I.U.T. Montpellier – Département informatique – 1^{re} année

Développement initiatique

TP : le taquin

Etes-vous débutant ?

L'idée de ce TP est de travailler en équipes de 4 étudiants environ, en mélangeant dans une même équipe des expérimentés et des débutants, les expérimentés jouant le rôle de "parrain".

Pour mieux (auto) évaluer si vous êtes débutant ou non, comptabilisez le nombre de notions ci-dessous que vous pensez maîtriser correctement :

1. affectation de variables
2. saisie au clavier
3. affichage à l'écran
4. alternative (si... alors... sinon...)
5. boucle pour
6. boucle tant-que
7. fonction/procédure sans paramètres
8. fonction/procédure avec paramètres
9. tableau à une dimension
10. tableau à deux dimensions (matrice)

Toutes ces notions sont utilisées dans ce TP. Si vous n'en connaissez pas ou en connaissez peu, pas de panique ! Elles seront vues en détail une à une en cours, TD et TP d'ici la Toussaint !

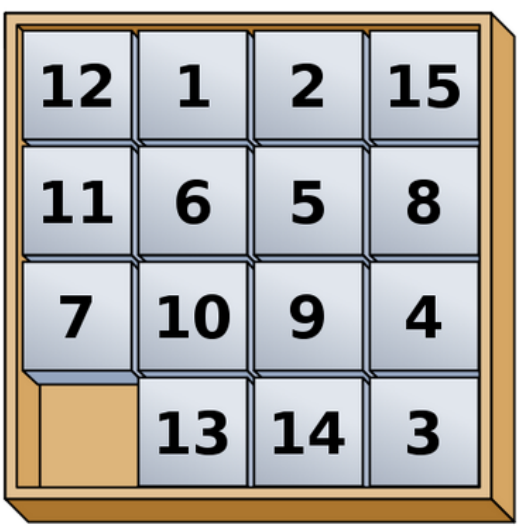
1 Règles du taquin

Un jeu (ou une grille) 4 x 4 est composé(e) de 15 petits carreaux numérotés de 1 à 15 qui glissent dans un cadre prévu pour 16 ; il y a donc un "trou". Le but est de remettre dans l'ordre les 15 carreaux, et le trou en bas à droite, à partir d'une configuration initiale quelconque. Les seuls coups possibles déplacent dans le trou un carreau "voisin" au nord, à l'est, au sud ou à l'ouest, ce qui revient à déplacer le trou vers le sud, l'ouest, le nord ou l'ouest respectivement.

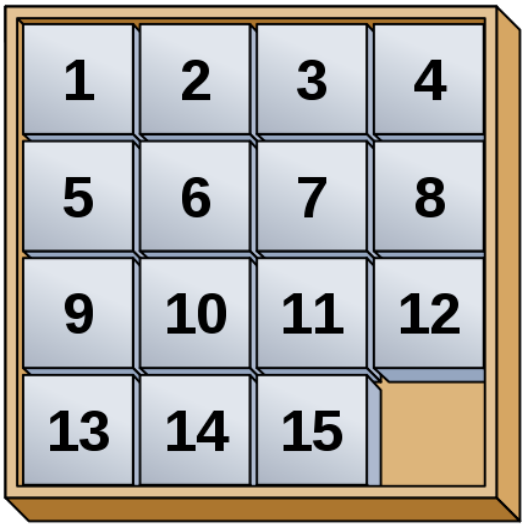
Remarque : dans ce TP nous nous intéressons uniquement à des grilles de taquin carrées (autant de lignes que de colonnes), d'au moins 2 lignes et 2 colonnes.

Un exemple de grille 4 x 4 apparaît à la figure 1.

Dans notre programme, le jeu est représenté par une matrice, plus précisément par un tableau de 4 tableaux de 4 entiers : les valeurs sont les entiers de 1 à 15 et le trou est représenté par le nombre 0.



(a) Exemple de grille initiale



(b) Grille gagnante

FIGURE 1 – Taquin

2 Préliminaires

Sous Linux, ouvrir un terminal. Créer un répertoire `devinit` (`cd ~/Documents ; mkdir devinit`) et un sous-répertoire `TAQUIN` (`cd devinit ; mkdir TAQUIN`). Placez-vous dans ce sous-répertoire (`cd TAQUIN`).

Dans le cours Moodle `S1-Initiation au développement (R.1.01)`, récupérer le fichier `Taquin.java` (dans la section `TD,TP/TP_TAQUIN`) et le fichier `UTILE/Ut.java`, que vous copierez dans votre répertoire `TAQUIN`.

Vous pouvez aussi récupérer le fichier `UTILE/syntaxe_3_langages.doc` qui vous aidera à découvrir la syntaxe du langage Java si vous ne la connaissez pas encore.

3 Afficher une grille de taquin

Exécuter le programme à l'aide des commandes suivantes :

1. `javac Taquin.java` (création du fichier Taquin.class)
2. `java Taquin` (exécution du fichier Taquin.class)

Ouvrir le fichier `Taquin.java`. Observer la définition des variables `grille1`, `grille2` (grilles mélangées) et `grille3` (grille gagnante).

Ouvrir le fichier `Ut.java` et y lire attentivement la fonction `void afficher (int [][] mat)` pour comprendre l'algorithme d'affichage de la grille.

4 Carreaux dans l'ordre ?

Nous définissons un ordre entre les carreaux sur la grille. Considérons deux carreaux c_1 et c_2 aux positions (i_1, j_1) et (i_2, j_2) respectivement. On dit que $c_1 < c_2$ si $i_1 < i_2$. En cas d'égalité sur les lignes, on considère l'ordre entre les numéros de colonnes. Dans cet ordre, le premier carreau se trouve en haut à gauche de la grille et le dernier carreau se trouve en bas à droite.

Compléter la fonction `sontDansLOrdre(int i1, int j1, int i2, int j2)` qui retourne vrai si les carreaux (i_1, j_1) et (i_2, j_2) sont dans cet ordre dans la grille, et faux sinon.

5 Calculer la position du trou

Compléter la fonction `int[] positionTrou(int[][] grille)` qui retoune la position du trou dans la `grille` sous la forme d'un tableau de deux valeurs entières indiquant respectivement le numéro de la ligne et le numéro de la colonne du trou. Par exemple, si le trou est situé à la troisième ligne et à la première colonne de la grille, la fonction retourne [2,0] (le premier indice d'un tableau en Java étant 0).

6 Détecter si une grille de taquin est gagnante

Compléter la fonction `boolean estSolution(int[][] grille)` qui retourne `vrai` si le paramètre `grille` est gagnante (nombres par ordre croissant et trou en bas à droite ; cf. figure) et `faux` sinon.

7 Effectuer un déplacement

7.1 Choix du déplacement

Afin de pouvoir déplacer le trou du taquin, le joueur doit préalablement choisir le mouvement à effectuer. Il est possible de déplacer le trou vers le haut (n, comme nord), vers le bas (s), vers la droite (e) ou vers la gauche (o). Dans cette première version, nous considérons que l'utilisateur va saisir l'un des caractères 'n', 's', 'e' et 'o', et que ce caractère représente un mouvement valide (pour la position courante du trou). Par exemple, si le trou se trouve sur la première ligne, le caractère n ne représente pas un mouvement valide car il ferait "sortir le trou de la grille".

Ecrire une fonction `char saisirDeplacement()` qui affiche le message suivant "Veuillez saisir un mouvement n/s/e/o :" et qui retourne la lettre (caractère) saisie par l'utilisateur. Vous pourrez utiliser les fonctions `Ut.afficher` et `Ut.saisirCaractere`.

7.2 Réaliser le déplacement choisi

Ecrire une fonction `void deplacerTrou(int[][] grille, int[] posTrou)` qui appelle la fonction précédente pour demander à l'utilisateur de saisir un mouvement, et qui réalise ce mouvement en modifiant les paramètres `grille` et `posTrou`, le paramètre `posTrou` contenant la position du trou dans la grille.

La fonction permet de modifier le contenu des tableaux `grille` et `posTrou`. En java en effet, si une fonction f_2 (ici `deplacerTrou`) modifie un de ses paramètres et que ce paramètre est de type tableau, le tableau est modifié aussi dans la fonction f_1 (ici la fonction principale (`main`)) qui appelle f_2. Attention, ce n'est pas le cas si ce paramètre est d'un type de base (entier, réel, caractère ou booléen).

On suppose que le déplacement choisi est valide.

8 Jouer

Ecrire une fonction `void jouer()` définissant une grille initiale et faisant appel aux fonctions précédentes. La fonction affiche itérativement la grille courante et demande à l'utilisateur de saisir un déplacement, tant que la grille n'est pas la solution.

Appeler `jouer` dans la fonction principale (`main`) et tester.

9 Nombre de coups maximal

Modifier la fonction `jouer` pour arrêter le jeu au bout de k déplacements (ou bien sûr en moins de k coups si la grille solution est obtenue).

Le nombre k est demandé au joueur dans la fonction principale ou bien mis sur la ligne de commande (`args[0]`).

10 Extensions possibles

Vous pouvez améliorer votre jeu en implantant une ou plusieurs des extensions suivantes.

10.1 Une fonction `saisirDeplacement` robuste

Modifier cette fonction pour que :

- elle retourne nécessairement l'un des caractères 'n', 'e', 's', 'o', quitte à redemander sa saisie à l'utilisateur en cas d'erreur.
- redemande également sa saisie à l'utilisateur si son choix de déplacement est invalide, c'est-à-dire déplacerait le trou en dehors de la grille (ex : 'n' si le trou est sur la première ligne).

10.2 Grille de taquin générée aléatoirement (difficile)

Compléter la fonction `int[][] genererGrille(int n)` qui retourne une grille de taquin initiale de côté n construite aléatoirement, c'est-à-dire contenant tous les nombres de 0 à $n^2 - 1$ dans n'importe quel ordre. Cette fonction est appelée pour générer la grille initiale.

Après avoir lu le code correspondant, on pourra utiliser `Ut.randomMinMax` pour tirer aléatoirement un entier entre deux valeurs extrêmes.

10.3 Jouer plusieurs parties

Demander le nombre de parties (`nbParties`) à l'utilisateur (ou bien l'indiquer sur la ligne de commandes) et ajouter une fonction `parties` qui effectue `nbParties` parties.

10.4 Note du joueur sur plusieurs parties

Calculer une note du joueur sur plusieurs parties, cette note étant le nombre de parties réussies (telles que nombre de déplacements effectués $\leq k$) sur les `nbParties` parties effectuées (par exemple 20).

10.5 Grille aléatoire et résoluble (très difficile)

Vous avez peut-être lu que la moitié des grilles de taquin d'une taille donnée possibles sont résolubles, c'est-à-dire permettent d'aboutir à la grille solution après plusieurs déplacements, alors que l'autre moitié des configurations sont insolubles.

Ecrire une fonction `boolean estGrilleResoluble(int[][] grille)` qui retourne vrai ssi la grille est résoluble.

Ecrire une fonction `int[][] genererGrilleResoluble()` qui retourne une grille résoluble de taquin construite aléatoirement.

Indications :

1. Vous pouvez trouver la condition de résolubilité sur le site suivant : `https://fr.wikipedia.org/wiki/Taquin`
2. Stratégie possible : construire des grilles aléatoirement jusqu'à obtenir une grille résoluble.