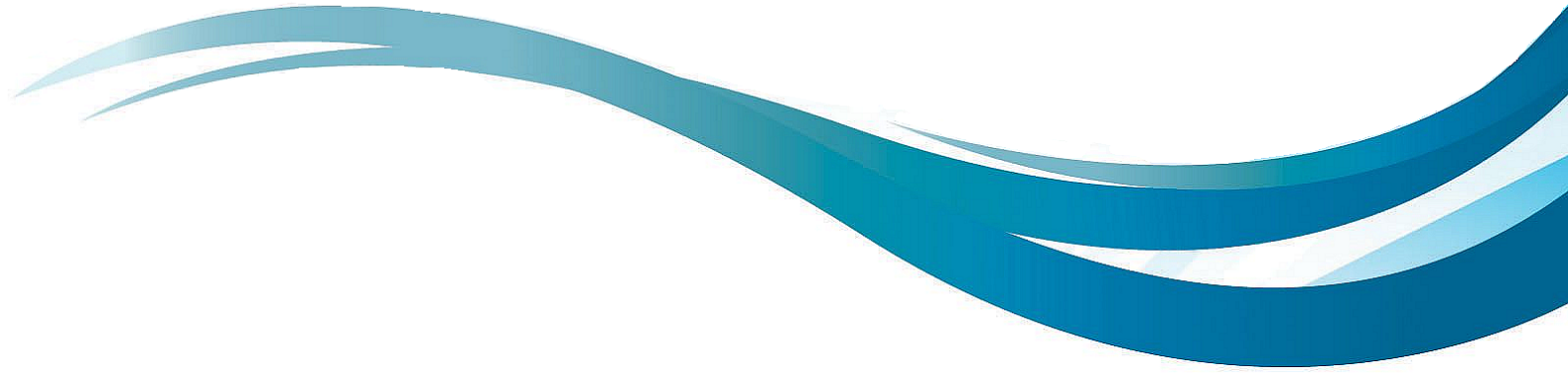
UNIVERSITÉ
Clermont Auvergne

Rendu Projet : Programmation Avancée
Le Jeu de Taquin

BALLEJOS Lilian
GUILLAUMIN Maxime
Université Clermont Auvergne
Année Universitaire 2020 - 2021



Enseignant TP : WAGNER Nicolas
Enseignant CM : LIMOUZY Vincent
Date du rendu : 15 janvier 2021

UCA
49 bd François Mitterrand - CS 60032 - 63001 Clermont-Ferrand
Tel : 04 73 17 79 79
Site web : uca.fr

Sommaire

1 Introduction

Nous allons vous présenter la modélisation ainsi que l'implémentation de notre jeu de **Taquin** en langage **C**. Nous expliquerons dans ce rapport comment lancer le jeu et les fonctionnalités disponibles pour le joueur. Nous détaillerons également les différents choix que nous avons fait et le contenu des fichiers permettant de faire tourner le jeu.

2 Lancement du jeu et fonctionnalités

2.1 Compilation et premier lancement

Vous pouvez compiler automatiquement tous les fichiers permettant le fonctionnement du jeu avec le **makefile** présent. Le nom de l'exécutable est **Jeu** : il suffit d'exécuter ce fichier (**./Jeu**) après compilation et le menu se lance dans le terminal.

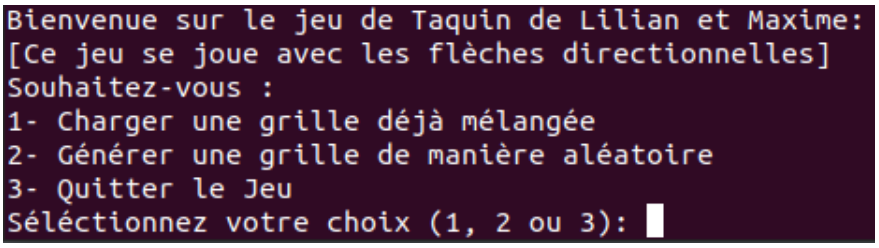


FIGURE 1 – Menu du jeu

2.2 Fonctionnalités

Vous pouvez charger une grille pré-remplie dans le dossier **Patern** (en renseignant le nom du fichier d'entrée) ou bien une grille de taille n générée aléatoirement. L'affichage graphique du jeu se fait dans une nouvelle fenêtre où vous pouvez commencer à jouer en utilisant les touches directionnelles pour effectuer les différents déplacements. Si vous abandonnez (le jeu

n'est quand même pas si simple à résoudre surtout sur des grilles de grande taille), il suffit de cliquer sur la croix en haut à droite pour fermer le jeu (nous vous conseillons tout de même de le résoudre au moins une fois, une petite surprise vous attend). À noter que vous pouvez redimensionner la fenêtre avec la taille de votre choix (pour s'adapter à la résolution que vous préférez), l'affichage sera adaptée en fonction, en modifiant la commande de préprocesseur :

```
1 #define DIM_FENETRE 800
```

dans le fichier **main.c** et **affichage.c**. Notez de plus que par la suite le programme adaptera automatiquement son affichage des grilles et la taille du texte à la dimension de fenêtre de votre choix.

3 Modélisation et structures utilisées

3.1 La grille

Notre grille sera stockée dans un tableau d'entiers **tab** à deux dimensions. La case vide correspond à la case de **tab** ayant la valeur **'0'**. L'affichage de notre tableau sous la forme d'une grille de taquin se fait à l'aide de la bibliothèque **SDL**. Ainsi, un déplacement se traduit par un échange de la valeur de la case **tab[i][j] = 0** avec celle de la case **tab[k][l]** qui est à gauche,à droite, en haut ou en bas (selon le déplacement effectué). On génère une nouvelle grille avec la **SDL** à partir de notre tableau actualisé si notre déplacement a bien eu lieu.

3.2 Gestion des entrées

Nous avons vu plus tôt que notre programme peut charger une grille déjà remplie stockée dans un fichier. Pour faire cela notre programme à l'aide d'un premier fscanf va récupérer le premier entier **n** du fichier qui correspond à la taille de la grille sauvegardée. Ensuite, grâce à cette valeur, on alloue notre tableau de **n entiers** et on récupère les différents nombres séparés par un **espace** à l'aide de fscanf.

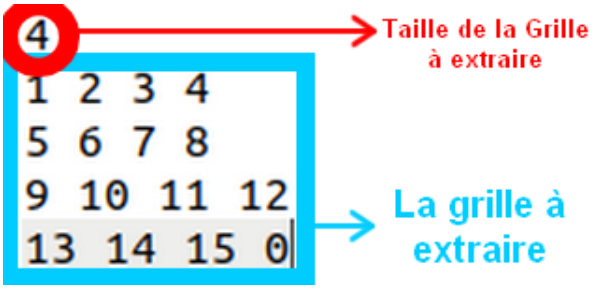


FIGURE 2 – Cas pour la matrice Identité de taille 4

Si le joueurs choisit de jouer sur une grille générée aléatoirement, on lui demande de saisir la taille de la grille, on alloue notre **tab** d'entiers et on le rempli avec l'identité. On effectue ensuite sur ce tableau un grand nombre de déplacements (on a choisit arbitrairement entre 1000 et 2000 déplacements) aléatoires et on obtient un tableau mélangé à partir duquel on affiche la grille du jeu.

3.3 Affichage

Nous utilisons trois SDL_Surface pour afficher la grille du jeu : **ecran** qui est notre fenêtre, **rectangle** qui correspond à nos différentes cases et **texte** sur laquelle on va copier les différentes

valeurs de notre tableau l'aide de la librairie **SDL_TTF** . Les surfaces **rectangle** et **texte** sont copiées autant de fois qu'il y a de cases sur notre surface **ecran**. À chaque copie, la taille de **rectangle** reste fixe mais on change ses coordonnées (et couleur pour une question d'esthétisme) et on change aussi les coordonnées de **texte** pour que le nombre soit bien affiché dans la bonne case.

4 Implémentation

Nous avons quatre fichiers **C** regroupant les différentes fonctions permettant au jeu de fonctionner.

4.1 Tableau 2D

Le fichier **tableau.c** regroupe toutes les fonctions relatives à notre tableau représentant la grille de jeu.

Nous avons implémenter une fonction **Allouetab** qui prend un entier **dim** en paramètre et alloue dynamiquement la mémoire pour un tableau 2D de cette dimension. De même, la fonction **freeTab** prend en paramètre un tableau tab et un entier dim et permet de libérer la mémoire allouée au tableau lorsque le jeu se termine.

On génère la grille identité à l'aide de la fonction **GenereId** qui prend en paramètre un entier dim et retourne le tableau identité de cette dimension. Pour cela, on initialise un compteur à 1 on parcourt notre tableau en affectant à chaque case la valeur du compteur que l'on incrémente. On affecte la valeur '0' à la dernière case.

Nous avons aussi développé la fonction **CompareTab** qui prend en paramètre un tableau 2D d'entiers et renvoie 1 si c'est l'identité, 0 sinon (on compare chaque case du tableau avec la valeur qu'elle est censée contenir pour vérifier si le joueur a gagné). Le mécanisme pour tester les valeurs est très semblable à celui fait dans **GenereId** (on compare la valeur de la case avec un compteur initalisé à 1 qu'on incrémente).

La fonction **ChargementPatern** prend comme arguments une chaîne de caractère (qui sera le nom d'un fichier) et un entier dim et renvoie le tableau 2D contenant la grille du fichier. Pour cela, on ouvre le fichier, on alloue un tableau 2D avec **AlloueTab** et on récupère chaque valeur de notre fichier avec fscanf. Si le fichier n'existe pas on quitte le programme et prévenant l'utilisateur. Il est important de noter que le fichier ne doit posséder **aucune extension** (.txt ....). Il doit juste possèder un nom (voir l'exemple du fichier **id** fourni).

Pour générer une grille aléatoire de taille n, nous avons implémenté la fonction **GenereTabRandom** qui part du tableau identité (renvoyé par **GenereId**) et effectue un nombre (**int NbCoup**) de déplacement de manière aléatoire dans une boucle while : pour cela on tire un nombre aléatoirement entre 0 et 3 à l'aide de la fonction **rand() % 4** et selon la valeur de ce nombre on applique une certaine fonction de déplacement (droite, gauche, haut, ou bas) à notre tableau (ces fonctions seront détaillées plus loin). Une fois que l'on a appliqué **NbCoup** déplacements, on renvoie notre tableau mélangé.

Un fonction d'affiche des tableaux dans le terminal à aussi été implémenté pour nous aider lors de la conception du jeu. Nous vous l'avons laissé au cas où.

4.2 Déplacements

Le fichier **mouvement.c** contient toutes les fonctions permettant d'effectuer les différents déplacements de case. On a d'abord une fonction void **DetectZero** qui prend en paramètre le

tableau 2D, sa dimension ainsi qu'un pointeur sur un entier **x** et un pointeur sur un entier **y**. La fonction cherche la case vide de notre grille (i.e == 0) et affecte à x et y les indices de notre case vide.

On utilise ensuite ces indices dans nos fonctions de déplacement pour intervertir la valeur de la case vide avec celle concernée par notre mouvement. Par exemple si l'on veut effectuer un mouvement à droite (les 3 autres mouvements possibles suivent le même principe à des conditions près) :

```
int MouvDroite(int ** tab, int dim)
{
    int x,y;
    DetectZero(tab,dim,&x,&y);
    if(x - 1 >= 0) //si on a une valeur a gauche du vide
    {
        tab[y][x] = tab[y][x-1];
        tab[y][x-1] = 0;
        return 1;
    }
    else
    return 0;
}
```

On teste aussi dans nos fonction si le mouvement est possible (on ne sort pas de la grille ou si il y a bien une valeur à interchanger) et s'il a été réalisé alors on renvoie 1, 0 sinon. Cela nous permettra par la suite de limiter le nombre d'actualisation de la grille est optimiser la mémoire (on actualisera la fenêtre seulement si il y a eu un mouvement !)

4.3 Implémentation graphique

Le fichier **animation.c** contient toutes les fonctions permettant de gérer l'interface graphique.

La fonction **retrace** prend en paramètre une SDL_ surface (**ecran**), notre tableau 2D, sa dimension, une TTF police et permet de générer la fenêtre correspondant à notre grille de taquin (stockée dans le tableau 2D).

```
SDL_Surface * rectangle = NULL;
rectangle = SDL_CreateRGBSurface(SDL_HWSURFACE, DimCase, DimCase, 32, 0, 0, 0, 0);
SDL_Rect position;
position.x=0;
position.y=0;

/*Creation d'une surface pour ecrire*/

SDL_Surface * texte = NULL;
SDL_Rect positionTexte;
positionTexte.x= DimCase/4;
positionTexte.y= DimCase/4;
SDL_Color textColor;
textColor.r = 0;
textColor.g = 0;
textColor.b = 0;
```

On créé d'abord notre première surface **rectangle** (position (0,0)) sachant que sa dimension est égale à la dimension de notre fenetre / par le nombre de case. On essaie de placer notre surface **texte** sur laquelle on va copier la valeur de la première case de tab au milieu de la surface **rectangle**.

On créé ensuite une surface pour le texte, une position pour cette surface sur l'écran et une variable SDL_Color pour colorer l'écriture qui sera dans notre cas en noir le plus foncé possible en RGB (initialisé à R=0, G=0, B=0)

On parcourt notre tableau dans une double boucle for et on alterne la couleur de notre surface avant de l'appliquer à l'écran (pour l'esthétisme)

```
if(change == 0) //soit une case gris foncé
                {
                    SDL_FillRect(rectangle, NULL,SDL_MapRGB(rectangle->format
    ,200,200,200));
                    change = 1;
                }
                else //soit une case gris clair
                {
                    SDL_FillRect(rectangle, NULL,SDL_MapRGB(rectangle->format
    ,220,220,220));
                    change = 0;
                }
                SDL_BlitSurface(rectangle,NULL, ecran,&position);
```

On change la position x de **rectangle** pour appliquer la case suivante :

```
position.x += DimCase;
```

Lorsqu'on change de ligne :

```
position.y += DimCase;
position.x = 0;
```

Pour afficher les différentes valeurs de notre tableau sur les cases, il est nécessaire de convertir l'entier de notre tableau en chaîne de caractère car la fonction **TTF_RenderText_Blended** ne peut copier que des chaînes de caractères. On utilise donc un buffer **OutilTexte** qui va contenir les caractères de notre valeur (on utilise sprintf pour caster notre nombre entier stocké dans le tableau en chaîne de caractère).

```
if(tab[i][j] != 0) //si l'element du tableau n'est pas la case vide alors on l'
    affiche dans la fenetre
                {
                    sprintf(OutilTexte,"%d",tab[i][j]); //cast int vers str
                    texte = TTF_RenderText_Blended(police, OutilTexte, textColor);
                    SDL_BlitSurface(texte,NULL, ecran,&positionTexte);
                }
```

On actualisera de même les coordonnées de **texte** à l'itération suivante et lorsqu'on a parcouru entièrement notre tableau, on prend en compte toutes les modifications sur notre surface principale qui affichera la grille.

```
SDL_Flip(ecran); //on refresh les changements
```

Nous avons ensuite créé la fonction pause qui attend un événement (que l'utilisateur appuie sur une touche ou ferme la fenêtre) à l'aide de la fonction SDL_ WaitEvent.

```
int Programme_tourne = 1 ;
    SDL_Event event;
    while(Programme_tourne)
```

```
4       {
5           SDL_WaitEvent(&event) ;
6           switch(event.type)
7           {
8               case SDL_QUIT:
9                   Programme_tourne = 0;
10                  break;
11              case SDL_KEYDOWN:
12                  switch(event.key.keysym.sym)
```

Si l'utilisateur appuie par exemple sur la flèche de droite, on appelle notre fonction qui effectue le déplacement à droite et si le mouvement a bien pu être réalisé, alors on appelle **retrace** qui va actualiser la fenêtre à partir du tableau modifié et donc faire notre déplacement.

```
1   case SDLK_RIGHT:
2                           if (MouvDroite(tab,dim))
3                           {
4                               retrace(tab,ecran,dim,police);
5                               if(CompareTab(tab,dim))
6                               {
7                                   return 1; //si il a fini le jeu
8                               }
9                           }
```

Si le tableau est l'identité alors l'utilisateur a gagné, on sort de la fonction en retournant 1 ce qui informera la fonction main que l'utilisateur à complété la grille.
A l'inverse, l'autre sortie possible intervient si le joueur ferme la fenêtre sans terminer le jeu, dans ce cas là, la fonction renvoie 0.

La fonction **EcranWin()** permet d'afficher une fenêtre avec une petite image pour féliciter l'utilisateur de sa victoire. Elle restera ouverte tant que le joueur ne la ferme pas !

Enfin la fonction **Acceuil()** se charge d'afficher notre menu dans le terminal et renvoie le choix du joueur demandé à l'aide de scanf. Si la personne souhaite quitter le jeu dès le menu la fonction quitte simplement le programme.

4.4 Déroulement du programme

Comme vu plus tôt, au lancement du jeu, le menu dans le terminal s'ouvre à l'aide de la première fonction appelée dans notre fichier **main.c** qui est **Acceuil()**. On stocke ensuite la grille de jeu dans notre tableau selon le choix du joueur (chargé depuis un fichier ou généré de manière aléatoire) et on lance la fenêtre graphique de jeu en appelant **retrace** une première fois. Ensuite, on attend que le joueur effectue des déplacements (fonction **pause**) et on rafraîchi la grille si des mouvement sont effectués. On appelle **ecranWin** seulement si la valeur de retour de **pause** est 1.

Enfin, on n'oublie pas de libérer la mémoire à la fin du jeu.

```
1   SDL_FreeSurface(ecran);
2   TTF_CloseFont(police);
3   FreeTab(tab,dim);
4   TTF_Quit();
5   SDL_Quit();
```

Notons que la mémoire des surfaces utilisées pour les cases et écritures sont libérées dans la fonction générant l'écran (**retrace()**). La police, elle, est allouée et libérée dans le main afin de ne pas devoir la ré-allouer à chaque actualisation de la grille.