



Le Club Developpez.com n'affiche que des publicités IT, discrètes et non intrusives.

Afin que le Club puisse rester gratuit, nous vous serions reconnaissant d'ajouter Developpez.com dans la liste d'exceptions de votre bloqueur de publicité.

Les Défis DELPHI - Jouez à Puissance 4 !



Table des matières

- I. Sujet du défi
 - I-A. Prérequis
 - I-B. Les objectifs du défi
- II. La solution du défieur
 - II-A. Introduction
 - II-A-1. TJoueur
 - II-A-2. TPlateau
 - II-B. Algorithme général
 - II-B-1. Étapes principales pendant le déroulement du jeu
 - II-B-2. Analyse du dernier jeton joué
 - II-C. Interface
 - II-C-1. Non graphique
 - II-C-1-a. Support
 - II-C-1-b. Affichage

Le troisième défi proposé par l'équipe Delphi est la création du jeu Puissance 4 !

7 commentaires ★★★★★

Article lu 4355 fois.

L'auteur

Équipe Delphi 

L'article

Publié le 12 décembre 2006 - Mis à jour le 21 avril 2013

TOUT PUBLIC

Version PDF Version hors-ligne

ePub, Azw et Mobi

Liens sociaux



Partager

I. Sujet du défi▲

Le troisième défi proposé par l'équipe Delphi est la création du jeu du jeu Puissance 4 !

Il s'agit de réaliser le jeu selon les règles du Puissance 4, afin que deux joueurs sur le même ordinateur puissent s'affronter.

I-A. Prérequis▲

Pour réaliser ce défi, une simple édition personnelle de Delphi suffit. Pas besoin d'avoir les bibliothèques spécifiques aux versions Pro/Entreprise/Architecte !

Certaines versions personnelles de DELPHI sont disponibles au téléchargement sur Developpez.com.

En utilisant ce site, vous acceptez l'utilisation de cookies permettant de vous proposer des contenus et des services adaptés à vos centres d'intérêts - [J'accepte](#)

- la F.A.Q. DELPHI ;
- les Sources DELPHI ;
- les tutoriels DELPHI ;
- les forums DELPHI.

I-B. Les objectifs du défi▲

Le logiciel doit comporter les éléments suivants :

1. créer l'interface utilisateur du jeu ;
2. respecter les règles du Puissance 4 :
 1. établir un plateau de 7 colonnes sur 6 lignes,
 2. chaque joueur possède 21 jetons de sa couleur,
 3. chaque jeton joué est lâché au sommet d'une colonne et prend sa place dans la première cellule libre en partant du bas,
 4. Le premier joueur alignant 4 jetons dans sa couleur (horizontal, vertical ou diagonal) a gagné ; si les 42 jetons sont joués, la partie est nulle.
3. permettre à deux joueurs sur le même ordinateur de jouer l'un contre l'autre ;
4. laissez libre cours à votre imagination et proposez vos propres fonctionnalités.

Les participants doivent respecter les règles du défi et le déroulement du défi, et plus précisément que « l'utilisation de composantes ou bibliothèques autres que celles fournies en standard par Borland sont interdites, qu'elles soient commerciales, freewares, open source, etc. ».

II. La solution du défieur▲

TicTacToe, selon les règles du défi, a réussi à mettre au point un tel logiciel avant que le défi ne soit lancé sur le forum.

II-A. Introduction▲

Le programme s'articule autour des deux classes **TPlateau** et **TJoueur** :

II-A-1. TJoueur▲

La classe **TJoueur** garde les informations relatives à chaque joueur.

- **Plateau** : le plateau d'appartenance ;
- **Snom** : son nom ;
- **Icamp** : son camp (actuellement 1 = rouge, 2 = jaune) ;
- **IjetonsRestants** : son nombre de jetons restants au cours d'une partie.

Deux méthodes également, **TJoueur.Init** et **TJoueur.Jouer**, permettent de faire interagir le joueur au cours de la partie.

II-A-2. TPlateau▲

La classe **TPlateau** garde les informations du plateau, principalement :

- **Joueur[x]** : les joueurs de classe **TJoueur**, indicés de 1 à 2 ;
- **Grille[col,lig]** : grille virtuelle aux dimensions du puissance 4 (7 colonnes par 6 lignes). Les valeurs des cellules sont : 0 si la cellule est vide, 1 ou 2 pour spécifier le camp du jeton ;
- **DernierJeton** : les coordonnées du dernier jeton joué ;
- **Icurseur** : curseur en cours du lâcher de jeton ;
- **Ijoueur** : joueur actuellement en cours (1 ou 2). Vaut 0 si aucune partie n'est en cours ;

Ainsi que des informations pour l'affichage :

- **BmpFond**, **BmpJeu** sont les images de fond et de grille de Puissance 4 ;
- **BmpDoubleFond** est l'image de fond, pour gérer un affichage correct pendant les animations des jetons ;
- **BmpPion[x]** sont les images des jetons.

TPlateau intègre également ses propres méthodes pour le faire vivre, principalement les méthodes :

- **NouveauJeu** : initialise une partie et ses joueurs ;
- **Jouer** : lâche un jeton selon l'état courant du jeu (à la position **Icurseur**) et gère l'analyse des jetons alignés ;
- **Fin** : partie terminée, affiche le message de match nul ou de victoire.

Les autres méthodes sont des outils ; elles seront expliquées par la suite.

II-B. Algorithme général▲

II-B-1. Étapes principales pendant le déroulement du jeu▲

1. Le plateau est initialisé avec **IJoueur** = x (x = 1, par exemple, si le joueur N°1 commence) ;
2. La propriété **Icurseur** est mise à jour en temps réel, lorsque la souris se déplace sur le plateau ;
3. Le clic souris sur le plateau déclenche le lâcher de jeton par **Plateau.Jouer** :
 1. le joueur courant **IJoueur** lâche un jeton dans la colonne courante **Icurseur**,
 2. **TPlateau.CaseVide** indique quelle est la cellule libre à remplir ; l'algorithme est suffisamment simple à comprendre en lisant le code.
4. L'analyse de gain / nul est lancée avec la méthode **Plateau.ResultatDernierJeton**, expliquée ci-dessous ;
5. Si la partie continue, le programme boucle en 2.

TPlateau.CaseVide permet de savoir, en fonction de la colonne du lâcher de jeton, quelle est la case finale du jeton.

La recherche d'une case vide est faite par un parcours séquentiel dans la colonne désignée, de haut en bas, à la recherche d'une cellule déjà remplie par un jeton.

II-B-2. Analyse du dernier jeton joué▲

En utilisant ce site, vous acceptez l'utilisation de cookies permettant de vous proposer des contenus et des services adaptés à vos centres d'intérêts - [J'accepte](#)

J'ai choisi cette méthode car un minimum de jetons seront soumis à des tests ; le code est également assez compact.

De plus, il est permis d'imaginer de faire évoluer le plateau en nombre de cellules et/ou en nombre de dimensions sans changer l'algorithme.

Le vocabulaire utilisé est : **SENS** = axe non orienté, **DIRECTION** = axe orienté.

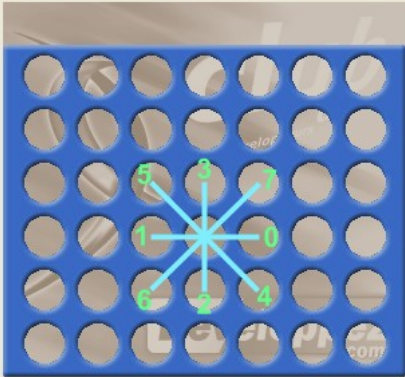
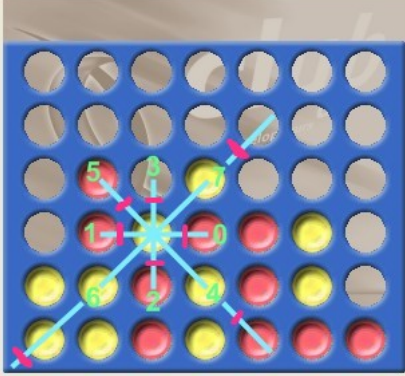
L'analyse repose sur le « comptage » des jetons dans les huit **directions** de cellules identiques au dernier jeton joué.

Puis ensuite au comptage des jetons pour les quatre **sens**.

Pour effectuer ce comptage dans les quatre sens, la somme des compteurs de directions deux à deux opposées et effectuées + 1 (le jeton central).

Si, pour un des quatre sens, le compteur vaut au moins 4, le joueur correspondant au dernier jeton a gagné.

Voici une présentation un peu plus parlante du fonctionnement de comptage.
Les captures sont prises du programme graphique, mais cela revient strictement au même pour la version non graphique :

Captures d'écran	Commentaires
	<p>La constante DIR_INCR spécifie les directions par incrémentation sur les deux</p> <p>Sélectionnez</p> <pre>DIR_INCR : array[0..7] of TPoint = ((x:1;y:0),(x:-1;y:0), // horizontale (x:0;y:1),(x:0;y:-1), // vertical (x:1;y:1),(x:-1;y:-1), // oblique slash (x:-1;y:1),(x:1;y:-1)); // oblique anti-slash</pre> <ul style="list-style-type: none">• Le sens horizontal est spécifié par les directions 0 et 1 ;• Le sens vertical est spécifié par les directions 2 et 3 ;• Le sens oblique \ est spécifié par les directions 4 et 5 ;• Le sens oblique / est spécifié par les directions 6 et 7.
 <p>Les marques rouges marquent l'arrêt du comptage dans la direction ; la cellule n'est plus identique à la cellule centrale.</p>	<p>Comptage dans chaque direction. Et ensuite, comptage dans les quatre sens : l'oblique / ajouté de 1 vaut au moins 4, les jaunes gagnent.</p> <ul style="list-style-type: none">• Dir 0 = 0 ;• Dir 1 = 0 ;• Dir 2 = 0 ;• Dir 3 = 0 (en pratique, on pourrait se passer de compter dans cette direc• Dir 4 = 1 ;• Dir 5 = 0 ;• Dir 6 = 2 ;• Dir 7 = 1 ;• Sens horizontal = Dir 0 + Dir 1 = 0 ;• Sens vertical = Dir 2 + Dir 3 = 0 ;• Sens oblique \ = Dir 4 + Dir 5 = 1 ;• Sens oblique / = Dir 6 + Dir 7 = 3.

II-C. Interface▲

Il y a deux versions : une version utilisant un **TStringGrid** et une version graphique.

Bien qu'il ne soit nullement nécessaire de faire le programme graphiquement, j'expose les deux solutions.

II-C-1. Non graphique▲

II-C-1-a. Support▲

Un **TStringGrid** est utilisé pour afficher les jetons.

Une ligne supplémentaire de titre est ajoutée, par rapport aux dimensions de la grille du Puissance 4, afin de pouvoir afficher le jeton à lâcher.

Chaque cellule peut prendre respectivement les valeurs : chaîne vide, lettre 'X', lettre 'O' en fonction de **TPlateau.Grille** valant respectivement 0, 1, 2.

II-C-1-b. Affichage d'un jeton▲

La procédure suivante montre l'affichage d'un jeton dans le **TstringGrid**, le paramètre **ICamp** étant le camp du joueur, la valeur 0 efface le jeton.

Sélectionnez

```

case ICamp of
  1 : CCar := ' X';
  2 : CCar := ' O';
  else CCar := '';
end;
// le tableau logique commence en indice -1 pour le laché de jeton
StringGrid.Cells[ PPos.X, PPos.Y + 1 ] := CCar;
end;

```

II-C-1-c. Animation▲

L'animation est réalisée par la méthode **TPlateau.PionGlisser**, qui s'occupe d'afficher et d'effacer le jeton successivement de la position initiale à la position finale.

La position finale est récupérée grâce à la méthode **TPlateau.CaseVide**.

II-C-2. Graphique▲

II-C-2-a. Support▲

Pour établir une animation fluide, j'ai eu besoin de deux **TImage**, pour contenir les deux bitmaps, nommés **BmpFond** et **BmpJeu**.

La classe **TPlateau** fait abstraction des **TImage** et ne se sert que des bitmaps contenues par ces **TImages**.

BmpFond représente le fond gris avec le logo Developpez.com. C'est sur cette image que sera dessiné le jeton.

BmpJeu représente le bitmap de la grille. Le **TImage** supportant ce bitmap a sa propriété **Transparent = True** afin de laisser **BmpFond** visible dans les trous.

De plus, il est nécessaire que le composant **TImage** supportant **BmpFond** soit derrière le **TImage** de **BmpJeu** dans l'ordre de création.

Une troisième bitmap nommée **BmpDoubleFond** est utilisée afin de garder l'image originale du fond.

II-C-2-b. Affichage d'un jeton▲

L'affichage d'un jeton est la recopie de la bitmap du jeton **BmpPion[x]** dans la bitmap **BmpFond**, à la bonne position.

La bonne position est calculée en réalisant une conversion entre coordonnées logiques du plateau et coordonnées physiques en pixels sur la Bitmap.

Nous constatons donc une altération de la bitmap **BmpFond**.

Pour « effacer » le jeton de **BmpFond**, il suffit de recopier **BmpDoubleFond** (qui est toujours intact) dans **BmpFond**.

Cette copie ne concerne que la zone (le rectangle) contenant le jeton précédemment affiché.

Une seule routine affiche ou efface un jeton dans **BmpFond**. **BmpJeu** reste intact durant toute la partie.

Sélectionnez

```

procedure TPlateau.PionAfficher( ICamp: Integer; PPos: TPoint; IDelta: Integer = 0 )
var
  RRect: TRect;
  BmpSource: TBitmap;
begin
  // zone de dessin
  RRect.Left := OFFSET.X + SEGMENT.X * PPos.X;
  RRect.Top := OFFSET.Y + SEGMENT.Y * PPos.Y + IDelta;
  RRect.Right := RRect.Left + JETON.X;
  RRect.Bottom := RRect.Top + JETON.Y;
  // Type de dessin
  if ICamp > 0
  then BmpSource := BmpPion[ ICamp ]
  else BmpSource := Nil;
  // Affichage du dessin
  if Assigned( BmpSource )
  then BmpFond.Canvas.Draw( RRect.Left, RRect.Top, BmpSource ) // affichage Jet
  else BmpFond.Canvas.CopyRect( RRect, BmpDoubleFond.Canvas, RRect ); // restit
end;

```

Le paramètre **IDelta** permet de dessiner le jeton décalé par rapport aux coordonnées de la cellule fournie, ceci pour l'animation.

II-C-2-c. Animation▲

L'animation est réalisée par la méthode **TPlateau.PionGlisser**, qui s'occupe d'afficher et d'effacer le jeton successivement à différentes positions.

Contrairement à la version non graphique, les différentes positions d'affichage/d'effacement du jeton ne se font plus cellule par cellule, mais avec un « cran » défini au niveau du pixel.

Voici les étapes nécessaires pour donner l'illusion du glissement de jeton :

- le jeton est dessiné dans **BmpFond** à la hauteur Y ;
- le jeton est effacé de **BmpFond** à la hauteur Y.
- L'effacement est en réalité une copie partielle (zone du jeton actuelle) de **BmpDoubleFond** dans **BmpFond** ;
- Le jeton est dessiné dans **BmpFond** à la hauteur Y = Y + delta

En utilisant ce site, vous acceptez l'utilisation de cookies permettant de vous proposer des contenus et des services adaptés à vos centres d'intérêts - [J'accepte](#)

Après une conversion des coordonnées de grille en coordonnées en pixels sur les bitmaps, il est donc simple de réaliser le glisser du jeton.

Pour donner une impression d'accélération du jeton, une grossière augmentation de **delta** à chaque tour d'affichage est réalisée.

La position finale (le dernier Y + delta) ne tombant pas forcément sur la position exacte de la cellule à atteindre, le jeton est effacé et réaffiché une dernière fois dans sa position finale exacte. Le jeton dans sa position finale n'est pas effacé et reste sur **BmpFond** (jusqu'à une nouvelle partie).

II-C-3. Commun : graphique ou non▲

Pour éviter ce que l'on appelle le « Flickering » ou le scintillement du jeton pendant son animation, je laisse soin aux composants VCL, le **TPanel** contenant le **TStringGrid** ou les **TImage**, de gérer le double-buffering.

Pour cela, la propriété **Conteneur.DoubleBuffered** est activée à **True**.

L'image générale n'est réaffichée en visuel qu'une fois complètement constituée dans un buffer.

II-C-3-a. Autres▲

II-C-3-a-i. Clignotement du jeton▲

Un timer est utilisé pour gérer le clignotement du jeton.

Pour afficher et effacer le jeton dans sa position courante, les mêmes routines d'affichage du jeton sont utilisées.

II-C-3-a-ii. Empêcher le « double lâcher »▲

Pour éviter le double lâcher d'un jeton, un second lâcher ne peut intervenir qu'après un temps minimum après un premier lâcher.

II-D. Conclusion▲


Un défi amusant et accessible pour beaucoup.

Bravo aux nombreux participants, qui se sont surpassés pour imaginer des améliorations surprenantes. :)

Rendez-vous au prochain défi !

II-E. Téléchargement▲

II-E-1. Version non graphique▲

Téléchargez le code source de TicTacToe	 Télécharger
Téléchargez l'exécutable de TicTacToe	 Télécharger

II-E-2. Version graphique▲

Téléchargez le code source de TicTacToe	 Télécharger
Téléchargez l'exécutable de TicTacToe	 Télécharger

III. Les résultat du défi▲

III-A. Conclusion▲

Tout d'abord, l'équipe tient à féliciter tous les participants d'une part pour le temps qu'il ont accordé à ce défi, et d'autre part pour la patience dont ils ont fait preuve pour attendre les résultats.

Vous étiez tout de même 17 participants, et l'équipe se réjouit de ce succès.

À noter qu'il nous a été particulièrement difficile de départager les deux participants arrivant en tête de classement !

Un grand BRAVO à

Cl@udius

pour son programme, qui a réussi à allier simplicité et jouabilité du jeu, avec un code source clair et cohérent.

L'équipe du Défi Delphi offre au vainqueur un chèque cadeau.

III-B. Grille de notation▲

En utilisant ce site, vous acceptez l'utilisation de cookies permettant de vous proposer des contenus et des services adaptés à vos centres d'intérêts - [J'accepte](#)

Cette grille a été établie et utilisée par l'équipe pour noter chaque participant.

Chaque rubrique est précédée du nombre de points sur laquelle elle a été notée.

- **Interface :**
 - 10 - Mission accomplie ;
 - 5 - Ergonomie (rapport design/utilisabilité) ;
 - 5 - Design ;
 - 10 - Jouabilité / simplicité ;
 - 10 - Bogues et dysfonctionnements.
- **Programme :**
 - 10 - Clarté générale/élégance du code ;
 - 10 - Optimisations/code optimal ;
 - 10 - Indépendance des fonctions ;
 - 10 - Warnings/Conseils/Bogues.

Total des rubriques = 80 points.

Chaque noteur a ramené les notes de cette grille sur 20 points, (très scolaire tout ça ;-)

Les notes présentées dans la section suivante sont les moyennes de tous les noteurs confondus.

III-C. Classement des participants et téléchargements▲


Classement des participants et moyennes des notes sur 20.

Les notes sont établies selon le fonctionnement décrit en section précédente.

Participant	Moyenne	Classement	Source	Exécutable
Cl@udius	18.94	1	 Télécharger	 Télécharger
Andry	18.68	2	 Télécharger	 Télécharger
dadane	17.80	3	 Télécharger	 Télécharger
DevilSpirit	17.76	4	 Télécharger	 Télécharger
sidahmed	17.02	5	 Télécharger	 Télécharger
Haywire (Mushroom7)	17.02	5	 Télécharger	 Télécharger
fred978	16.80	7	 Télécharger	 Télécharger
patquoi	16.10	8	 Télécharger	 Télécharger
Masterglob	15.76	9	 Télécharger	 Télécharger
OutOfRange	14.66	10	 Télécharger	 Télécharger
Bleys	14.46	11	 Télécharger	 Télécharger
Mixermode	14.26	12	 Télécharger	 Télécharger
Jeannot Alpin	13.90	13	 Télécharger	 Télécharger
David	13.83	14	 Télécharger	 Télécharger
korntex5	13.82	15	 Télécharger	 Télécharger
edam	12.00	16	 Télécharger	 Télécharger
Blue Strike (inactif)	11.41	17	 Télécharger	 Télécharger
Sub0	hors concours		 Télécharger	 Télécharger
Whiler	hors concours		Page de l'auteur et téléchargementPage de Whiler sur Developpez	

IV. La solution du vainqueur▲

Cl@udius nous présente sa solution.

Vous avez aimé ce tutoriel ? Alors partagez-le en cliquant sur les boutons suivants :  Partager

Copyright © 2006 Équipe Delphi Developpez LLC. Tous droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site ni de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

En utilisant ce site, vous acceptez l'utilisation de cookies permettant de vous proposer des contenus et des services adaptés à vos centres d'intérêts - [J'accepte](#)

