Synthèse : Puissance 4 (en VBA)
Corrigé

1 puissance4

Le jeu « Puissance 4 » est proposé par MB Jeux. Sur la boîte, on peut lire les indications suivantes.

> Puissance 4 est un jeu de stratégie verticale passionnant et plein d'astuce. C'est un jeu facile à apprendre et amusant à jouer. Ses règles sont simples. Chaque joueur essaie de faire une rangée de quatre pions dans le cadre – horizontalement, verticalement ou diagonalement – tout en essayant d'empêcher son adversaire de faire de même. Croyez-moi ce n'est pas aussi facile que cela en a l'air ! La position verticale du jeu demande au joueur beaucoup de concentration et de réflexion.
>
> **Montage :**
>
> 1. Suivez les instructions illustrées pour l'assemblage des parties en plastiques (non reproduites ici).
> 2. Placez le jeu entre les joueurs.
> 3. Chaque joueur choisit une couleur de pions.
>
> **But du jeu :**
> Être le premier joueur à placer quatre de ses pions sur une ligne horizontale, verticale ou diagonale continue.
>
> **Règles du jeu :**
>
> 1. Choisissez le premier joueur. Le joueur qui commence la première partie sera le deuxième joueur au cours de la deuxième partie.
> 2. À tour de rôle, chaque joueur fait tomber un de ses pions dans une des fentes au sommet de la grille.
> 3. Le jeu continue jusqu'à ce qu'un des joueurs ait un alignement continue de quatre pions de sa couleur. L'alignement peut être vertical, horizontal ou en diagonale.
> 4. Pour vider la grille, poussez la barre de retenue qui se trouve au vase de celle-ci et les pions tomberont. Vous pouvez maintenant commencer la partie suivante.

Conseil : Lire attentivement les questions suivantes et ébaucher une solution au brouillon avant de commencer la rédaction.

Exercice 1 : Préparation du jeu

Le jeu « puissance 4 » se joue sur une grille verticale de six lignes et sept colonnes avec 21 pions rouges et 21 pions jaunes.

Pour faciliter les calculs ultérieurs, nous allons ajouter une bordure à cette grille. Ses dimensions sont donc de huit lignes et neuf colonnes. Bien entendu, ces cases supplémentaires ne peuvent pas contenir de pions (qu'ils soient jaunes ou rouges). Elles sont donc toujours vides.

Définir les types nécessaires pour représenter la grille de jeu.

```
1  Const TITRE = "PUISSANCE 4"
2  Const NB_LIGNES As Integer = 6          ' nb de lignes de l'aire de jeu
3  Const NB_COLONNES As Integer = 7        ' nb de colonnes de l'aire de jeu
4  Const NB_ALIGNES_GAGNANT As Integer = 4 ' nb de pions alignés pour gagner
5  Const NB_JOUEURS As Integer = 2         ' Nombre de joueurs dans une partie
6
7  ' Définir le type pour une case de la grille
8  Const VIDE = 0, ROUGE = 1, JAUNE = 2
9
10 ' Définition d'un joueur
11 Type typ_joueur
12     couleur As Integer      ' En fait, soit ROUGE, soit JAUNE
13     nom As String           ' Le nom du joueur
14     ordinateur As Boolean   ' Est-ce l'ordinateur ?
15     nb_victoires As Integer ' Nombre de victoires
16 End Type
17
18 ' Définition des composants du jeu
19 Type typ_jeu
20     grille(0 To NB_LIGNES + 1, 0 To NB_COLONNES + 1) As Integer
21       ' la grille de jeu avec une bordure supplémentaire
22     joueurs(1 To NB_JOUEURS) As typ_joueur ' les 2 joueurs
23 End Type
```

Le jeu est représenté sous forme d'un enregistrement `typ_jeu` contenant la grille de jeu (champ grille de type tableau bidimensionnel d'entiers) et les 2 joueurs (champ joueurs de type tableau monodimensionnel d'éléments de type `typ_joueur`).

Exercice 2 : Vider la grille

Pour commencer chaque partie, il faut commencer par vider la grille des jetons qu'elle contient. Écrire un sous-programme qui réalise cette opération.

Solution : Ce sous-programme a un seul paramètre, la grille, en **`out`**. Il s'agit donc d'une initialisation. Nous en faisons donc une procédure.

```
1 Procédure vider(grille: out Grille) Est
2     -- vider la grille.
```

Ce sous-programme consiste à mettre toutes les cases de la grille à VIDE, y compris les cases de la bordure. On le fait donc naturellement avec deux boucles imbriquées, l'une qui balaie les lignes, l'autre les colonnes.

```
Sub vider_grille(grille() As Integer)
'ROLE : vide la grille du jeu. Ceci correspond à actionner la barre
'       située sous la grille du jeu réel.
'SORTIE Jeu
    Dim i As Integer, j As Integer
    For i = 0 To NB_LIGNES + 1
        For j = 0 To NB_COLONNES + 1
            grille(i, j) = VIDE
        Next j
    Next i
End Sub
```

Exercice 3 : Afficher la grille

Écrire un sous-programme qui affiche le contenu de la grille. Par convention, les jetons de couleur rouge seront représentés par un astérisque (*) et les jetons jaunes par la lettre o minuscule.

Pour faciliter, la localisation d'une case de la grille, les numéros de lignes et de colonnes seront affichés comme sur la figure 1.

```
   1 2 3 4 5 6 7
6        *        6
5        o        5
4      o o   *    4
3      * o   o o  3
2    * o *   o *  2
1  * o * * o o *  1
   1 2 3 4 5 6 7
```

FIGURE 1 – Affichage de la grille

Afin de pouvoir effectuer un affichage séquentiel de chaque pion d'une même ligne sans passer à la ligne (ce que fait implicitement la procédure Afficher), on pourra utiliser les 2 procédures suivantes (mises à disposition) :

```
Procédure AfficherCellule(Info : in ToutType) Est
  -- afficher une information unique de tout type dans la cellule
  -- courante et positionne la cellule courante sur la colonne
  -- suivante sans passer à la ligne

Procédure AllerALaLigne Est
  -- Positionner la cellule active sur la 1ère colonne de la
  -- ligne suivante
```

On pourra facultativement remplacer l'affichage d'un symbole pour chaque pion par la coloration de la cellule Excel correspondante avec la couleur du pion. À cet effet, la procédure suivante est mise à disposition :

```
Procédure ColorerCellule(couleur : in Entier) Est
  -- Colorer le fond de la cellule active avec la couleur spécifiée
  -- ENTRÉE Couleur : constante de couleur (vbWhite,vbYellow, ...etc.)
  --                  ou valeur issue de la fonction RGB (Red,Green,Blue)
```

Solution : Ce sous-programme prend un seul paramètre, la grille. C'est un paramètre en entrée. De plus ce sous-programme fait de l'affichage. Nous en faisons donc une procédure.

```
Procédure afficher(grille: in Grille) Est
    -- Afficher la grille.
```

L'implémentation de ce sous-programme n'est pas très facile à structurer, en particulier si on veut essayer de limiter les redondances qui rendent plus difficiles les évolutions ultérieures.

Comme on ne sait pas se positionner sur l'écran, il faut donc afficher ligne par ligne. On se rend compte rapidement qu'il nous faudra donc faire deux boucles imbriquées : l'une pour les lignes, l'autre pour les colonnes. On peut réaliser une boucle qui s'appuie sur l'affichage à l'écran et qui inclut donc la bordure correspondant aux numéros. Il suffit alors de tester les valeurs des indices de boucle des lignes et colonnes pour savoir si :

- on n'affiche rien (les quatre coins) ;
- on affiche un numéro de ligne ou de colonne (les bordures) ;
- on afficher le contenu d'une case (les autres cases).

Une autre solution, celle retenue ici, consiste à séparer la première et la dernière ligne qui consiste à afficher les numéros de colonne. Notons que pour afficher les lignes, on commence par celle de numéro `NB_LIGNES` car les indices de notre tableau correspondent aux numéros de ligne du puissance 4 (on aurait pu prendre la convention inverse).

```
R1 : Raffinage De « afficher »
  | Afficher les numéros de colonnes
  | Pour i <- NB_LIGNES Décrémenter JusquÀ i = 1_LIGNES Faire
  |   | Afficher la ligne i
  | FinPour
  | Afficher les numéros de colonnes

R2 : Raffinage De « Afficher la ligne i »
  | Afficher le numéro de ligne
  | Pour j <- 1 JusquÀ j = NB_COLONNES Faire
  |   | afficher_case(grille[i][j])
  | FinPour
  | Afficher le numéro de ligne
```

Comme l'affichage des numéros de colonne est à faire en début et fin, nous en faisons un sous-programme pour factoriser le code.

```
' Afficher les numéros de colonnes
Sub afficher_numeros_colonnes()
    Dim j As Integer ' index pour parcourir les colonnes de la grille
    AfficherCellule "" 'sauter la 1ere colonne
    For j = 1 To NB_COLONNES
        AfficherCellule j
    Next j
    AllerALaLigne
End Sub

Sub afficher_grille(grille() As Integer)
'ROLE : affiche la grille du jeu et son contenu
'ENTREE grille
```

```
14      Dim i As Integer, j As Integer  ' index pour parcourir les cases de la grille
15      EffacerEcran TITRE
16      afficher_numeros_colonnes
17      For i = NB_LIGNES To 1 Step -1 'on affiche d'abord la ligne supérieure
18         'afficher le numéro de ligne à gauche
19          AfficherCellule i
20         'afficher la ligne i
21          For j = 1 To NB_COLONNES
22              ColorerCellule couleur_case(grille(i, j))
23              AfficherCellule "" 'symbole_case(grille(i, j)) ' pour avoir o ou *
24          Next j
25         'afficher le numéro de ligne à droite et passer à la ligne
26          AfficherCellule i
27          AllerALaLigne
28      Next i
29      afficher_numeros_colonnes
30  End Sub
```

Exercice 4 : Décider si un coup est possible

Écrire un sous-programme qui indique s'il est possible de jouer dans une colonne donnée.

Exemple : Sur l'exemple de la figure 1, il est possible de jouer les colonnes 1, 2, 3, 5, 6 ou 7. Il n'est pas possible de jouer la colonne 4.

Solution : Ce sous-programme prend en paramètre :

- la grille (**in**) ;
- le numéro de colonne envisagé (**in**) ;
- l'information de si le coup est possible (**out**).

Un seul paramètre en **out**, d'autres paramètres en **in**. On en fait donc une fonction.

```
1  Fonction est_coup_possible(grille: in Grille; colonne: in Entier): Booléen Est
2      -- Est-ce qu'il est possible de jouer dans la colonne donnée
3      -- de la grille ?
```

Pour savoir s'il est possible de jouer, il suffit de regarder si la colonne est pleine ou non, c'est-à-dire s'il sa dernière case, celle du haut, est vide ou non. Si elle est vide, on peut jouer dans la colonne. Si elle n'est pas vide, on ne peut pas y jouer. Notons que si elle est vide, ça ne veut pas dire que le pion restera dans cette case. Il tombera...

On en déduit donc naturellement le code.

```
1  Function est_coup_possible(grille() As Integer, ByVal colonne As Integer) As Boolean
2  'ROLE : renvoie vrai si la colonne spécifiée peut être jouée, c'est-à-dire si
3  '       elle n'est pas encore pleine
4  'ENTREE grille, colonne
5      est_coup_possible = grille(NB_LIGNES, colonne) = VIDE 'test dernière ligne
6  End Function
```

Exercice 5 : Lâcher un jeton

Lorsqu'un joueur joue, il lâche un jeton de sa couleur au dessus d'une colonne de la grille. En raison de la force gravitationnelle, le jeton tombe dans la grille jusqu'à ce qu'il soit arrêté par un autre jeton ou le fond de la grille.

Écrire un sous-programme qui réalise cette opération : il calcule le nouvel état de la grille lorsqu'un jeton est lâché au dessus d'une colonne donnée.
Exemple : Partant de la figure 1, si les rouges jouent la colonne 5, le nouvel état de la grille est celui donné par la figure 2.

```
   1 2 3 4 5 6 7
6        *        6
5        o        5
4      o o   *    4
3      * o   o o  3
2    * o * * o *  2
1  * o * * o o *  1
   1 2 3 4 5 6 7
```

FIGURE 2 – Les rouges ont joué la colonne 5

Solution : Le sous-programme prend en paramètre :
- la grille (**in out**) ;
- la colonne donnée (**in**) ;
- le jeton, ou plutôt sa couleur (**in**).

Comme il y a un paramètre en **in out**, nous en faisons une procédure.

```
1  Procédure lâcher(grille: in out Grille;
2                  colonne: in Entier;
3                  couleur: in Case) Est
4      -- La couleur joue un pion dans la colonne.
5      --
6      -- Nécessite
7      --  est_coup_possible(grille, colonne)
```

Le principe de l'algorithme est alors de regarder jusqu'où va tomber le jeton, soit il n'y a encore aucun pion et il est donc en fond de grille, soit il est dans la dernière case qui est vide. En fait, le plus simple est de partir du fond de la colonne et de chercher la première case vide en remontant. Comme on sait qu'il est possible de jouer dans la colonne (précondition), on n'a pas besoin de tester le débordement de la colonne.

```
1  R1 : Raffinage De « lâcher »
2    | Déterminer la ligne de chute
3    | Mettre le pion dans cette case
4
5  R2 : Raffinage De « Déterminer la ligne de chute »
6    | ligne <- 1
7    | TantQue grille[colonne, ligne] <> VIDE Faire
8    |    | ligne <- ligne + 1
9    | FinTQ
```

Remarque : On décide de faire un sous-programme, en l'occurrence une fonction, de « Déterminer la ligne de chute » car nous aurons par la suite l'utilité de trouver cette ligne de chute. Il n'est pas évident ici, du premier coup, de voir qu'il peut être utile d'en faire un sous-programme.

```
Function ligne_de_chute(grille() As Integer, ByVal colonne As Integer) As Integer
'ROLE : renvoie la ligne sur laquelle tombe un jeton lâché à la colonne spécifiée
'HYPOTHESE : on suppose la colonne non pleine
'ENTREE grille, colonne
    Dim ligne As Integer ' index pour parcourir les lignes de la colonne considérée
   'on recherche la 1ere ligne vide. On sait qu'il y en a une par hypothèse
    ligne = 1
    Do While grille(ligne, colonne) <> VIDE
        ligne = ligne + 1
    Loop
    ligne_de_chute = ligne
End Function

Sub lacher(grille() As Integer, ByVal colonne As Integer, ByVal couleur As Integer)
'ROLE : lache sur la grille à la colonne précisée un jeton de la couleur donnée.
'HYPOTHESE : on suppose la colonne non pleine
'ENTREE colonne, couleur
'M.A J. grille : elle contient un pion de plus
    grille(ligne_de_chute(grille, colonne), colonne) = couleur
End Sub
```

Exercice 6 : Compter les jetons alignés

Écrire un sous-programme qui indique quelle est la longueur de l'alignement le plus long qui inclut un jeton donné (repéré par son numéro de colonne et son numéro de ligne). Les alignements doivent être cherchés dans toutes les directions (horizontale, verticale ou en diagonale).

Ce sous-programme sera ensuite utilisé pour déterminer la fin d'une partie (alignement ≥ 4) ou aider l'ordinateur à choisir la colonne où jouer (exercice 8).

Sur l'exemple de la figure 2, la case (1,1) correspond à un alignement de trois jetons en diagonale ; la case (7,1) correspond à un alignement de deux jetons verticalement ; la case (6,2) correspond à trois jetons alignés verticalement ou en diagonale...

Indication : Quel est l'intérêt d'avoir défini une bordure supplémentaire autour de la grille ?

Solution : Ce sous-programme prend en paramètre une grille (**in**), une ligne (**in**) et une colonne (**in**) et détermine le plus long alignement (out). Un paramètre en sortie. Tous les autres en entrée. Nous pouvons donc en faire une fonction.

```
Fonction nb_pions_alignés(grille: in Grille;
                ligne: in Entier;
                colonne: in Entier): Entier
    -- Plus grand nombre de jetons de la même couleur alignés
    -- horizontalement, verticalement ou en diagonale et incluant
    -- le jeton de la position (colonne, ligne).
    --
    -- Nécessite
    --  grille[colonne, ligne] <> VIDE
```

L'algorithme est un peu compliqué, ou tout au moins fastidieux. Il faut regarder dans les quatre directions (horizontalement, verticalement et en diagonale) pour compter l'alignement le plus grand. Pour chaque direction, il faut compter la case (colonne, ligne) et les deux sens de la direction.

Supposons que l'on sache calculer un nombre de pions alignés selon une direction. On peut alors en déduire le code de « nb_pions_alignés »

```
R1 : Raffinage De « nb_pions_alignés »
  | Résultat <- max(nb_pions_dir(grille, colonne, ligne, 1, 1),
  |                 max(nb_pions_dir(grille, colonne, ligne, 1, 0),
  |                     max(nb_pions_dir(grille, colonne, ligne, 1, -1),
  |                         nb_pions_dir(grille, colonne, ligne, 0, 1))))
```

Il ne reste plus qu'à écrire nb_pions_dir. C'est une fonction.

```
Fonction nb_pions_dir(grille: in Grille; colonne, ligne: in Entier;
                      dir_x, dir_y: in Entier) Est
    -- Compter le nombre de pions dans la direction (dir_x, dir_y)
    -- et incluant le pion en (colonne, ligne).
    --
    -- Nécessite
    --  grille[colonne, ligne] <> VIDE
    --  (dir_x <> 0) Ou (dir_y <> 0)    -- direction valide
    --
    -- Assure
    --  Résultat >= 1           -- au moins le jeton de (colonne, ligne)
```

Pour son implémentation, il suffit de compter le jeton de (colonne, ligne) puis compter les jetons de même couleur dans la direction (dir_x, dir_y) et ceux dans la direction opposée (-dir_x, -dir_y).

```
R1 : Raffinage De « nb_pions_dir »
  | couleur <- grille[colonne, ligne]
  | Résultat <- 1
  | Comptabiliser les jetons dans la direction (dir_x, dir_y)
  | Comptabiliser les jetons dans la direction (-dir_x, -dir_y)
```

```
Function nb_pions_dir(grille() As Integer, ByVal ligne As Integer, ByVal colonne As In
                      ByVal dir_x As Integer, ByVal dir_y As Integer) As Integer
'ROLE : renvoie le plus grand nombre de jetons de la même couleur alignés suivant la
'       direction (dir_x, dir_y) et incluant le jeton de la case (ligne,colonne)
'HYPOTHESES : la position spécifiée par ligne, colonne contient bien un jeton.
'             l'une au moins des 2 directions n'est pas nulle.
'ENTREE grille, ligne, colonne, dir_x, dir_y
    Dim l As Integer        ' ligne dans la direction +/- dir_y
    Dim c As Integer        ' colonne dans la direction +/- dir_x
    Dim couleur As Integer  ' la couleur de l'alignement
    Dim nb_pions As Integer ' nombre de pions alignés en cours de calcul

   '1.initialiser
    couleur = grille(ligne, colonne)
    nb_pions = 1 ' le jeton en (ligne, colonne)

   '2.comptabiliser les jetons dans la direction (dir_x, dir_y)
    l = ligne + dir_y
    c = colonne + dir_x
    Do While grille(l, c) = couleur 'boucle finie car les "bords" de la grille sont vi
        nb_pions = nb_pions + 1
```

```
22          l = l + dir_y
23          c = c + dir_x
24      Loop
25
26     '3.Comptabiliser les jetons dans la direction (-dir_x, -dir_y)
27      l = ligne - dir_y
28      c = colonne - dir_x
29      Do While grille(l, c) = couleur 'boucle finie car les "bords" de la grille sont vi
30          nb_pions = nb_pions + 1
31          l = l - dir_y
32          c = c - dir_x
33      Loop
34     '4.renvoi du résultat
35      nb_pions_dir = nb_pions
36  End Function
37
38  Function max(ByVal Entier1 As Integer, ByVal Entier2 As Integer) As Integer
39  'ROLE : renvoie la plus grande des 2 valeurs spécifiée
40      If Entier1 > Entier2 Then max = Entier1 Else max = Entier2
41  End Function
42
43  Function nb_pions_alignes(grille() As Integer, ByVal ligne As Integer, ByVal colonne A
44                            As Integer
45  'ROLE : renvoie le plus grand nombre de jetons de la même couleur alignés horizontaleme
46  '       verticalement ou en diagonale et incluant le jeton de la position (ligne, colo
47  'ENTREE grille, ligne, colonne
48      Dim nb_pions As Integer
49      nb_pions = 0
50      If grille(ligne, colonne) <> VIDE Then
51          ' Déterminer le plus grand alignement
52          nb_pions = max(nb_pions, nb_pions_dir(grille, ligne, colonne, 1, 1))
53          nb_pions = max(nb_pions, nb_pions_dir(grille, ligne, colonne, 1, 0))
54          nb_pions = max(nb_pions, nb_pions_dir(grille, ligne, colonne, 1, -1))
55          nb_pions = max(nb_pions, nb_pions_dir(grille, ligne, colonne, 0, 1))
56      End If
57      nb_pions_alignes = nb_pions
58  End Function
```

Exercice 7 : Conseiller une colonne où jouer

Écrire un sous-programme qui étant donné une grille propose une colonne où jouer. Le numéro de la colonne sera choisi aléatoirement.

Solution : Ce sous-programme est une fonction car il a un seul paramètre en sortie (la colonne conseillée) et un paramètre en entrée (la grille de jeu), éventuellement deux si l'on considère aussi la couleur de celui que l'on doit conseiller (couleur qui n'est pas exploité dans cette version très naïve du conseil).

```
1  Fonction colonne_conseillée_aléatoire(grille: in Grille; couleur: in Case) Est
2      -- colonne conseillée pour jouer
```

Le principe est de tirer une colonne au hasard jusqu'à ce qu'elle correspondent à une colonne où l'on peut jouer.

```
Sub colonne_conseillee_aleatoire(grille() As Integer, colonne As Integer)
'ROLE : calcule une colonne conseillée pour jouer le prochain coup
'PRINCIPE : la colonne conseillée est choisie au hasard entre les colonnes possibles
'ENTREE grille
'SORTIE colonne
    Do
        colonne = Int(Rnd * NB_COLONNES + 1)
    Loop Until est_coup_possible(grille, colonne)

End Sub
```

Exercice 8 : Améliorer le conseil

Pour améliorer le conseil, on ajoute une contrainte supplémentaire : conseiller la colonne qui permet l'alignement le plus long.

Remarque : Cette contrainte implique que si un coup gagnant existe, la colonne conseillée conduira à la victoire.

Exemple : Partant de la figure 2, les jaunes joueront en colonne 5. Ceci provoque un alignement de 4 pions et donc la victoire.

Solution : Les paramètres sont les mêmes que pour le conseil précédent sauf que la couleur du joueur est ici nécessaire.

```
Fonction colonne_conseillée(grille: in Grille; couleur: in Case) Est
    -- colonne conseillée pour jouer
```

Le principe du conseil est un peu plus compliqué car il faut proposer le coup correspondant à l'alignement le plus grand. On a déjà écrit un sous-programme pour calculer un alignement mais il suppose que l'on ait déjà jouer dans la case. Il nous faut donc jouer dans la case. Et si l'on joue, il faut également pouvoir annuler le coup pour ne pas fausser la partie.

```
R1 : Raffinage De « colonne_conseillée »
  | max <- 0            -- alignement maximal trouvé
  | Pour c <- 1 JusquÀ c = NB_COLONNES Faire
  |  | Jouer en c pour couleur
  |  | Si alignement > max Alors
  |  |  | max <- alignement
  |  |  | Résultat <- c
  |  | FinSi
  |  | Annuler le coup en c
  | FinPour
```

```
Sub colonne_conseillee(grille() As Integer, ByVal couleur As Integer, colonne As Integ
'ROLE : calcule une colonne conseillée pour jouer le prochain coup
'PRINCIPE : la colonne conseillée est celle qui permet :
'            - en priorité d'empecher l'adversaire de gagner au coup suivant
'            - sinon, d'obtenir l'alignement le plus long.
'ENTREE grille, couleur (couleur du joueur qui va jouer)
'SORTIE colonne
    Dim c As Integer          ' index de parcours des colonnes
    Dim nb_pions As Integer   ' Nombre de pions alignés pour c
    Dim max As Integer        ' le nb max de pions alignés (donc de resultat)
    Dim ligne   As Integer    ' conserver la ligne de chute du pion lâché
```

```
12      Dim couleur_adverse As Integer ' couleur de l'adversaire
13      Dim parer As Boolean       ' mis a vrai si l'on doit parer a un coup de l'adversair
14                                 ' lui permettant de gagner au coup suivant
15
16      couleur_adverse = NB_JOUEURS - couleur + 1
17      parer = False
18      max = 0 ' nécessairement un minorant strict
19      c = 1 ' colonne à tester
20      Do While Not parer And c <= NB_COLONNES
21          If est_coup_possible(grille, c) Then
22              ' déterminer la ligne de chute
23              ligne = ligne_de_chute(grille, c)
24
25              ' simuler la pose du pion adverse pour voir si ca le fait gagner
26              grille(ligne, c) = couleur_adverse
27              If nb_pions_alignes(grille, ligne, c) >= NB_ALIGNES_GAGNANT Then
28                  colonne = c ' coup à parer d'urgence !
29                  parer = True
30
31              Else
32                  ' simuler la pose de son propre pion
33                  grille(ligne, c) = couleur
34
35                  ' évaluer le coup
36                  nb_pions = nb_pions_alignes(grille, ligne, c)
37
38                  ' choisir le coup (par rapport au précédent)
39                  If (nb_pions > max) Or (nb_pions = max And Rnd > 0.5) Then
40                      max = nb_pions
41                      colonne = c
42                  End If
43              End If
44
45              ' retirer le coup testé
46               grille(ligne, c) = VIDE
47
48          End If
49
50          ' passer à la colonne suivante
51          c = c + 1
52      Loop
53  End Sub
```