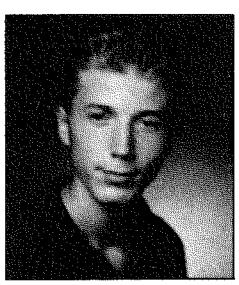
les arêtes 2 et 6 du sommet de coordonnées i,j sont occupées, Q %(i,j) contiendra 34 qui s'écrit 100010 en binaire. Le deuxième et le sixième bits en partant de la droite valent 1, indiquant que les arêtes 2 et 6 sont indisponibles. En arrivant à chaque intersection, le programme décode cette information, effectue une rotation pour se ramener à la configuration de l'une des règles et en déduit la direction à prendre. Il met alors à jour le tableau Q %() et trace l'arête parcourue. Le cycle peut alors recommencer à partir de la nouvelle intersection atteinte.

On peut imaginer très simplement des extensions au problème du ver de Paterson. Tout d'abord, rien n'interdit de multiplier les vers sur le terrain et d'organiser un combat entre deux ou plusieurs vers, obéissant au même jeu de règles ou à des jeux différents. De même, on peut changer la portée des règles et permettre au ver de voir plus loin que l'intersection où il se trouve pour choisir sa direction. Enfin, on peut imaginer de changer le terrain pour enrichir encore les possibilités, en adoptant des réseaux plus complexes et pas nécessairement uniformes, ou en passant à trois dimensions. On peut imaginer un ver se déplaçant dans un réseau cubique ou tétraédrique. Les variations possibles sont pratiquement infinies. Les lecteurs qui exploreraient ces questions sont invités à nous transmettre leurs trouvailles.

Frédéric NEUVILLE

LABYRINTHES

Le labyrinthe idéal se doit de posséder une entrée, une sortie, un seul itinéraire possible et, pour compliquer un peu la recherche, de nombreuses fausses pistes... De plus, il doit être impossible de tourner en rond, et tous les chemins doivent être accessibles. Si vous avez essayé de réaliser un programme en Basic capable de créer de tels labyrinthes, vous avez dû être terriblement déçu par la lenteur du dessin, et la difficulté à réaliser un algorithme tout à fait clair et efficace. Le programme de notre gagnant du mois, Edouard Thiel, étudiant à la faculté de mathématiques de Strasbourg, est en Turbo Pascal pour IBM PC et compatibles (avec carte graphique CGA, EGA ou Hercules). Il est d'une rapidité exceptionnelle : un labyrinthe de 38 cases sur 11 (il occupe tout l'écran) est dessiné en quelques secondes, et l'algorithme de recherche de la sortie est également très performant. Le programme est parfaitement structuré, ce qui facilitera son adaptation dans d'autres langages et sur d'autres machines.



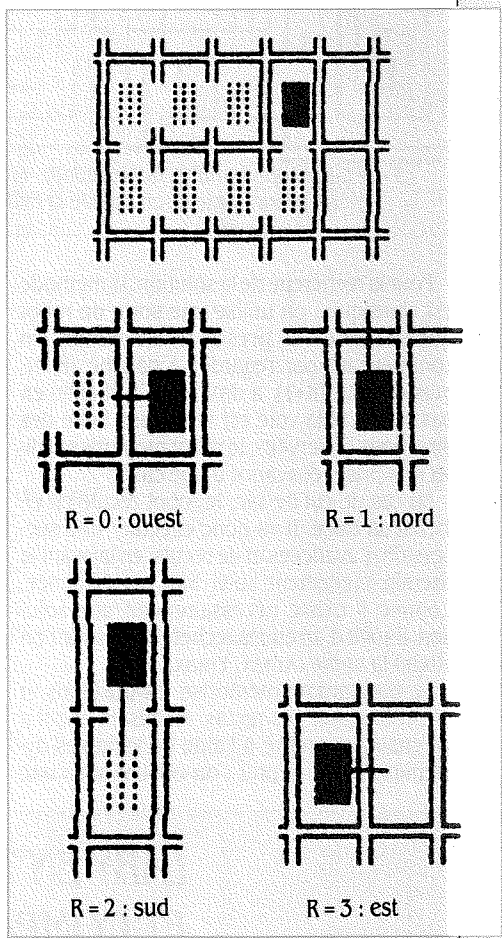
CE GÉNÉRATEUR de labyrinthes comprend deux parties : la création du labyrinthe proprement dite, et la recherche de la solution. Le labyrinthe possède une entrée, située dans le coin supérieur gauche, et une sortie, dans le coin inférieur droit. Les deux constantes Xmax et Ymax définissent la taille du labyrinthe (voir en début de programme). Au départ, on utilise une grille : une sorte d'enzyme glouton est placé sur une case du bord, avec pour mission de dévorer les murs jusqu'à la sortie, en laissant une trace sur son passage. Mais il devra accepter certaines contraintes :

- ne pas sortir du labyrinthe (autrement dit, ne pas dévorer un mur de clôture) ;
- ne pas manger tous les murs ;
- s'arrêter suffisamment tôt pour que le labyrinthe ait un seul chemin menant de l'entrée à la sortie.

Les traces laissées par l'enzyme correspondent donc à la création du couloir du labyrinthe. En pratique, il s'agit, à partir de l'entrée, puis de n'importe quelle autre case, de tirer au hasard l'une des cases contiguës (0 :Ouest ; 1 : nord ; 2 :sud ; 3 :est) déterminées dans le programme par la procédure Direction. La case ne sera accessible que si l'on respecte certaines règles. Pour mieux illustrer notre propos, prenons l'exemple d'un petit labyrinthe de 2 x 5 cases en cours de création (figure ci-contre). Le rectangle noir représente la position actuelle de notre enzyme glouton, les rectangles grisés montrant le trajet déjà effectué.

A partir de cette position, on peut définir les quatre directions : ouest, nord, sud et est. Dans le premier cas, l'enzyme va percer un mur, et prendre un chemin existant, ce qui crée une boucle. C'est interdit. Le contrôle est effectué par la procédure Possible-avancer. S'il se dirige vers le nord, l'enzyme va sortir du labyrinthe ce qui est interdit par la fonction Distance (qui renvoie 0). Vers le sud, l'enzyme ne rencontre pas de mur, mais revient sur ses pas, ce qui est autorisé. Enfin, côté est, la case n'a jamais été visitée : on peut donc agrandir le chemin en choisissant cette case et en détruisant le mur.

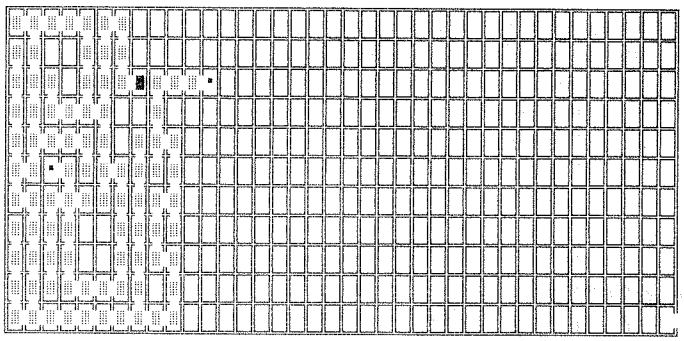
La portion de couloir ainsi créée sera quittée au bout d'un certain temps (nombre moyen de tirages), une trace particulière (un petit carré noir) marquant cette interruption. On recherche alors la première case non visitée sur la première rangée de cases. Si toutes les cases ont été visitées, on passe à la 2e rangée, et ainsi de suite. La première case encore vierge est nécessairement contiguë à une autre case déjà visitée. On ouvre donc le mur entre les deux, et on reprend la procédure précédente, pour agrandir le couloir. La méthode retenue permet ainsi d'agrandir le couloir, sans jamais réaliser de boucle : il ne peut donc y avoir qu'un seul couloir dans tout le labyrinthe. Au bout d'un certain temps (nombre moyen de tirages, comme précédemment), on quitte à nouveau le couloir avec une marque , et on recommence la recherche de la première case non visitée.



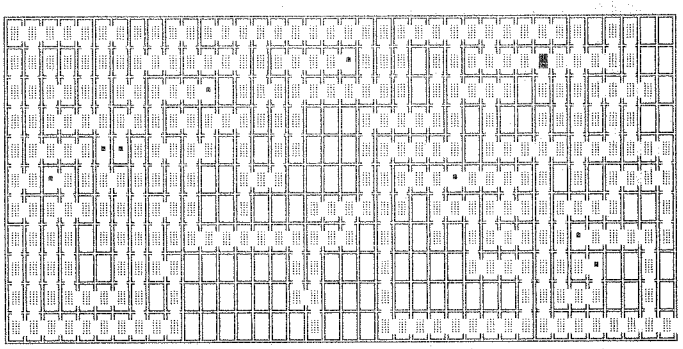
Les chemins autorisés pour créer un couloir obéissent à des règles précises. Ici, les directions nord et ouest sont interdites.

Le processus s'arrête lorsque la recherche de la première case vide nous conduit à tester la case de sortie : cela signifie que toutes les pièces ont été visitées, et donc que le labyrinthe est terminé. Pour fignoler la présentation, on « rabote » les morceaux de mur qui dépassent, avec la procédure Remplissage.

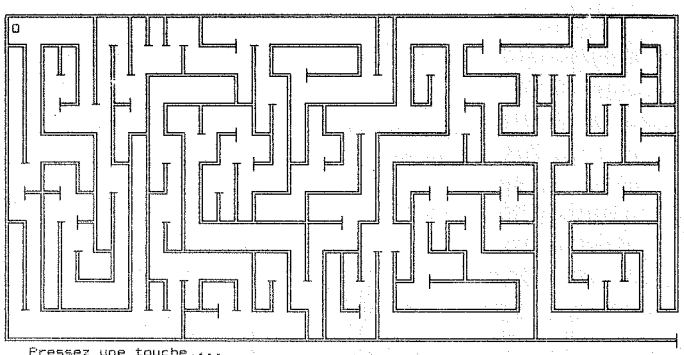
LES QUATRE ÉTAPES DE LA CRÉATION D'UN LABYRINTHE



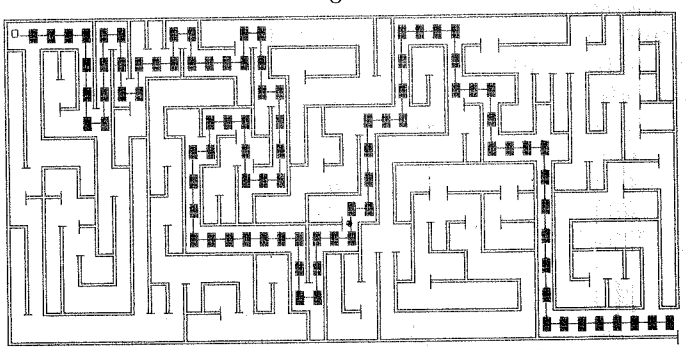
1 - A partir de la première case, l'enzyme se fraie une première portion de couloir.



2 - Petit à petit, la totalité de la surface de la grille est visitée.



3 - Le labyrinthe terminé, la procédure « remplissage » du programme se charge de la finition.



4 - L'affichage de la solution est obtenu presque instantanément.

Pour la recherche de la solution, la méthode est classique : on utilise une sorte de robot qui va tester la présence d'un mur, en commençant par regarder à gauche (R=0), puis en haut (R=1), à droite (R=2) et enfin en bas (R=3). Si la voie est libre dans l'une des directions observées, le robot mémorise la direction prise, et avance d'une case.

En cas de cul-de-sac, le robot recule en effaçant sa trace. Il va donc effectuer une succession d'avancées et de reculs, en laissant le chemin réellement suivi bien en évidence. Comme il existe nécessairement une solution, il suffit d'arrêter la recherche lorsque l'on atteint la sortie (Xmax, Ymax).

Le programme mémorise les positions et directions pour les reculs, en faisant appel à un chaînage arrière à l'aide de variables dynamiques ptr1 et ptr2 : on obtient ainsi une vitesse nettement supérieure à ce qu'il est possible de réaliser avec une pile, plus classique. Il fonctionne par adressage direct de l'écran. Il sera donc prudent, lors des premiers essais, de le sauvegarder avant toute mise en route et de prévoir au début une directive de compilation {$U+}, qui ralentira l'exécution, mais permettra une interruption par Ctrl et Break (sortie en cas de plantage).

Modifications possibles

On pourra mieux observer la création d'un labyrinthe en ralentissant le programme, en ajoutant par exemple une ligne Delay (500) dans le sous-programme « Fabrication du labyrinthe » (entre D :=D - 1 et End). On pourra également ralentir la recherche de la solution, en ajoutant de la même manière Delay (500) dans le sous-programme « Déplacement » (juste après le REPEAT). Les commentaires inscrits dans le corps du programme permettront de suivre très facilement les diverses étapes. On notera après l'instruction Randomize la définition de Moy :=(xmax+ymax)*2. Cette valeur a été choisie par tâtonnement, en fonction de la taille du labyrinthe. C'est elle qui permet d'abandonner une portion de couloir donnée, au bout d'un certain nombre d'essais. Il est évidemment possible de la modifier, notamment si l'on envisage d'autres tailles de labyrinthes.

Reste à envisager des extensions et des modifications possibles du programme, pour la production d'autres types de labyrinthes et, pourquoi pas, une ouverture vers des labyrinthes en trois dimensions.

Edouard THIEL

LABY EN TURBO PASCAL. POUR IBM PC ET COMPATIBLES. ADAPTATION ASSEZ DIFFICILE

```
{------------------------------LABY-32.PAS-------------------------------------}
{------------------------------THIEL Edouard-----------------------------------}
{                                                                              }
{ Cette version de LABY contient:                                              }
{                                                                              }
{       - Création d'un labyrinthe 38*11                          27/11/1987 }
{       - Recherche de la solution                                27/01/1988 }
{       - Optimisation de l'algorythme:                           14/02/1988 }
{         génération du laby => 9"50 sur XT , 2"30 sur AT                      }

CONST xmax=38;
      ymax=11;
      ad=$B800;          { $B800 pour cartes CGA/EGA, $B000 pour carte Hercules }

TYPE Pointeur_vers_piece = ^piece;
```

●●●/●●●

```
                    Piece = RECORD
                              Precedent : Pointeur_vers_piece;
                                  x,y,r : byte;
                              END;

VAR debut_de_liste,ptr1,ptr2:Pointeur_vers_piece;
      r,d,i,x,xp,y,yp,moy:byte;
                    dx,dy:integer;
                      rep:char;
{-------------------------------------------------------------------------------}
{                Procedures pour générer le labyrinthe                          }
{-------------------------------------------------------------------------------}

PROCEDURE Direction(dr:byte;var sx,sy:integer);                  { Fonction       }
          Begin                                                  { de déplacement }
          sx:=((2*dr+1) div 3)-1;
          sy:=((2*dr+1) mod 3)-1;
          End;

FUNCTION Trace(mr,mx,my:byte):boolean;    { Y a-t-il une trace de l'autre côté }
         Var vx,vy:integer;               { du mur dans la direction mr ?      }
         Begin                                                 { Coord laby }
         Direction(mr,vx,vy);
         if Mem[ad:320*(my+vy)+4*(mx+vx)]=32 then Trace:=false else Trace:=true;
         End;

PROCEDURE Enlever_mur(our,ox,oy:byte);       { Enlève le mur de la pièce ox,oy }
          Var odx,ody:integer;               { dans la direction our           }
          Begin                                                  { Coord laby }
          Direction(our,odx,ody);
          Mem[ad:160*(2*oy+ody)+2*(2*ox+odx)]:=32;
          End;

FUNCTION Distance(drr,dix,diy:byte):byte;  { Donne la distance en nombre      }
         Begin                             { de pièces entre la piece dix,diy }
         Case drr of                       { et le bord du labyrinthe         }
           0: Distance:=dix-1;             { dans la direction drr            }
           1: Distance:=diy-1;                                   { Coord laby }
           2: Distance:=ymax-diy;
           3: Distance:=xmax-dix;
           end;
         End;

FUNCTION Possible_avancer(pr,a,b:byte):boolean;   { Est-il possible d'avancer }
         Var vx,vy:integer;                       { dans la direction pr      }
         Begin                                    { sans créer de boucle      }
         Direction(pr,vx,vy);                                  { Coord laby }
         if (Trace(pr,a,b)) and (Mem[ad:160*(b*2+vy)+2*(a*2+vx)]<>32)
         then Possible_avancer:=false
         else Possible_avancer:=true;
         End;

PROCEDURE Remplissage;                          { Remplis les coins suivant     }
          Var d,i,exp2,n,x,y,rx,ry:byte;        { la position des murs attenant }
                              vx,vy:integer;
          Begin
          for x:=0 to xmax do begin
              for y:=0 to ymax do begin
              rx:=2*x+1;
              ry:=2*y+1;
              n:=0;exp2:=1;
              for i:=0 to 3 do begin
                  Direction(i,vx,vy);
               if Mem[ad:160*(ry+vy)+2*(rx+vx)]<>32 then n:=n+exp2;
                  exp2:=exp2*2;
                  end;
              Case n of
                    0: d:= 32;    1: d:= 181;    2: d:= 208;   3: d:= 188;
                    4: d:= 210;   5: d:= 187;    6: d:= 186;   7: d:= 185;
                    8: d:= 198;   9: d:= 205;   10: d:= 200;  11: d:= 202;
                   12: d:= 201;  13: d:= 203;   14: d:= 204;  15: d:= 206;
                    end;
          Mem[ad:160*ry+2*rx]:=d;
              end;
           end;
          End;
{-------------------------------------------------------------------------------}
{                  Procedure pour rechercher la solution                        }
{-------------------------------------------------------------------------------}

FUNCTION Sens_interdit(r,x,y:byte):boolean;       { Permet de savoir s'il est }
         Var ax,ay:integer;                       { possible d'avancer dans   }
         Begin                                    { la direction r            }
         Direction(r,ax,ay);
         if ( Mem[ad:160*(2*y+ay)+2*(2*x+ax)]<>32)
         or ( Mem[ad:320*(y+ay)+4*(x+ax)]<>32)
         then Sens_interdit:=true
         else Sens_interdit:=false;
         End;

{-------------------------------------------------------------------------------}
{                      Génération du labyrinthe                                 }
{-----------------------  -----------------------------------------------------}

BEGIN

 Clrscr;{---------------Préparation de l'écran-------------------------------}

 for y:=0 to ymax do for x:=1 to xmax do Mem[ad:320*y+4*x+160]:=205; { dessin }
 for y:=1 to ymax do for x:=0 to xmax do Mem[ad:320*y+4*x+2]:=186;   {des murs}

 Remplissage;                                                  {Dessin des coins}
```

```
 Mem[ad:324]:=79;                                                    {entrée}
 Mem[ad:320*ymax+4*xmax+2]:=32;                                      {Sortie}
 for dx:=0 to 1999 do Mem[ad:2*dx+1]:=7;                          {Encre normale}

 Randomize;
 Moy:=(xmax+ymax) *2;
{--------------------Fabrication du Labyrinthe-------------------------------}

 For yp:=1 to ymax do begin                    { Balayage de toutes les pièces }
     for xp:=1 to xmax do begin
                                          (** Recherche du début d'un couloir **)
 if Mem[ad:320*yp+4*xp]=32 then

    BEGIN
    r:=random(4);                                          { Choix direction }
    while (not Trace(r,xp,yp)) or (Distance(r,xp,yp)=0)
       do r:=(r+1) mod 4;

    Enlever_mur(r,xp,yp);                  { Ouverture sur un autre couloir }
    x:=xp; y:=yp;
                                                 (** Génération du couloir **)
    For i:=1 to moy do begin
        r:=random(4);                                      { Choix direction }
        while Distance(r,x,y)=0 do r:=(r+1) mod 4;         { éviter le bord  }

        Direction(r,dx,dy);
        D:=random(Distance(r,x,y)) mod 3 +1;     { Choix longueur des petites }
                                                                { avancées }
        while Possible_avancer(r,x,y) and (D>0) do
              begin                                  { On avance en détruisant }
              Enlever_mur(r,x,y);                    { des murs et en laissant }
              Mem[ad:160*2*y+2*2*x]:=176;            { des traces derrière soi }
              x:=x+dx;
              y:=y+dy;
              Mem[ad:160*2*y+2*2*x]:=219;
              D:=D-1;
              end;
    end;
    Mem[ad:160*2*y+2*2*x]:=254;                        { On quitte ce couloir }
    END;                                  { Fin de la génération du couloir }
        End;                                    { Fin couloir, Fin balayage }
  End;

{-------------------------------Finition-------------------------------------}

 Remplissage;                                                  { Dessin des coins }
                                                              { Efface les traces }
 for y:=1 to ymax do for x:=1 to xmax do Mem[ad:320*y+4*x]:=32;
 Mem[ad:324]:=79;

gotoxy(5,25);write('Pressez une touche ...');read(kbd,rep);gotoxy(1,25);clreol;
{------------------------------------------------------------------------------}
{                        Recherche de la solution                              }
{------------------------------------------------------------------------------}

 NEW ( Debut_de_liste );                       { Création de la première pièce }

 With Debut_de_liste^ do begin
                          Precedent:=NIL;
                                 x:=1;
                                 y:=1;
                                 r:=0;
                          end;
 Ptr1:=Debut_de_liste;
 x:=1; y:=1; r:=0;

 REPEAT {--------------------Déplacement-------------------------------------}

 While (Sens_interdit(r,x,y)) and (r<4) do r:=r+1;   { Cherche r pour avancer }
 IF r=4
 THEN begin                                                      { On recule }
      Ptr2:=Ptr1^.precedent;
      r:=Ptr2^.r;
      x:=Ptr2^.x;
      y:=Ptr2^.y;
      Direction(r,dx,dy);
      Mem[ad:160*(2*y+dy)+2*(2*x+dx)]:=32;
      Mem[ad:320*(y+dy)+4*(x+dx)]:=32;
      DISPOSE(Ptr1);
      Ptr1:=Ptr2;
      r:=r+1;
      end
 ELSE begin                                                     { On avance }
      NEW(Ptr2);
      Ptr2^.precedent:=Ptr1;
      Ptr1^.r:=r;
      Direction(r,dx,dy);
      if (r=0) or (r=3) then Mem[ad:160*(2*y+dy)+2*(2*x+dx)]:=196
                        else Mem[ad:160*(2*y+dy)+2*(2*x+dx)]:=179;
      x:=x+dx;
      y:=y+dy;
      Mem[ad:320*y+4*x]:=219;
      Ptr2^.x:=x;
      Ptr2^.y:=y;
      Ptr2^.r:=0;
      r:=0;
      Ptr1:=Ptr2;
      end;
 UNTIL (x=xmax) and (y=ymax);

{--------------------------Fin du programme-----------------------------------}
 sound(1500);delay(50);Nosound;gotoxy(1,24);
End.
```