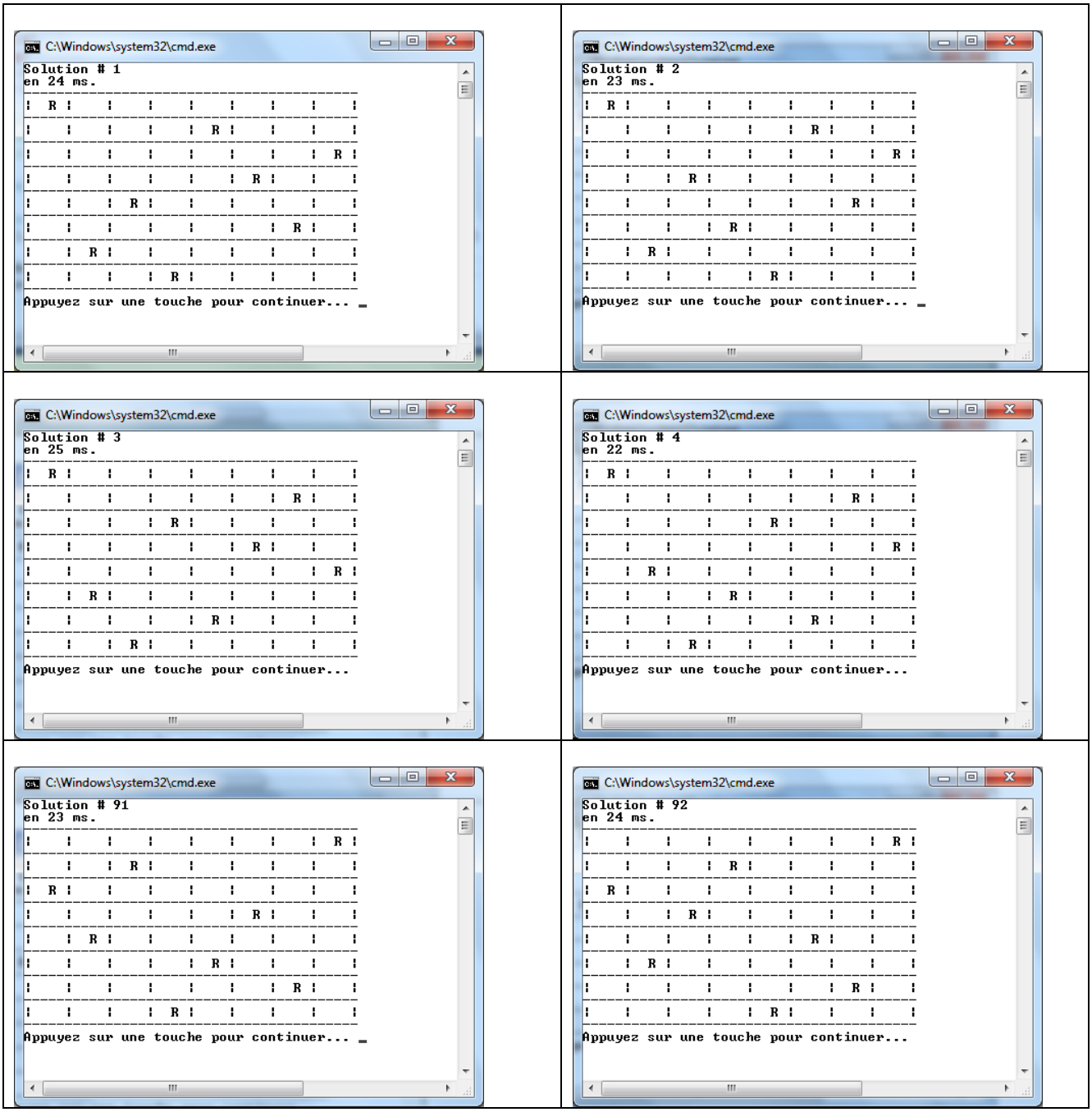
Voici un autre problème classique de récursivité qui fait appel à une notion de *backtracking*. Le problème consiste à placer 8 reines sur un échiquier sans qu'aucune ne puisse attaquer une autre reine qui s'y trouve. Voici quelques unes des 92 solutions possibles…

Quelques solutions possibles au problème



On trouve donc la première solution en 0,0023 seconde, la deuxième en 0,0023 seconde, la troisième en 0,0025 seconde et ainsi de suite.

La résolution de ce problème fait appel à une approche que l'on appelle le « ***backtracking*** » qui est une variante de l'idée de récursivité. Alors que traditionnellement la récursivité permet de trouver la solution à un problème de manière directe – pensez au parcours d'un arbre binaire – le *backtracking* consiste à progresser vers une solution en faisant l'hypothèse d'un prochain pas puis, à partir de ce point, on trouve le prochain pas et ainsi de suite. Si on arrive à un cul-de-sac, on revient d'un pas (d'où l'idée de *backtracking*) puis on repart de l'avant. On procède ainsi jusqu'à ce qu'on trouve la solution.

Tout l'art se trouve dans la manière de programmer les essais récursifs des possibilités afin d'évacuer rapidement celles qui ne sont pas porteuses d'un espoir de solution.

Dans le cas du problème des 8 reines, l'algorithme qui permet de placer 8 Reines consiste simplement à

Pour toutes les positions à partir du point de départ reçu
- Placer une Reine 'R' sur une case libre de l'échiquier
- Bloquer les cases que cette Reine peut attaquer car elles ne sont plus disponibles

- Si (les 8 Reines sont placées) le problème est résolu, on affiche
- Sinon placer les N-1 reines qui restent plus loin sur l'échiquier

- Débloquer les cases que la Reine 'R' peut atteindre parce qu'elles redeviennent disponibles
- Retirer la Reine 'R' de la case où elle se trouve présentement

Voici le code du programme qui permet de résoudre ce problème. Nous avons d'abord une classe **CBoard** qui permet de définir notre échiquier :

```
1   // Board.h: interface for the CBoard class.
2   //
3   //////////////////////////////////////////////////////////////////////
4   #pragma once
5   #include "Matrice.h"
6   #include <chrono>
7   #include <iostream>
8   #include <vector>
9   using namespace std;
10  using namespace std::chrono;
11
12  class CBoard
13  {
14  public:
15        //--- Constructeur et destructeur
16        CBoard(ostream &, bool bVeutTrace);
17
18        //--- Méthode publique pour mettre en marche la recherche
19        void RechercherSolution();
20
21  private:
22        void PlacerReine   (int NbReines, int i, int j);
23        void Afficher      (bool bTrace = false);
24        void Afficher      (int TempsAttente) const;
25        bool EstDisponible (int x, int y) const;
26        void DebloquerCases(int x, int y);
27        void BloquerCases  (int x, int y);
28        void ChangerEtat   (int x, int y, int iValeur);
29        void Wait          (int iNbMilliSec) const;
30        int  GetNbSolutions() const;
31        void SetNbSolutions(int);
32
33  private:
34        static const int iMAXCASES;     // Nombre de cases de l'échiquier
35        Matrice<int> CaseUtilisable;    // Un échiquier d'entiers
36        vector <int> TabPositionReines; // Vecteur des positions des
37  reines (colonne seulement)
38        bool bFaireTrace_;              // Signification évidente
39        int iNbSolutions_;              // idem
40        ostream & rOut_;
41        system_clock::time_point debut_;
42  };
43
```

Définition de la classe CarreMagique

```
44  // Board.cpp: implementation of the CBoard class.
45  //
46  //////////////////////////////////////////////////////////////////////
47  #include "Board.h"
48  #include <string>
49  using namespace std;
50  using namespace std::chrono;
51
52  //////////////////////////////////////////////////////////////////////
53  // Construction/Destruction
54  //////////////////////////////////////////////////////////////////////
55
56  const int CBoard::iMAXCASES = 8;  // Nombre de cases de l'échiquier
57
58  CBoard::CBoard(ostream & out, bool bVeutTrace)
59        : rOut_(out), bFaireTrace_(bVeutTrace), CaseUtilisable(iMAXCASES,
60  iMAXCASES)
61  {
62        TabPositionReines.resize(iMAXCASES,-1);
63        SetNbSolutions(0);
64  }
65
66
```

```
67  // NbReines est le nombre de Reines qui restent à placer
68  // i et j sont les coordonnées à partir desquelles placer
69  // les reines qui restent
70  void CBoard::PlacerReine(int NbReines, int i, int j)
71  {
72      for (int x = i; x < iMAXCASES; x++)
73        for (int y = j; y < iMAXCASES; y++)
74        {
75          // Peut-on placer une reine à cette position ligne, colonne?
76          if (EstDisponible(x, y))
77          {
78            // Si oui, placer la reine et bloquer les cases qu'elle
79                        // peut 'attaquer'
80            BloquerCases(x,y);
81            NbReines--;        // il y a une reine de moins à placer
82
83            // Le paramètre entier représente le nombre de
84            // millisecondes à attendre
85            // un nombre négatif fait une pause
86
87            // Afficher(500);
88
89            if (NbReines == 0)
90            { // Fini ! On affiche
91              SetNbSolutions(GetNbSolutions() + 1);
92              Afficher();
93            }
94            else
95            {
96              // sinon on poursuit la recherche à partir de la prochaine ligne
97              PlacerReine(NbReines, x+1, 0);
98            }
99
100           // arrivé ici, la recherche récursive d'une
101           // solution pour n-1 reine est terminée
102           // donc on enlève la reine pour la placer plus
103           // loin sur l'échiquier et reprendre le processus
104           NbReines++;
105           DebloquerCases(x,y);
106         }
107       }
108 }
109
110 void CBoard::BloquerCases(int x, int y)
111 {
112       // Retenir la position de la reine placée dans l'échiquier
113       TabPositionReines[x] = y;
114
115       // Augmenter de 1 les compteurs de blocages
116       ChangerEtat(x, y, +1);
117       if (bFaireTrace_) Afficher(true);
118 }
119
120
```

```
121  void CBoard::DebloquerCases(int x, int y)
122  {
123        // Retirer la reine de sa position sur l'échiquier
124        TabPositionReines[x] = -1;
125
126        // Diminuer de 1 les compteurs de blocages
127        ChangerEtat(x, y, -1);
128        if (bFaireTrace_) Afficher(true);
129  }
130
131  void CBoard::ChangerEtat(int x, int y, int iValeur)
132  {
133        int i, j;
134
135        // Modifier les compteurs de blocages pour la ligne
136        for (j=0; j<iMAXCASES; j++)
137              CaseUtilisable[x][j] += iValeur;
138
139        // Modifier les compteurs de blocages pour la colonne
140        for (i=0; i<iMAXCASES; i++)
141              CaseUtilisable[i][y]+= iValeur;
142
143        // Modifier les compteurs de blocages pour les diagonales
144        for (i=1; i<iMAXCASES; i++)
145        {
146              if (((x-i) >= 0) && ((y-i) >= 0))
147                    CaseUtilisable[x-i][y-i]+= iValeur;
148              if (((x-i) >= 0) && ((y+i) < iMAXCASES))
149                    CaseUtilisable[x-i][y+i]+= iValeur;
150              if (((x+i) < iMAXCASES) && ((y-i) >= 0))
151                    CaseUtilisable[x+i][y-i]+= iValeur;
152              if (((x+i) < iMAXCASES) && ((y+i) < iMAXCASES))
153                    CaseUtilisable[x+i][y+i]+= iValeur;
154        }
155  }
156
157  bool CBoard::EstDisponible(int x, int y) const
158  {
159        return CaseUtilisable[x][y] == 0;
160  }
161
162  void CBoard::Afficher(bool bTrace)
163  {
164        if (&rOut_ == &cout) system("cls");
165        if (!bTrace)
166        {
167              auto stop = system_clock::now();
168              auto tempsEcoule = stop - debut_;
169              rOut_ << "Solution # " << iNbSolutions_ << endl;
170              rOut_ << "en " << duration_cast<milliseconds>(tempsEcoule).count()
171                    << " ms." << endl;
172        }
173        rOut_ << string(41,'-') << endl;
174        for (int i=0; i<iMAXCASES; i++)
175        {
176              rOut_ << "| ";
```

```
177               for (int j=0; j<iMAXCASES; j++)
178               {
179                     if ( j == this->TabPositionReines[i])
180                     {
181                           rOut_ << " R | ";
182                     }
183                     else if (bTrace && CaseUtilisable[i][j] != 0
184                                       && CaseUtilisable[i][j] != -1)
185                     {
186                           rOut_.width(2);
187                           rOut_ << CaseUtilisable[i][j] << " | ";
188                     }
189                     else
190                     {
191                           rOut_ << "   | ";
192                     }
193               }
194               rOut_ << endl;
195               rOut_ << string(41,'-') << endl;
196         }
197         if (&rOut_ == &cout && !bTrace) system("pause");
198         if (bTrace) Wait(1);
199         debut_ = system_clock::now();  // pour trouver le nb de millisecondes
200                               // requises pour la prochaine solution (pas un cumul)
201
202   }
203
204   void CBoard::Afficher(int TempsAttente) const
205   {
206         system("cls");
207         rOut_ << string(41,'-') << endl;
208         for (int i=0; i<iMAXCASES; i++)
209         {
210               rOut_ << "| ";
211               for (int j=0; j<iMAXCASES; j++)
212               {
213                     if ( j == this->TabPositionReines[i])
214                     {
215                           rOut_ << " R | ";
216                     }
217                     else
218                     {
219                           rOut_ << "   | ";
220                     }
221               }
222               rOut_ << endl;
223               rOut_ << string(41,'-') << endl;
224         }
225         if (TempsAttente >= 0)
226               Wait(TempsAttente);
227         else
228               system("pause");
229   }
230
231   void CBoard::Wait(int iNbMilliSec) const
232   {
```

```
233         auto start = system_clock::now();
234         auto stop = system_clock::now();
235         auto tempsEcoule = stop - start;
236
237         while (duration_cast<milliseconds>(tempsEcoule).count() < iNbMilliSec)
238         {
239               stop = system_clock::now();
240               tempsEcoule = stop - start;
241         }
242   }
243
244   int  CBoard::GetNbSolutions () const
245   {
246         return iNbSolutions_;
247   }
248
249   void CBoard::SetNbSolutions (int i)
250   {
251         iNbSolutions_ = i;
252   }
253
254
255   void CBoard::RechercherSolution()
256   {
257         debut_ = system_clock::now();
258         PlacerReine(8,0,0);
259   }
260
261
```

```
262  //---------------------------------------------------------- main.cpp ---
263  //  Programme principal en mode console permettant de
264  //  mettre en marche la solution au problème des
265  //  8 reines
266  //
267  //  par Pierre Prud'homme, décembre 2009, révision novembre 2015
268  //-----------------------------------------------------------------------
269  #include "board.h"
270  #include <fstream>
271  #include <iostream>
272  using namespace std;
273
274  bool VeutUneTrace();
275
276  int main()
277  {
278        system("Color f0");
279        ofstream out("sortie.txt");
280
281        CBoard oEchiquier (cout, VeutUneTrace());
282        oEchiquier.RechercherSolution();
283  }
284
285  bool VeutUneTrace()
286  {
287        char cReponse;
288
289        cout << "Voulez-vous voir la trace d'execution (O/N) : ";
290        cin >> cReponse;
291        while (toupper(cReponse) != 'N' &&
292                  toupper(cReponse) != 'O')
293        {
294              cout << "Voulez-vous voir la trace d'execution (O/N) : ";
295              cin >> cReponse;
296        }
297        return (toupper(cReponse) == 'O');
298  }
299
```

Exemple de trace avec les compteurs (telle que mise en commentaire dans cette version)

```
| R | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 |   |   |   |   |   |   |
| 1 |   | 1 |   |   |   |   |   |
| 1 |   |   | 1 |   |   |   |   |
| 1 |   |   |   | 1 |   |   |   |
| 1 |   |   |   |   | 1 |   |   |
| 1 |   |   |   |   |   | 1 |   |
| 1 |   |   |   |   |   |   | 1 |
```

```
| R | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
| 2 | 2 | R | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 1 |   |   |   |   |
| 2 |   | 1 | 1 | 1 |   |   |   |
| 1 |   | 1 |   | 1 | 1 |   |   |
| 1 |   | 1 |   |   | 1 | 1 |   |
| 1 |   | 1 |   |   |   | 1 | 1 |
| 1 |   | 1 |   |   |   |   | 1 |
```

```
| R | 2 | 3 | 2 | 2 | 1 | 2 | 1 |
| 2 | 2 | R | 2 | 2 | 2 | 1 | 1 |
| 2 | 2 | 3 | 2 | R | 1 | 1 | 1 |
| 2 |   | 1 | 2 | 2 | 1 |   |   |
| 1 |   | 2 |   | 2 | 1 | 1 |   |
| 1 | 1 | 1 |   | 1 | 1 | 1 | 1 |
| 2 |   | 1 |   | 1 |   | 1 | 1 |
| 1 |   | 1 |   | 1 |   |   | 1 |
```

```
| R | 3 | 3 | 2 | 3 | 1 | 2 | 1 |
| 2 | 3 | R | 3 | 2 | 2 | 1 | 1 |
| 3 | 3 | 4 | 2 | R | 1 | 1 | 1 |
| 3 | R | 2 | 3 | 3 | 2 | 1 | 1 |
| 2 | 1 | 3 |   | 2 | 1 | 1 |   |
| 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 |   | 2 |   | 1 | 1 |
| 1 | 1 | 1 |   | 1 | 1 |   | 1 |
```

```
| R | 3 | 3 | 3 | 3 | 1 | 2 | 2 |
| 3 | 3 | R | 4 | 2 | 2 | 2 | 1 |
| 3 | 4 | 4 | 3 | R | 2 | 1 | 1 |
| 3 | R | 3 | 4 | 4 | 2 | 1 | 1 |
| 3 | 2 | 4 | R | 3 | 2 | 2 | 1 |
| 1 | 2 | 2 | 2 | 2 | 1 | 1 | 1 |
| 2 | 2 | 1 | 1 | 2 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
```

```
| R | 3 | 3 | 2 | 3 | 1 | 2 | 1 |
| 2 | 3 | R | 3 | 2 | 2 | 1 | 1 |
| 3 | 3 | 4 | 2 | R | 1 | 1 | 1 |
| 3 | R | 2 | 3 | 3 | 2 | 1 | 1 |
| 2 | 1 | 3 |   | 2 | 1 | 1 |   |
| 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 |   | 2 |   | 1 | 1 |
| 1 | 1 | 1 |   | 1 | 1 |   | 1 |
```