Pour le comité d'édition, à décommenter au moment de la publication

Cet article est rédigé par des élèves. Il peut comporter des oublis ou des imperfections, autant que possible signalés par nos relecteurs dans les notes d'édition.

Taquin

2016-2017

Nom, prénom et niveaux des élèves : Pierre ZAMPIN, Kyllian HOLDER, Camille Marin-Cudraz : élèves de Première S.

Établissement : Lycée du Mont-Blanc, Passy

Enseignant(s) : Hélene Langlais, Cyril Masson

Chercheur(s) : Pierre-Alain Chérix; Martin J. Gander, Université de Genève

Table des matières

Résumé

Nous avons travaillé sur le taquin un jeu composé de 8 cases chiffrées de 1 à 8 et d'une case vide. Nous avons tenté de prouver que l'on pouvait savoir si le taquin est résoluble en calculant les permutations nécessaires puis en le passant dans un algorithme qui le résout.

1 Introduction

Inventé à la fin du XIX ème siècle par Sam Loyd, le taquin est un jeu en forme de damier qui consiste à déplacer des cases pour reconstituer une image ou une suite de chiffres proposée avant le mélange des pièces tout en respectant les limites physiques du jeu. Ce jeu s'inscrit dans le thème "Quelle est la différence entre la façon de jouer d'un ordinateur et celle d'un humain". Nous avons essayé de chercher des techniques pour pouvoir le résoudre avec et sans l'usage de l'ordinateur. Nous avons d'abord travaillé sur un taquin composé de 4 cases réparties en 2*2, puis nous avons ensuite étudié un taquin de 9 cases, en 3*3. Et après avoir cherché des renseignements sur le taquin, et sur les techniques permettant de le résoudre, nous avons essayé de créer avec le logiciel Scilab, un algorithme permettant de voir si toutes les configurations générées sont résolubles.

2 Historique du jeu

Ce jeu fut créé il y a environ 140 ans, dans le années 1870 par Sam Loyd, un créateur américain de casses-têtes utilisants les mathématiques. Il avait réussi a inventer un jeu qui pouvait ne pas être résolu. Ceci lui a permis de pouvoir lancer des paris dans les rues New-Yorkaises, et de les remporter sans aucune difficultés. Certains diront qu'il était opportuniste, d'autres diront que c'était plutôt un escroc. Il s'agit d'un jeu très varié, puisque si on prend l'exemple du taquin 3X3, le nombre de combinaisons possibles (c'est à dire de matrice créées) est 9! (la factorielle de 9). C'est à dire qu'il existe 362880 combinaisons différentes possibles.

3 Les Permutations

Après quelques recherches, et l'aide d'internet, nous avons trouvé qu'il existait une façon de savoir si le jeu était résolvable ou non. Cette dernière consistait à inverser les cases du jeu pour les remettre à leur place initiale en oubliant (uniquement dans ce cas) les limites physiques du jeu, tout en comptant le nombre de permutations effectuées. Dès que la configuration initiale de la matrice était reconstituée, c'est à dire dès que chaque case était remise à sa place, il suffisait de compter le nombre de permutations qu l'on venait de faire, puis, 2 cas de figures différents se présentaient après :

-le nombre de permutations trouvé est pair (0,2,4,...) la matrice présentée possédait une solution, et il était donc possible de la résoudre.

-le nombre de permutations trouvé est impair (1,3,5,...), dans ce cas, la matrice ne possède pas de solution et il est inutile d'essayer de la résoudre.

3.1 La programmation avec le logiciel Scilab

Avec ces nouvelles informations, nous avons essayé de metre en pratique la résolution du taquin avec un logiciel de programmation : Scilab. Nous avons divisé la résolution de ce jeu en plusieurs petits programmes en commençant tout d'abord par générer une matrice aléatoire de 9 cases réparties en 3x3.

Puis nous avons faits plusieurs programmes pour résoudre notre problème initial.

Nous avons créé un programme qui compte le nombre d'erreurs. Le principe de ce programme est simple, l'ordinateur scrute toutes les cases du jeu, et dès qu'une case ne contient pas le bon chiffre, l'ordinateur crée un compteur et ajoute 1 erreur. Il continue ensuite avec toutes les autres cases du jeu jusqu'à ce qu'il arrive à la neuvième et dernière case.

Il affiche ensuite le nombre d'erreurs trouvés, et à partir de ce résultat, un autre programme entre en jeu et décide si la matrice ne doit pas être changée, (si le nombre d'erreurs est égal à 0) ou bien si il faut commencer à chercher le nombre de permutations dans les 9 cases, si le nombre d'erreurs trouvé précedemment est cette fois supérieur à 0.

Ce second programme fonctionne avec le même principe que le premier, il scrute chaque case, et compte ensuite le nombre de permutations nécessaires pour remetre la matrice à son état intial. Et selon le résultat trouvé, on utilise la conjecture marquée avant qui dit que si le nombre trouvé est pair, le taquin sera résolvable, alors que si le nobmre trouvé est impair, ce sera le contraire.

3.2 Les modifications de la matrice

Trouver le nomvre de permutations correspondait seulement au début des recherches, il fallait ensuite continuer la programmation pour réussir à modifier la grille du taquin, et essayer de remettre toutes le cases à leur bon emplacement. Nous avons donc repris les bouts de papiers que nous avions découpé au début des recherches, et nous nous en sommes servis pour trouver un moyen de résoudre le jeu. Nous nous sommes donc concentrés sur les déplacements possibles selon la position des cases dans la matrice, puis nous avons resorti une feuille de papier pour noter les différentes positions existantes

1 –> 2 déplacements	2 –> 3 déplacements	3 –> 2 déplacements
4 –> 3 déplacements	5 –> 4 déplacements	5 –> 3 déplacements
7 –> 2 déplacements	8 –> 3 déplacements	9 –> 2 déplacements

On constate que selon la position du vide dans la matrice, entre les cases 1 et 9, il existe entre 2 et 4 déplacements possibles, ce qui implique la céation de 2 à 4 nouvelles matrices. Pour expliquer le tableau :

-Il existe 4 déplacements possible :
Vers le haut
Vers le bas
Vers la gauche
Vers la droite
Mais encore une fois, l'application de ces déplacements reste compliqué :
Déplacement vers le haut : 5,6,7,8,9

vers le bas : 1,2,3,4,5,6

vers la droite : 1,2,4,5,7,8

vers la gauche : 2,3,5,6,8,9

Il fallait donc trouver un moyen de différencier les déplacements pour pouvoir former de nouvelles matrices. Pour ce faire, nous sommes partis de l'hypothèse que la disposition des cases proposée permettait de résoudre le taquin. Mais le fait de savoir le nombre de déplacements ne suffisait pas, il a donc fallu continuer à extraire les informations pour chercher à résoudre le jeu.

3.3 Les différents petits programmes

Nous n'avons pas conçu un gros programme qui pemettait de résoudre le taquin en un seul clic, nous avons d'abord créé plusieurs petits programmes qui effectuaient chacun une étape de résolution. Au total, nous avons eu le temps de créer 9 programmes pour entamer la résolution. Il y a eu :

1. Matrice, un programme permettant de générer aléatoirement une matrice
2. mattonb, (matrice to number) qui transfome la matrice en un nombre pour nous permettre de travailler plus facilement dessus.
3. nbtovec, qui transforme le nombre trouvé en vecteur
4. vectomat, qui transforme le vecteur en matrice

La conception de ces programmes fut l'étape la plus compliqué dans nos recherches, puisque nos compétences en informatique étaient toutes différentes, et certains découvraient seulement le logiciel Scilab. Mais grâce aux notions des 2 élèves qui connaissaient le logiciel, et grâce à l'aide des chercheurs et de nos professeurs, nous avons réussi à conçevoir tous ces petits progammes, melant ainsi mathématiques et informatique.

3.4 Principe du programme général

Cet ensemble de petits programmes devait nous servir à créer un programme pour pouvoir résoudre le taquin. Il devait permettre de créer une matrice aléatoirement avec à chaque fois une position différente pour la case vide, puis ensuite calculer le nombre d'erreurs, c'est à dire le nombre de cases à la mauvaise place pour voir si le taquin n'était pas déjà résolu. Il fallait ensuite compter le nombre de permutations nécessaires pour voir si l'on trouvait un nombre impair ou pair, nous permettant de résoudre la matrice. Il fallait ensuite transformer la matrice créée en un vecteur, puis en un nombre pour pouvoir ensuite voir le nombre de nouvelles configurations possibles. Il devait ensuite stocker toutes ces combinaisons jusqu'à en trouver une avec un nombre de cases mal placées inférieur à celui intial. Le programme qui compte les erreurs devait revenir à chaque étape pour voir si il restait des modifications à effectuer, et dès lors qu'il trouvait un nombre égal à zéro, il arrêtait le programme et affichait la matrice finale qui devait être résolue et le nombre de permutations trouvé initialement. Mais si le compteur n'arrivait jamais à zéro, un programme d'arrêt devait stopper le programme pour lui éviter de tourner indéfiniment. Ce programme devait nous permettre de tester une quantité importante de matrices pour pouvoir essayer de confirmer notre conjecture qui disait que seuls les taquins avec un nombre de permutations pair avaient une solution.

4 La fin du travail

4.1 La présentation au congrès

Après quelques mois de travail, nous avons pu exploiter quelques pistes de recherche, que nous avons rassemblé pour pouvoir faire sa présentation au mois de mars, au congrès de Grenoble, devant les autres groupes de recherche.

4.2 Poursuite des recherches ...

Après avoir présenté notre travail, nous avons et même nous sommes en train de continuer notre travail pour pouvoir finaliser le travail, et répondre à la problématique posée précédemment en finissant le programme, pour qu'il puisse nous permettre de de confirmer notre conjecture sur la position du vide, et le déplacement des cases, et leur infuences sur la possibilité de résoudre ou non un taquin, en le testant sur une grande quantité de matrice.

5 Annexes : Les différents programmes créés

Le premier porgramme qui consiste à créer de façon aléatoire une matrice qui représente la grille du taquin :

```
    function M=taquin()
  y=[];
  for i = 1:8;
     x=round(rand()*8+0.5);
     while find(y==x);
        x=round(rand()*8+0.5);
     end;
     y(i)=x;
  end
  M=matrix([y;0],3,3)
endfunction
```

Voici ensuite le programme qui vérifie si tous les chiffres sont à la bonne place dans la matrice, pour voir le nombre de permutations nécessaires :

```
function NbDeplacements=ResTaquin(L)
    NbDeplacements=0;
    [i,j]=find(L==1);
    if L(1,1)<>1 then
        L(i,j)=L(1,1);
        L(1,1)=1;
        NbDeplacements=NbDeplacements+1
    end
    [i,j]=find(L==2);
    if L(1,2)<>2 then
        L(i(1),j(1))=L(1,2);
        L(1,2)=2;
        NbDeplacements=NbDeplacements+1
    end
    [i,j]=find(L==3);
    if L(1,3)<>3 then
        L(i(1),j(1))=L(1,3);
        L(1,3)=3;
        NbDeplacements=NbDeplacements+1
    end
    [i,j]=find(L==4);
    if L(2,1)<>4 then
```

```
        L(i(1),j(1))=L(2,1);
        L(2,1)=4;
        NbDeplacements=NbDeplacements+1
    end
    [i,j]=find(L==5);
    if L(2,2)<>5 then
        L(i(1),j(1))=L(2,2);
        L(2,2)=5;
        NbDeplacements=NbDeplacements+1
    end
    [i,j]=find(L==6);
    if L(2,3)<>6 then
        L(i(1),j(1))=L(2,3);
        L(2,3)=6;
        NbDeplacements=NbDeplacements+1
    end
    [i,j]=find(L==7);
    if L(3,1)<>7 then
        L(i(1),j(1))=L(3,1);
        L(3,1)=7;
        NbDeplacements=NbDeplacements+1
    end
    [i,j]=find(L==8);
    if L(3,2)<>8 then
        L(i(1),j(1))=L(3,2);
        L(3,2)=8;
        NbDeplacements=NbDeplacements+1
    end
    [i,j]=find(L==1);
    if L(3,3)<>0 then
        L(i(1),j(1))=L(3,2);
        L(3,3)=0;
        NbDeplacements=NbDeplacements+1
    end
    NbDeplacements
endfunction
```

Ce programme ci nous a permis de trouver le nombre de nouvelles matrices créées après avoir modifié la position de la case vide, c'est à dire après avoir effectué un déplacement :

```
function ListS=PositionV(M)
    [i,j]=find(M==0)
    ListS=[]
    M2=M //déplacement vers le haut
    M3=M //déplacement vers le bas
    M4=M // déplacement vers la gauche
    M5=M // déplacement vers la droite
    if i >=2 then //si i>= 2 alors on peut deplacer vers le haut
    M2(i,j)=M(i-1,j)
    M2(i-1,j)=0
    ListS=[[M2],ListS]
   end
   if i <=2 then //si i<= 2 alors on peut deplacer vers le bas
       M3(i,j)=M(i+1,j)
    M3(i+1,j)=0
   ListS=[[M3],ListS]
   end
   if j >=2 then //si j>= 2 alors on peut deplacer vers la gauche
    M4(i,j)=M(i,j-1)
    M4(i,j-1)=0
    ListS=[[M4],ListS]
   end
   if j <=2 then //si j<= 2 alors on peut deplacer vers la droite
    M5(i,j)=M(i,j+1)
    M5(i,j+1)=0
    ListS=[[M5],ListS]
   end
endfunction
```

Pour finir, ce programme nous a servi pour compter le nombre d'erreurs, c'est à dire de cases à modifier, qui vient en complément du programme NbDeplacments, mais au lieu de nous donner le nombre de permutations, celui-ci affiche un chiffre qui correspond au nombre de cases à modifier :

```
function NbErreur=Nbe(M)
    NbErreur=0
    if M(1,1)<>1 then
        NbErreur=NbErreur+1
    end
        if M(1,2)<>2 then
        NbErreur=NbErreur+1
    end
        if M(1,3)<>3 then
        NbErreur=NbErreur+1
    end
        if M(2,1)<>4 then
        NbErreur=NbErreur+1
    end
        if M(2,2)<>5 then
        NbErreur=NbErreur+1
    end
        if M(2,3)<>6 then
        NbErreur=NbErreur+1
    end
        if M(3,1)<>7 then
        NbErreur=NbErreur+1
    end
        if M(3,2)<>8 then
        NbErreur=NbErreur+1
    end
        if M(3,3)<>0 then
        NbErreur=NbErreur+1
    end
endfunction
```

Il s'agit ici d'une partie des programmes créés durant l'année, qui nous ont permis de pouvoir avancer dans nos recherches et dans la création de nouveaux programmes au fil de l'année.