Rapport de projet Programmation avancée : Taquin

Thomas Ekindy - Paul Nautré
L2 informatique UCA

15 janvier 2021

Chapitre 1

Introduction du sujet

Introduction au Taquin sur Wikipédia :

> Le taquin est un jeu solitaire en forme de damier créé vers 18701 aux États-Unis. Sa théorie mathématique a été publiée par *l'American Journal of mathematics pure and applied* en 1879. En 1891, son invention fut revendiquée par Sam Loyd, au moment où le jeu connaissait un engouement considérable, tant aux États-Unis qu'en Europe. Il est composé de 15 petits carreaux numérotés de 1 à 15 qui glissent dans un cadre prévu pour 16. Il consiste à remettre dans l'ordre les 15 carreaux à partir d'une configuration initiale quelconque.
>
> Le principe a été étendu à toutes sortes d'autres jeux. La plupart sont à base de blocs rectangulaires plutôt que carrés, mais le but est toujours de disposer les blocs d'une façon déterminée par un nombre minimal de mouvements. Le Rubik's Cube est aujourd'hui considéré comme l'un des « descendants » du taquin.

Objectifs :
Notre objectif ici est d'implémenter une version numérique de ce jeu. On aura un damier composé selon une configuration solvable aléatoire et le joueur pourra en déplacer les pièces selon les règles pour aboutir à la formation arrangée. Le jeu comptera le nombre de mouvements effectués par le joueur, lui indiquera quand la partie est terminé et lui proposera de sauvegarder/recharger la configuration à tout moment (ainsi le joueur pourra progresser et évaluer sa progression). L'interface graphique sera faite avec la SDL mais il sera également possible de lancer le jeu en mode console. En mode console, il sera possible de choisir la taille du damier (le jeu est originellement prévu sur une grille 4x4 mais peut être étendu à tout entier).

Chapitre 2

Développement

2.1 Méthode de travail

Nous découpons le développement du projet en deux parties : d'une part l'algorithme qui génère les configurations, assure la mécanique de jeu, vérifie les solutions de l'utilisateur et assure les fonctionnalité auxiliaires comme le système de sauvegarde, d'autre part la surcouche graphique qui permet au joueur de visualiser et de commander le jeu intuitivement. Ces deux parties peuvent être développée simultanément (donc répartie dans le binôme) si l'on commence par se mettre d'accord sur les interactions qui ont lieu entre l'algorithme et la surcouche graphique. Par exemple, celui qui créé le bouton "sauvegarder" n'a pas besoin de savoir précisément comment fonctionne la sauvegarde, il lui suffit de connaître la signature de la fonction correspondante.

2.2 Algorithme

2.2.1 Générateur de configuration

Il faut savoir que sur les $D^2!$ configurations possibles du damier (ou D est la dimension), seule la moitié sont solvables. Il existe un algorithme qui permet de savoir si une configuration donnée est solvable ou non, mais celui-ci est relativement complexe et il peut en vérité être contourné. En effet, une fois les mouvements possibles sur le damier bien définie (sous-section suivante), il suffit de partir du damier ordonné (très facile a générer) et d'effectuer un certains nombres de mouvements aléatoires pour obtenir une configuration aléatoire nécessairement solvable.

La question est de savoir combien de mouvement aléatoire réaliser pour obtenir des grilles biens mélangées et peu redondantes. Nous avons effectué quelques tests et nous avons remarqué qu'après 10000 permutations sur une configuration 4x4, le 1 se retrouvait presque aussi souvent sur la troisième ou quatrième ligne que sur la première ou deuxième. Nous avons pensé que c'était un critère recevable pour dire que le damier était suffisamment mélangé. Par ailleurs ce calcul se fait si vite que l'on observe aucun délai à l'œil nu (le nombre de calculs effectués est seulement proportionnel au nombre de mouvements).

```c
int ** melanger(int D){

    // Creation du tableau
    int **t = 0;
    t = malloc(D * sizeof(int*));
    if (t == 0){  // Gestion d'erreur
        fprintf(stderr, "Erreur : Echec à la creation du tableau (niveau 1)\n");
        exit(1);
    }

    for (int i = 0; i<D; i++){
        t[i] = malloc(D * sizeof(int));
        if (t[i] == 0){ // Gestion d'erreur
            fprintf(stderr, "Erreur : Echec à la creation du tableau (niveau 2)\n");
            exit(1);
        }
    }
    // Remplissage ordonné du tableau
    int c = 1;
    for (int i = 0; i<D; i++)  // Parcours du tableau
        for (int j = 0; j<D; j++){

            t[i][j] = c; // On place chaque case à sa place
            c++;
    }
```

```
26
27      // mélange par permutations aléatoires
28      int n = 10000; //  nombre de tentatives de permutation
29      char d; // deplacement ligne et colonne
30      char direction[] = "bhdg";
31
32      srand (time (NULL)); // initialisation du module alea
33
34      for (int k = 0; k<n; k++){ // n fois
35
36          // On fait en sorte d'avoir un déplacement aléatoire sur la ligne OU sur la colonne (
        pas les deux)
37          d = direction[rand() % 4]; // On choisi une direction au hasard
38          permuter(t, d, D); // On effectue la permutation si possible
39
40      }
41
42      sauvegarder(t, 0, D);
43      return t;
44
45  }
```

../codes/algo.c

2.2.2 Mécanique de jeu

Permutation

Dans le jeu, le joueur peut permuter la case vide et la case à sa gauche, à sa droite, au dessus ou en dessous. Dans tout les cas, il ne peut la pas sortir du damier. Ainsi, quand il commande un mouvement, le programme doit trouver la case vide, déterminer si le mouvement est possible, puis, si oui, l'exécuter. C'est très littéralement ce que fait la fonction permuter. Elle retourne ensuite 1 si une permutation à été effectuée, 0 sinon.

```
1   int permuter(int **t, char d, int D){
2
3       for (int i = 0; i<D; i++)  // Parcours du tableau
4           for (int j = 0; j<D; j++){
5
6               if (t[i][j] == VIDE){ // On trouve la case à vide
7
8                   if (d == 'h' && i-1 >= 0){ // Si on va en haut
9                       t[i][j] = t[i-1][j]; // On permute
10                      t[i-1][j]  = VIDE;
11                      return 1;
12
13                  }
14
15                  else if (d == 'b' && i+1 < D){ // Si on va en bas
16                      t[i][j] = t[i+1][j]; // On permute
17                      t[i+1][j]  = VIDE;
18                      return 1;
19
20                  }
21
22                  else if (d == 'g' && j-1 >= 0){ // Si on va a gauche
23                      t[i][j] = t[i][j-1];  // On permute
24                      t[i][j-1] = VIDE;
25                      return 1;
26
27                  }
28
29                  else if (d == 'd' && j+1 < D){ // Si on va a droite
30                      t[i][j] = t[i][j+1];  // On permute
31                      t[i][j+1] = VIDE;
32                      return 1;
33
34                  }
35
36                  else{ // Si elle n'est pas dans le voisinage
37                      return 0; // On ne peut pas permuter
38                  }
39              }
40          }
41
42      return 0;
43  }
```

../codes/algo.c

Vérification
Chaque mouvement du joueur est susceptible de terminer le jeu (par victoire). Aussi après chaque permutation, on vérifie si le damier est ordonnée. Pour cela il suffit de le parcourir et de vérifier que chaque case est à sa place (jusqu'à la case vide en dernière position).

```c
int verifier(int **t, int D){

    int c = 1;

    for (int i = 0; i<D; i++)  // Parcours du tableau
        for (int j = 0; j<D; j++){

            if (t[i][j] == c) // On verifie si chaque case est bien à sa place
                c++;
            else
                return 0;  // Si une ne l'est pas, le tableau n'est pas solvé
        }
    return 1; // Sinon, il est solvé
}
```

../codes/algo.c

2.2.3 Sauvegarde

Il était attendu dans le sujet une fonctionnalité impliquant l'utilisation de fichiers. Nous avons donc opté pour un système de sauvegarde par lequel le joueur peut enregistrer une configuration à un instant donné (il écrase alors la précédente sauvegarde) et la restaurer plus tard. Lorsqu'une configuration est générée en début de partie, elle est automatiquement sauvegardée.

Le système se décompose en deux fonctions : une pour écrire la sauvegarde, l'autre pour la restaurer. Les deux fonctions sont syntaxiquement et sémantiquement très similaires. Il faut noter qu'il serait problématique d'essayer d'importer une configuration de taille 4x4 depuis un fichier utilisé pour la sauvegarde d'une configuration 5x5 par exemple. Mais si on s'assure de ne jamais avoir ce type d'erreur dans notre programme, on n'a pas besoin de formater le fichier de façon complexe, on se contente d'écrire une valeur par ligne.

```c
void sauvegarder(int **t, int nbCoup, int D){
    FILE * f = NULL;
    f = fopen("sauv.txt", "w"); // Creation/ouverture du fichier
    if (f == NULL){ // Gestion d'erreur
        printf("Erreur : Echec creation de la sauvegarde\n");
        exit(1);
    }

    fprintf(f, "%d\n", nbCoup); //Sauvegarde du nombre de coup
    for (int i = 0; i<D; i++) // Parcours du tableau
        for (int j = 0; j < D; j++)
        {
            fprintf(f, "%d\n", t[i][j]); // écriture d'une case par ligne
        }
    fclose(f); // Fermeture du fichier
}


int recharger(int **t, int D){
    char ligne[4];
    FILE * f = NULL;
    int nbCoup;

    f = fopen("sauv.txt", "r"); // Ouverture du fichier
    if (f == NULL){ // Gestion d'erreur
        printf("Erreur : Echec creation de la sauvegarde\n");
        exit(1);
    }

    fgets(ligne, 4, f); // Récuperation du nombre de coup
    sscanf(ligne, "%d\n", &nbCoup); // Extraction de la valeur

    for (int i = 0; i<D; i++) // Parcours du tableau
        for (int j = 0; j < D; j++)
        {
            fgets(ligne, 4, f); // Récuperation de la ligne correspondante
```

```
37              sscanf(ligne, "%d\n", &t[i][j]); // Extraction de la valeur
38          }
39      fclose(f);
40
41      return nbCoup; // On retourne le nombre de coup à l'instant de la sauvegarde
42
43  }
```

../codes/algo.c

2.2.4 Fonctions auxiliaires

Libération de la mémoire

Supposons que notre joueur aime beaucoup le jeu et y soit très bon : il pourrait gagner beaucoup de partie à la suite et donc générer beaucoup de configuration soit demander au programme de créer beaucoup de tableau ce qui pourrait finir par encombrer considérablement la mémoire de sa machine. Il est donc nécessaires de libérer la mémoire au fur et à mesure. On veut que à un instant donné, le programme n'exploite qu'une configuration (une seconde est sauvegardée dans un fichier). Pour cela on créé une fonction qui détruit le tableau à la fin de chaque partie et qui libère la mémoire correspondante. Ce type de fonction est très classique, le tout est de l'utiliser au bon moment. On aurait aussi pu avoir un seul tableau qui aurait pris la nouvelle configuration à chaque début de partie mais il nous semblait plus élégant d'avoir une fonction qui génère, remplit et mélange un tableau toute seule.

```
1  void liberer(int **t, int D){
2      for (int i = 0; i < D; i++)
3          free(t[i]);
4      free(t);
5  }
```

../codes/algo.c

Mode console

On voulait que le jeu soit utilisable en mode graphique ou en mode console. On a donc créé deux fonctions, chacune correspondant à un mode. L'exécution du mode graphique est détaillée dans la section suivante. Pour le mode console, on génère une configuration puis on se contente de demander en boucle à l'utilisateur ce qu'il veut faire (mouvements, sauvegarde, restauration). On affiche à chaque étape le damier (d'où la fonction d'affichage) et on vérifie à chaque tour si la partie est gagnée. Si oui, on félicite le joueur, on efface la mémoire et le programme se termine.

```
1  void console(int D){
2
3      int **t = melanger(D); // Génération du tableau
4      char d;
5      printf("TAQUIN mode console by IZIGANG tm\n"); // Instructions
6      printf("controles : \n");
7      printf("- s : Sauvegarder\n");
8      printf("- r : Recharger la sauvegarde\n");
9      printf("h,b,d,g : Haut, Bas, Droite, Gauche\n");
10
11     int nbCoup = 0; // compteur de coup
12
13     while(!verifier(t, D)){ // Tant que le joueur n'a pas gagné
14         printf("\n\n");
15         afficher(t, D); // Affichage de la grille
16         printf("\n");
17         printf("Vers ou aller ? ( %d coup ) >",nbCoup); // Saisie utilisateur
18         scanf("%c",&d);
19
20         if (d=='s'){ // Si sauvegarde
21             sauvegarder(t, nbCoup, D); // Sauvegarde
22             printf("La grille à été sauvegardée !\n");
23         }
24
25         else if (d =='r'){ // Si restauration
26             nbCoup = recharger(t,D); // Restauration
27             printf("La grille à été rechargée !\n");
28         }
29
30         else { // Sinon (mouvement ou faute de frappe)
31
32             if (!permuter(t,d, D)) // Tentative de permutation
33                 printf("permutation impossible !\n");
34         nbCoup++; // Incrémentation du nombre de coup
```

```
35          }
36
37          scanf("%c",&d);
38          printf("%c\n", d);
39      }
40
41      printf("\n\nBien joué ! \n"); // Si victoire, félicitation
42      liberer(t, D); // Liberation de la mémoire
43
44  }
```

../codes/main.c

2.3 Surcouche graphique

2.3.1 Lancement du mode graphique

Initialisations

Le mode graphique est lancé grâce à l'appel de la fonction "graphique". Dans cette fonction nous initialisons les modules indispensables au bon fonctionnement du jeu, c'est à dire les modules audio et vidéo, le module permettant d'écrire du texte sur une fenêtre ainsi que le module aléa. Cette fonction va jouer la musique de fond et faire tourner en boucle la création de l'interface "menu" tant que l'utilisateur n'aura pas demandé l'arrêt totale du jeu.

```
1  void graphique(){
2
3    if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO)==-1){ //Si l'initialisation des modules audio
       et video échoue
4      fprintf(stderr,"Unable to initialize VIDEO or AUDIO:%s \n",SDL_GetError()); //On écrit un
       message d'erreur
5      exit(EXIT_FAILURE); //On sort du programme
6    }
7
8    if (TTF_Init()==-1){ //Si l'initialisation de SDL_Ttf échoue
9      fprintf(stderr,"Unable to initialize TTF: %s\n",TTF_GetError()); //On écrit un message d'
       erreur
10     exit(EXIT_FAILURE); //On sort du programme
11   }
12
13   if (Mix_OpenAudio(44100,MIX_DEFAULT_FORMAT,2,256)==-1){//Si l'initialisation de SDL_Mixer é
       choue
14     fprintf(stderr,"Unable to open an audio with Mixer: %s",Mix_GetError());//On écrit un
       message d'erreur
15     exit(EXIT_FAILURE);//On sort du programme
16   }
17
18   srand(time(NULL)); //On enclenche le module alea
19
20   Mix_Music * musique=NULL;//On crée un pointeur vers une musique
21
22   SDL_Thread* threadMusic = SDL_CreateThread(randomMusic,musique);//On lance un thread qui va
       jouer des musiques aléatoires en fond
23
24   int stop=0; //On initialise la fonction qui va décider de si le jeu s'arrête ou pas
25   while(stop==0||stop==-1){//Si stop est à 0 ou -1 le jeu continue et on relance menu()
26     stop=menu(); //La fonction menu renvoi soi 0, soi -1, soi 1
27   }
28
29   done = 1;//Le jeu est terminé donc on met la variable globale done à 1 pour stopper le
       thread
30
31   SDL_WaitThread(threadMusic,NULL); //On attend que le thread se stop grâce à la variable done
32
33   Mix_FreeMusic(musique);//La musique n'est plus utilisée donc on libère la mémoire
       correspondante
34
35   Mix_Quit(); //On quitte SDL_Mixer
36   TTF_Quit(); //On quitte SDL_Ttf
37   SDL_Quit(); //On quitte SDL_Quit
38 }
```

../codes/graphique.c

Multi-threading

Pour faire tourner la musique de fond sans cesse il aurait fallut vérifier assez régulièrement si une musique

était en train d'être jouée, or si l'utilisateur ne saisie aucune entrée clavier ni souris le programme resterait en pause et aucune nouvelle musique ne serait jouée après la fin de la précèdente. Utiliser la fonction "PollEvent" au lieu de "MainEvent" dans la boucle des évènements ferait utiliser le CPU à 100% de performances, ce qui n'est pas très optimisé. Utiliser "PollEvent" et ajouter un délai dans la boucle des évènements aurait poser un gros problème en terme de latence pour l'affichage des boutons en surbrillance. Puisque "MainEventTimeout" n'existe que sur SDL2.0 nous avons dû utiliser le multi-threading pour faire tourner la musique en fond du jeu sur un thread secondaire. L'état d'une variable "done", exceptionnellement globales aux fonctions graphiques, est donc modifié par le thread principal et passe de 0 à 1 pour que le fonction du thread secondaire puisse savoir à quel moment elle doit se terminer.

Thread secondaire

Dans le thread secondaire tournera une fonction ("randomMusic") qui jouera de la musique. Toutes les secondes elle vérifiera qu'il y a bien une musique qui est en train d'être jouée, si on ne mettait pas ce délai de 1 seconde le CPU fonctionnerait à 100% pour rien. Cette fonction lit aussi l'état de la variable globale "done" pour savoir si elle doit se terminer ou non afin de pouvoir terminé le thread principale en toute sécurité.

```c
int randomMusic(void * musique){
  Mix_Music * newMusique = (Mix_Music*) musique;//On cast musique pour pouvoir l'utiliser en tant que pointeur vers une musique
  Mix_VolumeMusic(65); //Le volume de base est à 128, ce qui était trop fort donc on le met à 65
  while (done==0){ //Tant que done==0 donc tant que le main thread tourne donc tant que le jeu n'est pas terminé
    SDL_Delay(1000); //On delay pour éviter de faire souffrir le CPU avec une boucle qui se répète trop vite pour rien. La boucle va donc se répéter toutes les secondes
    if (Mix_PlayingMusic()==0){//Si aucune musique ne joue actuellement
      char nomFichier[14];//On crée une chaine de char qui va contenir le chemin d'une nouvelle musique
      sprintf(nomFichier,"music/%i.mp3",rand()%54); //On met dans cette chaine le chemin d'une nouvelle musique aléatoire, il y en a 53 donc on utilise %54
      Mix_FreeMusic(newMusique); //On libère la mémoire allouée par une potentielle musique pointée par newMusique
      newMusique=NULL;//On set le pointeur à NULL, si l'allocation de la ligne suivante ne s'effectue pas ne pointeur ne pointera pas n'importe où
      newMusique= Mix_LoadMUS(nomFichier);//On alloue de la mémoire, on y met la musique aléatoire et on pointe cette case avec newMusique
      Mix_PlayMusic(newMusique,0); //On lance la musique, elle ne se répète pas
    }
  }
  return 0;
}
```

../codes/graphique.c

2.3.2 Affichage et fonctionnement du menu

Initialisations et affichages

Une fois qu'un appel à la fonction "menu" a été effectué, le programme va afficher le menu. Pour se faire, on va initialiser une fenêtre de menu avec une image de fond, créer une liste de boutons et les afficher sur la fenêtre. A partir du moment où le menu est affiché, on lance la fonction qui contient la boucle des évènements.

```c
int menu(){

  SDL_Surface * screen=initialisation(0,480,480); //On appelle initialisation() pour créer la fenêtre et la pointer avec screen

  SDL_Surface ** button = createRect(8,"button/button"); //On utilise createRect() pour créer une une liste de 8 surfaces correspondant aux boutons "nouvelle partie", "recharger" et "quitter" qui s'appellera button. (8 car il y a les images grises et bleus)

  setButtonMenu(screen,button); // On utilise cette fonction pour afficher les boutons sur l'écran screen

  int stop=eventMenu(screen,button);//Dans stop on va stocker la décision du joueur quant au fait d'arrêter le jeu totalement (1) ou de juste relancer le menu pour arrêter le taquin (0)

  SDL_FreeSurface(screen);//Puisqu'on va redémarrer le menu ou arrêter le jeu, on ferme l'écran donc on libère la mémoire allouée par screen pour l'écran

  freeSurface(button,8);//On libère la mémoire allouée pour tous les boutons dans la liste button.

```

```
15    return stop;//On renvoie 1 pour arrêter le jeu totalement, 0 pour redémarrer le menu et
        juste arrêter le taquin
16  }
17
18  SDL_Surface * initialisation(int n,int width,int height){
19
20    SDL_Surface * screen=NULL; //screen va contenir la fenêtre
21    if ((screen= SDL_SetVideoMode(width,height,32,SDL_HWSURFACE))==NULL){
22      fprintf(stderr,"Ok erreur sdl:%s \n",SDL_GetError());
23      exit(EXIT_FAILURE);
24    } //Si on ne parvient pas à créer la fenêtre on stop le prog et un message d'erreur s'envoie
25
26    SDL_Rect pos; //Variable contenant la position de l'image de fond de la fenêtre
27    pos.x=0;//coord axe horizontal
28    pos.y=0;//coord axe vertical
29
30    char nomFichier[22]; //chaine de char contenant le chemin vers l'image de fond
31    sprintf(nomFichier,"image/background%i.bmp",n); //Ecriture du chemin dans la chaine
32    SDL_Surface *background=SDL_LoadBMP(nomFichier);//Creation de la surface pointant vers l'
        image de fond
33
34    SDL_BlitSurface(background,NULL,screen,&pos); //Posage de l'image de fond au coord [0;0]
35
36    SDL_FreeSurface(background);//On libère la mémoire allouée pour l'image de fond puisqu'on l'
        a déjà posé
37
38    return screen;//On renvoi le pointeur vers la fenêtre qui contient désormais un fond
39  }
40
41  SDL_Surface** createRect(int n,char debutNom[20]){
42
43    SDL_Surface ** rect=(SDL_Surface**) malloc(n*sizeof(SDL_Surface*)); //On crée une liste de n
        surfaces
44
45    char nomFichier[27]; //Chaine de char contenant le chemin du type d'image que va contenir la
        liste rect
46
47    for (int i=0;i<n;i++){//Pour le nombre de surface
48      sprintf(nomFichier,"%s%i.bmp",debutNom,i);//On met dans la chaine le chemin du type d'
        image que va contenir la liste rect
49      rect[i]=SDL_LoadBMP(nomFichier);//On fait pointer chaque surface sur leur image
        correspondante
50    }
51
52    return rect;//On renvoi la liste
53  }
54
55  void setButtonMenu(SDL_Surface*screen,SDL_Surface**button){
56
57    SDL_Rect pos; //Variable contenant la position des boutons que l'on va placer
58    pos.x=80; //coord axe horizontal
59    pos.y=80; //coord axe vertical
60    //Position du bouton Nouvelle partie
61
62    SDL_BlitSurface(button[0],NULL,screen,&pos); //Posage du bouton Nouvelle partie aux dernié
        res coordonnées de pos sur la fenêtre
63
64    pos.y=pos.y+125; //Position du bouton Recharger
65
66    SDL_BlitSurface(button[1],NULL,screen,&pos);//Posage du bouton Recharger aux dernières
        coordonnées de pos sur la fenêtre
67
68    pos.y=pos.y+175; //Position du bouton Quitter
69
70    SDL_BlitSurface(button[2],NULL,screen,&pos); //Posage du bouton Quitter aux dernières
        coordonnées de pos sur la fenêtre
71
72    SDL_Flip(screen);//On met en place les modifications pour obtenir une belle fenêtre de menu
73
74  }
```

../codes/graphique.c

Boucle des événements du menu

Dans la boucle des événements le programme va être mit en pause tant qu'aucune entrée clavier ou souris n'a été faite par le joueur. Pour rendre l'interface plus vivante nous avons décidé de mettre en surbrillance les boutons sur lesquels le joueur passe avec la souris ainsi que de jouer un effet sonore lorsqu'il clique sur un des boutons. Pour savoir sur quel bouton le joueur est passé nous interprétons la position de la souris à l'instant du

mouvement de souris et la comparons aux positions des boutons sur l'interface. La surbrillance s'effectue avec l'utilisation de la fonction "colorButton" et chaque clic est traité par la fonction "clickOnMenu".

```c
int eventMenu(SDL_Surface*screen,SDL_Surface**button){
  int stop=0; //Variable à 1 si on arrête totalement le jeu, à -1 si on relance le menu
  int x,y;//Variables qui vont comprendre la position de la souris quand elle bouge
  Mix_Chunk * sonClic= NULL;//Pointeur qui va pointer sur un effet sonore de clic
  sonClic = Mix_LoadWAV("sound_effect/1.wav");//On le fait pointer vers l'effet sonore
  SDL_Event event; //Variable contenant l'input du joueur
  while(stop==0){//Tant qu'on ne décide pas d'arrête ni de relancer le menu
    SDL_WaitEvent(&event); //On attend que le joueur interagisse avec le jeu
    switch(event.type){ //On étudie tous les cas d'input
    case SDL_QUIT: //Si il ferme d'une quelconque façon la fenêtre le jeu s'arrête totalement
      stop=1;//le jeu s'arrête totalement
      break;
    case SDL_MOUSEBUTTONDOWN: //Si il clique à un endroit
      if (event.button.button==SDL_BUTTON_LEFT){//avec le bouton gauche de la souris
    stop=clickOnMenu(sonClic,event.button.x,event.button.y);//On arrête le programme en
    fonction de l'endroit du clic
      }
      break;
    case SDL_MOUSEMOTION://Si il bouge la souris
      x=event.motion.x;//x contient sa position horizontal
      y=event.motion.y;//y contient sa position vertical
      if (x>=80 && x<=400 && y>=80 && y<=144){//si il passe sur le bouton Nouvelle partie
    colorButton(screen,button,4,80,80);//On met le bouton nouvelle partie en bleu
      }

      else if (x>=80 && x<=400 && y>=205 && y<=269){//Si il passe sur le bouton Recharger
    colorButton(screen,button,5,80,205);//On met le bouton Recharger en bleu
      }

      else if (x>=80 && x<=400 && y>=380 && y<=444){ //Si il passe sur le bouton Quitter
    colorButton(screen,button,6,80,380);// On met le bouton Quitter en bleu
      }
      else { //S'il ne passe sur aucun bouton
    setButtonMenu(screen,button);//On réaffiche tous les boutons au cas où certains seraient
    affichés en bleu.
      }
      break;

    default:
    break;
    }
    }
  Mix_FreeChunk(sonClic);//On libère la mémoire allouée pour le son de clic
  return stop;//Si on doit arrêter totalement le jeu on renvoie 1, si on redémarre juste le
    menu on renvoie -1
}
```

../codes/graphique.c

Gestion des clics sur le menu

On compare donc la position de la souris à l'instant du clic aux positions des boutons sur l'interface avec la fonction "clickOnMenu". Un son est joué si le joueur a cliqué sur un bouton. Si le clic tombe sur la position du bouton "Nouvelle partie" on mélange un tableau à l'aide des fonctions de la partie algorithmique du projet et on lance le jeu de taquin avec ce tableau. Si l'utilisateur a cliqué sur le bouton "Recharger", on utilise également les fonctions de la partie algorithmique du projet pour recharger un tableau et un nombre de coups sauvegardés et lancer le jeu de taquin avec ces données. Si l'utilisateur clique sur le bouton quitter, la fonction "clickOnMenu" retourne qu'il a été décidé qu'il fallait arrêter totalement le jeu. Dans le cas où l'utilisateur clique à un endroit neutre on retourne à la boucle des événements.

```c
int clickOnMenu(Mix_Chunk*sonClic,int x,int y){
  int ** tab=NULL; //On crée un pointeur sur un tableau contenant la grille de taquin
  int nbCoups; //Variable contenant le nombre de coups du joueur

  if (x>=80 && x<=400 && y>=80 && y<=144){ //Si il clique sur le bouton Nouvelle partie
    Mix_PlayChannel(-1,sonClic,0);//On active un son de clic
    tab=melanger(4);//On met dans tab une nouvelle grille mélangée de taquin
    jeu(tab,0);//lance le jeu de taquin avec la nouvelle grille et 0 en tant que nombre de
    coups
    return -1;//Renvoi -1 quand le jeu de taquin se termine de sorte à ce que le menu se
    relance entièrement
  }

  else if (x>=80 && x<=400 && y>=205 && y<=269){//Si il clique sur le bouton Recharger
```

```
13      Mix_PlayChannel(-1,sonClic,0);//On lance le son de clic
14      tab=(int**)malloc(4*sizeof(int*));//On alloue de la place pour la grille de 4*4
15      for (int i=0;i<4;i++){
16        tab[i]=(int*)malloc(4*sizeof(int));
17      }
18      nbCoups =recharger(tab,4); //On charge la grille du contenu du fichier de sauvegarde de
        grille et on stock le nombre de coups de la sauvegarde dans nbCoups
19      jeu(tab,nbCoups);//On lance le jeu de taquin avec la grille rechargée et le nombre de
        coups correspondant
20      return -1;//Renvoi -1 pour relancer le menu
21      }
22
23    else if (x>=80 && x<=400 && y>=380 && y<=444){//Si il clique sur le bouton Quitter
24      Mix_PlayChannel(-1,sonClic,0);//On lance le son du clic
25      return 1;//On renvoie 1 pour terminer totalement le jeu
26    }
27
28    else { //Si il clique à un endroit neutre
29      return 0;//La boucle des event continue
30    }
31  }
```

../codes/graphique.c

2.3.3 Affichage et fonctionnement du jeu de taquin

Initialisations et affichages

Comme pour le menu, pour afficher l'interface de jeu de taquin, le programme va initialiser une fenêtre avec une image de fond, créer une liste de cases représentant la grille, une liste de boutons et afficher cette dernière sur la fenêtre. La fonction "jeu" va également faire appel à la fonction "synchro" afin de synchroniser les positions des cases représentant la grille avec le tableau contenant la liste. Une fois que tous les éléments sont affichés on lancer la boucle des événements du jeu de taquin avec la fonction "eventJeu".

```
1   void jeu(int**tab,int nbCoups){
2
3     SDL_Surface * screen=initialisation(2,900,480);//On initialise la fenêtre de jeu dans screen
4
5     SDL_Surface ** rect=createRect(16,"rect/");//On fait pointer les surfaces de la liste rect
        sur les images des cases de la grille de taquin
6
7     synchro(screen,rect,tab,nbCoups);//On synchronise l'affichage des images des cases avec la
        grille de taquin dans tab
8
9     SDL_Surface ** button = createRect(12,"button/button");//On fait pointer les 12 surfaces de
        button sur es images des boutons Sauvegarder, Recharger et Quitter (12 car il y a les
        images des boutons en gris, bleu et bleu foncé)
10
11    setButtonJeu(screen,button,tab);//On affiche correctement les images des boutons
12
13    eventJeu(screen,rect,button,tab,nbCoups);//On lance la boucle qui gère les inputs du joueur
        durant le jeu de taquin
14
15    liberer(tab,4); //Le jeu de taquin est terminé donc on libère la mémoire allouée pour le
        tableau
16    freeSurface(button,12);//On libère la mémoire allouée pour la liste de surfaces pointant sur
         des images de boutons
17    freeSurface(rect,16);//On libère la mémoire allouée pour la liste des images de la grille de
         taquin
18    SDL_FreeSurface(screen); //On libère la mémoire allouée pour l'image de fond
19
20  }
```

../codes/graphique.c

```
1   void setButtonJeu(SDL_Surface*screen,SDL_Surface**button,int **tab){
2
3     SDL_Rect pos;//Variable contenant les positions des boutons que l'on va placer
4     pos.x=540;//Coord axe horizontal
5     pos.y=156;//coord axe vertical
6     //Coord de l'image du bouton sauvegarder
7     if (verifier(tab,4)==0){//Si le jeu de taquin n'est pas encore réussi. S'il est réussi on
        remettrait l'image de sauvegarder en gris alors qu'il serait censé être gris foncé
8       SDL_BlitSurface(button[3],NULL,screen,&pos);//On met l'image de sauvegarder en gris
9     }
10    pos.y=pos.y+104;
11    //Coord de l'image du bouton recharger
```

```
12
13    SDL_BlitSurface(button[1],NULL,screen,&pos);//On place l'image du bouton Recharger en gris
14
15    pos.y=pos.y+104;//Coord du bouton Quitter
16
17    SDL_BlitSurface(button[2],NULL,screen,&pos);//On place l'image du bouton Quitter en gris
18
19    SDL_Flip(screen);//On actualise le visuel
20  }
```

../codes/graphique.c

Boucle des événements du jeu de taquin

Dès le début de la fonction "eventJeu", qui contient la boucle des événements du jeu de taquin, on vérifie que la grille chargée ne soit pas déjà ordonnée, si elle l'est, on affiche directement l'écran de réussite sans jouer d'effet sonore. Une fois dans la boucle, le programme se met en pause tant qu'aucune entrée eclavier ou souris n'a été faite par le joueur. Les effets sonores et la surbrillance des boutons sont gérés de la même façon que pour le menu. Si l'utilisateur appuie sur un touche, on vérifie que le jeu de taquin ne soit pas réussi avant d'effectuer une action. Si le joueur appuie sur une touche directionnelle un effet sonore de permutation se lance et le programme fait permuter la case juste à gauche, à droite, en haut ou en bas de la case vide avec cette dernière si cela est possible. Si après la permutation le taquin est réussi alors un écran de succès s'affiche et un effet sonore correspondant est joué. On fait appel à la fonction "clickOnJeu" dès l'instant où le joueur utilise le clic gauche de sa souris. S'il clique sur un bouton alors celui arbore une couleur plus foncée, c'est pourquoi lorsque l'utilisateur relâche le clic gauche on redonne sa couleur bleu au bouton sur lequel se trouve la souris ou alors on redonne leur couleur normale à tous les boutons si la souris n'est sur aucun bouton.

```
1  void eventJeu(SDL_Surface*screen,SDL_Surface**rect,SDL_Surface**button,int**tab,int nbCoups){
2    int success=verifier(tab,4); //Variable contenant 1 si le jeu de taquin est réussi, 0 sinon
3    int stop=0;//variable contenant 1 si on doit relancer un écran de menu
4    int *nbCoupsPoint=&nbCoups; //On crée un pointeur vers le nombre de coups pour pouvoir le
       modifier dans les autres fonctions
5    if (success==1){//Si le jeu est réussi
6      achievement(screen); //On affiche l'écran de succès
7    }
8    SDL_Event event; //Variable contenant les inputs du joueur
9    Mix_Chunk * sonPermu = Mix_LoadWAV("sound_effect/2.wav");//Pointeur vers l'effet sonore de
       permutation de case
10   Mix_Chunk * sonClic = Mix_LoadWAV("sound_effect/1.wav");//Pointeur vers l'effet sonore de
       clic
11   Mix_Chunk * sonSuccess = Mix_LoadWAV("sound_effect/3.wav");//Pointeur vers l'effet sonore de
        réussite du jeu de taquin
12   int x,y;//Variables contenant la position de la souris quand on la déplace
13   while(stop==0){//Tant que stop n'est pas à 1 on lit les inputs du joueur
14     SDL_WaitEvent(&event);//Attente d'une entrée du joueur
15     switch(event.type){//On étudie les cas suivants
16     case SDL_QUIT: //Si le joueur ferme la fenêtre
17       stop=1;//le jeu de taquin s'arrête et on relance un écran de menu
18       break;
19     case SDL_KEYDOWN://S'il appuie sur une touche
20       if ((success==0) | (verifier(tab,4)==0)){//S'il n'a pas encore réussi le jeu. On utilise
        l'opérateur "|" pour ne pas avoir à faire trop de calculs pour rien. Il est possible que
       success soit à 1 mais que le taquin ne soit pas réussi, dans le cas où on a rechargé une
       sauvegarde de grille alors qu'on vient de réussir le jeu
21     success=0;//La grille n'est donc pas réussi
22     switch(event.key.keysym.sym){
23     case SDLK_LEFT: //Si la touche est la flèche de gauche
24       if (permuter(tab,'g',4)==1){//On permute la case vide avec la case juste à sa gauche si
       possible, et si possible on rentre dans le if
25         Mix_PlayChannel(-1,sonPermu,0);//On joue le son de permutation
26         *nbCoupsPoint = *nbCoupsPoint + 1;//Le nombre de coups augmente de 1
27         synchro(screen,rect,tab,*nbCoupsPoint);//On synchronise les positions des images des
       cases de la grille avec les cases de la grille
28         SDL_Flip(screen); //On actualise le visuel
29       }
30       break;
31     case SDLK_RIGHT: //Si la flèche est la flèche de droite
32       if (permuter(tab,'d',4)==1){//On permute la case vide avec la case juste à sa droite si
       possible, et si possible on rentre dans le if
33         Mix_PlayChannel(-1,sonPermu,0);//On joue le son de permutation
34         *nbCoupsPoint = *nbCoupsPoint + 1;//Le nombre de coups augmente de 1
35         synchro(screen,rect,tab,*nbCoupsPoint);//On synchronise les positions des images des
        cases de la grille avec les cases de la grille
36         SDL_Flip(screen); //On actualise le visuel
37       }
38       break;
```

```
39        case SDLK_UP://Si la touche est la flèche du haut
40        if (permuter(tab,'h',4)==1){//On permute la case vide avec la case juste en haut si
      possible, et si possible on rentre dans le if
41            Mix_PlayChannel(-1,sonPermu,0);//On joue le son de permutation
42            *nbCoupsPoint = *nbCoupsPoint + 1;//On incrémente de 1 le nombre de coups
43            synchro(screen,rect,tab,*nbCoupsPoint);//On synchronise les positions des images des
      cases de la grille avec les cases de la grille
44            SDL_Flip(screen);//On actualise le visuel
45        }
46        break;
47        case SDLK_DOWN://Si la touche est la flèche du bas
48        if (permuter(tab,'b',4)==1){//On permute la case vide avec la case juste en bas si
      possible, et si possible on rentre dans le if
49            Mix_PlayChannel(-1,sonPermu,0);//On joue le son de permutation
50            *nbCoupsPoint = *nbCoupsPoint + 1;//On incrémente de 1 le nombre de coups
51            synchro(screen,rect,tab,*nbCoupsPoint);//On synchronise les positions des images des
      cases de la grille avec les cases de la grille
52            SDL_Flip(screen);//On actualise le visuel
53        }
54        break;
55      default:
56      break;
57      }
58        if (verifier(tab,4)==1){//Si le taquin est réussi après la permutation qui vient d'être
      effectuée
59            achievement(screen);//On affiche l'écran de succès
60            Mix_PlayChannel(-1,sonSuccess,0);//On joue le son de succès
61            success=1;//On indique à la variable que le jeu de taquin a été réussi
62      }
63        }
64        break;
65        case SDL_MOUSEBUTTONDOWN://Si il clique sur un bouton de la souris
66      if (event.button.button==SDL_BUTTON_LEFT){//Si il fait un clic gauche
67        stop=clickOnJeu(screen,rect,button,tab,nbCoupsPoint,sonClic,event.button.x,event.button.
      y);//On rentre dans stop la valeur retournée par la fonction qui gère les clic de la
      souris pour savoir si le jeu de taquin se termine
68        }
69        break;
70        case SDL_MOUSEBUTTONUP://Si on relâche un bouton de souris
71      if (event.button.button==SDL_BUTTON_LEFT){//Si on relâche le clic gauche
72        x=event.motion.x;//Coord axe horizontal
73        y=event.motion.y;//Coord axe vertical
74        //Position de la souris au relâchement du clic
75        if (x>=540 && x<=860 && y>=156 && y<=220 && success==0){//Si on clique sur sauvegarder
76      colorButton(screen,button,7,540,156);//On fait passer le bouton du bleu foncé au bleu
77        }
78
79        else if (x>=540 && x<=860 && y>=260 && y<=324){//Si on clique sur Recharger
80      colorButton(screen,button,5,540,260);//On fait passer le bouton du bleu foncé au bleu
81        }
82
83        else if (x>=540 && x<=860 && y>=364 && y<=428){//Si on clique sur Quitter
84      colorButton(screen,button,6,540,364);//On fait passer le bouton du bleu foncé au bleu
85        }
86        else {//Si on clic sur un endroit qui n'est pas un bouton
87      setButtonJeu(screen,button,tab);//On réaffiche tous les boutons au cas où certains
      seraient affichés en bleu foncé
88        }
89        }
90        break;
91        case SDL_MOUSEMOTION://Si on bouge la souris
92      x=event.motion.x;//Coord axe horizontal
93      y=event.motion.y;//Coord axe vertical
94        //Position de la souris
95      if (x>=540 && x<=860 && y>=156 && y<=220 && success==0){//Si il passe sur le bouton
      Sauvegarder et que success est à 0, donc que le jeu n'est pas encore réussi
96        colorButton(screen,button,7,540,156);//On passe du gris au bleu sur ce bouton
97        }
98
99      else if (x>=540 && x<=860 && y>=260 && y<=324){//Si il passe sur le bouton Recharger
100       colorButton(screen,button,5,540,260);//On passe du gris au bleu sur ce bouton
101       }
102
103     else if (x>=540 && x<=860 && y>=364 && y<=428){//Si il passe sur le bouton Quitter
104     colorButton(screen,button,6,540,364);//On passe du gris au bleu sur ce bouton
105       }
106     else {//S'il ne passe sur aucun bouton particulier
107       setButtonJeu(screen,button,tab);//On réaffiche tous les boutons au cas où certains
      seraient affichés en bleu.
```

```
108         }
109         break;
110         default:
111         break;
112       }
113     }
114       Mix_FreeChunk(sonPermu); //On libère la mémoire allouée pour le son de permutation
115       Mix_FreeChunk(sonClic); //On libère la mémoire allouée pour le son du clic
116       Mix_FreeChunk(sonSuccess); //On libère la mémoire allouée pour le son de succès du jeu de
          taquin
117   }
```

../codes/graphique.c

Gestion des clics sur le jeu de taquin

La fonction "clickOnJeu" gère les clics en partie comme le fait la fonction "clickOnMenu" pour la gestion des clics sur le menu. La différence majeure vient du fait que quand on clique sur un bouton, ce dernier devient plus foncé. Si l'utilisateur clique sur le bouton "Sauvegarder", on utilise les fonctions de la partie algorithmique du projet pour sauvegarder la grille ainsi que le nombre de coups effectués. Si l'utilisateur clique sur le bouton "Recharger, celà synchronise la grille affichée avec celle rechargée. Si cette dernière est déjà terminé alors on affiche l'écran de réussite. Si l'utilisateur clique sur le bouton "Quitter" il quitte le jeu de taquin et un menu est de nouveau crée.

```
1   int clickOnJeu(SDL_Surface*screen,SDL_Surface**rect,SDL_Surface**button,int **tab,int*nbCoups,
        Mix_Chunk*sonClic,int x, int y){
2
3     if (x>=540 && x<=860 && y>=156 && y<=220 && verifier(tab,4)==0){ //Si il clique sur le
        bouton Sauvegarder
4       Mix_PlayChannel(-1,sonClic,0); //On lance le son de clic
5       sauvegarder(tab,*nbCoups,4); //On sauvegarde la grille de taquin ainsi que le nombre de
        coups passé en paramètre
6       colorButton(screen,button,11,540,156); //On affiche le bouton Sauvegarder en bleu foncé (
        pressé)
7       return 0; //On reste dans la boucle d'input du joueur
8     }
9
10    else if (x>=540 && x<=860 && y>=260 && y<=324){//Si il clique sur le bouton Recharger
11      Mix_PlayChannel(-1,sonClic,0);//On joue le son de clic
12      *nbCoups=recharger(tab,4); //On recharge la grille et on met dans la case mémoire du
        nombre de coups la valeur du nombre de coups sauvegardée
13
14      synchro(screen,rect,tab,*nbCoups);//On synchronise les images des cases de la grille avec
        la grille
15      setButtonJeu(screen,button,tab);//On remet tous les boutons en bleu au cas où le taquin
        soit auparavant réussi et que le bouton Sauvegarder soit bleu foncé
16
17      if (verifier(tab,4)==1){//Si la grille rechargée était réussie
18        achievement(screen);//On affiche l'écran de réussite
19      }
20
21      colorButton(screen,button,9,540,260);//On met en bleu foncé le bouton Recharger car il est
         pressé
22      return 0;//On reste dans la boucle d'input du joueur
23    }
24
25    else if (x>=540 && x<=860 && y>=364 && y<=428){//Si il clique sur le bouton quitter
26      Mix_PlayChannel(-1,sonClic,0);//On lance le son de clic
27      return 1;//On sort de la boucle d'input du joueur dans le but de relancer un menu
28    }
29    else {//Si il ne clique sur aucun bouton
30      return 0;//On reste dans la boucle d'input du joueur
31    }
32
33
34  }
```

../codes/graphique.c

Synchronisation de l'interface avec la grille

La fonction de synchronisation, "synchro", lis le tableau qui contient la grille de taquin et affiche les cases de la grille de façon à ce qu'elles correspondent au tableau. Si un écran de réussite du taquin était précédement affiché celui est recouvert par une image de fond avant d'afficher les cases de la grille. Cette fonction permet également d'afficher le nombre de coups effectués par le joueur. On recouvre également le nombre de coups précédemment affiché en affichant un cadre à la place.

```c
void synchro(SDL_Surface*screen,SDL_Surface**rect,int**tab,int nbCoups){
  SDL_Rect pos;//Variable contenant les positions des différents éléments que l'on va positionner
  pos.x=0; //Coord axe horizontal
  pos.y=0;//Coord axe vertical
  TTF_Font * font = NULL;
  font = TTF_OpenFont("font/Adobe_Dia.ttf",41);//Pointeur pointant sur une police de taille 41
  if (font == NULL){ //Si font n'a pas réussi à pointer vers la police
    fprintf(stderr,"\nUnable to load TTF: %s\n",TTF_GetError() );//Message d'erreur
  }

  SDL_Surface * background = SDL_LoadBMP("image/background1.bmp");//Surface pointant vers une image de fond

  SDL_Color colorText;//couleur de la police (blanc)
  colorText.r = 255;
  colorText.g = 255;
  colorText.b = 255;



  char nbCoupsTxt[30];//Chaîne de caractère contenant le texte à écrire pour donner le nombre de coups

  sprintf(nbCoupsTxt,"Nombre de coups: %i",nbCoups);//On écrit à la fin du texte le nombre de coups

  SDL_Surface * nbCoupsTTF = TTF_RenderText_Blended(font,nbCoupsTxt,colorText); //Surface pointenant vers l'image du rendu du texte

  SDL_Surface * blocNbCoups = SDL_LoadBMP("image/nbCoups.bmp");//Surface pointant vers l'image de la case dans laquelle on va écrire le texte

  SDL_BlitSurface(background,NULL,screen,&pos);//On pose l'image de fond en [0;0] pour cacher l'écran de réussite du jeu de taquin au cas où il soit affiché

   pos.x=540;
  pos.y=62;
  //Coord de l'image de la case dans laquelle on va écrire le texte

  SDL_BlitSurface(blocNbCoups,NULL,screen,&pos);//On met l'image de la case aux dernières coordonnées

  pos.x=575;
  pos.y=76;
  //Coord de l'emplacement du texte
  SDL_BlitSurface(nbCoupsTTF,NULL,screen,&pos);//On place le texte au dernières coordonnées

  for (int i=0;i<4;i++){
    for (int j=0;j<4;j++){//Pour toutes les cases de la grille de taquin
      pos.x=60+95*j;
      pos.y=60+95*i;
      //Coord de la case [i;j]
      SDL_BlitSurface(rect[tab[i][j]-1],NULL,screen,&pos);//On place l'image de la case en fonction de la grille dans tab
    }
  }

  TTF_CloseFont(font); //On libère la mémoire allouée pour pointer sur la police
  SDL_FreeSurface(background); //On libère la mémoire allouée pour l'image de fond
  SDL_FreeSurface(nbCoupsTTF); //On libère la mémoire allouée pour l'image du texte

}
```

../codes/graphique.c

2.3.4 Fonctions partagées par le menu et le jeu de taquin

Coloration des boutons

Pour colorer un bouton nous avons décidé de simplement le recouvrir d'une autre image du même bouton mais avec une différente couleur avec la fonction "colorButton".

```c
void colorButton(SDL_Surface*screen,SDL_Surface**button,int num,int x, int y){

  SDL_Rect pos;//Variable contenant la position du bouton que l'on va changer de couleur
  pos.x=x;//Position horizontal du bouton
  pos.y=y;//Position vertical du bouton
```

```
6
7     SDL_BlitSurface(button[num],NULL,screen,&pos);//On place l'image en bleu du bouton
8
9     SDL_Flip(screen);//On actualise le visuel du tout
10
11  }
```

../codes/graphique.c

Libération de la mémoire

La libération des listes d'images se fait avec la fonction "freeSurface" qui libère la mémoire allouée pour chaque élément de la liste ainsi que la mémoire allouée pour chaque image.

```
1  void freeSurface(SDL_Surface**rect,int n){
2    for (int i=0;i<n;i++){ //Pour toutes les surfaces de la liste
3      SDL_FreeSurface(rect[i]);//On libère la mémoire allouée pour ce qui est pointé par le
       pointeur de surface
4    }
5    free(rect);//On libère le la mémoire allouée par le pointeur sur les pointeur de surface.
6  }
```

../codes/graphique.c

Chapitre 3

Conclusion

Le programme final respecte les consignes données : On a un jeu de taquin avec des grilles générées aléatoirement, des fonctionnalités impliquant l'utilisation de fichiers, une surcouche graphique utilisant la librairie SDL et le jeu est étendu à sa version généralisée (en mode console exclusivement).
Par ailleurs l'aspect graphique à été particulièrement travaillé, on a essayé de donner une identité visuelle et auditive adapté à la concentration du joueur.
La répartition des taches a bien marché puisque les deux parties on pu être développées en parallèle. On retient cependant qu'il aurait été astucieux de se mettre d'accord sur les conventions de nommage avant de commencer afin d'avoir une plus grande cohérence dans les codes.