Rapport-Jeu de Sokoban

Eldis YMERAJ, Rodrigo FERREIRA RODRIGUES et M'Hammed Salmân ABIBOU

06/01/2023

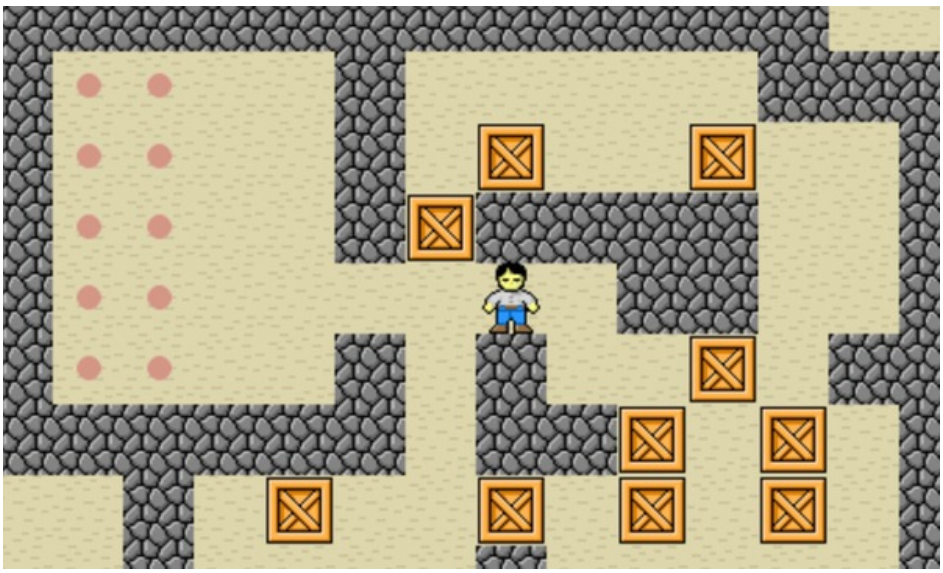

Figure 1 – Jeu de Sokoban

Table des matières

Introduction **3**

0.1 Règles du jeu 3

0.2 Plateau de jeu 3

0.3 Fonctionnalités 4

1 Structure du code **4**

2 Implémentation des fonctionnalités **5**

2.1 Affichage 5

2.2 Déplacement 6

2.3 Sauvegarde/Chargement 7

2.4 Menu/Niveaux 8

2.5 Chronomètre 9

2.6 Score 10

Introduction

Le but de ce projet est d'implementer en langage de programmation **C** le jeu **'SOKOBAN'**. Seules certaines parties du code seront présentées ici, et nous avons notamment pris soin d'omettre les aspects les plus mondains ou encombrants du code, comme l'allocation et la libération de la mémoire, la déclaration de certaines variables, certaines fonctions auxiliaires, la validation des inputs, quelques affichages console, et les demandes d'input via console. L'ensemble du programme offre plus de rigueur, et nous nous sommes efforcés de fournir du code propre et lisible quand nous en étions capables.

0.1 Règles du jeu

Les règles du jeu que l'on souhaite implémenter sont les suivantes : Nous disposons d'un plateau de jeu constitué de murs et de cases libres. Nous pouvons déplacer un personnage dans les 4 directions (haut, bas, gauche et droite) pourvu que la case visée ne soit pas un mur. Sur le plateau sont disposés des caisses et des points d'intérêts sur des cases particulières. Le but étant de couvrir par des caisses tous les points désignés. Pour ce faire le personnage peut déplacer les caisses en les poussant. Cependant, pour pouvoir déplacer une caisse, il faut qu'il y ait une case libre de part et d'autre de la caisse. Si par exemple le personnage est à droite de la caisse et qu'une case est libre à gauche de la caisse, le personnage pourra déplacer la caisse vers la gauche. Ainsi une fois que la caisse est contre un mur, il est souvent difficile de la décoller du mur.

0.2 Plateau de jeu

Pour représenter le plateau de jeu, nous avons décidé d'utiliser un tableau à deux dimensions de taille 20×20. Il s'agit d'un tableau d'entiers :

- **-1** représente un **mur**
- **0** représente un **casse vide**
- **1** représente le **personnage**
- **2** représente une **caisse**
- **3** représente un **objectif**
- **4** représente le **personnage sur un objectif**
- **5** représente une **caisse sur un objectif**

On a choisi cette méthode car nous pensons qu'elle est plus facile à traiter et moins coûteuse en temps.

0.3 Fonctionnalités

A travers notre jeu, nous avons mis en place plusieurs fonctionnalités :

- le jeu en lui-même, c'est-à-dire la possibilité de bouger le personnage et de pousser les caisses sur des objectifs
- des niveaux avec un ordre de difficulté et la possibilité de naviguer entre eux
- un système de détection de fin de niveau pour savoir quand le joueur a gagné ou est bloqué
- un chronomètre, commençant lorsque la partie débute et se mettant à jour à chaque input
- un système de score basé sur le nombre d'input du joueur et du temps écoulé
- un système de sauvegarde permettant au joueur de reprendre sa partie où il l'a laissée

1 Structure du code

Nous avons structuré le code de jeu dans trois fichiers principaux :

- 1. Le fichier **"inc"** qui contient les prototypes des fonctions utilisées pour le jeu.
- 2. Le fichier **"map"** qui contient tous les plateux de jeu pour chaque niveau.
- 3. Le fichier **"src"** contient toutes les fonctions du jeu qu'on a developpées.

2 Implémentation des fonctionnalités

2.1 Affichage

Ici, nous allons répertorier toutes les fonctions liés à l'affichage du plateau lors du jeu. A noter que # représente un mur, **P** le personnage, **c** une caisse, **ç** une caisse sur un objectif, **I** un objectif et une casse vide.

affichage(...) :

— Est une fonction qui affiche le plateau du jeu.

Signature de fonction :

```
1 void affichage(int **plateau, int taille);
```

affiche-score(...) :

— Est une fonction qui affiche le score.

Signature de fonction :

```
1 void affiche_score(int best_score, int score);
```

2.2 Déplacement

Ici, nous allons vous expliquer toutes les fonctions en charge du déplacement du personnage à travers le plateau. Le joueur peut le faire grâce aux touches **z,q,s,d**.

deplacement-possible(...) :

- Fonction renvoyant si le deplacement d'une entite est possible dans une telle direction.
- Si entite = 1 alors on s'interesse au personnage si entite = 2 alors il s'agit d'une caisse.
- x, y sont les coordonnées de l'entite dans le tableau.
- La fonction renvoie 1 si deplacement possible, 0 sinon.

Signature de fonction :

```
1 int deplacement_possible(int **plateau, int taille, int x, int y, char direc,
    int entite);
```

maj-carte(...) :

- Cette fonction met à jour les valeurs des cases du tableau selon les dépalcements effectués.
- Elle prend en argument, le plateau, les coordonnées du personnage.
- depla-x indique si le personnage bouge de façon verticale(-1 s'il monte, +1 s'il descend).
- depla-y indique si le personnage bouge de façon horizontale(-1 s'il va à gauche, +1 s'il va à droite).

Signature de fonction :

```
1 void maj_carte(int ** plateau, int *x, int *y, int depla_x, int depla_y);
```

deplacememt(...) :

- Cette fonction s'occupe de gérer le déplacement du personnage dans le tableau.
- En plus elle sert aussi à compter le nombre de tours utiles pour le calcul du score.

Signature de fonction :

```
1 void deplacememt(int direc, int ** plateau, int taille, int *x, int *y, int *
    nb_tours);
```

detect-fin(...) :

- Cette fonction sert à détecter une fin de jeu.
- Elle renvoie 1 si tous les objectifs ont été complété.
- Elle renvoie 0 si la partie est toujours en cours.
- Elle renvoie -1 si le joueur est bloqué.

Signature de fonction :

```c
int detect_fin(int **plateau, int taille);
```

2.3 Sauvegarde/Chargement

Ici, nous détaillerons comment fonctionne le système de sauvegarde et de chargement de la sauvegarde et d'un niveau.
Les niveaux sont stockés dans le répertoire **./maps** et sont des fichiers du nom **"map$x$.txt"** avec x étant le numéro du niveau. Chaque fichier contient une suite de 400 entiers représentant la map de jeu.
Le fichier **save.txt** est aussi dans le même répertoire et contient la dernière sauvegarde faite par le joueur (le fichier n'existe pas si le joueur n'a jamais sauvegardé). Le fichier est de la même forme que ceux contenant les maps, hormis qu'au début de celui-ci sont aussi stockés le numéro du niveau, le temps écoulé lors de la partie et le score de la partie.

save(...) :
- Cette fonction permet de sauvegarder la progression dans le fichier **save.txt**.
- Elle note le temps auquel la sauvegarde a été faite, le nombre de tours écoulés durant la partie, et le niveau.
- Elle renvoie un entier afin d'indiquer la réussite ou non de la sauvegarde.

Signature de fonction :

```c
int save(int** plateau, int niveau, int time, int nb_tours);
```

charger(...) :
- Cette fonction permet de charger un niveau.

Signature de fonction :

```c
void charger(int *num_niveau, int **plateau, int *x, int *y, int charge_save,
    int *time, int *nb_tours);
```

if_file_exist(...) :
- Cette fonction permet de contrôler si un fichier existe.

Signature de fonction :

```c
int if_file_exist(char *fichier);
```

2.4 Menu/Niveaux

Ici, nous allons détailler les fonctions utilisées pour l'affichage du menu et le déplacement entre les niveaux lors de la partie.

menu(...) :

- Cette fonction affiche le menu et permettant au joueur de choisir ce qu'il veut faire.
- Les options de joueur sont :
  - Si le joueur appuye "1", il va commencer une nouvelle partie.
  - Si le joueur appuye "2", il va changer la partie.
  - Si le joueur appuye "3", les régles de jeu seront affichées.
  - Si le joueur appuye "4", il quitte le jeu.

Signature de fonction :

```
1 void menu(int ** plateau, int *score_tab);
```

lance-niveau(...) :

- Cette fonction permet de lancer le niveau voulu.
- *charger_niveau* permet de savoir si on charge le fichier sauvegardé ou non.

Signature de fonction :

```
1 void lance_niveau(int num_niveau, int ** plateau,int charger_niveau, int *
    score_tab);
```

fin-niveau(...) :

- Cette fonction gére la transition entre les niveaux.
- Si l'utilisateur a terminé un niveau, le programme lui demande s'il veut continuer au niveau suivant ou non.
- Si l'utilisateur a terminé le jeu le programme lui demande s'il veut revenier au 1er niveau ou retourner au menu.
- Si l'utilisateur a perdu le niveau, le programme lui demande s'il veut retenter le niveau ou retourner au menu.

Signature de fonction :

```
1 void fin_niveau(int fin,int num_niveau, int **plateau, int *score_tab);
```

regles() :

- Cette fonction affiche tous les règles de jeu et les instructions.

Signature de fonction :

```
1 void regles();
```

2.5 Chronomètre

Ici, nous allons expliquer les fonctions permettant le fonctionnnement du chronomètre ainsi que son affichage. A noter, que le chronomètre n'est pas en temps réel mais qu'il se met à jour à chaque input par l'utilisateur.

get_time(...) :

- Cette fonction renvoie dans une variable **time_t**, le moment où elle est appelée.
- **time_t** représente le temps écoulé depuis le 1^{er} janvier 1970 à 00 :00 :00.

Signature de fonction :

```
1 time_t get_time();
```

affiche_timer(...) :

- Cette fonction affiche le chronomètre lors de la partie.
- Il est de la forme **heures :minutes :secondes**. Pour chaque, deux chiffres sont affichés.
- Le paramètre *start* contient le moment auquel la partie a commencé.
- Quand la fonction est appelée, on fait appel à la fonction **get_time**. On fait ensuite la différence avec la valeur contenue dans *start*. On obtient un résultat en secondes.
- Le paramètre *decalage* contient la valeur obtenue dans le fichier sauvegarde. Comme ça le chronomètre commence aura au commencement la même valeur que lors de la sauvegarde.

Signature de fonction :

```
1 void affiche_timer(time_t start, int *diff_int, int decalage);
```

2.6 Score

Ici, nous détaillerons la fonctionnalité du score et sa mise en place dans le jeu.
Un fichier **score.txt** permet au joueur de voir son meilleur score à chaque niveau. Celui-ci est formaté et initialisé à 0 pour chaque score. Lorsque le joueur termine un niveau et qu'il a battu son meilleur score, le fichier est mis à jour.
Le calcul du score se fait par la formule suivante :

score = 1000*num_niveau - (temps_ecoule*nombre_tours_ecoules)

calcule_score(...) :
- Cette fonction calcule le score à un instant t de la partie selon la formule vu au-dessus.

Signature de fonction :

```
1 int calcule_score(int nb_tours, int temps, int num_niveau);
```

get_best_score(...) :
- Cette fonction récupère les scores stockés dans le document **score.txt** et les range dans le tableau passé en paramètre.

Signature de fonction :

```
1 void get_best_score(int *tab);
```

maj_best_score(...) :
- Cette fonction permet quant à elle de changer le score du niveau passé en paramètre dans le fichier **score.txt**.
- La valeur est aussi remplacée dans le tableau.

Signature de fonction :

```
1 void maj_best_score(int* score_tab, int best_score, int num_niveau);
```

fichier_score(...) :
- Cette fonction crée le fichier **score.txt**, utile s'il est supprimé par erreur.

Signature de fonction :

```
1 void fichier_score();
```

Table des figures

1 Jeu de Sokoban 1