

Projet Sokoban

1 Index des espaces de nommage	1
1.1 Liste des espaces de nommage	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 espaces de nommage	7
4.1 Référence de l'espace de nommage iostream	7
4.1.1 Description détaillée	7
4.2 Référence de l'espace de nommage pragmaOnce	7
4.2.1 Description détaillée	7
4.3 Référence de l'espace de nommage SDL2	7
4.3.1 Description détaillée	7
4.4 Référence de l'espace de nommage SDL2image	8
4.4.1 Description détaillée	8
4.5 Référence de l'espace de nommage stdio	8
4.5.1 Description détaillée	8
5 Documentation des classes	9
5.1 Référence de la classe Game	9
5.1.1 Description détaillée	10
5.1.2 Documentation des constructeurs et destructeur	10
5.1.2.1 Game()	10
5.1.2.2 ~Game()	10
5.1.3 Documentation des fonctions membres	11
5.1.3.1 clean()	11
5.1.3.2 handleEvents()	11
5.1.3.3 init()	12
5.1.3.4 render()	13
5.1.3.5 running()	13
5.1.3.6 update()	14
5.1.4 Documentation des données membres	14
5.1.4.1 event	14
5.1.4.2 renderer	14
5.2 Référence de la classe GameObject	15
5.2.1 Description détaillée	15
5.2.2 Documentation des constructeurs et destructeur	16
5.2.2.1 GameObject()	16
5.2.2.2 ~GameObject()	16
5.2.3 Documentation des fonctions membres	16
5.2.3.1 getPosX()	17

5.2.3.2	getY()	17
5.2.3.3	Render()	17
5.2.3.4	setX()	18
5.2.3.5	setY()	18
5.2.3.6	Update()	19
5.3	Référence de la classe Map	19
5.3.1	Description détaillée	20
5.3.2	Documentation des constructeurs et destructeur	20
5.3.2.1	Map()	20
5.3.2.2	~Map()	21
5.3.3	Documentation des fonctions membres	21
5.3.3.1	DrawMap()	21
5.3.3.2	getMap()	22
5.3.3.3	LoadMap()	22
5.4	Référence de la classe TextureManag	22
5.4.1	Description détaillée	23
5.4.2	Documentation des constructeurs et destructeur	23
5.4.2.1	TextureManag()	24
5.4.2.2	~TextureManag()	24
5.4.3	Documentation des fonctions membres	24
5.4.3.1	Draw()	24
5.4.3.2	loadTexture()	24
6	Documentation des fichiers	27
6.1	Référence du fichier include/Game.h	27
6.2	Game.h	28
6.3	Référence du fichier include/GameObject.h	28
6.4	GameObject.h	29
6.5	Référence du fichier include/Map.h	29
6.6	Map.h	30
6.7	Référence du fichier include/TextureManag.h	30
6.8	TextureManag.h	31
6.9	Référence du fichier main.cpp	32
6.9.1	Documentation des fonctions	32
6.9.1.1	main()	32
6.9.2	Documentation des variables	33
6.9.2.1	game	33
6.10	Référence du fichier src/Game.cpp	33
6.10.1	Documentation des variables	34
6.10.1.1	box1	34
6.10.1.2	boxOk	34
6.10.1.3	boxXleft	34

6.10.1.4 boxXright	34
6.10.1.5 boxYdown	34
6.10.1.6 boxYup	35
6.10.1.7 goal	35
6.10.1.8 mape	35
6.10.1.9 player	35
6.11 Référence du fichier src/GameObject.cpp	35
6.12 Référence du fichier src/Map.cpp	36
6.12.1 Documentation des variables	36
6.12.1.1 lvl1	36
6.13 Référence du fichier src/TextureManag.cpp	37
Index	39

Chapitre 1

Index des espaces de nommage

1.1 Liste des espaces de nommage

Liste de tous les espaces de nommage avec une brève description:

iostream	Pour inclure des objets de flux globaux dans c++	7
pragmaOnce	Ce package permet d'utiliser les classes une seule fois pour eviter la repition	7
SDL2	Une bibliotheque utilisee en programmation multimedia	7
SDL2image	Il est construit sur le framework OpenGL et utilise SDL pour la lectures des images	8
stdio	Pour Standard Input Output Header ou En-tete Standard d'Entree/Sortie	8

Chapitre 2

Index des classes

2.1 Liste des classes

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

Game	Une class du jeu	9
GameObject	Classe d'Objet	15
Map	Classe de la carte	19
TextureManag	Classe de la texture	22

Chapitre 3

Index des fichiers

3.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

main.cpp	32
include/Game.h	27
include/GameObject.h	28
include/Map.h	29
include/TextureManag.h	30
src/Game.cpp	33
src/GameObject.cpp	35
src/Map.cpp	36
src/TextureManag.cpp	37

Chapitre 4

Documentation des espaces de nommage

4.1 Référence de l'espace de nommage `iostream`

pour inclure des objets de flux globaux dans c++.

4.1.1 Description détaillée

pour inclure des objets de flux globaux dans c++.

4.2 Référence de l'espace de nommage `pragmaOnce`

ce package permet d'utiliser les classes une seule fois pour eviter la repition.

4.2.1 Description détaillée

ce package permet d'utiliser les classes une seule fois pour eviter la repition.

/*!

4.3 Référence de l'espace de nommage `SDL2`

une bibliotheque utilisee en programmation multimedia.

4.3.1 Description détaillée

une bibliotheque utilisee en programmation multimedia.

4.4 Référence de l'espace de nommage SDL2image

Il est construit sur le framework OpenGL et utilise SDL pour la lectures des images.

4.4.1 Description détaillée

Il est construit sur le framework OpenGL et utilise SDL pour la lectures des images.

4.5 Référence de l'espace de nommage stdio

pour Standard Input Output Header ou En-tete Standard d'Entree/Sortie

4.5.1 Description détaillée

pour Standard Input Output Header ou En-tete Standard d'Entree/Sortie

Chapitre 5

Documentation des classes

5.1 Référence de la classe Game

une class du jeu.

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

Graphe de collaboration de Game:

Game
+ static SDL_Renderer * renderer + static SDL_Event event
+ Game() + virtual ~Game() + void init(const char *title, int xpos, int ypos, int widthpos, int heightpos, bool fullscreen) + void handleEvents() + void update() + void render() + void clean() + bool running()

Fonctions membres publiques

- [Game](#) ()
un constructeur
 - virtual [~Game](#) ()
-

un destructeur

- void `init` (const char *title, int xpos, int ypos, int widthpos, int heightpos, bool fullscreen)
initialisation.
- void `handleEvents` ()
Gerer les evenements.
- void `update` ()
Rafraichir l'image.
- void `render` ()
Dessiner.
- void `clean` ()
Nettoyer.
- bool `running` ()
booleen

Attributs publics statiques

- static SDL_Renderer * `renderer` = nullptr
- static SDL_Event `event`

5.1.1 Description détaillée

une class du jeu.

l'utilisation de cette a etait necessaire pour afficher et dessiner et rafraichir les dessins et la fenetre.

5.1.2 Documentation des constructeurs et destructeur

5.1.2.1 Game()

```
Game::Game ( )
```

un constructeur

Le constructeur est la fonction membre appelee automatiquement lors de la creation d'un objet (en statique ou en dynamique). Cette fonction membre est la premiere fonction membre a etre executee, il s'agit donc d'une fonction permettant l'initialisation des variables.

5.1.2.2 ~Game()

```
Game::~~Game ( ) [virtual]
```

un destructeur

Le destructeurs sont en quelque sorte au constructeur ce que la mort est a la vie, c'est-a-dire qu'il s'agit d'une fonction membre qui intervient automatiquement lors de la destruction d'un objet.

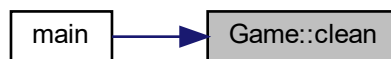
5.1.3 Documentation des fonctions membres

5.1.3.1 clean()

```
void Game::clean ( )
```

Nettoyer.

Nettoyer le contenu actuel de la fenetre Voici le graphe des appelants de cette fonction :

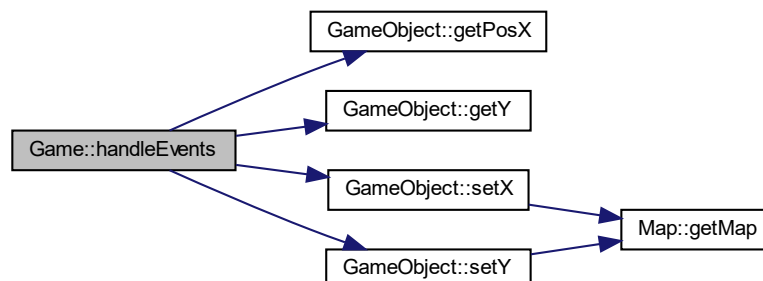


5.1.3.2 handleEvents()

```
void Game::handleEvents ( )
```

Gerer les evenements.

Nous appelons evenement toute action exterieure a notre programme et qui peut avoir un effet sur lui. L'appui sur une touche du clavier, le déplacement de la souris, le redimensionnement d'une fenetre, et meme une demande de fermeture du programme sont des evenements. Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.1.3.3 init()

```
void Game::init (
    const char * title,
    int xpos,
    int ypos,
    int widhtpos,
    int heightpos,
    bool fullscreen )
```

initialisation.

on doit initialiser, pour cela nous recourons a cette fonction d'initialisation ou on peut saisir le chemin de l'image et ces references

Paramètres

<i>title</i>	le lien de l'image
<i>xpos</i>	la position x
<i>ypos</i>	la position y
<i>widhtpos</i>	la largeur
<i>heightpos</i>	la taille
<i>fullscreen</i>	plein ecran

Voici le graphe des appelants de cette fonction :

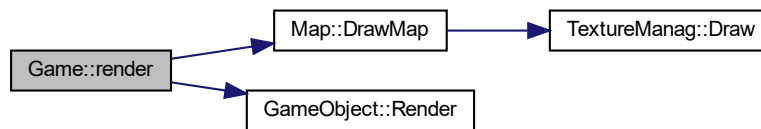


5.1.3.4 render()

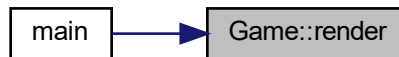
```
void Game::render ( )
```

Dessiner.

Dessiner ce qu'on a a dessiner Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.1.3.5 running()

```
Game::running ( ) [inline]
```

booleen

Renvoie

si la fenetre et toujours ON

Voici le graphe des appelants de cette fonction :

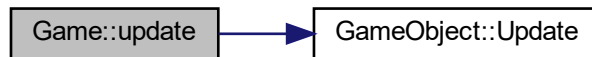


5.1.3.6 update()

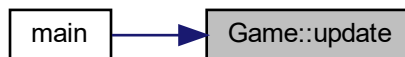
```
void Game::update ( )
```

Rafraichir l'image.

Mettre a jour la fenetre Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.1.4 Documentation des données membres

5.1.4.1 event

```
SDL_Event Game::event [static]
```

Declaration pour l'evenement

5.1.4.2 renderer

```
SDL_Renderer * Game::renderer = nullptr [static]
```

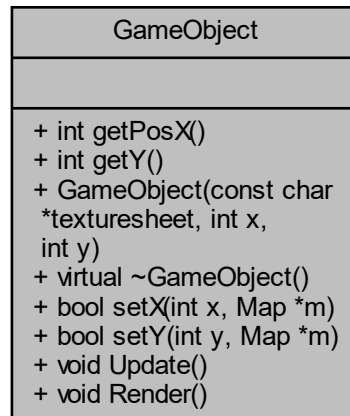
Declaration pour Renouveler l'affichage

5.2 Référence de la classe GameObject

classe d'Objet

```
#include <GameObject.h>
```

Graphe de collaboration de GameObject:



Fonctions membres publiques

- int [getPosX](#) ()
un getteur X
- int [getY](#) ()
un getteur Y
- [GameObject](#) (const char *texturesheet, int x, int y)
un constructeur
- virtual [~GameObject](#) ()
un destructeur
- bool [setX](#) (int x, [Map](#) *m)
- bool [setY](#) (int y, [Map](#) *m)
- void [Update](#) ()
Rafraichir l'image.
- void [Render](#) ()
Dessiner.

5.2.1 Description détaillée

classe d'Objet

cette classe sert a la modification et la manipulation du joueur ou objet par ex : ces position,taille...

5.2.2 Documentation des constructeurs et destructeur

5.2.2.1 GameObject()

```
GameObject::GameObject (
    const char * texturesheet,
    int x,
    int y )
```

un constructeur

Le constructeur est la fonction membre appelee automatiquement lors de la creation d'un objet (en statique ou en dynamique). Cette fonction membre est la premiere fonction membre a etre executee, il s'agit donc d'une fonction permettant l'initialisation des variables. Avec 3 para

Paramètres

<i>texturesheet</i>	le lien de l'image
<i>x</i>	position x
<i>y</i>	position y

Voici le graphe d'appel pour cette fonction :



5.2.2.2 ~GameObject()

```
GameObject::~~GameObject ( ) [virtual]
```

un destructeur

Le destructeurs sont en quelque sorte au constructeur ce que la mort est a la vie, c'est-a-dire qu'il s'agit d'une fonction membre qui intervient automatiquement lors de la destruction d'un objet.

5.2.3 Documentation des fonctions membres

5.2.3.1 getPosX()

```
int GameObject::getPosX ( )
```

un getteur X

Pour avoir la position du x Voici le graphe des appelants de cette fonction :



5.2.3.2 getY()

```
int GameObject::getY ( )
```

un getteur Y

Pour avoir la position du y Voici le graphe des appelants de cette fonction :

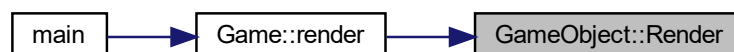


5.2.3.3 Render()

```
void GameObject::Render ( )
```

Dessiner.

Dessiner ce qu'on a a dessiner Voici le graphe des appelants de cette fonction :



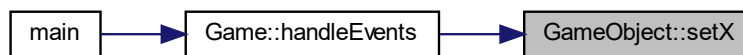
5.2.3.4 setX()

```
bool GameObject::setX (
    int x,
    Map * m )
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



5.2.3.5 setY()

```
bool GameObject::setY (
    int y,
    Map * m )
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

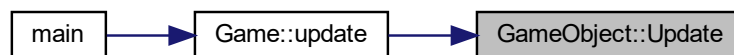


5.2.3.6 Update()

```
void GameObject::Update ( )
```

Rafraichir l'image.

Mettre a jour la fenetre Voici le graphe des appelants de cette fonction :

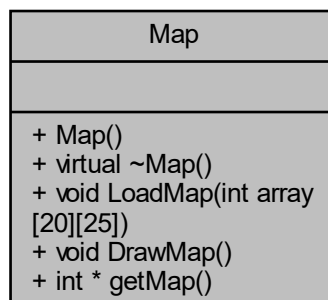


5.3 Référence de la classe Map

classe de la carte

```
#include <Map.h>
```

Graphe de collaboration de Map:



Fonctions membres publiques

- `Map ()`
- `virtual ~Map ()`
un destructeur
- `void LoadMap (int array[20][25])`
Upload la carte.
- `void DrawMap ()`
Dessiner.
- `int * getMap ()`
un getteur `Map`

5.3.1 Description détaillée

classe de la carte

cette class sert a la modification et la manipulation de la carte

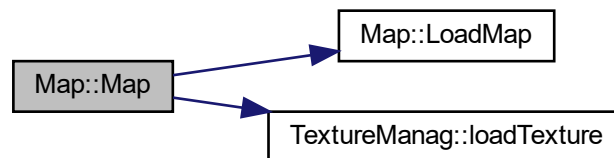
5.3.2 Documentation des constructeurs et destructeur

5.3.2.1 Map()

```
Map::Map ( )
```

un constructeur

Le constructeur est la fonction membre appelee automatiquement lors de la creation d'un objet (en statique ou en dynamique). Cette fonction membre est la premiere fonction membre a etre executee, il s'agit donc d'une fonction permettant l'initialisation des variables. Voici le graphe d'appel pour cette fonction :



5.3.2.2 ~Map()

```
Map::~Map ( ) [virtual]
```

un destructeur

Le destructeurs sont en quelque sorte au constructeur ce que la mort est à la vie, c'est-à-dire qu'il s'agit d'une fonction membre qui intervient automatiquement lors de la destruction d'un objet.

5.3.3 Documentation des fonctions membres

5.3.3.1 DrawMap()

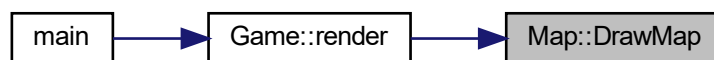
```
void Map::DrawMap ( )
```

Dessiner.

Dessiner ce qu'on a dessiné Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

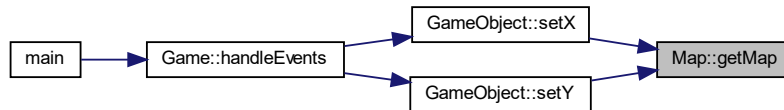


5.3.3.2 getMap()

```
int * Map::getMap ( )
```

un getteur [Map](#)

Pour avoir les references de la carte et l'utiliser dans [GameObject](#) pour limiter la circulation du joueur Voici le graphe des appelants de cette fonction :



5.3.3.3 LoadMap()

```
void Map::LoadMap (
    int array[20][25] )
```

Upload la carte.

l'utilisation de loadMap c'est pour la creation de la carte.

Paramètres

<code>array</code>	on a utilise un tableau pour la creation de la carte
--------------------	--

Voici le graphe des appelants de cette fonction :

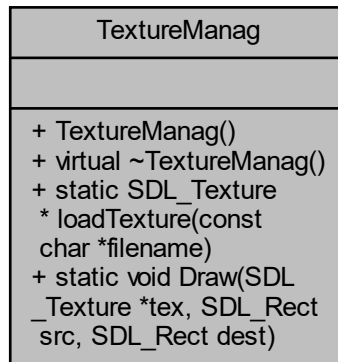


5.4 Référence de la classe TextureManag

classe de la texture

```
#include <TextureManag.h>
```

Graphe de collaboration de TextureManag:



Fonctions membres publiques

- [TextureManag](#) ()
un constructeur
- virtual [~TextureManag](#) ()
un destructeur

Fonctions membres publiques statiques

- static SDL_Texture * [loadTexture](#) (const char *filename)
pour telecharger le fichier
- static void [Draw](#) (SDL_Texture *tex, SDL_Rect src, SDL_Rect dest)

5.4.1 Description détaillée

classe de la texture

cette class sert a telecharger le fichier(image) et la manipuler

5.4.2 Documentation des constructeurs et destructeur

5.4.2.1 TextureManag()

```
TextureManag::TextureManag ( )
```

un constructeur

Le constructeur est la fonction membre appelee automatiquement lors de la creation d'un objet (en statique ou en dynamique). Cette fonction membre est la premiere fonction membre a etre executee, il s'agit donc d'une fonction permettant l'initialisation des variables.

5.4.2.2 ~TextureManag()

```
TextureManag::~~TextureManag ( ) [virtual]
```

un destructeur

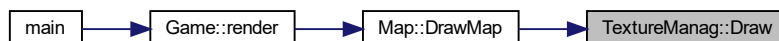
Le destructeurs sont en quelque sorte au constructeur ce que la mort est a la vie, c'est-a-dire qu'il s'agit d'une fonction membre qui intervient automatiquement lors de la destruction d'un objet.

5.4.3 Documentation des fonctions membres

5.4.3.1 Draw()

```
void TextureManag::Draw (
    SDL_Texture * tex,
    SDL_Rect src,
    SDL_Rect dest ) [static]
```

Voici le graphe des appelants de cette fonction :



5.4.3.2 loadTexture()

```
SDL_Texture * TextureManag::loadTexture (
    const char * filename ) [static]
```

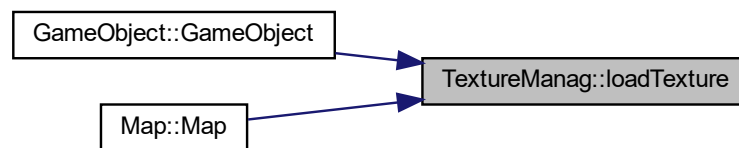
pour telecharger le fichier

pour dessiner le fichier

Paramètres

<i>filename</i>	le lien du fichier
<i>tex</i>	la texture
<i>src</i>	les references du fichier
<i>dest</i>	les references du fichier

Voici le graphe des appelants de cette fonction :



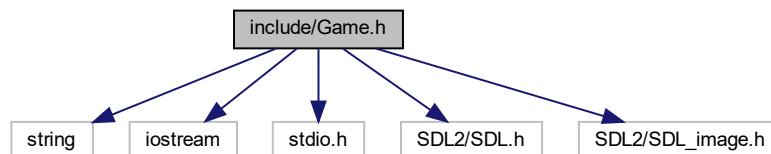
Chapitre 6

Documentation des fichiers

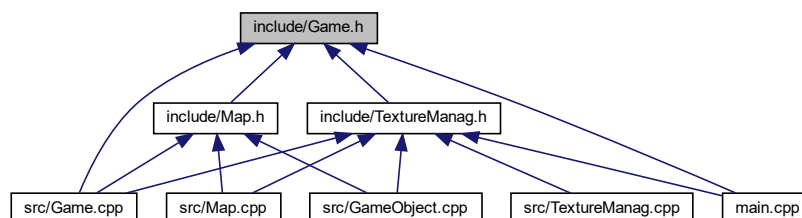
6.1 Référence du fichier include/Game.h

```
#include <string>
#include <iostream>
#include <stdio.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
```

Graphe des dépendances par inclusion de Game.h:



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



Classes

— class `Game`
une class du jeu.

Espaces de nommage

- namespace [iostream](#)
pour inclure des objets de flux globaux dans c++.
- namespace [stdio](#)
pour Standard Input Output Header ou En-tete Standard d'Entree/Sortie
- namespace [SDL2](#)
une bibliotheque utilisee en programmation multimedia.
- namespace [SDL2image](#)
Il est construit sur le framework OpenGL et utilise SDL pour la lectures des images.

6.2 Game.h

[Aller à la documentation de ce fichier.](#)

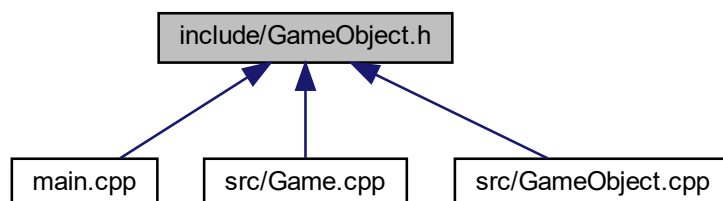
```

1 #ifndef GAME_H
2 #define GAME_H
3 #include <string>
4 #include <iostream>
13 #include <stdio.h>
17 #include <SDL2/SDL.h>
22 #include <SDL2/SDL_image.h>
23
24
34 class Game
35 {
36     public:
43         Game();
50         virtual ~Game();
64         void init(const char* title, int xpos,int ypos,int widhtpos,int heightpos, bool fullscreen);
72         void handleEvents();
73
79         void update();
85         void render();
91         void clean();
97         bool running(){return isRunning;}
98
99         static SDL_Renderer *renderer;
100         static SDL_Event event;
101     protected:
102
103     private:
104
105         bool isRunning;
106         SDL_Window *window;
107 };
108
109 #endif // GAME_H

```

6.3 Référence du fichier include/GameObject.h

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



Classes

- class [GameObject](#)
classe d'Objet

Espaces de nommage

- namespace [pragmaOnce](#)
ce package permet d'utiliser les classes une seule fois pour eviter la repition.

6.4 GameObject.h

[Aller à la documentation de ce fichier.](#)

```

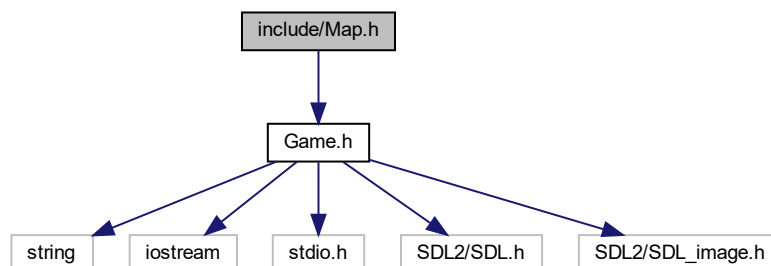
1 #ifndef GAMEOBJECT_H
2 #define GAMEOBJECT_H
3 #pragma once
4 #include "Game.h"
5 #include "Map.h"
6 class GameObject
7 {
8     public:
9         int getPosX();
10        int getY();
11        GameObject(const char* texturesheet,int x,int y);
12        virtual ~GameObject();
13        bool setX(int x , Map* m);
14        bool setY(int y, Map* m);
15        void Update();
16        void Render();
17
18    protected:
19
20    private:
21        int xpos;
22        int ypos;
23        SDL_Event event;
24        SDL_Texture* objTexture;
25        SDL_Rect srcRect, destRect;
26 };
27 #endif // GAMEOBJECT_H

```

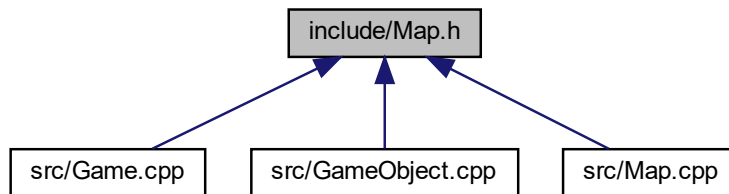
6.5 Référence du fichier include/Map.h

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

Graphe des dépendances par inclusion de Map.h:



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



Classes

— class `Map`
classe de la carte

6.6 Map.h

[Aller à la documentation de ce fichier.](#)

```

1 #ifndef MAP_H
2 #define MAP_H
3 #pragma once
4 #include "Game.h"
5
13 class Map
14 {
15     public:
22     Map();
29     virtual ~Map();
36     void LoadMap(int array[20][25]);
42     void DrawMap();
49     int* getMap();
50
51     protected:
52
53     private:
54
55     SDL_Rect src,dest;
56     SDL_Texture* box;
57     SDL_Texture* dirt;
58     SDL_Texture * grass;
59     SDL_Texture * floor;
60     SDL_Texture * goal;
62     int map[20][25];
64 };
65
66 #endif // MAP_H
  
```

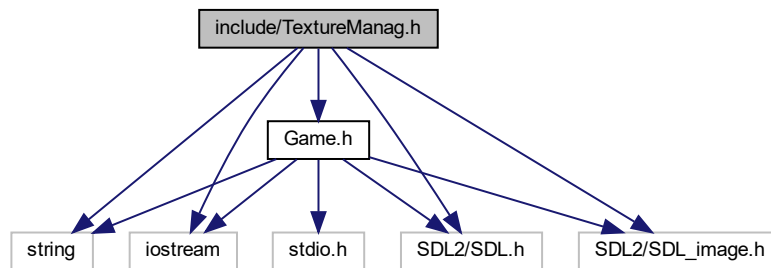
6.7 Référence du fichier include/TextureManag.h

```

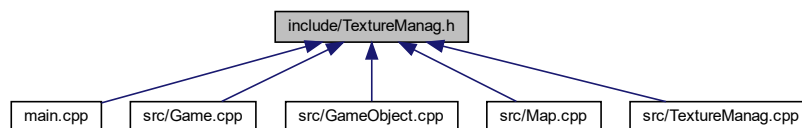
#include "Game.h"
#include <string>
#include <iostream>
#include <SDL2/SDL.h>
  
```

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

Graphe des dépendances par inclusion de TextureManag.h:



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



Classes

— class `TextureManag`
classe de la texture

6.8 TextureManag.h

[Aller à la documentation de ce fichier.](#)

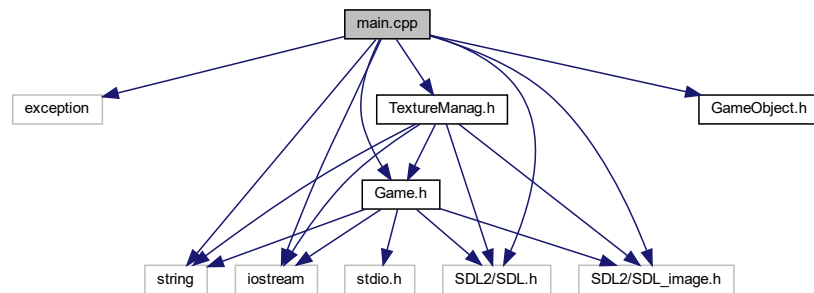
```

1 #ifndef TEXTUREMANAG_H
2 #define TEXTUREMANAG_H
3 #pragma once
4 #include "Game.h"
5 #include <string>
6 #include <iostream>
7 #include <SDL2/SDL.h>
8 #include <SDL2/SDL_image.h>
15 class TextureManag
16 {
17     public:
23         static SDL_Texture* loadTexture(const char* filename);
31         static void Draw(SDL_Texture* tex, SDL_Rect src, SDL_Rect dest);
38         TextureManag();
45         virtual ~TextureManag();
46
47     protected:
48
49     private:
50 };
51
52 #endif // TEXTUREMANAG_H
  
```

6.9 Référence du fichier main.cpp

```
#include <exception>
#include <string>
#include <iostream>
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include "Game.h"
#include "TextureManag.h"
#include "GameObject.h"
```

Graphe des dépendances par inclusion de main.cpp:



Fonctions

— int **main** (int argc, char *argv[])

Variables

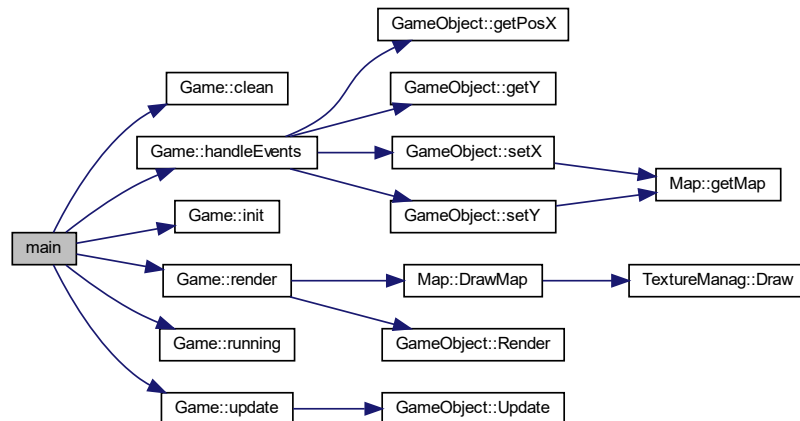
— **Game** * **game** = nullptr

6.9.1 Documentation des fonctions

6.9.1.1 main()

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

Voici le graphe d'appel pour cette fonction :



6.9.2 Documentation des variables

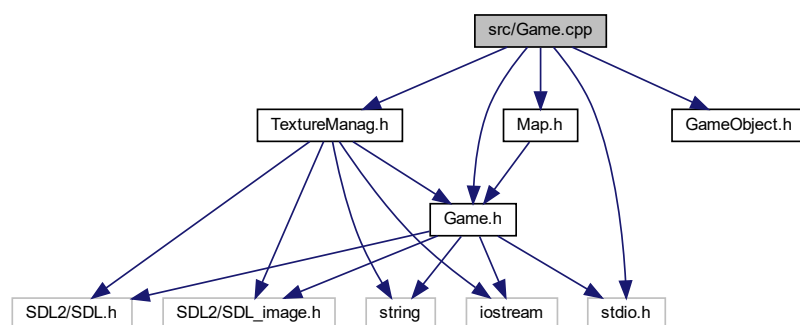
6.9.2.1 game

```
Game* game = nullptr
```

6.10 Référence du fichier src/Game.cpp

```
#include "Game.h"
#include "TextureManag.h"
#include "GameObject.h"
#include "Map.h"
#include <stdio.h>
```

Graphe des dépendances par inclusion de Game.cpp:



Variables

- `GameObject * player`
- `GameObject * box1`
- `GameObject * goal`
- `GameObject * boxOk`
- `GameObject * boxYup`
- `GameObject * boxYdown`
- `GameObject * boxXleft`
- `GameObject * boxXright`
- `Map * mape`

6.10.1 Documentation des variables

6.10.1.1 `box1`

`GameObject* box1`

6.10.1.2 `boxOk`

`GameObject* boxOk`

6.10.1.3 `boxXleft`

`GameObject* boxXleft`

6.10.1.4 `boxXright`

`GameObject* boxXright`

6.10.1.5 `boxYdown`

`GameObject* boxYdown`

6.10.1.6 boxYup

`GameObject* boxYup`

6.10.1.7 goal

`GameObject* goal`

6.10.1.8 mape

`Map* mape`

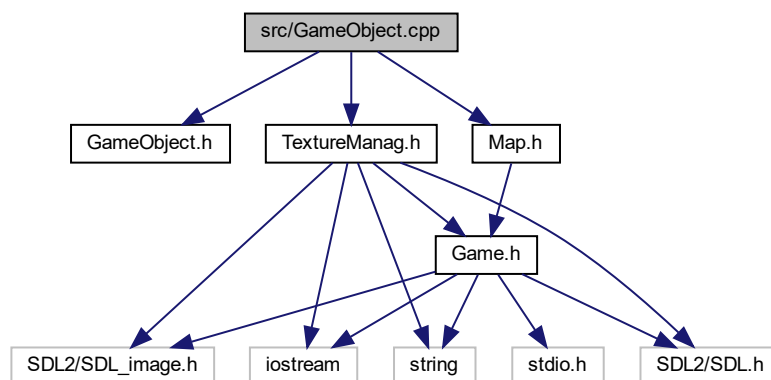
6.10.1.9 player

`GameObject* player`

6.11 Référence du fichier src/GameObject.cpp

```
#include "GameObject.h"  
#include "TextureManag.h"  
#include "Map.h"
```

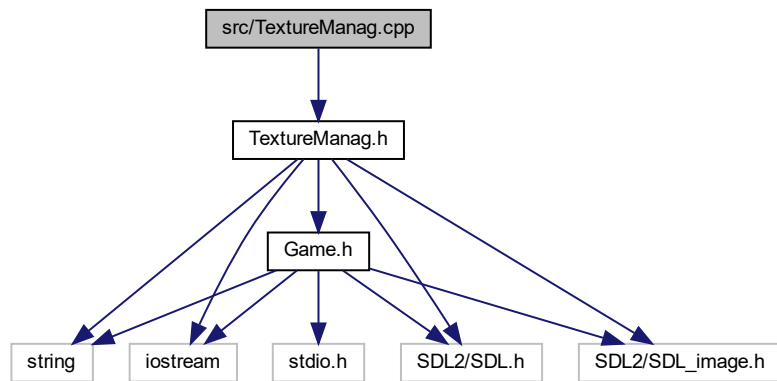
Graphe des dépendances par inclusion de GameObject.cpp:



6.13 Référence du fichier src/TextureManag.cpp

```
#include "TextureManag.h"
```

Graphe des dépendances par inclusion de TextureManag.cpp:



Index

- ~Game
 - Game, [10](#)
- ~GameObject
 - GameObject, [16](#)
- ~Map
 - Map, [20](#)
- ~TextureManag
 - TextureManag, [24](#)
- box1
 - Game.cpp, [34](#)
- boxOk
 - Game.cpp, [34](#)
- boxXleft
 - Game.cpp, [34](#)
- boxXright
 - Game.cpp, [34](#)
- boxYdown
 - Game.cpp, [34](#)
- boxYup
 - Game.cpp, [34](#)
- clean
 - Game, [11](#)
- Draw
 - TextureManag, [24](#)
- DrawMap
 - Map, [21](#)
- event
 - Game, [14](#)
- Game, [9](#)
 - ~Game, [10](#)
 - clean, [11](#)
 - event, [14](#)
 - Game, [10](#)
 - handleEvents, [11](#)
 - init, [12](#)
 - render, [12](#)
 - renderer, [14](#)
 - running, [13](#)
 - update, [13](#)
- game
 - main.cpp, [33](#)
- Game.cpp
 - box1, [34](#)
 - boxOk, [34](#)
 - boxXleft, [34](#)
 - boxXright, [34](#)
 - boxYdown, [34](#)
 - boxYup, [34](#)
 - goal, [35](#)
 - mape, [35](#)
 - player, [35](#)
- GameObject, [15](#)
 - ~GameObject, [16](#)
 - GameObject, [16](#)
 - getPosX, [16](#)
 - getY, [17](#)
 - Render, [17](#)
 - setX, [17](#)
 - setY, [18](#)
 - Update, [19](#)
- getMap
 - Map, [21](#)
- getPosX
 - GameObject, [16](#)
- getY
 - GameObject, [17](#)
- goal
 - Game.cpp, [35](#)
- handleEvents
 - Game, [11](#)
- include/Game.h, [27](#), [28](#)
- include/GameObject.h, [28](#), [29](#)
- include/Map.h, [29](#), [30](#)
- include/TextureManag.h, [30](#), [31](#)
- init
 - Game, [12](#)
- iostream, [7](#)
- LoadMap
 - Map, [22](#)
- loadTexture
 - TextureManag, [24](#)
- lvl1
 - Map.cpp, [36](#)
- main
 - main.cpp, [32](#)
- main.cpp, [32](#)
 - game, [33](#)
 - main, [32](#)
- Map, [19](#)
 - ~Map, [20](#)
 - DrawMap, [21](#)
 - getMap, [21](#)

- LoadMap, [22](#)
- Map, [20](#)
- Map.cpp
 - lvl1, [36](#)
- mape
 - Game.cpp, [35](#)
- player
 - Game.cpp, [35](#)
- pragmaOnce, [7](#)
- Render
 - GameObject, [17](#)
- render
 - Game, [12](#)
- renderer
 - Game, [14](#)
- running
 - Game, [13](#)
- SDL2, [7](#)
- SDL2image, [8](#)
- setX
 - GameObject, [17](#)
- setY
 - GameObject, [18](#)
- src/Game.cpp, [33](#)
- src/GameObject.cpp, [35](#)
- src/Map.cpp, [36](#)
- src/TextureManag.cpp, [37](#)
- stdio, [8](#)
- TextureManag, [22](#)
 - ~TextureManag, [24](#)
 - Draw, [24](#)
 - loadTexture, [24](#)
 - TextureManag, [23](#)
- Update
 - GameObject, [19](#)
- update
 - Game, [13](#)
