Exercices sur les tableaux à deux dimensions

E. Thirion - Dernière mise à jour : 04/07/2015

Les exercices suivants sont en majorité des projets à complèter. L'interface graphique de ces projets est déjà réalisée, ce qui vous permettra un gain de temps important. Ces projets sont disponibles par téléchargement. Pour savoir comment y accéder cliquez ici.

D'autre part, ce document fait partie d'un ensemble de cours du même auteur (programmation procédurale et objet, programmation web, bases de données) auxquels vous pouvez accéder en cliquant ici.

I - Le Noir et Blanc

Ouvrir : Exo-Tableaux/NoirEtBlanc/ProjetNoirEtBlanc.lpi

Objectif

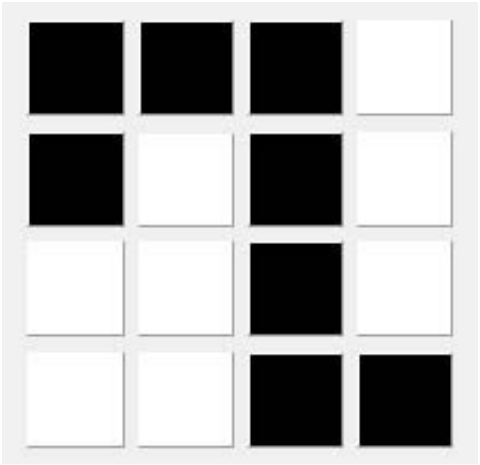
Il s'agit de réaliser différentes opérations sur un tableau à deux dimensions de booléens dont voici la déclaration:

```
TabBool : array [1..NL,1..NC] of boolean;
```

NL et **NC** sont des constantes dont la valeur a été fixé à 4. Dans tout le programme, vous prendrez soin d'utiliser ces constantes et non pas la valeur 4, afin que les dimensions du tableau puisse être aisément modifiées.

Le tableau est représenté graphiquement par 16 carrés noirs ou blancs selon que l'élément du tableau correspondant est vrai ou faux.

La valeur vrai correspond au blanc et la valeur faux au noir. Exemple:



	1	2	3	4
1	false	false	false	true
2	false	true	false	true
3	true	true	false	true
4	true	true	false	false

Représentation graphique	**TabBool**

1) Contrôle d'indices et accès à un élément

1-A) La fonction DansTableau

```
function DansTableau (i, j : integer) : boolean;
```

Cette fonction doit retourner la valeur vrai si et seulement si les indices **i** (ligne) et **j** (colonne) sont des indices valides du tableau **TabBool**.

1-B) La fonction TB

```
function TB (i, j : integer) : boolean;
```

- Si les indices **i** et **j** sont en dehors du tableau, elle affiche un message d'erreur précisant la valeur des indices. On utilisera ici la fonction **DansTableau**.
- Sinon elle retourne **TabBool** [i , j].

Dans la suite de l'exercice, afin éviter les erreurs d'exécution dues à des indices hors tableau, utilisez cette fonction plutot que l'accès direct par **TabBool** [i , j].

1-C) La procédure InverserElement

```
procedure InverserElement (i, j: integer);
```

Elle inverse la valeur de **TabBool** [i , j]. Utilisez la fonction **TB** pour controler les indices et afficher un message d'erreur s'ils sortent du tableau !

Test: Si cette procédure fonctionne correctement un clic sur un carré doit inverser sa couleur (les boutons radio **Inverser** et **Elément** doivent être cochés).

2) Inversion de différentes parties du tableau

Dans cette partie de l'exercice, on vous demande de complèter différentes procédures permettant d'inverser (changer vrai en faux et inversement) différentes parties du tableau.

La partie du tableau à inverser est définie par les boutons radios: élément, ligne, colonne, ...

L'inversion est déclenchée par un clic sur un des carrés.

Les questions sont ordonnées par ordre de difficultée.

2-A) Inversion d'une ligne

```
procedure InverserLigne (i, j: integer);
```

Inverse tous les éléments de la ligne contenant l'élément **TabBool** [i , j].

2-B) Inversion d'une colonne

```
procedure InverserColonne (i, j: integer);
```

Inverse tous les éléments de la colonne contenant l'élément **TabBool** [i , j].

2-C) Inversion du tableau entier

```
procedure InverserTout () ;
```

Pensez à réutiliser le code existant !

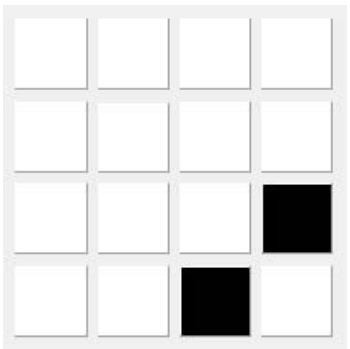
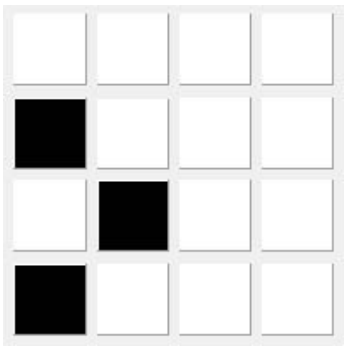
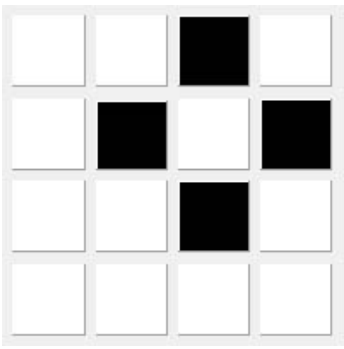
2-D) Inversion des quatres voisins

```
procedure Inverser4Voisins (i, j: integer);
```

Inverse les quatres voisins (explication ci dessous) de **TabBool** [i , j].

Les quatre voisins d'un élément de tableau à deux dimensions sont les éléments se situant à gauche, à droite, en dessous et au-dessus de celui-ci.

Selon sa position dans le tableau, un élément peut avoir 2, 3 ou 4 voisins de ce type:

		
Dans le coin	**Sur le bord**	**A l'intérieur**

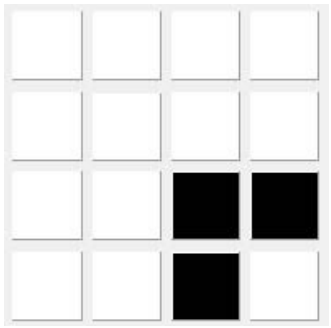
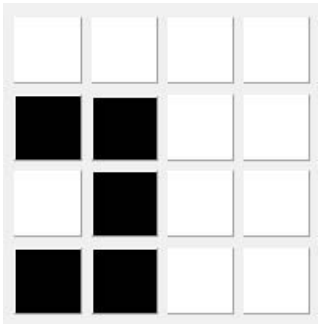
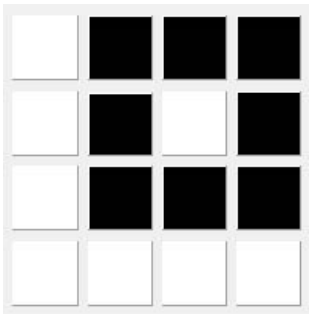
2-E) Inversion des huit voisins

```
procedure Inverser8Voisins (i,j: integer);
```

Inverse les huit voisins (explication ci dessous) de **TabBool** [i , j].

Les huit voisins d'un élément de tableau à deux dimensions sont les éléments qui "touchent" celui-ci par un coté ou un coin.

Selon sa position dans le tableau, un élément peut avoir 3, 5 ou 8 voisins de ce type:

		
Dans le coin	**Sur le bord**	**A l'intérieur**

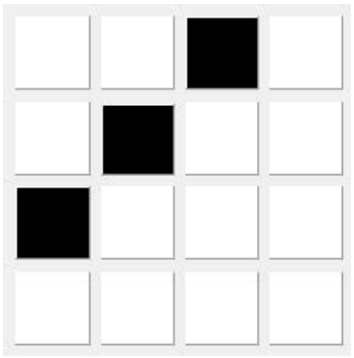
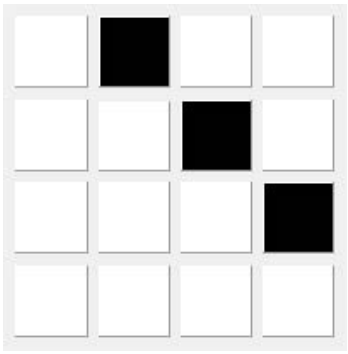
2-F) Inversion des diagonales

procedure InverserDiagonale1 (i, j: **integer**);

Inverse les éléments se trouvant dans la même diagonale (de type 1) que **TabBool** [i , j].

procedure InverserDiagonale2 (i, j: **integer**);

Inverse les éléments se trouvant dans la même diagonale (de type 2) que **TabBool** [i , j].

	
Une diagonale de type 1	**Une diagonale de type 2**

3) Dénombrement du nombre de carrés blancs et noirs

Dans cette partie de l'exercice, on vous demande de complèter différentes fonctions calculant le nombre d'éléments vrais ou faux dans une certaine partie du tableau.

La partie du tableau dans laquelle on compte est définie par les boutons radios: élément, ligne, colonne, ...

Le dénombrement est déclenché par un clic sur un des carrés à condition que le bouton radio **Compter** soit coché. Le nombre de carré blancs et noirs est affiché dans deux zones de texte.

Les questions sont données par ordre de difficultée.

3-A) Dénombrement dans une ligne

```
function CompterDansUneLigne (i: integer; Blanc : boolean) : integer;
```

Si **Blanc** est vrai, cette fonction retourne le nombre d'éléments de valeur **True** dans la ligne numéro **i**. Sinon elle retourne le nombre d'éléments de valeur **False**.

3-B) Dénombrement dans une colonne

```
function CompterDansUneColonne (j: integer; Blanc : boolean) : integer;
```

Si **Blanc** est vrai, cette fonction retourne le nombre d'éléments de valeur **True** dans la colonne numéro **j**. Sinon elle retourne le nombre d'éléments de valeur **False**.

3-C) Dénombrement dans tout le tableau

```
function CompterDansTout (Blanc: boolean): integer;
```

Si **Blanc** est vrai, cette fonction retourne le nombre d'éléments de valeur **True** dans le tableau. Sinon elle retourne le nombre d'éléments de valeur **False.**

Pensez à réutiliser le code existant !

3-D) Dénombrement parmis les quatre voisins

```
function CompterParmisLes4Voisins (i,j: integer; Blanc: boolean) : integer;
```

Si **Blanc** est vrai, cette fonction retourne le nombre de voisins de valeur **True** parmis les quatre (au plus) voisins de **TabBool** [i , j]. Sinon elle retourne le nombre d'éléments de valeur **False.**

3-E) Dénombrement parmis les huit voisins

```
function CompterParmisLes8Voisins (i,j: integer; Blanc: boolean) : integer;
```

Si **Blanc** est vrai, cette fonction retourne le nombre de voisins de valeur **True** parmis les huit (au plus) voisins de **TabBool** [i , j]. Sinon elle retourne le nombre d'éléments de valeur **False.**

3-E) Dénombrement dans les diagonales

```
function CompterDansDiag1 (i, j: integer; Blanc: boolean) : integer;
```

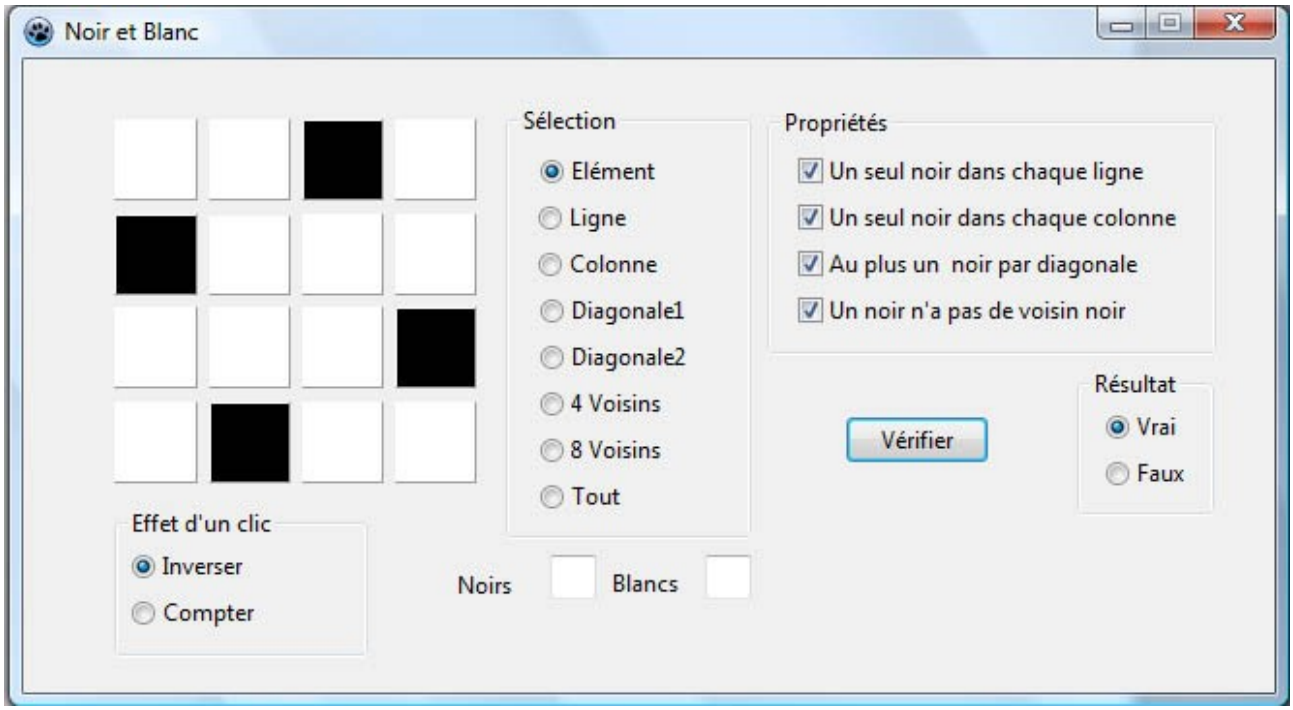
Si **Blanc** est vrai, cette fonction retourne le nombre d'éléments de valeur **True** dans la diagonale de type 1 de **TabBool** [i , j]. Sinon elle retourne le nombre d'éléments de valeur **False.**

function CompterDansDiag2 (i, j: **integer**; Blanc: **boolean**) : **integer**;

Si **Blanc** est vrai, cette fonction retourne le nombre d'éléments de valeur **True** dans la diagonale de type 2 de **TabBool** [i , j]. Sinon elle retourne le nombre d'éléments de valeur **False.**

4) Vérification de propriétés

Le programme permet de vérifier si le tableau satisfait certaines propriétés en cliquant sur le bouton **Vérifier** après avoir sélectionné les propriétés. Dans l'exemple suivant toutes les propriétés sont vérifiées:



Indication: Ces propriétés peuvent être vérifiées en utilisant les fonctions de dénombrement de la question précédente.

4-A) Un seul noir dans chaque ligne

function UnSeulNoirParLigne () : **boolean**;

Cette fonction retourne la valeur **True** si et seulement si chaque ligne du tableau contient un et un seul élément faux.

4-B) Un seul noir dans chaque colonne

function UnSeulNoirParColonne () : **boolean**;

Cette fonction retourne la valeur **True** si et seulement si chaque colonne du tableau contient un et un seul élément faux.

4-C) Un noir n'a pas de voisins noirs

```
function UnNoirNaPasDeVoisinNoir () : boolean;
```

Cette fonction retourne la valeur **True** si et seulement si chaque carré noir du tableau n'a aucun voisin (au sens 8 voisins) noir.

4-C) Au plus un noir par diagonale

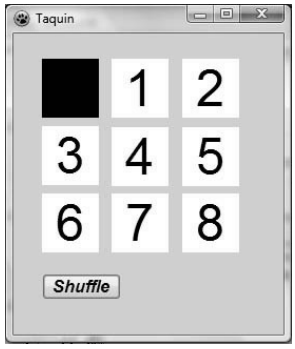
```
function AuPlusUnNoirParDiagonale ( ) : boolean;
```

Cette fonction retourne la valeur **True** si et seulement si chaque diagonale (de type 1 ou 2) du tableau contient au plus un carré noir.

II - Le taquin

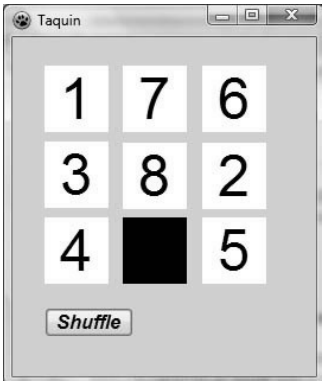
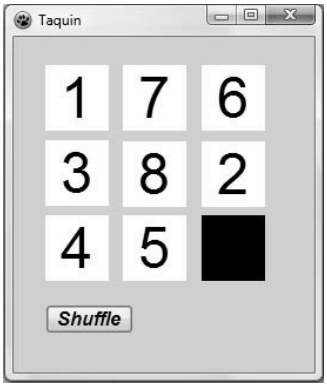
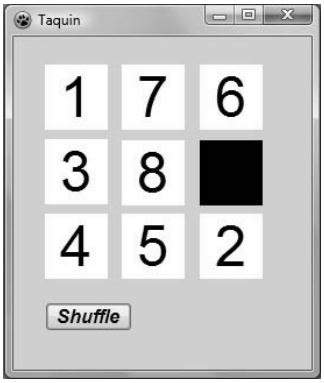
Ouvrir : Exo-Tableaux/Taquin/ProjetTaquin.lpi

Le formulaire



Principe du jeu

Lorsqu'on clique sur un chiffre se trouvant à coté du trou (le carré noir), le chiffre est échangé avec le trou. Le but du jeu est d'arrivé à ranger les chiffres dans l'ordre (c.a.d comme ci-dessus) en effectuant plusieurs fois cette opération. Voilà par exemple deux coups joué depuis le début du jeu:

		
Situation initiale	*Après avoir cliqué sur le 5*	*Après avoir cliqué sur le 2*

Le bouton **Shuffle** permet de mettre les chiffres dans le désordre. Cette opération est assez complexe à mettre en oeuvre. Elle sera donc laissé en projet.

1) Déclaration du tableau

Le jeu est représenté en mémoire par le tableau **TabJeu**. Il s'agit d'un tableau d'entiers à deux dimensions de trois lignes et trois colonnes. Les lignes et les colonnes sont indicées de 0 à 2.

Déclarez ce tableau en utilisant de préférence les constantes **NC** (nombre de colonnes) et **NL** (nombre de lignes).

2) Initialisation du tableau

Pour que le programme affiche correctement le jeu au début, il vous faudra complèter la procédure **MettreLesChiffresDansLOrdre**.

Cette procédure affecter les nombres 0, 1, 2, 3, ..., 8 au éléments du tableau afin qu'ils soient rangés dans l'ordre. Le nombre 0 représente la case vide.

3) Gestion du clic sur une case du jeu

Lorsque l'utilisateur clique sur une case, la procédure **GererClicSurCase** est appelé avec les indices de l'élément correspondant du tableau **TabJeu**.

```
GererClicSurCase (i,j: integer);
// Echange TabJeu[i,j] avec l'élément de TabJeu contenant le trou
```

Cette procédure est à complèter.

C'est dans cette procédure qu'il faudra intervertir le trou avec le chiffre, à condition bien sur que l'utilisateur clique sur un chiffre voisin du trou.

Pour échanger les deux cases du tableau **TabJeu** vous utiliserez la procédure **EchangerLesCases:**

```
EchangerLesCases(i1,j1,i2,j2:integer);
//Echange TabJeu[i1,j1] avec TabJeu[i2,j2]
```

Pour afficher le jeu après avoir modifié le tableau, appelez la procédure **AfficherLeJeu**.

4) Mélange des chiffres

La procédure **MelangerLesChiffres** est appelée, lorsque l'utilisateur clique sur le bouton **Shuffle**.

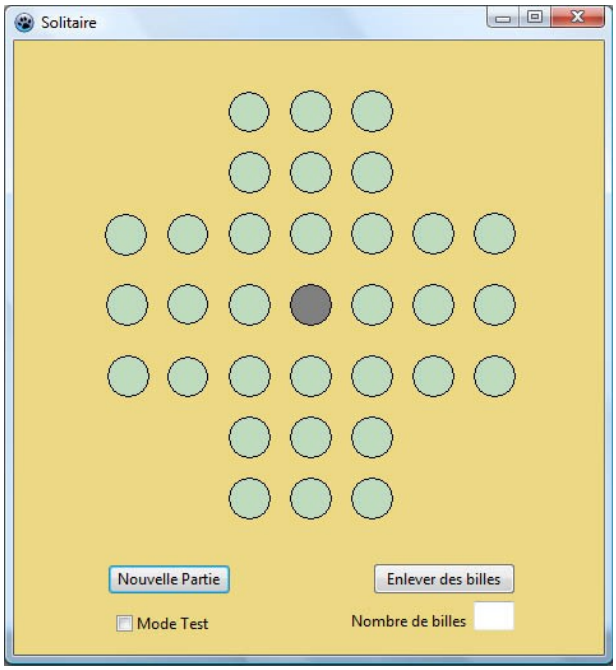
Cette procédure doit changer aléatoirement l'ordre des éléments du tableau **TabJeu** de manière à ce qu'il soit possible de les remettre dans l'ordre.

On pourra utiliser la fonction **Random** (n) qui retourne un nombre entier au hasard compris entre 0 inclu et **n** exclu. Pour ne pas générer systématiquement la même suite de nombre aléatoire appelez la procédure **Randomize** (elle n'a pas de paramètres).

III - Le Solitaire

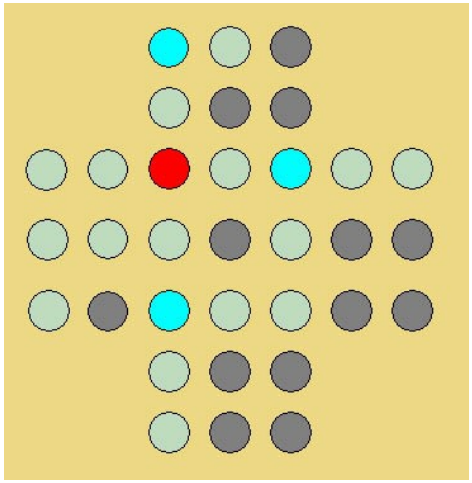
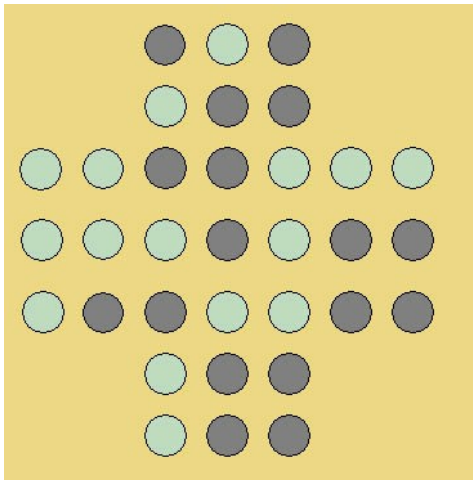
Ouvrir : Exo-Tableaux/Solitaire/ProjetSolitaire.lpi

Le solitaire est un jeu compsé de 32 billes déposées dans des creux et disposées en croix. Le creux central est initialement vide:



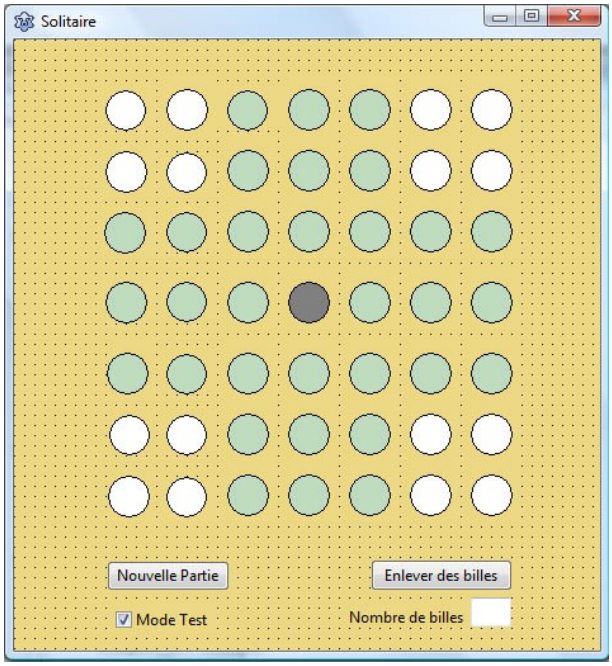
Le but du jeu est d'enlever toutes les billes du jeu en respectant la règle suivante: pour enlever une bille, il faut la faire sauter par dessus une bille voisine et cela n'est possible que si le creux se situant derrière elle est inoccupé. Les sauts en diagonale sont interdits.

Pour jouer, l'utilisateur clique sur une bille. S'il peut la jouer elle apparait en rouge et les creux dans lequels il peut la déplacer apparaissent en bleu. Il clique ensuite sur un des creux en bleu (le plus souvent, il n'y en a qu'un seul) pour déplacer la bille à cet endroit. Dans l'exemple suivant, il y a trois creux possibles:

	
La bille (en rouge) peut être déplacée dans trois creux (en bleu)	Etat du jeu après avoir avoir sélectionné le creux se situant à droite de la bille.

Représentation graphique du jeu

Le jeu est représenté graphiquement par 49 composants de type **TShape** disposés en carré:



Les **TShapes** blancs sont invisibles à l'exécution du programme.

Ils ont été nommés en fonction de leur position dans le carré et stockés par la procédure **InitialiserTabShape** dans le tableau **TabShape** à 7 lignes et 7 colonnes:

Shape1_1	Shape1_2	Shape1_3	Shape1_4	Shape1_5	Shape1_6	Shape1_7
Shape2_1	Shape2_2	Shape2_3	Shape2_4	Shape2_5	Shape2_6	Shape2_7
Shape3_1	Shape3_2	Shape3_3	Shape3_4	Shape3_5	Shape3_6	Shape3_7
Shape4_1	Shape4_2	Shape4_3	Shape4_4	Shape4_5	Shape4_6	Shape4_7
Shape5_1	Shape5_2	Shape5_3	Shape5_4	Shape5_5	Shape5_6	Shape5_7
Shape6_1	Shape6_2	Shape6_3	Shape6_4	Shape6_5	Shape6_6	Shape6_7
Shape7_1	Shape7_2	Shape7_3	Shape7_4	Shape7_5	Shape7_6	Shape7_7

Représentation interne du jeu

Le jeu est représenté de manière interne par un tableau à deux dimensions nommé **TabJeu** possédant 7 lignes et 7 colonnes indicées de 1 à 7.

Un élément **TabJeu** [i , j] correspond au composant graphique **Shape**i_j. Il a cinq valeurs possibles:

- 1 : creux occupé par une bille (représenté graphiquement en gris clair)
- 0 : creux vide (représenté en gris foncé)
- -1: absence de creux (invisible)

- 2 : creux où il est possible de déposer une bille (représentée en bleu)
- 3 : bille sur laquelle l'utilisateur à cliqué et qu'il peut déplacer (représenté en rouge).

Au début ,le tableau contient donc les valeurs suivantes:

	1	*2*	*3*	*4*	*5*	*6*	*7*
1	-1	-1	1	1	1	-1	-1
2	-1	-1	1	1	1	-1	-1
3	1	1	1	1	1	1	1
4	1	1	1	0	1	1	1
5	1	1	1	1	1	1	1
6	-1	-1	1	1	1	-1	-1
7	-1	-1	1	1	1	-1	-1

La représentation graphique du jeu à partir de cette représentation interne est déjà réalisée. Votre travail consistera uniquement à agir sur la représentation interne. Pour mettre à jour la représentation graphique après avoir modifié la représentation interne, il vous suffira d'appeler la procédure **AfficherJeu**.

Question 1: Initialisation du jeu

Complètez la procédure **InitialiserTabJeu**, de manière à que cette procédure initialise correctement le tableau **TabJeu**. Cette procédure est appelée dans **FormCreate**. Il vous suffira donc de lancer le programme pour voir si elle fonctionne correctement.

Question 2: Indication des possibilités

Dans cette question, il s'agit simplement de faire fonctionner l'indication des possibilités de jeu. Pour cela, le programme doit être en mode test. Vous pouvez activer ou désactiver ce mode en cochant la case à cocher du formulaire. Le booléen **ModeTest** vaut **true** si et seulement si elle est cochée.

Votre travail consistera à complèter deux procédures:

- **IndiquerLesPossibilites** (i, j)
  Indique les possibilités de jeu lorsque l'utilisateur clique sur le composant **Shape**i_j.
  L'effet de cette procédure doit être le suivant:
  - Le nombre de possibilités de jeu doit être stocker dans la variable globale **NP** prévue à cet effet.
  - Si l'emplacement **i**, **j** est vide et que le programme est en mode test, affichage du message "Ce creux ne contient aucune bille !!".
  - Si l'emplacement **i**, **j** contient une bille, mais qu'elle ne peut pas être jouée et que le programme est en mode test, affichage du message"Impossible de jouer cette bille !!".
  - Si l'emplacement **i**, **j** contient une bille pouvant être jouée:
    - Cette bille doit s'afficher en rouge.
    - Les creux où elle peut être déposée doivent s'afficher en bleu.
    - En mode test uniquement:
      - Le nombre de coups possibles est affiché dans une boite de dialogue.
      - Dès que l'utilisateur clique sur Ok, la bille réapparait en gris clair et les creux vides où pouvait être déposé la bille, réapparaissent en gris foncé. Vous utiliserez pour cela la procédure **EffacerLesPossibilites** (i, j).

- **EffacerLesPossibilites** (i, j)
  L'appel de cette procédure suppose que l'utilisateur a cliqué une bille jouable située à la ligne **i** et à la colonne **j**. Elle apparait donc pour l'instant en rouge et les creux où elle peut être déposée apparaissent en bleu. L'effet de cette procédure doit être de faire réapparaitre les creux en gris foncé et la bille en gris clair.

Pour tester le bon fonctionnement de ces deux procédures vous pouvez utiliser le bouton permettant de retirer des billes du jeu.

Question 3: Déplacement d'une bille

Décochez le mode test et completez la procédure suivante:

JouerLeCoup (i1, j1, i2, j2): déplace la bille située en ligne **i1**, colonne **j1** dans le creux situé en ligne **i2**, colonne **j2**. La bille se trouvant entre ces deux emplacements est retirée du jeu.

Question 4: Impossibilité de gagner

Complèter le programme de manière à ce que l'utilisateur soit averti de l'impossibilité de gagner dès que possible.