Battle-ground

Généré par Doxygen 1.9.2

1 Battle-ground 1
1.1 Portabilité . . . 1
1.2 prérequis . . . 1
1.3 compilation . . . 1
1.3.1 GANTT . . . 1
1.3.1.1 effectif . . . 1
1.3.1.2 previsionnel . . . 1
1.3.2 site web : . . . 1

2 Index des structures de données 3
2.1 Structures de données . . . 3

3 Index des fichiers 5
3.1 Liste des fichiers . . . 5

4 Documentation des structures de données 7
4.1 Référence de la structure def . . . 7
4.1.1 Description détaillée . . . 7
4.1.2 Documentation des champs . . . 7
4.1.2.1 degat . . . 7
4.1.2.2 indice_vie . . . 8
4.1.2.3 nom_fichier . . . 8
4.1.2.4 temps . . . 8
4.1.2.5 x . . . 8
4.1.2.6 y . . . 8
4.2 Référence de la structure entity . . . 8
4.2.1 Description détaillée . . . 9
4.2.2 Documentation des champs . . . 9
4.2.2.1 attaque . . . 9
4.2.2.2 charger_img . . . 9
4.2.2.3 degat . . . 10
4.2.2.4 h . . . 10
4.2.2.5 h_image . . . 10
4.2.2.6 met_a_jour . . . 10
4.2.2.7 montant . . . 10
4.2.2.8 nb_pos . . . 10
4.2.2.9 nb_pos_attaque . . . 11
4.2.2.10 nom_fichier . . . 11
4.2.2.11 nom_fichier_attaque . . . 11
4.2.2.12 pv . . . 11
4.2.2.13 temps . . . 11
4.2.2.14 type . . . 11
4.2.2.15 w . . . 12

4.2.2.16 w_image . . . 12
4.2.2.17 x . . . 12
4.2.2.18 x_barre . . . 12
4.2.2.19 x_image . . . 12
4.2.2.20 y . . . 12
4.2.2.21 y_barre . . . 13
4.2.2.22 y_image . . . 13
4.3 Référence de la structure joue . . . 13
4.3.1 Description détaillée . . . 14
4.3.2 Documentation des champs . . . 14
4.3.2.1 argent . . . 14
4.3.2.2 argent_img . . . 14
4.3.2.3 camp . . . 14
4.3.2.4 create . . . 14
4.3.2.5 def . . . 14
4.3.2.6 nom . . . 15
4.3.2.7 pv . . . 15
4.3.2.8 t . . . 15
4.3.2.9 x_create . . . 15
4.4 Référence de la structure mes . . . 15
4.4.1 Description détaillée . . . 15
4.4.2 Documentation des champs . . . 16
4.4.2.1 nom_fichier . . . 16
4.4.2.2 temps . . . 16
4.4.2.3 x . . . 16
4.4.2.4 y . . . 16
4.5 Référence de la structure tir_s . . . 16
4.5.1 Description détaillée . . . 17
4.5.2 Documentation des champs . . . 17
4.5.2.1 indice_vie . . . 17
4.5.2.2 nom_fichier . . . 17
4.5.2.3 x . . . 17
4.5.2.4 y . . . 17
4.6 Référence de la structure wave . . . 18
4.6.1 Description détaillée . . . 18
4.6.2 Documentation des champs . . . 18
4.6.2.1 ent . . . 18
4.6.2.2 prec . . . 18
4.6.2.3 suiv . . . 18

5 Documentation des fichiers **19**

5.1 Référence du fichier ajout_entites.c . . . 19

5.1.1 Description détaillée . . . 20
5.1.2 Documentation des macros . . . 20
5.1.2.1 TAILLE_FENETRE . . . 20
5.1.2.2 Y_ENTITY . . . 21
5.1.3 Documentation des fonctions . . . 21
5.1.3.1 ajouter_bandit() . . . 21
5.1.3.2 ajouter_fighter() . . . 22
5.1.3.3 ajouter_voisin() . . . 23
5.2 Référence du fichier ajout_entites.h . . . 24
5.2.1 Documentation des fonctions . . . 25
5.2.1.1 ajouter_bandit() . . . 25
5.2.1.2 ajouter_fighter() . . . 26
5.2.1.3 ajouter_voisin() . . . 27
5.3 Référence du fichier animations.c . . . 28
5.3.1 Description détaillée . . . 29
5.3.2 Documentation des macros . . . 29
5.3.2.1 TAILLE_FENETRE . . . 30
5.3.3 Documentation des fonctions . . . 30
5.3.3.1 afficher_survivant() . . . 30
5.3.3.2 animation_attaque() . . . 31
5.3.3.3 animation_tir_gauche() . . . 31
5.4 Référence du fichier animations.h . . . 32
5.4.1 Documentation des fonctions . . . 33
5.4.1.1 afficher_survivant() . . . 33
5.4.1.2 animation_attaque() . . . 34
5.4.1.3 animation_tir_gauche() . . . 35
5.5 Référence du fichier audio.c . . . 35
5.5.1 Documentation des macros . . . 36
5.5.1.1 AUDIO_PATH . . . 36
5.5.1.2 PFLAG . . . 36
5.5.2 Documentation des fonctions . . . 36
5.5.2.1 audio_initialise() . . . 36
5.5.2.2 print_init_flags() . . . 37
5.6 Référence du fichier audio.h . . . 38
5.6.1 Documentation des fonctions . . . 39
5.6.1.1 audio_initialise() . . . 39
5.6.1.2 print_init_flags() . . . 39
5.7 Référence du fichier classique.c . . . 40
5.7.1 Description détaillée . . . 41
5.7.2 Documentation des macros . . . 41
5.7.2.1 TAILLE_FENETRE . . . 41
5.7.2.2 X_DEF . . . 41

5.7.2.3 X_TIR . . . 42
5.7.2.4 Y_DEF . . . 42
5.7.2.5 Y_TIR . . . 42
5.7.3 Documentation des fonctions . . . 42
5.7.3.1 demarrer_classique() . . . 42
5.7.3.2 etat_partie_classique() . . . 43
5.7.3.3 gestion_environnement_classique() . . . 44
5.8 Référence du fichier classique.h . . . 45
5.8.1 Documentation des fonctions . . . 46
5.8.1.1 demarrer_classique() . . . 46
5.9 Référence du fichier interactions.c . . . 47
5.9.1 Description détaillée . . . 48
5.9.2 Documentation des fonctions . . . 49
5.9.2.1 attaque_entites() . . . 49
5.9.2.2 creer_defense() . . . 50
5.9.2.3 creer_tir() . . . 50
5.9.2.4 etat_defense() . . . 51
5.9.2.5 etat_tir() . . . 52
5.9.2.6 fichier_existe() . . . 52
5.9.2.7 gestion_environnement() . . . 53
5.9.2.8 gestion_ligne_entite() . . . 54
5.10 Référence du fichier interactions.h . . . 54
5.10.1 Documentation des fonctions . . . 56
5.10.1.1 attaque_entites() . . . 56
5.10.1.2 creer_defense() . . . 57
5.10.1.3 creer_tir() . . . 57
5.10.1.4 etat_defense() . . . 58
5.10.1.5 etat_tir() . . . 59
5.10.1.6 fichier_existe() . . . 59
5.10.1.7 gestion_environnement() . . . 60
5.10.1.8 gestion_ligne_entite() . . . 61
5.11 Référence du fichier interface.c . . . 61
5.11.1 Description détaillée . . . 63
5.11.2 Documentation des macros . . . 63
5.11.2.1 MENU_X . . . 63
5.11.3 Documentation des fonctions . . . 63
5.11.3.1 charger_image() . . . 63
5.11.3.2 charger_partie_image() . . . 64
5.11.3.3 convert_argent() . . . 65
5.11.3.4 demarrage() . . . 65
5.11.3.5 dessiner_rectangle_plein() . . . 66
5.11.3.6 dessiner_rectangle_vide() . . . 67

5.11.3.7 initialise_jeu() . . . 67
5.11.3.8 menu() . . . 68
5.11.3.9 quit_message() . . . 70
5.11.3.10 select_multi() . . . 70
5.11.3.11 sous_menu_jouer() . . . 71
5.11.4 Documentation des variables . . . 72
5.11.4.1 audio . . . 73
5.12 Référence du fichier interface.h . . . 73
5.12.1 Documentation des fonctions . . . 74
5.12.1.1 charger_image() . . . 74
5.12.1.2 charger_partie_image() . . . 75
5.12.1.3 convert_argent() . . . 76
5.12.1.4 demarrage() . . . 76
5.12.1.5 dessiner_rectangle_plein() . . . 77
5.12.1.6 dessiner_rectangle_vide() . . . 78
5.12.1.7 initialise_jeu() . . . 78
5.12.1.8 menu() . . . 79
5.12.1.9 menu_jouer_difficulte() . . . 81
5.12.1.10 quit_message() . . . 81
5.12.1.11 select_multi() . . . 82
5.12.1.12 sous_menu_jouer() . . . 82
5.13 Référence du fichier jeu.c . . . 84
5.13.1 Description détaillée . . . 85
5.13.2 Documentation des macros . . . 85
5.13.2.1 NB_NIV_SURVIVANT . . . 85
5.13.3 Documentation des fonctions . . . 85
5.13.3.1 jeu_classique() . . . 85
5.13.3.2 jeu_survivant() . . . 86
5.14 Référence du fichier jeu.h . . . 88
5.14.1 Documentation des fonctions . . . 88
5.14.1.1 jeu_classique() . . . 89
5.14.1.2 jeu_survivant() . . . 90
5.15 Référence du fichier main.c . . . 91
5.15.1 Description détaillée . . . 92
5.15.2 Documentation des fonctions . . . 92
5.15.2.1 main() . . . 92
5.16 Référence du fichier README.md . . . 93
5.17 Référence du fichier structures.h . . . 93
5.17.1 Documentation des définitions de type . . . 94
5.17.1.1 defense . . . 94
5.17.1.2 joueur . . . 94
5.17.1.3 message . . . 94

5.17.1.4 tir . . . 95
5.18 Référence du fichier survivant.c . . . 95
5.18.1 Description détaillée . . . 96
5.18.2 Documentation des macros . . . 96
5.18.2.1 argent_joueur . . . 96
5.18.2.2 pv_joueur . . . 96
5.18.2.3 V . . . 97
5.18.2.4 x_def . . . 97
5.18.2.5 y_def . . . 97
5.18.2.6 y_entity . . . 97
5.18.3 Documentation des fonctions . . . 97
5.18.3.1 charger_niveau() . . . 97
5.18.3.2 creer_joueur() . . . 98
5.18.3.3 demarrer_survivant() . . . 99
5.18.3.4 deroulement_vague() . . . 100
5.18.3.5 etat_partie_survivant() . . . 102
5.18.3.6 fin_partie() . . . 103
5.18.3.7 message_box() . . . 104
5.18.3.8 return_message() . . . 104
5.19 Référence du fichier survivant.h . . . 105
5.19.1 Documentation des fonctions . . . 106
5.19.1.1 charger_niveau() . . . 106
5.19.1.2 creer_joueur() . . . 107
5.19.1.3 demarrer_survivant() . . . 108
5.19.1.4 deroulement_vague() . . . 109
5.19.1.5 etat_partie_survivant() . . . 111
5.19.1.6 fin_partie() . . . 112
5.19.1.7 message_box() . . . 113
5.19.1.8 return_message() . . . 113
5.20 Référence du fichier vague.c . . . 114
5.20.1 Description détaillée . . . 115
5.20.2 Documentation des fonctions . . . 115
5.20.2.1 ajouter_entite_survivant() . . . 115
5.20.2.2 charger_img_separees() . . . 116
5.20.2.3 charger_img_sprite() . . . 117
5.20.2.4 charger_img_sprite_reverse() . . . 118
5.20.2.5 compter_elem() . . . 118
5.20.2.6 creer_vague() . . . 119
5.20.2.7 deb_liste_vague() . . . 120
5.20.2.8 fin_liste_survivant() . . . 120
5.20.2.9 liste_vide_survivant() . . . 121
5.20.2.10 met_a_jour_images_separees() . . . 122

5.20.2.11 met_a_jour_images_sprite() . . . 123
5.20.2.12 met_a_jour_img_argent() . . . 123
5.20.2.13 precedent_entite_survivant() . . . 124
5.20.2.14 suivant_entite_survivant() . . . 125
5.20.2.15 supprimer_entite_survivant() . . . 125
5.20.2.16 vider_liste_survivant() . . . 126
5.21 Référence du fichier vague.h . . . 127
5.21.1 Documentation des définitions de type . . . 128
5.21.1.1 entite . . . 128
5.21.1.2 t_wave . . . 129
5.21.2 Documentation des fonctions . . . 129
5.21.2.1 ajouter_entite_survivant() . . . 129
5.21.2.2 charger_img_separees() . . . 130
5.21.2.3 charger_img_sprite() . . . 130
5.21.2.4 charger_img_sprite_reverse() . . . 131
5.21.2.5 compter_elem() . . . 132
5.21.2.6 creer_char() . . . 133
5.21.2.7 creer_vague() . . . 133
5.21.2.8 deb_liste_vague() . . . 133
5.21.2.9 fin_liste_survivant() . . . 134
5.21.2.10 hors_liste_survivant() . . . 135
5.21.2.11 liste_vide_survivant() . . . 135
5.21.2.12 met_a_jour_images_separees() . . . 136
5.21.2.13 met_a_jour_images_sprite() . . . 137
5.21.2.14 met_a_jour_img_argent() . . . 137
5.21.2.15 precedent_entite_survivant() . . . 138
5.21.2.16 suivant_entite_survivant() . . . 139
5.21.2.17 supprimer_entite_survivant() . . . 139
5.21.2.18 vider_liste_survivant() . . . 140

Index **143**

Chapitre 1

Battle-ground

1.1 Portabilité

disponible sous Linux, Windows et MacOS

1.2 prérequis

SDL_image et SDL_Mixer

1.3 compilation

```
git clone https://github.com/Mat7813/Battle-ground
cd Battle-ground
make all
cd bin/
./jeu
```

1.3.1 GANTT

1.3.1.1 effectif

https://docs.google.com/spreadsheets/d/1nGvW8YsbBU_SToFOijIJNXJWGJEKc↩
Oc36ztXyP-i-XU/edit?usp=sharing

1.3.1.2 previsionnel

https://docs.google.com/spreadsheets/d/13q1ZOdnywWDGVXpLznJHODp4Z-5laKuh↩
WFcjdYSvfuU/edit?usp=sharing

1.3.2 site web :

https://mat7813.github.io/Battle-ground/doc/html/index.html

Chapitre 2

Index des structures de données

2.1 Structures de données

Liste des structures de données avec une brève description :

def 7
entity 8
joue 13
mes 15
tir_s 16
wave 18

Chapitre 3

Index des fichiers

3.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

ajout_entites.c
Contient toutes les fonctions qui servent à ajouter des entités (tous les types d'entités) 19
ajout_entites.h 24
animations.c
Contient les fonctions pour gérer toutes les animations du jeu (complète le fichier interface.c). (Déplacement des entités, gestion des tirs des canons etc..) Il s'agit des animations communes 28
animations.h 32
audio.c 35
audio.h 38
classique.c
Contient toutes les fonctions utiles au mode de jeu classique de battle ground (gestion de l'évolution de la partie, implémentation des structures de jeu, etc..) 40
classique.h 45
interactions.c
Contient des fonctions variées utilisées potentiellement par les 2 modes de jeu. des fonctions telles que la gestion de l'environnement(les collisions) 47
interactions.h 54
interface.c
Contient toutes les fonctions utiles à l'interface graphique principale du jeu (gestion des menus, fonction de chargement des images etc...) 61
interface.h 73
jeu.c
Contient les fonctions principales qui lancent les 2 modes de jeu (survvivant et classique) avec quelques autres fonctions diverses 84
jeu.h 88
main.c
Contient le main qui initialise le jeu 91
structures.h 93
survivant.c
Contient toutes les fonctions utiles au mode de jeu survivant de battle ground (gestion de l'évolution de la partie, implémentation des structures de jeu, etc..) 95
survivant.h 105
vague.c
Contient toutes les fonctions relatives à la gestion et à la manipulation des vague d'entités . . 114
vague.h 127

Chapitre 4

Documentation des structures de données

4.1 Référence de la structure def

`#include <structures.h>`

Champs de données

- int indice_vie
- char nom_fichier [100]
- int degat
- int temps
- int x
- int y

4.1.1 Description détaillée

Définition à la ligne 14 du fichier structures.h.

4.1.2 Documentation des champs

4.1.2.1 degat

`int degat`

Définition à la ligne 18 du fichier structures.h.

4.1.2.2 indice_vie

```
int indice_vie
```

Définition à la ligne 16 du fichier structures.h.

4.1.2.3 nom_fichier

```
char nom_fichier[100]
```

Définition à la ligne 17 du fichier structures.h.

4.1.2.4 temps

```
int temps
```

Définition à la ligne 19 du fichier structures.h.

4.1.2.5 x

```
int x
```

Définition à la ligne 20 du fichier structures.h.

4.1.2.6 y

```
int y
```

Définition à la ligne 21 du fichier structures.h.

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

— structures.h

4.2 Référence de la structure entity

```
#include <vague.h>
```

Champs de données

- void(∗ met_a_jour)(struct entity ∗)
- void(∗ charger_img)(struct entity ∗, SDL_Renderer ∗)
- int attaque
- int pv
- int type
- int degat
- int x
- int y
- int x_barre
- int y_barre
- int w
- int h
- int x_image
- int y_image
- int w_image
- int h_image
- int temps
- int nb_pos
- int nb_pos_attaque
- char nom_fichier [100]
- char nom_fichier_attaque [100]
- int montant

4.2.1 Description détaillée

Définition à la ligne 6 du fichier vague.h.

4.2.2 Documentation des champs

4.2.2.1 attaque

```
int attaque
```

Définition à la ligne 10 du fichier vague.h.

4.2.2.2 charger_img

```
void(* charger_img) (struct entity *, SDL_Renderer *)
```

Définition à la ligne 9 du fichier vague.h.

4.2.2.3 degat

```
int degat
```

Définition à la ligne 13 du fichier vague.h.

4.2.2.4 h

```
int h
```

Définition à la ligne 19 du fichier vague.h.

4.2.2.5 h_image

```
int h_image
```

Définition à la ligne 23 du fichier vague.h.

4.2.2.6 met_a_jour

```
void(* met_a_jour) (struct entity *)
```

Définition à la ligne 8 du fichier vague.h.

4.2.2.7 montant

```
int montant
```

Définition à la ligne 29 du fichier vague.h.

4.2.2.8 nb_pos

```
int nb_pos
```

Définition à la ligne 25 du fichier vague.h.

4.2.2.9 nb_pos_attaque

```
int nb_pos_attaque
```

Définition à la ligne 26 du fichier vague.h.

4.2.2.10 nom_fichier

```
char nom_fichier[100]
```

Définition à la ligne 27 du fichier vague.h.

4.2.2.11 nom_fichier_attaque

```
char nom_fichier_attaque[100]
```

Définition à la ligne 28 du fichier vague.h.

4.2.2.12 pv

```
int pv
```

Définition à la ligne 11 du fichier vague.h.

4.2.2.13 temps

```
int temps
```

Définition à la ligne 24 du fichier vague.h.

4.2.2.14 type

```
int type
```

Définition à la ligne 12 du fichier vague.h.

4.2.2.15 w

```
int w
```

Définition à la ligne 18 du fichier vague.h.

4.2.2.16 w_image

```
int w_image
```

Définition à la ligne 22 du fichier vague.h.

4.2.2.17 x

```
int x
```

Définition à la ligne 14 du fichier vague.h.

4.2.2.18 x_barre

```
int x_barre
```

Définition à la ligne 16 du fichier vague.h.

4.2.2.19 x_image

```
int x_image
```

Définition à la ligne 20 du fichier vague.h.

4.2.2.20 y

```
int y
```

Définition à la ligne 15 du fichier vague.h.

4.2.2.21 y_barre

```
int y_barre
```

Définition à la ligne 17 du fichier vague.h.

4.2.2.22 y_image

```
int y_image
```

Définition à la ligne 21 du fichier vague.h.

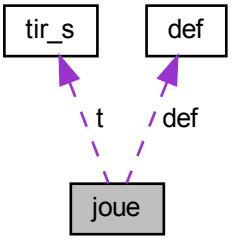
La documentation de cette structure a été générée à partir du fichier suivant :

— vague.h

4.3 Référence de la structure joue

```
#include <structures.h>
```

Graphe de collaboration de joue:



Champs de données

— tir ∗ t
— defense ∗ def
— int argent
— int pv
— int create
— int x_create
— char nom [50]
— char argent_img [15][22]
— int camp

4.3.1 Description détaillée

Définition à la ligne 32 du fichier structures.h.

4.3.2 Documentation des champs

4.3.2.1 argent

```
int argent
```

Définition à la ligne 36 du fichier structures.h.

4.3.2.2 argent_img

```
char argent_img[15][22]
```

Définition à la ligne 41 du fichier structures.h.

4.3.2.3 camp

```
int camp
```

Définition à la ligne 42 du fichier structures.h.

4.3.2.4 create

```
int create
```

Définition à la ligne 38 du fichier structures.h.

4.3.2.5 def

```
defense* def
```

Définition à la ligne 35 du fichier structures.h.

4.3.2.6 nom

```
char nom[50]
```

Définition à la ligne 40 du fichier structures.h.

4.3.2.7 pv

```
int pv
```

Définition à la ligne 37 du fichier structures.h.

4.3.2.8 t

```
tir* t
```

Définition à la ligne 34 du fichier structures.h.

4.3.2.9 x_create

```
int x_create
```

Définition à la ligne 39 du fichier structures.h.

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

4.4 Référence de la structure mes

```
#include <structures.h>
```

Champs de données

- int temps
- char nom_fichier [100]
- int x
- int y

4.4.1 Description détaillée

Définition à la ligne 6 du fichier structures.h.

4.4.2 Documentation des champs

4.4.2.1 nom_fichier

```
char nom_fichier[100]
```

Définition à la ligne 9 du fichier structures.h.

4.4.2.2 temps

```
int temps
```

Définition à la ligne 8 du fichier structures.h.

4.4.2.3 x

```
int x
```

Définition à la ligne 10 du fichier structures.h.

4.4.2.4 y

```
int y
```

Définition à la ligne 11 du fichier structures.h.

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

4.5 Référence de la structure tir_s

```
#include <structures.h>
```

Champs de données

- int indice_vie
- int x
- int y
- char nom_fichier [100]

4.5.1 Description détaillée

Définition à la ligne 24 du fichier structures.h.

4.5.2 Documentation des champs

4.5.2.1 indice_vie

```
int indice_vie
```

Définition à la ligne 26 du fichier structures.h.

4.5.2.2 nom_fichier

```
char nom_fichier[100]
```

Définition à la ligne 29 du fichier structures.h.

4.5.2.3 x

```
int x
```

Définition à la ligne 27 du fichier structures.h.

4.5.2.4 y

```
int y
```

Définition à la ligne 28 du fichier structures.h.

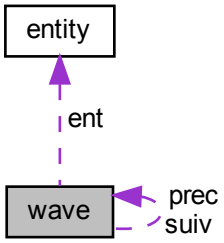
La documentation de cette structure a été générée à partir du fichier suivant :

— structures.h

4.6 Référence de la structure wave

`#include <vague.h>`

Graphe de collaboration de wave:



Champs de données

- entite * ent
- struct wave * suiv
- struct wave * prec

4.6.1 Description détaillée

Définition à la ligne 32 du fichier vague.h.

4.6.2 Documentation des champs

4.6.2.1 ent

`entite* ent`

Définition à la ligne 34 du fichier vague.h.

4.6.2.2 prec

`struct wave* prec`

Définition à la ligne 36 du fichier vague.h.

4.6.2.3 suiv

`struct wave* suiv`

Définition à la ligne 35 du fichier vague.h.

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

- vague.h

Chapitre 5

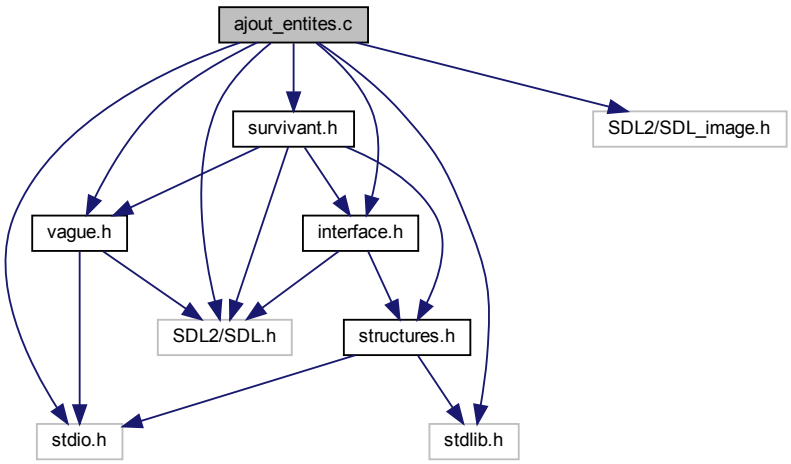
Documentation des fichiers

5.1 Référence du fichier ajout_entites.c

Contient toutes les fonctions qui servent à ajouter des entités (tous les types d'entités)

```
#include <stdio.h>
#include <stdlib.h>
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "interface.h"
#include "vague.h"
#include "survivant.h"
```

Graphe des dépendances par inclusion de ajout_entites.c:



Macros

- #define Y_ENTITY 470
- #define TAILLE_FENETRE 1250

Fonctions

— t_wave ∗ ajouter_voisin (joueur ∗player, t_wave ∗vague, message ∗msg)
fonction qui crée une entité voisin et renvoie le pointeur sur celle-ci
— t_wave ∗ ajouter_bandit (joueur ∗player, t_wave ∗vague, message ∗msg)
fonction qui crée une entité bandit et renvoie le pointeur sur celle-ci
— t_wave ∗ ajouter_fighter (joueur ∗player, t_wave ∗vague, message ∗msg)
fonction qui crée une entité fighter et renvoie le pointeur sur celle-ci

5.1.1 Description détaillée

Contient toutes les fonctions qui servent à ajouter des entités (tous les types d'entités)

Contient les fonctions pour l'audio du jeu (la musique principale qui est jouée en boucle)

Auteur

Lazare Maclouf

Version

2

Date

30/03/2022

Auteur

Matthieu Brière, Lazare Maclouf

Version

1

Date

25/03/2022

5.1.2 Documentation des macros

5.1.2.1 TAILLE_FENETRE

```
#define TAILLE_FENETRE 1250
```

Définition à la ligne 17 du fichier ajout_entites.c.

5.1.2.2 Y_ENTITY

```
#define Y_ENTITY 470
```

Définition à la ligne 16 du fichier ajout_entites.c.

5.1.3 Documentation des fonctions

5.1.3.1 ajouter_bandit()

```
t_wave* ajouter_bandit (
            joueur * player,
            t_wave * vague,
            message * msg )
```

fonction qui crée une entité bandit et renvoie le pointeur sur celle-ci

Paramètres

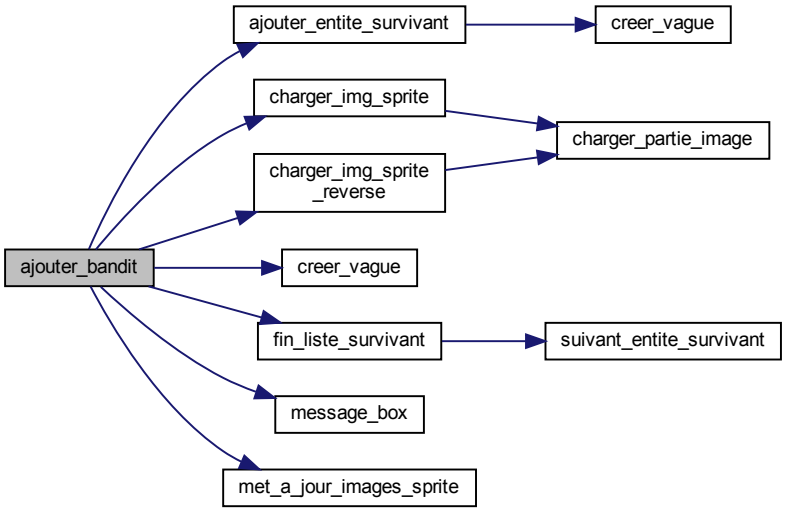
joueur	∗player, t_wave ∗vague, message ∗msg

Renvoie

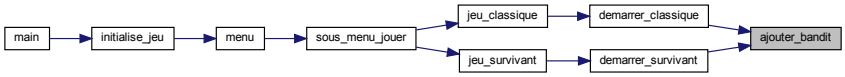
t_wave ∗

Définition à la ligne 120 du fichier ajout_entites.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.1.3.2 ajouter_fighter()

```
t_wave* ajouter_fighter (
            joueur * player,
            t_wave * vague,
            message * msg )
```

fonction qui crée une entité fighter et renvoie le pointeur sur celle-ci

Paramètres

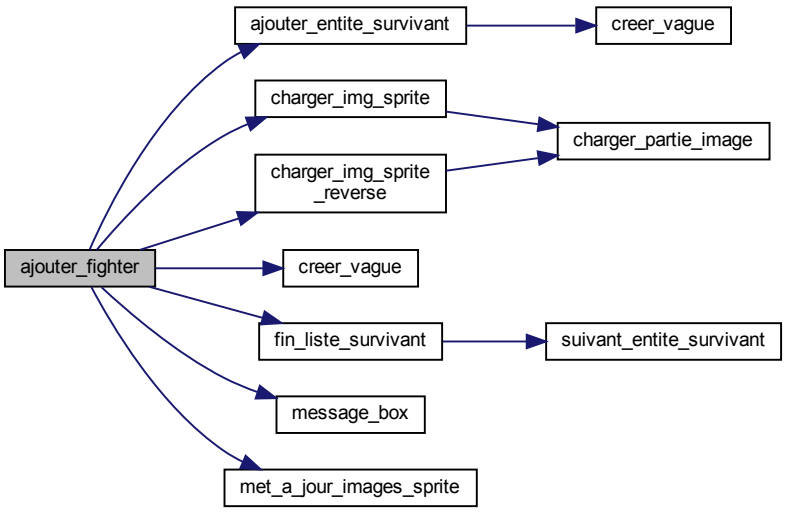
joueur	∗player, t_wave ∗vague, message ∗msg

Renvoie

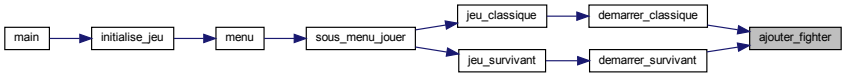
t_wave ∗

Définition à la ligne 219 du fichier ajout_entites.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.1.3.3 ajouter_voisin()

```
t_wave* ajouter_voisin (
            joueur * player,
            t_wave * vague,
            message * msg )
```

fonction qui crée une entité voisin et renvoie le pointeur sur celle-ci

Paramètres

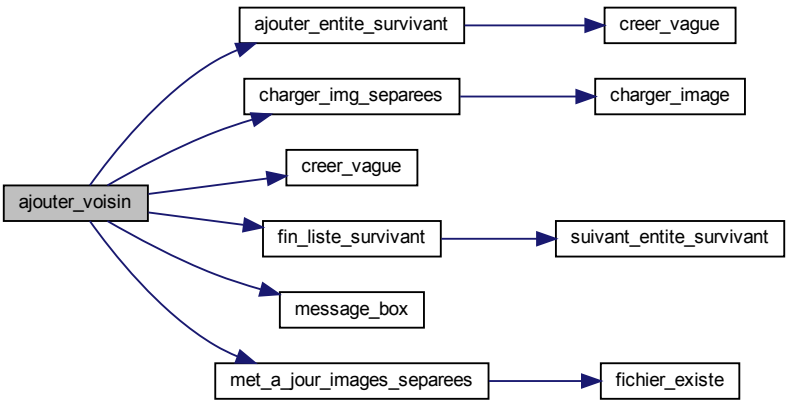
joueur	∗player, t_wave ∗vague, message ∗msg

Renvoie

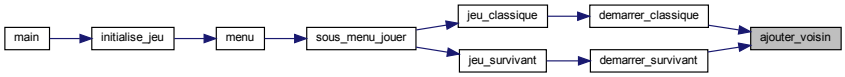
t_wave ∗

Définition à la ligne 25 du fichier ajout_entites.c.

Voici le graphe d'appel pour cette fonction :



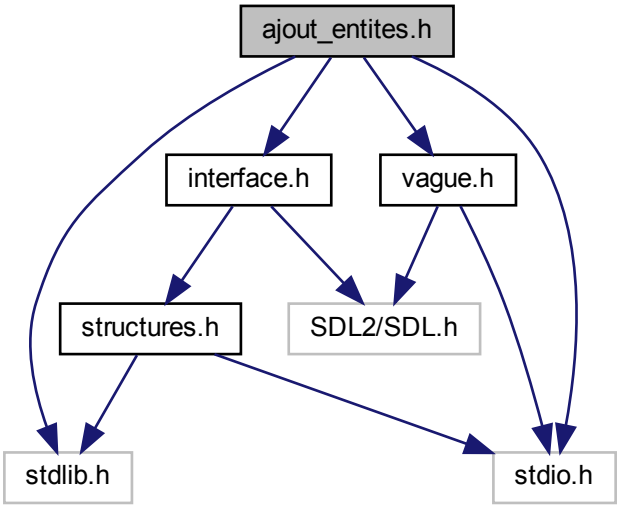
Voici le graphe des appelants de cette fonction :



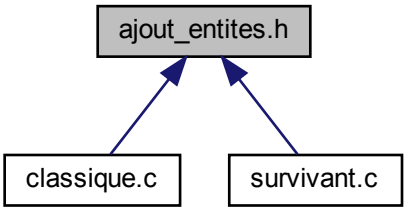
5.2 Référence du fichier ajout_entites.h

```
#include <stdio.h>
#include <stdlib.h>
#include "interface.h"
#include "vague.h"
```

Graphe des dépendances par inclusion de ajout_entites.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- t_wave ∗ ajouter_voisin (joueur ∗player, t_wave ∗vague, message ∗msg)
  fonction qui crée une entité voisin et renvoie le pointeur sur celle-ci
- t_wave ∗ ajouter_bandit (joueur ∗player, t_wave ∗vague, message ∗msg)
  fonction qui crée une entité bandit et renvoie le pointeur sur celle-ci
- t_wave ∗ ajouter_fighter (joueur ∗player, t_wave ∗vague, message ∗msg)
  fonction qui crée une entité fighter et renvoie le pointeur sur celle-ci

5.2.1 Documentation des fonctions

5.2.1.1 ajouter_bandit()

```
t_wave * ajouter_bandit (
            joueur * player,
            t_wave * vague,
            message * msg )
```

fonction qui crée une entité bandit et renvoie le pointeur sur celle-ci

Paramètres

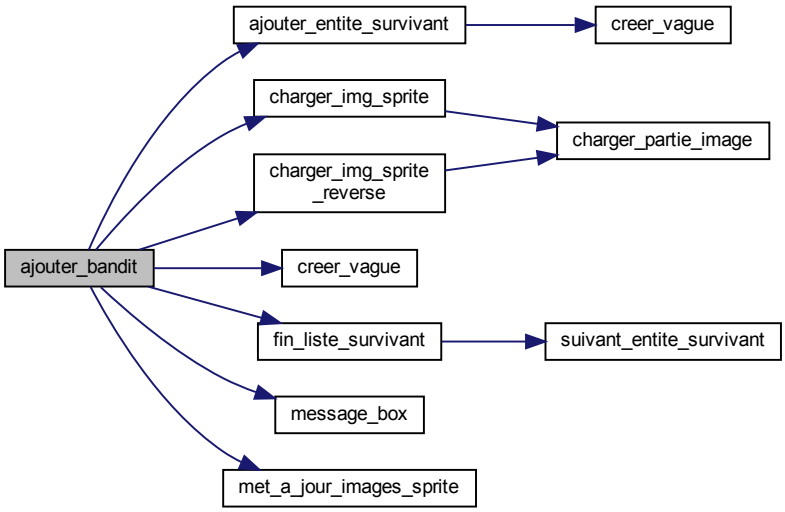
joueur	∗player, t_wave ∗vague, message ∗msg

Renvoie

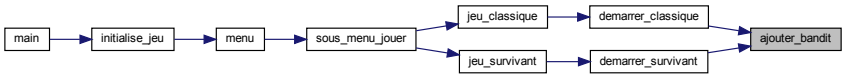
t_wave ∗

Définition à la ligne 120 du fichier ajout_entites.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.2.1.2 ajouter_fighter()

```
t_wave * ajouter_fighter (
            joueur * player,
            t_wave * vague,
            message * msg )
```

fonction qui crée une entité fighter et renvoie le pointeur sur celle-ci

Paramètres

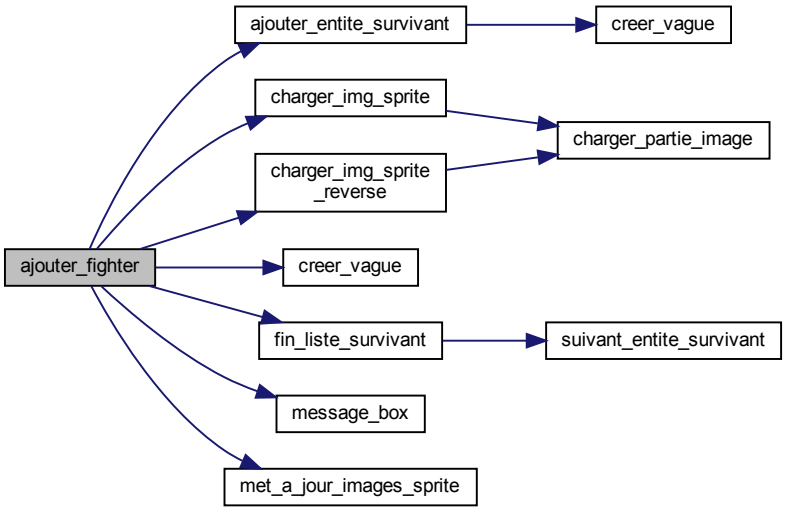
joueur	∗player, t_wave ∗vague, message ∗msg

Renvoie

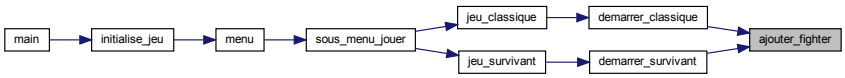
t_wave ∗

Définition à la ligne 219 du fichier ajout_entites.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.2.1.3 ajouter_voisin()

```
t_wave * ajouter_voisin (
            joueur * player,
            t_wave * vague,
            message * msg )
```

fonction qui crée une entité voisin et renvoie le pointeur sur celle-ci

Paramètres

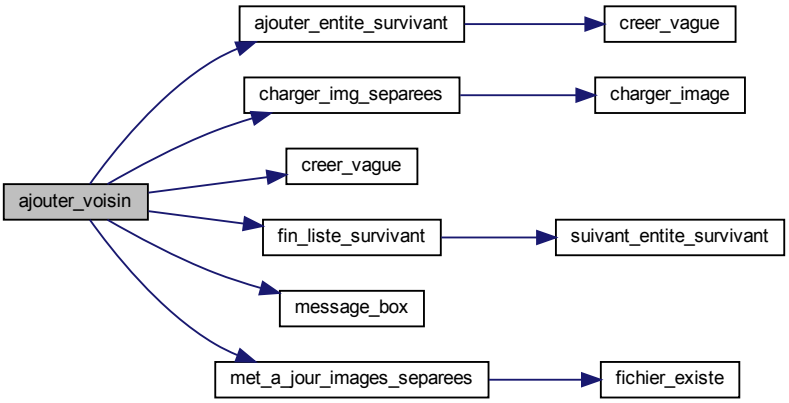
joueur	∗player, t_wave ∗vague, message ∗msg

Renvoie

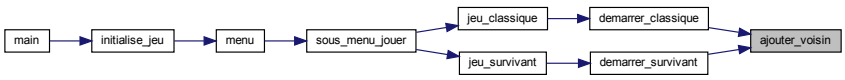
t_wave ∗

Définition à la ligne 25 du fichier ajout_entites.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

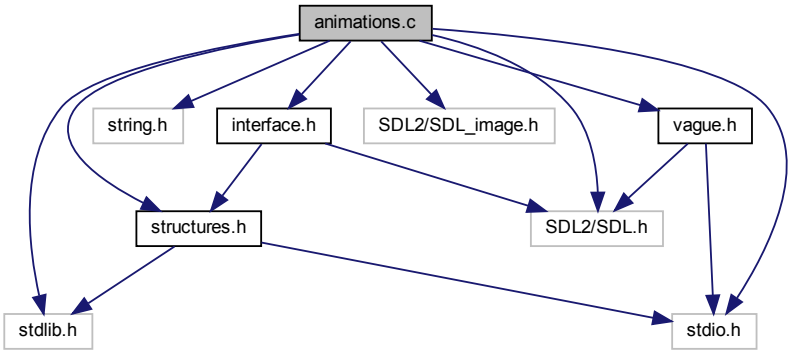


5.3 Référence du fichier animations.c

Contient les fonctions pour gérer toutes les animations du jeu (complète le fichier interface.c). (Déplacement des entités, gestion des tirs des canons etc..) Il s'agit des animations communes.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "interface.h"
#include "structures.h"
#include "vague.h"
```

Graphe des dépendances par inclusion de animations.c:



Macros

— #define TAILLE_FENETRE 1250

Fonctions

— void afficher_survivant (SDL_Renderer ∗rendu, joueur ∗player, int pause)
fonction qui affiche le décors en mode survivant avec la barre de vie des joueurs, son argent, et les menus
— void animation_tir_gauche (SDL_Renderer ∗rendu, joueur ∗player)
fonction qui effectue l'animation d'un tir provenant de gauche jusqu'à sa cible
— void animation_attaque (SDL_Renderer ∗rendu, t_wave ∗vague)
fonction qui met anime l'attaque de certaines entités

5.3.1 Description détaillée

Contient les fonctions pour gérer toutes les animations du jeu (complète le fichier interface.c). (Déplacement des entités, gestion des tirs des canons etc..) Il s'agit des animations communes.

Auteur

Lazare Maclouf

Version

3

Date

25/03/2022

5.3.2 Documentation des macros

5.3.2.1 TAILLE_FENETRE

```
#define TAILLE_FENETRE 1250
```

Définition à la ligne 16 du fichier animations.c.

5.3.3 Documentation des fonctions

5.3.3.1 afficher_survivant()

```
void afficher_survivant (
            SDL_Renderer * rendu,
            joueur * player,
            int pause )
```

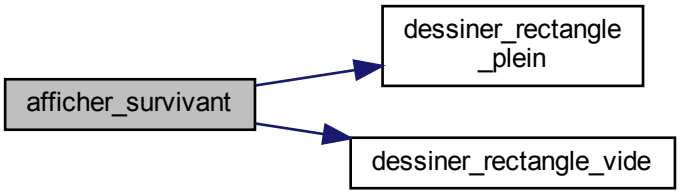
fonction qui affiche le décors en mode survivant avec la barre de vie des joueurs, son argent, et les menus

Paramètres

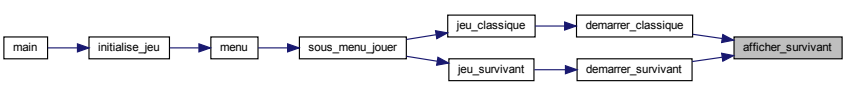
SDL_Renderer	∗rendu, joueur ∗player, int pause

Définition à la ligne 23 du fichier animations.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.3.3.2 animation_attaque()

```
void animation_attaque (
            SDL_Renderer * rendu,
            t_wave * vague )
```

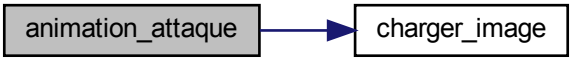
fonction qui met anime l'attaque de certaines entités

Paramètres

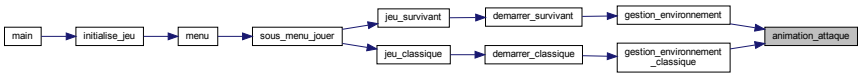
SDL_Renderer	∗rendu, t_wave ∗vague

Définition à la ligne 92 du fichier animations.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.3.3.3 animation_tir_gauche()

```
void animation_tir_gauche (
            SDL_Renderer * rendu,
            joueur * player )
```

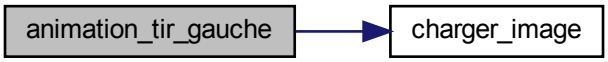
fonction qui effectue l'animation d'un tir provenant de gauche jusqu'à sa cible

Paramètres

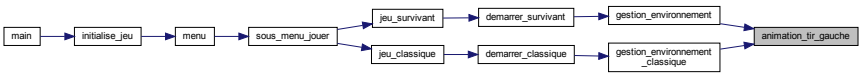
wave↩ _t	∗ vague, SDL_Renderer ∗rendu

Définition à la ligne 75 du fichier animations.c.

Voici le graphe d'appel pour cette fonction :



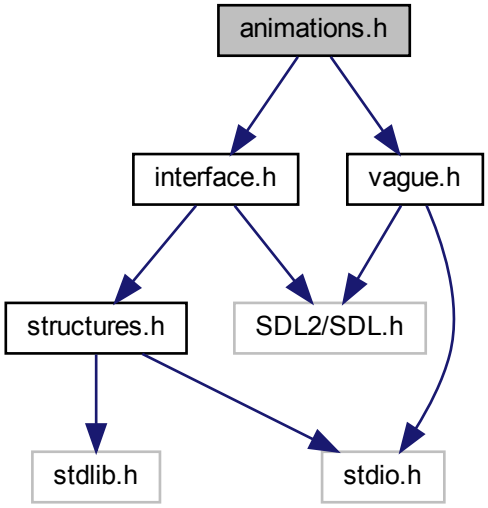
Voici le graphe des appelants de cette fonction :



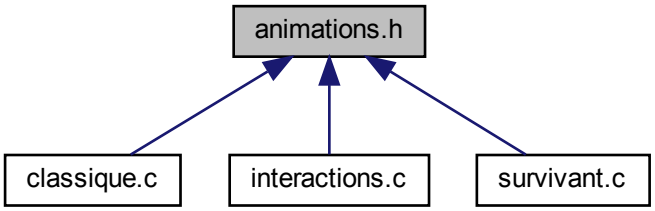
5.4 Référence du fichier animations.h

```
#include "interface.h"
#include "vague.h"
```

Graphe des dépendances par inclusion de animations.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- void afficher_survivant (SDL_Renderer ∗rendu, joueur ∗player, int pause)
  fonction qui affiche le décors en mode survivant avec la barre de vie des joueurs, son argent, et les menus
- void animation_tir_gauche (SDL_Renderer ∗rendu, joueur ∗player)
  fonction qui effectue l'animation d'un tir provenant de gauche jusqu'à sa cible
- void animation_attaque (SDL_Renderer ∗rendu, t_wave ∗vague)
  fonction qui met anime l'attaque de certaines entités

5.4.1 Documentation des fonctions

5.4.1.1 afficher_survivant()

```
void afficher_survivant (
            SDL_Renderer * rendu,
            joueur * player,
            int pause )
```

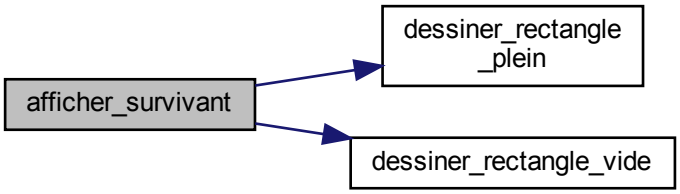
fonction qui affiche le décors en mode survivant avec la barre de vie des joueurs, son argent, et les menus

Paramètres

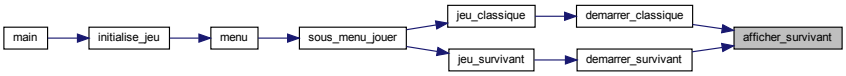
SDL_Renderer	∗rendu, joueur ∗player, int pause

Définition à la ligne 23 du fichier animations.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.4.1.2 animation_attaque()

```
void animation_attaque (
            SDL_Renderer * rendu,
            t_wave * vague )
```

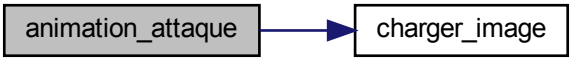
fonction qui met anime l'attaque de certaines entités

Paramètres

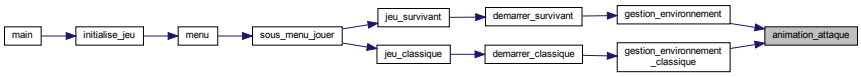
SDL_Renderer	∗rendu, t_wave ∗vague

Définition à la ligne 92 du fichier animations.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.4.1.3 animation_tir_gauche()

```
void animation_tir_gauche (
            SDL_Renderer * rendu,
            joueur * player )
```

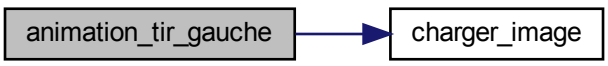
fonction qui effectue l'animation d'un tir provenant de gauche jusqu'à sa cible

Paramètres

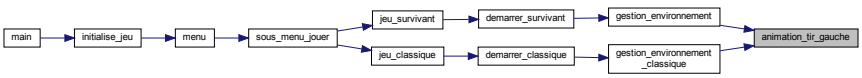
wave↩ _t	∗ vague, SDL_Renderer ∗rendu

Définition à la ligne 75 du fichier animations.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

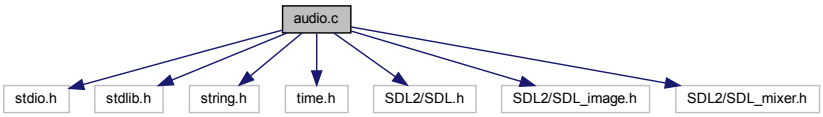


5.5 Référence du fichier audio.c

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

```
#include <string.h>
#include <time.h>
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "SDL2/SDL_mixer.h"
```

Graphe des dépendances par inclusion de audio.c:



Macros

- #define AUDIO_PATH "data/sounds/"
- #define PFLAG(a) if(flags&MIX_INIT_##a) printf(#a " ")

Fonctions

- void print_init_flags (int flags)
  fonction qui affiche le type de fichier audio (mp3, wav, ogg etc...)
- void * audio_initialise (void *v)
  fonction qui lance la musique à l'aide de la librairie SDL_mixer

5.5.1 Documentation des macros

5.5.1.1 AUDIO_PATH

```
#define AUDIO_PATH "data/sounds/"
```

Définition à la ligne 16 du fichier audio.c.

5.5.1.2 PFLAG

```
#define PFLAG(
              a ) if(flags&MIX_INIT_##a) printf(#a " ")
```

5.5.2 Documentation des fonctions

5.5.2.1 audio_initialise()

```
void* audio_initialise (
            void * v )
```

fonction qui lance la musique à l'aide de la librairie SDL_mixer

Paramètres

void	∗v

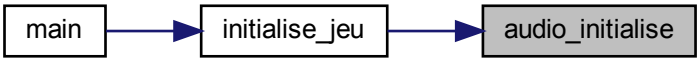
Renvoie

void ∗

Définition à la ligne 43 du fichier audio.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.5.2.2 print_init_flags()

```
void print_init_flags (
            int flags )
```

fonction qui affiche le type de fichier audio (mp3, wav, ogg etc...)

Paramètres

int	flags

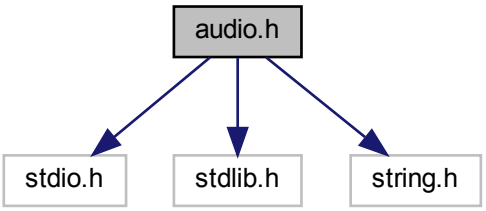
Définition à la ligne 24 du fichier audio.c.

Voici le graphe des appelants de cette fonction :

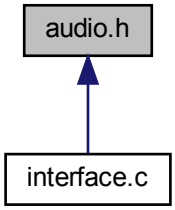
5.6 Référence du fichier audio.h

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

Graphe des dépendances par inclusion de audio.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- void print_init_flags (int flags)
  fonction qui affiche le type de fichier audio (mp3, wav, ogg etc...)
- void * audio_initialise (void *)
  fonction qui lance la musique à l'aide de la librairie SDL_mixer

5.6.1 Documentation des fonctions

5.6.1.1 audio_initialise()

```
void * audio_initialise (
            void * v )
```

fonction qui lance la musique à l'aide de la librairie SDL_mixer

Paramètres

void	∗v

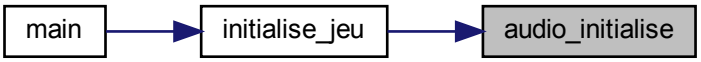
Renvoie

void ∗

Définition à la ligne 43 du fichier audio.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.6.1.2 print_init_flags()

```
void print_init_flags (
            int flags )
```

fonction qui affiche le type de fichier audio (mp3, wav, ogg etc...)

Paramètres

int	flags

Définition à la ligne 24 du fichier audio.c.

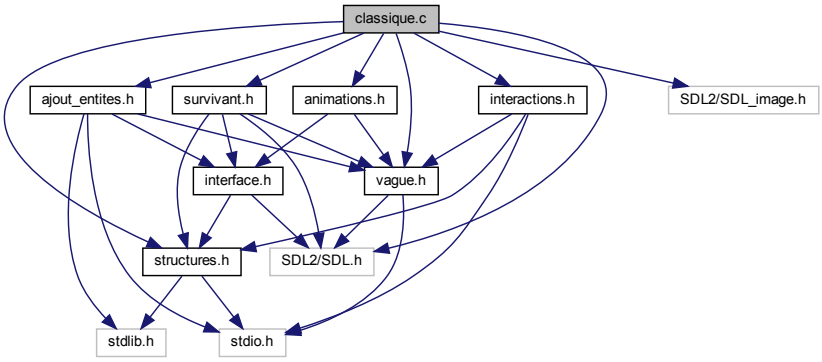
Voici le graphe des appelants de cette fonction :

5.7 Référence du fichier classique.c

Contient toutes les fonctions utiles au mode de jeu classique de battle ground (gestion de l'évolution de la partie, implémentation des structures de jeu, etc..)

```
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "ajout_entites.h"
#include "animations.h"
#include "interactions.h"
#include "structures.h"
#include "vague.h"
#include "survivant.h"
```

Graphe des dépendances par inclusion de classique.c:



Macros

- #define X_DEF 100
- #define Y_DEF 470
- #define TAILLE_FENETRE 1250
- #define Y_TIR 475
- #define X_TIR 165

Fonctions

— void gestion_environnement_classique (t_wave ∗vague_ennemie, t_wave ∗vague_joueur, joueur ∗player, joueur ∗player2, SDL_Renderer ∗rendu)

fonction qui sert à gérer l'environnement et ses interactions avec les entités ainsi que l'interaction des entités elles-mêmes. Une entité s'arrete lorsqu'elle se trouve face à un ennemi en face d'elle et l'attaque. Les autres qui suivent derrière s'arrêtent aussi. Il s'agit de paramétrer et de configurer les limites physiques de l'environnement ainsi que leur conditions.

— int etat_partie_classique (t_wave ∗vague, joueur ∗player, joueur ∗player2)

fonction qui sert à vérifier si la partie est finie ou si elle est toujours en cours (et si elle est finie si le joueur1 a perdu ou le joueur2 a perdu, pour cela, le premier a 0 en point de vie a perdu)

— int demarrer_classique (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event)

cette fonction démarre le mode classique qui est finalement le mode 1 vs 1. C'est la fonction principale du mode jeu classique et s'occupe de tout gérer (les évènements, l'argent et les unités de l'utilisateur; charger le niveau et toutes les entités qui vont avec etc/..)

5.7.1 Description détaillée

Contient toutes les fonctions utiles au mode de jeu classique de battle ground (gestion de l'évolution de la partie, implémentation des structures de jeu, etc..)

Auteur

Geoffrey Posé

Version

3.2.4

Date

31/03/2022

5.7.2 Documentation des macros

5.7.2.1 TAILLE_FENETRE

```
#define TAILLE_FENETRE 1250
```

Définition à la ligne 20 du fichier classique.c.

5.7.2.2 X_DEF

```
#define X_DEF 100
```

Définition à la ligne 18 du fichier classique.c.

5.7.2.3 X_TIR

```
#define X_TIR 165
```

Définition à la ligne 23 du fichier classique.c.

5.7.2.4 Y_DEF

```
#define Y_DEF 470
```

Définition à la ligne 19 du fichier classique.c.

5.7.2.5 Y_TIR

```
#define Y_TIR 475
```

Définition à la ligne 22 du fichier classique.c.

5.7.3 Documentation des fonctions

5.7.3.1 demarrer_classique()

```
int demarrer_classique (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

cette fonction démarre le mode classique qui est finalement le mode 1 vs 1. C'est la fonction principale du mode jeu classique et s'occupe de tout gérer (les évènements, l'argent et les unités de l'utilisateur; charger le niveau et toutes les entités qui vont avec etc/..)

Paramètres

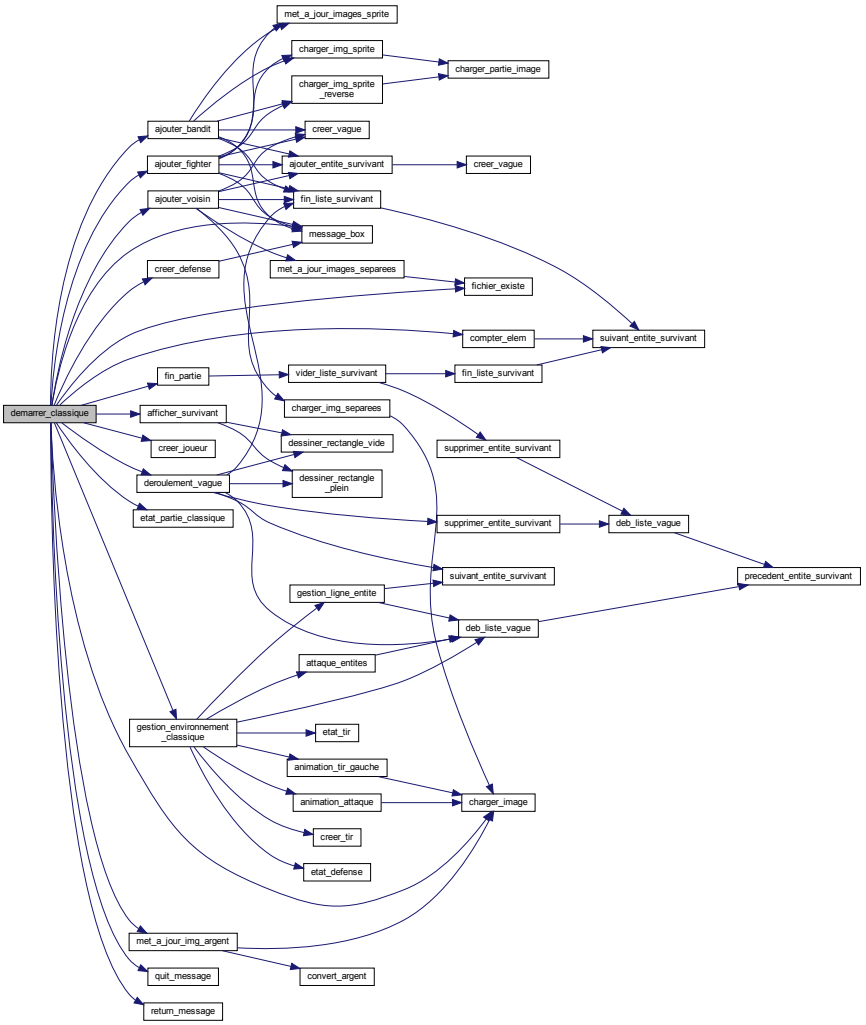
SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event

Renvoie

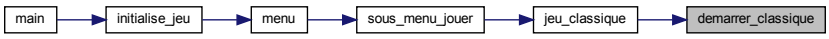
int

Définition à la ligne 117 du fichier classique.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.7.3.2 etat_partie_classique()

```
int etat_partie_classique (
            t_wave * vague,
            joueur * player,
            joueur * player2 )
```

fonction qui sert à vérifier si la partie est finie ou si elle est toujours en cours (et si elle est finie si le joueur1 a perdu ou le joueur2 a perdu, pour cela, le premier a 0 en point de vie a perdu)

Paramètres

t_wave	∗vague, joueur ∗player

Renvoie

int

Définition à la ligne 96 du fichier classique.c.

Voici le graphe des appelants de cette fonction :

5.7.3.3 gestion_environnement_classique()

```
void gestion_environnement_classique (
            t_wave * vague_ennemie,
            t_wave * vague_joueur,
            joueur * player,
            joueur * player2,
            SDL_Renderer * rendu )
```

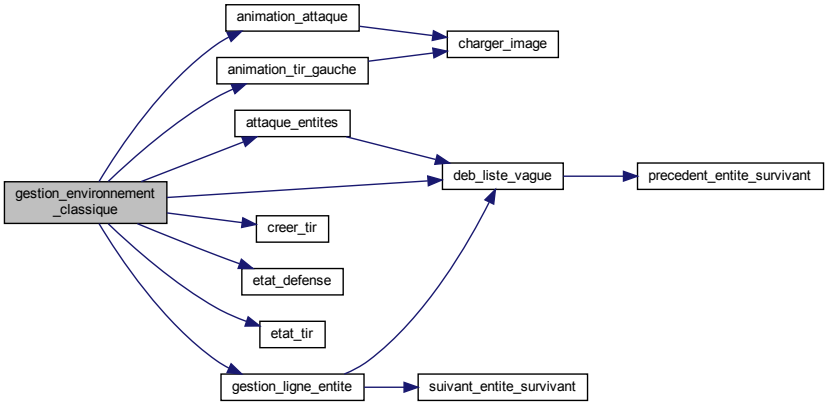
fonction qui sert à gérer l'environnement et ses interactions avec les entités ainsi que l'interaction des entités elles-mêmes. Une entité s'arrete lorsqu'elle se trouve face à un ennemi en face d'elle et l'attaque. Les autres qui suivent derrière s'arrêtent aussi. Il s'agit de paramétrer et de configurer les limites physiques de l'environnement ainsi que leur conditions.

Paramètres

t_wave	∗vague_ennemie, t_wave ∗vague_joueur, joueur ∗player, SDL_Renderer ∗rendu

Définition à la ligne 30 du fichier classique.c.

Voici le graphe d'appel pour cette fonction :

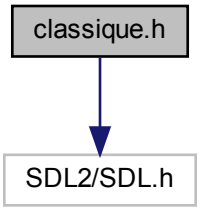


Voici le graphe des appelants de cette fonction :

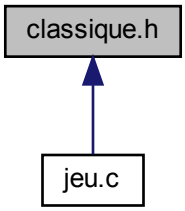
5.8 Référence du fichier classique.h

```
#include "SDL2/SDL.h"
```

Graphe des dépendances par inclusion de classique.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

— int demarrer_classique (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event)

cette fonction démarre le mode classique qui est finalement le mode 1 vs 1. C'est la fonction principale du mode jeu classique et s'occupe de tout gérer (les évènements, l'argent et les unités de l'utilisateur; charger le niveau et toutes les entités qui vont avec etc/..)

5.8.1 Documentation des fonctions

5.8.1.1 demarrer_classique()

```
int demarrer_classique (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

cette fonction démarre le mode classique qui est finalement le mode 1 vs 1. C'est la fonction principale du mode jeu classique et s'occupe de tout gérer (les évènements, l'argent et les unités de l'utilisateur; charger le niveau et toutes les entités qui vont avec etc/..)

Paramètres

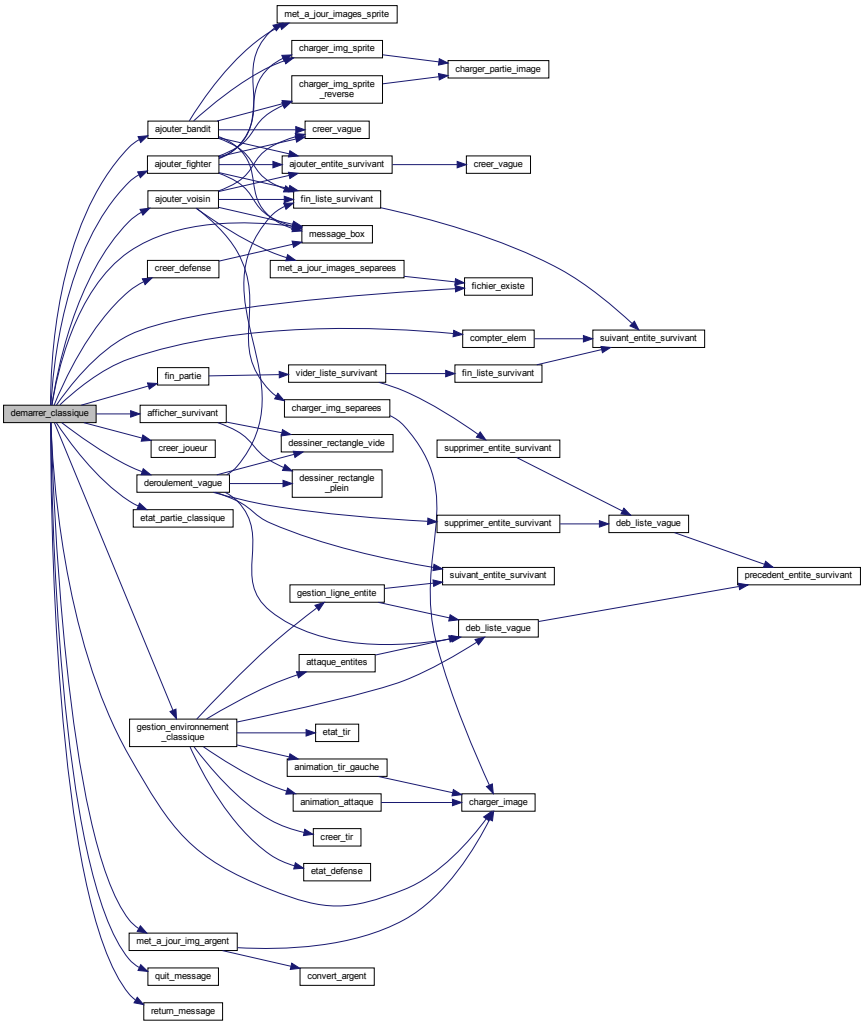
SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event

Renvoie

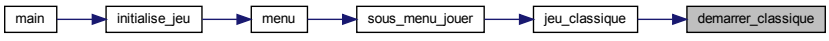
int

Définition à la ligne 117 du fichier classique.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



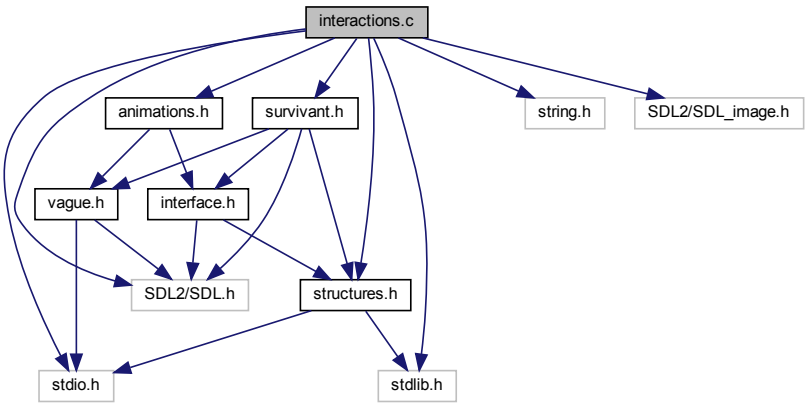
5.9 Référence du fichier interactions.c

Contient des fonctions variées utilisées potentiellement par les 2 modes de jeu. des fonctions telles que la gestion de l'environnement(les collisions).

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

```
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "structures.h"
#include "animations.h"
#include "survivant.h"
```

Graphe des dépendances par inclusion de interactions.c:



Fonctions

- void attaque_entites (t_wave *vague_ennemie, t_wave *vague_joueur, joueur *player)
  fonction qui sert à gérer les attaques des entités entre elles (les dégats qu'elles se causent lorsqu'elles s'affrontent)
- tir * creer_tir (tir *t, int x, int y)
  Sert à créer un tir en allouant la mémoire nécéssaire et en initialisant ses attributs avec les valeurs passées en paramètres.
- void creer_defense (joueur *p, int x, int y, int degat, message *msg)
  Sert à créer une défense en allouant la mémoire nécéssaire et en initialisant ses attributs avec les valeurs passées en paramètres.
- tir * etat_tir (tir *t)
  Sert à mettre à jour un tir. Si il a été en contact avec une entité et lui a fait des dégats (son indice_vie sera à 0). Il sera donc détruit et la fonction retournera NULL.
- defense * etat_defense (defense *def)
  Sert à mettre à jour une défense.
- void gestion_ligne_entite (t_wave *vague, int camp)
  Sert à la gestion des listes chaînées d'entités. Si la première entité de la file s'arrête à cause d'un obstacle les autres derrières suivront et s'arrêteront elles aussi.
- void gestion_environnement (t_wave *vague_ennemie, t_wave *vague_joueur, joueur *player, SDL_↩ Renderer *rendu)
  fonction qui sert à gérer l'environnement et ses interactions avec les entités ainsi que l'interaction des entités elles-mêmes. Une entité s'arrete lorsqu'elle se trouve face à un ennemi en face d'elle et l'attaque. Les autres qui suivent derrière s'arrêtent aussi. Il s'agit de paramétrer et de configurer les limites physiques de l'environnement ainsi que leur conditions.
- int fichier_existe (char *nom)
  fonction qui sert à verifier l'existance d'un fichier dont le nom est passé en paramètre en tentant de l'ouvrir en mode lecture.

5.9.1 Description détaillée

Contient des fonctions variées utilisées potentiellement par les 2 modes de jeu. des fonctions telles que la gestion de l'environnement(les collisions).

Auteur

Lazare Maclouf

Version

1

Date

08/02/2022

5.9.2 Documentation des fonctions

5.9.2.1 attaque_entites()

```
void attaque_entites (
            t_wave * vague_ennemie,
            t_wave * vague_joueur,
            joueur * player )
```

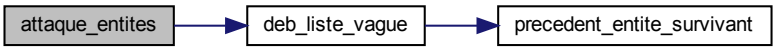
fonction qui sert à gérer les attaques des entités entre elles (les dégats qu'elles se causent lorsqu'elles s'affrontent)

Paramètres

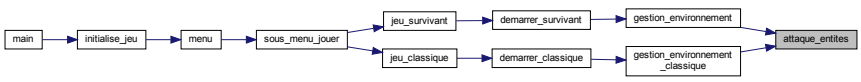
t_wave	∗vague_ennemie, t_wave ∗vague_joueur, joueur ∗player

Définition à la ligne 23 du fichier interactions.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.9.2.2 creer_defense()

```
void creer_defense (
            joueur * p,
            int x,
            int y,
            int degat,
            message * msg )
```

Sert à créer une défense en allouant la mémoire nécéssaire et en initialisant ses attributs avec les valeurs passées en paramètres.

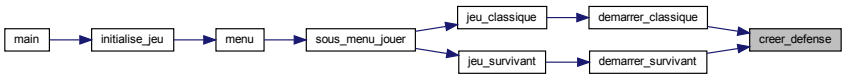
Paramètres

joueur	∗p, int x, int y, int degat, message ∗msg

Définition à la ligne 142 du fichier interactions.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.9.2.3 creer_tir()

```
tir* creer_tir (
            tir * t,
            int x,
            int y )
```

Sert à créer un tir en allouant la mémoire nécéssaire et en initialisant ses attributs avec les valeurs passées en paramètres.

Paramètres

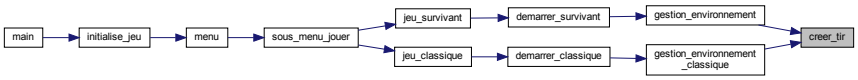
tir	∗t, int x, int y

Renvoie

tir ∗t

Définition à la ligne 128 du fichier interactions.c.

Voici le graphe des appelants de cette fonction :



5.9.2.4 etat_defense()

```
defense* etat_defense (
            defense * def )
```

Sert à mettre à jour une défense.

Paramètres

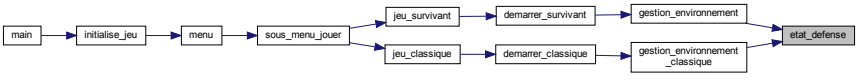
defense	∗def

Renvoie

defense ∗def

Définition à la ligne 196 du fichier interactions.c.

Voici le graphe des appelants de cette fonction :



5.9.2.5 etat_tir()

```
tir* etat_tir (
            tir * t )
```

Sert à mettre à jour un tir. Si il a été en contact avec une entité et lui a fait des dégats (son indice_vie sera à 0). Il sera donc détruit et la fonction retournera NULL.

Paramètres

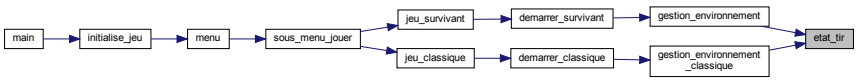
tir	*t

Renvoie

tir *t

Définition à la ligne 176 du fichier interactions.c.

Voici le graphe des appelants de cette fonction :



5.9.2.6 fichier_existe()

```
int fichier_existe (
            char * nom )
```

fonction qui sert à verifier l'existance d'un fichier dont le nom est passé en paramètre en tentant de l'ouvrir en mode lecture.

Paramètres

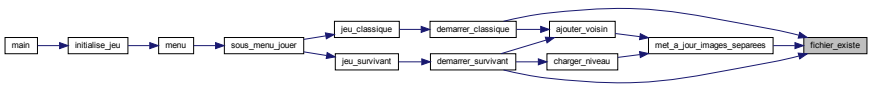
char	*nom

Renvoie

int (0 si il n'existe pas et 1 si il existe)

Définition à la ligne 304 du fichier interactions.c.

Voici le graphe des appelants de cette fonction :



5.9.2.7 gestion_environnement()

```
void gestion_environnement (
            t_wave * vague_ennemie,
            t_wave * vague_joueur,
            joueur * player,
            SDL_Renderer * rendu )
```

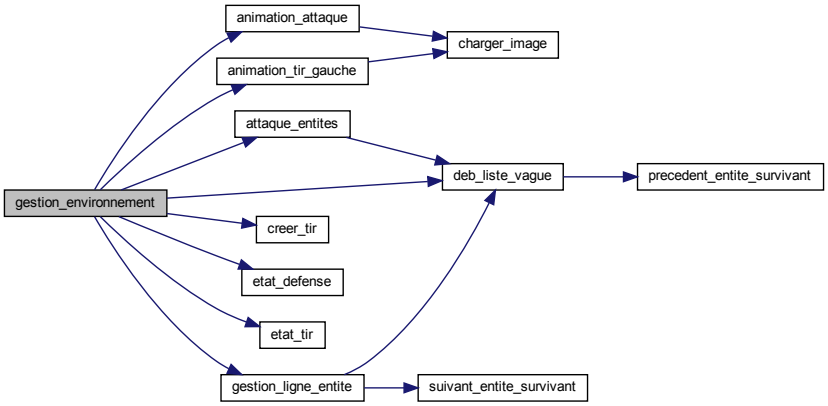
fonction qui sert à gérer l'environnement et ses interactions avec les entités ainsi que l'interaction des entités elles-mêmes. Une entité s'arrete lorsqu'elle se trouve face à un ennemi en face d'elle et l'attaque. Les autres qui suivent derrière s'arrêtent aussi. Il s'agit de paramétrer et de configurer les limites physiques de l'environnement ainsi que leur conditions.

Paramètres

t_wave	∗vague_ennemie, t_wave ∗vague_joueur, joueur ∗player, SDL_Renderer ∗rendu

Définition à la ligne 264 du fichier interactions.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

5.9.2.8 gestion_ligne_entite()

```
void gestion_ligne_entite (
            t_wave * vague,
            int camp )
```

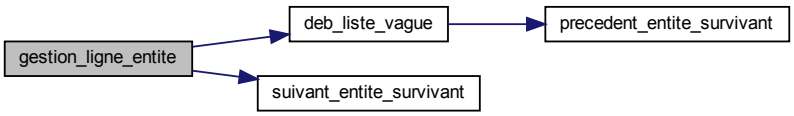
Sert à la gestion des listes chaînées d'entités. Si la première entité de la file s'arrête à cause d'un obstacle les autres derrières suivront et s'arrêteront elles aussi.

Paramètres

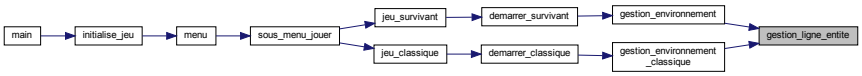
t_wave	*vague, int camp

Définition à la ligne 216 du fichier interactions.c.

Voici le graphe d'appel pour cette fonction :



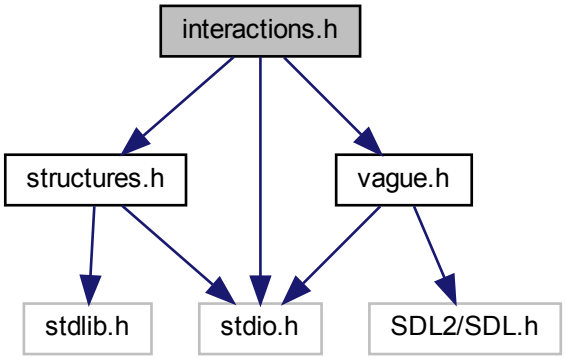
Voici le graphe des appelants de cette fonction :



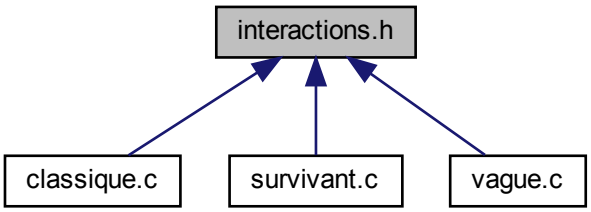
5.10 Référence du fichier interactions.h

```
#include <stdio.h>
#include "structures.h"
#include "vague.h"
```

Graphe des dépendances par inclusion de interactions.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- void attaque_entites (t_wave *vague_ennemie, t_wave *vague_joueur, joueur *player)
  fonction qui sert à gérer les attaques des entités entre elles (les dégats qu'elles se causent lorsqu'elles s'affrontent)
- void gestion_environnement (t_wave *vague_ennemie, t_wave *vague_joueur, joueur *player, SDL_Renderer *rendu)
  fonction qui sert à gérer l'environnement et ses interactions avec les entités ainsi que l'interaction des entités elles-mêmes. Une entité s'arrete lorsqu'elle se trouve face à un ennemi en face d'elle et l'attaque. Les autres qui suivent derrière s'arrêtent aussi. Il s'agit de paramétrer et de configurer les limites physiques de l'environnement ainsi que leur conditions.
- tir * creer_tir (tir *t, int x, int y)
  Sert à créer un tir en allouant la mémoire nécéssaire et en initialisant ses attributs avec les valeurs passées en paramètres.
- tir * etat_tir (tir *t)
  Sert à mettre à jour un tir. Si il a été en contact avec une entité et lui a fait des dégats (son indice_vie sera à 0). Il sera donc détruit et la fonction retournera NULL.
- void creer_defense (joueur *p, int x, int y, int degat, message *msg)

Sert à créer une défense en allouant la mémoire nécéssaire et en initialisant ses attributs avec les valeurs passées en paramètres.

— defense * etat_defense (defense *def)

Sert à mettre à jour une défense.

— void gestion_ligne_entite (t_wave *vague, int camp)

Sert à la gestion des listes chaînées d'entités. Si la première entité de la file s'arrête à cause d'un obstacle les autres derrières suivront et s'arrêteront elles aussi.

— int fichier_existe (char *nom)

fonction qui sert à verifier l'existance d'un fichier dont le nom est passé en paramètre en tentant de l'ouvrir en mode lecture.

5.10.1 Documentation des fonctions

5.10.1.1 attaque_entites()

```
void attaque_entites (
            t_wave * vague_ennemie,
            t_wave * vague_joueur,
            joueur * player )
```

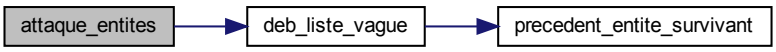
fonction qui sert à gérer les attaques des entités entre elles (les dégats qu'elles se causent lorsqu'elles s'affrontent)

Paramètres

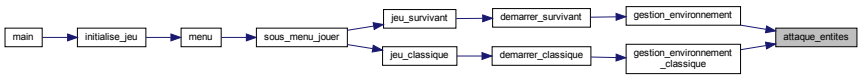
t_wave	*vague_ennemie, t_wave *vague_joueur, joueur *player

Définition à la ligne 23 du fichier interactions.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.10.1.2 creer_defense()

```
void creer_defense (
            joueur * p,
            int x,
            int y,
            int degat,
            message * msg )
```

Sert à créer une défense en allouant la mémoire nécéssaire et en initialisant ses attributs avec les valeurs passées en paramètres.

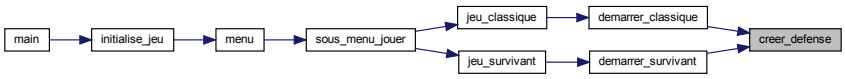
Paramètres

joueur	∗p, int x, int y, int degat, message ∗msg

Définition à la ligne 142 du fichier interactions.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.10.1.3 creer_tir()

```
tir *t creer_tir (
            tir * t,
            int x,
            int y )
```

Sert à créer un tir en allouant la mémoire nécéssaire et en initialisant ses attributs avec les valeurs passées en paramètres.

Paramètres

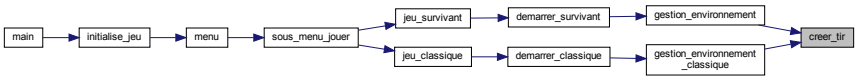
tir	∗t, int x, int y

Renvoie

tir ∗t

Définition à la ligne 128 du fichier interactions.c.

Voici le graphe des appelants de cette fonction :



5.10.1.4 etat_defense()

```
defense * etat_defense (
            defense * def )
```

Sert à mettre à jour une défense.

Paramètres

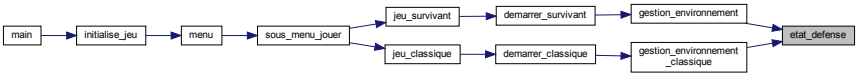
defense	∗def

Renvoie

defense ∗def

Définition à la ligne 196 du fichier interactions.c.

Voici le graphe des appelants de cette fonction :



5.10.1.5 etat_tir()

```
tir *t etat_tir (
            tir * t )
```

Sert à mettre à jour un tir. Si il a été en contact avec une entité et lui a fait des dégats (son indice_vie sera à 0). Il sera donc détruit et la fonction retournera NULL.

Paramètres

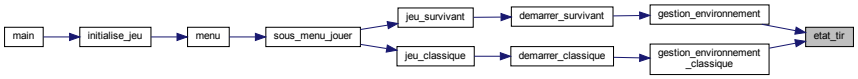
tir	*t

Renvoie

tir *t

Définition à la ligne 176 du fichier interactions.c.

Voici le graphe des appelants de cette fonction :



5.10.1.6 fichier_existe()

```
int fichier_existe (
            char * nom )
```

fonction qui sert à verifier l'existance d'un fichier dont le nom est passé en paramètre en tentant de l'ouvrir en mode lecture.

Paramètres

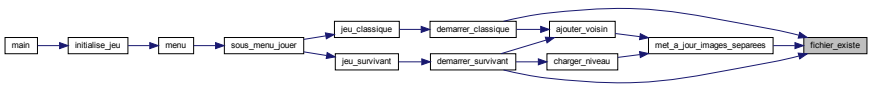
char	*nom

Renvoie

int (0 si il n'existe pas et 1 si il existe)

Définition à la ligne 304 du fichier interactions.c.

Voici le graphe des appelants de cette fonction :



5.10.1.7 gestion_environnement()

```
void gestion_environnement (
            t_wave * vague_ennemie,
            t_wave * vague_joueur,
            joueur * player,
            SDL_Renderer * rendu )
```

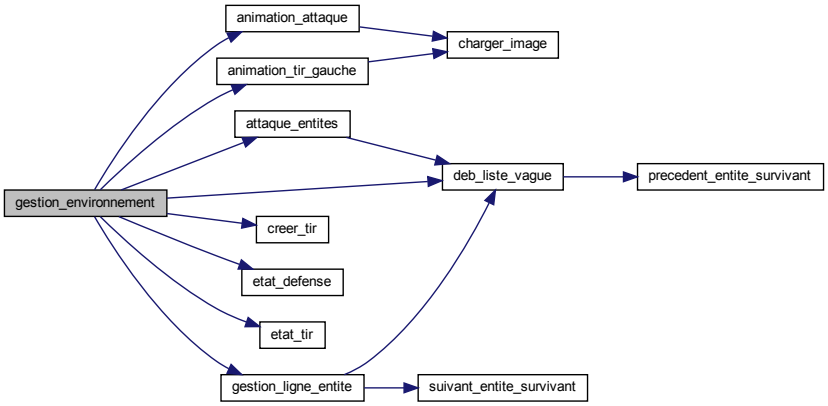
fonction qui sert à gérer l'environnement et ses interactions avec les entités ainsi que l'interaction des entités elles-mêmes. Une entité s'arrete lorsqu'elle se trouve face à un ennemi en face d'elle et l'attaque. Les autres qui suivent derrière s'arrêtent aussi. Il s'agit de paramétrer et de configurer les limites physiques de l'environnement ainsi que leur conditions.

Paramètres

t_wave	∗vague_ennemie, t_wave ∗vague_joueur, joueur ∗player, SDL_Renderer ∗rendu

Définition à la ligne 264 du fichier interactions.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

5.10.1.8 gestion_ligne_entite()

```
gestion_ligne_entite (
            t_wave * vague,
            int camp )
```

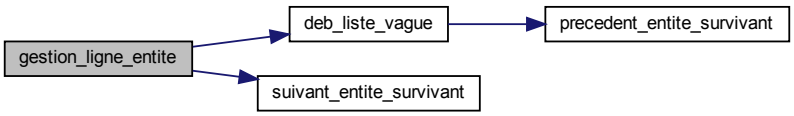
Sert à la gestion des listes chaînées d'entités. Si la première entité de la file s'arrête à cause d'un obstacle les autres derrières suivront et s'arrêteront elles aussi.

Paramètres

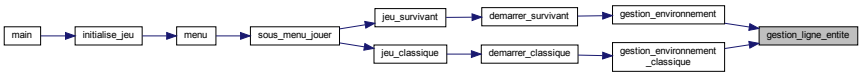
t_wave	∗vague, int camp

Définition à la ligne 216 du fichier interactions.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

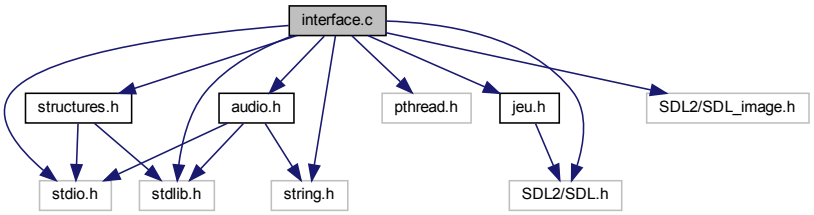


5.11 Référence du fichier interface.c

Contient toutes les fonctions utiles à l'interface graphique principale du jeu (gestion des menus, fonction de chargement des images etc...)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include "audio.h"
#include "jeu.h"
#include "structures.h"
#include "SDL2/SDL.h"
```

`#include "SDL2/SDL_image.h"`
Graphe des dépendances par inclusion de interface.c:



Macros

— #define MENU_X 400

Fonctions

— void convert_argent (joueur *p)
Fonction qui sert à savoir quels chiffres composent le budget du joueur pour les afficher à l'écran (pour 1000 par exemple on aura 1.png, 0.png, 0.png, 0.png)
— int quit_message (void)
affiche un message qui demande si on veut quitter la partie
— void dessiner_rectangle_vide (SDL_Renderer *rendu, int code_couleur[4], int x, int y, int h, int w, int mode)
dessine un rectangle vide
— void dessiner_rectangle_plein (SDL_Renderer *rendu, int code_couleur[4], int x, int y, int h, int w, int mode)
dessine un rectangle plein
— int charger_image (char *nom, SDL_Renderer *rendu, int x, int y, int option)
sert à charger une image avec tous les formats (grâce à SDL_image) sur le rendu et à présenter le rendu sur la fenetre (le format géré par défaut par la SDL est BMP)
— int charger_partie_image (char *nom, SDL_Renderer *rendu, int x, int y, int w, int h, int w_image, int h_image, int x_image, int y_image, int option)
sert à charger un morceau d'une image (la taille de la figure à rogner, sa position et autres passés en paramètre) avec tous les formats (grâce à SDL_image) sur le rendu et à présenter le rendu sur la fenetre (le format géré par défaut par la SDL est BMP)
— void demarrage (SDL_Renderer *rendu)
Fonction servant d'introduction au jeu en faisant une animation à l'écran.
— int select_multi (SDL_Window *window, SDL_Renderer *rendu, SDL_Event *event)
fonction servant à afficher la sélection de l'hote ou du client.
— int sous_menu_jouer (SDL_Window *window, SDL_Renderer *rendu, SDL_Event *event)
fonction servant à afficher les options de jeu et à charger les autres fonctions en conséquence (mode classique, mode suvivant ou mode multi-joueur).
— void menu (SDL_Window *window, SDL_Renderer *rendu)
fonction servant à afficher le menu principal du jeu avec ses différentes options (jouer, paramètres, quitter) et à agir en fonction du choix de l'utilisateur.
— int initialise_jeu ()
fonction principale du jeu. Elle est chargée de la Création et du lancement de la fenêtre du jeu, du rendu ainsi que de l'appel des autres fonctions pour jouer. Enfin, elle s'occupe de détruire le rendu, la fenêtre et de quitter la sdl une fois sorti des fonction de jeu (juste avant de quitter le programme)

Variables

— pthread_t audio

5.11.1 Description détaillée

Contient toutes les fonctions utiles à l'interface graphique principale du jeu (gestion des menus, fonction de chargement des images etc...)

Auteur

Lazare Maclouf

Version

1

Date

22/01/2022

5.11.2 Documentation des macros

5.11.2.1 MENU_X

```
#define MENU_X 400
```

Définition à la ligne 20 du fichier interface.c.

5.11.3 Documentation des fonctions

5.11.3.1 charger_image()

```
int charger_image (
            char * nom,
            SDL_Renderer * rendu,
            int x,
            int y,
            int option )
```

sert à charger une image avec tous les formats (grâce à SDL_image) sur le rendu et à présenter le rendu sur la fenetre (le format géré par défaut par la SDL est BMP)

Paramètres

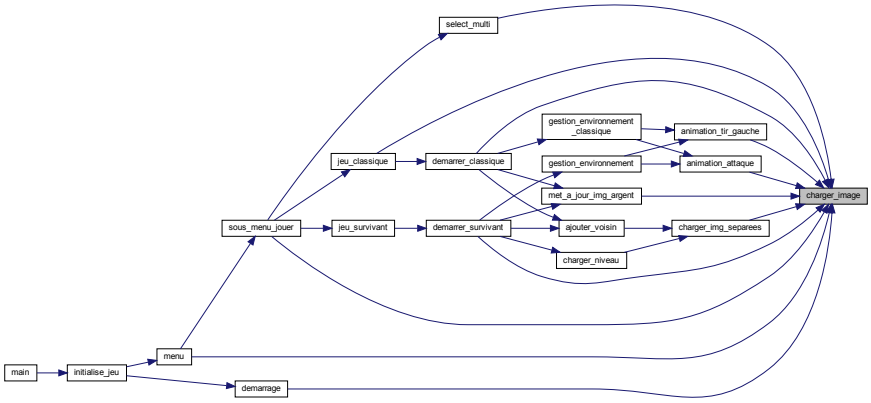
char	∗nom, SDL_Renderer ∗rendu, int x, int y, int option

Renvoie

-1 si un problème est survenu rien sinon

Définition à la ligne 219 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.11.3.2 charger_partie_image()

```
int charger_partie_image (
            char * nom,
            SDL_Renderer * rendu,
            int x,
            int y,
            int w,
            int h,
            int w_image,
            int h_image,
            int x_image,
            int y_image,
            int option )
```

sert à charger un morceau d'une image (la taille de la figure à rogner, sa position et autres passés en paramètre) avec tous les formats (grâce à SDL_image) sur le rendu et à présenter le rendu sur la fenetre (le format géré par défaut par la SDL est BMP)

Paramètres

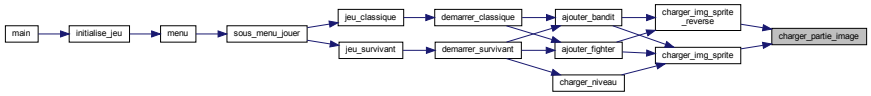
char	∗nom, SDL_Renderer ∗rendu, int x, int y, int w, int h, int w_image, int h_image, int x_image, int y_image, int option

Renvoie

-1 si un problème est survenu rien sinon

Définition à la ligne 248 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.11.3.3 convert_argent()

```
void convert_argent (
            joueur * p )
```

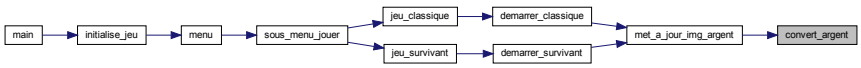
Fonction qui sert à savoir quels chiffres composent le budget du joueur pour les afficher à l'écran (pour 1000 par exemple on aura 1.png, 0.png, 0.png, 0.png)

Paramètres

joueur	∗p

Définition à la ligne 26 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.11.3.4 demarrage()

```
void demarrage (
            SDL_Renderer * rendu )
```

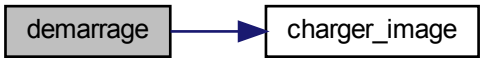
Fonction servant d'introduction au jeu en faisant une animation à l'écran.

Paramètres

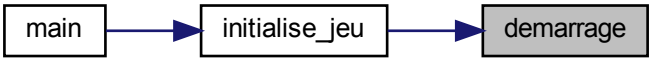
SDL_Renderer	∗rendu

Définition à la ligne 303 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.11.3.5 dessiner_rectangle_plein()

```
void dessiner_rectangle_plein (
            SDL_Renderer * rendu,
            int code_couleur[4],
            int x,
            int y,
            int h,
            int w,
            int mode )
```

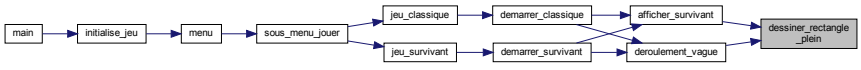
dessine un rectangle plein

Paramètres

SDL_Renderer	∗rendu, int code_couleur[4], int x, int y, int h, int w, int mode

Définition à la ligne 198 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.11.3.6 dessiner_rectangle_vide()

```
void dessiner_rectangle_vide (
            SDL_Renderer * rendu,
            int code_couleur[4],
            int x,
            int y,
            int h,
            int w,
            int mode )
```

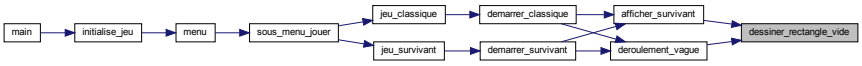
dessine un rectangle vide

Paramètres

SDL_Renderer	∗rendu, int code_couleur[4], int x, int y, int h, int w, int mode

Définition à la ligne 181 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.11.3.7 initialise_jeu()

```
int initialise_jeu ( )
```

fonction principale du jeu. Elle est chargée de la Création et du lancement de la fenêtre du jeu, du rendu ainsi que de l'appel des autres fonctions pour jouer. Enfin, elle s'occupe de détruire le rendu, la fenêtre et de quitter la sdl une fois sorti des fonction de jeu (juste avant de quitter le programme)

Paramètres

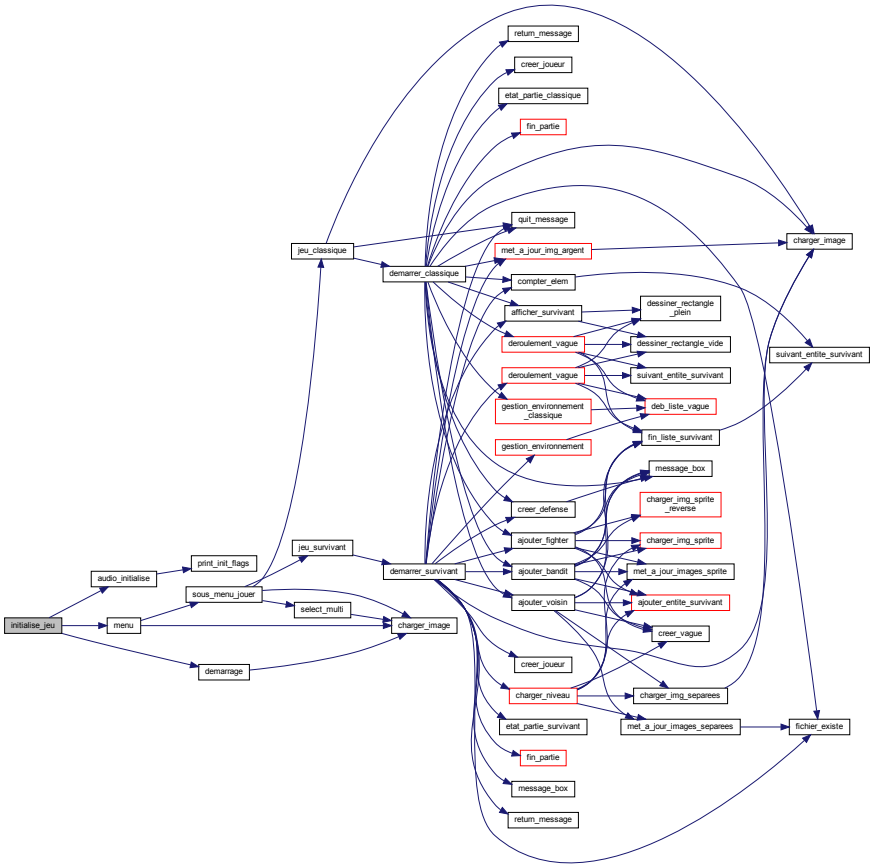
void	

Renvoie

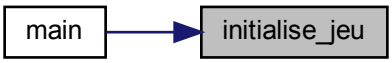
0

Définition à la ligne 514 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.11.3.8 menu()

```
void menu (
            SDL_Window * window,
            SDL_Renderer * rendu )
```

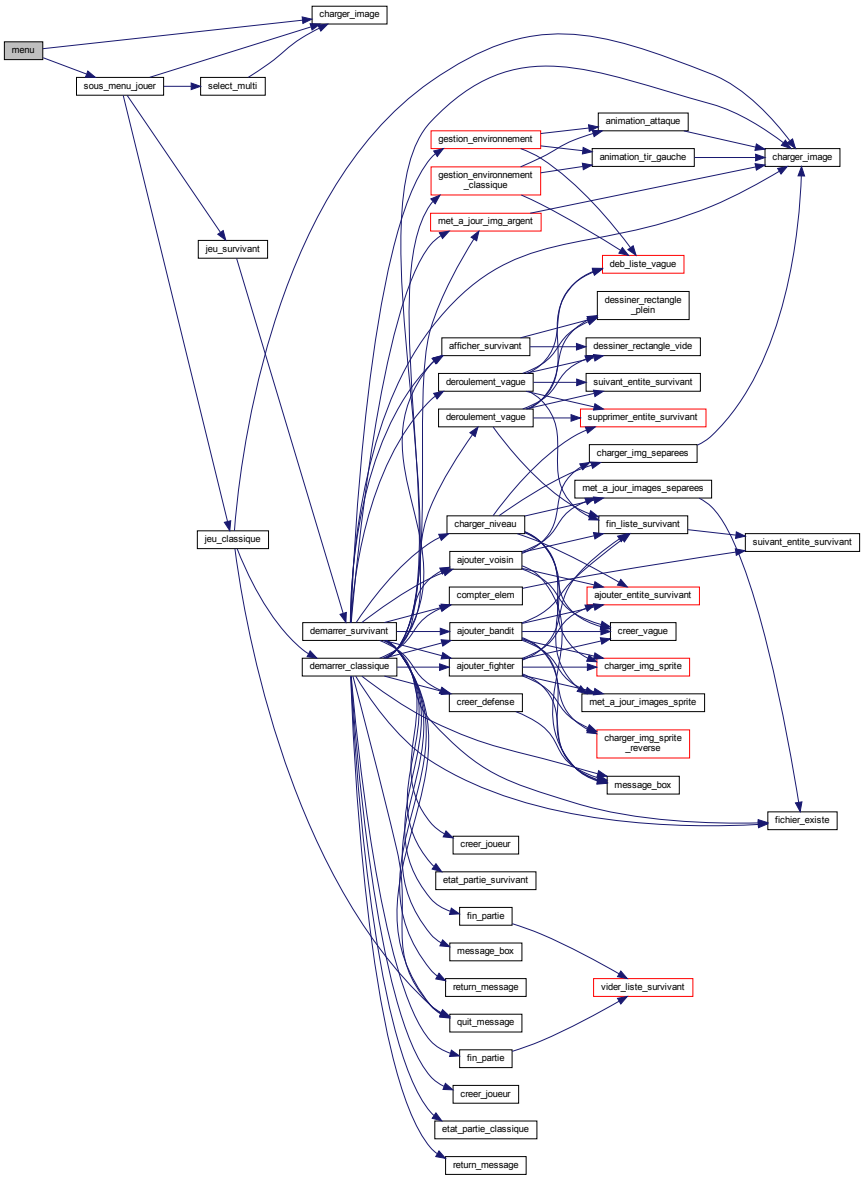
fonction servant à afficher le menu principal du jeu avec ses différentes options (jouer, paramètres, quitter) et à agir en fonction du choix de l'utilisateur.

Paramètres

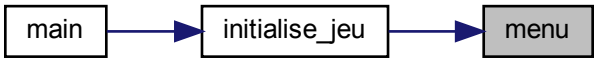
SDL_Window	∗window, SDL_Renderer ∗rendu

Définition à la ligne 450 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.11.3.9 quit_message()

```
int quit_message (
            void  )
```

affiche un message qui demande si on veut quitter la partie

Paramètres

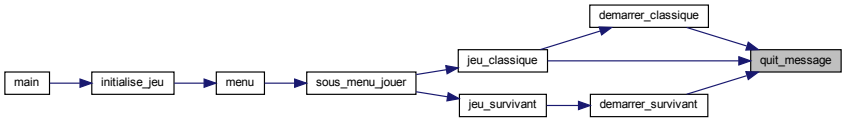
void	

Renvoie

-1 si l'utilisateur clique sur oui, 1 sinon

Définition à la ligne 141 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.11.3.10 select_multi()

```
int select_multi (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

fonction servant à afficher la sélection de l'hote ou du client.

Paramètres

SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event

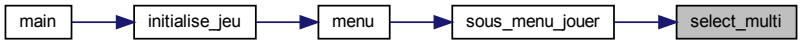
Renvoie

-1 ou 0

Définition à la ligne 323 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.11.3.11 sous_menu_jouer()

```
int sous_menu_jouer (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

fonction servant à afficher les options de jeu et à charger les autres fonctions en conséquence (mode classique, mode suvivant ou mode multi-joueur).

Paramètres

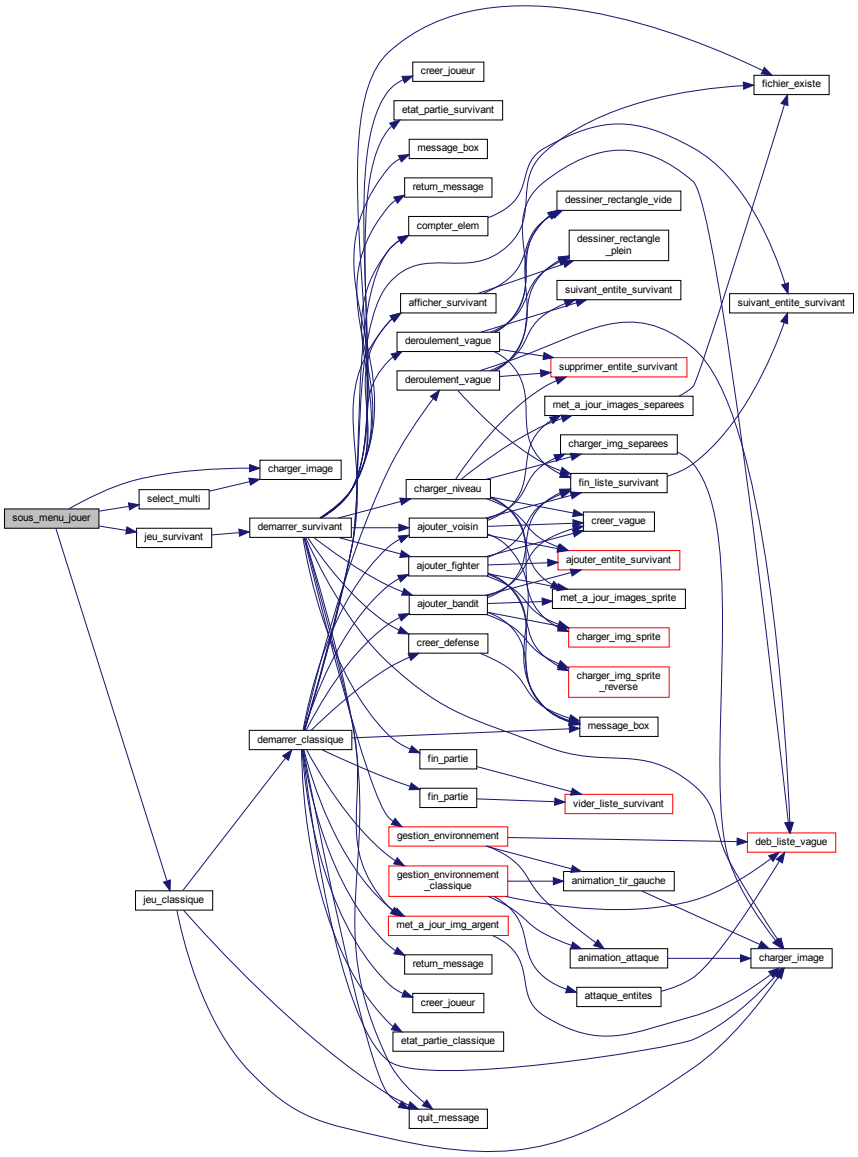
SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event

Renvoie

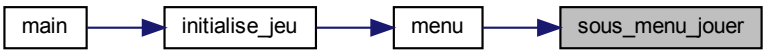
-1 ou 0

Définition à la ligne 379 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.11.4 Documentation des variables

5.11.4.1 audio

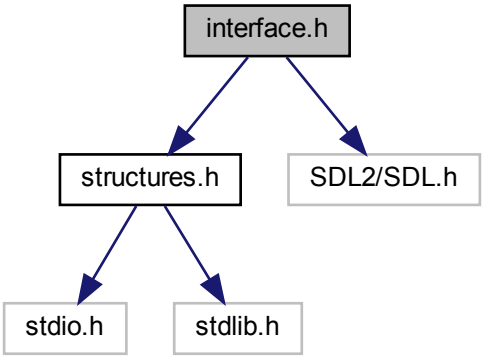
```
pthread_t audio
```

Définition à la ligne 18 du fichier interface.c.

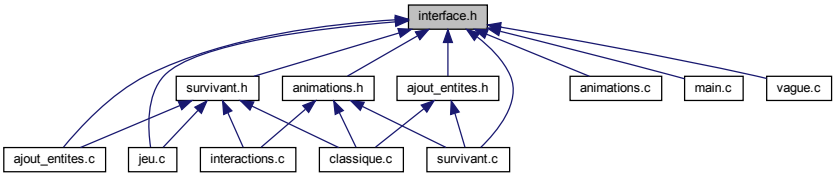
5.12 Référence du fichier interface.h

```
#include "structures.h"
#include "SDL2/SDL.h"
```

Graphe des dépendances par inclusion de interface.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- void convert_argent (joueur ∗p)
  Fonction qui sert à savoir quels chiffres composent le budget du joueur pour les afficher à l'écran (pour 1000 par exemple on aura 1.png, 0.png, 0.png, 0.png)
- int quit_message ()
  affiche un message qui demande si on veut quitter la partie
- void dessiner_rectangle_vide (SDL_Renderer ∗rendu, int code_couleur[4], int x, int y, int h, int w, int mode)
  dessine un rectangle vide

- void dessiner_rectangle_plein (SDL_Renderer ∗rendu, int code_couleur[4], int x, int y, int h, int w, int mode)
  dessine un rectangle plein
- int charger_image (char ∗nom, SDL_Renderer ∗rendu, int x, int y, int option)
  sert à charger une image avec tous les formats (grâce à SDL_image) sur le rendu et à présenter le rendu sur la fenetre (le format géré par défaut par la SDL est BMP)
- int charger_partie_image (char ∗nom, SDL_Renderer ∗rendu, int x, int y, int w, int h, int w_image, int h_image, int x_image, int y_image, int option)
  sert à charger un morceau d'une image (la taille de la figure à rogner, sa position et autres passés en paramètre) avec tous les formats (grâce à SDL_image) sur le rendu et à présenter le rendu sur la fenetre (le format géré par défaut par la SDL est BMP)
- void demarrage (SDL_Renderer ∗rendu)
  Fonction servant d'introduction au jeu en faisant une animation à l'écran.
- int menu_jouer_difficulte (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event)
- int select_multi (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event)
  fonction servant à afficher la sélection de l'hote ou du client.
- int sous_menu_jouer (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event)
  fonction servant à afficher les options de jeu et à charger les autres fonctions en conséquence (mode classique, mode suvivant ou mode multi-joueur).
- void menu (SDL_Window ∗window, SDL_Renderer ∗rendu)
  fonction servant à afficher le menu principal du jeu avec ses différentes options (jouer, paramètres, quitter) et à agir en fonction du choix de l'utilisateur.
- int initialise_jeu ()
  fonction principale du jeu. Elle est chargée de la Création et du lancement de la fenêtre du jeu, du rendu ainsi que de l'appel des autres fonctions pour jouer. Enfin, elle s'occupe de détruire le rendu, la fenêtre et de quitter la sdl une fois sorti des fonction de jeu (juste avant de quitter le programme)

5.12.1 Documentation des fonctions

5.12.1.1 charger_image()

```
int charger_image (
            char * nom,
            SDL_Renderer * rendu,
            int x,
            int y,
            int option )
```

sert à charger une image avec tous les formats (grâce à SDL_image) sur le rendu et à présenter le rendu sur la fenetre (le format géré par défaut par la SDL est BMP)

Paramètres

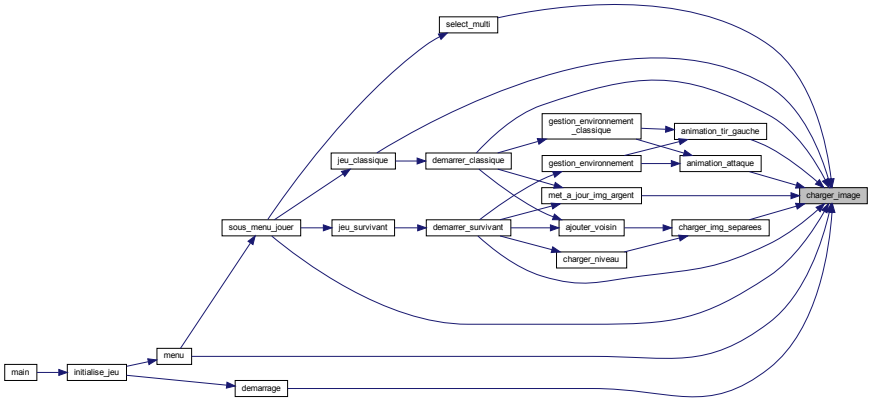
char	∗nom, SDL_Renderer ∗rendu, int x, int y, int option

Renvoie

-1 si un problème est survenu rien sinon

Définition à la ligne 219 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.12.1.2 charger_partie_image()

```
int charger_partie_image (
            char * nom,
            SDL_Renderer * rendu,
            int x,
            int y,
            int w,
            int h,
            int w_image,
            int h_image,
            int x_image,
            int y_image,
            int option )
```

sert à charger un morceau d'une image (la taille de la figure à rogner, sa position et autres passés en paramètre) avec tous les formats (grâce à SDL_image) sur le rendu et à présenter le rendu sur la fenetre (le format géré par défaut par la SDL est BMP)

Paramètres

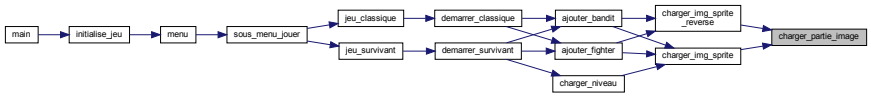
char	∗nom, SDL_Renderer ∗rendu, int x, int y, int w, int h, int w_image, int h_image, int x_image, int y_image, int option

Renvoie

-1 si un problème est survenu rien sinon

Définition à la ligne 248 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.12.1.3 convert_argent()

```
void convert_argent (
            joueur * p )
```

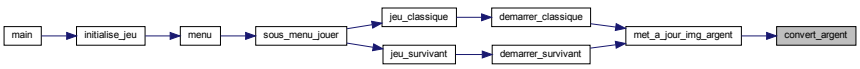
Fonction qui sert à savoir quels chiffres composent le budget du joueur pour les afficher à l'écran (pour 1000 par exemple on aura 1.png, 0.png, 0.png, 0.png)

Paramètres

joueur	∗p

Définition à la ligne 26 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.12.1.4 demarrage()

```
void demarrage (
            SDL_Renderer * rendu )
```

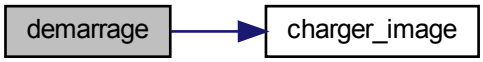
Fonction servant d'introduction au jeu en faisant une animation à l'écran.

Paramètres

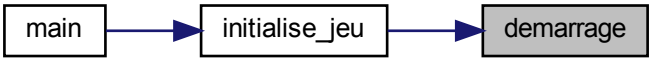
SDL_Renderer	∗rendu

Définition à la ligne 303 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.12.1.5 dessiner_rectangle_plein()

```
void dessiner_rectangle_plein (
            SDL_Renderer * rendu,
            int code_couleur[4],
            int x,
            int y,
            int h,
            int w,
            int mode )
```

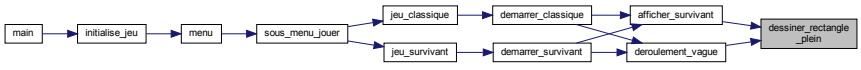
dessine un rectangle plein

Paramètres

SDL_Renderer	∗rendu, int code_couleur[4], int x, int y, int h, int w, int mode

Définition à la ligne 198 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.12.1.6 dessiner_rectangle_vide()

```
void dessiner_rectangle_vide (
            SDL_Renderer * rendu,
            int code_couleur[4],
            int x,
            int y,
            int h,
            int w,
            int mode )
```

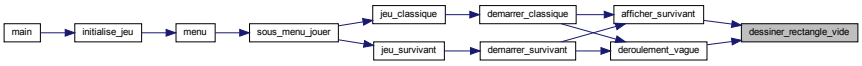
dessine un rectangle vide

Paramètres

SDL_Renderer	∗rendu, int code_couleur[4], int x, int y, int h, int w, int mode

Définition à la ligne 181 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.12.1.7 initialise_jeu()

```
int initialise_jeu ( )
```

fonction principale du jeu. Elle est chargée de la Création et du lancement de la fenêtre du jeu, du rendu ainsi que de l'appel des autres fonctions pour jouer. Enfin, elle s'occupe de détruire le rendu, la fenêtre et de quitter la sdl une fois sorti des fonction de jeu (juste avant de quitter le programme)

Paramètres

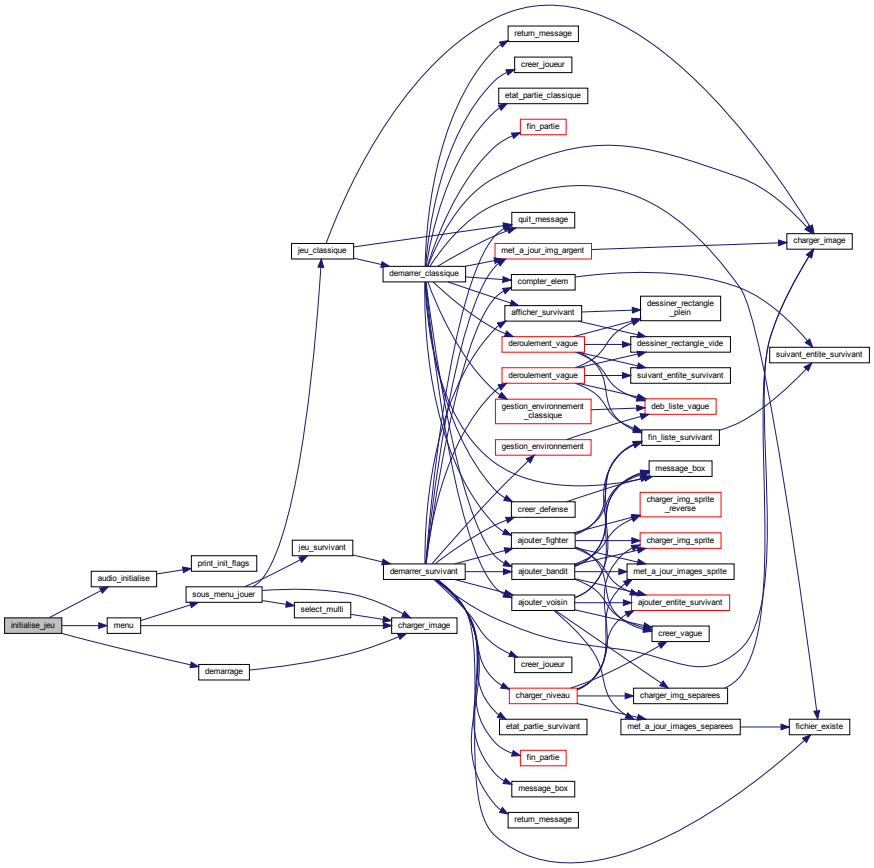
void	

Renvoie

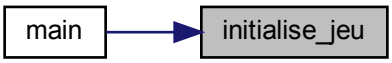
0

Définition à la ligne 514 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.12.1.8 menu()

```
void menu (
            SDL_Window * window,
            SDL_Renderer * rendu )
```

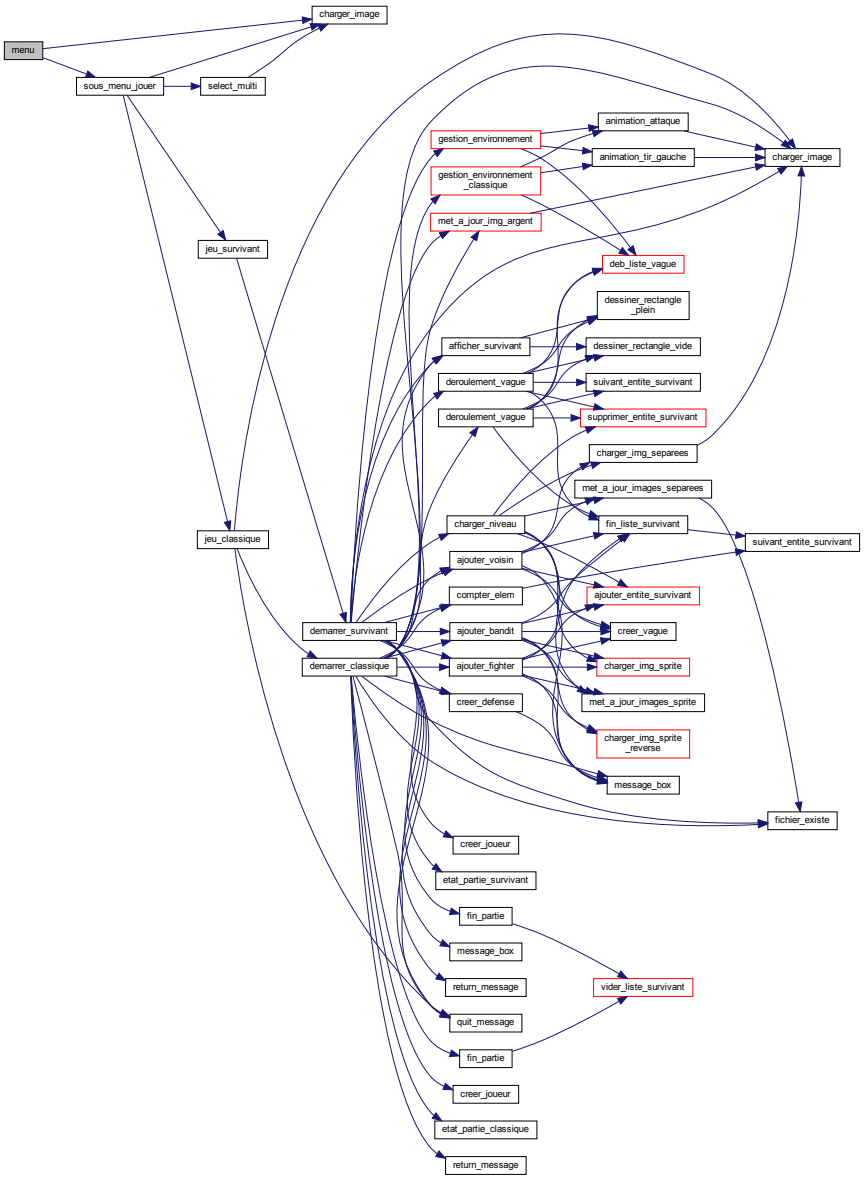
fonction servant à afficher le menu principal du jeu avec ses différentes options (jouer, paramètres, quitter) et à agir en fonction du choix de l'utilisateur.

Paramètres

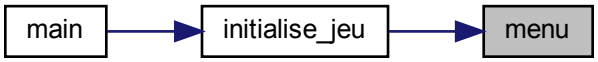
SDL_Window	∗window, SDL_Renderer ∗rendu

Définition à la ligne 450 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.12.1.9 menu_jouer_difficulte()

```
int menu_jouer_difficulte (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

5.12.1.10 quit_message()

```
int quit_message (
            void  )
```

affiche un message qui demande si on veut quitter la partie

Paramètres

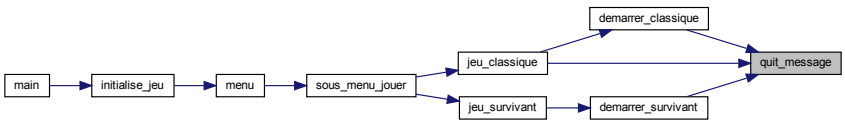
void	

Renvoie

-1 si l'utilisateur clique sur oui, 1 sinon

Définition à la ligne 141 du fichier interface.c.

Voici le graphe des appelants de cette fonction :



5.12.1.11 select_multi()

```
int select_multi (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

fonction servant à afficher la sélection de l'hote ou du client.

Paramètres

SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event

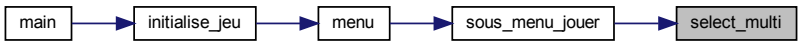
Renvoie

-1 ou 0

Définition à la ligne 323 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.12.1.12 sous_menu_jouer()

```
int sous_menu_jouer (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

fonction servant à afficher les options de jeu et à charger les autres fonctions en conséquence (mode classique, mode suvivant ou mode multi-joueur).

Paramètres

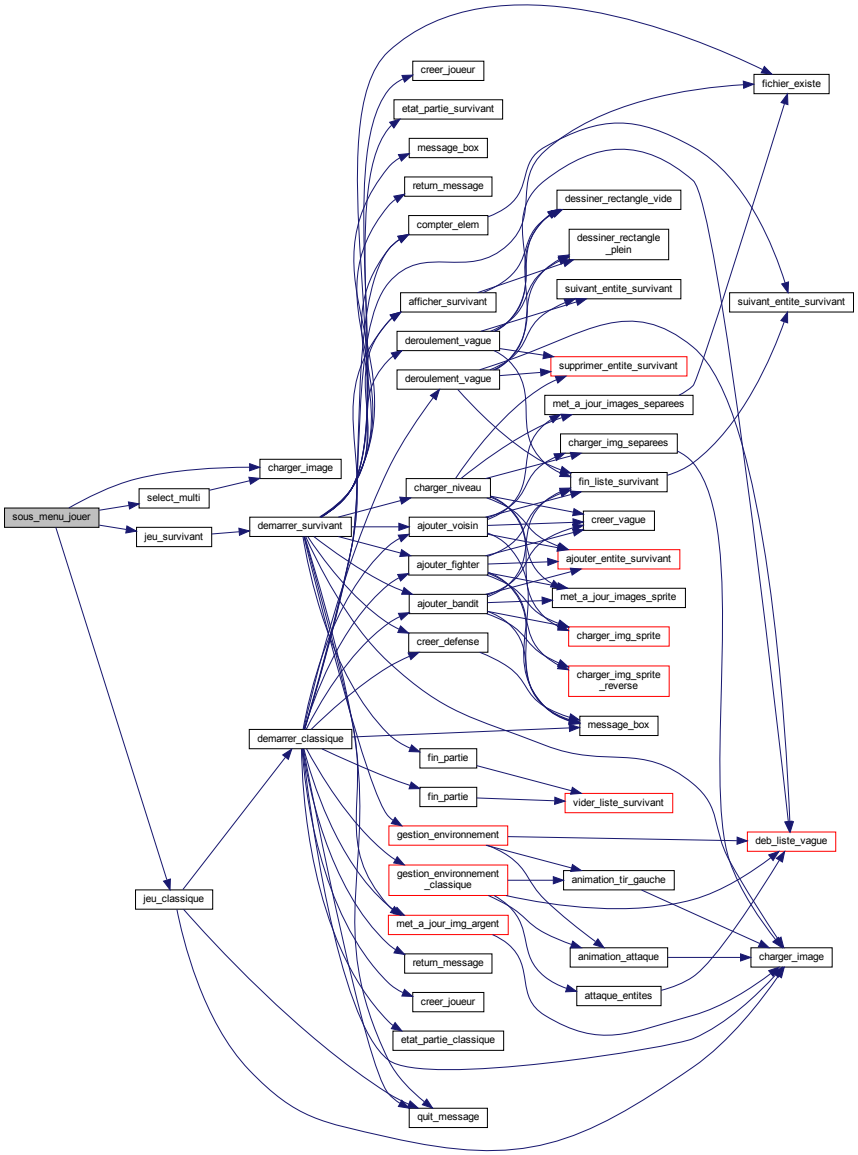
SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event

Renvoie

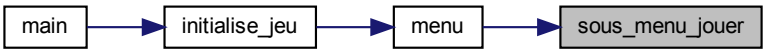
-1 ou 0

Définition à la ligne 379 du fichier interface.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

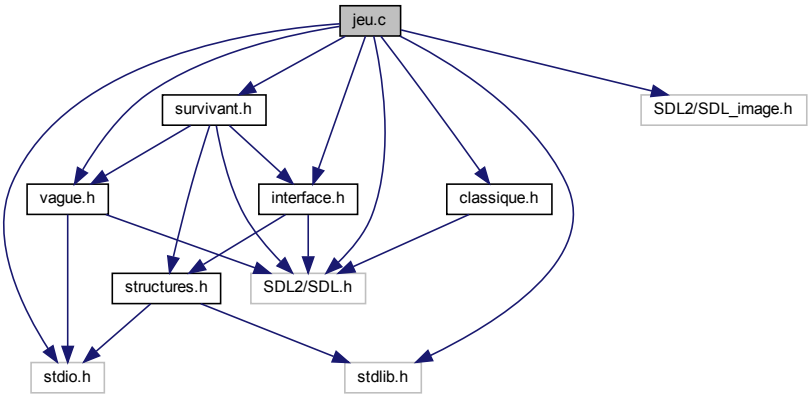


5.13 Référence du fichier jeu.c

Contient les fonctions principales qui lancent les 2 modes de jeu (survvivant et classique) avec quelques autres fonctions diverses.

```
#include <stdio.h>
#include <stdlib.h>
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "survivant.h"
#include "vague.h"
#include "interface.h"
#include "classique.h"
```

Graphe des dépendances par inclusion de jeu.c:



Macros

— #define NB_NIV_SURVIVANT 3

Fonctions

— int jeu_classique (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event)
— int jeu_survivant (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event)

lance une partie du jeu en mode survivant (gère tout l'affichage, les évènements etc...)

5.13.1 Description détaillée

Contient les fonctions principales qui lancent les 2 modes de jeu (survvivant et classique) avec quelques autres fonctions diverses.

Auteur

Lazare Maclouf

Version

1

Date

25/01/2022

5.13.2 Documentation des macros

5.13.2.1 NB_NIV_SURVIVANT

```
#define NB_NIV_SURVIVANT 3
```

Définition à la ligne 16 du fichier jeu.c.

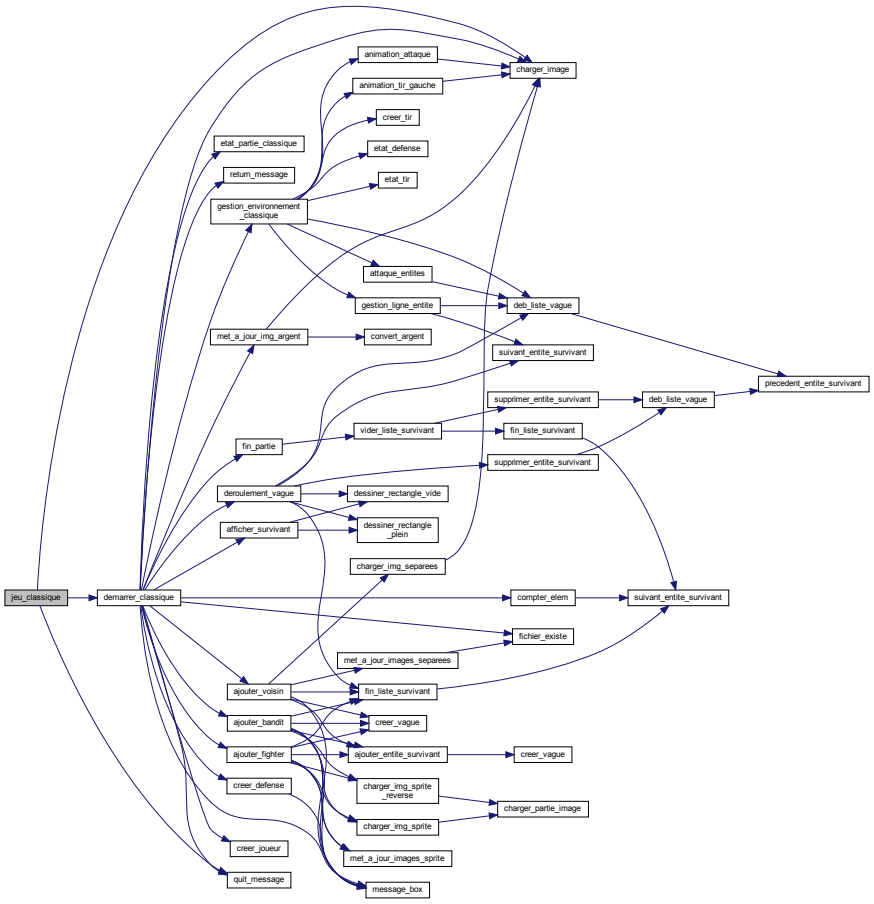
5.13.3 Documentation des fonctions

5.13.3.1 jeu_classique()

```
int jeu_classique (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

Définition à la ligne 24 du fichier jeu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

5.13.3.2 jeu_survivant()

```
int jeu_survivant (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

lance une partie du jeu en mode survivant (gère tout l'affichage, les évènements etc...)

Paramètres

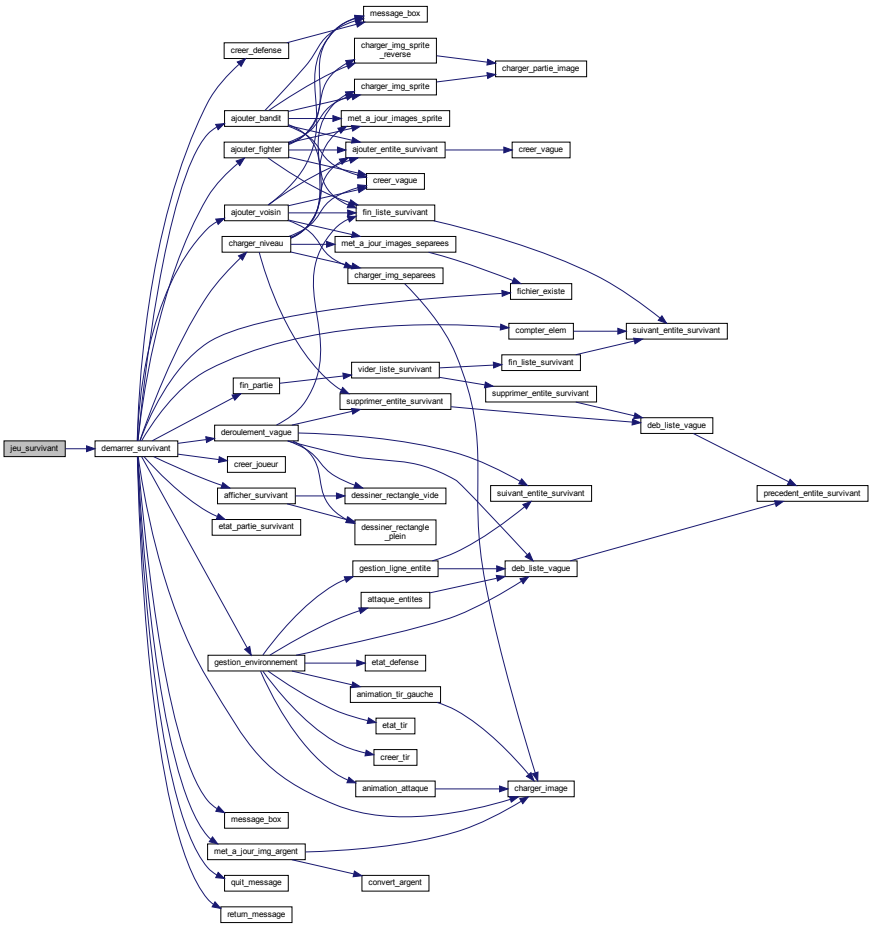
SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event

Renvoie

-1, 0 ou 1

Définition à la ligne 60 du fichier jeu.c.

Voici le graphe d'appel pour cette fonction :

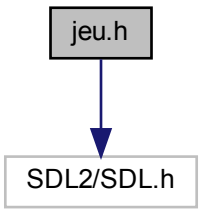


Voici le graphe des appelants de cette fonction :

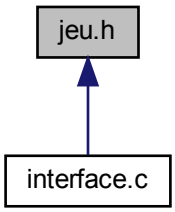
5.14 Référence du fichier jeu.h

```
#include "SDL2/SDL.h"
```

Graphe des dépendances par inclusion de jeu.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- int jeu_classique (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event)
- int jeu_survivant (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event)

  lance une partie du jeu en mode survivant (gère tout l'affichage, les évènements etc...)

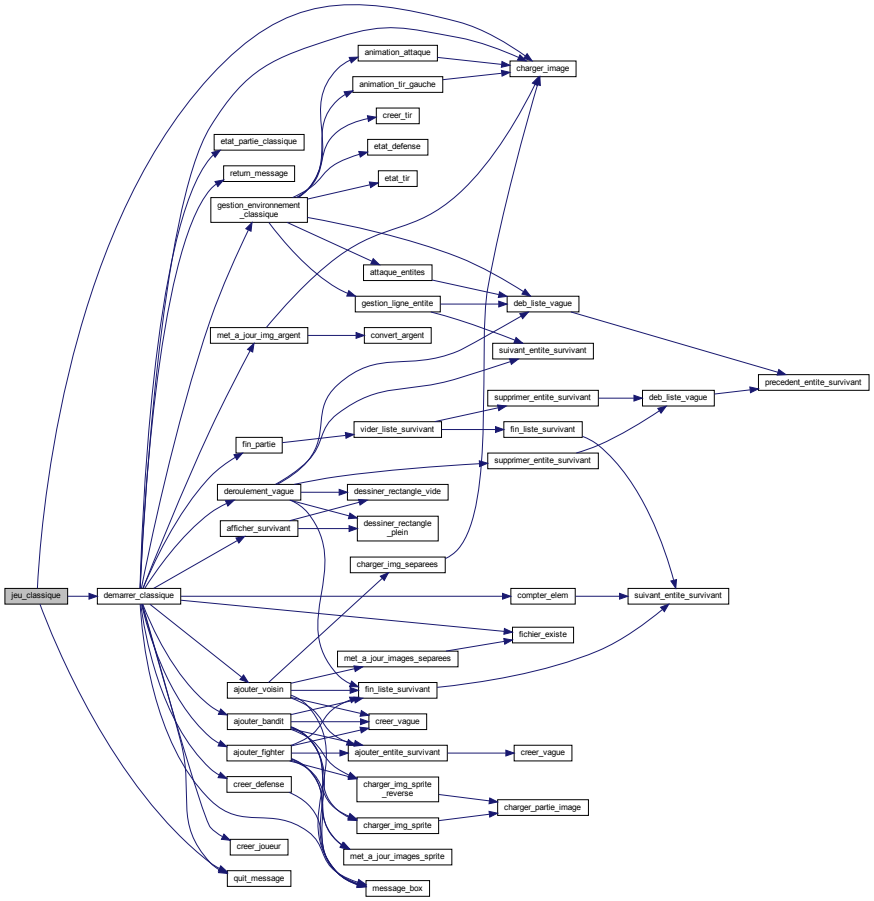
5.14.1 Documentation des fonctions

5.14.1.1 jeu_classique()

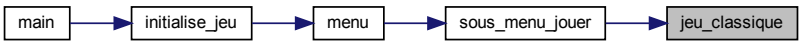
```
int jeu_classique (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

Définition à la ligne 24 du fichier jeu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.14.1.2 jeu_survivant()

```
int jeu_survivant (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event )
```

lance une partie du jeu en mode survivant (gère tout l'affichage, les évènements etc...)

Paramètres

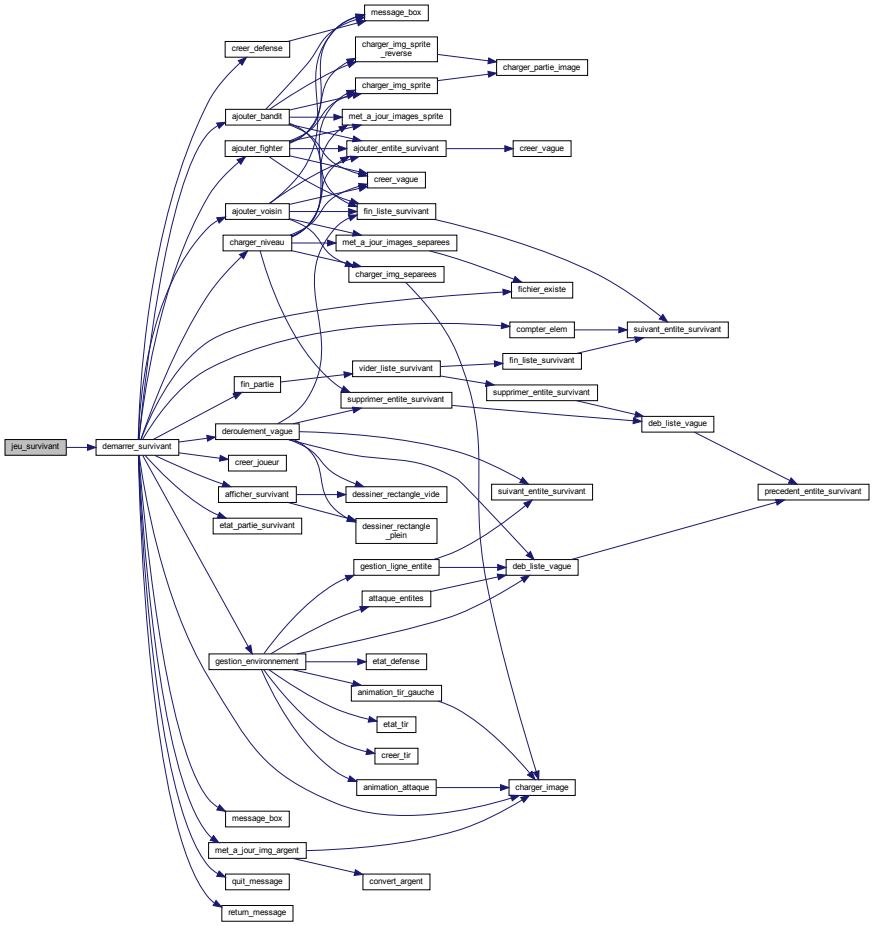
SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event

Renvoie

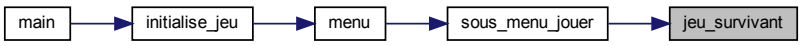
-1, 0 ou 1

Définition à la ligne 60 du fichier jeu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

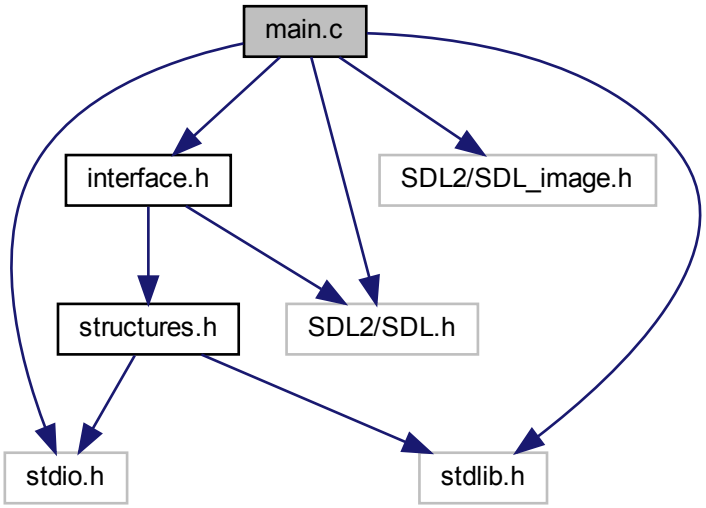


5.15 Référence du fichier main.c

contient le main qui initialise le jeu

```
#include <stdio.h>
#include <stdlib.h>
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "interface.h"
```

Graphe des dépendances par inclusion de main.c:



Fonctions

— int main (int argc, char **argv)

fonction qui sert à lancer le jeu en faisant appel à la fonction initialise_jeu

5.15.1 Description détaillée

contient le main qui initialise le jeu

Version

1

Date

22/01/2022

5.15.2 Documentation des fonctions

5.15.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

fonction qui sert à lancer le jeu en faisant appel à la fonction initialise_jeu

Paramètres

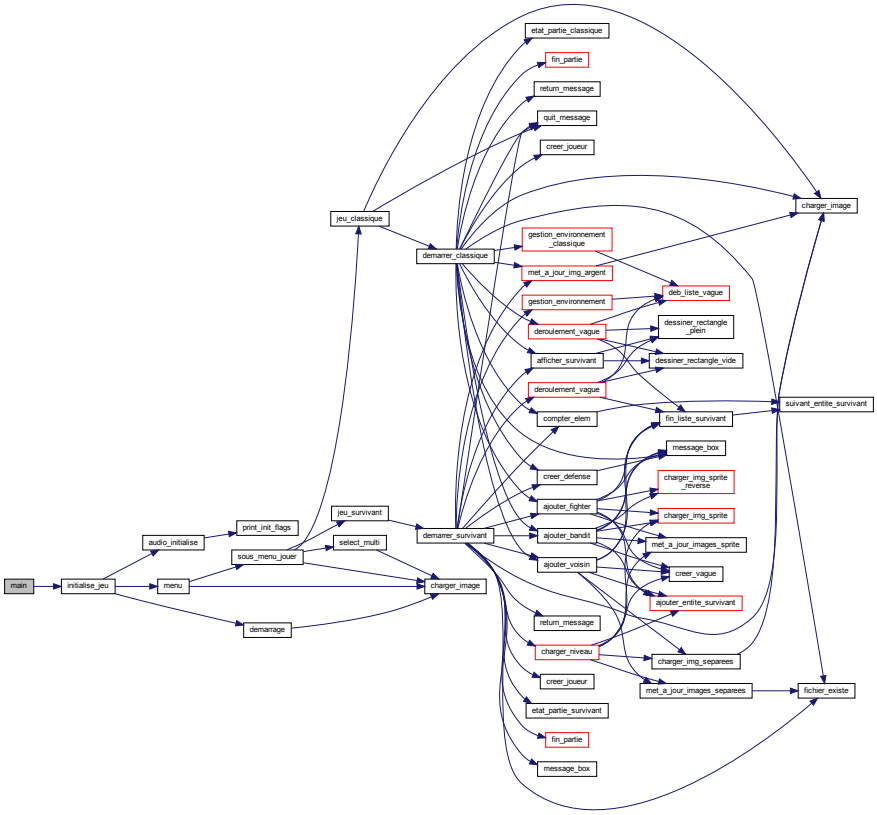
int	argc, char ∗∗argv

Renvoie

0

Définition à la ligne 19 du fichier main.c.

Voici le graphe d'appel pour cette fonction :

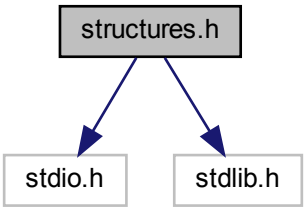


5.16 Référence du fichier README.md

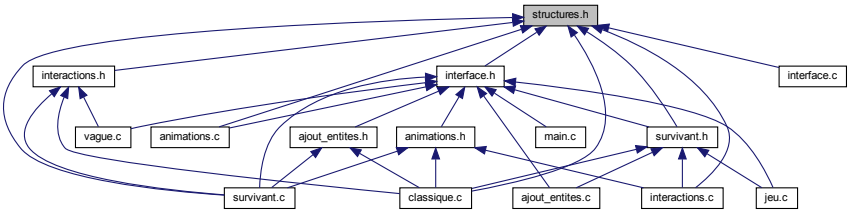
5.17 Référence du fichier structures.h

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

Graphe des dépendances par inclusion de structures.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Structures de données

- struct mes
- struct def
- struct tir_s
- struct joue

Définitions de type

- typedef struct mes message
- typedef struct def defense
- typedef struct tir_s tir
- typedef struct joue joueur

5.17.1 Documentation des définitions de type

5.17.1.1 defense

```
typedef struct def defense
```

5.17.1.2 joueur

```
typedef struct joue joueur
```

5.17.1.3 message

```
typedef struct mes message
```

5.17.1.4 tir

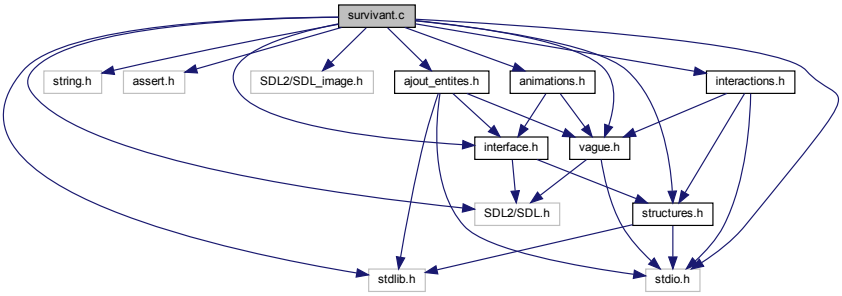
```
typedef struct tir_s tir
```

5.18 Référence du fichier survivant.c

Contient toutes les fonctions utiles au mode de jeu survivant de battle ground (gestion de l'évolution de la partie, implémentation des structures de jeu, etc..)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "interface.h"
#include "animations.h"
#include "structures.h"
#include "vague.h"
#include "ajout_entites.h"
#include "interactions.h"
```

Graphe des dépendances par inclusion de survivant.c:



Macros

- #define V 5
- #define y_entity 470
- #define x_def 100
- #define y_def 470
- #define pv_joueur 200
- #define argent_joueur 1000

Fonctions

- t_wave ∗ charger_niveau (char ∗nom)
  fonction qui sert à charger un niveau en mode survivant, en allant chercher dans un fichier dont le nom est passé en paramètres les informations du niveau (pour chaque vague le nombres d'entités, leur types, leur pv, le temps entre les vague etc..).
- int deroulement_vague (joueur ∗player, t_wave ∗vague, SDL_Renderer ∗rendu, int camp)

fonction qui sert à mettre automatiquement les images des entités à jour ainsi que d'autres champs (points de vie, position etc...).

— int fin_partie (t_wave ∗vague)

fonction qui sert à vider une liste chainée d'entité. c'est-à-dire supprimer toutes les entités qui la compose et ainsi libérer la mémoire.

— joueur ∗ creer_joueur ()

fonction qui sert à créer une variable de type joueur pour la partie.

— void message_box (message ∗msg, char nom_fic[100])

fonction qui sert à créer une variable de msg.

— int etat_partie_survivant (t_wave ∗vague, joueur ∗player)

fonction qui sert à vérifier si la partie est finie ou si elle est toujours en cours (et si elle est finie si le joueur a gagné ou perdu)

— int return_message ()

fonction qui sert à afficher un message de prévention lorsque l'utilisateur veut quitter une partie lancée et en fonction du retour de ce dernier (oui, non ou annuler) retourner une valeur entière

— int demarrer_survivant (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event, char lvl)

Comme son nom l'indique, cette fonction démarre le mode survivant. C'est la fonction principale du mode jeu survivant et s'occupe de tout gérer (les évènements, l'argent et les unités de l'utilisateur; charger le niveau et toutes les entités qui vont avec etc/..)

5.18.1 Description détaillée

Contient toutes les fonctions utiles au mode de jeu survivant de battle ground (gestion de l'évolution de la partie, implémentation des structures de jeu, etc..)

Auteur

Lazare Maclouf

Version

1

Date

25/02/2022

5.18.2 Documentation des macros

5.18.2.1 argent_joueur

```
#define argent_joueur 1000
```

Définition à la ligne 26 du fichier survivant.c.

5.18.2.2 pv_joueur

```
#define pv_joueur 200
```

Définition à la ligne 25 du fichier survivant.c.

5.18.2.3 V

```
#define V 5
```

Définition à la ligne 21 du fichier survivant.c.

5.18.2.4 x_def

```
#define x_def 100
```

Définition à la ligne 23 du fichier survivant.c.

5.18.2.5 y_def

```
#define y_def 470
```

Définition à la ligne 24 du fichier survivant.c.

5.18.2.6 y_entity

```
#define y_entity 470
```

Définition à la ligne 22 du fichier survivant.c.

5.18.3 Documentation des fonctions

5.18.3.1 charger_niveau()

```
t_wave* charger_niveau (
            char * nom )
```

fonction qui sert à charger un niveau en mode survivant, en allant chercher dans un fichier dont le nom est passé en paramètres les informations du niveau (pour chaque vague le nombres d'entités, leur types, leur pv, le temps entre les vague etc..).

Paramètres

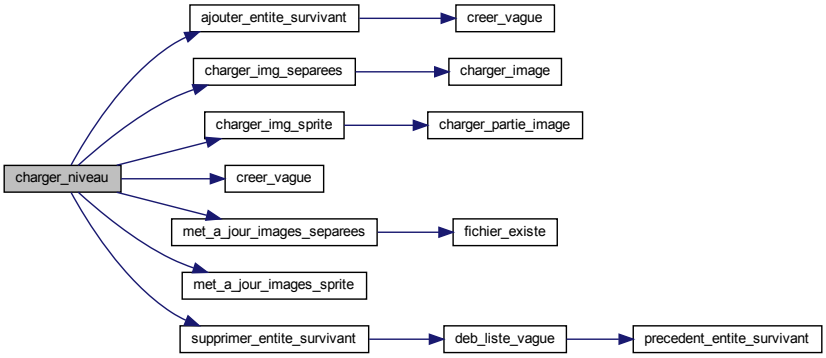
char	∗nom

Renvoie

t_wave ∗ (pointeur sur t_wave qui aura été rempli par les informations contenu dans le fichier.)

Définition à la ligne 34 du fichier survivant.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

5.18.3.2 creer_joueur()

```
joueur* creer_joueur ( )
```

fonction qui sert à créer une variable de type joueur pour la partie.

Paramètres

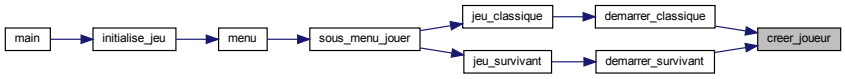
void	

Renvoie

joueur ∗

Définition à la ligne 289 du fichier survivant.c.

Voici le graphe des appelants de cette fonction :



5.18.3.3 demarrer_survivant()

```
int demarrer_survivant (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event,
            char lvl )
```

Comme son nom l'indique, cette fonction démarre le mode survivant. C'est la fonction principale du mode jeu survivant et s'occupe de tout gérer (les évènements, l'argent et les unités de l'utilisateur; charger le niveau et toutes les entités qui vont avec etc/..)

Paramètres

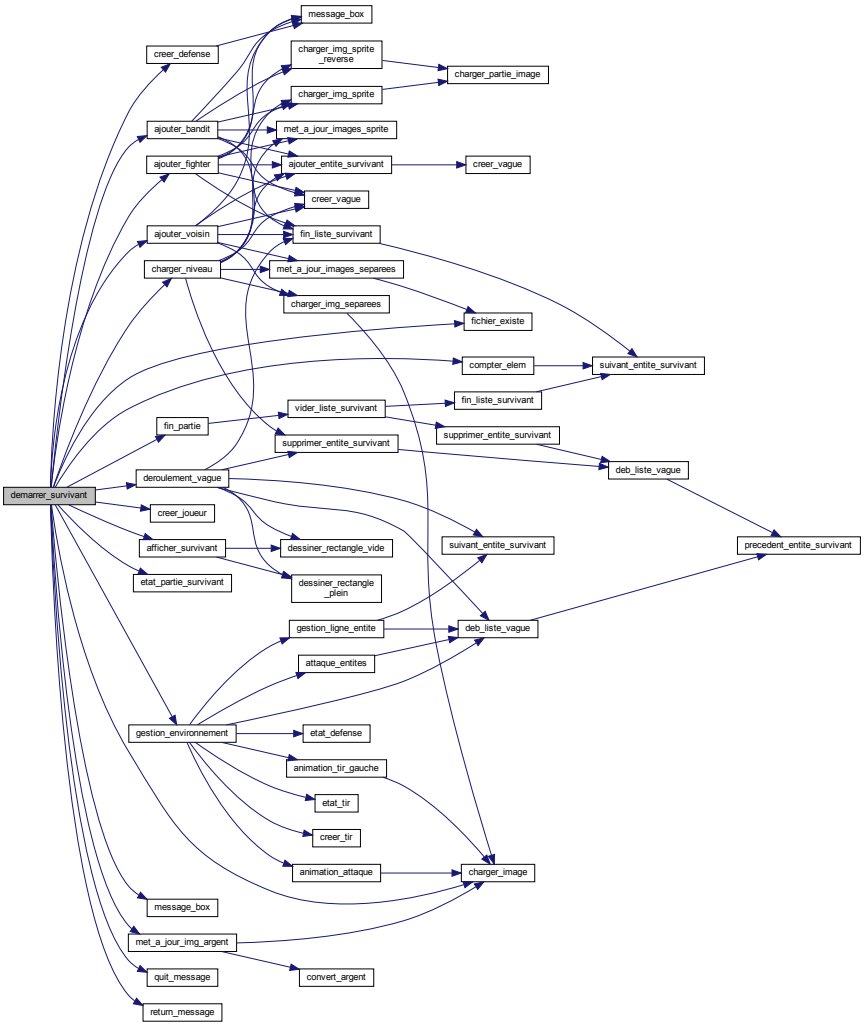
SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event, char lvl

Renvoie

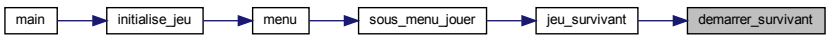
int

Définition à la ligne 395 du fichier survivant.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.18.3.4 deroulement_vague()

```
int deroulement_vague (
            joueur * player,
            t_wave * vague,
```

```
SDL_Renderer * rendu,
int camp )
```

fonction qui sert à mettre automatiquement les images des entités à jour ainsi que d'autres champs (points de vie, position etc...).

Paramètres

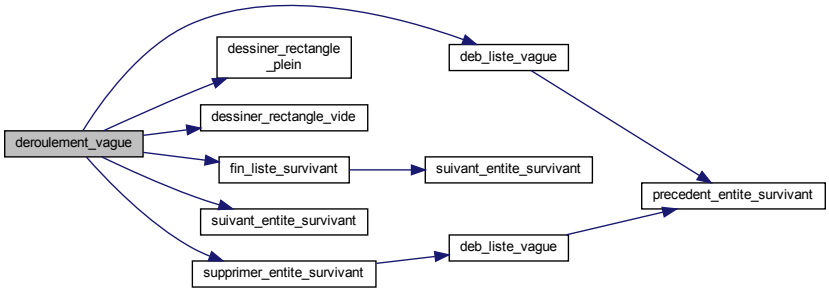
t_wave	∗vague, SDL_Renderer ∗rendu, int camp

Renvoie

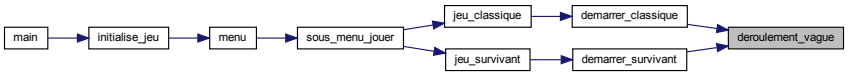
rien

Définition à la ligne 88 du fichier survivant.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.18.3.5 etat_partie_survivant()

```
int etat_partie_survivant (
            t_wave * vague,
            joueur * player )
```

fonction qui sert à vérifier si la partie est finie ou si elle est toujours en cours (et si elle est finie si le joueur a gagné ou perdu)

Paramètres

t_wave	∗vague, joueur ∗player

Renvoie

int

Définition à la ligne 329 du fichier survivant.c.

Voici le graphe des appelants de cette fonction :

5.18.3.6 fin_partie()

```
int fin_partie (
            t_wave * vague )
```

fonction qui sert à vider une liste chainée d'entité. c'est-à-dire supprimer toutes les entités qui la compose et ainsi libérer la mémoire.

Paramètres

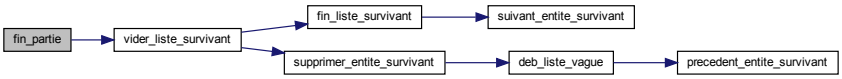
t_wave	*vague

Renvoie

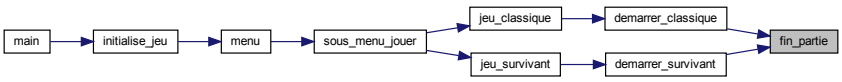
int

Définition à la ligne 277 du fichier survivant.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.18.3.7 message_box()

```
void message_box (
            message * msg,
            char nom_fic[100] )
```

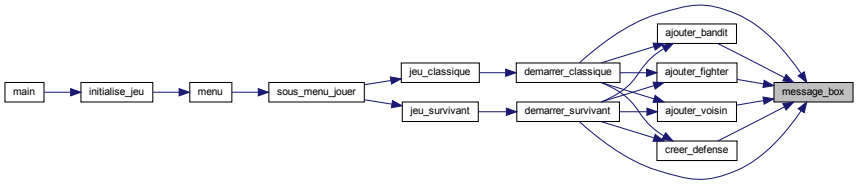
fonction qui sert à créer une variable de msg.

Paramètres

message	∗msg, char nom_fic[100]

Définition à la ligne 308 du fichier survivant.c.

Voici le graphe des appelants de cette fonction :



5.18.3.8 return_message()

```
int return_message ( )
```

fonction qui sert à afficher un message de prévention lorsque l'utilisateur veut quitter une partie lancée et en fonction du retour de ce dernier (oui, non ou annuler) retourner une valeur entière

Paramètres

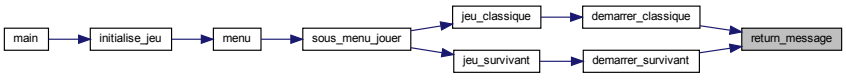
void	

Renvoie

int

Définition à la ligne 354 du fichier survivant.c.

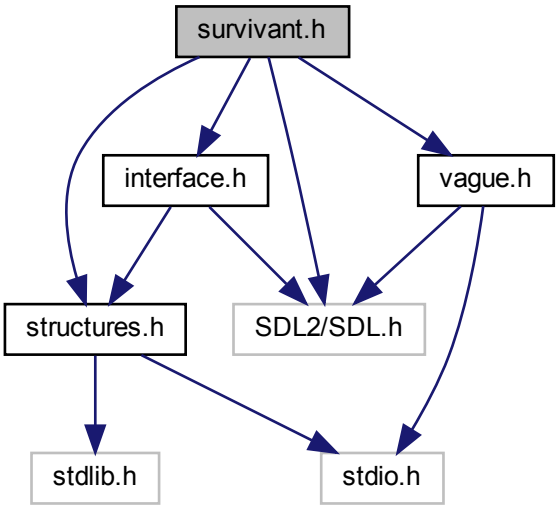
Voici le graphe des appelants de cette fonction :



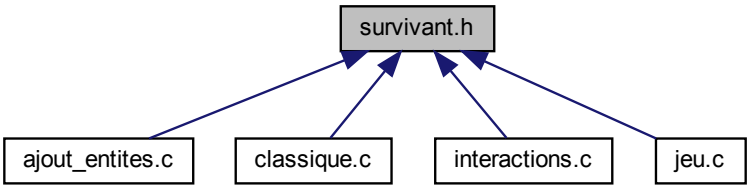
5.19 Référence du fichier survivant.h

```
#include "interface.h"
#include "structures.h"
#include "vague.h"
#include "SDL2/SDL.h"
```

Graphe des dépendances par inclusion de survivant.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- t_wave ∗ charger_niveau (char ∗nom)
  fonction qui sert à charger un niveau en mode survivant, en allant chercher dans un fichier dont le nom est passé en paramètres les informations du niveau (pour chaque vague le nombres d'entités, leur types, leur pv, le temps entre les vague etc..).
- joueur ∗ creer_joueur ()

fonction qui sert à créer une variable de type joueur pour la partie.

- int etat_partie_survivant (t_wave ∗vague, joueur ∗player)

  fonction qui sert à vérifier si la partie est finie ou si elle est toujours en cours (et si elle est finie si le joueur a gagné ou perdu)
- void message_box (message ∗msg, char nom_fic[100])

  fonction qui sert à créer une variable de msg.
- int demarrer_survivant (SDL_Window ∗window, SDL_Renderer ∗rendu, SDL_Event ∗event, char lvl)

  Comme son nom l'indique, cette fonction démarre le mode survivant. C'est la fonction principale du mode jeu survivant et s'occupe de tout gérer (les évènements, l'argent et les unités de l'utilisateur; charger le niveau et toutes les entités qui vont avec etc/..)
- int deroulement_vague (joueur ∗player, t_wave ∗vague, SDL_Renderer ∗rendu, int camp)

  fonction qui sert à mettre automatiquement les images des entités à jour ainsi que d'autres champs (points de vie, position etc...).
- int fin_partie (t_wave ∗vague)

  fonction qui sert à vider une liste chainée d'entité. c'est-à-dire supprimer toutes les entités qui la compose et ainsi libérer la mémoire.
- int return_message ()

  fonction qui sert à afficher un message de prévention lorsque l'utilisateur veut quitter une partie lancée et en fonction du retour de ce dernier (oui, non ou annuler) retourner une valeur entière

5.19.1 Documentation des fonctions

5.19.1.1 charger_niveau()

```
t_wave * charger_niveau (
            char * nom )
```

fonction qui sert à charger un niveau en mode survivant, en allant chercher dans un fichier dont le nom est passé en paramètres les informations du niveau (pour chaque vague le nombres d'entités, leur types, leur pv, le temps entre les vague etc..).

Paramètres

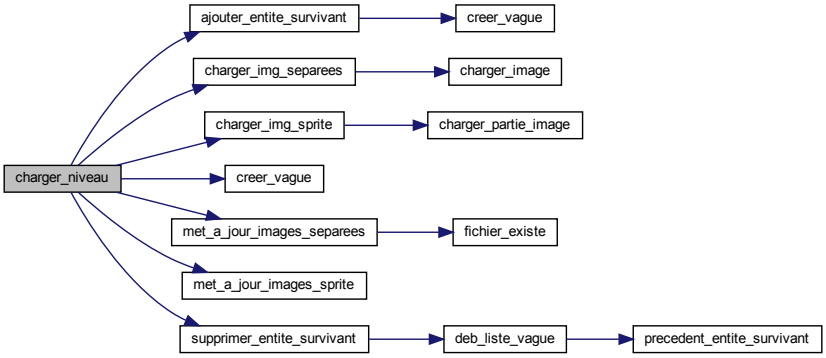
char	∗nom

Renvoie

t_wave ∗ (pointeur sur t_wave qui aura été rempli par les informations contenu dans le fichier.)

Définition à la ligne 34 du fichier survivant.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

5.19.1.2 creer_joueur()

```
joueur * creer_joueur ( )
```

fonction qui sert à créer une variable de type joueur pour la partie.

Paramètres

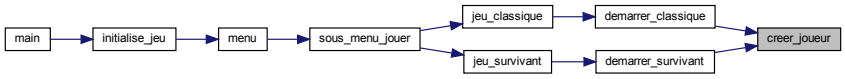
void	

Renvoie

joueur ∗

Définition à la ligne 289 du fichier survivant.c.

Voici le graphe des appelants de cette fonction :



5.19.1.3 demarrer_survivant()

```
int demarrer_survivant (
            SDL_Window * window,
            SDL_Renderer * rendu,
            SDL_Event * event,
            char lvl )
```

Comme son nom l'indique, cette fonction démarre le mode survivant. C'est la fonction principale du mode jeu survivant et s'occupe de tout gérer (les évènements, l'argent et les unités de l'utilisateur; charger le niveau et toutes les entités qui vont avec etc/..)

Paramètres

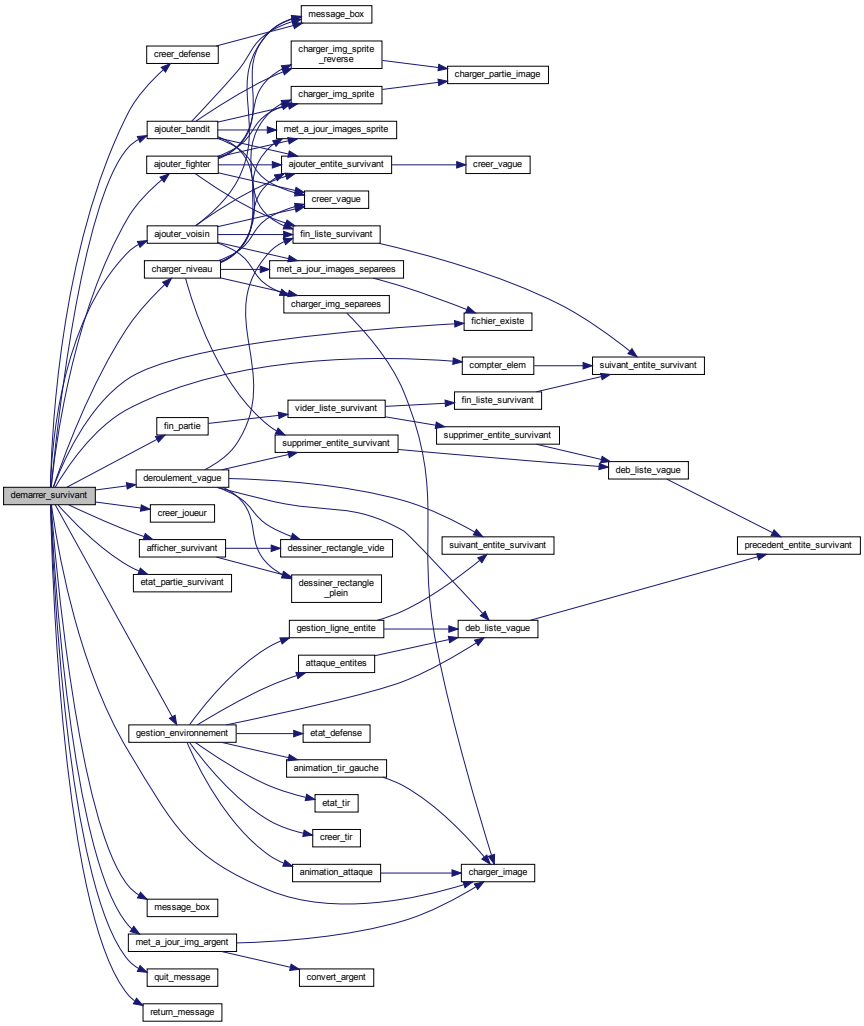
SDL_Window	∗window, SDL_Renderer ∗rendu, SDL_Event ∗event, char lvl

Renvoie

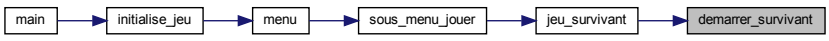
int

Définition à la ligne 395 du fichier survivant.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.19.1.4 deroulement_vague()

```
void deroulement_vague (
            joueur * player,
            t_wave * vague,
```

```
            SDL_Renderer * rendu,
            int camp )
```

fonction qui sert à mettre automatiquement les images des entités à jour ainsi que d'autres champs (points de vie, position etc...).

Paramètres

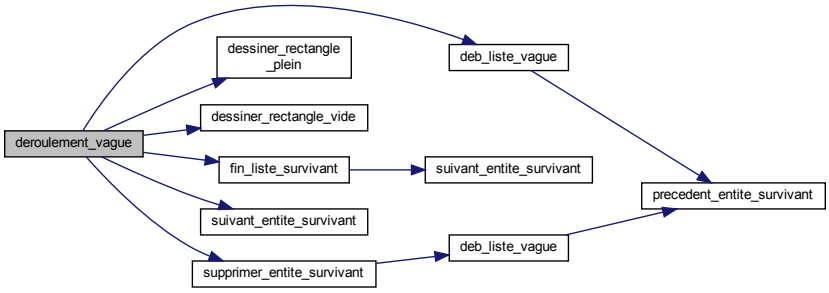
t_wave	∗vague, SDL_Renderer ∗rendu, int camp

Renvoie

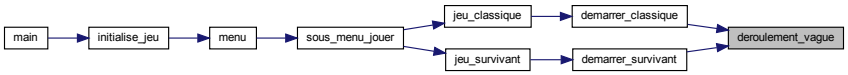
rien

Définition à la ligne 88 du fichier survivant.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.19.1.5 etat_partie_survivant()

```
int etat_partie_survivant (
            t_wave * vague,
            joueur * player )
```

fonction qui sert à vérifier si la partie est finie ou si elle est toujours en cours (et si elle est finie si le joueur a gagné ou perdu)

Paramètres

t_wave	∗vague, joueur ∗player

Renvoie

int

Définition à la ligne 329 du fichier survivant.c.

Voici le graphe des appelants de cette fonction :

5.19.1.6 fin_partie()

```
fin_partie (
            t_wave * vague )
```

fonction qui sert à vider une liste chainée d'entité. c'est-à-dire supprimer toutes les entités qui la compose et ainsi libérer la mémoire.

Paramètres

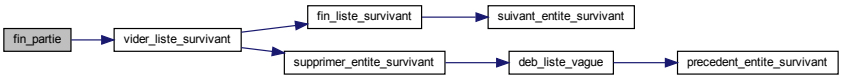
t_wave	*vague

Renvoie

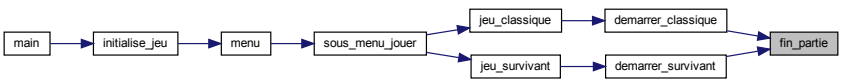
int

Définition à la ligne 277 du fichier survivant.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.19.1.7 message_box()

```
void message_box (
            message * msg,
            char nom_fic[100] )
```

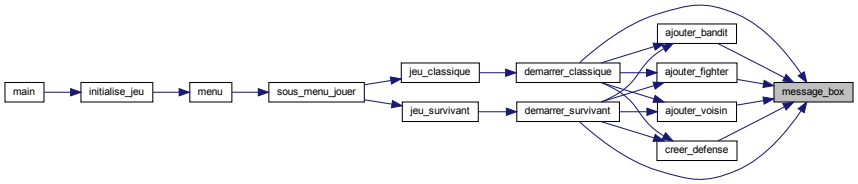
fonction qui sert à créer une variable de msg.

Paramètres

message	∗msg, char nom_fic[100]

Définition à la ligne 308 du fichier survivant.c.

Voici le graphe des appelants de cette fonction :



5.19.1.8 return_message()

```
int return_message ( )
```

fonction qui sert à afficher un message de prévention lorsque l'utilisateur veut quitter une partie lancée et en fonction du retour de ce dernier (oui, non ou annuler) retourner une valeur entière

Paramètres

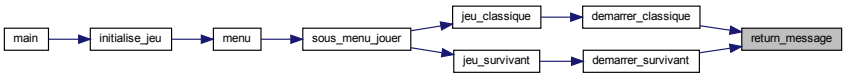
void	

Renvoie

int

Définition à la ligne 354 du fichier survivant.c.

Voici le graphe des appelants de cette fonction :

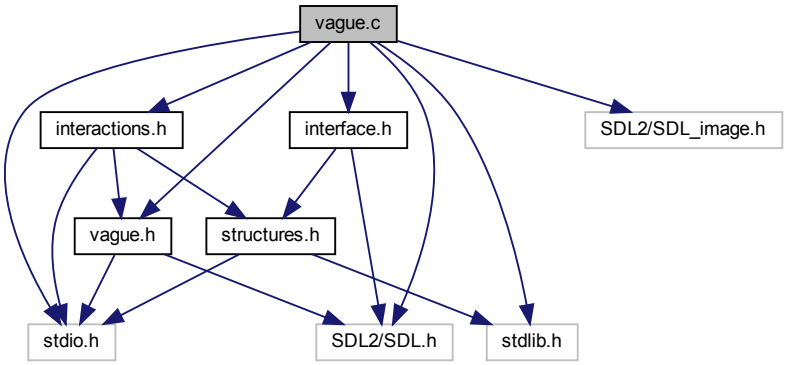


5.20 Référence du fichier vague.c

Contient toutes les fonctions relatives à la gestion et à la manipulation des vague d'entités.

```
#include <stdio.h>
#include <stdlib.h>
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "interface.h"
#include "vague.h"
#include "interactions.h"
```

Graphe des dépendances par inclusion de vague.c:



Fonctions

- t_wave * creer_vague ()
  crée une variable de type t_wave (une vague).
- t_wave * suivant_entite_survivant (t_wave *vague)
  place le pointeur passé en paramètre de la liste sur l'élément suivant
- t_wave * precedent_entite_survivant (t_wave *vague)
  place le pointeur passé en paramètre de la liste sur l'élément précédent
- t_wave * fin_liste_survivant (t_wave *vague)
  place le pointeur passé en paramètre à la fin de la liste il s'agit d'une boucle. tant qu'on est pas à la fin de la liste on continue à avancer dans la liste à l'aide la fonction suivant_entite_survivant
- t_wave * deb_liste_vague (t_wave *vague)
  place le pointeur passé en paramètre au début de la liste. il s'agit d'une boucle. tant qu'on est pas au début de la liste on continue à reculer dans la liste à l'aide la fonction precedent_entite_survivant
- int liste_vide_survivant (t_wave *vague)
  indique si la liste est vide ou pas au niveau survivant. Pour le savoir, on se place au début de la liste puis on regarde si le premier élément est null si c'est le cas, la liste est vide
- t_wave * ajouter_entite_survivant (t_wave *vague)
  ajoute une entité à la liste et met le pointeur passé en paramètre sur le nouvel élément. Pour ajouter un nouvel element on fait appel à la fonction creer_vague()
- t_wave * supprimer_entite_survivant (t_wave *vague)
  supprime l'entité au niveau du pointeur passé en paramètres. retourne un pointeur sur l'élément précédent si il ya un élement précédent, sinon le suivant, sinon retourne NULL
- t_wave * vider_liste_survivant (t_wave *vague)
  fonction qui sert à vider la liste.
- int compter_elem (t_wave *vague)

compte le nombre de vague restante (et renvoie le nombre d'entité encore à tuer)

— void met_a_jour_img_argent (joueur ∗player, SDL_Renderer ∗rendu)

met la somme d'agent a jour en fonction de ce que le player a dépensé et gangé en changeant les imgages pour monter la quantité d'argent

— void met_a_jour_images_separees (entite ∗ent)

fonction qui sert à mettre automatiquement les images de l'entité à jour. pour qu'elle continue son animation soit en incrémentant le chiffre correspondant au numéro de l'image soit en le décrémentant et en changeant l'indice d'animation à chaque fois que nécéssaire

— void met_a_jour_images_sprite (entite ∗ent)

fonction qui sert à mettre automatiquement les images de l'entité à jour. pour qu'elle continue son animation (il s'agit de la version 2 donc en découpant un morceau de l'image en question)

— void charger_img_separees (entite ∗ent, SDL_Renderer ∗rendu)

fonction qui sert à charger les images lorsque l'animation est faite avec des images séparées.

— void charger_img_sprite (entite ∗ent, SDL_Renderer ∗rendu)

— void charger_img_sprite_reverse (entite ∗ent, SDL_Renderer ∗rendu)

fonction qui sert à charger les images inverser pour le mode 1 vs 1

5.20.1 Description détaillée

Contient toutes les fonctions relatives à la gestion et à la manipulation des vague d'entités.

Auteur

Lazare Maclouf

Version

2

Date

30/03/2022

5.20.2 Documentation des fonctions

5.20.2.1 ajouter_entite_survivant()

```
t_wave* ajouter_entite_survivant (
            t_wave * vague )
```

ajoute une entité à la liste et met le pointeur passé en paramètre sur le nouvel élément. Pour ajouter un nouvel element on fait appel à la fonction creer_vague()

Paramètres

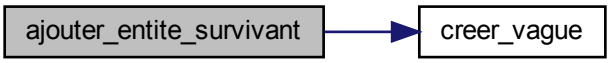
t_wave	∗vague

Renvoie

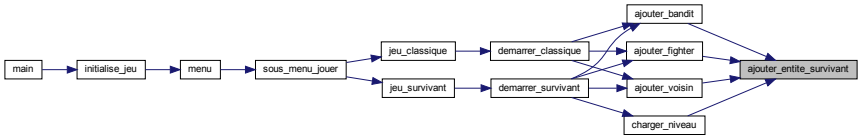
pointeur sur t_wave

Définition à la ligne 116 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.20.2.2 charger_img_separees()

```
void charger_img_separees (
            entite * ent,
            SDL_Renderer * rendu )
```

fonction qui sert à charger les images lorsque l'animation est faite avec des images séparées.

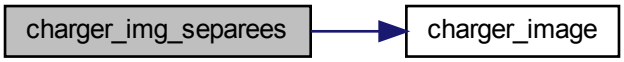
fonction qui sert à charger les images lorsque l'animation est faite avec une seule image pour toute les positions de l'entité

Paramètres

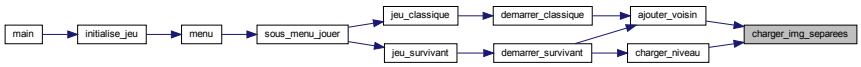
entite	∗ent

Définition à la ligne 344 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



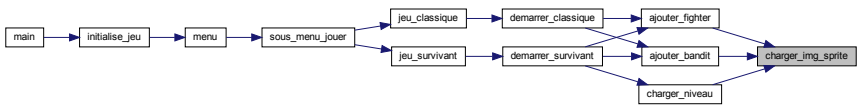
5.20.2.3 charger_img_sprite()

```
void charger_img_sprite (
            entite * ent,
            SDL_Renderer * rendu )
```

Définition à la ligne 357 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.20.2.4 charger_img_sprite_reverse()

```
void charger_img_sprite_reverse (
            entite * ent,
            SDL_Renderer * rendu )
```

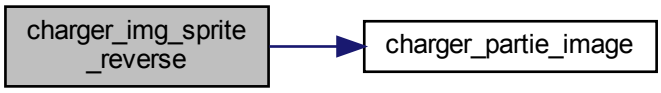
fonction qui sert à charger les images inverser pour le mode 1 vs 1

Paramètres

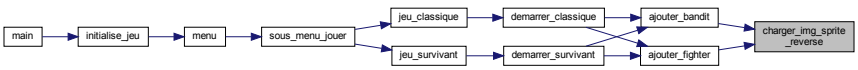
entite	∗ent

Définition à la ligne 367 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.20.2.5 compter_elem()

```
int compter_elem (
            t_wave * vague )
```

compte le nombre de vague restante (et renvoie le nombre d'entité encore à tuer)

Paramètres

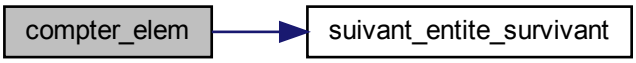
t_wave	∗vague

Renvoie

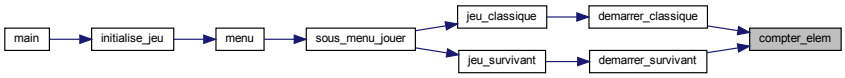
int nb

Définition à la ligne 214 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.20.2.6 creer_vague()

```
t_wave* creer_vague ( )
```

crée une variable de type t_wave (une vague).

Paramètres

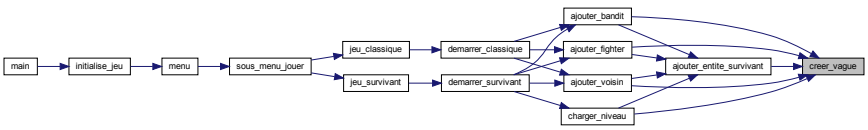
rien	

Renvoie

pointeur sur t_wave

Définition à la ligne 22 du fichier vague.c.

Voici le graphe des appelants de cette fonction :



5.20.2.7 deb_liste_vague()

```
t_wave* deb_liste_vague (
            t_wave * vague )
```

place le pointeur passé en paramètre au début de la liste. il s'agit d'une boucle. tant qu'on est pas au début de la liste on continue à reculer dans la liste à l'aide la fonction precedent_entite_survivant

Paramètres

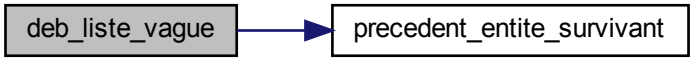
t_wave	∗vague

Renvoie

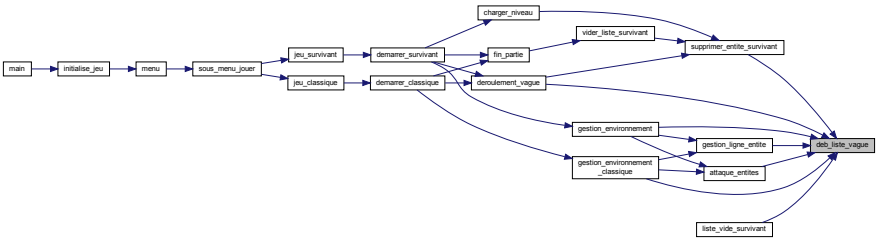
pointeur sur t_wave

Définition à la ligne 86 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.20.2.8 fin_liste_survivant()

```
t_wave* fin_liste_survivant (
            t_wave * vague )
```

place le pointeur passé en paramètre à la fin de la liste il s'agit d'une boucle. tant qu'on est pas à la fin de la liste on continue à avancer dans la liste à l'aide la fonction suivant_entite_survivant

Paramètres

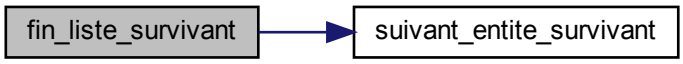
t_wave	∗vague

Renvoie

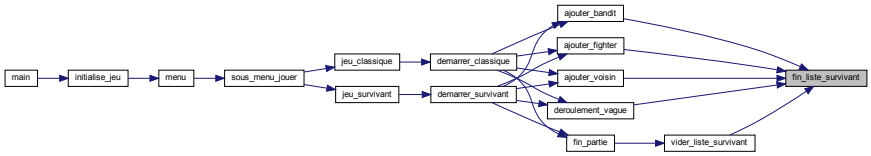
pointeur sur t_wave (NULL si on est hors liste et l'élément suivant si il est différent de NULL)

Définition à la ligne 68 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.20.2.9 liste_vide_survivant()

```
int liste_vide_survivant (
            t_wave * vague )
```

indique si la liste est vide ou pas au niveau survivant. Pour le savoir, on se place au début de la liste puis on regarde si le premier élément est null si c'est le cas, la liste est vide

Paramètres

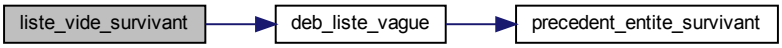
wave	∗vague

Renvoie

1 si hors_liste 0 sinon

Définition à la ligne 105 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



5.20.2.10 met_a_jour_images_separees()

```
void met_a_jour_images_separees (
            entite * ent )
```

fonction qui sert à mettre automatiquement les images de l'entité à jour. pour qu'elle continue son animation soit en incrémentant le chiffre correspondant au numéro de l'image soit en le décrémentant et en changeant l'indice d'animation à chaque fois que nécéssaire

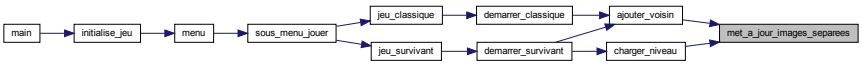
Paramètres

entite	∗ent

Définition à la ligne 249 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.20.2.11 met_a_jour_images_sprite()

```
void met_a_jour_images_sprite (
            entite * ent )
```

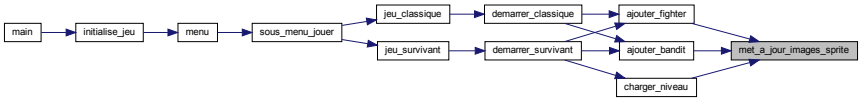
fonction qui sert à mettre automatiquement les images de l'entité à jour. pour qu'elle continue son animation (il s'agit de la version 2 donc en découpant un morceau de l'image en question)

Paramètres

entite	∗ent

Définition à la ligne 292 du fichier vague.c.

Voici le graphe des appelants de cette fonction :



5.20.2.12 met_a_jour_img_argent()

```
void met_a_jour_img_argent (
            joueur * player,
            SDL_Renderer * rendu )
```

met la somme d'agent a jour en fonction de ce que le player a dépensé et gangé en changeant les imgages pour monter la quantité d'argent

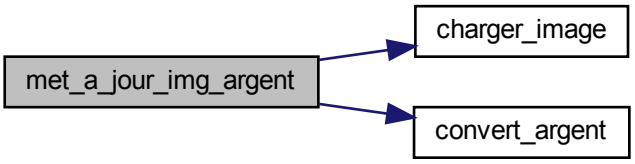
\fnvoid met_a_jour_img_argent(joueur ∗player, SDL_Renderer ∗rendu)

Paramètres

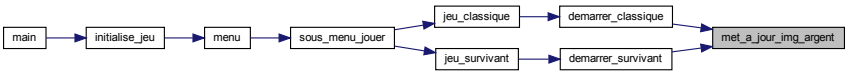
entite	joueur ∗player, SDL_Renderer ∗rendu

Définition à la ligne 232 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.20.2.13 precedent_entite_survivant()

```
t_wave* precedent_entite_survivant (
            t_wave * vague )
```

place le pointeur passé en paramètre de la liste sur l'élément précédent

Paramètres

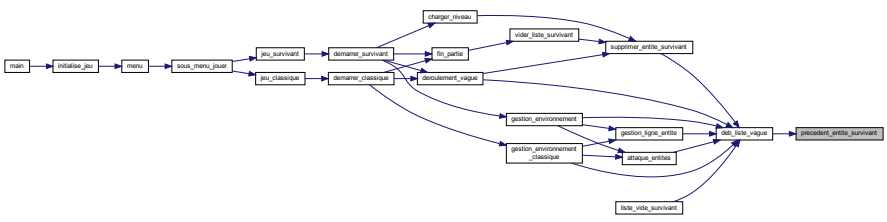
t_wave	∗vague

Renvoie

pointeur sur t_wave (NULL si on est hors liste et l'élément précédent si il est différent de NULL)

Définition à la ligne 53 du fichier vague.c.

Voici le graphe des appelants de cette fonction :



5.20.2.14 suivant_entite_survivant()

```
t_wave* suivant_entite_survivant (
            t_wave * vague )
```

place le pointeur passé en paramètre de la liste sur l'élément suivant

Paramètres

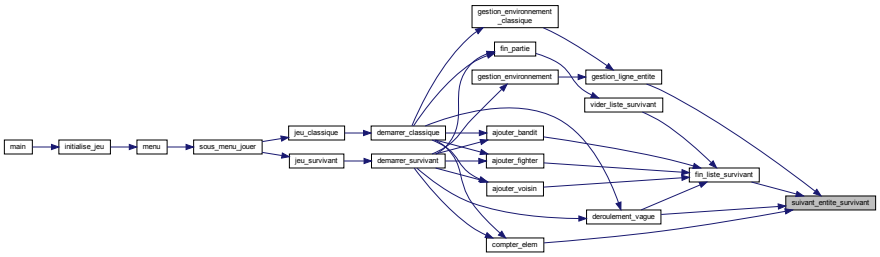
t_wave	∗vague

Renvoie

pointeur sur t_wave

Définition à la ligne 39 du fichier vague.c.

Voici le graphe des appelants de cette fonction :



5.20.2.15 supprimer_entite_survivant()

```
t_wave* supprimer_entite_survivant (
            t_wave * vague )
```

supprime l'entité au niveau du pointeur passé en paramètres. retourne un pointeur sur l'élément précédent si il ya un élement précédent, sinon le suivant, sinon retourne NULL

Paramètres

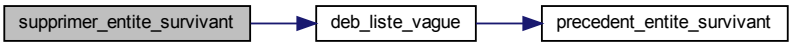
t_wave	∗vague

Renvoie

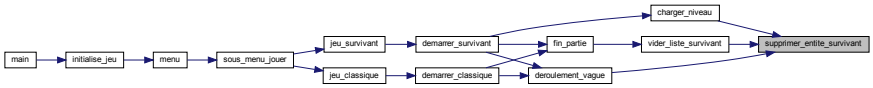
pointeur sur t_wave (sur l'élément précédent, le suivant ou NULL).

Définition à la ligne 132 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.20.2.16 vider_liste_survivant()

```
t_wave* vider_liste_survivant (
            t_wave * vague )
```

fonction qui sert à vider la liste.

Paramètres

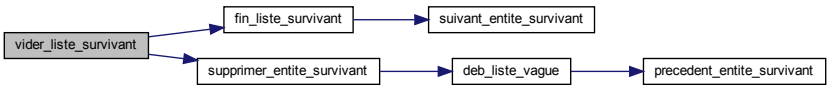
t_wave	*vague

Renvoie

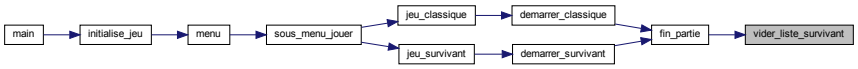
pointeur sur t_wave

Définition à la ligne 198 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



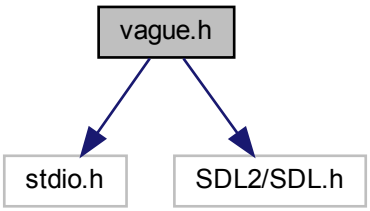
Voici le graphe des appelants de cette fonction :



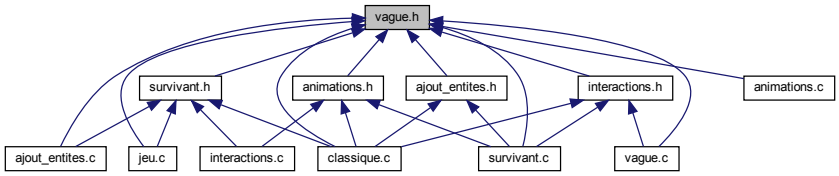
5.21 Référence du fichier vague.h

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

Graphe des dépendances par inclusion de vague.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Structures de données

— struct entity
— struct wave

Définitions de type

— typedef struct entity entite
— typedef struct wave t_wave

Fonctions

- t_wave * creer_vague ()
  crée une variable de type t_wave (une vague).
- char * creer_char ()
- int hors_liste_survivant (t_wave *vague)
- t_wave * suivant_entite_survivant (t_wave *vague)
  place le pointeur passé en paramètre de la liste sur l'élément suivant
- t_wave * precedent_entite_survivant (t_wave *vague)
  place le pointeur passé en paramètre de la liste sur l'élément précédent
- t_wave * fin_liste_survivant (t_wave *vague)
  place le pointeur passé en paramètre à la fin de la liste il s'agit d'une boucle. tant qu'on est pas à la fin de la liste on continue à avancer dans la liste à l'aide la fonction suivant_entite_survivant
- t_wave * deb_liste_vague (t_wave *vague)
  place le pointeur passé en paramètre au début de la liste. il s'agit d'une boucle. tant qu'on est pas au début de la liste on continue à reculer dans la liste à l'aide la fonction precedent_entite_survivant
- int liste_vide_survivant (t_wave *vague)
  indique si la liste est vide ou pas au niveau survivant. Pour le savoir, on se place au début de la liste puis on regarde si le premier élément est null si c'est le cas, la liste est vide
- t_wave * ajouter_entite_survivant (t_wave *vague)
  ajoute une entité à la liste et met le pointeur passé en paramètre sur le nouvel élément. Pour ajouter un nouvel element on fait appel à la fonction creer_vague()
- t_wave * supprimer_entite_survivant (t_wave *vague)
  supprime l'entité au niveau du pointeur passé en paramètres. retourne un pointeur sur l'élément précédent si il ya un élement précédent, sinon le suivant, sinon retourne NULL
- t_wave * vider_liste_survivant (t_wave *vague)
  fonction qui sert à vider la liste.
- int compter_elem (t_wave *vague)
  compte le nombre de vague restante (et renvoie le nombre d'entité encore à tuer)
- void met_a_jour_img_argent (joueur *j, SDL_Renderer *rendu)
  met la somme d'agent a jour en fonction de ce que le player a dépensé et gangé en changeant les imgages pour monter la quantité d'argent
- void met_a_jour_images_separees (entite *ent)
  fonction qui sert à mettre automatiquement les images de l'entité à jour. pour qu'elle continue son animation soit en incrémentant le chiffre correspondant au numéro de l'image soit en le décrémentant et en changeant l'indice d'animation à chaque fois que nécéssaire
- void met_a_jour_images_sprite (entite *ent)
  fonction qui sert à mettre automatiquement les images de l'entité à jour. pour qu'elle continue son animation (il s'agit de la version 2 donc en découpant un morceau de l'image en question)
- void charger_img_separees (entite *ent, SDL_Renderer *rendu)
  fonction qui sert à charger les images lorsque l'animation est faite avec des images séparées.
- void charger_img_sprite (entite *ent, SDL_Renderer *rendu)
- void charger_img_sprite_reverse (entite *ent, SDL_Renderer *rendu)
  fonction qui sert à charger les images inverser pour le mode 1 vs 1

5.21.1 Documentation des définitions de type

5.21.1.1 entite

```
typedef struct entity entite
```

5.21.1.2 t_wave

```
typedef struct wave t_wave
```

5.21.2 Documentation des fonctions

5.21.2.1 ajouter_entite_survivant()

```
t_wave * ajouter_entite_survivant (
            t_wave * vague )
```

ajoute une entité à la liste et met le pointeur passé en paramètre sur le nouvel élément. Pour ajouter un nouvel element on fait appel à la fonction creer_vague()

Paramètres

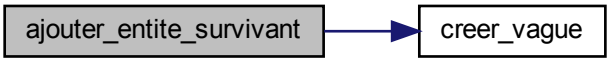
t_wave	*vague

Renvoie

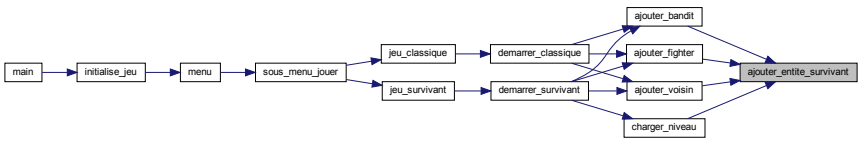
pointeur sur t_wave

Définition à la ligne 116 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.21.2.2 charger_img_separees()

```
void charger_img_separees (
            entite * ent,
            SDL_Renderer * rendu )
```

fonction qui sert à charger les images lorsque l'animation est faite avec des images séparées.

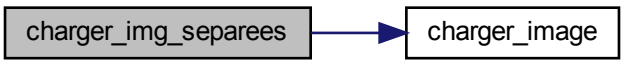
fonction qui sert à charger les images lorsque l'animation est faite avec une seule image pour toute les positions de l'entité

Paramètres

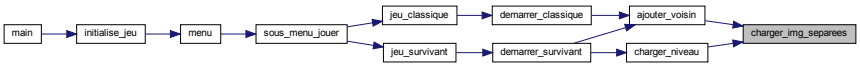
entite	∗ent

Définition à la ligne 344 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



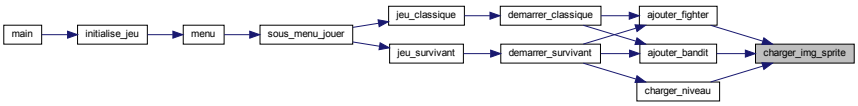
5.21.2.3 charger_img_sprite()

```
void charger_img_sprite (
            entite * ent,
            SDL_Renderer * rendu )
```

Définition à la ligne 357 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.21.2.4 charger_img_sprite_reverse()

```
void charger_img_sprite_reverse (
            entite * ent,
            SDL_Renderer * rendu )
```

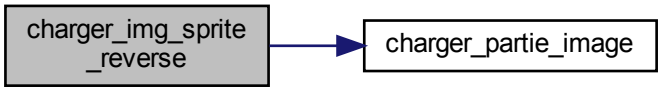
fonction qui sert à charger les images inverser pour le mode 1 vs 1

Paramètres

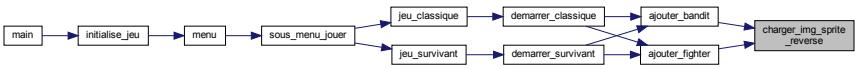
entite	∗ent

Définition à la ligne 367 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.21.2.5 compter_elem()

```
int compter_elem (
            t_wave * vague )
```

compte le nombre de vague restante (et renvoie le nombre d'entité encore à tuer)

Paramètres

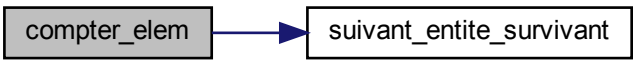
t_wave	*vague

Renvoie

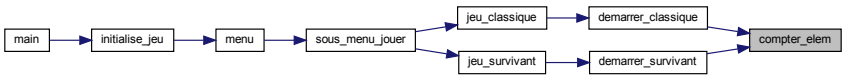
int nb

Définition à la ligne 214 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.21.2.6 creer_char()

```
char* creer_char ( )
```

5.21.2.7 creer_vague()

```
t_wave * creer_vague ( )
```

crée une variable de type t_wave (une vague).

Paramètres

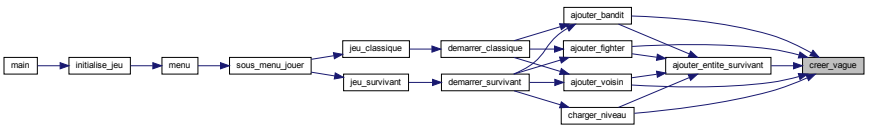
rien	

Renvoie

pointeur sur t_wave

Définition à la ligne 22 du fichier vague.c.

Voici le graphe des appelants de cette fonction :



5.21.2.8 deb_liste_vague()

```
t_wave * deb_liste_vague (
            t_wave * vague )
```

place le pointeur passé en paramètre au début de la liste. il s'agit d'une boucle. tant qu'on est pas au début de la liste on continue à reculer dans la liste à l'aide la fonction precedent_entite_survivant

Paramètres

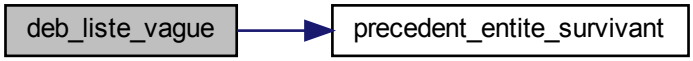
t_wave	*vague

Renvoie

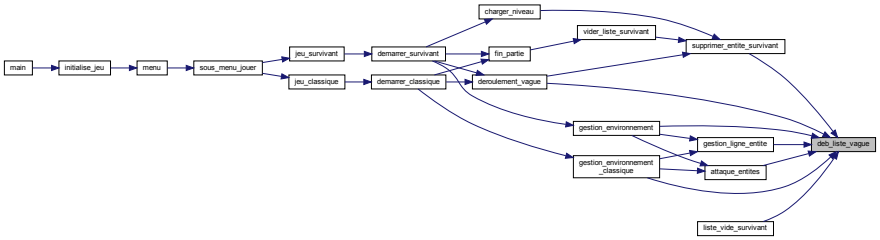
pointeur sur t_wave

Définition à la ligne 86 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.21.2.9 fin_liste_survivant()

```
t_wave * fin_liste_survivant (
            t_wave * vague )
```

place le pointeur passé en paramètre à la fin de la liste il s'agit d'une boucle. tant qu'on est pas à la fin de la liste on continue à avancer dans la liste à l'aide la fonction suivant_entite_survivant

Paramètres

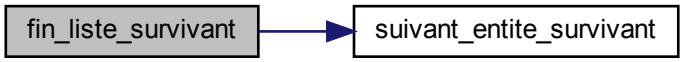
t_wave	*vague

Renvoie

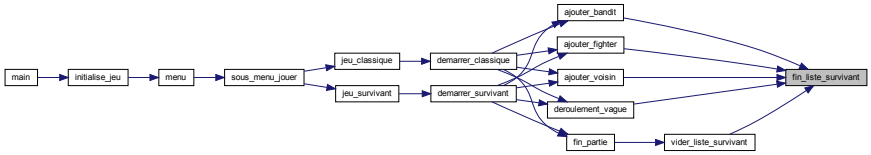
pointeur sur t_wave (NULL si on est hors liste et l'élément suivant si il est différent de NULL)

Définition à la ligne 68 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.21.2.10 hors_liste_survivant()

```
int hors_liste_survivant (
            t_wave * vague )
```

5.21.2.11 liste_vide_survivant()

```
int liste_vide_survivant (
            t_wave * vague )
```

indique si la liste est vide ou pas au niveau survivant. Pour le savoir, on se place au début de la liste puis on regarde si le premier élément est null si c'est le cas, la liste est vide

Paramètres

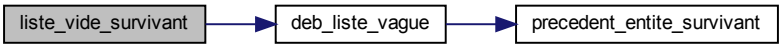
wave	∗vague

Renvoie

1 si hors_liste 0 sinon

Définition à la ligne 105 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



5.21.2.12 met_a_jour_images_separees()

```
void met_a_jour_images_separees (
            entite * ent )
```

fonction qui sert à mettre automatiquement les images de l'entité à jour. pour qu'elle continue son animation soit en incrémentant le chiffre correspondant au numéro de l'image soit en le décrémentant et en changeant l'indice d'animation à chaque fois que nécéssaire

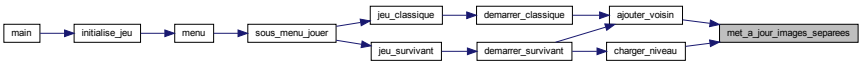
Paramètres

entite	∗ent

Définition à la ligne 249 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



5.21.2.13 met_a_jour_images_sprite()

```
void met_a_jour_images_sprite (
            entite * ent )
```

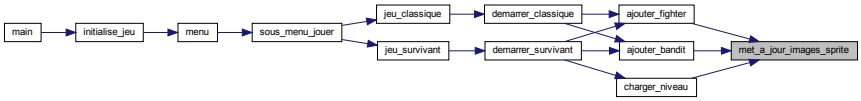
fonction qui sert à mettre automatiquement les images de l'entité à jour. pour qu'elle continue son animation (il s'agit de la version 2 donc en découpant un morceau de l'image en question)

Paramètres

entite	∗ent

Définition à la ligne 292 du fichier vague.c.

Voici le graphe des appelants de cette fonction :



5.21.2.14 met_a_jour_img_argent()

```
void met_a_jour_img_argent (
            joueur * player,
            SDL_Renderer * rendu )
```

met la somme d'agent a jour en fonction de ce que le player a dépensé et gangé en changeant les imgages pour monter la quantité d'argent

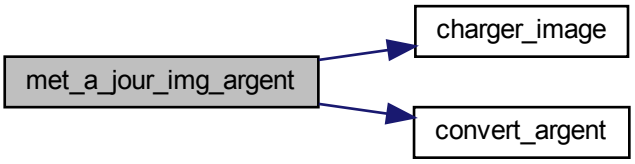
\fnvoid met_a_jour_img_argent(joueur ∗player, SDL_Renderer ∗rendu)

Paramètres

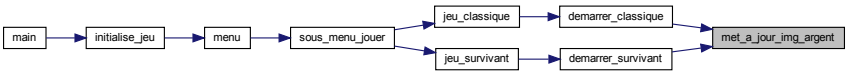
entite	joueur ∗player, SDL_Renderer ∗rendu

Définition à la ligne 232 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.21.2.15 precedent_entite_survivant()

```
t_wave * precedent_entite_survivant (
            t_wave * vague )
```

place le pointeur passé en paramètre de la liste sur l'élément précédent

Paramètres

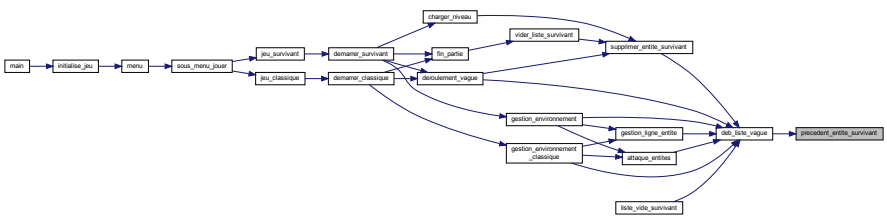
t_wave	∗vague

Renvoie

pointeur sur t_wave (NULL si on est hors liste et l'élément précédent si il est différent de NULL)

Définition à la ligne 53 du fichier vague.c.

Voici le graphe des appelants de cette fonction :



5.21.2.16 suivant_entite_survivant()

```
t_wave * suivant_entite_survivant (
            t_wave * vague )
```

place le pointeur passé en paramètre de la liste sur l'élément suivant

Paramètres

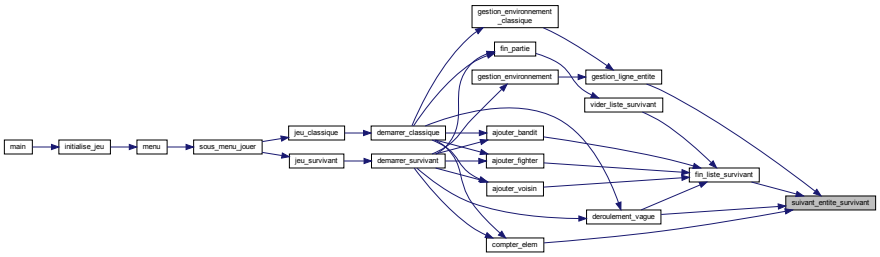
t_wave	∗vague

Renvoie

pointeur sur t_wave

Définition à la ligne 39 du fichier vague.c.

Voici le graphe des appelants de cette fonction :



5.21.2.17 supprimer_entite_survivant()

```
t_wave * supprimer_entite_survivant (
            t_wave * vague )
```

supprime l'entité au niveau du pointeur passé en paramètres. retourne un pointeur sur l'élément précédent si il ya un élement précédent, sinon le suivant, sinon retourne NULL

Paramètres

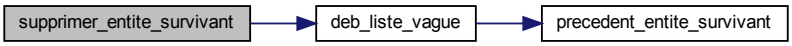
t_wave	∗vague

Renvoie

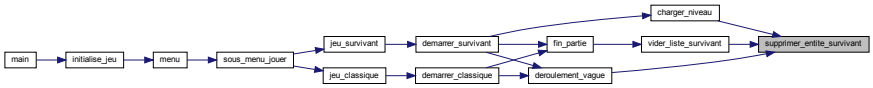
pointeur sur t_wave (sur l'élément précédent, le suivant ou NULL).

Définition à la ligne 132 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.21.2.18 vider_liste_survivant()

```
t_wave * vider_liste_survivant (
            t_wave * vague )
```

fonction qui sert à vider la liste.

Paramètres

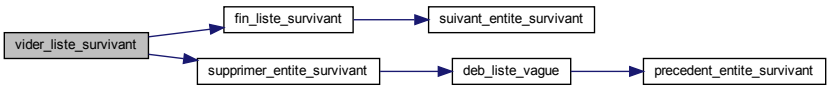
t_wave	*vague

Renvoie

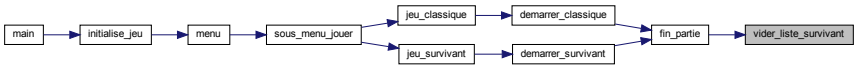
pointeur sur t_wave

Définition à la ligne 198 du fichier vague.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



Index

afficher_survivant
    animations.c, 30
    animations.h, 33
ajout_entites.c, 19
    ajouter_bandit, 21
    ajouter_fighter, 22
    ajouter_voisin, 23
    TAILLE_FENETRE, 20
    Y_ENTITY, 20
ajout_entites.h, 24
    ajouter_bandit, 25
    ajouter_fighter, 26
    ajouter_voisin, 27
ajouter_bandit
    ajout_entites.c, 21
    ajout_entites.h, 25
ajouter_entite_survivant
    vague.c, 115
    vague.h, 129
ajouter_fighter
    ajout_entites.c, 22
    ajout_entites.h, 26
ajouter_voisin
    ajout_entites.c, 23
    ajout_entites.h, 27
animation_attaque
    animations.c, 30
    animations.h, 34
animation_tir_gauche
    animations.c, 31
    animations.h, 35
animations.c, 28
    afficher_survivant, 30
    animation_attaque, 30
    animation_tir_gauche, 31
    TAILLE_FENETRE, 29
animations.h, 32
    afficher_survivant, 33
    animation_attaque, 34
    animation_tir_gauche, 35
argent
    joue, 14
argent_img
    joue, 14
argent_joueur
    survivant.c, 96
attaque
    entity, 9
attaque_entites
    interactions.c, 49
    interactions.h, 56
audio
    interface.c, 72
audio.c, 35
    audio_initialise, 36
    AUDIO_PATH, 36
    PFLAG, 36
    print_init_flags, 37
audio.h, 38
    audio_initialise, 39
    print_init_flags, 39
audio_initialise
    audio.c, 36
    audio.h, 39
AUDIO_PATH
    audio.c, 36

camp
    joue, 14
charger_image
    interface.c, 63
    interface.h, 74
charger_img
    entity, 9
charger_img_separees
    vague.c, 116
    vague.h, 129
charger_img_sprite
    vague.c, 117
    vague.h, 130
charger_img_sprite_reverse
    vague.c, 117
    vague.h, 131
charger_niveau
    survivant.c, 97
    survivant.h, 106
charger_partie_image
    interface.c, 64
    interface.h, 75
classique.c, 40
    demarrer_classique, 42
    etat_partie_classique, 43
    gestion_environnement_classique, 44
    TAILLE_FENETRE, 41
    X_DEF, 41
    X_TIR, 41
    Y_DEF, 42
    Y_TIR, 42
classique.h, 45

demarrer_classique, 46
compter_elem
vague.c, 118
vague.h, 132
convert_argent
interface.c, 65
interface.h, 76
create
joue, 14
creer_char
vague.h, 132
creer_defense
interactions.c, 49
interactions.h, 56
creer_joueur
survivant.c, 98
survivant.h, 107
creer_tir
interactions.c, 50
interactions.h, 57
creer_vague
vague.c, 119
vague.h, 133

deb_liste_vague
vague.c, 120
vague.h, 133
def, 7
degat, 7
indice_vie, 7
joue, 14
nom_fichier, 8
temps, 8
x, 8
y, 8
defense
structures.h, 94
degat
def, 7
entity, 9
demarrage
interface.c, 65
interface.h, 76
demarrer_classique
classique.c, 42
classique.h, 46
demarrer_survivant
survivant.c, 99
survivant.h, 108
deroulement_vague
survivant.c, 100
survivant.h, 109
dessiner_rectangle_plein
interface.c, 66
interface.h, 77
dessiner_rectangle_vide
interface.c, 67
interface.h, 78

ent
wave, 18
entite
vague.h, 128
entity, 8
attaque, 9
charger_img, 9
degat, 9
h, 10
h_image, 10
met_a_jour, 10
montant, 10
nb_pos, 10
nb_pos_attaque, 10
nom_fichier, 11
nom_fichier_attaque, 11
pv, 11
temps, 11
type, 11
w, 11
w_image, 12
x, 12
x_barre, 12
x_image, 12
y, 12
y_barre, 12
y_image, 13
etat_defense
interactions.c, 51
interactions.h, 58
etat_partie_classique
classique.c, 43
etat_partie_survivant
survivant.c, 102
survivant.h, 111
etat_tir
interactions.c, 51
interactions.h, 58

fichier_existe
interactions.c, 52
interactions.h, 59
fin_liste_survivant
vague.c, 120
vague.h, 134
fin_partie
survivant.c, 103
survivant.h, 112

gestion_environnement
interactions.c, 53
interactions.h, 60
gestion_environnement_classique
classique.c, 44
gestion_ligne_entite
interactions.c, 53
interactions.h, 60

h

entity, 10
h_image
entity, 10
hors_liste_survivant
vague.h, 135

indice_vie
def, 7
tir_s, 17
initialise_jeu
interface.c, 67
interface.h, 78
interactions.c, 47
attaque_entites, 49
creer_defense, 49
creer_tir, 50
etat_defense, 51
etat_tir, 51
fichier_existe, 52
gestion_environnement, 53
gestion_ligne_entite, 53
interactions.h, 54
attaque_entites, 56
creer_defense, 56
creer_tir, 57
etat_defense, 58
etat_tir, 58
fichier_existe, 59
gestion_environnement, 60
gestion_ligne_entite, 60
interface.c, 61
audio, 72
charger_image, 63
charger_partie_image, 64
convert_argent, 65
demarrage, 65
dessiner_rectangle_plein, 66
dessiner_rectangle_vide, 67
initialise_jeu, 67
menu, 68
MENU_X, 63
quit_message, 70
select_multi, 70
sous_menu_jouer, 71
interface.h, 73
charger_image, 74
charger_partie_image, 75
convert_argent, 76
demarrage, 76
dessiner_rectangle_plein, 77
dessiner_rectangle_vide, 78
initialise_jeu, 78
menu, 79
menu_jouer_difficulte, 81
quit_message, 81
select_multi, 81
sous_menu_jouer, 82

jeu.c, 84
jeu_classique, 85
jeu_survivant, 86
NB_NIV_SURVIVANT, 85
jeu.h, 88
jeu_classique, 88
jeu_survivant, 89
jeu_classique
jeu.c, 85
jeu.h, 88
jeu_survivant
jeu.c, 86
jeu.h, 89
joue, 13
argent, 14
argent_img, 14
camp, 14
create, 14
def, 14
nom, 14
pv, 15
t, 15
x_create, 15
joueur
structures.h, 94

liste_vide_survivant
vague.c, 121
vague.h, 135

main
main.c, 92
main.c, 91
main, 92
menu
interface.c, 68
interface.h, 79
menu_jouer_difficulte
interface.h, 81
MENU_X
interface.c, 63
mes, 15
nom_fichier, 16
temps, 16
x, 16
y, 16
message
structures.h, 94
message_box
survivant.c, 103
survivant.h, 112
met_a_jour
entity, 10
met_a_jour_images_separees
vague.c, 122
vague.h, 136
met_a_jour_images_sprite
vague.c, 122
vague.h, 136
met_a_jour_img_argent

vague.c, 123
vague.h, 137
montant
entity, 10

NB_NIV_SURVIVANT
jeu.c, 85
nb_pos
entity, 10
nb_pos_attaque
entity, 10
nom
joue, 14
nom_fichier
def, 8
entity, 11
mes, 16
tir_s, 17
nom_fichier_attaque
entity, 11

PFLAG
audio.c, 36
prec
wave, 18
precedent_entite_survivant
vague.c, 124
vague.h, 138
print_init_flags
audio.c, 37
audio.h, 39
pv
entity, 11
joue, 15
pv_joueur
survivant.c, 96

quit_message
interface.c, 70
interface.h, 81

README.md, 93
return_message
survivant.c, 104
survivant.h, 113

select_multi
interface.c, 70
interface.h, 81
sous_menu_jouer
interface.c, 71
interface.h, 82
structures.h, 93
defense, 94
joueur, 94
message, 94
tir, 94
suiv
wave, 18
suivant_entite_survivant
vague.c, 125
vague.h, 139
supprimer_entite_survivant
vague.c, 125
vague.h, 139
survivant.c, 95
argent_joueur, 96
charger_niveau, 97
creer_joueur, 98
demarrer_survivant, 99
deroulement_vague, 100
etat_partie_survivant, 102
fin_partie, 103
message_box, 103
pv_joueur, 96
return_message, 104
V, 96
x_def, 97
y_def, 97
y_entity, 97
survivant.h, 105
charger_niveau, 106
creer_joueur, 107
demarrer_survivant, 108
deroulement_vague, 109
etat_partie_survivant, 111
fin_partie, 112
message_box, 112
return_message, 113

t
joue, 15
t_wave
vague.h, 128
TAILLE_FENETRE
ajout_entites.c, 20
animations.c, 29
classique.c, 41
temps
def, 8
entity, 11
mes, 16
tir
structures.h, 94
tir_s, 16
indice_vie, 17
nom_fichier, 17
x, 17
y, 17
type
entity, 11

V
survivant.c, 96
vague.c, 114
ajouter_entite_survivant, 115
charger_img_separees, 116
charger_img_sprite, 117

charger_img_sprite_reverse, 117
compter_elem, 118
creer_vague, 119
deb_liste_vague, 120
fin_liste_survivant, 120
liste_vide_survivant, 121
met_a_jour_images_separees, 122
met_a_jour_images_sprite, 122
met_a_jour_img_argent, 123
precedent_entite_survivant, 124
suivant_entite_survivant, 125
supprimer_entite_survivant, 125
vider_liste_survivant, 126
vague.h, 127
ajouter_entite_survivant, 129
charger_img_separees, 129
charger_img_sprite, 130
charger_img_sprite_reverse, 131
compter_elem, 132
creer_char, 132
creer_vague, 133
deb_liste_vague, 133
entite, 128
fin_liste_survivant, 134
hors_liste_survivant, 135
liste_vide_survivant, 135
met_a_jour_images_separees, 136
met_a_jour_images_sprite, 136
met_a_jour_img_argent, 137
precedent_entite_survivant, 138
suivant_entite_survivant, 139
supprimer_entite_survivant, 139
t_wave, 128
vider_liste_survivant, 140
vider_liste_survivant
vague.c, 126
vague.h, 140

w
entity, 11
w_image
entity, 12
wave, 18
ent, 18
prec, 18
suiv, 18

x
def, 8
entity, 12
mes, 16
tir_s, 17
x_barre
entity, 12
x_create
joue, 15
X_DEF
classique.c, 41
x_def
survivant.c, 97
x_image
entity, 12
X_TIR
classique.c, 41

y
def, 8
entity, 12
mes, 16
tir_s, 17
y_barre
entity, 12
Y_DEF
classique.c, 42
y_def
survivant.c, 97
Y_ENTITY
ajout_entites.c, 20
y_entity
survivant.c, 97
y_image
entity, 13
Y_TIR
classique.c, 42