Jeu du Sokoban

Projet Programmation Avancée

par Léo Peyronnet

Janvier 2023

Table des matières

1 Du jeu en tant que concept

1.1 Rappel des règles du jeu

Sokoban est un jeu de puzzle dans lequel le joueur doit pousser des caisses sur des cibles. Voici comment le jeu fonctionne :

- Le joueur peut se déplacer dans quatre directions : haut, bas, gauche, droite.
- Le joueur doit pousser les caisses sur les cibles, mais il ne peut pousser qu'une caisse à la fois et ne peut pas tirer une caisse.
- Les caisses ne peuvent être poussées que sur des espaces vides ou sur des cibles. Elles ne peuvent pas être poussées contre des murs ou d'autres caisses.
- Le joueur doit utiliser sa stratégie et ses habiletés de résolution de problèmes pour déplacer toutes les caisses sur les cibles dans le niveau le plus efficacement possible.
- Le jeu se termine lorsque toutes les caisses ont été déplacées sur les cibles ou lorsque le joueur abandonne.

1.2 Modélisation du jeu

À partir de ces règles, nous avons du faire des choix arbitraires pour la représentation des différents concepts que prend en compte le jeu :

1.2.1 Terrain/Carte

Pour modéliser le terrain de jeu, nous avons fait le choix de l'afficher dans le terminal avec les symboles suivants :

- Les murs sont représentés par le symbole "#".
- Les caisses sont représentés par la lettre "O".
- Les cibles sont représentés par la lettre "x".
- Les caisses se trouvant sur une cible sont représentés par le chiffre "0".
- Le joueur est symbolisé par la lettre "P".

1.2.2 Contrôles du personnage

Pour pouvoir déplacer le personnage sur le terrain dans les quatre directions possibles, nous avons choisi les contrôles suivants :

- Déplacement vers le haut : touche h.
- Déplacement vers le bas : touche b.
- Déplacement vers la gauche : touche g.
- Déplacement vers la droite : touche d.

L'utilisateur a également la possibilité d'abandonner à tout moment la partie, il lui suffit de rentrer la lettre 'a' dans le terminal.

2 Au programme informatique

2.1 Structuration des données

Maintenant que les éléments du jeu sont modélisés, il faut les structurer en données compréhensible pour un ordinateur. Nous allons donc utiliser la fonctionnalité "struct" du langage C afin de regrouper plusieurs éléments/variables proches sémantiquement dans un même objet.

2.1.1 terrain

Un terrain de jeu de Sokoban pour être représenté par un tableau à deux entrées. Ainsi, en C, nous inclurons dans notre structure "terrain" un pointeur de pointeurs de caractères dans le but d'allouer dynamiquement un tableau 2D plus tard dans le programme.

Afin de délimiter la taille de ce tableau, nous ajoutons également à cette structure deux entiers représentant le nombre de lignes et de colonnes que devra faire le tableau.

Cette structure possède également un dernier entier correspondant au nombre de cibles présentes sur le terrain. Nous avons donc :

```
typedef struct terrain terrain;
struct terrain{
    char ** data;
    int nbLigns;
    int nbCols;
    int nbCibles;
};
```

2.1.2 perso

Notre structure "perso" regroupe l'ensemble des variables définissant le personnage jouable sur le terrain. Ainsi, elle possède une paire d'entiers permettant de déterminer respectivement sur quelle ligne et sur quelle colonne du terrain se trouve notre personnage.

Elle possède également un troisième entier $\in \{0, 1\}$(booléen) permettant de savoir si le personnage se trouve sur une cible ou non. Cet entier se justifie par le fait que notre modélisation ne comprend pas de caractère pour le cas où le personnage se trouve sur une cible (cela aurait pu être "R" par exemple). Nous avons donc :

```
typedef struct perso perso;
struct perso{
int lign;
int col;
short int surCible;
};
```

2.1.3 partie

Enfin, nous avons fait le choix de regrouper nos deux structures définies ci-dessus dans une ultime structure "partie". Elle possède donc un pointeur vers

une structure "terrain" et un autre pointeur vers une structure "perso". Ces pointeurs seront alloués à l'initialisation d'une partie.

La structure "partie" possède également un pointeur de caractères qui permettra que garder en mémoire le nom de la partie. Elle possède également un entier "puntos" indiquant le nombres de cibles couvertes par une caisse. Nous avons donc :

```
typedef struct partie partie;
struct partie{
    terrain *terrain;
    perso *perso;
    int puntos;
    char *nom;
};
```

2.2 Structure du programme

2.3 Fonctions d'affichage

2.4 Fonction partieSokoban()

3 Conclusion