

Tutored project - C++
dev

Généré par Doxygen 1.8.11

Table des matières

1	Index hiérarchique	1
1.1	Hiérarchie des classes	1
2	Index des classes	3
2.1	Liste des classes	3
3	Index des fichiers	5
3.1	Liste des fichiers	5
4	Documentation des classes	7
4.1	Référence de la classe Bomb	7
4.1.1	Description détaillée	10
4.1.2	Documentation des constructeurs et destructeur	10
4.1.2.1	Bomb(TileSystem *tilesys, sf : :Vector2f center, sf : :Vector2f position, int health, Player *player, sf : :Time duration, unsigned int blastRadius)	10
4.1.3	Documentation des fonctions membres	10
4.1.3.1	draw(GameWindow *window)	10
4.1.3.2	getPlayer()	11
4.1.3.3	updateState(Controller *controller, sf : :Time &elapsed, Tilemap *world)	11
4.1.4	Documentation des données membres	11
4.1.4.1	m_blastRadius	11
4.1.4.2	m_duration	11
4.1.4.3	m_maxSize	11
4.1.4.4	m_player	11
4.2	Référence de la classe Button	12

4.2.1	Description détaillée	14
4.2.2	Documentation des constructeurs et destructeur	14
4.2.2.1	Button(TileSystem *tilesys, int x, int y, string text, sf : :Font *font, void(*callback)(void *))	14
4.2.2.2	~Button()	14
4.2.3	Documentation des fonctions membres	14
4.2.3.1	callback(void *args)	14
4.2.3.2	draw(GameWindow *)	15
4.2.3.3	manageEvents(sf : :Event &event, void *args=NULL)	15
4.2.4	Documentation des données membres	15
4.2.4.1	m_callback	15
4.2.4.2	m_text	15
4.3	Référence de la classe Controller	15
4.3.1	Description détaillée	18
4.3.2	Documentation des constructeurs et destructeur	18
4.3.2.1	Controller(GameWindow *window)	18
4.3.2.2	~Controller()	18
4.3.3	Documentation des fonctions membres	19
4.3.3.1	manageEvents()=0	19
4.3.3.2	notifyUpdate()=0	19
4.3.3.3	start()=0	19
4.3.4	Documentation des données membres	19
4.3.4.1	m_fonts	19
4.3.4.2	m_musics	19
4.3.4.3	m_textures	19
4.3.4.4	m_tilesets	19
4.3.4.5	m_tilesystems	19
4.3.4.6	m_window	20
4.4	Référence de la classe Entity	20
4.4.1	Description détaillée	23
4.4.2	Documentation des constructeurs et destructeur	23

4.4.2.1	Entity(TileSystem *tilesys, map< Orientation, sf::IntRect > bBoxes, sf::Vector2f center, sf::Vector2f position, int health)	23
4.4.2.2	~Entity()	23
4.4.3	Documentation des fonctions membres	23
4.4.3.1	getHealth()	24
4.4.3.2	getPosition()	24
4.4.3.3	updateState(Controller *controller, sf::Time &elapsed, Tilemap *world)=0 . . .	24
4.4.4	Documentation des données membres	24
4.4.4.1	m_bBoxes	24
4.4.4.2	m_center	24
4.4.4.3	m_health	24
4.4.4.4	m_maxHealth	25
4.4.4.5	m_position	25
4.4.4.6	m_tick	25
4.4.4.7	m_timeAlive	25
4.4.4.8	m_watcher	25
4.5	Référence de la classe Game	25
4.5.1	Description détaillée	28
4.5.2	Documentation des constructeurs et destructeur	28
4.5.2.1	Game(GameWindow *window)	28
4.5.2.2	~Game()	29
4.5.3	Documentation des fonctions membres	29
4.5.3.1	getEntities()	29
4.5.3.2	getPlayer(int i)	29
4.5.3.3	manageEvents()	29
4.5.3.4	notifyUpdate()	29
4.5.3.5	start()	30
4.5.4	Documentation des données membres	30
4.5.4.1	hasWon	30
4.5.4.2	m_backgroundMap	30
4.5.4.3	m_entities	30

4.5.4.4	<code>m_isPlaying</code>	30
4.5.4.5	<code>m_physicalMap</code>	30
4.5.4.6	<code>m_player1</code>	30
4.5.4.7	<code>m_player2</code>	30
4.5.4.8	<code>m_timer</code>	30
4.5.4.9	<code>m_view</code>	31
4.6	Référence de la classe <code>GameWindow</code>	31
4.6.1	Description détaillée	33
4.6.2	Documentation des fonctions membres	33
4.6.2.1	<code>create(int width, int height)</code>	33
4.6.2.2	<code>create(int tileSize, int width, int height)</code>	33
4.6.2.3	<code>getHeight()</code>	33
4.6.2.4	<code>getTileSize()</code>	33
4.6.2.5	<code>getWidth()</code>	34
4.6.2.6	<code>manageEvents(sf : :Event &event, void *args=NULL)</code>	34
4.6.2.7	<code>rectifyRatio()</code>	34
4.6.3	Documentation des données membres	34
4.6.3.1	<code>m_height</code>	34
4.6.3.2	<code>m_tileSize</code>	34
4.6.3.3	<code>m_toRectifyRatio</code>	34
4.6.3.4	<code>m_width</code>	34
4.7	Référence de la classe <code>IDrawable</code>	35
4.7.1	Description détaillée	37
4.7.2	Documentation des constructeurs et destructeur	37
4.7.2.1	<code>IDrawable(TileSystem *tilesys)</code>	37
4.7.3	Documentation des fonctions membres	37
4.7.3.1	<code>draw(GameWindow *window)=0</code>	37
4.7.4	Documentation des données membres	37
4.7.4.1	<code>m_tilesys</code>	37
4.8	Référence de la classe <code>IHandlable</code>	37

4.8.1	Description détaillée	38
4.8.2	Documentation des fonctions membres	39
4.8.2.1	manageEvents(sf : :Event &event, void *args=NULL)=0	39
4.9	Référence de la classe LivingEntity	39
4.9.1	Description détaillée	42
4.9.2	Documentation des constructeurs et destructeur	42
4.9.2.1	LivingEntity(TileSystem *tilesys, map< Orientation, sf : :IntRect > bBoxes, sf↵ : :Vector2f center, sf : :Vector2f position, int health)	42
4.9.3	Documentation des fonctions membres	42
4.9.3.1	draw(GameWindow *window)	42
4.9.3.2	move(sf : :Vector2f potential, Tilemap *world)	42
4.9.3.3	setSpeed(sf : :Vector2f speed)	43
4.9.3.4	updateState(Controller *controller, sf : :Time &elapsed, Tilemap *world)	43
4.9.4	Documentation des données membres	43
4.9.4.1	m_orientation	43
4.9.4.2	m_speed	43
4.10	Référence de la classe MapTile	44
4.10.1	Description détaillée	46
4.10.2	Documentation des constructeurs et destructeur	46
4.10.2.1	MapTile(sf : :Sprite &s, TileType type=TileType : :DEFAULT)	46
4.10.2.2	MapTile(TileType type=TileType : :DEFAULT)	46
4.10.3	Documentation des fonctions membres	46
4.10.3.1	isBreakable()	46
4.10.3.2	isCollidable()	46
4.10.3.3	setBreakable(bool b)	46
4.10.3.4	setCollidable(bool c)	47
4.10.4	Documentation des données membres	47
4.10.4.1	m_breakable	47
4.10.4.2	m_collidable	47
4.11	Référence de la classe Menu	47
4.11.1	Description détaillée	50

4.11.2	Documentation des constructeurs et destructeur	50
4.11.2.1	Menu(GameWindow *window)	50
4.11.2.2	~Menu()	50
4.11.3	Documentation des fonctions membres	50
4.11.3.1	manageEvents()	50
4.11.3.2	notifyUpdate()	51
4.11.3.3	start()	51
4.11.4	Documentation des données membres	51
4.11.4.1	background	51
4.11.4.2	logo	51
4.11.4.3	playButton	51
4.11.4.4	quitButton	51
4.12	Référence de la structure nStream	52
4.12.1	Description détaillée	53
4.12.2	Documentation des fonctions membres	53
4.12.2.1	overflow(char c)	53
4.13	Référence de la classe Player	53
4.13.1	Description détaillée	56
4.13.2	Documentation des constructeurs et destructeur	56
4.13.2.1	Player(TileSystem *tilesys, map< Orientation, sf : :IntRect > bBoxes, sf : :↔ Vector2f center, sf : :Vector2f position, int health, vector< sf : :Keyboard : :Key > controls, TileSystem *bombTilesys)	56
4.13.3	Documentation des fonctions membres	56
4.13.3.1	dropBomb(sf : :Vector2f pos, vector< Entity * > *list)	56
4.13.3.2	hit()	57
4.13.3.3	isOnCoord(unsigned int i, unsigned int j, Tilemap *world)	57
4.13.3.4	manageEvents(sf : :Event &event, void *args=NULL)	57
4.13.3.5	notifyExplosion()	57
4.13.4	Documentation des données membres	57
4.13.4.1	m_blastPower	57
4.13.4.2	m_bombCount	58

4.13.4.3	<code>m_bombHasBeenDropped</code>	58
4.13.4.4	<code>m_bombInventory</code>	58
4.13.4.5	<code>m_bombTilesys</code>	58
4.13.4.6	<code>m_controls</code>	58
4.14	Référence de la classe <code>ResourceAllocator</code>	58
4.14.1	Description détaillée	59
4.14.2	Documentation des fonctions membres	59
4.14.2.1	<code>allocateFont(map< string, sf::Font * > &m, const string index, const string path)</code>	59
4.14.2.2	<code>allocateMusic(map< string, sf::Music * > &m, const string index, const string path, bool isLoop=false)</code>	59
4.14.2.3	<code>allocateTexture(map< string, sf::Texture * > &m, const string index, const string path)</code>	59
4.14.2.4	<code>allocateTileset(map< string, Tileset * > &m, const string index, Tileset *t)</code>	60
4.14.2.5	<code>allocateTileSystem(map< string, TileSystem * > &m, const string index, Tileset *t)</code>	60
4.15	Référence de la classe <code>Tile</code>	60
4.15.1	Description détaillée	62
4.15.2	Documentation des constructeurs et destructeur	62
4.15.2.1	<code>Tile(sf::Sprite &s, TileType type=TileType::DEFAULT)</code>	62
4.15.2.2	<code>Tile(TileType type=TileType::DEFAULT)</code>	62
4.15.2.3	<code>~Tile()</code>	63
4.15.3	Documentation des fonctions membres	63
4.15.3.1	<code>addSprite(sf::Sprite &s)</code>	63
4.15.3.2	<code>getSprite(float x, float y)</code>	63
4.15.3.3	<code>getSprite(unsigned int index, float x, float y)</code>	63
4.15.3.4	<code>getSpriteCount()</code>	64
4.15.3.5	<code>getType()</code>	64
4.15.4	Documentation des données membres	64
4.15.4.1	<code>m_sprites</code>	64
4.15.4.2	<code>m_type</code>	64
4.16	Référence de la classe <code>Tilemap</code>	65
4.16.1	Description détaillée	67

4.16.2	Documentation des constructeurs et destructeur	67
4.16.2.1	Tilemap(TileSystem *tilesys, int width, int height)	67
4.16.2.2	~Tilemap()	68
4.16.3	Documentation des fonctions membres	68
4.16.3.1	attemptDestruction(unsigned int i, unsigned int j)	68
4.16.3.2	draw(GameWindow *)	68
4.16.3.3	getHeight()	68
4.16.3.4	getMetadata(unsigned int i, unsigned int j)	68
4.16.3.5	getSprite(int i, int j)	68
4.16.3.6	getSprite(unsigned int index, int i, int j)	69
4.16.3.7	getTile(unsigned int i, unsigned int j)	69
4.16.3.8	getTileSystem()	69
4.16.3.9	getWidth()	69
4.16.3.10	setMap(const vector< vector< unsigned int >> &map)	69
4.16.3.11	setTileIndex(unsigned int index, unsigned int i, unsigned int j, int metadata=-1)	70
4.16.3.12	toTileCoord(sf : :Vector2f c)	70
4.16.3.13	updateState(sf : :Time &elapsed)	70
4.16.4	Documentation des données membres	70
4.16.4.1	m_height	70
4.16.4.2	m_map	70
4.16.4.3	m_metadata	70
4.16.4.4	m_watcher	71
4.16.4.5	m_width	71
4.17	Référence de la classe Tileset	71
4.17.1	Description détaillée	72
4.17.2	Documentation des constructeurs et destructeur	72
4.17.2.1	Tileset(const sf : :Texture *tex, int rows, int cols, int width, int height, int display← Width, int displayHeight)	72
4.17.3	Documentation des fonctions membres	73
4.17.3.1	createTile(int i, int j, TileType ty, bool c, bool b)	73
4.17.3.2	createTile(vector< int > i, vector< int > j, TileType ty, bool c, bool b)	73

4.17.3.3	<code>createTile(int i, int j, TileType ty=TileType : :DEFAULT, bool c=false, bool b=false)</code>	73
4.17.3.4	<code>createTile(vector< int > i, vector< int > j, TileType ty=TileType : :DEFAULT, bool c=false, bool b=false)</code>	73
4.17.3.5	<code>getCols()</code>	73
4.17.3.6	<code>getDisplayHeight()</code>	74
4.17.3.7	<code>getDisplayWidth()</code>	74
4.17.3.8	<code>getHeight()</code>	74
4.17.3.9	<code>getRows()</code>	74
4.17.3.10	<code>getTexture()</code>	74
4.17.3.11	<code>getWidth()</code>	75
4.17.4	Documentation des données membres	75
4.17.4.1	<code>m_cols</code>	75
4.17.4.2	<code>m_displayHeight</code>	75
4.17.4.3	<code>m_displayWidth</code>	75
4.17.4.4	<code>m_height</code>	75
4.17.4.5	<code>m_rows</code>	75
4.17.4.6	<code>m_texture</code>	75
4.17.4.7	<code>m_width</code>	75
4.18	Référence de la classe <code>TileSystem</code>	76
4.18.1	Description détaillée	77
4.18.2	Documentation des constructeurs et destructeur	77
4.18.2.1	<code>TileSystem(Tileset *ts)</code>	77
4.18.2.2	<code>~TileSystem()</code>	77
4.18.3	Documentation des fonctions membres	77
4.18.3.1	<code>getSize()</code>	77
4.18.3.2	<code>getTile(unsigned int index)</code>	78
4.18.3.3	<code>getTs()</code>	78
4.18.3.4	<code>registerTile(unsigned int index, int i, int j, TileType ty, bool c, bool b)</code>	78
4.18.3.5	<code>registerTile(unsigned int index, vector< int > i, vector< int > j, TileType ty, bool c, bool b)</code>	78
4.18.3.6	<code>registerTile(unsigned int index, int i, int j, TileType ty=TileType : :DEFAULT, bool c=NULL, bool b=NULL)</code>	78
4.18.3.7	<code>registerTile(unsigned int index, vector< int > i, vector< int > j, TileType ty=TileType : :DEFAULT, bool c=NULL, bool b=NULL)</code>	78
4.18.4	Documentation des données membres	79
4.18.4.1	<code>m_tilesList</code>	79
4.18.4.2	<code>m_ts</code>	79

5	Documentation des fichiers	81
5.1	Référence du fichier src/Bomb.cpp	81
5.2	Référence du fichier src/Bomb.h	81
5.3	Référence du fichier src/Button.cpp	83
5.4	Référence du fichier src/Button.h	84
5.5	Référence du fichier src/Controller.cpp	85
5.6	Référence du fichier src/Controller.h	86
5.7	Référence du fichier src/Entity.cpp	88
5.8	Référence du fichier src/Entity.h	88
5.9	Référence du fichier src/Game.cpp	90
5.10	Référence du fichier src/Game.h	90
5.11	Référence du fichier src/GameWindow.cpp	92
5.12	Référence du fichier src/GameWindow.h	92
5.13	Référence du fichier src/Globals.h	93
5.13.1	Documentation des macros	94
5.13.1.1	ABS	94
5.13.1.2	DEBUG_MODE	94
5.13.1.3	DEFAULT_TILE_SIZE	94
5.13.1.4	FONT_ERROR	94
5.13.1.5	G_AUTHORS	94
5.13.1.6	G_VERSION	95
5.13.1.7	GENERIC_ERROR	95
5.13.1.8	MUSIC_ERROR	95
5.13.1.9	SGN	95
5.13.1.10	SPEED_FACTOR	95
5.13.1.11	TEXTURE_ERROR	95
5.13.1.12	TILESET_ERROR	95
5.13.2	Documentation du type de l'énumération	95
5.13.2.1	Orientation	95
5.13.2.2	TileType	95

5.14	Référence du fichier src/IDrawable.cpp	96
5.15	Référence du fichier src/IDrawable.h	96
5.16	Référence du fichier src/IHandlable.h	98
5.17	Référence du fichier src/LivingEntity.cpp	99
5.18	Référence du fichier src/LivingEntity.h	99
5.19	Référence du fichier src/main.cpp	101
5.19.1	Documentation des fonctions	101
5.19.1.1	displayHelp(const char *name)	101
5.19.1.2	main(int argc, char *argv[])	102
5.20	Référence du fichier src/MapTile.cpp	102
5.21	Référence du fichier src/MapTile.h	102
5.22	Référence du fichier src/Menu.cpp	104
5.23	Référence du fichier src/Menu.h	104
5.24	Référence du fichier src/Player.cpp	105
5.25	Référence du fichier src/Player.h	106
5.26	Référence du fichier src/ResourceAllocator.cpp	107
5.27	Référence du fichier src/ResourceAllocator.h	107
5.28	Référence du fichier src/Tile.cpp	109
5.29	Référence du fichier src/Tile.h	109
5.30	Référence du fichier src/Tilemap.cpp	110
5.31	Référence du fichier src/Tilemap.h	111
5.32	Référence du fichier src/Tileset.cpp	113
5.33	Référence du fichier src/Tileset.h	113
5.34	Référence du fichier src/TileSystem.cpp	115
5.35	Référence du fichier src/TileSystem.h	115
	Index	117

Chapitre 1

Index hiérarchique

1.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

Controller	15
Game	25
Menu	47
IDrawable	35
Button	12
Entity	20
Bomb	7
LivingEntity	39
Player	53
Tilemap	65
IHandlable	37
Button	12
GameWindow	31
Player	53
IntRect	
Button	12
RenderWindow	
GameWindow	31
ResourceAllocator	58
streambuf	
nStream	52
Tile	60
MapTile	44
Tileset	71
TileSystem	76

Chapitre 2

Index des classes

2.1 Liste des classes

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

Bomb	Une bombe est l'élément central du jeu : elle explose les pierres et tue les joueurs	7
Button	Gestion de boutons cliquables	12
Controller	Classe abstraite des contrôleurs de jeu	15
Entity	Base abstraite des entités du jeu	20
Game	Contrôleur de partie	25
GameWindow	Vue principale du jeu	31
IDrawable	Interface implémentée par chaque classe qui peut être affichée	35
IHandlable	Interface implémentée par chaque classe qui a des évènements individuels à gérer	37
LivingEntity	Une entité vivante peut se mouvoir, et possède une orientation	39
MapTile	Stocke les données relatives à un Tile de Tilemap	44
Menu	Contrôleur qui gère le menu principal	47
nStream	Une structure qui décrit un stream buffer nul (équivalent en C++ du /dev/null sur les systèmes *nix)	52
Player	Un joueur est une entité vivante controlable	53
ResourceAllocator	Classe statique chargée de fournir des fonctions d'allocation de ressource	58
Tile	Un tile est une brique élémentaire de tout bon jeu 2D de base	60
Tilemap	Une carte est une grille de tuiles qui peut être dessinée	65
Tileset	Un set de tuiles stocke des informations brutes à propos d'une texture de tuiles	71
TileSystem	Un système de tuiles lie les données brutes d'un Tileset avec tous les Tile extraits de ce Tileset	76

Chapitre 3

Index des fichiers

3.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

src/Bomb.cpp	81
src/Bomb.h	81
src/Button.cpp	83
src/Button.h	84
src/Controller.cpp	85
src/Controller.h	86
src/Entity.cpp	88
src/Entity.h	88
src/Game.cpp	90
src/Game.h	90
src/GameWindow.cpp	92
src/GameWindow.h	92
src/Globals.h	93
src/IDrawable.cpp	96
src/IDrawable.h	96
src/IHandlable.h	98
src/LivingEntity.cpp	99
src/LivingEntity.h	99
src/main.cpp	101
src/MapTile.cpp	102
src/MapTile.h	102
src/Menu.cpp	104
src/Menu.h	104
src/Player.cpp	105
src/Player.h	106
src/ResourceAllocator.cpp	107
src/ResourceAllocator.h	107
src/Tile.cpp	109
src/Tile.h	109
src/Tilemap.cpp	110
src/Tilemap.h	111
src/Tilesset.cpp	113
src/Tilesset.h	113
src/TileSystem.cpp	115
src/TileSystem.h	115

Chapitre 4

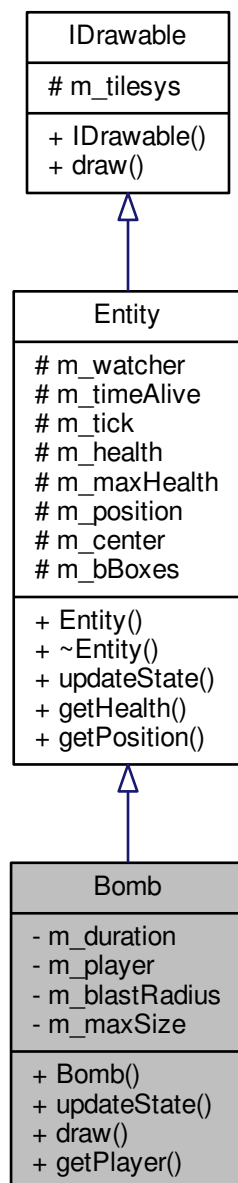
Documentation des classes

4.1 Référence de la classe Bomb

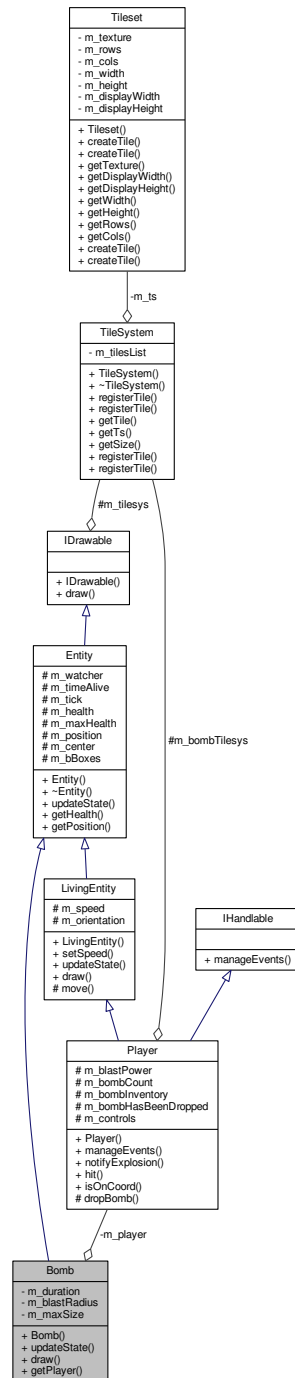
Une bombe est l'élément central du jeu : elle explose les pierres et tue les joueurs.

```
#include <Bomb.h>
```

Graphe d'héritage de Bomb :



Grphe de collaboration de Bomb :



Fonctions membres publiques

- **Bomb** (**TileSystem** *tilesys, sf : :Vector2f center, sf : :Vector2f position, int health, **Player** *player, sf : :Time duration, unsigned int blastRadius)
crée une bombe
- void **updateState** (**Controller** *controller, sf : :Time &elapsed, **Tilemap** *world)
met à jour l'état de l'entité en fonction du temps écoulé
- void **draw** (**GameWindow** *window)

- *contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée*
- `Player * getPlayer ()`
accesseur de m_player

Attributs privés

- `sf : :Time m_duration`
temps que met la bombe pour perdre 1Pdv
- `Player * m_player`
le joueur à l'origine de la bombe
- `unsigned int m_blastRadius`
le rayon de la bombe
- `unsigned int m_maxSize [4]`
décrit la taille maximale de chaque rayon selon l'environnement

Membres hérités additionnels

4.1.1 Description détaillée

Une bombe est l'élément central du jeu : elle explose les pierres et tue les joueurs.

4.1.2 Documentation des constructeurs et destructeur

- 4.1.2.1 `Bomb : :Bomb (TileSystem * tilesys, sf : :Vector2f center, sf : :Vector2f position, int health, Player * player, sf : :Time duration, unsigned int blastRadius)`

crée une bombe

Paramètres

<i>player</i>	joueur poseur de bombe
<i>duration</i>	temps mis par la bombe pour exploser

Voir également

`Entity : :Entity(TileSystem *tilesys, map<Orientation, sf : :IntRect> bBoxes, sf : :Vector2f center, sf : :Vector2f position, int health)`

4.1.3 Documentation des fonctions membres

- 4.1.3.1 `void Bomb : :draw (GameWindow * window) [virtual]`

contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée

Paramètres

<i>window</i>	fenêtre sur laquelle on dessine
---------------	---------------------------------

Implémente [IDrawable](#).

4.1.3.2 `Player * Bomb::getPlayer ()`

accesseur de `m_player`

Renvoie

`m_player`

4.1.3.3 `void Bomb::updateState (Controller * controller, sf::Time & elapsed, Tilemap * world) [virtual]`

met à jour l'état de l'entité en fonction du temps écoulé

Paramètres

<i>controller</i>	contrôleur à notifier si besoin
<i>elapsed</i>	temps écoulé
<i>world</i>	le monde dans lequel évolue l'entité

Implémente [Entity](#).

4.1.4 Documentation des données membres

4.1.4.1 `unsigned int Bomb::m_blastRadius [private]`

le rayon de la bombe

4.1.4.2 `sf::Time Bomb::m_duration [private]`

temps que met la bombe pour perdre 1Pdv

4.1.4.3 `unsigned int Bomb::m_maxSize[4] [private]`

décrit la taille maximale de chaque rayon selon l'environnement

4.1.4.4 `Player* Bomb::m_player [private]`

le joueur à l'origine de la bombe

La documentation de cette classe a été générée à partir des fichiers suivants :

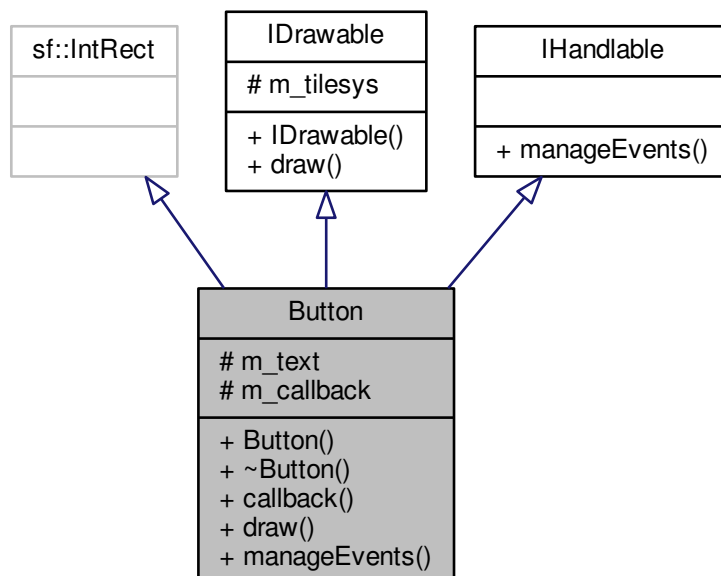
- [src/Bomb.h](#)
- [src/Bomb.cpp](#)

4.2 Référence de la classe Button

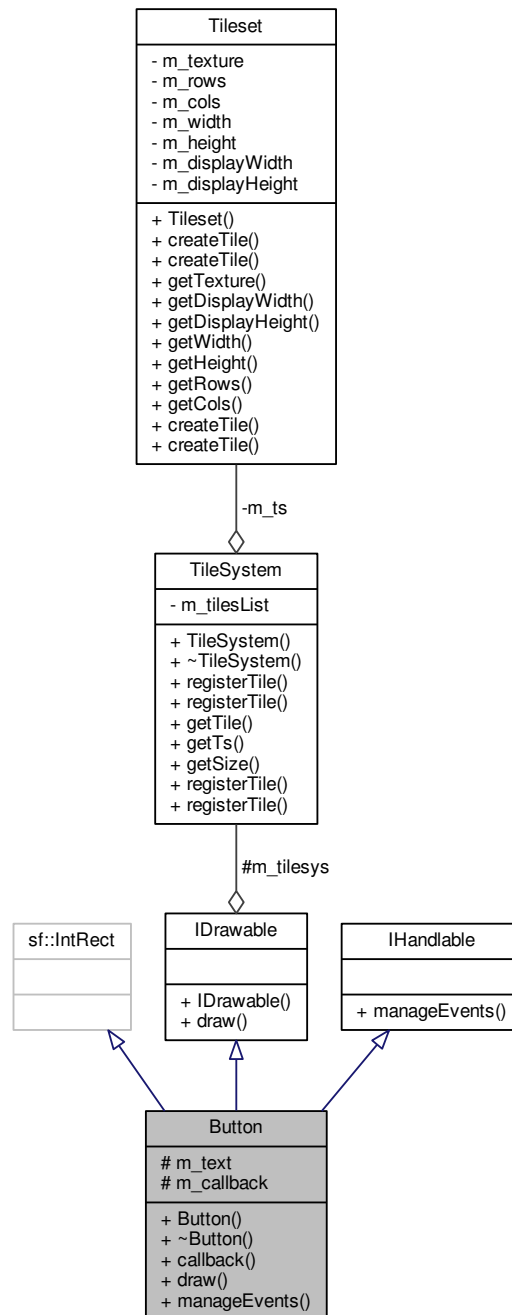
gestion de boutons cliquables

```
#include <Button.h>
```

Graphe d'héritage de Button :



Graphe de collaboration de Button :



Fonctions membres publiques

- **Button** (**TileSystem** *tilesys, int x, int y, string text, sf : :Font *font, void(*callback)(void *))
crée un bouton avec les paramètres renseignés
- virtual **~Button** ()
- void **callback** (void *args)
combiné d'appel pour la fonction de callback
- void **draw** (**GameWindow** *)

- *contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée*
- `void manageEvents (sf : :Event &event, void *args=NULL)`
contrat de l'interface : une classe fille devra implémenter cette méthode en fonction des évènements qui peuvent lui être liés

Attributs protégés

- `sf : :Text m_text`
texte affiché par le bouton
- `void(* m_callback)(void *)`
fonction de callback (à déclencher en cas de clic)

4.2.1 Description détaillée

gestion de boutons cliquables

4.2.2 Documentation des constructeurs et destructeur

4.2.2.1 `Button : :Button (TileSystem * tilesys, int x, int y, string text, sf : :Font * font, void(*) (void *) callback)`

crée un bouton avec les paramètres renseignés

Paramètres

<i>x</i>	abscisse du bouton
<i>y</i>	ordonnée du bouton
<i>tilesys</i>	TileSystem utilisé par le bouton
<i>text</i>	légende du bouton
<i>font</i>	fonte de caractère de la légende
<i>callback</i>	action à entreprendre en cas de clic du bouton

4.2.2.2 `Button : :~Button () [virtual]`

4.2.3 Documentation des fonctions membres

4.2.3.1 `void Button : :callback (void * args)`

combiné d'appel pour la fonction de callback

Paramètres

<i>args</i>	argument générique de la fonction de callback (utiliser une structure si la fonction de callback a besoin de plusieurs paramètres)
-------------	--

4.2.3.2 void Button::draw (GameWindow * window) [virtual]

contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée

Paramètres

<i>window</i>	fenêtre sur laquelle on dessine
---------------	---------------------------------

Implémente [IDrawable](#).

4.2.3.3 void Button::manageEvents (sf::Event & event, void * args=NULL) [virtual]

contrat de l'interface : une classe fille devra implémenter cette méthode en fonction des évènements qui peuvent lui être liés

Paramètres

<i>event</i>	évènement utilisateur
<i>args</i>	éventuels arguments dont on a besoin

Implémente [IHandlable](#).

4.2.4 Documentation des données membres

4.2.4.1 void(* Button::m_callback)(void *) [protected]

fonction de callback (à déclencher en cas de clic)

4.2.4.2 sf::Text Button::m_text [protected]

texte affiché par le bouton

La documentation de cette classe a été générée à partir des fichiers suivants :

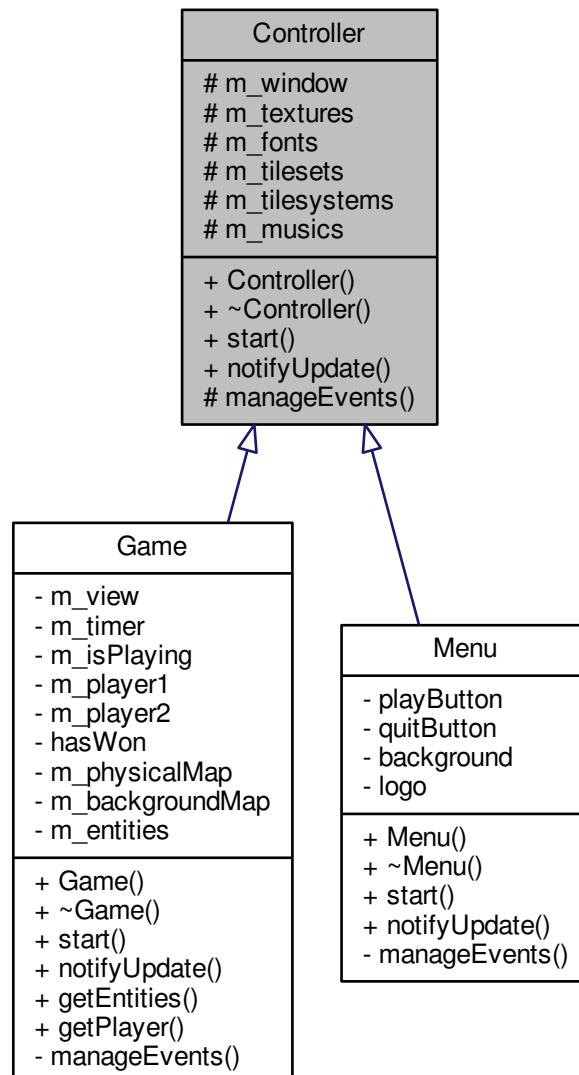
- [src/Button.h](#)
- [src/Button.cpp](#)

4.3 Référence de la classe Controller

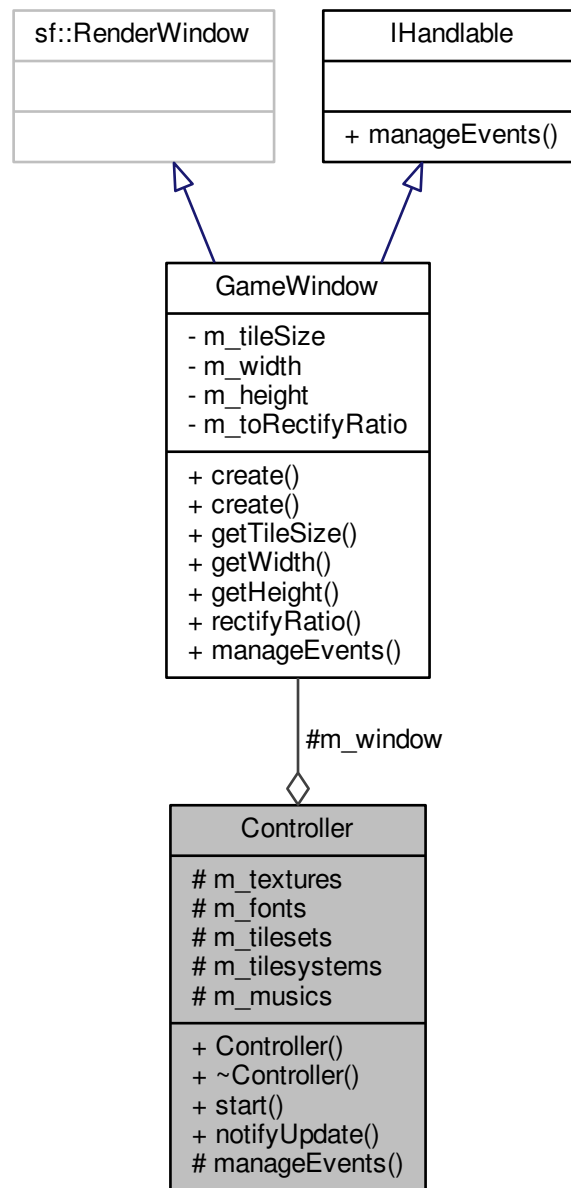
classe abstraite des contrôleurs de jeu

```
#include <Controller.h>
```

Graphe d'héritage de Controller :



Graphe de collaboration de Controller :



Fonctions membres publiques

- **Controller** (**GameWindow** *window)
crée un contrôleur
- virtual **~Controller** ()
met fin au contrôleur
- virtual void **start** ()=0
démarre le contrôleur
- virtual void **notifyUpdate** ()=0
indique au contrôleur qu'un changement d'état d'un de ses composants nécessite de rafraîchir d'autres composants

Fonctions membres protégées

- virtual void [manageEvents](#) ()=0
fonction technique contenant la routine de gestion des évènements utilisateurs

Attributs protégés

- [GameWindow](#) * [m_window](#)
la vue que le contrôleur gère
- map< string, sf::Texture * > [m_textures](#)
les textures que le contrôleur contient, référencées par un nom
- map< string, sf::Font * > [m_fonts](#)
les fontes que le contrôleur contient, référencées par un nom
- map< string, [Tileset](#) * > [m_tilesets](#)
les [Tileset](#) que le contrôleur contient, référencées par un nom
- map< string, [TileSystem](#) * > [m_tilesystems](#)
les [TileSystem](#) que le contrôleur contient, référencées par un nom
- map< string, sf::Music * > [m_musics](#)
les musiques que le contrôleur contient, référencées par un nom

4.3.1 Description détaillée

classe abstraite des contrôleurs de jeu

Un contrôleur est la partie décisionnelle du modèle MVC : il récupère les données du modèle, gère les évènements utilisateur, et programme la mise à jour de la vue.

4.3.2 Documentation des constructeurs et destructeur

4.3.2.1 Controller : :Controller ([GameWindow](#) * *window*)

crée un contrôleur

Habituellement, la construction d'un contrôleur est aussi la phase d'initialisation des ressources, au moyen des méthodes de la classe statique [ResourceAllocator](#).

Paramètres

<i>window</i>	la vue que le contrôleur gère
---------------	-------------------------------

4.3.2.2 Controller : :~Controller () [virtual]

met fin au contrôleur

Il faut libérer les ressources du contrôleur à ce moment là. Attention à n'en oublier aucune pour éviter les fuites de mémoire.

4.3.3 Documentation des fonctions membres

4.3.3.1 `virtual void Controller::manageEvents () [protected],[pure virtual]`

fonction technique contenant la routine de gestion des évènements utilisateurs

Implémenté dans [Game](#), et [Menu](#).

4.3.3.2 `virtual void Controller::notifyUpdate () [pure virtual]`

indique au contrôleur qu'un changement d'état d'un de ses composants nécessite de rafraîchir d'autres composants

Implémenté dans [Game](#), et [Menu](#).

4.3.3.3 `virtual void Controller::start () [pure virtual]`

démarre le contrôleur

Cette méthode doit contenir la boucle d'évènements. Ainsi, cette méthode ne se termine que quand le moment est venu de cesser ses activités (par exemple lorsque l'on ferme la vue qu'il gère, ou lorsque la partie se termine...).

Implémenté dans [Game](#), et [Menu](#).

4.3.4 Documentation des données membres

4.3.4.1 `map<string, sf::Font*> Controller::m_fonts [protected]`

les fontes que le contrôleur contient, référencées par un nom

4.3.4.2 `map<string, sf::Music*> Controller::m_musics [protected]`

les musiques que le contrôleur contient, référencées par un nom

4.3.4.3 `map<string, sf::Texture*> Controller::m_textures [protected]`

les textures que le contrôleur contient, référencées par un nom

4.3.4.4 `map<string, Tileset*> Controller::m_tilesets [protected]`

les [Tileset](#) que le contrôleur contient, référencées par un nom

4.3.4.5 `map<string, TileSystem*> Controller::m_tilesystems [protected]`

les [TileSystem](#) que le contrôleur contient, référencées par un nom

4.3.4.6 `GameWindow* Controller : :m_window` `[protected]`

la vue que le contrôleur gère

La documentation de cette classe a été générée à partir des fichiers suivants :

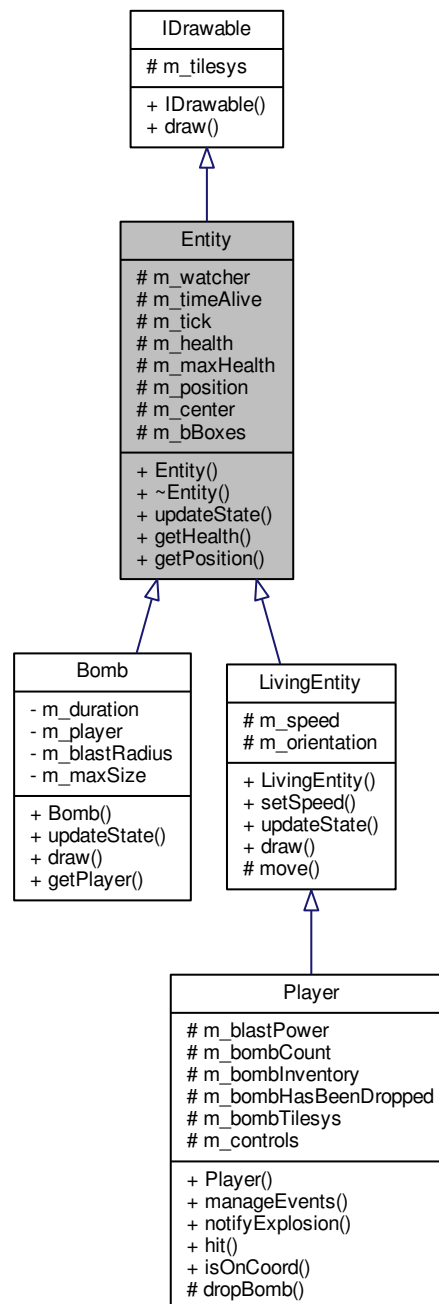
- [src/Controller.h](#)
- [src/Controller.cpp](#)

4.4 Référence de la classe Entity

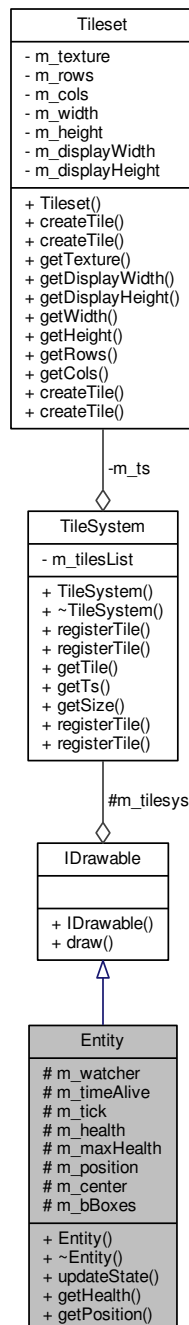
Base abstraite des entités du jeu.

```
#include <Entity.h>
```

Graphe d'héritage de Entity :



Graphe de collaboration de Entity :



Fonctions membres publiques

- **Entity** (**TileSystem** *tilesys, map< **Orientation**, sf : :IntRect > bBoxes, sf : :Vector2f center, sf : :Vector2f position, int health)
crée une entité
- virtual **~Entity** ()
- virtual void **updateState** (**Controller** *controller, sf : :Time &elapsed, **Tilemap** *world)=0
met à jour l'état de l'entité en fonction du temps écoulé

- int `getHealth` ()
accesseur de `m_health`
- sf : :Vector2f `getPosition` ()
accesseur de `m_position`

Attributs protégés

- sf : :Clock `m_watcher`
cette horloge permet de gérer la mise à jour du tick
- sf : :Clock `m_timeAlive`
temps depuis lequel l'entité est spawnée
- int `m_tick`
valeur particulière qui permet de gérer un état générique de l'entité (son animation par exemple)
- int `m_health`
la santé d'une entité peut avoir de multiples utilités : typiquement une vie ≤ 0 signifie que l'entité est morte, et nécessite un traitement particulier
- int `m_maxHealth`
la santé maximale d'un tick
- sf : :Vector2f `m_position`
position de l'entité
- sf : :Vector2f `m_center`
centre d'affichage de l'entité par rapport au coin supérieur gauche du `Tile`
- map< `Orientation`, sf : :IntRect > `m_bBoxes`
boite de collision de chaque orientation (il n'y en a qu'une si l'entité n'est pas vivante)

4.4.1 Description détaillée

Base abstraite des entités du jeu.

Une entité est un `IDrawable` potentiellement évolutif : il s'affiche sur la `TileMap` et peut se mouvoir, mourir ou agir dessus.

4.4.2 Documentation des constructeurs et destructeur

4.4.2.1 `Entity : :Entity (TileSystem * tilesys, map< Orientation, sf : :IntRect > bBoxes, sf : :Vector2f center, sf : :Vector2f position, int health)`

crée une entité

Paramètres

<i>tilesys</i>	<code>TileSystem</code> de l'entité
<i>bBoxes</i>	boites de collision
<i>center</i>	le centre d'affichage de l'entité
<i>position</i>	position initiale de l'entité
<i>health</i>	santé maximale (et initiale) de l'entité

4.4.2.2 `Entity : :~Entity () [virtual]`

4.4.3 Documentation des fonctions membres

4.4.3.1 `int Entity::getHealth ()`

accesseur de `m_health`

Renvoie

`m_health`

4.4.3.2 `sf::Vector2f Entity::getPosition ()`

accesseur de `m_position`

Renvoie

`m_position`

4.4.3.3 `virtual void Entity::updateState (Controller * controller, sf::Time & elapsed, Tilemap * world)` [pure virtual]

met à jour l'état de l'entité en fonction du temps écoulé

Paramètres

<i>controller</i>	contrôleur à notifier si besoin
<i>elapsed</i>	temps écoulé
<i>world</i>	le monde dans lequel évolue l'entité

Implémenté dans [LivingEntity](#), et [Bomb](#).

4.4.4 Documentation des données membres

4.4.4.1 `map<Orientation, sf::IntRect> Entity::m_bBoxes` [protected]

boite de collision de chaque orientation (il n'y en a qu'une si l'entité n'est pas vivante)

4.4.4.2 `sf::Vector2f Entity::m_center` [protected]

centre d'affichage de l'entité par rapport au coin supérieur gauche du [Tile](#)

4.4.4.3 `int Entity::m_health` [protected]

la santé d'une entité peut avoir de multiples utilités : typiquement une vie ≤ 0 signifie que l'entité est morte, et nécessite un traitement particulier

4.4.4.4 `int Entity::m_maxHealth` [protected]

la santé maximale d'un tick

4.4.4.5 `sf::Vector2f Entity::m_position` [protected]

position de l'entité

4.4.4.6 `int Entity::m_tick` [protected]

valeur particulière qui permet de gérer un état générique de l'entité (son animation par exemple)

4.4.4.7 `sf::Clock Entity::m_timeAlive` [protected]

temps depuis lequel l'entité est spawnée

4.4.4.8 `sf::Clock Entity::m_watcher` [protected]

cette horloge permet de gérer la mise à jour du tick

La documentation de cette classe a été générée à partir des fichiers suivants :

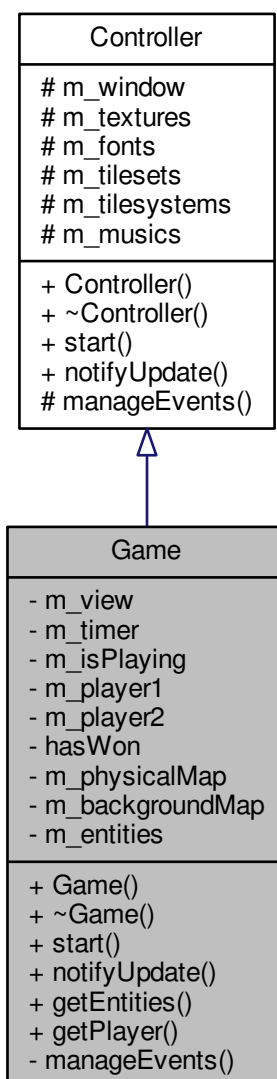
- [src/Entity.h](#)
- [src/Entity.cpp](#)

4.5 Référence de la classe Game

contrôleur de partie

```
#include <Game.h>
```

Graphe d'héritage de Game :



- void `notifyUpdate` ()
indique au controlleur qu'un changement d'état d'un de ses composants nécessite de rafraîchir d'autres composants
- vector< `Entity` * > * `getEntities` ()
accesseur de m_entities
- `Player` * `getPlayer` (int i)
récupère le joueur d'indice donné

Fonctions membres privées

- void `manageEvents` ()
appelé durant la boucle de jeu

Attributs privés

- sf : :View `m_view`
conteneur du contexte de jeu
- sf : :Clock `m_timer`
permet la gestion du temps écoulé entre deux itérations de la boucle de jeu
- bool `m_isPlaying`
permet de vérifier si la partie est en cours
- `Player` * `m_player1`
le joueur 1
- `Player` * `m_player2`
le joueur 2
- int `hasWon`
0 si personne n'est mort, 1 ou 2 en fonction de qui a gagné
- `Tilemap` * `m_physicalMap`
plateau de jeu
- `Tilemap` * `m_backgroundMap`
arrière plan de jeu
- vector< `Entity` * > `m_entities`
entités non joueur présentes sur le plateau

Membres hérités additionnels

4.5.1 Description détaillée

contrôleur de partie

4.5.2 Documentation des constructeurs et destructeur

4.5.2.1 `Game : :Game (GameWindow * window)`

crée une partie

Voir également

`Controller : :Controller(GameWindow *window)`

4.5.2.2 Game::~Game () [virtual]

met fin au contrôleur de partie

Voir également

[Controller : ~Controller\(\)](#)

4.5.3 Documentation des fonctions membres

4.5.3.1 vector< Entity * > * Game : getEntities ()

accesseur de m_entities

Renvoie

m_entities

4.5.3.2 Player * Game : getPlayer (int i)

récupère le joueur d'indice donné

Paramètres

<i>i</i>	indice du joueur
----------	------------------

Renvoie

joueur i

4.5.3.3 void Game : manageEvents () [private], [virtual]

appelé durant la boucle de jeu

La gestion des évènements du jeu comprend les entrées utilisateur, la demande de mise en pause, etc.

Implémente [Controller](#).

4.5.3.4 void Game : notifyUpdate () [virtual]

indique au controlleur qu'un changement d'état d'un de ses composants nécessite de rafraîchir d'autres composants

Implémente [Controller](#).

4.5.3.5 void Game::start () [virtual]

lance une partie et gère son cycle de vie

Voir également

void [Controller::start\(\)](#)

Implémente [Controller](#).

4.5.4 Documentation des données membres

4.5.4.1 int Game::hasWon [private]

0 si personne n'est mort, 1 ou 2 en fonction de qui a gagné

4.5.4.2 Tilemap* Game::m_backgroundMap [private]

arrière plan de jeu

4.5.4.3 vector<Entity*> Game::m_entities [private]

entités non joueur présentes sur le plateau

4.5.4.4 bool Game::m_isPlaying [private]

permet de vérifier si la partie est en cours

4.5.4.5 Tilemap* Game::m_physicalMap [private]

plateau de jeu

4.5.4.6 Player* Game::m_player1 [private]

le joueur 1

4.5.4.7 Player * Game::m_player2 [private]

le joueur 2

4.5.4.8 sf::Clock Game::m_timer [private]

permet la gestion du temps écoulé entre deux itérations de la boucle de jeu

4.5.4.9 `sf::View Game::m_view` [private]

conteneur du contexte de jeu

La documentation de cette classe a été générée à partir des fichiers suivants :

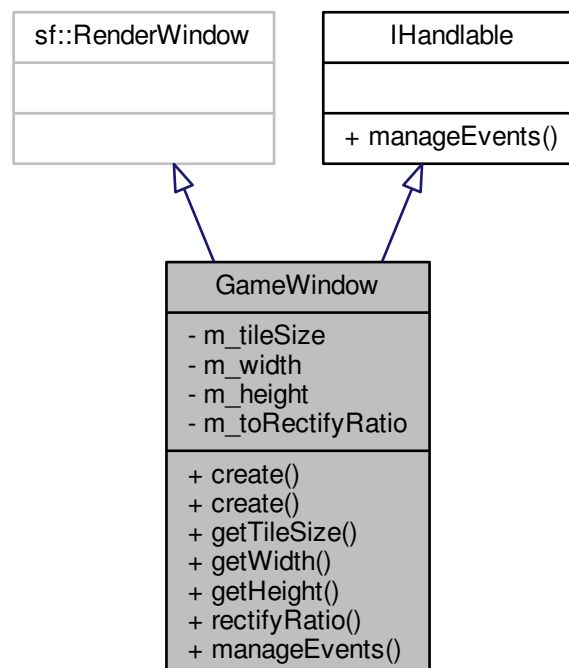
- [src/Game.h](#)
- [src/Game.cpp](#)

4.6 Référence de la classe `GameWindow`

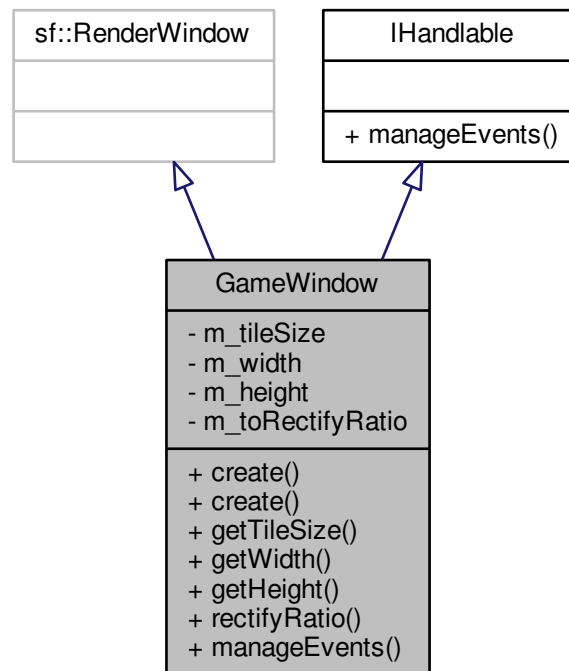
vue principale du jeu

```
#include <GameWindow.h>
```

Graphe d'héritage de `GameWindow` :



Graphe de collaboration de GameWindow :



Fonctions membres publiques

- void `create` (int width, int height)
crée une fenêtre avec la taille par défaut (DEFAULT_TILE_SIZE tel que défini dans Global.h)
- void `create` (int tileSize, int width, int height)
crée une fenêtre avec une taille de tile personnalisée
- int `getTileSize` ()
accesseur de m_tileSize
- int `getWidth` ()
accesseur de m_width
- int `getHeight` ()
accesseur de m_height
- void `rectifyRatio` ()
rectifie le ratio à m_width : m_height
- void `manageEvents` (sf : :Event &event, void *args=NULL)
contrat de l'interface : une classe fille devra implémenter cette méthode en fonction des évènements qui peuvent lui être liés

Attributs privés

- int `m_tileSize`
unité principale d'affichage, la taille régulière d'un tile carré classique
- int `m_width`
largeur en m_tileSize de la fenêtre
- int `m_height`
hauteur en m_tileSize de la fenêtre
- bool `m_toRectifyRatio`
booléen technique utilisé pour savoir si une rectification du ratio est nécessaire

4.6.1 Description détaillée

vue principale du jeu

Une vue est la partie affichage du modèle MVC. Elle présente à l'écran tout ce que l'utilisateur doit voir.

4.6.2 Documentation des fonctions membres

4.6.2.1 void GameWindow : :create (int *width*, int *height*)

crée une fenêtre avec la taille par défaut (DEFAULT_TILE_SIZE tel que défini dans Global.h)

Paramètres

<i>width</i>	largeur
<i>height</i>	hauteur

4.6.2.2 void GameWindow : :create (int *tileSize*, int *width*, int *height*)

crée une fenêtre avec une taille de tile personnalisée

Paramètres

<i>tileSize</i>	taille de tile personnalisée
<i>width</i>	largeur
<i>height</i>	hauteur

4.6.2.3 int GameWindow : :getHeight ()

accesseur de m_height

Renvoie

m_height

4.6.2.4 int GameWindow : :getTileSize ()

accesseur de m_tileSize

Renvoie

m_tileSize

4.6.2.5 `int GameWindow : :getWidth ()`

accesseur de `m_width`

Renvoie

`m_width`

4.6.2.6 `void GameWindow : :manageEvents (sf : :Event & event, void * args =NULL) [virtual]`

contrat de l'interface : une classe fille devra implémenter cette méthode en fonction des évènements qui peuvent lui être liés

Paramètres

<i>event</i>	évènement utilisateur
<i>args</i>	éventuels arguments dont on a besoin

Implémente [IHandlable](#).

4.6.2.7 `void GameWindow : :rectifyRatio ()`

rectifie le ratio à `m_width : m_height`

4.6.3 Documentation des données membres

4.6.3.1 `int GameWindow : :m_height [private]`

hauteur en `m_tileSize` de la fenêtre

4.6.3.2 `int GameWindow : :m_tileSize [private]`

unité principale d'affichage, la taille régulière d'un tile carré classique

4.6.3.3 `bool GameWindow : :m_toRectifyRatio [private]`

booléen technique utilisé pour savoir si une rectification du ratio est nécessaire

4.6.3.4 `int GameWindow : :m_width [private]`

largeur en `m_tileSize` de la fenêtre

La documentation de cette classe a été générée à partir des fichiers suivants :

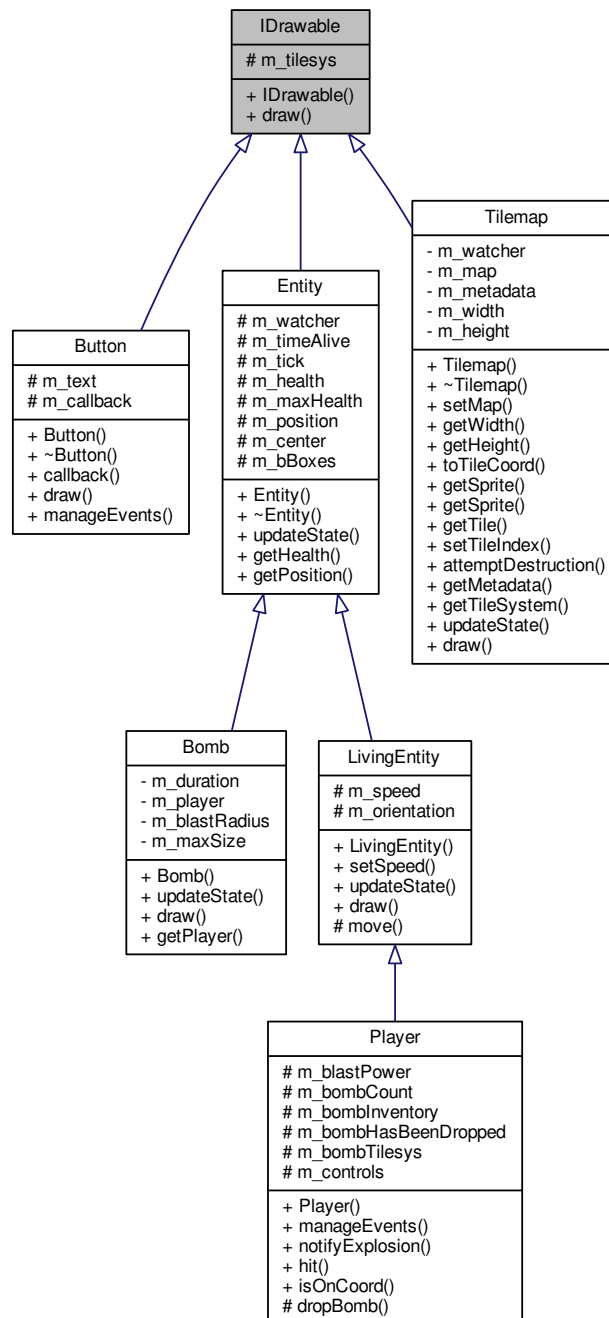
- [src/GameWindow.h](#)
- [src/GameWindow.cpp](#)

4.7 Référence de la classe IDrawable

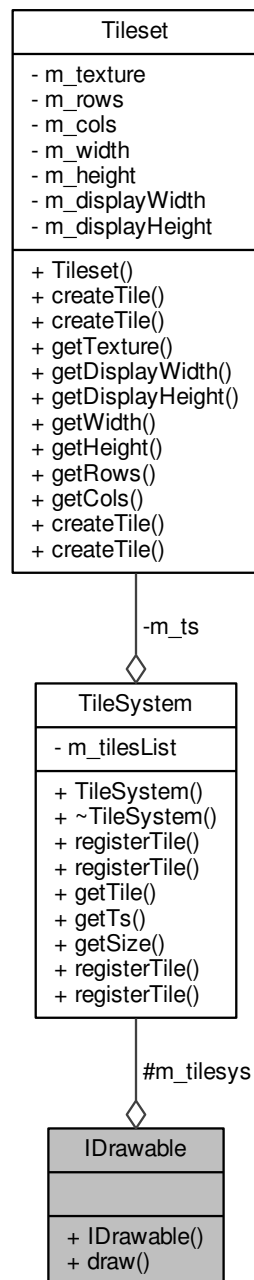
Interface implémentée par chaque classe qui peut être affichée.

```
#include <IDrawable.h>
```

Graphe d'héritage de IDrawable :



Graphe de collaboration de IDrawable :



Fonctions membres publiques

- **IDrawable** (**TileSystem** *tilesys)
constructeur à appeler lors de la construction des classes filles pour assigner une valeur au pointeur sur le [TileSystem](#) du [IDrawable](#)
- virtual void **draw** (**GameWindow** *window)=0
contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée

Attributs protégés

- [TileSystem](#) * [m_tilesys](#)
Un objet dessinable a toujours un [TileSystem](#).

4.7.1 Description détaillée

Interface implémentée par chaque classe qui peut être affichée.

4.7.2 Documentation des constructeurs et destructeur

4.7.2.1 IDrawable : :IDrawable ([TileSystem](#) * *tilesys*)

constructeur à appeler lors de la construction des classes filles pour assigner une valeur au pointeur sur le [TileSystem](#) du [IDrawable](#)

4.7.3 Documentation des fonctions membres

4.7.3.1 virtual void IDrawable : :draw ([GameWindow](#) * *window*) [pure virtual]

contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée

Paramètres

<i>window</i>	fenêtre sur laquelle on dessine
---------------	---------------------------------

Implémenté dans [Tilemap](#), [Button](#), [LivingEntity](#), et [Bomb](#).

4.7.4 Documentation des données membres

4.7.4.1 [TileSystem](#)* IDrawable : :m_tilesys [protected]

Un objet dessinable a toujours un [TileSystem](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

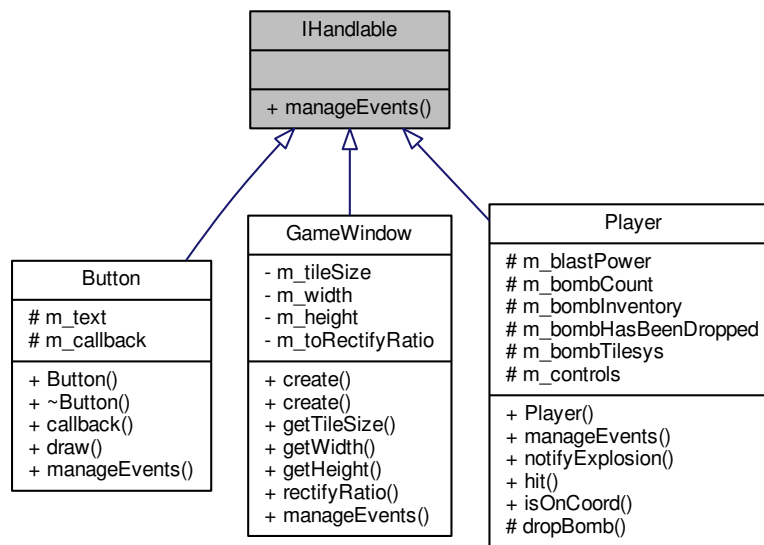
- [src/IDrawable.h](#)
- [src/IDrawable.cpp](#)

4.8 Référence de la classe IHandlable

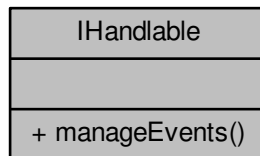
Interface implémentée par chaque classe qui a des événements individuels à gérer.

```
#include <IHandlable.h>
```

Graphe d'héritage de IHandlable :



Graphe de collaboration de IHandlable :



Fonctions membres publiques

- virtual void `manageEvents` (sf : `Event` &event, void *args=NULL)=0
contrat de l'interface : une classe fille devra implémenter cette méthode en fonction des événements qui peuvent lui être liés

4.8.1 Description détaillée

Interface implémentée par chaque classe qui a des événements individuels à gérer.

4.8.2 Documentation des fonctions membres

4.8.2.1 `virtual void IHandlable :manageEvents (sf :Event & event, void * args = NULL) [pure virtual]`

contrat de l'interface : une classe fille devra implémenter cette méthode en fonction des évènements qui peuvent lui être liés

Paramètres

<i>event</i>	évènement utilisateur
<i>args</i>	éventuels arguments dont on a besoin

Implémenté dans [GameWindow](#), [Player](#), et [Button](#).

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

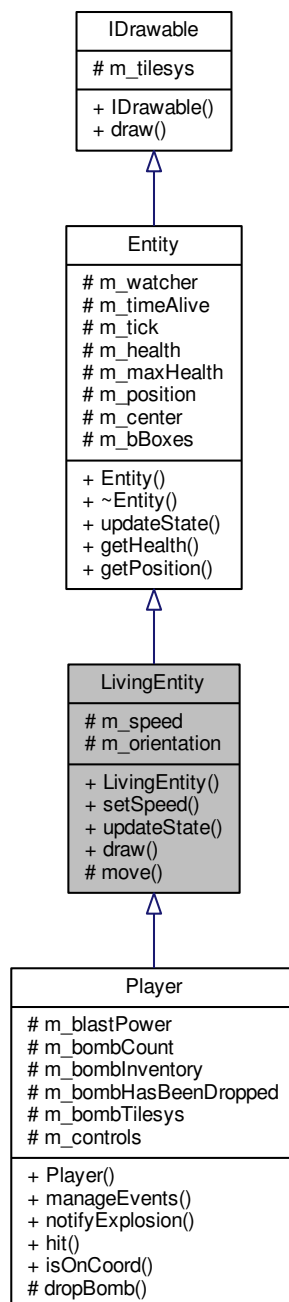
— [src/IHandlable.h](#)

4.9 Référence de la classe LivingEntity

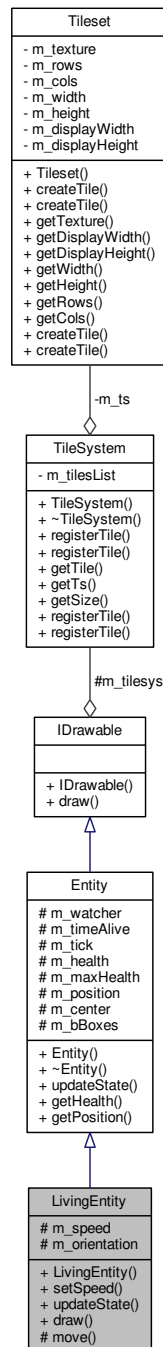
une entité vivante peut se mouvoir, et possède une orientation

```
#include <LivingEntity.h>
```

Graphe d'héritage de LivingEntity :



Graphe de collaboration de LivingEntity :



Fonctions membres publiques

- **LivingEntity** (**TileSystem** *tilesys, map< **Orientation**, sf::IntRect > bBoxes, sf::Vector2f center, sf::Vector2f position, int health)
crée une entité vivante
- void **setSpeed** (sf::Vector2f speed)
mutateur de m_speed
- void **updateState** (**Controller** *controller, sf::Time &elapsed, **Tilemap** *world)

- *met à jour l'état de l'entité en fonction du temps écoulé*
- void `draw` (`GameWindow *window`)
contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée

Fonctions membres protégées

- void `move` (`sf : :Vector2f potential`, `Tilemap *world`)
tente le déplacement de l'entité

Attributs protégés

- `sf : :Vector2f m_speed`
vitesse actuelle de l'entité
- `Orientation m_orientation`
orientation de l'entité

4.9.1 Description détaillée

une entité vivante peut se mouvoir, et possède une orientation

4.9.2 Documentation des constructeurs et destructeur

- 4.9.2.1 `LivingEntity : :LivingEntity (TileSystem * tilesys, map< Orientation, sf : :IntRect > bBoxes, sf : :Vector2f center, sf : :Vector2f position, int health)`

crée une entité vivante

Voir également

`Entity : :Entity(TileSystem *tilesys, map<Orientation, sf : :IntRect> bBoxes, sf : :Vector2f center, sf : :Vector2f position, int health)`

4.9.3 Documentation des fonctions membres

- 4.9.3.1 void `LivingEntity : :draw (GameWindow * window)` `[virtual]`

contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée

Paramètres

<code>window</code>	fenêtre sur laquelle on dessine
---------------------	---------------------------------

Implémente `IDrawable`.

- 4.9.3.2 void `LivingEntity : :move (sf : :Vector2f potential, Tilemap * world)` `[protected]`

tente le déplacement de l'entité

Paramètres

<i>potential</i>	le vecteur de déplacement qu'on espère réaliser
<i>world</i>	le monde dans lequel évolue l'entité (pour gérer les collisions)

4.9.3.3 void LivingEntity : :setSpeed (sf : :Vector2f *speed*)

mutateur de m_speed

Paramètres

<i>speed</i>	nouvelle valeur de m_speed
--------------	----------------------------

4.9.3.4 void LivingEntity : :updateState (Controller * *controller*, sf : :Time & *elapsed*, Tilemap * *world*) [virtual]

met à jour l'état de l'entité en fonction du temps écoulé

Paramètres

<i>controller</i>	contrôleur à notifier si besoin
<i>elapsed</i>	temps écoulé
<i>world</i>	le monde dans lequel évolue l'entité

Implémente [Entity](#).

4.9.4 Documentation des données membres

4.9.4.1 Orientation LivingEntity : :m_orientation [protected]

orientation de l'entité

4.9.4.2 sf : :Vector2f LivingEntity : :m_speed [protected]

vitesse actuelle de l'entité

La documentation de cette classe a été générée à partir des fichiers suivants :

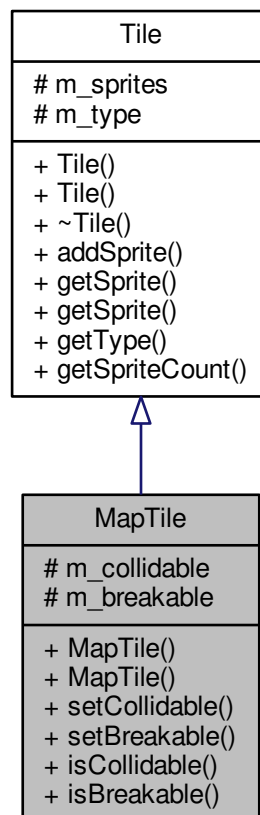
- [src/LivingEntity.h](#)
- [src/LivingEntity.cpp](#)

4.10 Référence de la classe MapTile

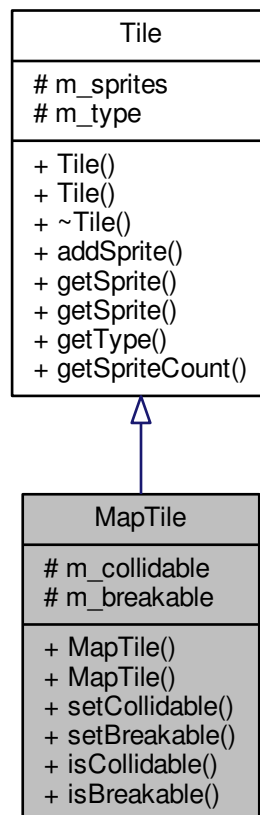
Stocke les données relatives à un [Tile](#) de [Tilemap](#).

```
#include <MapTile.h>
```

Graphe d'héritage de MapTile :



Graphe de collaboration de MapTile :



Fonctions membres publiques

- `MapTile (sf : :Sprite &s, TileType type=TileType : :DEFAULT)`
crée un [MapTile](#) avec un premier [sf : :Sprite](#)
- `MapTile (TileType type=TileType : :DEFAULT)`
crée un [MapTile](#) sans [sf : :Sprite](#)
- `void setCollidable (bool c)`
mutateur de [m_collidable](#)
- `void setBreakable (bool b)`
mutateur de [m_breakable](#)
- `bool isCollidable ()`
accesseur de [m_collidable](#)
- `bool isBreakable ()`
accesseur de [m_breakable](#)

Attributs protégés

- `bool m_collidable`
vrai si le [Tile](#) est physique
- `bool m_breakable`
vrai si le [Tile](#) est cassable

4.10.1 Description détaillée

Stocke les données relatives à un [Tile](#) de [Tilemap](#).

4.10.2 Documentation des constructeurs et destructeur

4.10.2.1 `MapTile : :MapTile (sf : :Sprite & s, TileType type = TileType::DEFAULT)`

crée un [MapTile](#) avec un premier `sf : :Sprite`

Paramètres

<code>s</code>	premier <code>sf : :Sprite</code> du MapTile
<code>type</code>	le type de tile

4.10.2.2 `MapTile : :MapTile (TileType type = TileType::DEFAULT)`

crée un [MapTile](#) sans `sf : :Sprite`

Paramètres

<code>type</code>	le type de tile
-------------------	-----------------

4.10.3 Documentation des fonctions membres

4.10.3.1 `bool MapTile : :isBreakable ()`

accesseur de `m_breakable`

Renvoie

`m_breakable`

4.10.3.2 `bool MapTile : :isCollidable ()`

accesseur de `m_collidable`

Renvoie

`m_collidable`

4.10.3.3 `void MapTile : :setBreakable (bool b)`

mutateur de `m_breakable`

Paramètres

<i>b</i>	nouvelle valeur de m_breakable
----------	--------------------------------

4.10.3.4 void MapTile : :setCollidable (bool *c*)

mutateur de m_collidable

Paramètres

<i>c</i>	nouvelle valeur de m_collidable
----------	---------------------------------

4.10.4 Documentation des données membres

4.10.4.1 bool MapTile : :m_breakable [protected]

vrai si le [Tile](#) est cassable

4.10.4.2 bool MapTile : :m_collidable [protected]

vrai si le [Tile](#) est physique

La documentation de cette classe a été générée à partir des fichiers suivants :

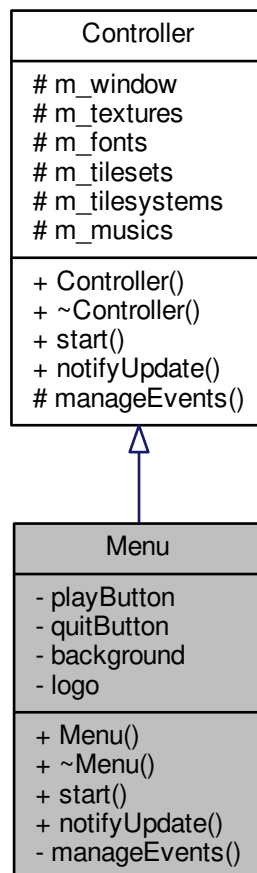
- [src/MapTile.h](#)
- [src/MapTile.cpp](#)

4.11 Référence de la classe Menu

contrôleur qui gère le menu principal

```
#include <Menu.h>
```

Graphe d'héritage de Menu :



- void [notifyUpdate](#) ()
indique au controleur qu'un changement d'état d'un de ses composants nécessite de rafraîchir d'autres composants

Fonctions membres privées

- void [manageEvents](#) ()
appelé durant la boucle principale

Attributs privés

- Button * [playButton](#)
bouton Jouer
- Button * [quitButton](#)
bouton quitter
- sf : :Sprite [background](#)
image d'arrière plan
- sf : :Sprite [logo](#)
logo du jeu

Membres hérités additionnels

4.11.1 Description détaillée

contrôleur qui gère le menu principal

4.11.2 Documentation des constructeurs et destructeur

4.11.2.1 Menu : :Menu ([GameWindow](#) * *window*)

crée un contrôleur de menu

Voir également

[Controller](#) : :[Controller](#)([GameWindow](#) **window*)

4.11.2.2 Menu : :~Menu () [virtual]

met fin au contrôleur du menu

Voir également

[Controller](#) : :~[Controller](#)()

4.11.3 Documentation des fonctions membres

4.11.3.1 void Menu : :manageEvents () [private], [virtual]

appelé durant la boucle principale

La gestion des évènements du menu comprend la vérification du clic.

Implémente [Controller](#).

4.11.3.2 `void Menu : :notifyUpdate () [virtual]`

indique au controleur qu'un changement d'état d'un de ses composants nécessite de rafraîchir d'autres composants

Implémente [Controller](#).

4.11.3.3 `void Menu : :start () [virtual]`

méthode principale qui gère la vie du menu

Voir également

`void Controller : :start()`

Implémente [Controller](#).

4.11.4 Documentation des données membres

4.11.4.1 `sf : :Sprite Menu : :background [private]`

image d'arrière plan

4.11.4.2 `sf : :Sprite Menu : :logo [private]`

logo du jeu

4.11.4.3 `Button* Menu : :playButton [private]`

bouton Jouer

4.11.4.4 `Button* Menu : :quitButton [private]`

bouton quitter

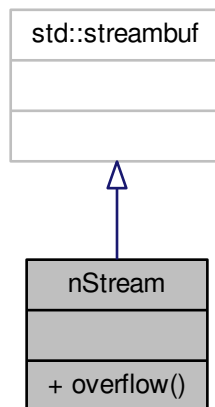
La documentation de cette classe a été générée à partir des fichiers suivants :

- [src/Menu.h](#)
- [src/Menu.cpp](#)

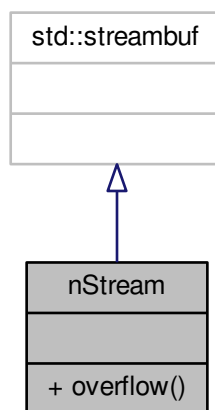
4.12 Référence de la structure nStream

une structure qui décrit un stream buffer nul (équivalent en C++ du /dev/null sur les systèmes *nix)

Graphe d'héritage de nStream :



Graphe de collaboration de nStream :



Fonctions membres publiques

- void [overflow](#) (char c)
fonction d'overflow, ne fait rien car on a un tampon nul

4.12.1 Description détaillée

une structure qui décrit un stream buffer nul (équivalent en C++ du /dev/null sur les systèmes *nix)

4.12.2 Documentation des fonctions membres

4.12.2.1 `void nStream::overflow (char c) [inline]`

fonction d'overflow, ne fait rien car on a un tampon nul

Paramètres

<code>c</code>	Regarde-moi dans les yeux. Regarde-moi. On s'en branle, c'est pas important !
----------------	---

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

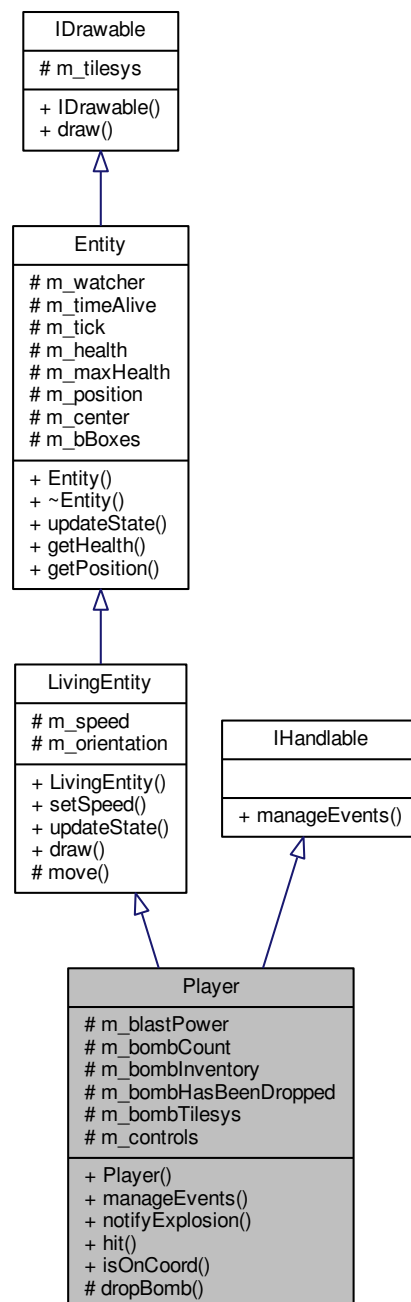
— [src/main.cpp](#)

4.13 Référence de la classe Player

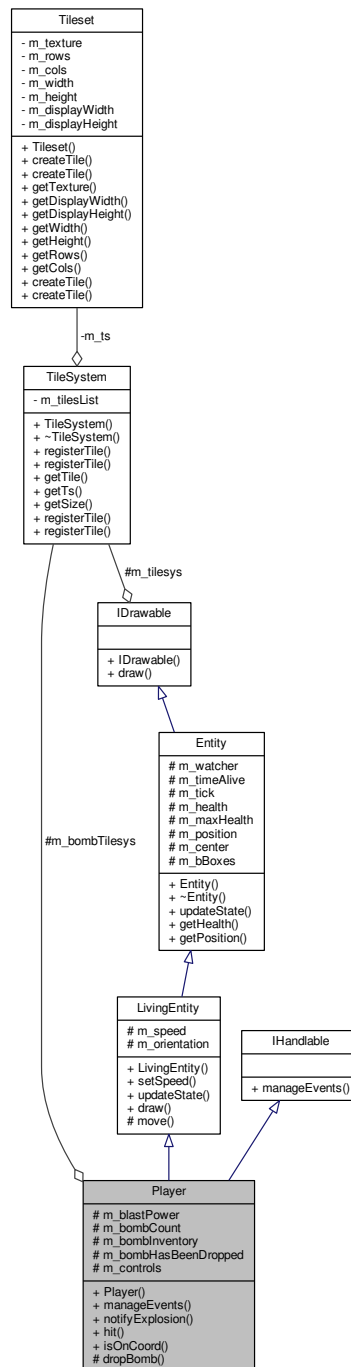
un joueur est une entité vivante controlable

```
#include <Player.h>
```

Graphe d'héritage de Player :



Graphe de collaboration de Player :



Fonctions membres publiques

- **Player** (**TileSystem** *tilesys, map< **Orientation**, sf : :IntRect > bBoxes, sf : :Vector2f center, sf : :Vector2f position, int health, vector< sf : :Keyboard : :Key > controls, **TileSystem** *bombTilesys)
crée un joueur
- void **manageEvents** (sf : :Event &event, void *args=NULL)
contrat de l'interface : une classe fille devra implémenter cette méthode en fonction des événements qui peuvent lui être liés

- void `notifyExplosion` ()
doit être appelé par la bombe d'un joueur quand elle explose, pour notifier ce dernier qu'elle a explosée
- void `hit` ()
fait prendre un point de dégât au joueur
- bool `isOnCoord` (unsigned int i, unsigned int j, `Tilemap` *world)
vérifie si le joueur touche la coordonnée donnée

Fonctions membres protégées

- void `dropBomb` (sf : :Vector2f pos, vector< `Entity` * > *list)
crée une bombe aux coordonnées indiquées, si possible

Attributs protégés

- unsigned int `m_blastPower`
La puissance des bombes que pose le joueur.
- unsigned int `m_bombCount`
Le nombre de bombes qu'un joueur peut poser simultanément.
- unsigned int `m_bombInventory`
Le nombre de bombes que le joueur peut encore poser.
- bool `m_bombHasBeenDropped`
boolean permettant d'éviter que plusieurs bombes soient posées par un même appui sur la touche de drop
- `TileSystem` * `m_bombTilesys`
le `TileSystem` des bombes que le joueur pose
- vector< sf : :Keyboard : :Key > `m_controls`
Les contrôles du joueur, dans l'ordre : Haut, Droite, Bas, Gauche, Poser bombe.

4.13.1 Description détaillée

un joueur est une entité vivante controlable

4.13.2 Documentation des constructeurs et destructeur

- 4.13.2.1 `Player` : :`Player` (`TileSystem` * `tilesys`, map< `Orientation`, sf : :IntRect > `bBoxes`, sf : :Vector2f `center`, sf : :Vector2f `position`, int `health`, vector< sf : :Keyboard : :Key > `controls`, `TileSystem` * `bombTilesys`)

crée un joueur

Voir également

`LivingEntity` : :`LivingEntity`(`TileSystem` *`tilesys`, map<`Orientation`, sf : :IntRect> `bBoxes`, sf : :Vector2f `center`, sf : :Vector2f `position`, int `health`)

4.13.3 Documentation des fonctions membres

- 4.13.3.1 void `Player` : :`dropBomb` (sf : :Vector2f `pos`, vector< `Entity` * > * `list`) [protected]

crée une bombe aux coordonnées indiquées, si possible

Paramètres

<i>pos</i>	position sur la map où larguer la bombe
<i>list</i>	registre d'entités sur lequel inscrire la bombe

4.13.3.2 void Player : :hit ()

fait prendre un point de dégat au joueur

4.13.3.3 bool Player : :isOnCoord (unsigned int *i*, unsigned int *j*, Tilemap * *world*)

vérifie si le joueur touche la coordonnée donnée

Paramètres

<i>i</i>	abscisse
<i>j</i>	ordonnée

Renvoie

vrai si le joueur touche

4.13.3.4 void Player : :manageEvents (sf : :Event & *event*, void * *args* = NULL) [virtual]

contrat de l'interface : une classe fille devra implémenter cette méthode en fonction des évènements qui peuvent lui être liés

Paramètres

<i>event</i>	évènement utilisateur
<i>args</i>	éventuels arguments dont on a besoin

Implémente [IHandlable](#).

4.13.3.5 void Player : :notifyExplosion ()

doit être appelé par la bombe d'un joueur quand elle explose, pour notifier ce dernier qu'elle a explosée

4.13.4 Documentation des données membres

4.13.4.1 unsigned int Player : :m_blastPower [protected]

La puissance des bombes que pose le joueur.

4.13.4.2 `unsigned int Player : :m_bombCount` `[protected]`

Le nombre de bombes qu'un joueur peut poser simultanément.

4.13.4.3 `bool Player : :m_bombHasBeenDropped` `[protected]`

boolean permettant d'éviter que plusieurs bombes soient posées par un même appui sur la touche de drop

4.13.4.4 `unsigned int Player : :m_bombInventory` `[protected]`

Le nombre de bombes que le joueur peut encore poser.

4.13.4.5 `TileSystem* Player : :m_bombTilesys` `[protected]`

le [TileSystem](#) des bombes que le joueur pose

4.13.4.6 `vector<sf : :Keyboard : :Key> Player : :m_controls` `[protected]`

Les contrôles du joueur, dans l'ordre : Haut, Droite, Bas, Gauche, Poser bombe.

La documentation de cette classe a été générée à partir des fichiers suivants :

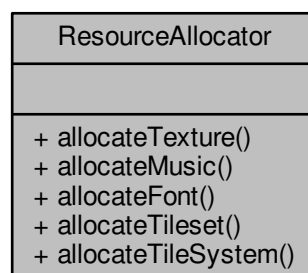
- [src/Player.h](#)
- [src/Player.cpp](#)

4.14 Référence de la classe ResourceAllocator

classe statique chargée de fournir des fonctions d'allocation de ressource

```
#include <ResourceAllocator.h>
```

Graphe de collaboration de ResourceAllocator :



Fonctions membres publiques statiques

- static void [allocateTexture](#) (map< string, sf : :Texture * > &m, const string index, const string path)
indexe une texture
- static void [allocateMusic](#) (map< string, sf : :Music * > &m, const string index, const string path, bool isLoop=false)
indexe une musique
- static void [allocateFont](#) (map< string, sf : :Font * > &m, const string index, const string path)
indexe une fonte
- static void [allocateTiledset](#) (map< string, [Tiledset](#) * > &m, const string index, [Tiledset](#) *t)
indexe un Tiledset
- static void [allocateTileSystem](#) (map< string, [TileSystem](#) * > &m, const string index, [Tiledset](#) *t)
indexe un TileSystem

4.14.1 Description détaillée

classe statique chargée de fournir des fonctions d'allocation de ressource

4.14.2 Documentation des fonctions membres

4.14.2.1 void ResourceAllocator : :allocateFont (map< string, sf : :Font * > & m, const string index, const string path)
[static]

indexe une fonte

Paramètres

<i>m</i>	liste référencée qui stockera la ressource
<i>index</i>	nom de la ressource
<i>path</i>	chemin d'accès de la ressource sur le disque

4.14.2.2 void ResourceAllocator : :allocateMusic (map< string, sf : :Music * > & m, const string index, const string path, bool isLoop = false) [static]

indexe une musique

Paramètres

<i>m</i>	liste référencée qui stockera la ressource
<i>index</i>	nom de la ressource
<i>path</i>	chemin d'accès de la ressource sur le disque
<i>isLoop</i>	vrai si la musique se joue en boucle

4.14.2.3 void ResourceAllocator : :allocateTexture (map< string, sf : :Texture * > & m, const string index, const string path)
[static]

indexe une texture

Paramètres

<i>m</i>	liste référencée qui stockera la ressource
<i>index</i>	nom de la ressource
<i>path</i>	chemin d'accès de la ressource sur le disque

4.14.2.4 `void ResourceAllocator : :allocateTileset (map< string, Tileset * > & m, const string index, Tileset * t)`
`[static]`

indexe un [Tileset](#)

Paramètres

<i>m</i>	liste référencée qui stockera la ressource
<i>index</i>	nom de la ressource
<i>t</i>	Tileset à indexer

4.14.2.5 `void ResourceAllocator : :allocateTileSystem (map< string, TileSystem * > & m, const string index, Tileset * t)`
`[static]`

indexe un [TileSystem](#)

Paramètres

<i>m</i>	liste référencée qui stockera la ressource
<i>index</i>	nom de la ressource
<i>t</i>	Tileset que le TileSystem utilisera

La documentation de cette classe a été générée à partir des fichiers suivants :

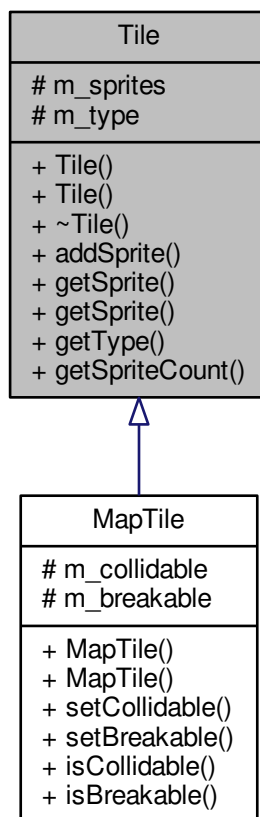
- [src/ResourceAllocator.h](#)
- [src/ResourceAllocator.cpp](#)

4.15 Référence de la classe Tile

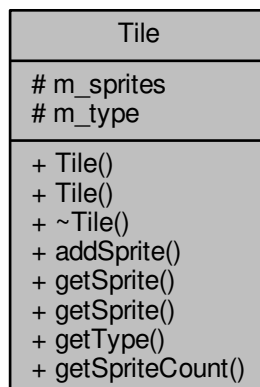
un tuile est une brique élémentaire de tout bon jeu 2D de base

```
#include <Tile.h>
```

Graphe d'héritage de Tile :



Graphe de collaboration de Tile :



Fonctions membres publiques

- `Tile` (`sf : :Sprite &s`, `TileType` `type=TileType : :DEFAULT`)
crée une tuile avec un premier sprite
- `Tile` (`TileType` `type=TileType : :DEFAULT`)
crée une tuile sans aucun sprite
- `virtual ~Tile` ()
détruit la tuile
- `void addSprite` (`sf : :Sprite &s`)
ajoute un nouveau sprite à la tuile
- `sf : :Sprite * getSprite` (`float x`, `float y`)
recupère le premier sprite du tableau de sprites aux coordonnées données (utile pour les tuiles mono-sprite)
- `sf : :Sprite * getSprite` (`unsigned int index`, `float x`, `float y`)
recupère le sprite d'indice donné du tableau de sprites aux coordonnées données (utile pour les tuiles multi-sprite)
- `TileType getType` ()
accesseur de `m_type`
- `int getSpriteCount` ()
retourne le nombre de sprites dans la tuile

Attributs protégés

- `vector< sf : :Sprite * > m_sprites`
le ou les sprite(s) de la tuile (une tuile peut-être mono ou multi-sprite)
- `TileType m_type`
le type de tuile

4.15.1 Description détaillée

un tuile est une brique élémentaire de tout bon jeu 2D de base

4.15.2 Documentation des constructeurs et destructeur

4.15.2.1 `Tile : :Tile (sf : :Sprite &s, TileType type = TileType::DEFAULT)`

crée une tuile avec un premier sprite

Une tuile ne devrait pas être créée directement. Au lieu de ça il faut idéalement utiliser `T *Tileset : :createTile(int i, int j)`.

Paramètres

<code>s</code>	le sprite à ajouter
----------------	---------------------

4.15.2.2 `Tile : :Tile (TileType type = TileType::DEFAULT)`

crée une tuile sans aucun sprite

Une tuile ne devrait pas être créée directement. Au lieu de ça il faut idéalement utiliser `T *Tileset : :createTile(vector<int> i, vector<int> j)`.

4.15.2.3 Tile::~Tile () [virtual]

détruit la tuile

Il désallouera tout le tableau de sprites, soyez donc EXTRÊMEMENT prudent, n'oubliez pas que quand vous ajoutez un sprite à une tuile, il doit être utilisé uniquement dans cette tuile !

4.15.3 Documentation des fonctions membres

4.15.3.1 void Tile::addSprite (sf::Sprite & s)

ajoute un nouveau sprite à la tuile

Paramètres

s	le sprite qui sera ajouté
---	---------------------------

4.15.3.2 sf::Sprite * Tile::getSprite (float x, float y)

récupère le premier sprite du tableau de sprites aux coordonnées données (utile pour les tuiles mono-sprite)

Attention : les coordonnées dépendent du sprite référencé par le pointeur retourné. De ce fait, appeler deux fois de suite cette méthode avec différentes coordonnées rendra le premier appel inutile. N'oubliez pas ceci si vous obtenez des résultats inattendus, par exemple si vous dessinez le même sprite un trouzillion de fois au même endroit !

Paramètres

x	abscisse où le sprite sera dessiné
y	ordonnée où le sprite sera dessiné

Renvoie

le sprite à dessiner

4.15.3.3 sf::Sprite * Tile::getSprite (unsigned int index, float x, float y)

récupère le sprite d'indice donné du tableau de sprites aux coordonnées données (utile pour les tuiles multi-sprite)

Attention : les coordonnées dépendent du sprite référencé par le pointeur retourné. De ce fait, appeler deux fois de suite cette méthode avec différentes coordonnées rendra le premier appel inutile. N'oubliez pas ceci si vous obtenez des résultats inattendus, par exemple si vous dessinez le même sprite un trouzillion de fois au même endroit !

Paramètres

index	l'indice du sprite à récupérer
x	abscisse où le sprite sera dessiné
y	ordonnée où le sprite sera dessiné

Renvoie

le sprite à dessiner

4.15.3.4 int Tile : :getSpriteCount ()

retourne le nombre de sprites dans la tuile

Renvoie

taille de m_sprites

4.15.3.5 TileType Tile : :getType ()

accesseur de m_type

Renvoie

m_type

4.15.4 Documentation des données membres**4.15.4.1 vector<sf::Sprite*> Tile : :m_sprites [protected]**

le ou les sprite(s) de la tuile (une tuile peut-être mono ou multi-sprite)

4.15.4.2 TileType Tile : :m_type [protected]

le type de tuile

La documentation de cette classe a été générée à partir des fichiers suivants :

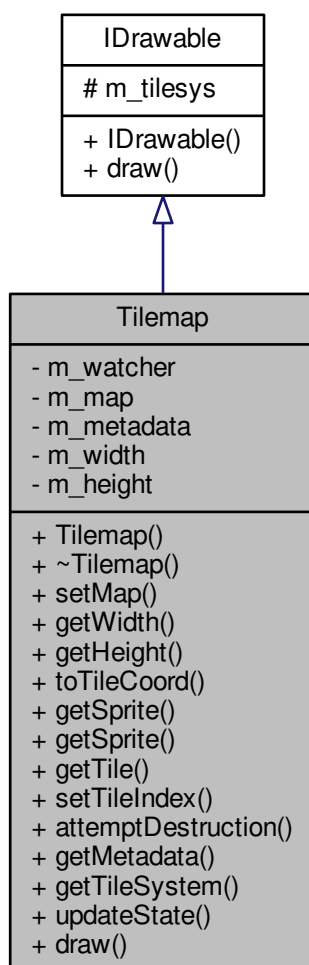
- src/[Tile.h](#)
- src/[Tile.cpp](#)

4.16 Référence de la classe Tilemap

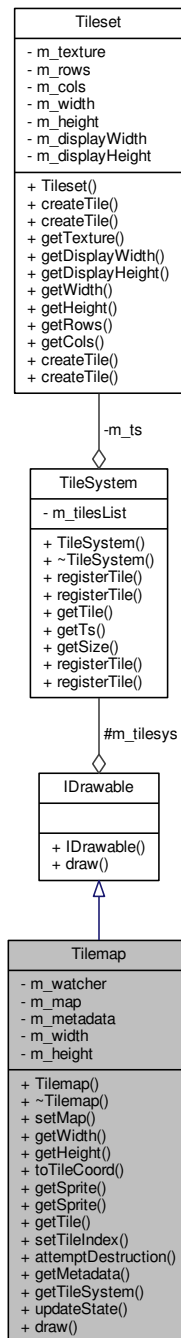
une carte est une grille de tuiles qui peut être dessinée

```
#include <Tilemap.h>
```

Graphe d'héritage de Tilemap :



Graphe de collaboration de Tilemap :



Fonctions membres publiques

- **Tilemap** (**TileSystem** *tilesys, int width, int height)
crée une carte vide avec une largeur et une hauteur définies
- virtual **~Tilemap** ()
- void **setMap** (const vector< vector< unsigned int >> &map)
mutateur de m_map
- int **getWidth** ()

- *accesseur de m_width*
int [getHeight](#) ()
- *accesseur de m_height*
sf : :Vector2i [toTileCoord](#) (sf : :Vector2f c)
- sf : :Sprite * [getSprite](#) (int i, int j)
récupère le premier sprite de la tuile aux coordonnées données
- sf : :Sprite * [getSprite](#) (unsigned int index, int i, int j)
récupère le sprite d'indice donnée de la tuile aux coordonnées données
- [Tile](#) * [getTile](#) (unsigned int i, unsigned int j)
récupère la tuile aux coordonnées données
- void [setTileIndex](#) (unsigned int index, unsigned int i, unsigned int j, int metadata=-1)
modifie l'index du [Tile](#) aux coordonnées données
- void [attemptDestruction](#) (unsigned int i, unsigned int j)
tente la destruction d'un tile (si destructible, modifie son index en l'id suivant)
- int [getMetadata](#) (unsigned int i, unsigned int j)
récupère les méta-données aux coordonnées données
- [TileSystem](#) * [getTileSystem](#) ()
accesseur de m_tilesys
- virtual void [updateState](#) (sf : :Time &elapsed)
met à jour la carte selon le temps écoulé
- void [draw](#) ([GameWindow](#) *)
contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée

Attributs privés

- sf : :Clock [m_watcher](#)
contrôle des [Tile](#) animés
- vector< vector< unsigned int > > [m_map](#)
carte d'indices (0 est un indice spécial signifiant pas de tuile, n'importe quel autre entier naturel doit correspondre à un indice de tuile dans m_tilesys)
- vector< vector< int > > [m_metadata](#)
meta-données de chaque tuile (doit faire la même taille que m_map)
- int [m_width](#)
largeur de m_map
- int [m_height](#)
hauteur de m_map

Membres hérités additionnels

4.16.1 Description détaillée

une carte est une grille de tuiles qui peut être dessinée

4.16.2 Documentation des constructeurs et destructeur

4.16.2.1 Tilemap : :Tilemap ([TileSystem](#) * *tilesys*, int *width*, int *height*)

crée une carte vide avec une largeur et une hauteur définies

Paramètres

<i>tilesys</i>	TileSystem utilisé par la carte
<i>width</i>	largeur initiale de la carte
<i>height</i>	hauteur initiale de la carte

4.16.2.2 Tilemap::~Tilemap () [virtual]

4.16.3 Documentation des fonctions membres

4.16.3.1 void Tilemap::attemptDestruction (unsigned int *i*, unsigned int *j*)

tente la destruction d'un tile (si destructible, modifie son index en l'id suivant)

Paramètres

<i>i</i>	abscisse
<i>j</i>	ordonnée

4.16.3.2 void Tilemap::draw (GameWindow * *window*) [virtual]

contrat de l'interface : une classe fille devra implémenter cette méthode en fonction de comment elle est affichée

Paramètres

<i>window</i>	fenêtre sur laquelle on dessine
---------------	---------------------------------

Implémente [IDrawable](#).

4.16.3.3 int Tilemap::getHeight ()

accesseur de m_height

Renvoie

m_height

4.16.3.4 int Tilemap::getMetadata (unsigned int *i*, unsigned int *j*)

récupère les méta-données aux coordonnées données

Une tentative de récupération d'un [Tile](#) hors carte cause un comportement indéfini.

Paramètres

<i>i</i>	abscisse
<i>j</i>	ordonnée

4.16.3.5 sf::Sprite * Tilemap::getSprite (int *i*, int *j*)

récupère le premier sprite de la tuile aux coordonnées données

Paramètres

<i>i</i>	abscisse
<i>j</i>	ordonnée

4.16.3.6 `sf::Sprite * Tilemap::getSprite (unsigned int index, int i, int j)`

récupère le sprite d'indice donnée de la tuile aux coordonnées données

Paramètres

<i>index</i>	indice du sprite
<i>i</i>	abscisse
<i>j</i>	ordonnée

4.16.3.7 `Tile * Tilemap::getTile (unsigned int i, unsigned int j)`

récupère la tuile aux coordonnées données

Paramètres

<i>i</i>	abscisse
<i>j</i>	ordonnée

4.16.3.8 `TileSystem * Tilemap::getTileSystem ()`

accesseur de `m_tilesys`

Renvoie

`m_tilesys`

4.16.3.9 `int Tilemap::getWidth ()`

accesseur de `m_width`

Renvoie

`m_width`

4.16.3.10 `void Tilemap::setMap (const vector< vector< unsigned int >> & map)`

mutateur de `m_map`

Cette méthode n'insérera pas de lignes plus larges que `m_width`, et omettra les dernières lignes si leur nombre dépasse `m_height`.

Paramètres

<i>map</i>	nouvelle valeur de m_map
------------	--------------------------

4.16.3.11 void Tilemap : :setTileIndex (unsigned int *index*, unsigned int *i*, unsigned int *j*, int *metadata* = -1)

modifie l'index du [Tile](#) aux coordonnées données

Paramètres

<i>index</i>	nouvel index
<i>i</i>	abscisse
<i>j</i>	ordonnée
<i>metadata</i>	méta-données

4.16.3.12 sf : :Vector2i Tilemap : :toTileCoord (sf : :Vector2f *c*)

4.16.3.13 void Tilemap : :updateState (sf : :Time & *elapsed*) [virtual]

met à jour la carte selon le temps écoulé

Paramètres

<i>controller</i>	contrôleur appelant
<i>elapsed</i>	temps écoulé

4.16.4 Documentation des données membres

4.16.4.1 int Tilemap : :m_height [private]

hauteur de m_map

4.16.4.2 vector<vector<unsigned int>> Tilemap : :m_map [private]

carte d'indices (0 est un indice spécial signifiant pas de tuile, n'importe quel autre entier naturel doit correspondre à un indice de tuile dans m_tilesys)

4.16.4.3 vector<vector<int>> Tilemap : :m_metadata [private]

meta-données de chaque tuile (doit faire la même taille que m_map)

4.16.4.4 `sf::Clock Tilemap::m_watcher` [private]

contrôle des [Tile](#) animés

4.16.4.5 `int Tilemap::m_width` [private]

largeur de m_map

La documentation de cette classe a été générée à partir des fichiers suivants :

- [src/Tilemap.h](#)
- [src/Tilemap.cpp](#)

4.17 Référence de la classe Tileset

un set de tuiles stocke des informations brutes à propos d'une texture de tuiles

```
#include <Tileset.h>
```

Graphe de collaboration de Tileset :

Tileset
<ul style="list-style-type: none">- m_texture- m_rows- m_cols- m_width- m_height- m_displayWidth- m_displayHeight
<ul style="list-style-type: none">+ Tileset()+ createTile()+ createTile()+ getTexture()+ getDisplayWidth()+ getDisplayHeight()+ getWidth()+ getHeight()+ getRows()+ getCols()+ createTile()+ createTile()

Fonctions membres publiques

- `Tileset` (const sf : :Texture *tex, int rows, int cols, int width, int height, int displayWidth, int displayHeight)
crée un set de tuiles avec les paramètres donnés
- template<typename T = Tile>
T * `createTile` (int i, int j, `TileType` ty=`TileType` : :DEFAULT, bool c=false, bool b=false)
crée une tuile avec seulement un sprite dedans
- template<typename T = Tile>
T * `createTile` (vector< int > i, vector< int > j, `TileType` ty=`TileType` : :DEFAULT, bool c=false, bool b=false)
crée une tuile avec de multiples sprites dedans
- const sf : :Texture * `getTexture` ()
accesseur de m_texture
- int `getDisplayWidth` ()
accesseur de m_displayWidth
- int `getDisplayHeight` ()
accesseur de m_displayHeight
- int `getWidth` ()
accesseur de m_width
- int `getHeight` ()
accesseur de m_height
- int `getRows` ()
accesseur de m_rows
- int `getCols` ()
accesseur de m_cols
- template<>
`MapTile` * `createTile` (int i, int j, `TileType` ty, bool c, bool b)
- template<>
`MapTile` * `createTile` (vector< int > i, vector< int > j, `TileType` ty, bool c, bool b)

Attributs privés

- const sf : :Texture * `m_texture`
la texture du set de tuiles
- int `m_rows`
nombre de lignes dans la texture
- int `m_cols`
nombre de colonnes dans la texture
- int `m_width`
largeur d'une tuile dans la texture
- int `m_height`
hauteur d'une tuile dans la texture
- int `m_displayWidth`
largeur d'affichage d'une tuile
- int `m_displayHeight`
hauteur d'affichage d'une tuile

4.17.1 Description détaillée

un set de tuiles stocke des informations brutes à propos d'une texture de tuiles

4.17.2 Documentation des constructeurs et destructeur

4.17.2.1 `Tileset` : :`Tileset` (const sf : :Texture * tex, int rows, int cols, int width, int height, int displayWidth, int displayHeight)

crée un set de tuiles avec les paramètres donnés

Paramètres

<i>tex</i>	texture à utiliser
<i>rows</i>	nombre de lignes
<i>cols</i>	nombre de colonnes
<i>width</i>	largeur d'une tuile
<i>height</i>	hauteur d'une tuile
<i>displayWidth</i>	largeur d'affichage d'une tuile
<i>displayHeight</i>	hauteur d'affichage d'une tuile

4.17.3 Documentation des fonctions membres

4.17.3.1 `template<> MapTile* Tileset::createTile (int i, int j, TileType ty, bool c, bool b)`

4.17.3.2 `template<> MapTile* Tileset::createTile (vector< int > i, vector< int > j, TileType ty, bool c, bool b)`

4.17.3.3 `template<typename T = Tile> T* Tileset::createTile (int i, int j, TileType ty = TileType::DEFAULT, bool c = false, bool b = false) [inline]`

crée une tuile avec seulement un sprite dedans

Paramètres

<i>i</i>	abscisse (en termes de colonnes de tuiles)
<i>j</i>	ordonnée (en termes de colonnes de tuiles)
<i>ty</i>	le type de tuile
<i>c</i>	vrai si la tuile est physique (n'aura aucun effet si on ne crée pas une tuile MapTile)
<i>b</i>	vrai si la tuile est destructible (n'aura aucun effet si on ne crée pas une tuile MapTile)

4.17.3.4 `template<typename T = Tile> T* Tileset::createTile (vector< int > i, vector< int > j, TileType ty = TileType::DEFAULT, bool c = false, bool b = false) [inline]`

crée une tuile avec de multiples sprites dedans

Paramètres

<i>i</i>	abscisses (en termes de colonnes de tuiles)
<i>j</i>	ordonnées (en termes de colonnes de tuiles)
<i>ty</i>	le type de tuile
<i>c</i>	vrai si la tuile est physique (n'aura aucun effet si on ne crée pas une tuile MapTile)
<i>b</i>	vrai si la tuile est destructible (n'aura aucun effet si on ne crée pas une tuile MapTile)

4.17.3.5 `int Tileset::getCols ()`

accesseur de m_cols

Renvoie

m_cols

4.17.3.6 int Tileset : :getDisplayHeight ()

accesseur de m_displayHeight

Renvoie

m_displayHeight

4.17.3.7 int Tileset : :getDisplayWidth ()

accesseur de m_displayWidth

Renvoie

m_displayWidth

4.17.3.8 int Tileset : :getHeight ()

accesseur de m_height

Renvoie

m_height

4.17.3.9 int Tileset : :getRows ()

accesseur de m_rows

Renvoie

m_rows

4.17.3.10 const sf : :Texture * Tileset : :getTexture ()

accesseur de m_texture

Renvoie

m_texture

4.17.3.11 int Tileset::getWidth ()

accesseur de m_width

Renvoie

m_width

4.17.4 Documentation des données membres

4.17.4.1 int Tileset::m_cols [private]

nombre de colonnes dans la texture

4.17.4.2 int Tileset::m_displayHeight [private]

hauteur d'affichage d'une tuile

4.17.4.3 int Tileset::m_displayWidth [private]

largeur d'affichage d'une tuile

4.17.4.4 int Tileset::m_height [private]

hauteur d'une tuile dans la texture

4.17.4.5 int Tileset::m_rows [private]

nombre de lignes dans la texture

4.17.4.6 const sf::Texture* Tileset::m_texture [private]

la texture du set de tuiles

4.17.4.7 int Tileset::m_width [private]

largeur d'une tuile dans la texture

La documentation de cette classe a été générée à partir des fichiers suivants :

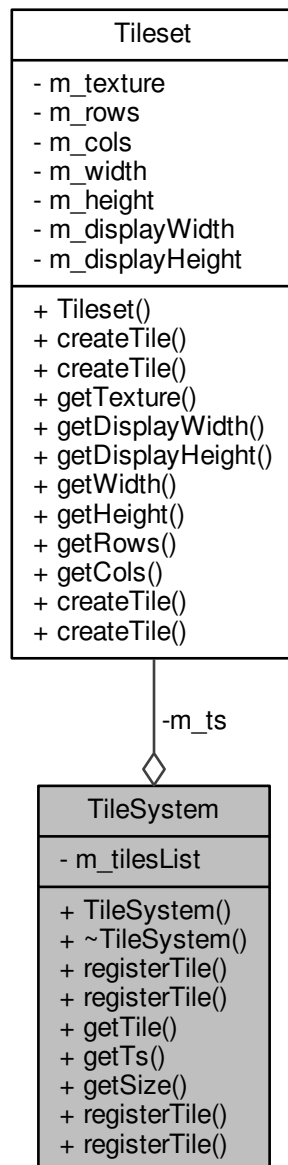
- src/[Tileset.h](#)
- src/[Tileset.cpp](#)

4.18 Référence de la classe TileSystem

un système de tuiles lie les données brutes d'un [Tileset](#) avec tous les [Tile](#) extraits de ce [Tileset](#)

```
#include <TileSystem.h>
```

Graphes de collaboration de TileSystem :



Fonctions membres publiques

— [TileSystem](#) ([Tileset](#) *ts)

```

    crée un système de tuiles
— virtual ~TileSystem ()
    détruit le système de tuiles
— template<typename T = Tile>
    void registerTile (unsigned int index, int i, int j, TileType ty=TileType : :DEFAULT, bool c=NULL, bool b=NULL)
        indexe un Tile mono-sprite
— template<typename T = Tile>
    void registerTile (unsigned int index, vector< int > i, vector< int > j, TileType ty=TileType : :DEFAULT, bool
        c=NULL, bool b=NULL)
        indexe un Tile multi-sprite
— Tile * getTile (unsigned int index)
        retourne le Tile d'indice donné
— Tileset * getTs ()
        accesseur de m_ts
— int getSize ()
        nombre de Tile indexés
— template<>
    void registerTile (unsigned int index, int i, int j, TileType ty, bool c, bool b)
— template<>
    void registerTile (unsigned int index, vector< int > i, vector< int > j, TileType ty, bool c, bool b)

```

Attributs privés

```

— Tileset * m_ts
    Tileset utilisé par le système.
— map< unsigned int, Tile * > m_tilesList
    liste de toutes les tuiles déclarées

```

4.18.1 Description détaillée

un système de tuiles lie les données brutes d'un Tileset avec tous les Tile extraits de ce Tileset

4.18.2 Documentation des constructeurs et destructeur

4.18.2.1 TileSystem : :TileSystem (Tileset * ts)

crée un système de tuiles

Paramètres

<i>ts</i>	le Tileset à utiliser
-----------	-----------------------

4.18.2.2 TileSystem : :~TileSystem () [virtual]

détruit le système de tuiles

Désallouera tous les Tiles.

4.18.3 Documentation des fonctions membres

4.18.3.1 int TileSystem : :getSize ()

nombre de Tile indexés

Renvoie

taille de `m_tilesList`

4.18.3.2 `Tile * TileSystem : :getTile (unsigned int index)`

retourne le `Tile` d'indice donné

Paramètres

<i>index</i>	indice du <code>Tile</code>
--------------	-----------------------------

Renvoie

pointeur sur le `Tile`

4.18.3.3 `Tileset * TileSystem : :getTs ()`

accesseur de `m_ts`

Renvoie

`m_ts`

4.18.3.4 `template<> void TileSystem : :registerTile (unsigned int index, int i, int j, TileType ty, bool c, bool b)`4.18.3.5 `template<> void TileSystem : :registerTile (unsigned int index, vector< int > i, vector< int > j, TileType ty, bool c, bool b)`4.18.3.6 `template<typename T = Tile> void TileSystem : :registerTile (unsigned int index, int i, int j, TileType ty = TileType::DEFAULT, bool c = NULL, bool b = NULL) [inline]`

indexe un `Tile` mono-sprite

Paramètres

<i>index</i>	indice du <code>Tile</code>
<i>i</i>	abscisse du sprite dans le <code>Tileset</code>
<i>j</i>	ordonnée du sprite dans le <code>Tileset</code>
<i>ty</i>	le type de tuile
<i>c</i>	vrai si la tuile est physique (n'aura aucun effet si on ne crée pas une tuile <code>MapTile</code>)
<i>b</i>	vrai si la tuile est destructible (n'aura aucun effet si on ne crée pas une tuile <code>MapTile</code>)

4.18.3.7 `template<typename T = Tile> void TileSystem : :registerTile (unsigned int index, vector< int > i, vector< int > j, TileType ty = TileType::DEFAULT, bool c = NULL, bool b = NULL) [inline]`

indexe un `Tile` multi-sprite

Paramètres

<i>index</i>	indice du Tile
<i>i</i>	abscisses des sprites dans le Tilesset
<i>j</i>	ordonnée des sprites dans le Tilesset
<i>ty</i>	le type de tuile
<i>c</i>	vrai si la tuile est physique (n'aura aucun effet si on ne crée pas une tuile MapTile)
<i>b</i>	vrai si la tuile est destructible (n'aura aucun effet si on ne crée pas une tuile MapTile)

4.18.4 Documentation des données membres

4.18.4.1 `map<unsigned int, Tile*> TileSystem : :m_tilesList` [private]

liste de toutes les tuiles déclarées

4.18.4.2 `Tilesset* TileSystem : :m_ts` [private]

[Tilesset](#) utilisé par le système.

La documentation de cette classe a été générée à partir des fichiers suivants :

- [src/TileSystem.h](#)
- [src/TileSystem.cpp](#)

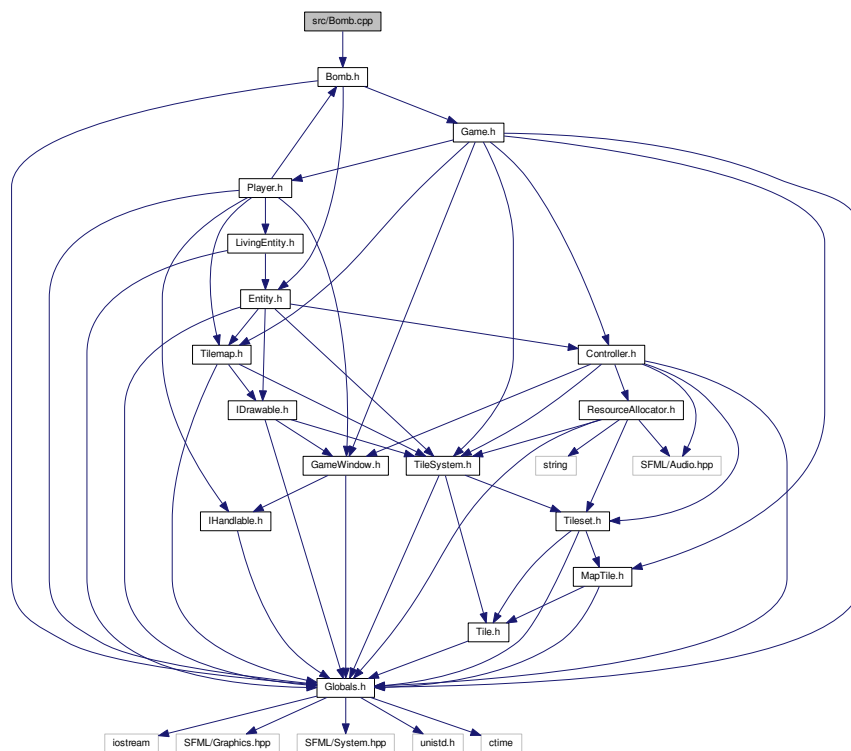
Chapitre 5

Documentation des fichiers

5.1 Référence du fichier src/Bomb.cpp

```
#include "Bomb.h"
```

Graphe des dépendances par inclusion de Bomb.cpp :



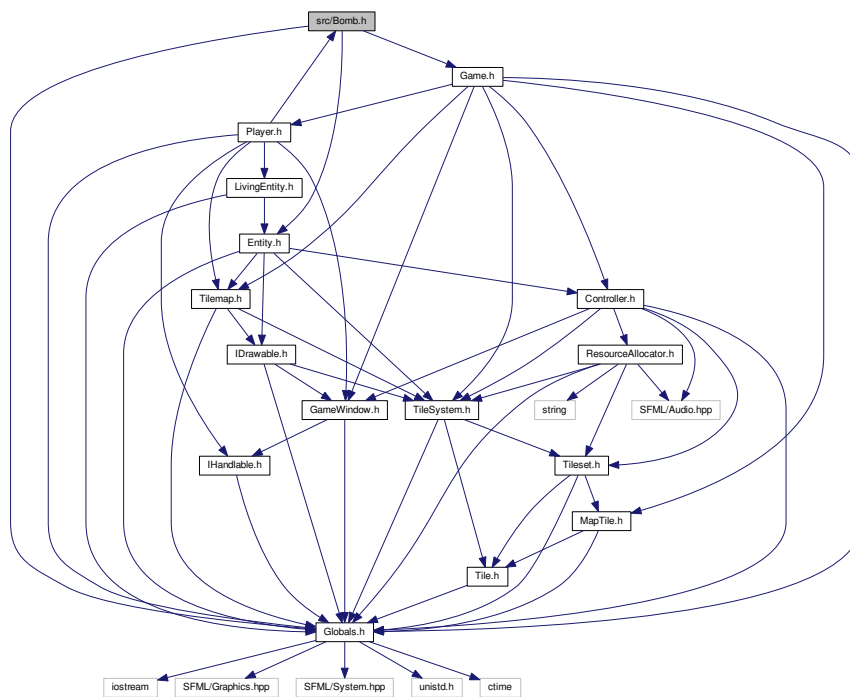
5.2 Référence du fichier src/Bomb.h

```
#include "Globals.h"
```

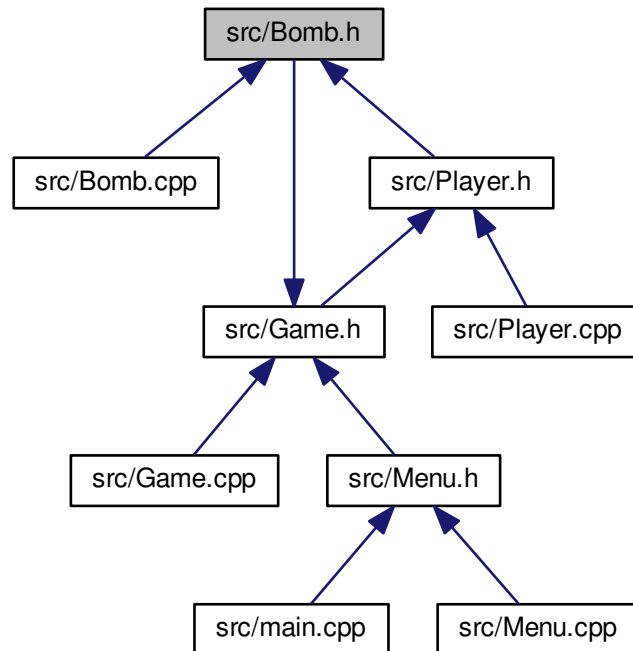
```
#include "Entity.h"
```

```
#include "Game.h"
```

Graphe des dépendances par inclusion de Bomb.h :



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



Classes

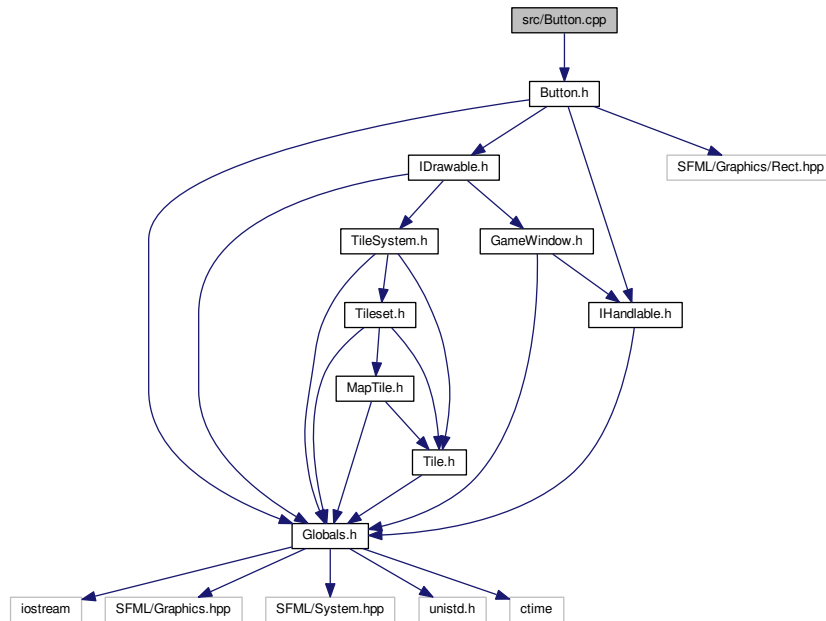
— class [Bomb](#)

Une bombe est l'élément central du jeu : elle explose les pierres et tue les joueurs.

5.3 Référence du fichier src/Button.cpp

```
#include "Button.h"
```

Graphe des dépendances par inclusion de Button.cpp :



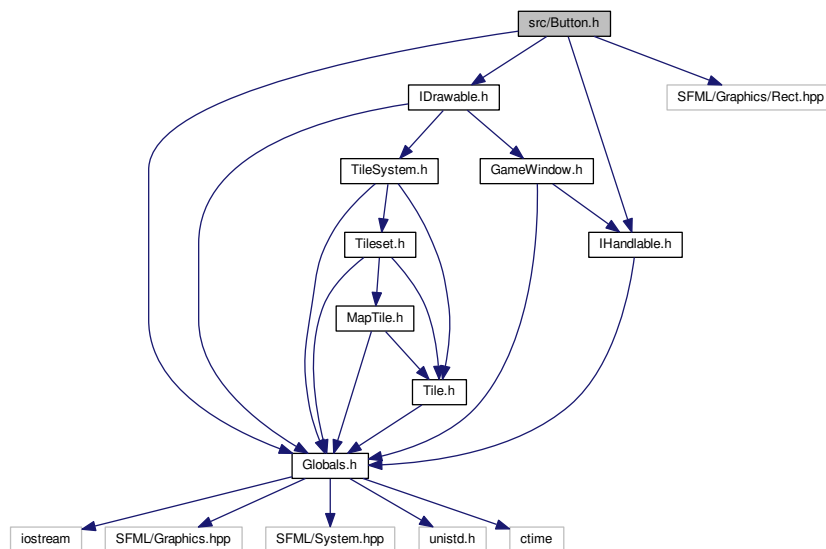
5.4 Référence du fichier src/Button.h

```

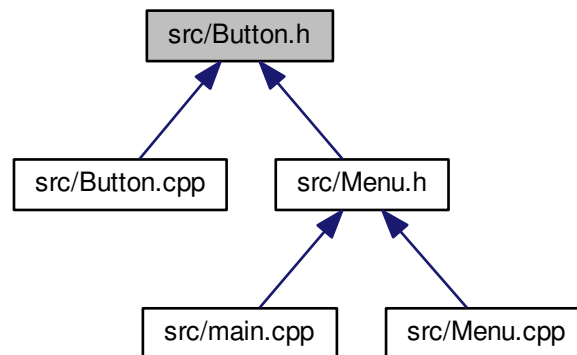
#include "Globals.h"
#include "IDrawable.h"
#include "IHandlable.h"
#include <SFML/Graphics/Rect.hpp>

```

Graphe des dépendances par inclusion de Button.h :



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



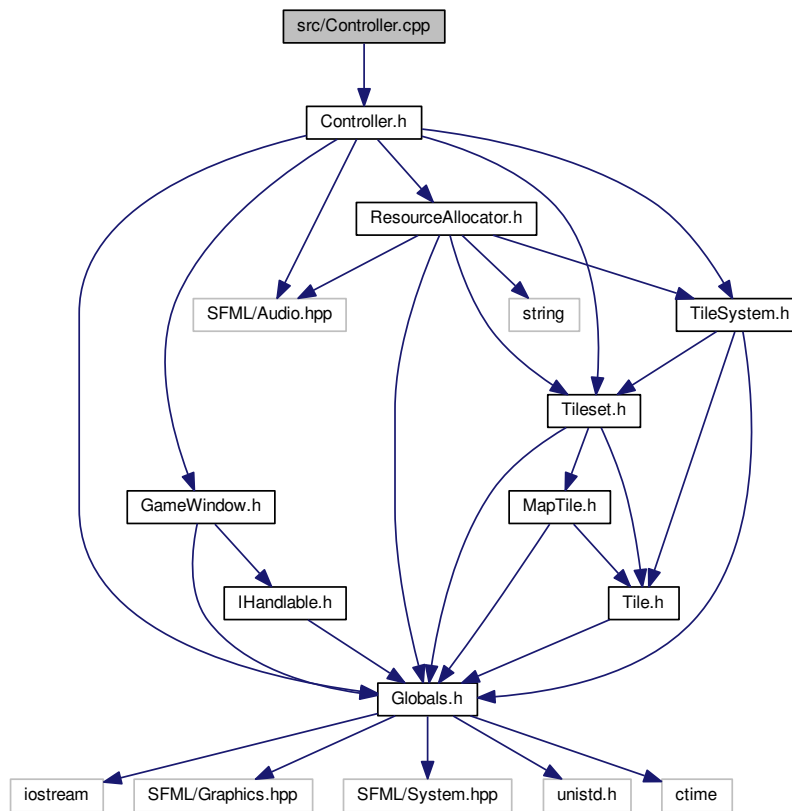
Classes

— class `Button`
gestion de boutons cliquables

5.5 Référence du fichier src/Controller.cpp

```
#include "Controller.h"
```

Graphe des dépendances par inclusion de Controller.cpp :



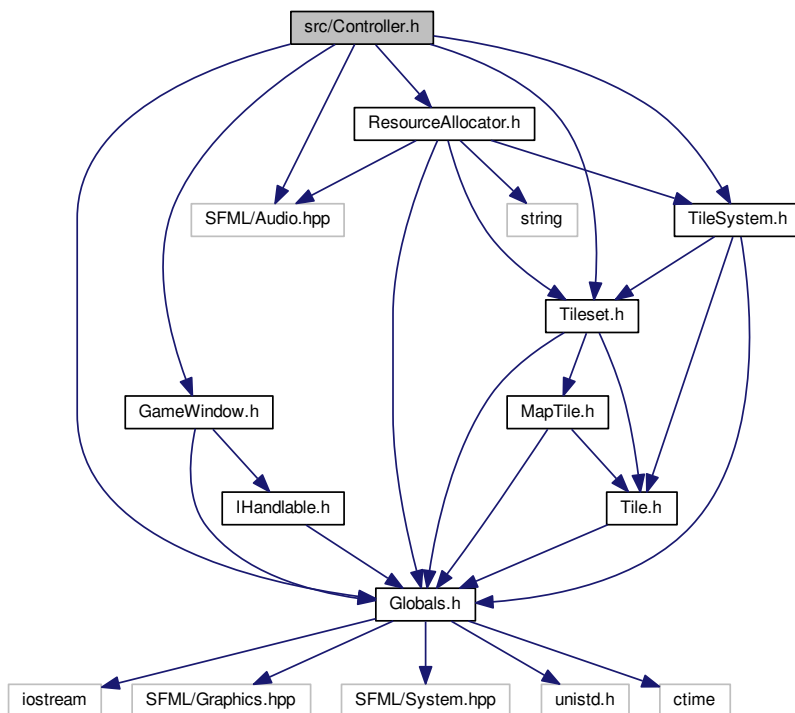
5.6 Référence du fichier src/Controller.h

```

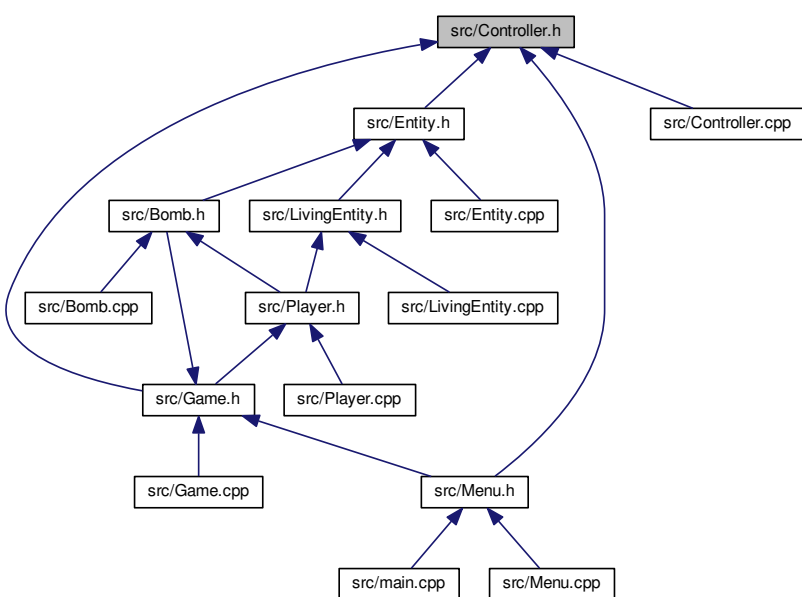
#include <SFML/Audio.hpp>
#include "Globals.h"
#include "GameWindow.h"
#include "Tileset.h"
#include "TileSystem.h"
#include "ResourceAllocator.h"

```

Graphe des dépendances par inclusion de Controller.h :



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



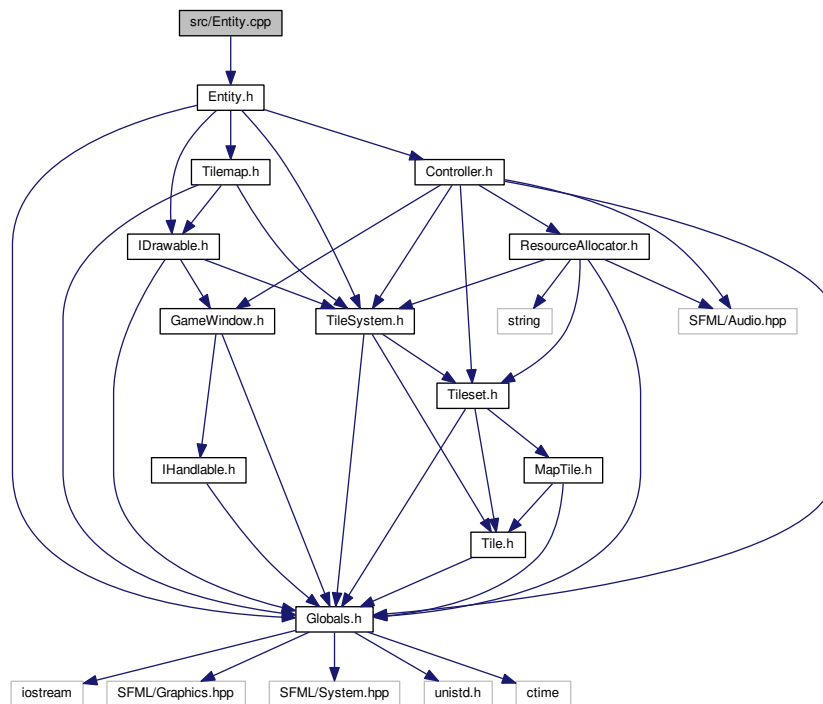
Classes

- class [Controller](#)
classe abstraite des contrôleurs de jeu

5.7 Référence du fichier src/Entity.cpp

```
#include "Entity.h"
```

Graphe des dépendances par inclusion de Entity.cpp :



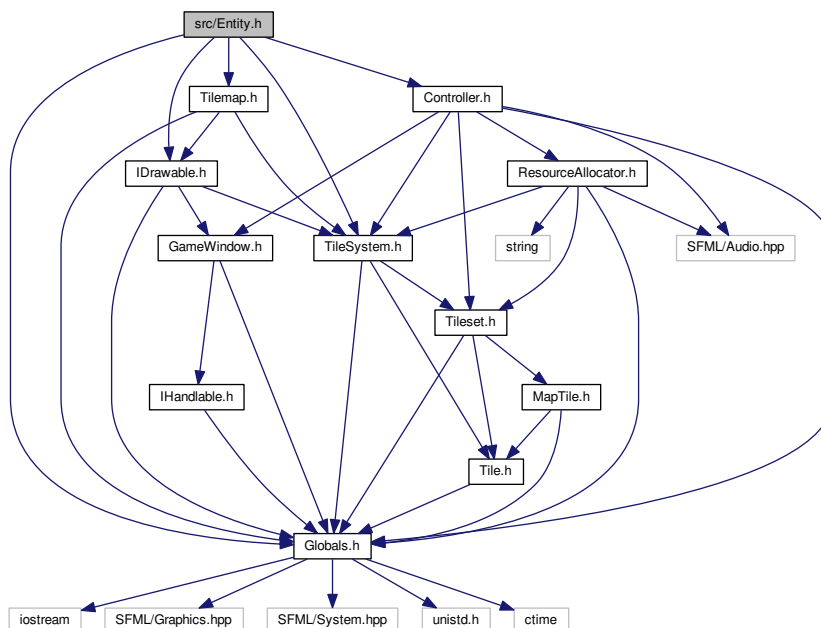
5.8 Référence du fichier src/Entity.h

```

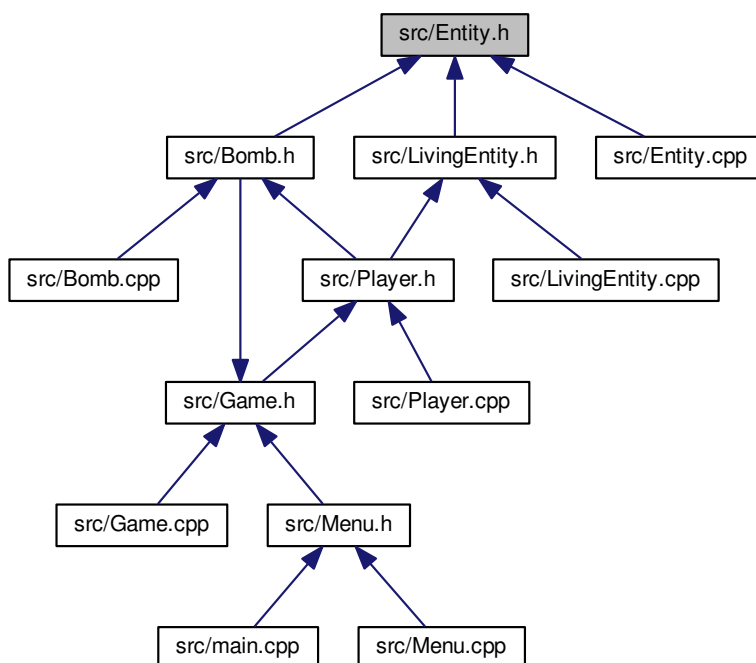
#include "Globals.h"
#include "IDrawable.h"
#include "TileSystem.h"
#include "Tilemap.h"
#include "Controller.h"

```

Graphe des dépendances par inclusion de Entity.h :



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



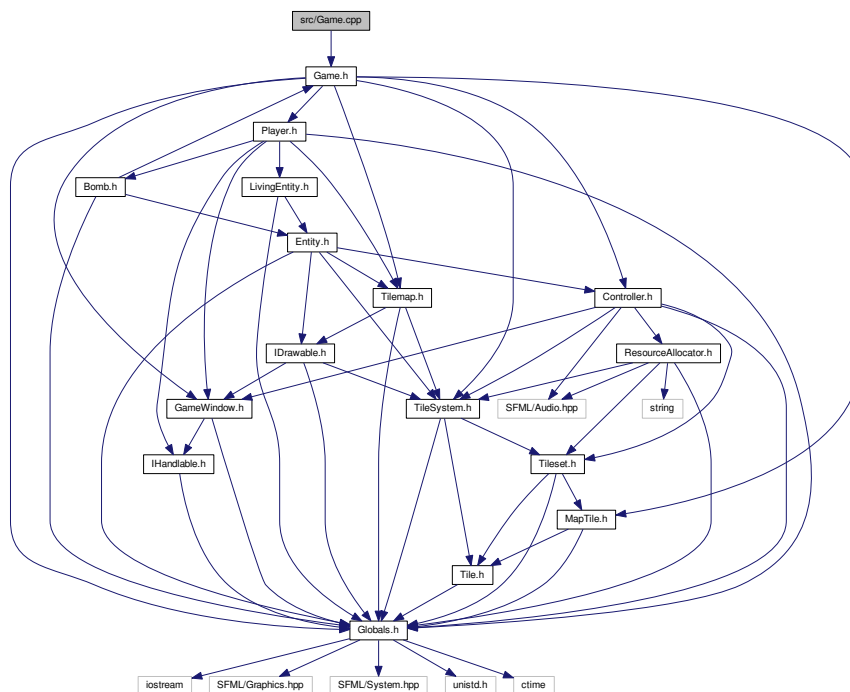
Classes

- class [Entity](#)
Base abstraite des entités du jeu.

5.9 Référence du fichier src/Game.cpp

```
#include "Game.h"
```

Graphe des dépendances par inclusion de Game.cpp :



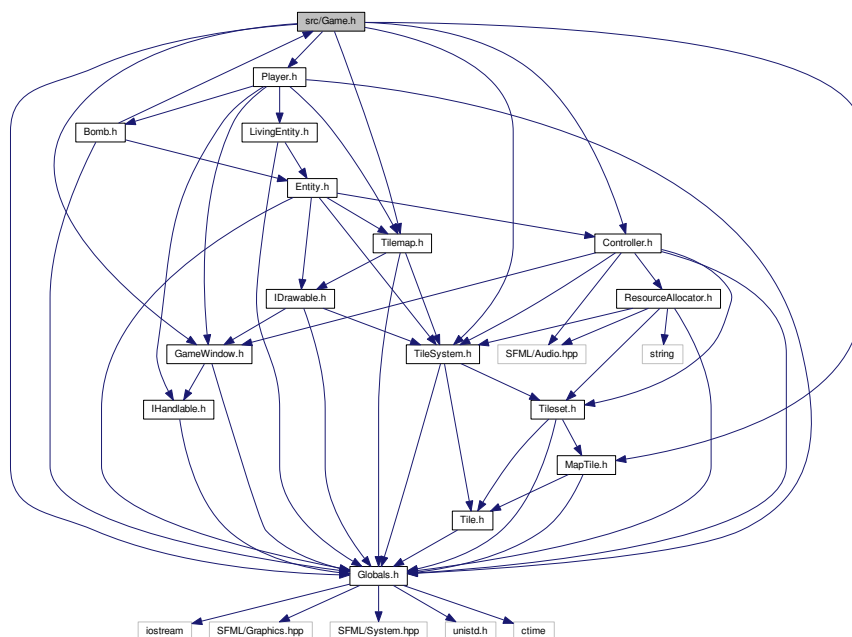
5.10 Référence du fichier src/Game.h

```

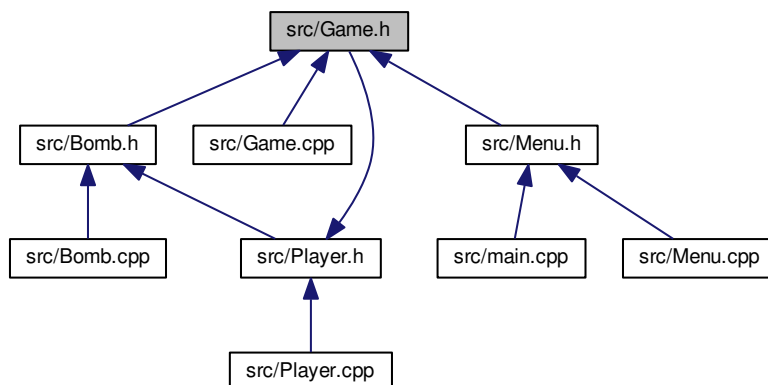
#include "Globals.h"
#include "GameWindow.h"
#include "TileSystem.h"
#include "Tilemap.h"
#include "MapTile.h"
#include "Controller.h"
#include "Player.h"

```


Graphe des dépendances par inclusion de Game.h :



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



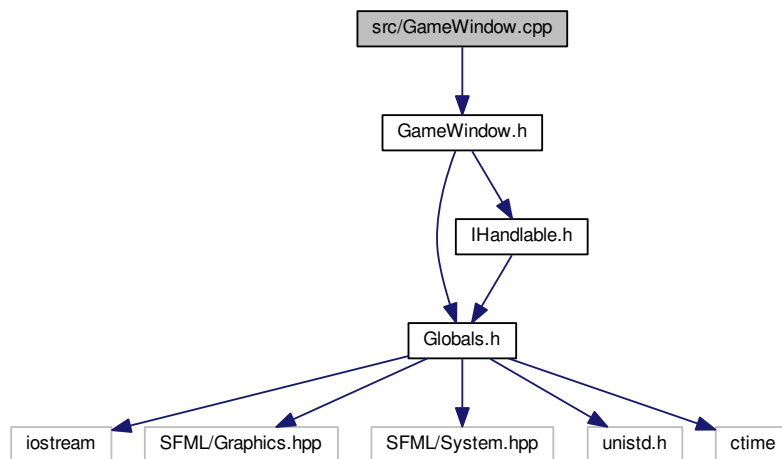
Classes

— class `Game`
contrôleur de partie

5.11 Référence du fichier src/GameWindow.cpp

```
#include "GameWindow.h"
```

Graphe des dépendances par inclusion de GameWindow.cpp :

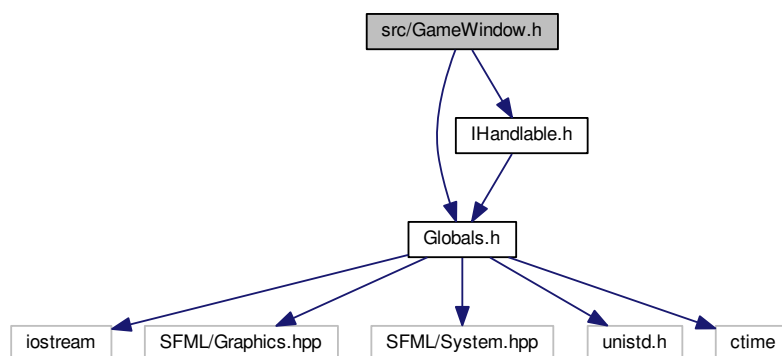


5.12 Référence du fichier src/GameWindow.h

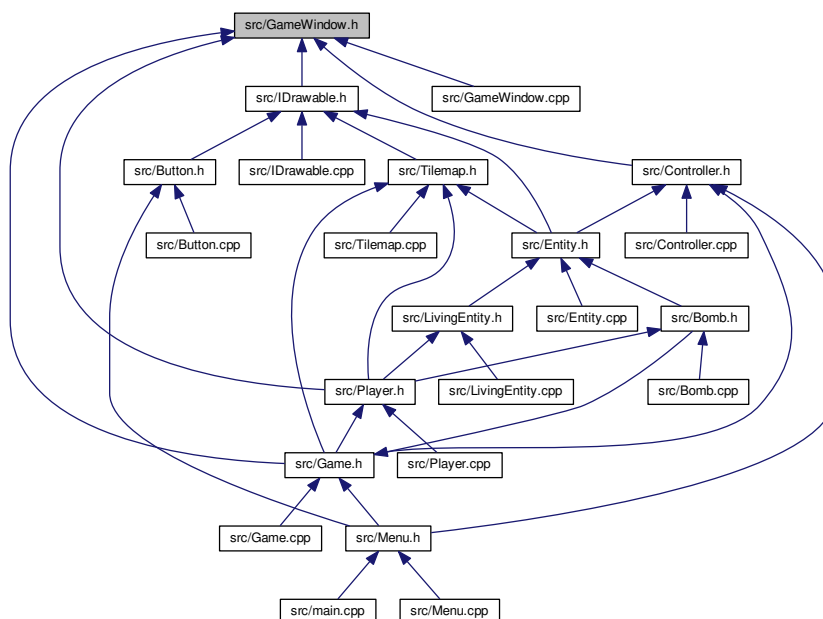
```
#include "Globals.h"
```

```
#include "IHandlable.h"
```

Graphe des dépendances par inclusion de GameWindow.h :



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



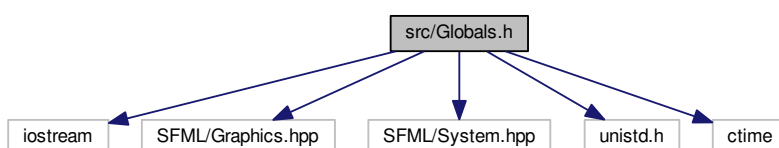
Classes

- class `GameWindow`
vue principale du jeu

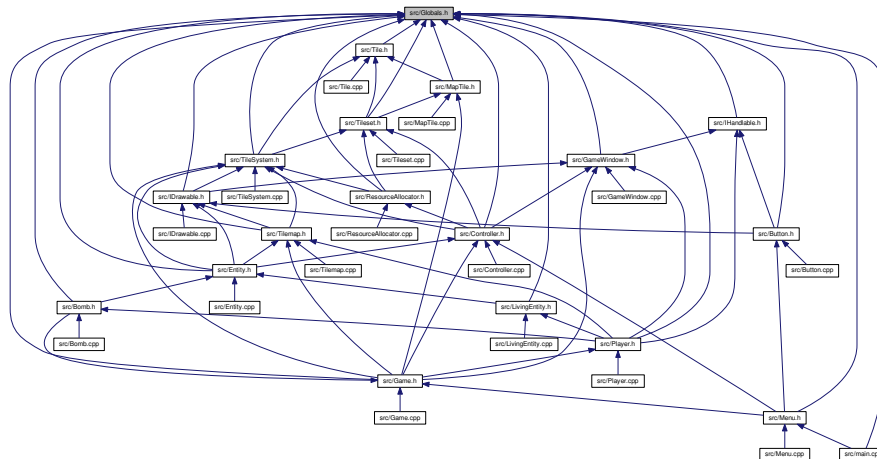
5.13 Référence du fichier src/Globals.h

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <unistd.h>
#include <ctime>
```

Graphe des dépendances par inclusion de Globals.h :



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



Macros

```
— #define DEFAULT_TILE_SIZE 50
— #define DEBUG_MODE false
— #define G_VERSION "dev"
— #define G_AUTHORS "Guillaume JORANDON & Clément SIMON"
— #define SPEED_FACTOR 3
— #define ABS(X) (((X)<0 ? (-X) : (X)))
— #define SGN(X) (((X)>0 ? (true) : (false)))
— #define GENERIC_ERROR 1
— #define TEXTURE_ERROR 2
— #define TILESET_ERROR 3
— #define MUSIC_ERROR 4
— #define FONT_ERROR 5
```

Énumérations

```
— enum TileType { DEFAULT =1, RANDOMIZED =2, ANIMATED =3, TRANSITIONAL =4 }
    différents types de Tile
— enum Orientation { TOP =1, RIGHT =2, BOTTOM =3, LEFT =4 }
    orientations des entités vivantes
```

5.13.1 Documentation des macros

5.13.1.1 `#define ABS(X) (((X)<0 ? (-X) : (X)))`

5.13.1.2 `#define DEBUG_MODE false`

5.13.1.3 `#define DEFAULT_TILE_SIZE 50`

5.13.1.4 `#define FONT_ERROR 5`

5.13.1.5 `#define G_AUTHORS "Guillaume JORANDON & Clément SIMON"`

5.13.1.6 `#define G_VERSION "dev"`

5.13.1.7 `#define GENERIC_ERROR 1`

5.13.1.8 `#define MUSIC_ERROR 4`

5.13.1.9 `#define SGN(X) (((X)>0?(true):(false)))`

5.13.1.10 `#define SPEED_FACTOR 3`

5.13.1.11 `#define TEXTURE_ERROR 2`

5.13.1.12 `#define TILESET_ERROR 3`

5.13.2 Documentation du type de l'énumération

5.13.2.1 enum Orientation

orientations des entités vivantes

Valeurs énumérées

TOP

RIGHT

BOTTOM

LEFT

5.13.2.2 enum TileType

différents types de [Tile](#)

Valeurs énumérées

DEFAULT

RANDOMIZED

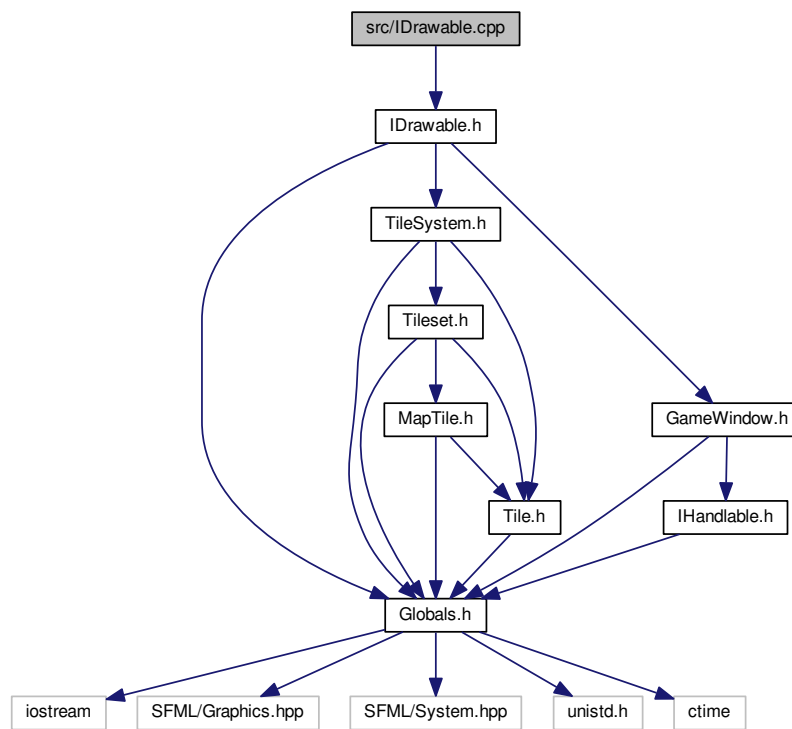
ANIMATED

TRANSITIONAL

5.14 Référence du fichier src/IDrawable.cpp

```
#include "IDrawable.h"
```

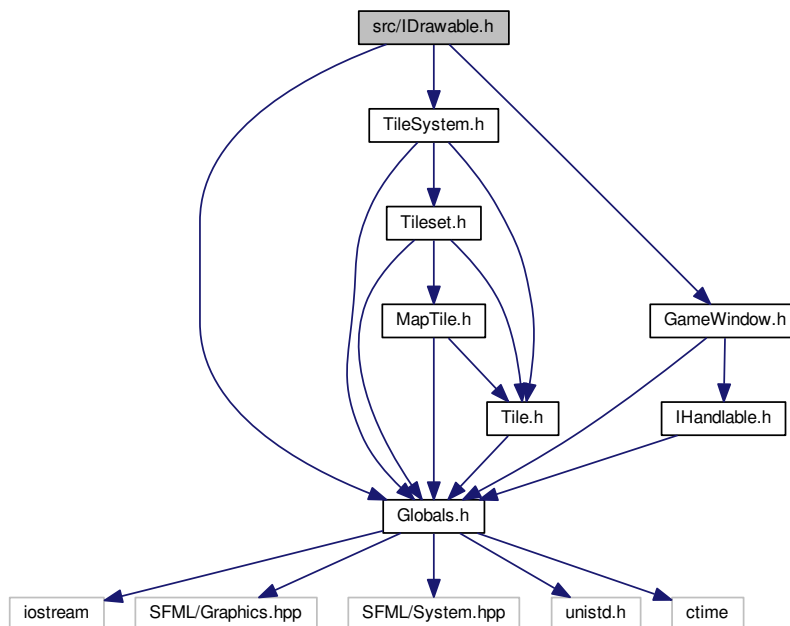
Graphe des dépendances par inclusion de IDrawable.cpp :



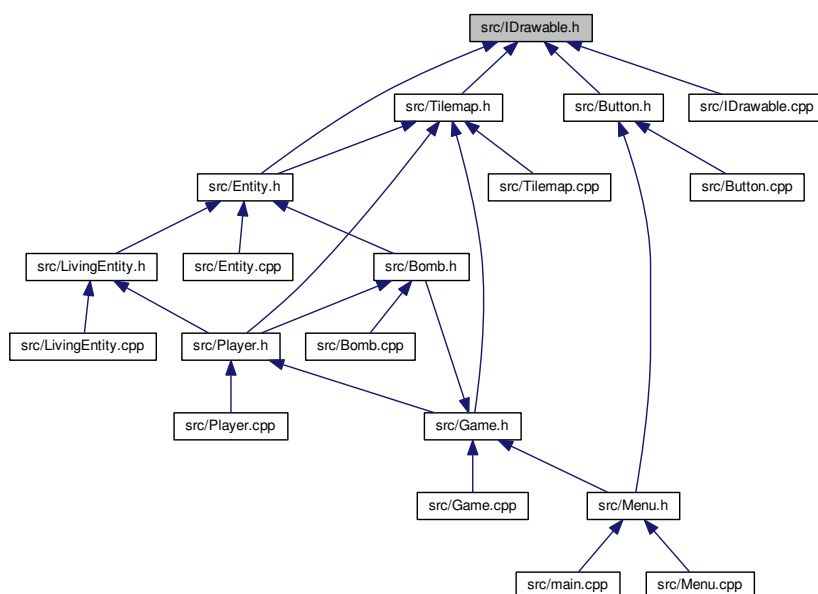
5.15 Référence du fichier src/IDrawable.h

```
#include "Globals.h"
#include "TileSystem.h"
#include "GameWindow.h"
```

Graphe des dépendances par inclusion de IDrawable.h :



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



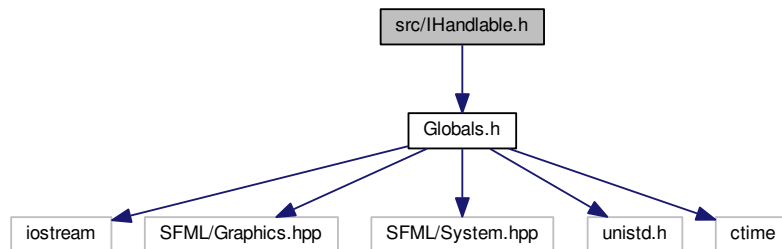
Classes

- class [IDrawable](#)
Interface implémentée par chaque classe qui peut être affichée.

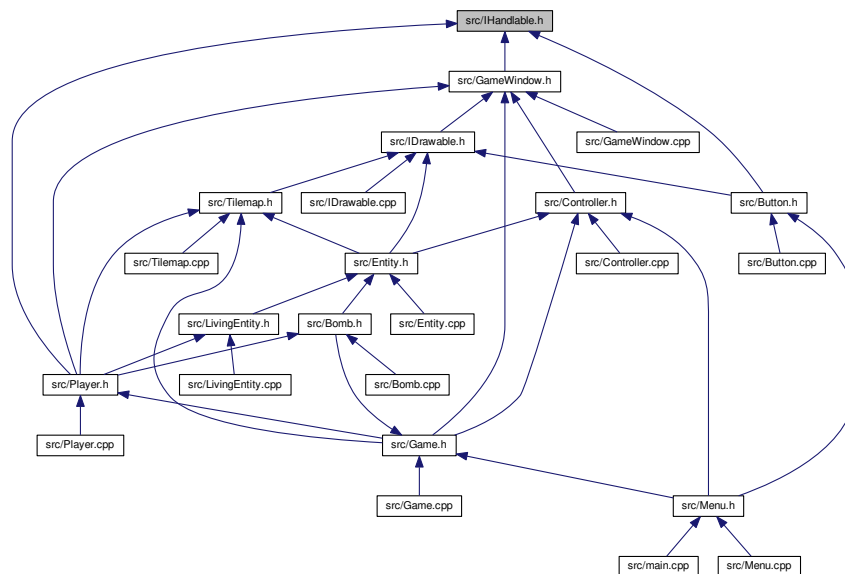
5.16 Référence du fichier src/IHandlable.h

```
#include "Globals.h"
```

Graphe des dépendances par inclusion de IHandlable.h :



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



Classes

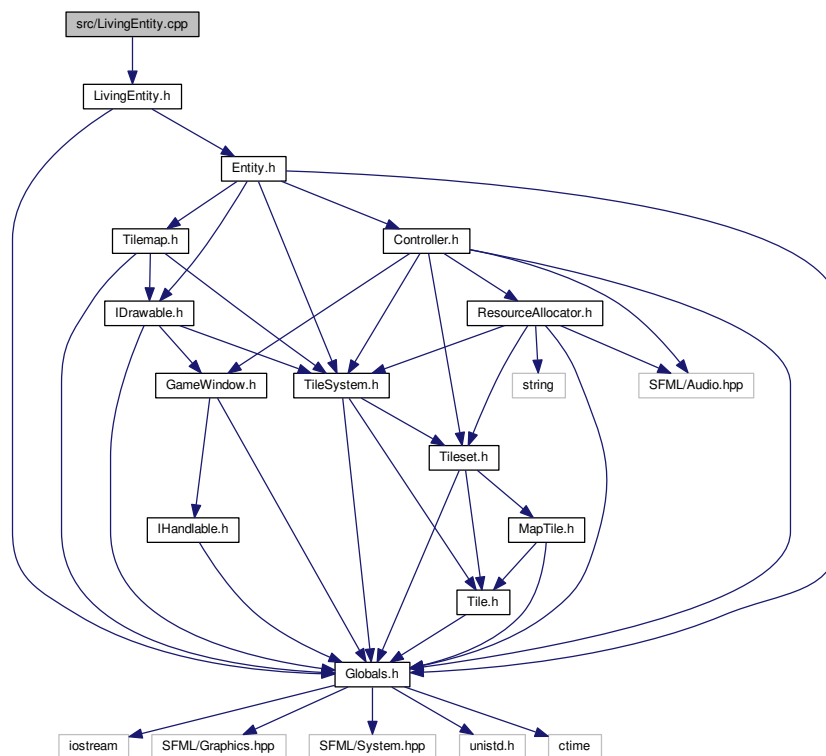
— class [IHandlable](#)

Interface implémentée par chaque classe qui a des évènements individuels à gérer.

5.17 Référence du fichier src/LivingEntity.cpp

```
#include "LivingEntity.h"
```

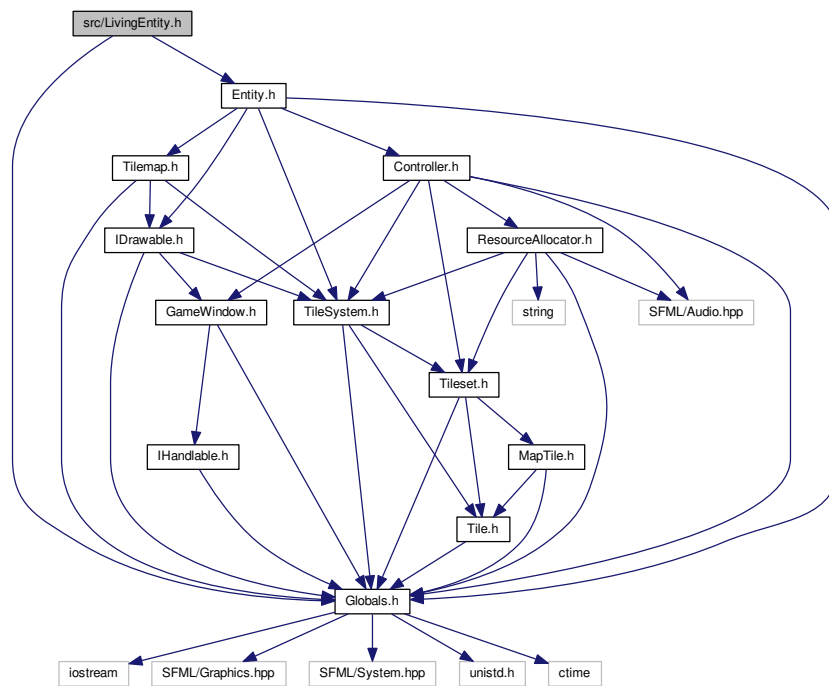
Graphe des dépendances par inclusion de LivingEntity.cpp :



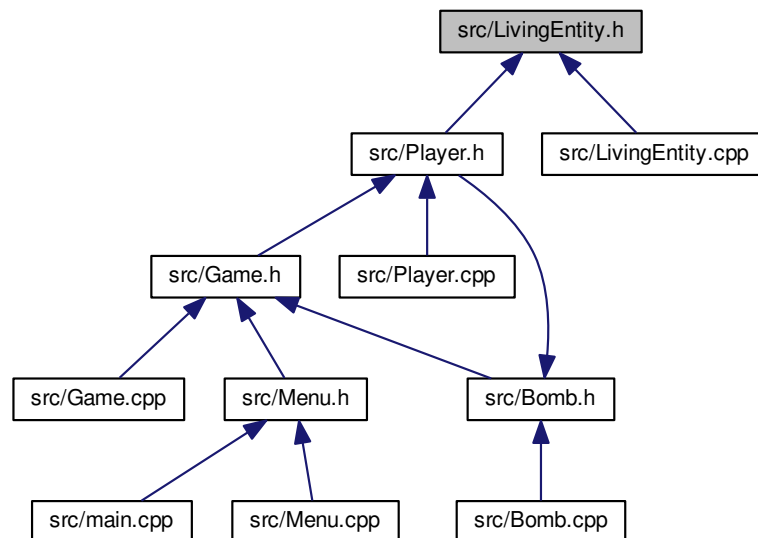
5.18 Référence du fichier src/LivingEntity.h

```
#include "Globals.h"
#include "Entity.h"
```

Graphe des dépendances par inclusion de LivingEntity.h :



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



Classes

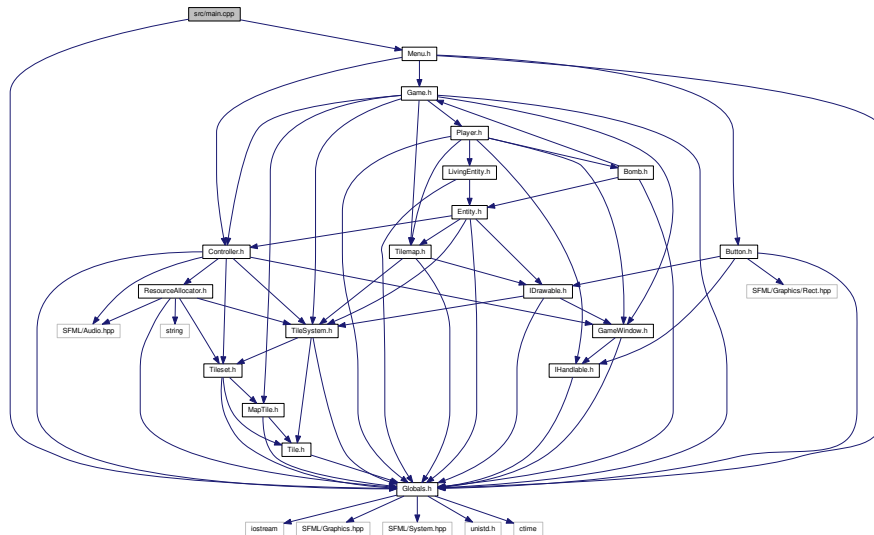
— class [LivingEntity](#)

une entité vivante peut se mouvoir, et possède une orientation

5.19 Référence du fichier src/main.cpp

```
#include "Globals.h"
#include "Menu.h"
```

Graphe des dépendances par inclusion de main.cpp :



Classes

- struct [nStream](#)
*une structure qui décrit un stream buffer nul (équivalent en C++ du /dev/null sur les systèmes *nix)*

Fonctions

- void [displayHelp](#) (const char *name)
affiche l'aide sur la sortie standard
- int [main](#) (int argc, char *argv[])
Fonction principale, lit les arguments de la ligne de commande et lance le jeu.

5.19.1 Documentation des fonctions

5.19.1.1 void displayHelp (const char * name)

affiche l'aide sur la sortie standard

Paramètres

<i>nom</i>	de la commande
------------	----------------

5.19.1.2 `int main (int argc, char * argv[])`

Fonction principale, lit les arguments de la ligne de commande et lance le jeu.

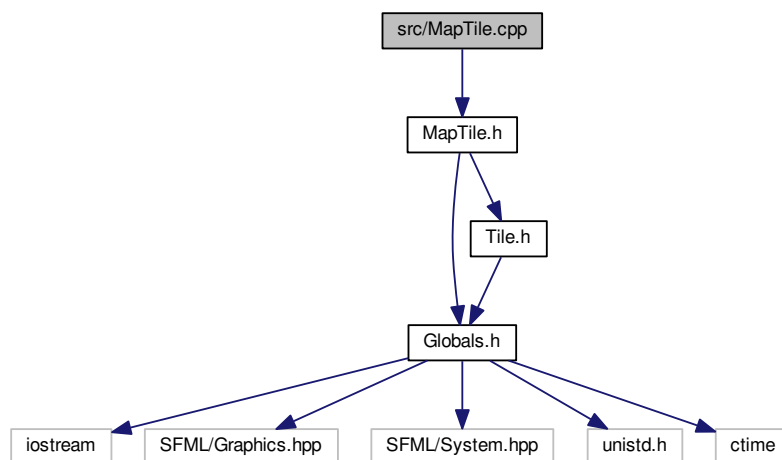
Paramètres

<code>argc</code>	nombre d'arguments (rly ?)
<code>argv</code>	les arguments de la ligne de commande (O, RLY ? ? ?)

5.20 Référence du fichier `src/MapTile.cpp`

```
#include "MapTile.h"
```

Graphe des dépendances par inclusion de `MapTile.cpp` :



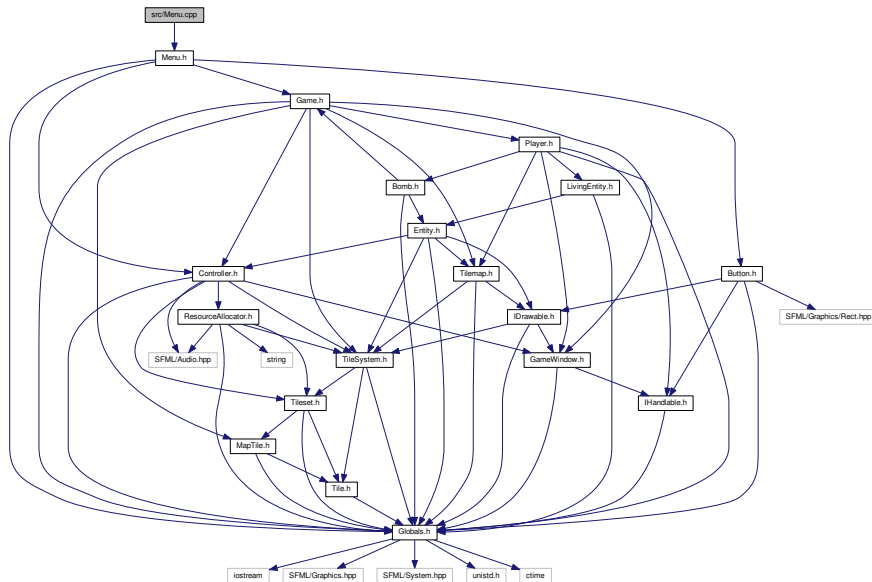
5.21 Référence du fichier `src/MapTile.h`

```
#include "Globals.h"  
#include "Tile.h"
```


5.22 Référence du fichier src/Menu.cpp

```
#include "Menu.h"
```

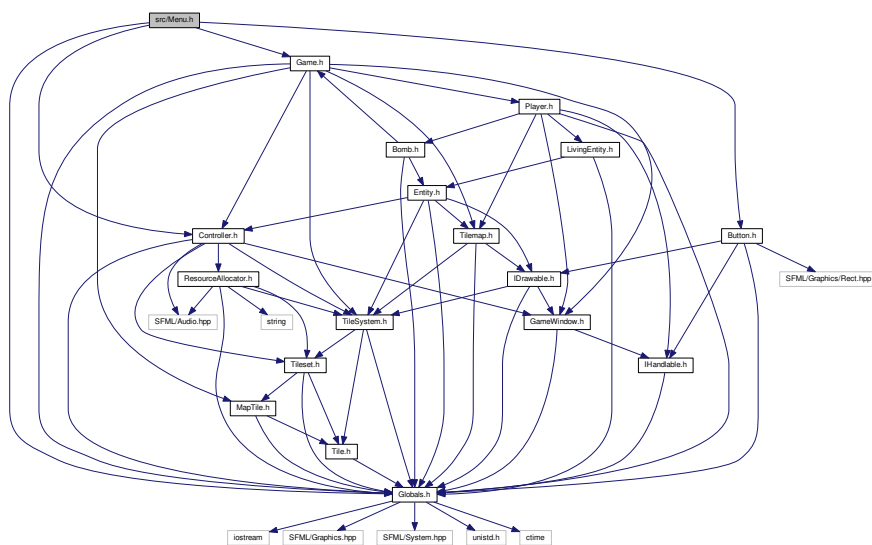
Graphe des dépendances par inclusion de Menu.cpp :



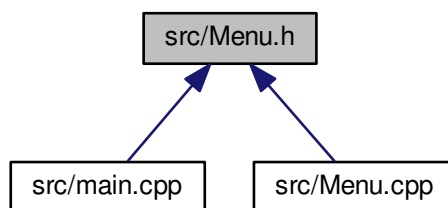
5.23 Référence du fichier src/Menu.h

```
#include "Globals.h"
#include "Controller.h"
#include "Game.h"
#include "Button.h"
```

Graphe des dépendances par inclusion de Menu.h :



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



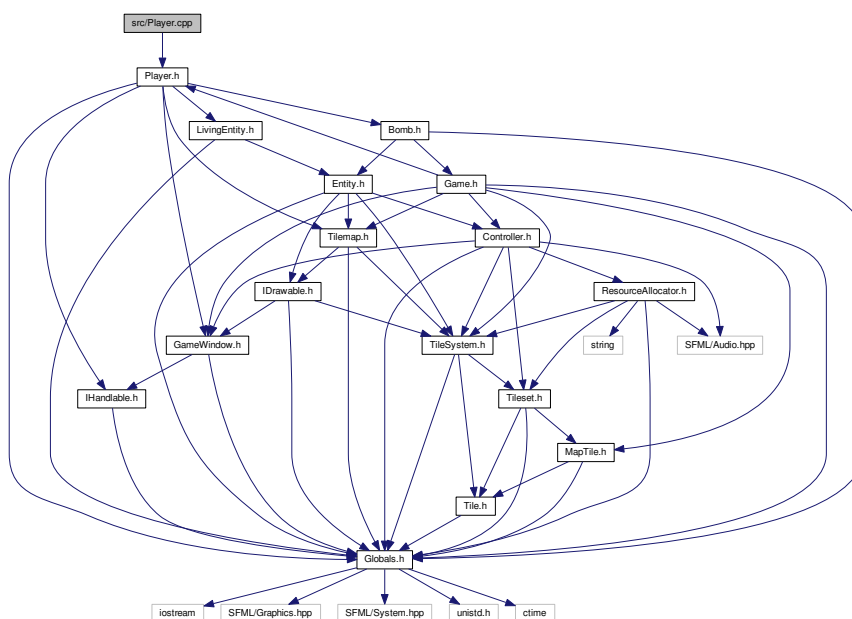
Classes

- class [Menu](#)
contrôleur qui gère le menu principal

5.24 Référence du fichier src/Player.cpp

```
#include "Player.h"
```

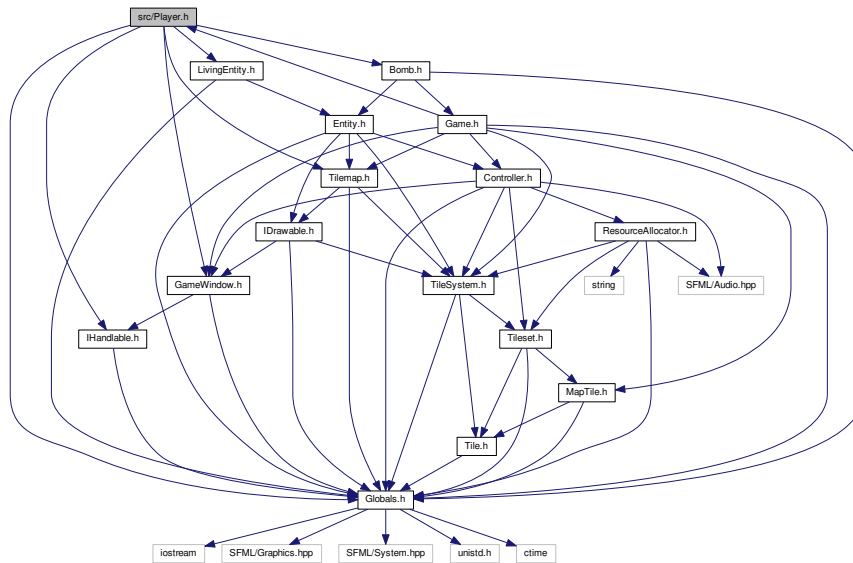
Graphe des dépendances par inclusion de Player.cpp :



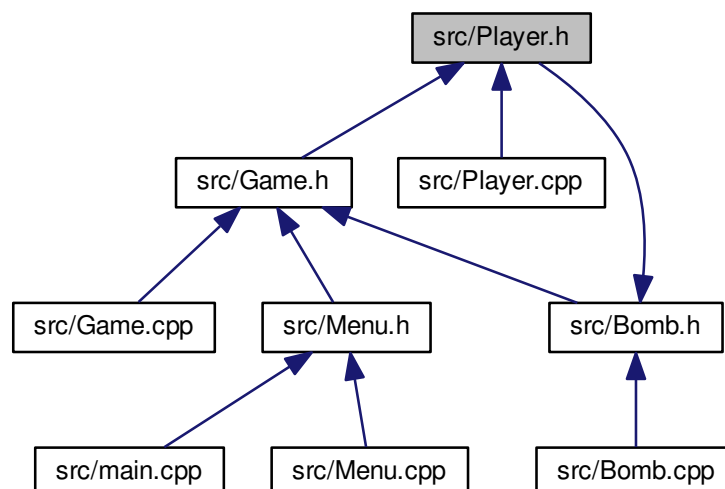
5.25 Référence du fichier src/Player.h

```
#include "Globals.h"
#include "LivingEntity.h"
#include "IHandlable.h"
#include "Tilemap.h"
#include "Bomb.h"
#include "GameWindow.h"
```

Graphe des dépendances par inclusion de Player.h :



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



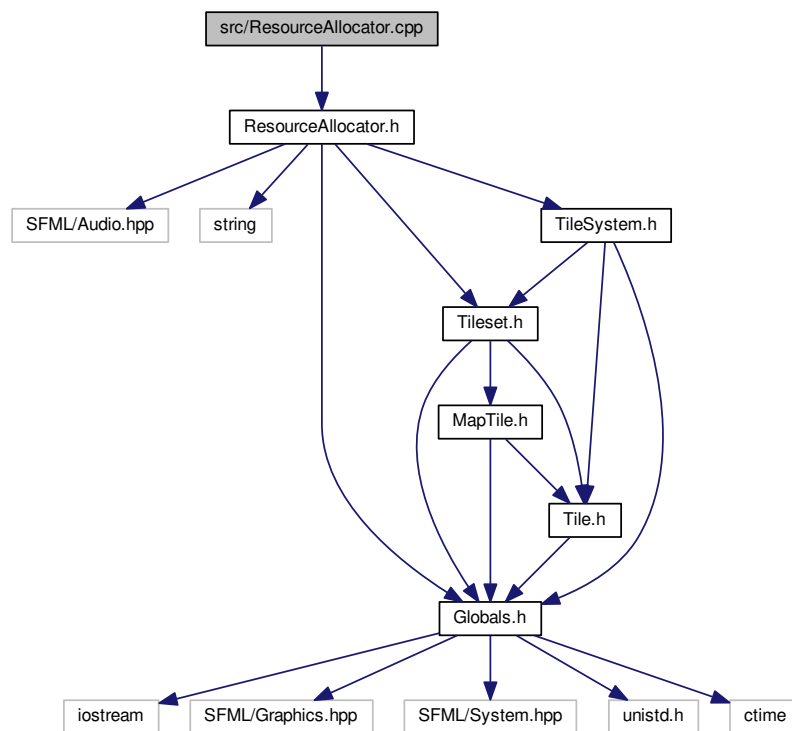
Classes

- class `Player`
un joueur est une entité vivante contrôlable

5.26 Référence du fichier src/ResourceAllocator.cpp

```
#include "ResourceAllocator.h"
```

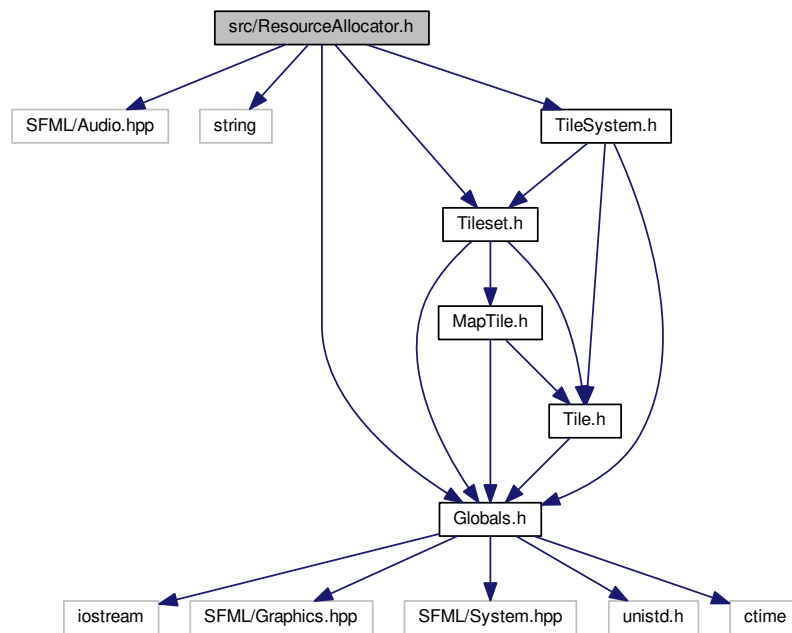
Graphe des dépendances par inclusion de ResourceAllocator.cpp :



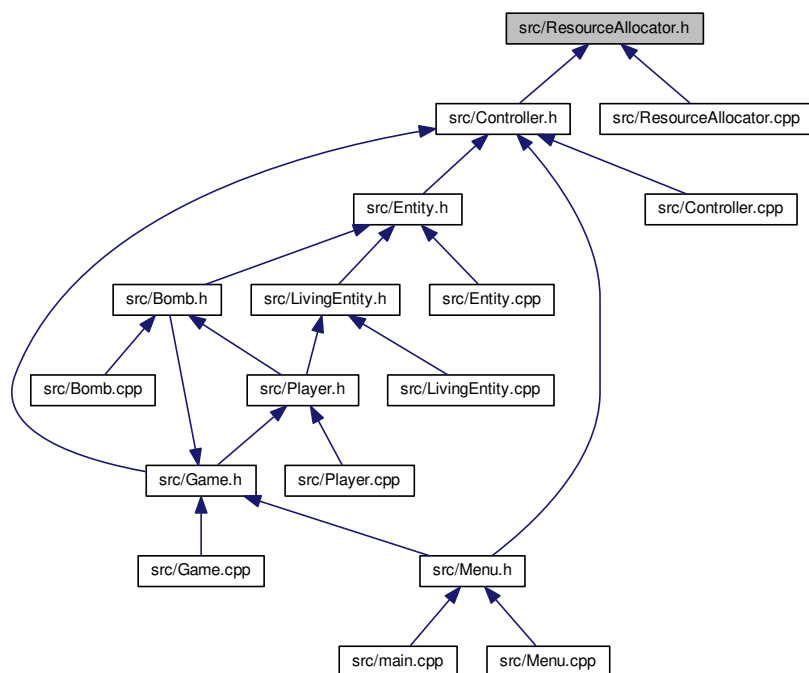
5.27 Référence du fichier src/ResourceAllocator.h

```
#include <SFML/Audio.hpp>
#include <string>
#include "Globals.h"
#include "Tileset.h"
#include "TileSystem.h"
```

Graphe des dépendances par inclusion de ResourceAllocator.h :



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



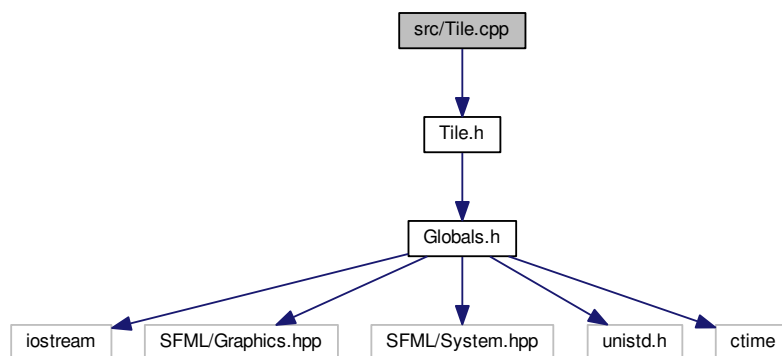
Classes

- class [ResourceAllocator](#)
classe statique chargée de fournir des fonctions d'allocation de ressource

5.28 Référence du fichier src/Tile.cpp

```
#include "Tile.h"
```

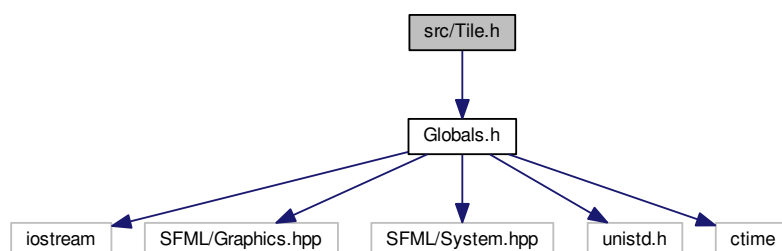
Graphe des dépendances par inclusion de Tile.cpp :



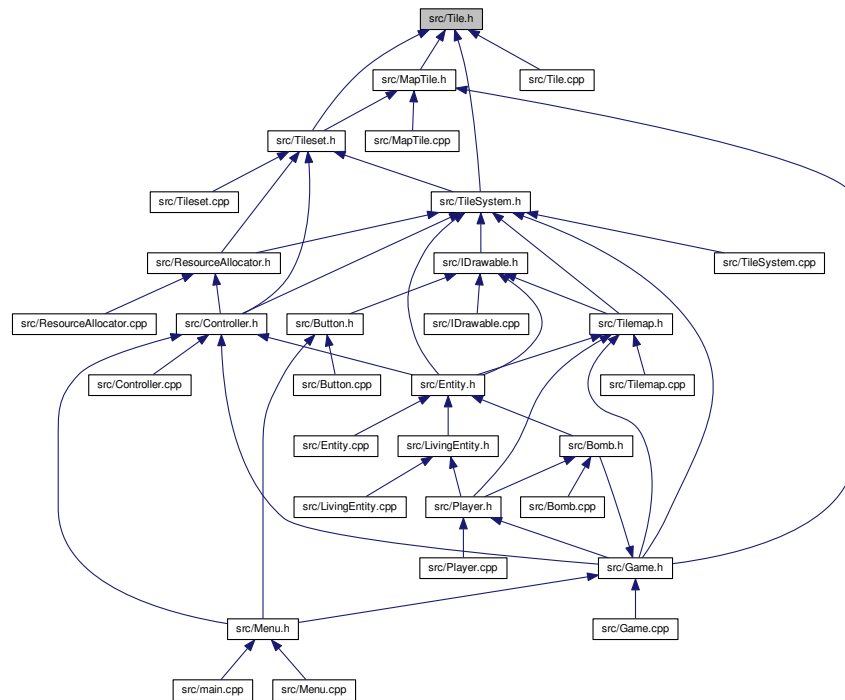
5.29 Référence du fichier src/Tile.h

```
#include "Globals.h"
```

Graphe des dépendances par inclusion de Tile.h :



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



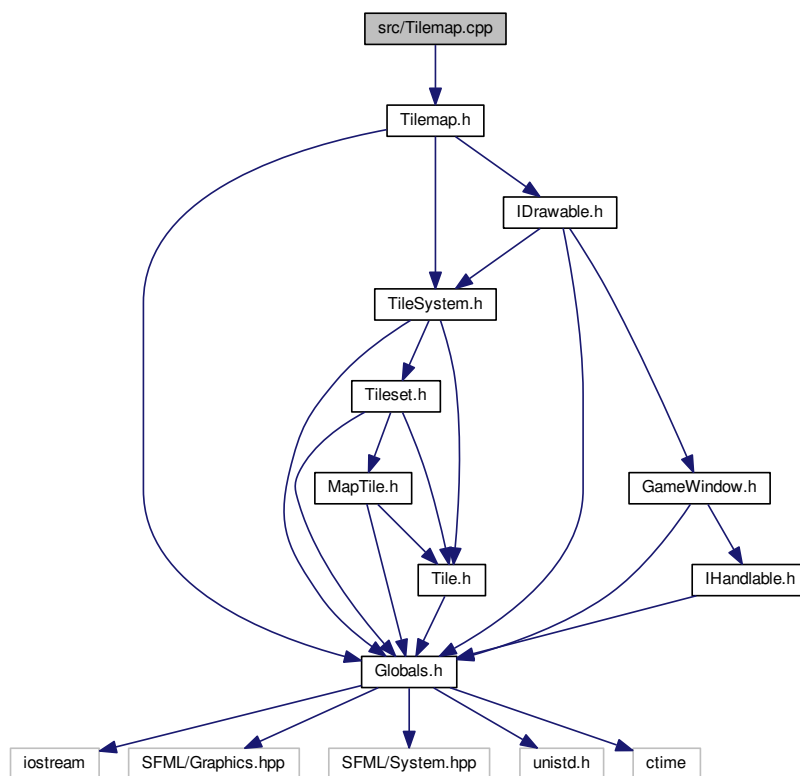
Classes

- class [Tile](#)
un tuile est une brique élémentaire de tout bon jeu 2D de base

5.30 Référence du fichier src/Tilemap.cpp

```
#include "Tilemap.h"
```

Graphe des dépendances par inclusion de Tilemap.cpp :



5.31 Référence du fichier src/Tilemap.h

```

#include "Globals.h"
#include "TileSystem.h"
#include "IDrawable.h"

```

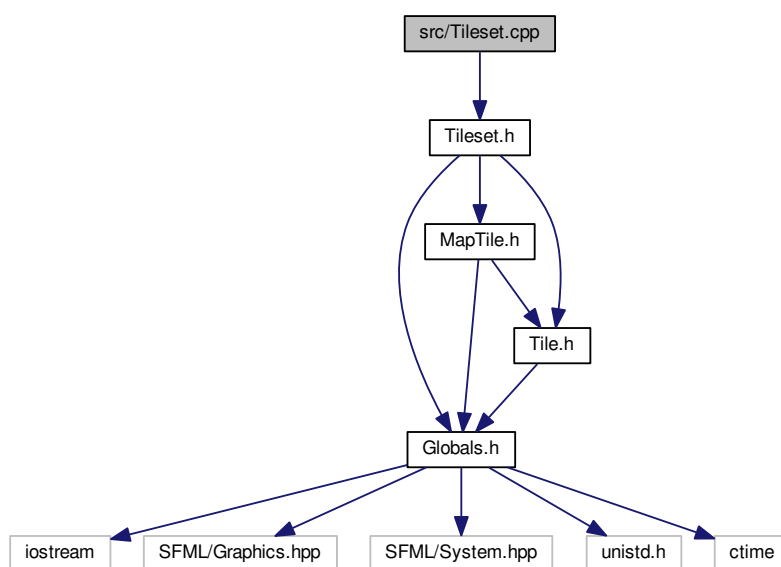

Classes

- class `Tilemap`
une carte est une grille de tuiles qui peut être dessinée

5.32 Référence du fichier src/Tileset.cpp

```
#include "Tileset.h"
```

Graphe des dépendances par inclusion de Tileset.cpp :



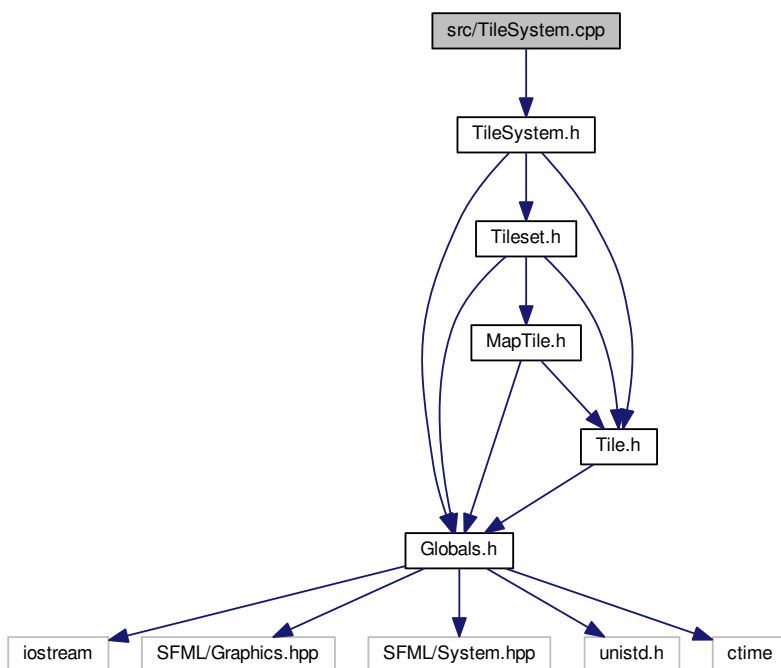
5.33 Référence du fichier src/Tileset.h

```
#include "Globals.h"  
#include "MapTile.h"  
#include "Tile.h"
```


5.34 Référence du fichier src/TileSystem.cpp

```
#include "TileSystem.h"
```

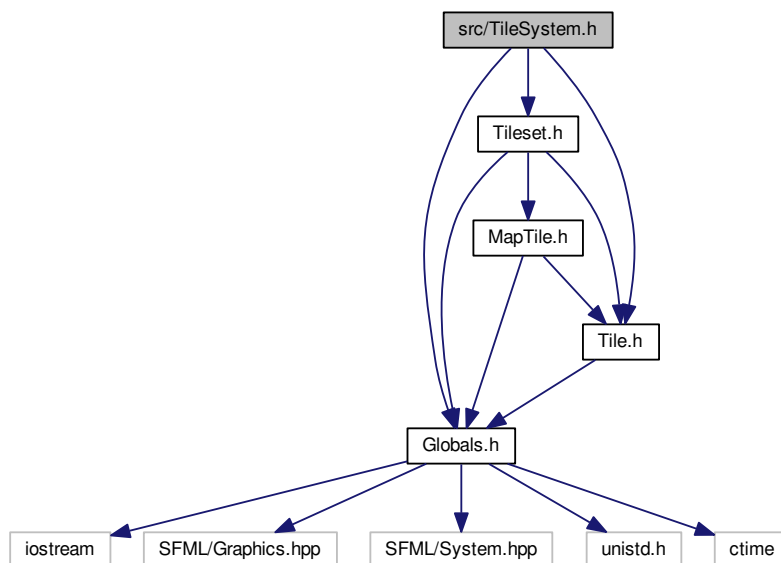
Graphe des dépendances par inclusion de TileSystem.cpp :



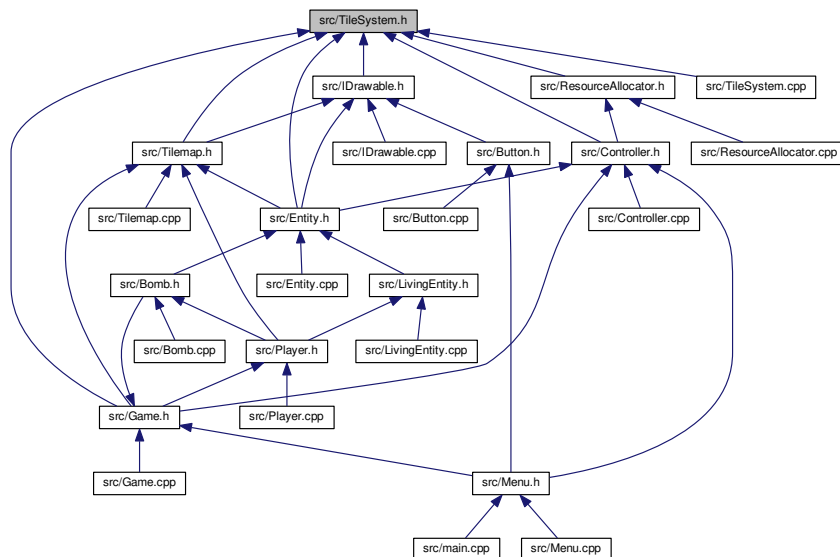
5.35 Référence du fichier src/TileSystem.h

```
#include "Globals.h"  
#include "Tileset.h"  
#include "Tile.h"
```

Graphe des dépendances par inclusion de `TileSystem.h` :



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



Classes

— class `TileSystem`

un système de tuiles lie les données brutes d'un `Tileset` avec tous les `Tile` extraits de ce `Tileset`

Index

- ~Button
 - Button, [14](#)
- ~Controller
 - Controller, [18](#)
- ~Entity
 - Entity, [23](#)
- ~Game
 - Game, [28](#)
- ~Menu
 - Menu, [50](#)
- ~Tile
 - Tile, [62](#)
- ~TileSystem
 - TileSystem, [77](#)
- ~Tilemap
 - Tilemap, [67](#)
- ABS
 - Globals.h, [94](#)
- ANIMATED
 - Globals.h, [95](#)
- addSprite
 - Tile, [63](#)
- allocateFont
 - ResourceAllocator, [59](#)
- allocateMusic
 - ResourceAllocator, [59](#)
- allocateTexture
 - ResourceAllocator, [59](#)
- allocateTileSystem
 - ResourceAllocator, [60](#)
- allocateTileset
 - ResourceAllocator, [60](#)
- attemptDestruction
 - Tilemap, [68](#)
- BOTTOM
 - Globals.h, [95](#)
- background
 - Menu, [51](#)
- Bomb, [7](#)
 - Bomb, [10](#)
 - draw, [10](#)
 - getPlayer, [11](#)
 - m_blastRadius, [11](#)
 - m_duration, [11](#)
 - m_maxSize, [11](#)
 - m_player, [11](#)
 - updateState, [11](#)
- Button, [12](#)
- ~Button, [14](#)
- Button, [14](#)
- callback, [14](#)
- draw, [14](#)
- m_callback, [15](#)
- m_text, [15](#)
- manageEvents, [15](#)
- callback
 - Button, [14](#)
- Controller, [15](#)
 - ~Controller, [18](#)
 - Controller, [18](#)
 - m_fonts, [19](#)
 - m_musics, [19](#)
 - m_textures, [19](#)
 - m_tilesets, [19](#)
 - m_tilesystems, [19](#)
 - m_window, [19](#)
 - manageEvents, [19](#)
 - notifyUpdate, [19](#)
 - start, [19](#)
- create
 - GameWindow, [33](#)
- createTile
 - Tileset, [73](#)
- DEBUG_MODE
 - Globals.h, [94](#)
- DEFAULT_TILE_SIZE
 - Globals.h, [94](#)
- DEFAULT
 - Globals.h, [95](#)
- displayHelp
 - main.cpp, [101](#)
- draw
 - Bomb, [10](#)
 - Button, [14](#)
 - IDrawable, [37](#)
 - LivingEntity, [42](#)
 - Tilemap, [68](#)
- dropBomb
 - Player, [56](#)
- Entity, [20](#)
 - ~Entity, [23](#)
 - Entity, [23](#)
 - getHealth, [23](#)
 - getPosition, [24](#)
 - m_bBoxes, [24](#)

- m_center, [24](#)
 - m_health, [24](#)
 - m_maxHealth, [24](#)
 - m_position, [25](#)
 - m_tick, [25](#)
 - m_timeAlive, [25](#)
 - m_watcher, [25](#)
 - updateState, [24](#)
- FONT_ERROR
 - Globals.h, [94](#)
- G_AUTHORS
 - Globals.h, [94](#)
- G_VERSION
 - Globals.h, [94](#)
- GENERIC_ERROR
 - Globals.h, [95](#)
- Game, [25](#)
 - ~Game, [28](#)
 - Game, [28](#)
 - getEntities, [29](#)
 - getPlayer, [29](#)
 - hasWon, [30](#)
 - m_backgroundMap, [30](#)
 - m_entities, [30](#)
 - m_isPlaying, [30](#)
 - m_physicalMap, [30](#)
 - m_player1, [30](#)
 - m_player2, [30](#)
 - m_timer, [30](#)
 - m_view, [30](#)
 - manageEvents, [29](#)
 - notifyUpdate, [29](#)
 - start, [29](#)
- GameWindow, [31](#)
 - create, [33](#)
 - getHeight, [33](#)
 - getTileSize, [33](#)
 - getWidth, [33](#)
 - m_height, [34](#)
 - m_tileSize, [34](#)
 - m_toRectifyRatio, [34](#)
 - m_width, [34](#)
 - manageEvents, [34](#)
 - rectifyRatio, [34](#)
- getCols
 - Tileset, [73](#)
- getDisplayHeight
 - Tileset, [74](#)
- getDisplayWidth
 - Tileset, [74](#)
- getEntities
 - Game, [29](#)
- getHealth
 - Entity, [23](#)
- getHeight
 - GameWindow, [33](#)
 - Tilemap, [68](#)
- Tileset, [74](#)
- getMetadata
 - Tilemap, [68](#)
- getPlayer
 - Bomb, [11](#)
 - Game, [29](#)
- getPosition
 - Entity, [24](#)
- getRows
 - Tileset, [74](#)
- getSize
 - TileSystem, [77](#)
- getSprite
 - Tile, [63](#)
 - Tilemap, [68](#), [69](#)
- getSpriteCount
 - Tile, [64](#)
- getTexture
 - Tileset, [74](#)
- getTile
 - TileSystem, [78](#)
 - Tilemap, [69](#)
- getTileSize
 - GameWindow, [33](#)
- getTileSystem
 - Tilemap, [69](#)
- getTs
 - TileSystem, [78](#)
- getType
 - Tile, [64](#)
- getWidth
 - GameWindow, [33](#)
 - Tilemap, [69](#)
 - Tileset, [74](#)
- Globals.h
 - ABS, [94](#)
 - ANIMATED, [95](#)
 - BOTTOM, [95](#)
 - DEBUG_MODE, [94](#)
 - DEFAULT_TILE_SIZE, [94](#)
 - DEFAULT, [95](#)
 - FONT_ERROR, [94](#)
 - G_AUTHORS, [94](#)
 - G_VERSION, [94](#)
 - GENERIC_ERROR, [95](#)
 - LEFT, [95](#)
 - MUSIC_ERROR, [95](#)
 - Orientation, [95](#)
 - RANDOMIZED, [95](#)
 - RIGHT, [95](#)
 - SGN, [95](#)
 - SPEED_FACTOR, [95](#)
 - TEXTURE_ERROR, [95](#)
 - TILESET_ERROR, [95](#)
 - TOP, [95](#)
 - TRANSITIONAL, [95](#)
 - TileType, [95](#)
- hasWon

- Game, [30](#)
- hit
 - Player, [57](#)
- IDrawable, [35](#)
 - draw, [37](#)
 - IDrawable, [37](#)
 - m_tilesys, [37](#)
- IHandlable, [37](#)
 - manageEvents, [39](#)
- isBreakable
 - MapTile, [46](#)
- isCollidable
 - MapTile, [46](#)
- isOnCoord
 - Player, [57](#)
- LEFT
 - Globals.h, [95](#)
- LivingEntity, [39](#)
 - draw, [42](#)
 - LivingEntity, [42](#)
 - m_orientation, [43](#)
 - m_speed, [43](#)
 - move, [42](#)
 - setSpeed, [43](#)
 - updateState, [43](#)
- logo
 - Menu, [51](#)
- m_bBoxes
 - Entity, [24](#)
- m_backgroundMap
 - Game, [30](#)
- m_blastPower
 - Player, [57](#)
- m_blastRadius
 - Bomb, [11](#)
- m_bombCount
 - Player, [57](#)
- m_bombHasBeenDropped
 - Player, [58](#)
- m_bombInventory
 - Player, [58](#)
- m_bombTilesys
 - Player, [58](#)
- m_breakable
 - MapTile, [47](#)
- m_callback
 - Button, [15](#)
- m_center
 - Entity, [24](#)
- m_collidable
 - MapTile, [47](#)
- m_cols
 - Tileset, [75](#)
- m_controls
 - Player, [58](#)
- m_displayHeight
 - Tileset, [75](#)
- m_displayWidth
 - Tileset, [75](#)
- m_duration
 - Bomb, [11](#)
- m_entities
 - Game, [30](#)
- m_fonts
 - Controller, [19](#)
- m_health
 - Entity, [24](#)
- m_height
 - GameWindow, [34](#)
 - Tilemap, [70](#)
 - Tileset, [75](#)
- m_isPlaying
 - Game, [30](#)
- m_map
 - Tilemap, [70](#)
- m_maxHealth
 - Entity, [24](#)
- m_maxSize
 - Bomb, [11](#)
- m_metadata
 - Tilemap, [70](#)
- m_musics
 - Controller, [19](#)
- m_orientation
 - LivingEntity, [43](#)
- m_physicalMap
 - Game, [30](#)
- m_player
 - Bomb, [11](#)
- m_player1
 - Game, [30](#)
- m_player2
 - Game, [30](#)
- m_position
 - Entity, [25](#)
- m_rows
 - Tileset, [75](#)
- m_speed
 - LivingEntity, [43](#)
- m_sprites
 - Tile, [64](#)
- m_text
 - Button, [15](#)
- m_texture
 - Tileset, [75](#)
- m_textures
 - Controller, [19](#)
- m_tick
 - Entity, [25](#)
- m_tileSize
 - GameWindow, [34](#)
- m_tilesList
 - TileSystem, [79](#)
- m_tilesets

- Controller, 19
- m_tilesys
 - IDrawable, 37
- m_tilesystems
 - Controller, 19
- m_timeAlive
 - Entity, 25
- m_timer
 - Game, 30
- m_toRectifyRatio
 - GameWindow, 34
- m_ts
 - TileSystem, 79
- m_type
 - Tile, 64
- m_view
 - Game, 30
- m_watcher
 - Entity, 25
 - Tilemap, 70
- m_width
 - GameWindow, 34
 - Tilemap, 71
 - Tileset, 75
- m_window
 - Controller, 19
- MUSIC_ERROR
 - Globals.h, 95
- main
 - main.cpp, 101
- main.cpp
 - displayHelp, 101
 - main, 101
- manageEvents
 - Button, 15
 - Controller, 19
 - Game, 29
 - GameWindow, 34
 - IHandlable, 39
 - Menu, 50
 - Player, 57
- MapTile, 44
 - isBreakable, 46
 - isCollidable, 46
 - m_breakable, 47
 - m_collidable, 47
 - MapTile, 46
 - setBreakable, 46
 - setCollidable, 47
- Menu, 47
 - ~Menu, 50
 - background, 51
 - logo, 51
 - manageEvents, 50
 - Menu, 50
 - notifyUpdate, 50
 - playButton, 51
 - quitButton, 51
 - start, 51
- move
 - LivingEntity, 42
- nStream, 52
 - overflow, 53
- notifyExplosion
 - Player, 57
- notifyUpdate
 - Controller, 19
 - Game, 29
 - Menu, 50
- Orientation
 - Globals.h, 95
- overflow
 - nStream, 53
- playButton
 - Menu, 51
- Player, 53
 - dropBomb, 56
 - hit, 57
 - isOnCoord, 57
 - m_blastPower, 57
 - m_bombCount, 57
 - m_bombHasBeenDropped, 58
 - m_bombInventory, 58
 - m_bombTilesys, 58
 - m_controls, 58
 - manageEvents, 57
 - notifyExplosion, 57
 - Player, 56
- quitButton
 - Menu, 51
- RANDOMIZED
 - Globals.h, 95
- RIGHT
 - Globals.h, 95
- rectifyRatio
 - GameWindow, 34
- registerTile
 - TileSystem, 78
- ResourceAllocator, 58
 - allocateFont, 59
 - allocateMusic, 59
 - allocateTexture, 59
 - allocateTileSystem, 60
 - allocateTileset, 60
- SGN
 - Globals.h, 95
- SPEED_FACTOR
 - Globals.h, 95
- setBreakable
 - MapTile, 46
- setCollidable
 - MapTile, 47

- setMap
 - Tilemap, 69
- setSpeed
 - LivingEntity, 43
- setTileIndex
 - Tilemap, 70
- src/Bomb.cpp, 81
- src/Bomb.h, 81
- src/Button.cpp, 83
- src/Button.h, 84
- src/Controller.cpp, 85
- src/Controller.h, 86
- src/Entity.cpp, 88
- src/Entity.h, 88
- src/Game.cpp, 90
- src/Game.h, 90
- src/GameWindow.cpp, 92
- src/GameWindow.h, 92
- src/Globals.h, 93
- src/IDrawable.cpp, 96
- src/IDrawable.h, 96
- src/IHandlable.h, 98
- src/LivingEntity.cpp, 99
- src/LivingEntity.h, 99
- src/MapTile.cpp, 102
- src/MapTile.h, 102
- src/Menu.cpp, 104
- src/Menu.h, 104
- src/Player.cpp, 105
- src/Player.h, 106
- src/ResourceAllocator.cpp, 107
- src/ResourceAllocator.h, 107
- src/Tile.cpp, 109
- src/Tile.h, 109
- src/TileSystem.cpp, 115
- src/TileSystem.h, 115
- src/Tilemap.cpp, 110
- src/Tilemap.h, 111
- src/Tileset.cpp, 113
- src/Tileset.h, 113
- src/main.cpp, 101
- start
 - Controller, 19
 - Game, 29
 - Menu, 51
- TEXTURE_ERROR
 - Globals.h, 95
- TILESET_ERROR
 - Globals.h, 95
- TOP
 - Globals.h, 95
- TRANSITIONAL
 - Globals.h, 95
- Tile, 60
 - ~Tile, 62
 - addSprite, 63
 - getSprite, 63
 - getSpriteCount, 64
 - getType, 64
 - m_sprites, 64
 - m_type, 64
 - Tile, 62
- TileSystem, 76
 - ~TileSystem, 77
 - getSize, 77
 - getTile, 78
 - getTs, 78
 - m_tilesList, 79
 - m_ts, 79
 - registerTile, 78
 - TileSystem, 77
- TileType
 - Globals.h, 95
- Tilemap, 65
 - ~Tilemap, 67
 - attemptDestruction, 68
 - draw, 68
 - getHeight, 68
 - getMetadata, 68
 - getSprite, 68, 69
 - getTile, 69
 - getTileSystem, 69
 - getWidth, 69
 - m_height, 70
 - m_map, 70
 - m_metadata, 70
 - m_watcher, 70
 - m_width, 71
 - setMap, 69
 - setTileIndex, 70
 - Tilemap, 67
 - toTileCoord, 70
 - updateState, 70
- Tileset, 71
 - createTile, 73
 - getCols, 73
 - getDisplayHeight, 74
 - getDisplayWidth, 74
 - getHeight, 74
 - getRows, 74
 - getTexture, 74
 - getWidth, 74
 - m_cols, 75
 - m_displayHeight, 75
 - m_displayWidth, 75
 - m_height, 75
 - m_rows, 75
 - m_texture, 75
 - m_width, 75
 - Tileset, 72
- toTileCoord
 - Tilemap, 70
- updateState
 - Bomb, 11
 - Entity, 24
 - LivingEntity, 43

Tilemap, [70](#)