** textures )
```

Libère la mémoire allouée pour les textures et mets son pointeur à NULL.

Paramètres

textures	La structure des textures du jeu.

4.30.2.3 splitTexture()

```
void splitTexture (
            SDL_Rect * rectangle,
            int x,
            int y )
```

Outil permettant de découper une tileset et récupérer une partie de la texture.

Paramètres

rectangle	Rectangle où stocker le résultat de calcul
x	Position en x de la tile désirée
y	Position en y de la tile désirée

4.30.2.4 tileNiveau()

```
void tileNiveau (
            SDL_Rect * rectangle,
            t_tile_type type )
```

Permet de récupérer la bonne partie du tileset de niveau en fonction du type de tile désiré.

Paramètres

rectangle	Rectangle où stocker le résultat de calcul
type	Type de tile désiré

4.31 Référence du fichier src/window.c

Module de création de la fenêtre du jeu et son rendu.

```
#include <stdio.h>
#include <SDL2/SDL.h>
```

Fonctions

— int creerFenetreEtRendu (SDL_Window ∗∗window, SDL_Renderer ∗∗renderer)
Génère une fenêtre et son rendu, pour les mettre dans les pointeurs donnés en paramètre.
— void detruireFenetreEtRendu (SDL_Window ∗∗window, SDL_Renderer ∗∗renderer)
Libère la mémoire allouée pour la fenêtre et son rendu, et met les pointeurs à NULL.

4.31.1 Description détaillée

Module de création de la fenêtre du jeu et son rendu.

Auteur

Julien Rouaux

4.31.2 Documentation des fonctions

4.31.2.1 creerFenetreEtRendu()

```
int creerFenetreEtRendu (
            SDL_Window ** window,
            SDL_Renderer ** renderer )
```

Génère une fenêtre et son rendu, pour les mettre dans les pointeurs donnés en paramètre.

Paramètres

window	L'adresse du pointeur de fenêtre
renderer	L'adresse du pointeur de rendu

Renvoie

0 si tout s'est bien passé, sinon une valeur négative.

4.31.2.2 detruireFenetreEtRendu()

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

Libère la mémoire allouée pour la fenêtre et son rendu, et met les pointeurs à NULL.

Paramètres

window	L'adresse du pointeur de fenêtre
renderer	L'adresse du pointeur de rendu

4.32 Référence du fichier test/test_niveau.c

Fichier ayant permis de tester de chargement de niveau en l'affichant dans le terminal.

```
#include <stdio.h>
#include <niveau.h>
```

Fonctions

- void **afficher** (const t_niveau *niveau)
- int **main** (void)

4.32.1 Description détaillée

Fichier ayant permis de tester de chargement de niveau en l'affichant dans le terminal.

CE FICHIER N'EST PLUS A JOUR

Auteur

Julien

Index