Jouer au démineur ! - La solution du défieur.

par Equipe DELPHI

Date de publication : 10 Août 2006

Dernière mise à jour : 10 Août 2006

Waskol nous propose de découvrir la solution qu'il a envisagé pour résoudre ce défi.

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Téléchargement de l'article au format PDF

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

I - Les sources et resources disponibles

Voici tout d'abord quelques fils de discussion qui m'ont inspirés :

- Pour simuler l'appui de touche de clavier :
  - Handle mais pas classname
  - Problème avec send message et WM_CHAR
  - CreateProcess, ShellExecute

- Pour simuler les clics de souris :
  - le bouton milieu de la souris

- Récupérer le canvas de n'importe quelle fenêtre (afin de le copier dans un bitmap puis de lire les pixels qui le composent)

Après, la FAQ regorgeait d'astuces utiles pour réaliser le programme :

- Comment exécuter une application extérieure ?
- Comment lancer et contrôler une application extérieure ?
- Comment savoir si une application est en cours d'exécution ?
- Comment fermer une application externe ?
- Comment connaître la position de la souris ?
- Comment simuler un clic de souris ?
- Comment obtenir la version de Windows ? (Pour déterminer la Version du démineur Windows 95, 98, Me, NT, 2000 ou XP)
- Comment obtenir des informations sur la langue de la session d'un utilisateur ? (Quel est le nom de l'exécutable ? Winmine.exe ou MineSweeper.exe ?)

Sinon, les fichiers d'aide de Delphi, y compris ceux de la SDK Microsoft et les moteurs de recherche sont une source d'informations extraordinaire pour réaliser ce genre de programme :

- Dans le SDK de Microsoft, on trouve de nombreuses explications sur les API Windows (en anglais)
- Sur Internet, de nombreux sites parlent du jeu lui-même et donnent des astuces pour résoudre certaines positions (que j'ai appelées shémas), d'autres proposent des astuces pour tricher, et à travers ces astuces fournissent des renseignement sur les différentes versions de démineur (Par exemple, selon la version de Windows, les scores sont situés dans un **fichier .ini** ou plutôt dans la **base de registre**; parfois le fichier éxécutable s'appelle s'appelle **winmine.exe**, parfois **minesweep.exe**; le titre de la fenêtre dans la version française est "**Démineur**", dans la version anglaise "**MineSweeper**".)

Au final, après diverses recherches, toutes les solutions au coté "technique" du problème étaient là :

- Lancer/Fermer le démineur --> lancer/Détecter/fermer une application extérieure
- Cliquer sur une mine, rentrer un nom dans la fenêtre des scores, choisir un niveau --> Simuler clics de souris et clavier.
- Déterminer si le jeu est en cours, gagné ou perdu --> Capturer l'écran, puis lire la couleur de quelques pixels bien choisis du petit smiley qui est tantôt souriant, tantôt dépité ou tout fier d'avoir gagné avec ses lunettes.
- Déterminer quelles sont les cases qui contiennent une indication (chiffre indiquant le nombre de mines), quelles sont les cases encore inconnues, etc...--> Capturer l'écran, puis lire la couleur de quelques pixels bien choisis de la case correspondante.

Pour le reste, c'est à dire la résolution elle même, c'était une question d'algorithme....

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

II - La structure globale du programme

II-A - Les procédure et fonctions utilitaires qui exploitent les API Windows

Ces fonctions sont pour la plupart regroupées dans l'unité **UnitSystem.pas**. Elle permettent entre autre :

- de fermer une application externe si on connait le Handle de sa fiche principale (**FermerApplication**). Cette fonction est utilisée dans le **OnClose** de la fiche principale.
- de simuler des clics de souris de deux façons différentes : clic réel et clic virtuel (voir les procédures **MouseClickOnScreen** et **VirtualMouseClickOnWindow**). La première procédure utilise la fonction de l'API **mouse_event**, la seconde utilise le sytème d'envoi de messages Windows classique avec **PostMessage**.
- de simuler les frappes de clavier (voir les procédures **SimulateKeyDown**, **SimulateKeyUp**, **SimulateKeystroke**, et **SendKeys**). On doit noter ici que l'on trouve sur internet une unité SendKeys.pas, beaucoup plus complexe, sans doute plus aboutie, mais que je n'ai pas réussi à utiliser. Ici, j'ai essayé de rester simple et ça a mieux marché en utilisant la fonction de l'API Windows **keybd_event** qui permet d'envoyer des évènements clavier à la fenêtre de Windows qui possède le Focus...
- ....Comment mettre le Focus sur une fenêtre ? En cliquant sur sa barre de titre (voir la procédure **ActiverFenetre**)
- d'obtenir la couleur d'un pixel d'une fenêtre particulière (voir la fonction **CouleurPixel** qui utilise l'API Windows **GetPixel**).

Pour mesurer le temps mis par mon programme pour résoudre une partie, je me suis appuyé sur la fonction API **GetTickCount** (méthodes **StartChrono** et **StopChrono** de la classe **TStatistiques**). En passant, tous les calculs des statistiques sur les pourcentages de réussite ou d'échec du Bot-démineur sont dévolues à la classe **TStatistiques**, implémentée dans la variable globale **Statistiques** de l'unité **UnitStats.pas**.

II-B - La structure des données

- Chaque case du jeu est représentée par un type record que j'ai appelé **TCase** (original non ?).
- Le terrain de jeu est un tableau dynamique à deux dimensions appelé **ZoneJeu** contenant des **TCase**. Les coordonnées d'une case sont représentées par les lettres **u** et **v**. (J'ai choisi de réserver **x** et **y** pour les coordonnées des pixels sur l'écran).
- D'autres tableaux à une dimension apparaissent ça et là afin de faciliter l'inventaire de certaines cases particulières

II-C - La boucle principale du programme

La visite commence par l'évènement OnClic du bouton Jouer ! qui est en fait le programme principal : procedure TFormMain.ButtonJouerClick(Sender: TObject);

Voici l'algorithme en épurant toutes les fioritures :

- Prise en compte des paramètres choisis par l'utilisateur
- Pour toutes les parties à Jouer : (**for i:=1 to ... do**)
  - Démarrer WinMine si il y a lieu de le faire (**DemarrerWinMine;**)
  - Initialiser les données nécessaires à la résolution du démineur (**InitZoneJeu;**)
  - Jouer tout seul comme un grand (**Jouer;**)
  - Si EtatDuJeu=Gagné ALORS Rentrer son nom (**RentreScore;**)

II-D - Démarrer le démineur

C'est la méthode privée **DemarrerWinMine** qui s'occupe de cette tâche . Pour ce faire j'ai utilisé deux fonctions de l'API Windows :

- **CreateProcess** (Démarrage d'une application)
- **FindWindow** (s'assurer de l'existence de la fenêtre d'une application en connaissant son titre, et récupération de son Handle)

II-E - La résolution d'une partie de démineur

Cette lourde tâche est dévolue à la méthode privée **Jouer** qui appelle bien d'autres méthodes.

Voici l'algorithme :

- Désactiver le Jeu Statistique (**JeuStatistique:=false;**)
- Affirmer que l'on joue le premier coup qui necessite un clic aléatoire puisque aucune case n'est connue (**PremierCoup:=true;**)
- Remplir le tableau **ZoneJeu** avec des données actualisées (**ScreenshotToZoneJeu;**)
- Affirmer que la partie à commencé (**EtatDuJeu:=edjEnCours;**)
- TANT QUE la partie n'est pas terminée (UnEtatDuJeu=edjEnCours) FAIRE
    - SI on en est au premier coup ALORS
        - Démarrer le Chronomètre
        - Jouer une case au Hasard (**JoueAuHasard;**)
        - Affirmer que l'on est plus au premier coup
    - SINON
    - SI l'utilisateur à choisi le Mode "Triche" ALORS
        - on utilise l'algorithme qui triche (**JoueTriche;**)
        - on Joue les cases "candidates", c'est à dire les cases marquées "à Jouer" (**JoueCasesAJouer;**)
    - SINON on joue sérieusement :) :
        - Remplir le tableau **ZoneJeu** avec des données actualisées (**ScreenshotToZoneJeu;**)
        - Compter le Nombre de cases déminées et le nombre de cases découvertes (**InventaireCases;**)
        - Faire un inventaire des différentes cases à Jouer à travers les trois algorithmes mis au point (**InventaireCasesAJouer;**)
        - SI la fonction précédente (**InventaireCasesAJouer**) n'a pas trouvé de coups évidents et que l'algorithme de jeu statitique n'a pas échoué, alors le **JeuStatistique** est actif et dans ce cas on joue la case la moins risquée grace à **JoueStatistiques**
        - SINON
            - SI on a trouvé des coups sûrs ou évidents (NBreAJouer>0)
            - ALORS on Joue les cases "candidates", c'est à dire les cases marquées "à Jouer" (**JoueCasesAJouer;**)
            - SINON on Joue une case au Hasard (**JoueAuHasard;**)
- FIN TANT QUE
- Remplir le tableau **ZoneJeu** avec des données actualisées (**ScreenshotToZoneJeu;**) afin de faire un bilan de ce que l'on vient de Jouer.
- Compter le Nombre de cases déminées et le nombre de cases découvertes (**InventaireCases;**)

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

- Obtenir le nouvel état du jeu en testant la tête du smiley du démineur (**fonction EtatduJeu;**)
- FIN

II-F - Pourquoi "ScreenshotToZoneJeu" (qui signifie Capture d'écran vers ZoneJeu) ? Il n'y a pourtant pas de capture d'écran...

Initialement, cette méthode effectuait une capture d'écran de la fiche du démineur (ce qui m'a permis de tester le bon fonctionnement de mes tests de pixels ainsi que la localisation correcte de mes clics de souris.

Une fois cette partie au point, j'ai éliminé la capture de la fiche du démineur et lu directement la couleur des pixels à l'écran (beaucoup plus rapide).

A ce propos, j'ai tout de même laissé les trois fonctions à titre indicatif qui m'ont permis de faire des captures d'écran, d'une fenêtre complète (barre de titre+bordure), ou de de la zone cliente d'une fenêtre (fenêtre complète - barre de titre - bordure) : **ScreenShotDesktop**, **CopyWindowRectToBitmap** et **CopyClientRectToBitmap**. Ces procédures se trouvent dans l'unité **UnitSystem.pas**.

II-G - L'inventaire des cases "à Jouer"

la méthode **InventaireCasesAJouer** s'articule autour de trois algorithmes qui s'enchaînent ou non successivement selon qu'a chacune des étapes des cases "jouables" ont été trouvées ou non :

- la recherche de cases évidentes
- la recherche de cases moins évidentes (découverte de schémas)
- la recherche de case par des méthodes statistiques

Mieux qu'un algorithme, le code qui tient en quelque ligne illustre la modularité de cette recherche de cases :

```
begin
        JeuStatistique:=False;
        NBreAJouer:=0;
        RechercheCasesEvidentes;
        if NBreAJouer=0 then RechercheSchemas;
        if NBreAJouer=0 then RechercheCSP;
end;
```

Le premier algorithme que j'ai mis au point est celui de recherche de cases évidentes, assez simple à mettre en oeuvre, c'est l'algorithme le plus rapide, mais aussi le moins "intelligent".

Le deuxième algorithme est exécuté lorsque le premier échoue (Nbre de cases à jouer=0).

Celui-ci découpe la ZoneDeJeu en zones plus petites et effectue une recherche systématique de toutes les combinaisons concernant les cases encore inconnues de cette zone en fonction des cases déjà connues. C'est l'algorithme que j'ai eu le plus de mal à mettre au point

A partir de là, mon bot à commencé à être un joueur "futé". Les participants du défi avaient du souci à se faire à ce moment là... Jusqu'a ce que TicTacToe arrive :p

L'idée du dernier algorithme ne m'est arrivée que tardivement. Après de multiples recherches sur les méthodes de résolution statistique sur internet, je suis "tombé" sur un code (pas en Delphi) parlant de "Programmation par satisfaction de Contraintes" pour résoudre le démineur, de site disant que le démineur était un problème "P-Complete" et j'en passe. J'avoue ne pas avoir tout compris tout de suite, j'en rigole encore...

Toujours est-il que le dit code, utilisant un langage de programmation "ensembliste" n'était pas du tout, mais alors vraiment pas transposable en Delphi. Et puis je me suis documenté et compris que l'on avait tout bêtement affaire à la résolution d'un système à N inconnues (N cases inconnues) et M équations (M1 cases indiquant un nombre de mines, 1 équation contabilisant le nombre de mines restant à découvrir) avec N>M. Du coup on se retrouve avec plusieurs solutions, et il reste à dénombrer l'état des inconnues pour chacune de ces solution ("mine" ou "pas mine" ?). Lorsque l'on divise par le nombre total de solutions valides (toutes les inconnues valent 1 ou 0 et rien d'autre), on se retrouve avec la probabilité d'avoir une mine (J'aime quand les maths sont présentés simplement, on comprend mieux)

Après avoir résolu plusieurs détails techniques question "dénombrement de solutions" (Dites, on fait comment avec les factorielles quand les nombres sont trop grand pour Delphi ? Comment éviter les erreurs d'arrondi lors d'un pivot de Gauss ? etc...), j'ai commencé à coder... J'y ai cru jusqu'à ce que je fasse un malaise dû à un trop grand surmenage (trop d'activités à la fois, trop de boulot, trop de tout...).

Heureusement, j'ai pu le finaliser mais par contre, santé oblige, je ne me suis pas attardé sur son optimisation.... ça marche, c'est le principal.

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

III - Les algorithmes de résolution en détail

III-A - recheche de cases évidentes

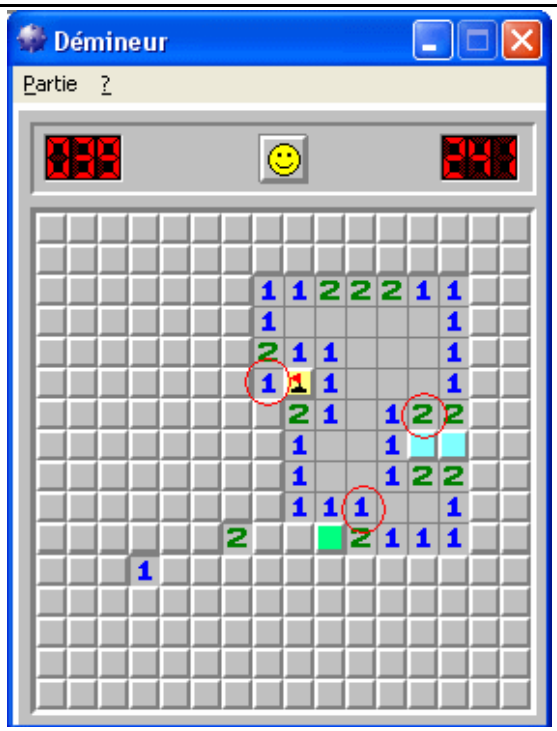
La recherche de cases évidentes est triviale.

Il consiste pour chaque case du terrain qui affiche un chiffre qui représente le nombre de mines qui se trouvent autour de vérifier si ce chiffre (appelons le **Nmines**) est égal au nombre de cases déminées qui se trouvent autour (**Ndeminees**) et au nombre de cases inconnues qui se trouvent autour (**Ninconnues**).

- SI on a **Nmines=NDeminees+NInconnues**, ALORS les cases inconnues sont toutes des mines et sont marquées comme cases évidentes.
- SI on a **NMines=NDeminees**, ALORS aucune des cases inconnues qui se trouvent autour ne sont des mines. C'est alors la case que l'on est en train de tester qui est marquée comme étant une case évidente.

Selon que la case evidente est une mine ou non, l'algorithme indique quelle méthode de clic sera envisagée (Clic-droit si c'est une case à déminer, Clic simultané Gauche+Droite pour faire de la place autour de la case).

Prenons cet exemple :



1er cas : nous avons un "1" et une seule case mitoyenne (case verte). Le résultat est évident, la case verte est marquée comme "case à jouer" avec un clic droit.

2ème cas : nous avons un "2" et deux cases mitoyennes (case verte). Le résultat est évident, les deux cases bleu ciel sont marquées comme "case à jouer" avec un clic droit.

3ème cas : nous avons un "1" et une case déminée autour (la jaune). Cette case (entourée en rouge) est marquée comme case évidente avec le Clic combiné des deux boutons de la souris

C'est, somme toute, la méthode de jeu que la plupart d'entre nous adopte en jouant au démineur sans trop réfléchir.

III-B - recherche de schémas

La recherche de schémas est beaucoup plus complexe !!!

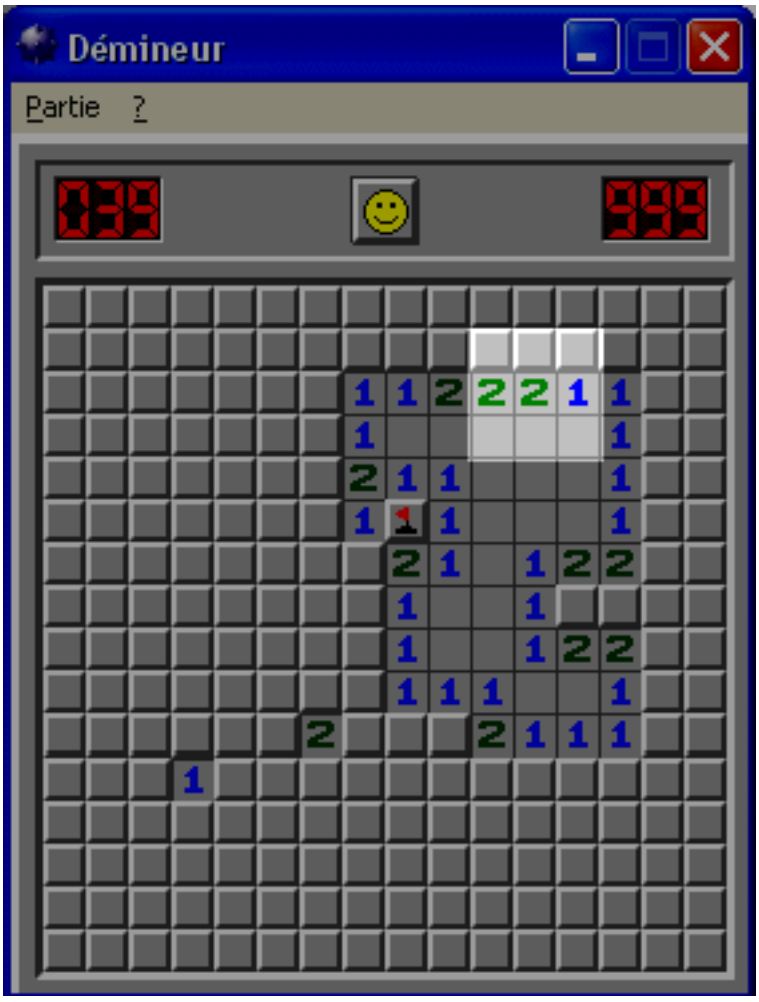
L'algorithme commence par construire une grille de 3 x3 cases, cette grille est déplacée "au -dessus" du terrain de Jeu pour le scanner entièrement et effectue une recopie des cases qui se trouvent "en dessous". A chaque fois que la grille de recherche est déplacée, toutes les solutions "locales" sont envisagées. Dès que le scan trouve une case AJouer, il rend la main au programme, sinon il continue.

Si le scan de 3x3 à échoué, le programme effectue alors un scan avec une grille de 4x4, puis 5x5 et enfin 6x6.

Si à ce stade, la recherche est encore un échec, cet algorithme rend la main et c'est l'algorithme suivant (recherche CSP) qui prend la relève.

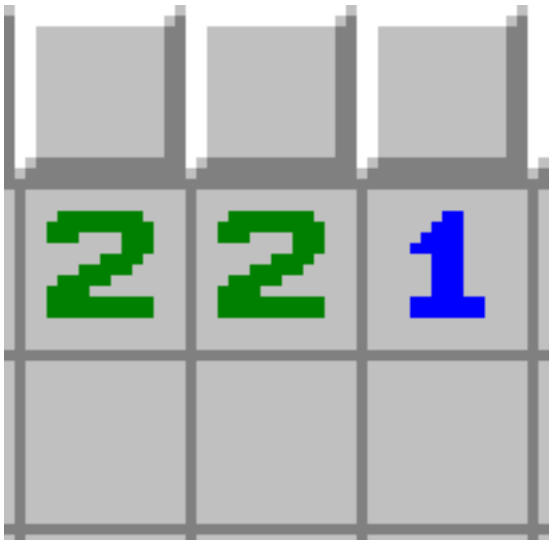
Pour ce qui est de la recherche de schéma dans la grille du scan, voici comment l'algorithme procède, du moins dans son principe :

Reprenons notre exemple et imaginons que notre programme est en train de passer un scan avec une grille de 3x3 :



Voici donc notre grille de recherche :

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.



- Nous avons trois cases inconnues, qui ont pour voisine des cases numérotées. Nous avons donc trois cases qui seront chacunes représentée par les trois premiers bits d'un entier. Nous allons alors faire prendre a notre entier, toutes les valeurs de 0 à 7, visuellement, cela donne :

  000

  001

  010

  011

  100

  101

  110

  111

  Un '1' représente une case avec une mine, un '0' une case sans mine. Chacune de ces combinaisons sera appelé un schéma de test.

- Ensuite nous avons six cases "découvertes", dont trois cases "chiffrées", nous devons déterminer pour chacune d'entre elles si ce sont des case à tester ou non.

  Une case sera dite "à tester" si elle répond à ces critères :

  - C'est une case "chiffrée" : c'est à dire une case découverte et dont le nombre de mines autour est supérieur à 0
  - Si le nombre de cases inconnues qui se trouvent autour, dans notre grille de recherche est supérieur à

0.

Celà tombe bien, nos trois cases chiffrées répondent à ces critères.

Maintenant, que nos cases à tester sont identifiées, nous devons ré-examiner le chiffre qui s'y trouve.

En fait, pour chacune de nos trois cases nous allons déterminer le nombre de mines potentiellement présentes, c'est-à-dire le nombre de cases encore inconnues sur la grille du terrain autour de notre grille de recherche :

NbreMinesPotentiellesAutour:=NbreInconnuesAutour-NbreInconnuesAutourDansSchema; ->NbreInconnuesAutour inclut les cases du terrain

if NbreMinesAutour < NbreMinesPotentiellesAutour then NbreMinesPotentiellesAutour:=NbreMinesAutour

Celà nous donne :

Pour la case...	Nbre de Mines Potentielles Autour
...de gauche	1
...du milieu	0
...de droite	1

Ensuite, nous devons tester la validité de chacun des schemas (000, 001, ..., 111), voici l'algorithme pour tester si un schéma est valide :

```
DEBUT
        REMPLACER CHAQUE "Case inconnue" DE LA "grille de test" PAR
        - "une case déminée" (une mine) SI on a un '1' DANS LE "schema de test"
        - "une case découverte" (une case vide, sans mine) SI on a un '0' DANS LE "schéma de test"

        AFFIRMER "Le schéma est Valide"
        POUR CHAQUE "Case a tester"
                "Nbre de Mines Supposees Autour"=0
                POUR CHAQUE "Case autour de la case a tester" DANS "la Grille de Test"
                        Si "Case Autour"="Case deminee" DANS LA "grille de test" MAIS PAS DANS "le
terrain du démineur" ALORS INCREMENTER "Nbre de Mines Supposees Autour"

        //Premier test
        SI "Nbre de Mines Supposees Autour">"Nbre de mines Autour" (terrain du démineur) ALORS "Le
schéma n'est pas valide"

        //2nd test
        SI (NbreMinesPotentiellesAutour=0) ALORS
        SI (NBredeMinesSupposeesAutour+NbreDemineeAutour)<>NbreMinesAutour ALORS "Le schéma n'est
pas valide"

        //3ème test
        SI (NbreMinesPotentiellesAutour>0) ALORS
        SI(NBredeMinesSupposeesAutour+NbreMinesPotentiellesAutour+NbreDemineeAutour)<NbreMinesAutour
ALORS "Le schéma n'est pas valide"

FIN
```

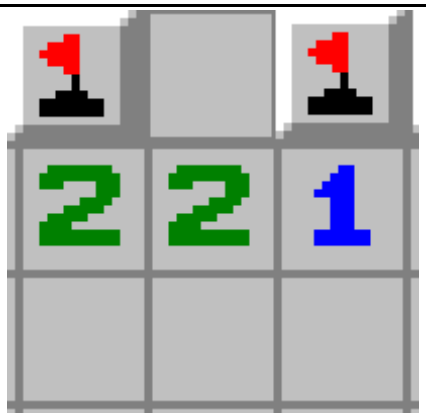
A partir de la nous savons si un schéma de jeu est plausible ou non, cela nous donne, pour chaque schéma de test concernant notre exemple :

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

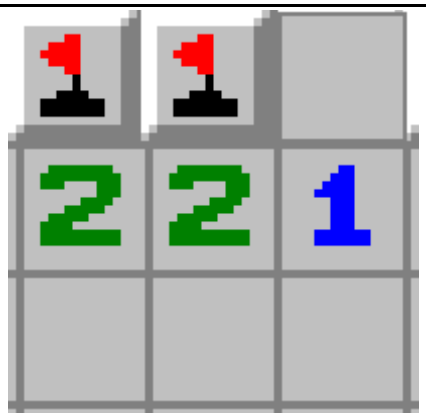
Schéma de test	Case à tester	Nbre de Mines Supposees autour	Nbre Mines Potentielles autour	Nbre de Mines autour	Nbre de cases déminees autour	Schéma valide pour la case à tester	Le Schéma passe le test de validité (donc schéma plausible)
0 0 0	Gauche	0	1	2	0	Non (au Test 3)	NON
	Milieu	0	0	2	0	Non (au Test 2)	
	Droite	0	1	1	0	Oui	
0 0 1	Gauche	0	1	2	0	Non (au Test 3)	NON
	Milieu	1	0	2	0	Non (au Test 2)	
	Droite	1	1	1	0	Oui	
0 1 0	Gauche	1	1	2	0	Oui	NON
	Milieu	1	0	2	0	Non (au Test 2)	
	Droite	1	1	1	0	Oui	
0 1 1	Gauche	1	1	2	0	Oui	NON
	Milieu	2	0	2	0	Oui	
	Droite	2	1	1	0	Non (au Test 1)	
1 0 0	Gauche	1	1	2	0	Oui	NON
	Milieu	1	0	2	0	Non (au Test 2)	
	Droite	0	1	1	0	Oui	
1 0 1	Gauche	1	1	2	0	Oui	OUI
	Milieu	2	0	2	0	Oui	
	Droite	1	1	1	0	Oui	
1 1 0	Gauche	2	1	2	0	Oui	OUI
	Milieu	2	0	2	0	Oui	
	Droite	1	1	1	0	Oui	
1 1 1	Gauche	2	1	2	0	Oui	NON
	Milieu	3	0	2	0	Non (au Test 1)	
	Droite	2	1	1	0	Non (au Test 1)	

Nous avons donc ici, deux schémas valides qui sont 101 et 110.

C'est-à-dire que, à coup sur, seules deux dispositions éventuelles dans notre grille de démineur sont plausibles :



Le Schéma 101



Le Schéma 110

- Dans les deux cas, seule la case de gauche est identique et représente une case déminée, cette case sera donc à coup sur une mine.
- Si dans plusieurs cas plausibles une case aurait toujours représenté une case "vide", cette case aurait été à coup sur une case sans mine.

Pour identifier ces cases "invariantes" nous effectuons une succession d'opérations bit à bit sur un masque égal à 111111111111 avec nos valeurs de schémas trouvés:

- Pour identifier les cases minées, nous effectuons une succession de ET logiques :

  Résultat=111111111 and 101 and 110, d'où Résultat=100.

  Ceci signifie clairement que notre case de gauche est une case minée à coup sur.

- Pour identifier les cases non-minées, nous effectuons une succession de ET logiques avec la négation binaire de nos valeurs

  Résultat=111111111 and (not 101) and (not 110), d'où Résultat=000

  Ceci signifie clairement qu'aucune de nos case n'est à coup sur une case "vide" (sinon, on aurait eu au moins un 1).

Du coup, nous obtenons à l'issue de notre algorithme un certain nombre de "Cases à Jouer" , soit avec un clic gauche, soit avec un clic droit (selon qu'il s'y trouve une mine ou non)

Bref, cela fut un vrai casse-tête à trouver l'algorithme, puis à le coder !!!

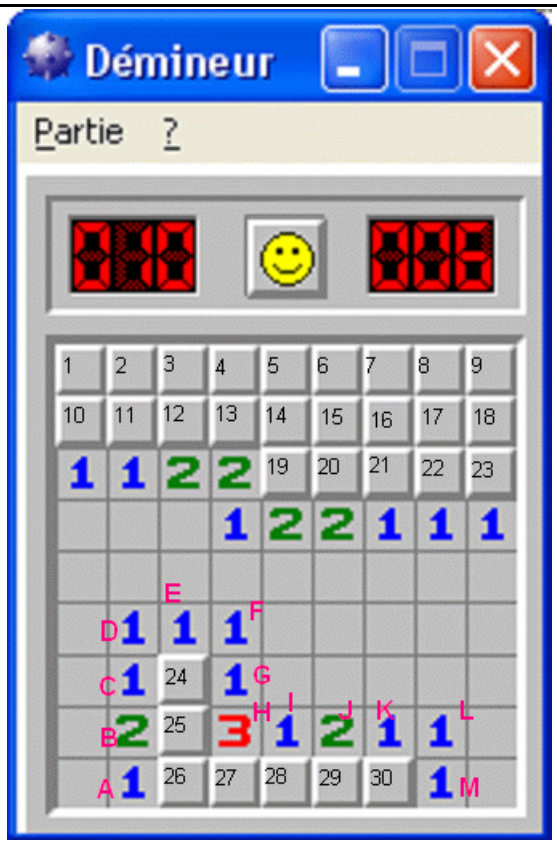
III-C - Recheche CSP (Constraint Satisfaction Problem)

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Sans vraiment rentrer dans les détails de l'algorithme qui est suffisamment commenté dans la procédure **TFormMain.RechercheCSP;**, en voici son principe de fonctionnement.

Ce dernier repose un principe qui consiste à représenter chaque "case inconnue" du plateau de jeu par une variable Xi.

- Chacune de ces variables peut prendre soit la valeur 1, soit la valeur 0 : c'est ce que l'on appelle la contrainte du problème.
- Ensuite, nous obtenons un certain nombre d'équations, chacunes construite à partir des cases "Chiffrées". Pour ce faire, prenons un nouvel exemple et construisons quelques-unes de nos équations à partir des cases chiffrées marquées en rose :



Equation A : X25+X26=1
Equation B : X24+X25+X26=2
Equation C : X24+X25=1
Equation D : X24=1
Equation E : X24=1
Equation F : X24=1
Equation G : X24+X25=1
Equation H : X24+X25+X26+X27+X28=3
Equation I : X27+X28+X29=1
Equation J : X28+X29+X30=2
Equation K : X29+X30=1
Equation L : X30=1
Equation M : X30=1

Pour les cases chiffrées du "haut" la même mise en équations est également effectuée, ces équations impliqueront donc les variables de X10 à X14, et de X19 à X23.

Nous remarquons alors, que certaines inconnues ne sont pas encore mises en équation : de X1 à X7 et de X15 à X18.

Nous allons tout d'abord placer ces dernières inconnues, dans une seule variable qui les regroupe toutes : Y=X1+...+x7+X15+...+X18. Comme de toutes façons ces variables ne peuvent être évaluées directement (elles dépendent de toutes les autres), je les ai qualifié de "non-devinables".

Puis nous allons obtenir notre toute dernière équation qui correspond à la somme de TOUS nos X, qui en toute logique, est égal au nombre de mines restant à découvrir sur le plateau : Y+X10+..+X14+X19+..+X30=10

Cela nous donne un système de 23 équations à 30 inconnues, avec des équations dupliquées.

Ce genre de système, mis sous forme de matrice (ici un tableau d'entiers de 19 colonnes x 30 lignes) est résolu en utilisant un pivot de Gauss maison qui préserve les valeurs entières des coefficients : cela évite les erreurs d'arrondi que l'on rencontre dans l'algorithme habituel que l'on trouve sur internet

A l'issue de ce pivot, certaines inconnues ne le seront plus, comme les inconnues allant de X19 à X30

D'autres seront encore sous forme d'équations. Parmi ces dernières, on pourra identifier un bon nombre d'entre elles comme étant "dépendantes" de quelques unes que j'ai nommé dans mon code "dépendances". Les **"dépendances" se retrouvent dans plusieurs équations du système**, alors que les **"dépendantes" ne se retrouvent que dans un seule équation à la fois**.

En testant toutes les valeurs possibles des dépendances (0 ou 1), on obtiendra des solutions plausibles (comme dans la recherche de schémas) où les variables dépendantes prennent éfectivement les valeurs 0 ou 1. Celà se traduira par le fait que :

- Si le résultat de l'équation est positif, la somme des coefficients positifs de l'équation sera supérieure ou égale à ce résultat.
- Si le résultat est négatif, la somme des coefficients négatifs sera inférieure ou égale à ce résultat.

Sinon, on obtiendrait des solutions aberrantes où les variables dépendantes prennent des valeurs qu'elles ne devraient pas avoir (-10, -1, 2, 3, 8 par exemple).

Si toutes les équations sont valides, le système est cohérent, et on a alors un schéma (ou solution) plausible.

Lorsque l'on sera confronté à une solution plausible, il s'agira alors de dénombrer le nombre de cas de figure ou le schéma apparaît dans l'ensemble de toutes les combinaisons possibles. C'est là où rentre en jeu le nombre de cases "non devinables" que j'ai cité précédemment.

- Admettons que l'on obtienne Y=3, cela signifie que 3 mines se trouvent parmi 10 cases. Le schéma existera donc N0 fois pour ces 10 cases, N0 étant calculé par la fonction Combinatoire C(3,10) qui donne le nombre de façons de placer 3 mines parmis 10 cases :
- Comme je l'ai dit, les variables dépendantes ne seront pas toutes déterminées de façon unique, par exemple, suite au pivot, nous pouvons très bien nous retrouver avec une équation du style : X27+X28+X29=2. Le nombre de solutions à cette équation sera N1=C(2,3) (une mine parmis 3).
- Parfois on aura qu'une case et 0 mines : N2=C(0,1)
- etc...

Le nombre de solutions que représente effectivement notre schéma sera égal à N=N0xN1xN2x...xNm

Comme avec ce genre de fonction qui normalement fait intervenir les factorielles, on se retrouve confronté à des nombres gigantesques, il a fallu "bidouiller" les mathématique officielles pour que l'algorithme de calcul sache simplifier les divisions "à la main" et utilise les propiétés du triangle de Pascal pour éviter les multiplications à n'en plus finir et les dépassements de capacité qui en résultent.

Revenons a nos cases : dans l'équation X27+X28+X29=2, chacune des cases aura 2 chances sur 3 de prendre la valeur 1, donc parmi N1=C(2,3) combinaisons différentes, la valeur prendra N1*2/3 fois la valeur 1.

- A chaque solution plausible, on totalisera le résultat **N** dans une variable (**NTot** dans le programme) qui nous permettra de calculer la probabilité de présence d'une mine pour une case donnée : P=(N1+N1'+N1''+...)/NTot

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Et voilà, nous obtenons ainsi des statistiques exactes pour toutes les cases de notre plateau.

Fin de l'algorithme.

Il faut préciser que cet algorithme abandonne sa recherche de lui-même s'il estime dépasser ses capacités de résolution (matrice trop grande, calcul statistique impossible, etc...) et dans le cas d'un échec passe la main à l'algorithme le plus simple du lot : celui qui joue, dépité, une case au hasard....

IV - Conclusion

On s'aperçoit vite ici du nombre considérable de solutions techniques qui permettaient d'aboutir à un automate démineur. A travers les trois algorithmes exposés ici, trois approches différentes au problème vous sont exposées parmi les très autres nombreux algorithmes existant ou pouvant sortir de votre imagination.

Les challengers nous l'ont prouvé en réalisant leur propres méthodes de résolution, toutes différentes, toutes reflétant, peut-être, leur différentes personalités.

Autant il est clair que la partie système demandait quelques connaissances des API Windows, toutes disponibles dans la FAQ de www.developpez.com, autant la partie "Intelligence Artificielle" faisait plus appel à l'imagination de chacun.

Copyright © 2006 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et 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 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

V - Sources

Téléchargez le code source du défieur	Télécharger	(miroir)
Téléchargez l'exécutable du défieur	Télécharger	(miroir)