Projet de Jeu de Taquin en C

Jean Barekzai, Saphir Gobbi

27 décembre 2023

Table des matières

1 Introduction

Le jeu de taquin, également connu sous le nom de jeu de quinze, est un puzzle classique qui a captivé l'intérêt de nombreux amateurs de jeux de société. Ce projet vise à mettre en œuvre une version informatique du jeu de taquin. L'objectif principal est de manipuler un plateau composé de 4 lignes et 4 colonnes, comprenant 15 pavés numérotés de 1 à 15, avec une case laissée libre.

Le défi consiste à réorganiser les pavés désordonnés sur le plateau pour parvenir à une disposition où les nombres sont triés de manière croissante, laissant ainsi la case vide à un emplacement spécifique. Seuls les pavés adjacents à la case libre peuvent être déplacés sur cette dernière, autorisant les déplacements horizontaux ou verticaux.

Ce projet combine des aspects de logique, de résolution de problèmes et d'algorithmique.

Nous verrons dans un premier temps, les objectifs du projets, puis nous analyserons le programme que nous avons crée ainsi que ses fonctionnalités avancées, et enfin nous conclurons avec les possibles améliorations à apporter au projet.

2 Objectifs du Projet

Les objectifs de ce projet sont les suivants :
— Définir une structures de données et les fonctions associées.
— Gestion des configurations et chargement depuis un fichier.
— Généralisation du jeu.
— Fonctionnalités Avancées (au choix).
— Utilisation d'un Makefile.

3 Conception du Projet

3.1 Structure de donnée

Nous avons commencé la conception du projet du Jeu de Taquin en choisissant de s'organiser de manière coordonné en utilisant un dépot Git.

Puis nous devions définir une structure de donnée adaptée aux besoins du programme.

Nous avons donc choisit la structure de données suivante :

```
struct grille {
        int **tab;
        int taille;
        int i_libre;
        int j_libre ;
};
typedef struct grille grille ;
```

Cette structure nous permet de créer un tableau d'entier en 2 dimensions, de connaitre la taille de ce tableau, et enfin de savoir la ligne et la colonne de la case vide. Les deux dernières variables seront actualisé au fur et a mesure, selon les actions du joueur. En gardant en mémoire la position de la case libre, on s'économise de nombreux parcours de tab. En effet, la position de la case libre dans le taquin est necessaire pour un certain nombre d'action ou d'évaluation sur la grille. Par exemple pour determinner si on peut le résoudre ou tout simplement effectuer un mouvement.

3.2 Fonctions utilisées

De plus, le programme fait appel aux fonctions suivantes :

— `void displayGrille(grille* G)` : Affiche l'état actuel du plateau du jeu.

— `void libereGrille(grille* g)` : Libère la mémoire allouée pour un seul plateau de jeu.

— `void libereTabGrille(grille** g, int taille)` : Libère la mémoire allouée pour un tableau de plateaux de jeu.

— `int ** newTab(int taille)` : Alloue la mémoire pour un nouveau tableau 2D représentant un plateau de jeu.

— `grille * newGrilleID(int n)` : Crée un nouveau plateau de jeu avec les nombres disposés en ordre croissant et une case vide. Initialise le plateau en fonction de la taille donnée n.

— `int * possible_moves(grille* g)` : Retourne un tableau indiquant les mouvements possibles (haut, bas, gauche, droite) en fonction de la position actuelle de la case vide.

— `void moving(grille* g, char direction)` : Effectue un déplacement dans la direction spécifiée ('h', 'b', 'g', 'd') en échangeant la case vide avec la case adjacente correspondant au mouvement. A noter qu'on n'évalue pas la légalité du mouvement dans cette fonction. Il faut donc veiller à le faire en amont avec possible_moves.

— `void melange(grille* g)` : Mélange le plateau du jeu en effectuant un nombre aléatoire de mouvements valides.

— `int verif( grille* user_G)` : Vérifie si le joueur a trouvé la matrice ID (état final du jeu), renvoie 1 si oui, 0 sinon.

— `grille ** import_grille(char* nom_fichier)` : Importe les plateaux de jeu à partir d'un fichier texte formaté comme suit. Le(s) premier(s) caractère(s) doivent être le nombre de grilles dans le fichier suivit d'un '.'. Ensuite, chaque plateau est défini par sa taille suivi d'un ";" et de ses éléments, chacun suivi d'une virgule. "2.2 ;1,0,3,2,3 ;1,2,3,4,5,6,8,7,0," est un exemple de formatage conforme avec un taquin de taille 2 et un taquin de taille 3.

— `void play( grille* new_G)` : Demande les mouvements de l'utilisateur ('h', 'b', 'g', 'd') jusqu'à ce que le joueur quitte ('q') ou atteigne la configuration identitée.

— `play_user_melange()` : Permet à l'utilisateur de jouer au jeu de taquin.

Demande la taille du plateau, initialise et mélange le plateau, puis fait appel à la fonction `play() :` afin de faire jouer l'utilisateur.

- `int * getPos(int ** tab, int val, int taille):` Renvoie un tableau d'entier de taille 2 contentant la ligne est la colonne de la première occurence de l'entier val dans le tableau 2D d'entiers tab. Si val n'est pas dans tab on retourne NULL.
- `int resolvable(grille * g):` Indique si la grille est résolvable ou non en utilisant l'égalité des parités entre celle du nombre de transposition(échange de cases adjacentes ou non, vide ou non) necessaires pour arriver au taquin ID et une parité associé à la case vide qui correspond à celle du nombre de mouvements (eux légaux) pour replacer la case vide en bas à droite. Il est interessant de noter que les taquins générés avec `melange` seront toujours resolvables car obtenu par une suite de mouvement légaux. En revanche si on importe le taquin de Loyd par exemple, celui-ci est bien non résolvable.
- `int menu():` Affiche un menu à l'utilisateur, lui permettant de choisir entre :
  1. Jouer une grille de taille demandé qui aura été mélangé par le programme ou jouer une grille importée si on a importé des grilles.
  2. Importer, via un fichier externe, une ou plusieurs grille dans le jeu et de les jouer avec l'option 1 du menu.
  3. Quitter le programme.
- `main() :` Fonction principale.

3.3 Création d'un Makefile

Par la suite, nous avons mis en place un fichier `Makefile` pour faciliter la compilation du programme.

```
CC = gcc
CFLAGS = -Wall
TARGET = taquin
SRC = main.c

all: $(TARGET)

$(TARGET): $(SRC)
	$(CC) $(CFLAGS) -o $(TARGET) $(SRC)

clean:
	rm -f $(TARGET)
```

Ainsi, en entrant la commande **make** (sous linux) dans le répertoire où se trouve le fichier `main.c` et le fichier `Makefile` (les deux doivent être dans le même répertoire), la compilation donnera un executable nommé **taquin**. En détail on

peut voir dans ce fichier Makefile :

- **CC :** Le compilateur que l'on utilise (ici, `gcc`).
- **CFLAGS :** Les options de compilation (-Wall pour activer les avertissements).
- **TARGET :** Le nom de l'exécutable que l'on souhaite générer (ici, `taquin`).
- **SRC :** Notre fichier source (ici, `main.c`).
- **all :** La cible par défaut, qui dépend de la cible `$(TARGET)`.
- **$(TARGET) :** La règle pour créer l'exécutable à partir du fichiers source.
- **clean :** Une cible pour supprimer les fichiers générés lors de la compilation.

4 Conclusion et améliorations

En conclusion, à travers l'implémentation de la structure `grille` et des fonctions associées nous avons modélisé et implémenté le jeu de taquin. La fonction `menu()` interface tout les différents éléments que nous avons modélisés puis implémentés. Depuis celle-ci, on peut importer des grilles depuis un fichier formaté, jouer avec ces grilles ou d'autres générées par un nombre aléatoire de mouvements légaux comme spécifié dans le sujet. De plus, les fonctionalités supportent des grilles générales (de taille n>2).

Finalement, un Makefile est adjoint dans le fichier de rendu pour automatiser la compilation. Dans l'ensemble nous pensons avoir répendu aux objectifs que nous nous étions fixés au début de ce projet.

Cependant, le travail que nous proposons est assurément perfectible. Par exemple, nous avons implémenté la fonction `resolvable(grille * g)` pour aider le joueur dans la résolution d'un taquin en lui permettant de savoir si c'est une tâche réalisable.

Il manque cependant une fonctionalité que nous aurions aimer lui adjoindre, à savoir la possibilité de demander un indice sous la forme d'un mouvement vers la résolution. Si le joueur voulait résoudre le taquin il aurait pu le faire en demandant des indices jusqu'à la résolution complète. La stabilité de la fonction `menu()` aurait egalement pu être améliorée. En l'état, nous avons constaté qu'entrer des caractères autre que des chiffres peut poser des problèmes et nécessiter un arret anticipé du programme.