

Les Défis DELPHI - Jouer au démineur !



par [Equipe DELPHI](#)

Date de publication : 25 Juillet 2006

Dernière mise à jour :

L'équipe [DELPHI](#) vous propose de revivre le tout premier défi qui vous as été proposé sur le [Forum DELPHI](#) et qui a connu un très fort engouement !

Voici donc, tout ce que vous avez toujours voulu savoir sur le premier Défi DELPHI ;) !

Téléchargement de l'article au format PDF

I - Sujet du défi

I-A - Pré-requis

I-B - Les objectifs du défi

II - Le défi

III - La solution du défieur

III-A - Comment m'est venue cette idée de défi ?

III-B - Le code source en détail

III-C - Conclusion

III-D - Les sources du défieur

IV - La solution du vainqueur

IV-A - Le témoignage du vainqueur

IV-A-1 - Le Contrôle du démineur :

IV-A-2 - Structure des données

IV-A-3 - Algorithmes

IV-A-3-a - Principe général

IV-A-3-b - Algorithme simple - cellules indépendantes

IV-A-3-c - Algorithme de résolution d'un îlot.

IV-A-3-d - Autres Points

IV-A-4 - Mode debug.

IV-A-5 - Conclusion

IV-B - Les sources de la solution

V - Les solutions des challengers

V-A - La solution d'Anapurna

V-B - La solution de Fabrice ROUXEL 1

V-C - La solution de Portu

Téléchargement de l'article au format PDF

--	--

I - Sujet du défi

Pour le premier défi proposé par [l'équipe Delphi](#) il s'agissait de créer un logiciel qui joue au démineur... pas avec vous, non, à votre place ! ...avec le démineur intégré à Windows !

I-A - Pré-requis

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

Certaines versions personnelles de DELPHI sont disponibles au téléchargement dans la page [Freeware](#) de la [rubrique DELPHI](#) de www.developpez.com !

Il était également nécessaire de savoir farfouiller sur le site de www.developpez.com dans la [rubrique DELPHI](#) et plus particulièrement dans la [FAQ F.A.Q. DELPHI](#), dans les [SOURCES DELPHI](#), dans les [tutoriels DELPHI](#) et dans les [forums DELPHI](#).

I-B - Les objectifs du défi

Les objectifs du défi sont très clairs, l'application doit savoir :

- 1 lancer Winmine.exe
- 2 cliquer sur les bonnes cases (donc analyser les cases déminées, non déminées, puis jouer le coup en conséquence.
- 3 sélectionner le niveau de difficulté
- 4 démarrer sa petite partie
- 5 entrer son nom s'il gagne (si votre logiciel à gagné, c'est que vous êtes suffisamment doué en programmation pour ajouter cette fonctionnalité cruciale)
- 6 fermer Winmine
- 7 laissez libre court à votre imagination et proposez vos propres fonctionnalités.

Les participant 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 - Le défi

Comme pour tous les défis, ce premier défi c'est déroulé sur le [forum DELPHI](#).

Vous pouvez désormais retrouver sur [cette page](#) l'archive du sujet concernant le défi du démineur.

III - La solution du défieur

[Waskol](#), selon les règles du défi, a réussi à mettre au point un tel logiciel avant que le défi ne soit lancé sur le forum.

Le code de son application a été révélé à l'issue du défi.

[Waskol nous présente en détail sa solution.](#)

III-A - Comment m'est venue cette idée de défi ?

L'idée m'est venue en jouant une partie de démineur (comme quoi...)... Je me suis dit alors qu'il serait assez marrant (l'était-ce ?) de réaliser un programme capable de jouer tout seul :p

En plus, en aidant les uns et les autres sur le forum, j'avais déjà plus ou moins abordé les différentes possibilités qui s'offraient à moi pour réaliser au moins une maquette du projet. Le code source final est très peu éloigné du premier jet que j'ai réalisé : j'ai en effet essayé de garder à l'esprit de préserver une certaine modularité pour la partie algorithmique (j'y reviens plus loin).

Lors donc et après avoir jeté un oeil attentif à notre FAQ, je me suis aperçu qu'en cherchant un peu sur le forum et la FAQ, il était tout à fait possible de trouver ça et là les différentes briques de code nécessaires à la réalisation d'un tel programme.

III-B - Le code source en détail

Je vous propose [sous ce lien](#) une description complète de ma solution.

Vous y trouverez les ressources qui m'ont inspirées ainsi que les algorithmes qui animent mon programme.

Bonne lecture !

III-C - Conclusion

Non seulement ce défi fut une rude épreuve, mais écrire ces lignes en fut une autre tout aussi douloureuse.....

Pendant le défi, il a été intéressant de noter que la partie purement technique (API Windows) a été rapidement cernée par les challengers; C'est à partir de ce moment que le défi est devenu un véritable défi et a pris toute sa dimension : nous n'avions plus rien d'autre à nous raccrocher que notre imagination pour essayer de faire mieux que l'adversaire !

Sur ce, mes concurrents, et en particulier **TicTacToe**, ont été des challengers hors pair. Je ne peux qu'applaudir les trésors d'ingéniosité qu'ils ont déployés et la ténacité dont ils ont fait preuve. Il m'en on vraiment fait voir de toutes les couleurs !!!

80% du temps passé à développer ce logiciel aussi inutile que surprenant a été de trouver les mécanismes qui permettaient de résoudre une grille de démineur, c'est dire...

Mise à part cette facette du défi, tout comme mes concurrents j'ai moi-même appris certains aspects de la programmation que je ne connaissais pas, et ça a été vraiment enrichissant.

Par exemple je ne m'étais jamais vraiment penché sur les différences notoires qu'impliquait l'utilisation de

PostMessage, à la place de **SendMessage**.

J'ai vraiment passé un bon moment, et je ne compte plus le nombre de fous rires qui m'ont pris en voyant mon programme faire n'importe quoi !

III-D - Les sources du défieur

Téléchargez le code source du défieur	
Téléchargez l'exécutable du défieur	

IV - La solution du vainqueur

Ce premier défi à été remporté par [TicTacToe](#)

IV-A - Le témoignage du vainqueur

[TicTacToe](#) nous présente la solution qu'il a mise en place afin de résoudre ce défi.

IV-A-1 - Le Contrôle du démineur :

Le contrôle du démineur s'articule autour de ces éléments, j'ai appris à les utiliser pour certains, pour d'autres j'avais déjà mes propres outils.

Rubrique	Principe	Utilisation
Démarrage et fermeture de l'application	Démarrage API d'une application	ShellExecute()
Récupération du <i>Handle</i> (identifiant) du démineur	Recherche de la bonne fenêtre dans Windows	EnumWindows()
Reconnaissance des cellules	Lecture d'un ou plusieurs pixels sur case démineur	GetPixel()
Clic dans une cellule (gauche ou droit)	Envoie d'un message avec attente de retour	SendMessage()
Niveau / nouveau jeu	Envoie d'un message touche (sans retour) et parcours dans le menu	keybd_event()

Pour rendre fiable l'application, toutes ces fonctions dites 'de bas niveau', sont rassemblées et ont chacune un validation ou non avant de s'exécuter.

Ceci afin de ne pas perdre le contrôle de l'application en cas d'anomalies (clics fous, Windows bloqué etc...).

IV-A-2 - Structure des données

Pour pouvoir travailler sur le démineur, en minimisant les accès externes, il faut une structure de données qui puisse non seulement refléter la fenêtre du démineur, mais aussi accueillir ses propres données de travail. De surcroît, la taille de la grille est dynamique selon le niveau.

Par habitude, j'utilise trop les TList et les TStringList.

Les différentes structures sont résumées ici:

Élément	Structure
La grille de cellules	Liste de lignes avec 1 ligne = liste de cellules
Une cellule	Classe TCellule avec propriétés et actions propre à une cellule
Les îlots	Liste des îlots avec les cellules associées (voir algo.)
Les solutions	Liste des solutions pour chaque îlot (voir algo)

A noter que pour s'affranchir du problème des cellules sur un bord ou dans les coins, des listes de cellules adjacentes propres à chaque cellule sont construites une fois pour toutes.

Par exemple: une cellule au centre du jeu possède une liste de 8 cellules adjacentes. Une cellule dans un coin possède une liste de 3 cellules adjacentes.

Cela simplifie grandement les parcours et autres tests tout au long du programme où il y a totalement abstraction des positions relatives des cellules.

IV-A-3 - Algorithmes

IV-A-3-a - Principe général

Si la partie n'est pas finie, faire un tour de calcul qui se compose de:

- Lire l'état du démineur et le stocker dans la structure de données
- Etablir à l'aide d'un algo. simple, la statistique de présence d'une bombe sur une cellule vierge (taux = 0% = aucune bombe) (voir [Algorithme simple - cellules indépendantes](#))

2 conclusions:

- des cellules sont à un taux de 0% ou 100%: de manière certaine, on peut poser un drapeau ou déminer : TOUR FINI
- sinon, recherche de solutions
 - construction des îlots
- recherche des solutions (combinaison) sur chaque îlots (voir [Algorithme de résolution d'un îlot.](#))

2 conclusions:

- Un îlot à une seule solution : on démine les cellules trouvées: TOUR FINI
- Un îlot, contient N solutions, alors on effectue un comptage pour chaque cellule des drapeaux apparaissant dans les solutions:
 - Ventilation sur N tours de toutes les cellules de l'îlot
 - Incrémentation pour chaque tour et chaque cellule de 1 si un drapeau est prévu dans la solution

2 conclusions:

- Des cellules ont un compteur à 0: le drapeau n'apparaît jamais quelque soient les solutions trouvées: on démine : TOUR FINI (*exemple)
- Des cellules ont un compteur à N: le drapeau apparaît systématiquement quelque soit la solution: on pose un drapeau : TOUR FINI (*exemple)
- Sinon, on établit un nouveau taux de présence de mine, à l'aide de N, du nombre de cellules de l'îlot, et du nombre de drapeaux moyen des solutions dans l'îlot puis
 - on démine la cellule à taux le plus bas: TOUR FINI (ou PERDU!)

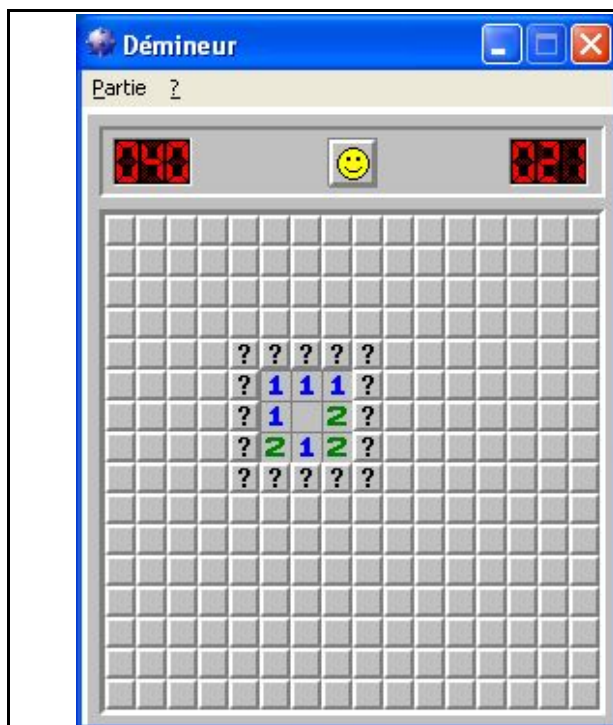
Qu'est-ce qu'un îlot ?

C'est l'ensemble des cellules, dont le changement d'état (drapeaux ou pas) d'une seule cellule, remet en question 1 ou plusieurs cellules de l'îlot.

Il est construit par prise d'une cellule initiale et propagation des cellules adjacentes reliées au moins par une cellule numérotée.

Le choix des îlots a été effectué pour tenter un découpage maximum des différentes zones indépendantes à résoudre dans le démineur. Ceci restreint le nombre de cellules dont il faut trouver des combinaisons de drapeaux valides.

Exemple d'un îlot, avec ses 4 solutions possibles de dépôt de drapeaux.



L'îlot

Îlot: toutes les cellules avec '?' (et Cote = toutes les cellules numérotées)

Ce cas est précisément celui noté par (*exemple).

Aucune solution évidente à priori.

Il y a 4 solutions trouvées par l'algorithme.

Après comptage des drapeaux virtuels pour chaque solution, il y a:

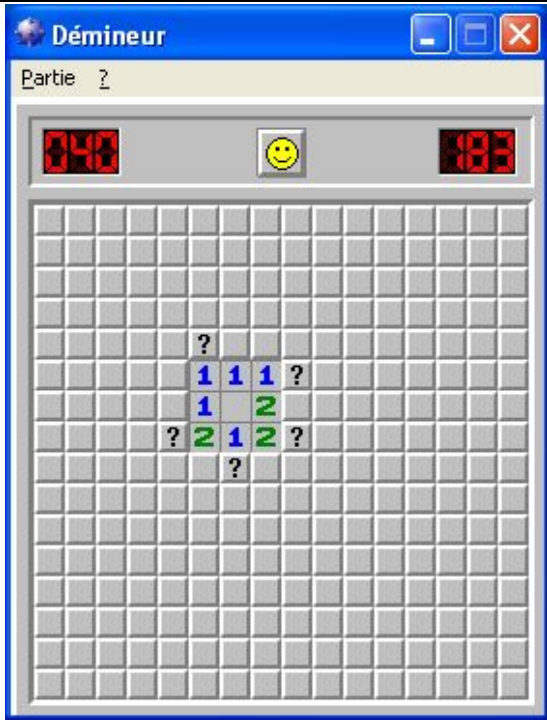
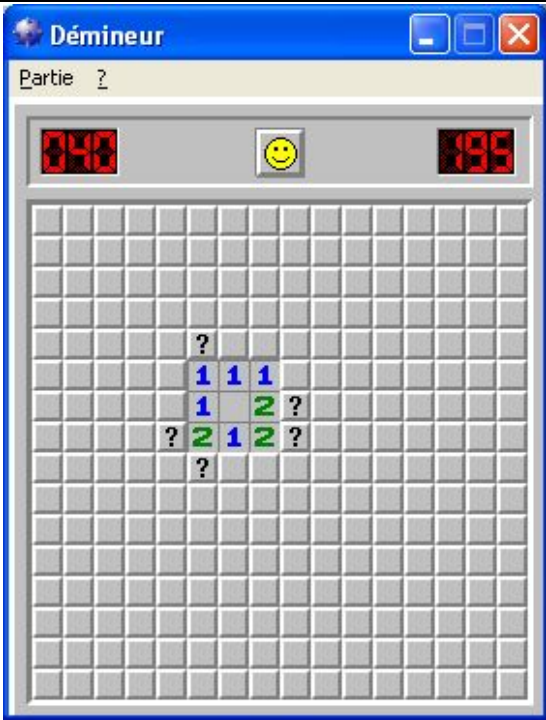
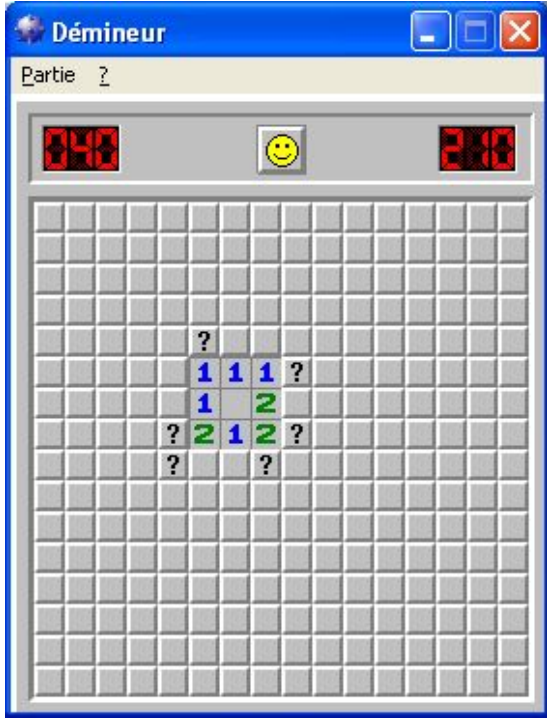
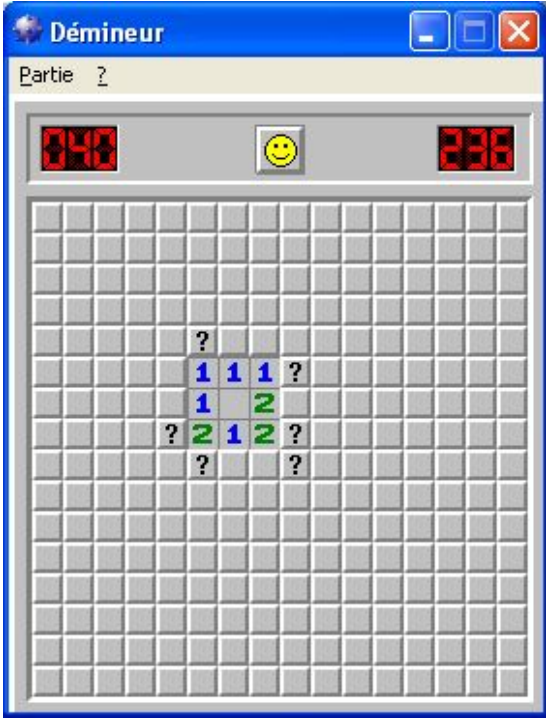
3 cellules qui ont un compteurs à 4.

6 cellules avec un compteurs à 0.

le reste des cellules est entre 1 et 3.

On peut déminer sans risque les 6 cellules à compteur=0

On peut poser un drapeaux sans risque sur les 3 cellules à compteur=4.

 <p><i>Solution 0</i></p>	 <p><i>Solution 1</i></p>
 <p><i>Solution 2</i></p>	 <p><i>Solution 3</i></p>

Une fois cet algorithme global mis en place, le point critique se situe dans la manière de trouver les solutions dans un minimum de temps et pour un îlot donné.

IV-A-3-b - Algorithme simple - cellules indépendantes

Principe: établir un taux statistique pour chaque cellule (vierge) de trouver une bombe - de 0 à 100%.

Autour d'une cellule numérotée: on compte les drapeaux présents, les drapeaux restant et la statistique est triviale.

Taux = (Numéro - Bombes présentes) / Cellules vierges autour du numéro.

IV-A-3-c - Algorithme de résolution d'un îlot.

1ère version: Chercher des combinaisons de drapeaux à l'aide d'un parcours de type CNP.

dont on ne connaît pas à l'avance le 'P' symbolisant le nombre de drapeaux, mais dont on connaît le 'N' qui est le nombre de cellules dans l'îlot.

L'objectif est de ventiler et d'établir toutes les combinaisons possibles des drapeaux sur un îlot.

Or, dès que le nombre de cellules d'un îlot dépasse un seuil critique (ce seuil est très souvent dépassé, surtout en Expert), le nombre d'itérations à effectuer devient draconien.

Soit on laisse calculer de nuit, soit on coupe le calcul au bout d'un certain temps. Mais cela renvoie un résultat (très) erroné.

De plus, il ne faut pas perdre de vue que, dans la plupart des cas, l'ensemble des combinaisons trouvées dans un îlot servira à ré-établir des statistiques, et parfois seulement des résultats certains. Donc trouver un résultat EXACT par rapport à un résultat APPROCHANT (en terme de nombre de solutions trouvées) n'aura que peu d'impact au final.

2ème version: Recherche de solutions non exhaustives, mais le plus possibles.

Au lieu de trouver des solutions en choisissant les cellules où il faut poser des drapeaux par combinaison, autant essayer par des parcours simples.

Une ventilation d'un îlot se compose de:

- La pose d'un drapeau sur une cellule X d'un îlot (toujours possible par définition de l'îlot)
- La ventilation des autres cellules de l'îlot, et pose d'un drapeau à chaque fois que c'est possible

Un parcours se compose de N ventilations, avec X variant sur toutes les cellules afin de donner à chaque cellule sa chance d'avoir un drapeau.

Il s'agit maintenant de trouver les meilleurs ordres de parcours, pour pouvoir ventiler efficacement les cellules.

1er cycle: Ventiler les cellules, en passant à la suivante avec un incrément variable. Dans ce 1er cycle, cet incrément est un nombre 1er afin de faire varier au maximum l'ordre de prise des cellules et de ne pas retomber sur des parcours identiques.

2ème cycle: alternative au 1er cycle.

3ème cycle: Ventiler les cellules aléatoirement (après le dépôt sur 1ère cellule X toujours), pour trouver des solutions supplémentaires.

La majorité des solutions sont trouvées dans le cycle 1. En traçant la courbe du nombre de solutions trouvées (sur Y) par rapport au nombre de parcours de l'algo (X), la courbe est un hyperbole.

Le reste des solutions, 20% grosso-modo sont trouvées dans les cycles 2 et 3.

IV-A-3-d - Autres Points

Il se peut que pour de gros îlots, ces ventilations ne trouvent pas TOUTES les solutions d'un îlot. Cela a peu d'impact au final, car cela débouchera sur un calcul de taux par cellule, et ce taux ne sera que grossier au lieu d'être exact... mais ce n'est qu'un taux de probabilité utilisé comme tel par la suite...

Vu la tête des parcours, les ventilations ont de fortes chances de retomber sur des solutions identiques. Pour ne pas stocker 2 fois la même solution, une liste d'Identificateur de solutions est maintenue à jour. Cette liste garde un CRC pour chaque solution trouvée, basé sur une somme de nombres premiers.

Les îlots sont traités indépendamment. Mais à l'approche de la fin du jeu, il se peut que la somme des drapeaux des solutions des 3 îlots restants par exemple, dépasse le nombre de mines restantes à trouver. Dans ce cas, on fusionne ces 3 îlots en 1 seul et on refait partir l'algorithme normalement (qui lui contrôle toujours qu'il n'y a pas de dépassement).

IV-A-4 - Mode debug.

Il permet de visualiser les solutions trouvées avec l'algorithme présenté.

Taper 'dvp' dans la zone réservée au nombre de parties, puis valider avec 'ENTER'.

Puis cliquer sur le logo bombe.

- Lancer Winmine d'abord avec le bouton pour tout initialiser correctement (même si le démineur est déjà ouvert).
- Le bouton 'Jouer un tour' simule 1 tour de calcul comme présenté en section [Principe général](#)
- Pour visualiser (et construire!) tous les îlots en cours, cliquer sur le bouton 'Créer et voir îlots'
 - Attention, si la modification (clic cellule) a été faite par l'utilisateur, cliquer au préalable sur le bouton 'MajGrille'.
- Pour visualiser l'ensemble des solutions de tous les îlots, cliquer sur 'Résoudre îlots'
 - Attention, les îlots doivent avoir été créés avant cette action
 - Chaque ligne représente 1 solution pour 1 îlot donné. Les lignes sont au format 'N°îlot | Nombre de drapeaux de la solution'
 - Cliquer sur une ligne pour faire apparaître la solution dans le démineur (sous forme de point d'interrogation sur les cellules)
 - Eventuellement, re cliquer sur 'Résoudre îlot' pour constater que l'algo ne trouve pas les mêmes solutions (!), notamment pour les gros îlots.

IV-A-5 - Conclusion

Le développement m'a bien pris quelques journées pleines, si on compte les premiers algorithmes combinatoires que j'ai jetés par la suite...

A peu près 5-10% du temps pour la partie technique et 90% pour la recherche du meilleur algo pour la résolution.

Sacré prise de tête pour... un programme qui ne sert à rien ;)

La manipulation technique du démineur et la résolution basique dépassées, il n'y avait finalement aucune limite à chacun pour faire mieux que son voisin.

C'est ce qui a donné tout l'intérêt de ce défi je pense. Et nous avons d'ailleurs tous des solutions assez différentes :).

[TicTacToe](#) le 28 juillet 2006

IV-B - Les sources de la solution

Téléchargez le code source du vainqueur	
Téléchargez l'exécutable du vainqueur	

V - Les solutions des challengers

Parmi les différents participants au [topic](#), voici différentes propositions. Certaines propositions ne sont pas finalisées.

V-A - La solution d'Anapurna

Téléchargez le code source d'Anapurna	
Téléchargez l'exécutable d'Anapurna	

V-B - La solution de Fabrice ROUXEL 1

Téléchargez le code source de Fabrice ROUXEL 1	
Téléchargez l'exécutable de Fabrice ROUXEL 1	

V-C - La solution de Portu

Téléchargez l'exécutable de Portu	
--	--