Sokoban

Généré par Doxygen 1.13.1

Chapter 1

Rapport

Nous avons choisi d'utiliser doxygen pour faire une documentation de notre code. La majorité de ce pdf est la documentation de notre code.

1.1 modélisation du jeu et décision de structure

Nous sommes parti sur un tableau 2d de `char`afin de représenter notre plateau. Nous avons utiliser des `char` car on sait qu'on a pas besoin de plus de possibilité de nombre que un octet. On a fait des `define` qui permettent de definir quelles nombres correspondent à quel objet dans le jeu (joueur , caisse, mur, ...).

Nous avons ensuite créer une structure `coor` qui est une structure representant des coordonnées x et y. Cela est pratique pour naviger dans le tableau 2d par exemple.

Nous utilisons une structure de score qui prend comme variable ce qu'il y a besoin pour calculer le score a la fin de la partie. Il y a le temps avant la partie et apres (pour obtenir le temps passer dans la partie), ensuite on a le nombre de déplacement fait par le joueur ainsi que les déplacement fait par les caisses.

Pour finir on a fais uns structure `dis` qui est une structure possédant tout ce qu'il faut pour faire l'affichage `SDL`. En effet on a la `window`, le `renderer`, on a la taille de la fenêtre, la taille des boites ainsi que la taille du menu.

Nous avons fait en sorte que pour n'importe quel écran la fenêtre s'adapte à celle-ci.

1.2 Comment lancer le jeu ?

Il vous faut installer SDL2, SDL2 Mixer, SDL2 Image, SDL2 ttf, make, gcc.

```
make all
./sokoban
```

1.3 Les contrôles:

- zqsd pour se déplacer
- échap pour sortir du niveau
- entrée pour sélectioner le niveau à jouer

1.4 Pour aller plus loin

(1) Tout nos niveau sont stocké dans des fichier .txt dans le dossier map. Les fichier nommé original_X sont au nombre de trois, et repésente des niveau que nous avons construit. Les fichier nommé custom_X sont les niveau crée avec l'éditeur de niveau. Nous chargement donc les niveaux depuis leurs fichier texte, de cette manièrere nous pouvons assez simplement ajouter ou retirer des niveau.

(2) Nous avons fait un affichage de score en fin de niveau, qui prends en compte:

- le temps passé dans le niveau.
- le nombre de mouvements du personnage.
- le nombre de mouvements des caisses.
- la réussite du niveau.

(3) Nous avons donc un affichage en temps réel du chronomètre, ainsi que le nombre de déplacement du joueur et le nombre de déplacement des caisses.

(4) Nous avons fait un menu dans lequel le joueur doit se déplacer sur un niveau, puis faire entré afin de lancer le niveau. Les niveau originaux sont imagé avec un escalier de couleur blanche tandis que les niveau crée avec l'éditeur sont imagé avec un escalier de couleur noir.

(5) La partie s'arrete automatiquement si le joueur ne peut plus gagner, dans ce cas il peut alors observer son score.

(+) L'éditeur de niveau: Nous avons créer un outil de création de niveau dans notre jeux, afin de créer des niveau plus rapidement. L'éditeur se trouve tout à droite du menu. Un fois rentrer dedans un fichier texte (custom_X.txt) va etre créer. Dans cette éditeur vous pouvez déplacer le personnages avec z,q,s,d, vous pouvez également modifier le plateau en faisant soit un clic gauche (sur la case voulue) pour placer un objet suivant, ou vous pouvez faire un clic droit pour placer l'objet précedent. Enfin il vous suffira de faire échape ou fermer le programme afin de sauvegarder le niveau. Il apparaitra donc en tant que niveau custom dans le menu principal. Attention : une fois un niveau enregistrer il n'est plus possible de le modifier via le programme, il vous faudra aller dans le dossier maps et modifier à la main les charactère.

(+) nous avons ajouter aussi quelques effet visuel... (génération avec seed de la végétation au sol, différents sprites pour le personnage suivant la direction ou encore un menu des titres)

Bon jeu !

Chapter 2

Index des structures de données

2.1 Structures de données

Liste des structures de données avec une brève description :

Chapter 3

Index des fichiers

3.1 Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

Chapter 4

Documentation des structures de données

4.1 Référence de la structure essential_sdl

Champs de données

- SDL_Window * **window**
- SDL_Renderer * **renderer**
- unsigned int **size_window**
- unsigned int **size_box**
- unsigned int **size_menu**

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

- function.h

4.2 Référence de la structure Score

Champs de données

- time_t **before**
- time_t **after**
- unsigned int **move_player**
- unsigned int **move_box**

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

- function.h

4.3 Référence de la structure Vecteur

Champs de données

- int **x**
- int **y**

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

- function.h

Chapter 5

Documentation des fichiers

5.1 display.h

```
00001 #ifndef DISPLAY_H
00002 #define DISPLAY_H
00003
00004 #include "../include/function.h"
00005 #include <SDL2/SDL.h>
00006 #include <SDL2/SDL_ttf.h>
00007
00008
00009
00010 extern unsigned int seed;
00011 void screenDisplay (char **tab, int x, int y);
00012 int getMaxSize (dis display_user);
00013 void displayImage (SDL_Renderer *renderer, SDL_Texture *texture, vect pos,
00014                    vect size);
00015 void initSDL (dis *display_user);
00016 void displayTextSDL(dis *display_user,char *text, vect coor, vect size, int font_size);
00017 void screenDisplayGameSDL (char **tab,vect dim_tab, dis *display_user, vect *player_pos, int fov, vect
      direction);
00018 void backgroundDisplay(dis *display_user,int bg);
00019 void playAudio(int sfx);
00020
00021 #endif // !DISPLAY_H
```

5.2 function.h

```
00001 #ifndef FONCTION_H
00002 #define FONCTION_H
00003
00004 #include <SDL2/SDL.h>
00005 #include <SDL2/SDL_render.h>
00006 #include <SDL2/SDL_video.h>
00007 #include <stdbool.h>
00008 #include <stdlib.h>
00009 #include <time.h>
00010
00011 #define EMPTY 1
00012 #define WALL 0
00013 #define BOX 2
00014 #define TARGET 3
00015 #define BOX_ON_TARGET 4
00016 #define PLAYER 5
00017 #define PLAYER_ON_TARGET 6
00018 #define BUTTON 7
00019 #define PLAYER_ON_BUTTON 8
00020 #define BUTTON_CUSTOM 9
00021
00022 typedef struct Vecteur
00023 {
00024     int x;
00025     int y;
00026 } vect;
00027
00028 typedef struct Score
```

```
00029 {
00030     time_t before;
00031     time_t after;
00032     unsigned int move_player;
00033     unsigned int move_box;
00034 } score;
00035
00036 typedef struct essential_sdl
00037 {
00038     SDL_Window *window;
00039     SDL_Renderer *renderer;
00040     unsigned int size_window;
00041     unsigned int size_box;
00042     unsigned int size_menu;
00043 } dis;
00044
00045 char **creatArea2D (const int x, const int y);
00046 void free2D (char **tab, int x);
00047 bool isWin (char **tab2d, vect *targets, int nbr_targets);
00048 bool islose (char **tab2d, const int N);
00049 bool blockBox (char **tab2d, vect box_coor);
00050 vect plusVect (vect one, vect two);
00051
00052 int inGameLoop (char **tab2d, vect *dim_tab, vect *playerPos, vect *targets,
00053                 int nbr_targets, dis *display_user, score *score_user, bool menu);
00054 char *timeToText(time_t time);
00055 void nullScore(score *player_score);
00056 void winOrLoseLoop(dis *display_user,score *score_user, bool win);
00057 unsigned int scoreCalculator(score *score_user, bool win);
00058 int inEditorLoop (char **tab2d, vect *dim_tab, vect *playerPos, dis *display_user,score *score_user,
      int num_fichier);
00059 int titleScreen (dis *display_user);
00060 #endif // FONCTION_H
```

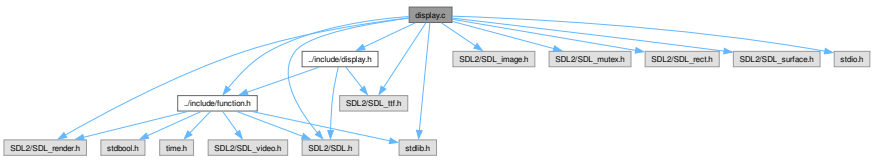
5.3 read.h

```
00001 #ifndef READ_H
00002 #define READ_H
00003 #include "function.h"
00004 vect *fileToTab2D (const char *name_file, char **tab,
00005                    const unsigned N, vect *player, int *nbr_targets);
00006 int countCustomMaps(char *name_directory);
00007 char ** generatorMenu(char *name_directory, vect *size, vect *pos_player);
00008 void save_grid_to_file(int filename, char **tab2D);
00009 #endif // !READ_H
```

5.4 Référence du fichier display.c

```
#include "../include/display.h"
#include "../include/function.h"
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_mutex.h>
#include <SDL2/SDL_mixer.h>
#include <SDL2/SDL_rect.h>
#include <SDL2/SDL_render.h>
#include <SDL2/SDL_surface.h>
#include <SDL2/SDL_ttf.h>
#include <stdio.h>
#include <stdlib.h>
```

Graphe des dépendances par inclusion de display.c:



Fonctions

- void **screenDisplay** (char ∗∗tab, int x, int y)

  La fonction permet d'afficher simplement le plateau de jeu dans le terminal.
- void **screenDisplayGameSDL** (char ∗∗tab, **vect** dim_tab, **dis** ∗display_user, **vect** ∗player_pos, int fov, **vect** direction)

  La fonction affiche a l'aide de SDL la zone de jeu.
- int **getMaxSize** (**dis** display_user)

  Taille de l'ecrant carre en fonction de l'ecrant du joueur avec une marge.
- void **initSDL** (**dis** ∗display_user)

  Initialise SDL.
- void **displayImage** (SDL_Renderer ∗renderer, SDL_Texture ∗texture, **vect** pos, **vect** size)

  Cette fonction affiche l'image dans la fenetre de l'utilisateur.
- void **displayTextSDL** (**dis** ∗display_user, char ∗text, **vect** coor, **vect** size, int font_size)

  Cette fonction affiche du texte dans la fenetre de l'utilisateur.
- void **backgroundDisplay** (**dis** ∗display_user, int bg)

  Afficher l'arriere plan.
- void **playAudio** (int sfx)

  Cette fonction permet de joueur des effet sonor.

5.4.1 Description détaillée

Fichier contient tout les fonctions pour l'affichage.

5.4.2 Documentation des fonctions

5.4.2.1 backgroundDisplay()

```
void backgroundDisplay (
              dis * display_user,
             int bg)
```

Afficher l'arriere plan.

Paramètres

display_user	Tout les information du display de l'utilisateur utile.
bg	quel back ground afficher.

Voici le graphe d'appel pour cette fonction :

5.5 Référence du fichier function.c

```
#include "../include/function.h"
#include "../include/display.h"
#include "../include/read.h"
#include <SDL2/SDL_events.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_keycode.h>
#include <SDL2/SDL_rect.h>
#include <SDL2/SDL_render.h>
#include <SDL2/SDL_scancode.h>
#include <SDL2/SDL_timer.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

Graphe des dépendances par inclusion de function.c:

Fonctions

- char ** **creatArea2D** (const int x, const int y)

  Cette fonction permet de creer une liste 2D.
- void **free2D** (char **tab, int x)

  Cette fontion permet de liberer l'espace tu tableau 2D de char.
- char **canIGoDirection** (char valueOfNCase, char valueOfNPlusOneCase)

  La fontction permet de savoir si le joueur peut ce deplacer dans une direction.
- int **move** (char **tab, **vect** *playerPos, **vect** direction, **score** *score_user)

  Cette fonction effectue les deplacements du joueur et des boites en fonction de la situation.
- int **inGameLoop** (char **tab2d, **vect** *dim_tab, **vect** *playerPos, **vect** *targets, int nbr_targets, **dis** *display_user, **score** *score_user, bool menu)

  La fonction permet de faire la boucle de jeu et le menu.
- int **inEditorLoop** (char **tab2d, **vect** *dim_tab, **vect** *playerPos, **dis** *display_user, **score** *score_user, int num_fichier)

  La fonction permet de faire la boucle de l'éditeur.
- int **titleScreen** (**dis** *display_user)

  La fonction permet d' afficher le title screen.
- bool **isWin** (char **tab2d, **vect** *targets, int nbr_targets)

  Cette fonction verrifie si la partie est gagnante.
- bool **islose** (char **tab2d, const int N)

  La fonction renvois si la partie est perdante.
- **vect** **plusVect** (**vect** one, **vect** two)

  La fonction fait une addition de vecteur, (x1+x2, y1+y2).
- int **lengthVect** (**vect** vector)

  Renvois la longeur Manhattan.
- bool **blockBox** (char **tab2d, **vect** box_coor)

  La fonction permet de savoir si une boite est dans une situation ou le joueur ne poura pas la debloqué.
- char * **timeToText** (time_t time)

  Cette fonction renvois transforme le forma time en texte. (min:sec)
- void **nullScore** (**score** *player_score)

  Mets à 0 le score.
- void **winOrLoseLoop** (**dis** *display_user, **score** *score_user, bool win)

  Fonction de loop pour la win ou la lose.
- unsigned int **scoreCalculator** (**score** *score_user, bool win)

5.5.1 Description détaillée

Ce fichier contient toute les fonction utile pour le jeu. Sauf pour l'affichage.

5.5.2 Documentation des fonctions

5.5.2.1 blockBox()

```
bool blockBox (
             char ** tab2d,
              vect box_coor)
```

La fonction permet de savoir si une boite est dans une situation ou le joueur ne poura pas la debloqué.

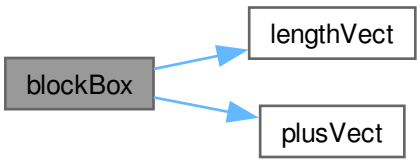
Paramètres

tab2d	Le tableau 2D carre du plateau de jeu.
box_coor	Les corrdonnée de la boite que la fonction test.

Renvoie

True si la la boite est bloquer, sinon false.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

5.5.2.2 canIGoDirection()

```
char canIGoDirection (
             char valueOfNCase,
             char valueOfNPlusOneCase)
```

La fontction permet de savoir si le joueur peut ce deplacer dans une direction.

Paramètres

valueOfNCase	La valeur de la case dans la direction que le joueur veut aller.
valueOfNPlusOneCase	La valeur de la case dans la direction que le joueur veut aller mais une fois de plus.

Renvoie

0 Si c'est un mur devant le joueur, 1 si c'est vide devant le joueur, 2 si c'est une boite mais qu'on peux la pousser ,3 si le joueur pousse une boite sur un point d'interer, 4 si le joueur bouge sur un point d'interer, 5 si le joueur peux pousser une boite mais le joueur se place sur un point d'interer et 6 si le joueur pousse une boite sur un point d'interer et que le joueur est aussi sur un point d'interer. 7 si valueOfNCase est un bouton.

Voici le graphe des appelants de cette fonction :

5.5.2.3 creatArea2D()

```
char ** creatArea2D (
            const int x,
            const int y)
```

Cette fonction permet de creer une liste 2D.

Paramètres

x	Nombre de ligne.
y	Nombre de colonne.

Renvoie

Le pointeur du tableau 2D de char (1 octet).

5.5.2.4 free2D()

```
void free2D (
            char ** tab,
            int x)
```

Cette fontion permet de liberer l'espace tu tableau 2D de char.

Paramètres

tab	Le tableau 2D.
x	Le nombre de ligne.

Renvoie

Void.

5.5.2.5 inEditorLoop()

```
int inEditorLoop (
             char ** tab2d,
              vect * dim_tab,
              vect * playerPos,
              dis * display_user,
              score * score_user,
             int num_fichier)
```

La fonction permet de faire la boucle de l'éditeur.

Paramètres

tab2d	Le tableau 2d carre.
N	LE nombre d'element de tab2d.
playerPos	La position de depart du joueur.
targets	Le tableau de toutes les positions des points d'interer de la maps.
int	nbr_targets Le nombre de point d'interer.
display_user	Tout les information SDL pour afficher le jeu.
score_user	Toute les données nécessaire pour calculer le score fini du joueur.
num_fichier	Simplement le numéro de la map actuel.

Renvoie

renvoie -2 ce qui signifie l'editeur

Voici le graphe d'appel pour cette fonction :

5.5.2.6 inGameLoop()

```
int inGameLoop (
             char ** tab2d,
              vect * dim_tab,
              vect * playerPos,
              vect * targets,
             int nbr_targets,
              dis * display_user,
              score * score_user,
             bool menu)
```

La fonction permet de faire la boucle de jeu et le menu.

Paramètres

tab2d	Le tableau 2d carre.
N	LE nombre d'element de tab2d.
playerPos	La position de depart du joueur.
targets	Le tableau de toutes les positions des points d'interer de la maps.
int	nbr_targets Le nombre de point d'interer.
display_user	Tout les information SDL pour afficher le jeu.
score_user	Toute les données nécessaire pour calculer le score fini du joueur.
menu	True si c'est la loop du menu.

Renvoie

L'indice de la map si c'est un menu, sinon renvoie -1

Voici le graphe d'appel pour cette fonction :

5.5.2.7 islose()

```
bool islose (
            char ** tab2d,
            const int N)
```

La fonction renvois si la partie est perdante.

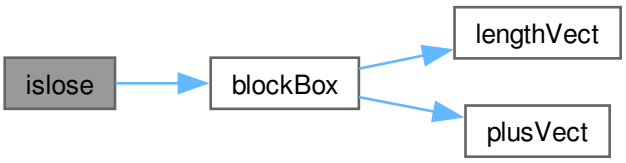
Paramètres

tab2d	Le tableau 2d carre du plateau de jeu.
N	Le nombre d'éléments dans le tab2d. (zone de jeu est carré)

Renvoie

True si c'est perdu, false si c'est pas perdu a cette instant.

Voici le graphe d'appel pour cette fonction :



5.5.2.8 isWin()

```
bool isWin (
            char ** tab2d,
             vect * targets,
            int nbr_targets)
```

Cette fonction verrifie si la partie est gagnante.

Paramètres

tab2d	Le tableau 2D du jeu.
targets	Le tableau de toute les positions des points d'interer.
nbr_targets	Le nombre de points d'interer.

Renvoie

True si le joueur a remplis tout les points d'interer, false si ce n'est pas le cas .

5.5.2.9 move()

```
int move (
             char ** tab,
              vect * playerPos,
              vect direction,
              score * score_user)
```

Cette fonction effectue les deplacements du joueur et des boites en fonction de la situation.

Paramètres

tab	Le tableau 2D du plateau de jeu.
playerpos	La position actuel du joueur.
direction	La direction que le joueur veut effectuer.
score_user	Toutes les données nécessaire pour calculer le score fini du joueur.

Renvoie

int return 1 si le joueur n'a pas bouger.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

5.5.2.10 nullScore()

```
void nullScore (
              score * player_score)
```

Mets à 0 le score.

Paramètres

player_score	Le score a mettre à 0.

Renvoie

void

5.5.2.11 plusVect()

```
vect plusVect (
              vect one,
              vect two)
```

La fonction fait une addition de vecteur, (x1+x2, y1+y2).

Paramètres

one	Premier vecteur.
two	Deuzieme vecteur.

Renvoie

vect Un vecteur de l'addition de one et two.

Voici le graphe des appelants de cette fonction :

5.5.2.12 timeToText()

```
char * timeToText (
            time_t time)
```

Cette fonction renvois transforme le forma time en texte. (min:sec)

Paramètres

time	Le temps qu'on veux convertire.

Renvoie

char Le string du texte.

Voici le graphe des appelants de cette fonction :

5.5.2.13 titleScreen()

```
int titleScreen (
            dis * display_user)
```

La fonction permet d' afficher le title screen.

Paramètres

display_user	Tout les information SDL pour afficher le jeu.

Renvoie

renvoie -3 ce qui signifie le title screen

5.5.2.14 winOrLoseLoop()

```
void winOrLoseLoop (
            dis * display_user,
            score * score_user,
           bool win)
```

Fonction de loop pour la win ou la lose.

Paramètres

display_user	Tout les information du display de l'utilisateur utile.
win	Si on veut un affichage de victoire ou non.

5.6 Référence du fichier main.c

```
#include "../include/display.h"
#include "../include/function.h"
#include "../include/read.h"
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_render.h>
#include <SDL2/SDL_mixer.h>
#include <time.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

Graphe des dépendances par inclusion de main.c:

Macros

- #define **SIZE_PLAY** 20
- #define **SIZE_MENU** 200

Fonctions

- int **main** ()

Variables

- unsigned int **seed** = 0

5.6.1 Description détaillée

Le main permet de stocker et de lancer les fonctions permetant de lancer le jeu.

5.7 Référence du fichier read.c

```
#include "../include/function.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Graphe des dépendances par inclusion de read.c:

Fonctions

- **vect** ∗ **fileToTab2D** (const char ∗name_file, char ∗∗tab, const unsigned N, **vect** ∗player, int ∗nbr_targets)

  La fonction permet de stocker la zone de jeu en fonction de la lecture d'un fichier.
- int **countCustomMaps** (char ∗name_directory)

  La fonction permet de compter le nombre de maps custom dans le fichier des maps.
- char ∗∗ **generatorMenu** (char ∗name_directory, **vect** ∗size, **vect** ∗pos_player)

  La fonction permet de creer la zone de jeu du menu en fonction du nombre de maps custom.
- void **save_grid_to_file** (int filename, char ∗∗tab2D)

  La fonction permet de retranscrire un niveau creer du tableau vers le fichier.txt.

5.7.1 Description détaillée

Ce fichier est le programe qui lit d'autre fichier, nottament les maps.

5.7.2 Documentation des fonctions

5.7.2.1 countCustomMaps()

```
int countCustomMaps (
            char * name_directory)
```

La fonction permet de compter le nombre de maps custom dans le fichier des maps.

Paramètres

name_directory	Le nom du dossier contenant les maps. return Le nombre de maps custom.

Voici le graphe des appelants de cette fonction :

5.7.2.2 fileToTab2D()

```
vect * fileToTab2D (
            const char * name_file,
            char ** tab,
            const unsigned N,
             vect * player,
            int * nbr_targets)
```

La fonction permet de stocker la zone de jeu en fonction de la lecture d'un fichier.

Paramètres

name_file	Le nom du fichier a ouvrir.
tab	Le tableau 2D carre du plateau de jeu a remplir.
N	La taille de tab.
player	Les coordonnée du joueur que le programe vas trouvé.
nbr_targets	Le nombre de points d'interer trouver.

Renvoie

Vect La fonction renvoie le tableau des coordonnée des points d'interer.

5.7.2.3 generatorMenu()

```
char ** generatorMenu (
            char * name_directory,
             vect * size,
             vect * pos_player)
```

La fonction permet de creer la zone de jeu du menu en fonction du nombre de maps custom.

Paramètres

name_directory	Le nom du dossier contenant les maps.
size	La taille de la zone de jeu.
pos_player	La position du joueur dans le menu.

Voici le graphe d'appel pour cette fonction :

5.7.2.4 save_grid_to_file()

```
void save_grid_to_file (
             int filename,
             char ** tab2D)
```

La fonction permet de retranscrire un niveau creer du tableau vers le fichier.txt.

Paramètres

filename	Le nom du fichier a remplir.
tab2d	Le tableau 2d carre.