L2 INFORMATIQUE – PROGRAMMATION AVANCEE
PROJET 2019-2020: JEU DU SOKOBAN

Wassim Saïdane et Florian Mouchet

12 Janvier 2020

Table des matières

Introduction

Le but du projet est de concevoir un Sokoban en langage C en utilisant la librairie SDL 1.2. Sokoban est un jeu vidéo de puzzle inventé au Japon.
le joueur doit ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions, et pousser (mais pas tirer) une seule caisse à la fois. Une fois toutes les caisses rangées, le niveau est réussi.
`https://fr.wikipedia.org/wiki/Sokoban`

Pour ce projet nous avons décidé d'adapter ce jeu à travers l'univers du mangas One Piece.

Dans ce rapport nous allons vous présentais le code de toutes les fonctions que nous avons codé afin de réaliser le projet.
Pour pouvoir compiler notre code nous avons utilisé une makefile :
- Compiler : make
- Exécuter : .*sokoban*

1 Le programme principal

Voici le programme principal, le principe étant l'affichage de la table de jeu à partir d'un fichier texte et la gestion des événements.

```
1  int main(void){
2      FILE * fd = fopen("soko.txt", "r");
3      char **tab=initTab(fd);
4      int * pos = malloc(3*sizeof(int));
5      pos = affiche(tab);
6      boucleEv(tab, pos);
7      SDL_Quit();
8      fclose(fd);
9      freeTab2D(tab);
10     return EXIT_SUCCESS;
11 }
```

Ligne 1 : Le programme ne prend pas d'argument.
Ligne 2 : On ouvre le fichier (voir 2) en lecture.
Ligne 3 : On crée un tableau 2D où l'ont stock les valeurs du fichier à l'aide de la fonction initTab (voir 3).
Ligne 4 : On crée un tableau (pos) de 3 cases contenant la position courante du joueur ainsi que la direction qu'il va prendre (voir 5).
Ligne 5 : On appel la fonction affiche qui affiche le tableau de jeu et qui fournit pos (voir 4).
Ligne 6 : On gère les événements (instruction de l'utilisateur) (voir 6).
Ligne 7 : On ferme la librairie.
Ligne 8 : On ferme le fichier.
Ligne 9 : On libère la mémoire du tableau 2d.

2 La carte initiale du jeu (voir soko.txt)

Comme nous l'avons dit dans la partie 1. Un fichier texte va être utiliser afin de générer la carte du jeu.
Pour notre projet ce fichier doit contenir au maximum 12 lignes et 12 colonnes.
Le caractère '#' représentera un mur.
Le caractère 'C' représentera une caisse.
Le caractère 'I' représentera une cible.
Les caractères ' ' (ou '0' par la suite) représente le sol.

3 Le stockage des valeurs du fichier

Nous avons décidé d'utiliser un tableau 2D pour stocké les valeurs du fichier. Pour cela nous avons codé la fonction initTab qui prend en argument un descripteur de ficher et renvoie un tableau 2D.

```
1  char **initTab(FILE *fd){
2      char ** tab = malloc(12*sizeof(char*));
3      for(int i = 0; i<12; i++)
4      tab[i] = malloc(12*sizeof(char));
5      char buffer[BUFSIZ];
6      int l = 0;
7      while(fgets(buffer, BUFSIZ, fd) != NULL){
8      int k = 0;
9      while(buffer[k] != '\n'){
10       tab[l][k] = buffer[k];
11       k++;
12     }
13     l++;
14   }
15   return tab;
16 }
```

Ligne 2 à 4 : On crée le tableau.
Ligne 5 : On crée un buffer.
Ligne 7 à 14 : On stock les valeurs du fichier dans le buffer, pour ensuite les stocké dans le tableau.
Ligne 15 : On retourne le tableau.

4 L'affichage

La fonction affiche permet l'affichage du plateau de jeu en fonction du tableau passé en paramètre et retourne la position courante (utile pour la gestion des évènements).

4.1 Première partie - fenêtre et chargement d'image

```
1  int * affiche(char **tab){
2      int * pos = malloc(2*sizeof(int));
3      SDL_Surface * ecran = NULL, * mur = NULL, * luffy=NULL, * caisse=NULL, *
       objectif=NULL, * fond=NULL;
4      SDL_Rect position, posluffy, poscaisse, posobjectif, posFond;
5      position.x=0;
6      position.y=0;
7      posluffy.x=0;
8      posluffy.y=0;
9      poscaisse.x=0;
10     poscaisse.y=0;
11     posobjectif.x=0;
12     posobjectif.y=0;
13     if(SDL_Init(SDL_INIT_VIDEO)!=0){
14         fprintf(stderr, "\nUnable to initialise SDL: %s\n",SDL_GetError());
15         exit(EXIT_FAILURE);
16     }
17     SDL_WM_SetIcon(SDL_LoadBMP("icone.bmp"), NULL);
18     if ((ecran = SDL_SetVideoMode(442,442,32,SDL_HWSURFACE))==NULL){
```

```
19          fprintf(stderr, "Erreur VideoMode %s\n",SDL_GetError() );
20          exit(EXIT_FAILURE);
21      }
22      SDL_WM_SetCaption("One Piece Sokoban", NULL);
23      mur = SDL_LoadBMP("mur.bmp");
24      luffy = SDL_LoadBMP("luffy.bmp");
25      caisse = SDL_LoadBMP("caisse.bmp");
26      objectif = SDL_LoadBMP("objectif.bmp");
27      fond = SDL_LoadBMP("fond.BMP");
```

Ligne 3 : On crée les différents objets de type SDL_surface et on les initialises à NULL.
Ces objets sont :
- ecran : La fenêtre.
- mur : Le mur.
- luffy : Le joueur.
- caisse : La caisse.
- objectif : La cible.
- fond : Il s'agit du fond de l'écran (en réalité le sol.).

Ligne 4 à 12 : On définie les positions de nos différents objets du type SDL _Rect et on les initialises à 0 (position est la position du mur).
Ligne 13 à 16 : Cas d'erreur.
Ligne 17 : Chargement de l'icône de la fenêtre.
Ligne 18 à 21 : On fixe la taille de la fenêtre à 442 par 442 (une unité faisant 34 par 34 | 13*34=442) à l'aide de la fonction SDL _SetVideoMode.
Ligne 22 à 27 : Chargement des images.

4.2 Deuxième partie - Affichage des objets

```
1       for (int i=0;i<12;i++){
2           position.x=i*34;
3           posluffy.x=i*34;
4           poscaisse.x=i*34;
5           posobjectif.x=i*34;
6           posFond.x=i*34;
7           for (int j=0;j<12;j++){
8               position.y=j*34;
9               posluffy.y=j*34;
10              poscaisse.y=j*34;
11              posobjectif.y=j*34;
12              posFond.y=j*34;
13              SDL_BlitSurface(fond, NULL, ecran, &posFond);
14              SDL_Flip(ecran);
15              if (tab[i][j]=='#'){
16                  SDL_BlitSurface(mur, NULL, ecran, &position);
17                  SDL_Flip(ecran);
18              }
19              else if (tab[i][j]=='P' || tab[i][j]=='@'){
20                  SDL_BlitSurface(luffy, NULL, ecran, &posluffy);
21                  SDL_Flip(ecran);
22                  pos[0] = i;
23                  pos[1] = j;
24              }
25              else if (tab[i][j]=='C' || tab[i][j]=='X'){
26                  SDL_BlitSurface(caisse, NULL, ecran, &poscaisse);
```

```
27                  SDL_Flip(ecran);
28              }
29              else if (tab[i][j]=='I'){
30                  SDL_BlitSurface(objectif, NULL, ecran, &posobjectif);
31                  SDL_Flip(ecran);
32              }
33          }
34      }
35      SDL_Flip(ecran);
36      return pos;
37  }
```

Ligne 1 à 12 : On place les positions des différents objets selon les positions des valeur du tableau, on multiplie la valeur la position du tableau par 34 pour passé d'une unité à l'autre.
Ligne 13 et 14 : On place le sol sur toutes la fenêtres à l'aide de SDL_BlitSurface.
Ligne 15 à 35 : On place les différents objets selon le caractère du tableau ('X' signifie que la caisse est sur la cible).

Dans cette fonction nous avons également crée le tableau pos désignant la position courante du joueur. Lorsque que la valeur du tableau correspond au joueur, les coordonnées de celui-ci sont enregistré dans les deux premières cases de pos. Ce tableau est ensuite renvoyé.

5 La position courante

Afin de pouvoir garder la position du joueur pour la gestion des événements. Nous avons crée un tableau contenant cette dernière.

$pos[0]$	$pos[1]$	$pos[2]$
Position horizontale courante	Position verticale courante	Déplacement du joueur

Si pos[2]=0, le joueur se déplace vers le haut.
Si pos[2]=1, le joueur se déplace vers le bas.
Si pos[2]=2, le joueur se déplace vers la gauche.
Si pos[2]=3, le joueur se déplace vers la droite.

6 La gestion des événements

La gestion des événements se fait dans la fonction boucleEv, qui prend en argument le tableau ainsi que la position courante.
Le but est de modifier le tableau afinn d'appeler la fonction affiche à chaque modification de ce dernier.

```
void boucleEv(char **tab, int * pos){
    int cont = 1;
    SDL_Event event;
    while(cont){
        SDL_WaitEvent(&event);
        switch(event.type){
            case SDL_KEYDOWN:
                switch(event.key.keysym.sym){
                    case SDLK_UP:
                        if (tab[pos[0]][pos[1]-1]!='#'){
                            pos[2] = 0;
                            pos[1] -=1;
                        }
                    break;
                    case SDLK_DOWN:
                        if (tab[pos[0]][pos[1]+1]!='#'){
                            pos[2] = 1;
                            pos[1] += 1;
                        }
                    break;
                    case SDLK_LEFT:
                        if (tab[pos[0]-1][pos[1]]!='#'){
                            pos[2] = 2;
                            pos[0] -= 1;
                        }
                    break;
                    case SDLK_RIGHT:
                        if (tab[pos[0]+1][pos[1]]!='#'){
                            pos[2] = 3;
                            pos[0] += 1;
                        }
                    break;
                    default:
                    break;
                }
                affiche(modifTab(tab, pos));
            break;
            case SDL_QUIT:
                cont=0;
        break;
        }
        if (gagne(tab))
            cont = 0;
    }
}
```

Ligne 2 : la valeur cont est un entier (utilisé en réalité comme un booléen), lorsqu'il vaut 1 le jeu est en cours d'exécution, 0 si non.
Ligne 3 : On crée un évènement event du type SDL_Event.
Ligne 6 : On utilise un switch où on attend deux types d'événements :
- Une saisie clavier
Selon la flèche directionnelle enfoncer, on va modifier la position courante si le joueur n'est pas face à un mur. Par exemple si la touche enfoncer est la flèche du haut, pos[2] prend la valeur du haut (0) et la position du joueur est modifié verticalement vers le haut (pos[1]-=1).
Ensuite modifier le tableau avec cette nouvelle position (voir 6.1) et afficher le plateau correspondant à ce nouveau tableau. - L'utilisateur quitte le jeu.

Ligne 42 et 43 : On détermine à l'aide de la fonction gagne (voir 6.2) si l'utilisateur à gagné.

6.1 La modification du tableau (voir code source)

La modification du tableau se fait avec la fonction modifTab elle prend en argument le tableau ainsi que la position courante et renvoie un nouveau tableau.
Le code de cette fonction est assez long car on traitre les événement au cas par cas.
Voici les différents cas que nous avons traité :
- Le cas où le joueur est devant un mur ou devant rien à déjà était partiellement traité dans boucleEv.
- Le cas où le joueur est devant une caisse, dans ce cas là soit après la caisse il n'y a rien ou la cible donc il peut bouger. Soit il y a un mur ou une autre caisse et il ne peut pas se déplacer.
- Le cas où ou le joueur est devant une cible, afin d'éviter que le personnage "écrase" la cible.

Pour tous ces cas on a traité les 4 directions et on à modifier les valeurs du tableau afin que l'affichage reste cohérent.

6.2 La fonction gagne

Cette fonction permet de savoir si le joueur à gagné. Elle prend en argument un tableau et renvoie 1 si le joueur à gagner.

```
1  int gagne(char **tab){
2      int compteur = 0;
3      for (int i = 0; i < 12; i++) {
4          for (int j = 0; j < 12; j++) {
5              if (tab[j][i] == 'I' || tab[j][i] == '@')
6                  compteur++;
7          }
8      }
9      return (compteur == 0);
10 }
```

On initialise un compteur à 0, on l'incrémente si on observe une cible qui n'a pas de caisse (soit on a la cible visite, soit le joueur est dessus). On retourne 1, si le compteur vaut 0 (cela veut dire qu'on observe ni de cible libre ni de joueur sur la cible donc toutes les caisses sont sur les cibles.).

Conclusion

Au cours de ce projet nous avons découvert et manipulés la librairie SDL, le jeu est loin d'être optimal compte tenu du fait qu'on affiche tous l'écran à chaque déplacement. Les principales difficultés étaient la gestion des collisions entre objets.