

UNIVERSITÉ LIBRE DE BRUXELLES  
FACULTÉ DES SCIENCES  
DÉPARTEMENT D'INFORMATIQUE

# INFO-F-302 Informatique Fondamentale : Rapport de projet

PERALE Thomas  
REQUENA Carlos



# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Question 1 : Le problème d'indépendance</b>	<b>2</b>
2.1	Définition du problème . . . . .	2
2.2	Définition des variables . . . . .	2
2.3	Contraintes . . . . .	2
<b>3</b>	<b>Question 2 : Le problème de domination</b>	<b>2</b>
3.1	Définition du problème . . . . .	2
3.2	Définition des variables . . . . .	3
3.3	Contraintes . . . . .	3
<b>4</b>	<b>Question 4 : Les chevaliers minimum</b>	<b>3</b>
4.1	Définition du problème . . . . .	3
4.2	Définition des variables . . . . .	4
4.3	Contraintes . . . . .	4
<b>5</b>	<b>Question 5 : La surveillance de musée</b>	<b>4</b>
<b>A</b>	<b>Code Listing</b>	<b>5</b>

# 1 Introduction

Le but du projet est de modéliser des problèmes de satisfaction de contraintes à l'aide de l'outil de résolution de contraintes **ChocoSolver**<sup>1</sup>. Pour cela on nous présente plusieurs problèmes en un premier temps basé sur un échiquier et la modélisation des pièces de celui-ci. En un deuxième temps basé sur la disposition d'un musée et de l'emplacement que doivent avoir les cameras dans celui-ci.

## 2 Question 1 : Le problème d'indépendance

### 2.1 Définition du problème

Comme il a été dit dans l'énoncé le problème d'indépendance consiste à déterminer dans un échiquier de taille donné si il est possible d'assigner à chacune des pièces mises à notre disposition (c'est à dire autant de chevalier, tour ou fou que l'on veut) une position distincte de sorte qu'aucune pièce ne menace une autre pièce.

### 2.2 Définition des variables

- $n$  la taille de l'échiquier ;
- $k_1$  tours ;
- $k_2$  fous ;
- $k_3$  cavalier ;

Donc nos variables sont l'union de tous les types des pièces, avec leur tuples indiquant : (type, coordonnée<sub>x</sub>, coordonnée<sub>y</sub>) :

$$X = \{k_{1,i,j} | 1 \leq i, j < n\} \cup \{k_{2,i,j} | 1 \leq i, j < n\} \cup \{k_{3,i,j} | 1 \leq i, j < n\}$$

Pour chaque pièce, on peut attribuer une valeur à sa valeur  $i$  et  $j$ , donc le domaine de chaque variable devient :

$$D_t = \{(i, j) \dots (n, n)\} \text{ pour tout type } t = 1, 2, 3\}$$

### 2.3 Contraintes

- Chaque pièce doit occuper une case différente.

$$C_{diff} = (\forall(k_1, k_2, k_3)), \{\forall(i, j) | i \neq i' \wedge j \neq j'\}$$

- Aucune pièce ne peut occuper les cases situées à gauche, à droite, en haut et en bas des toutes les tours.

$$C_t = (\forall(k_1, k_2, k_3)), \{\forall(i, j) \in k_1 | i \neq i' \vee j \neq j'\}$$

- Aucune pièce ne peut occuper les cases situées en diagonale des tous les fous.

$$C_f = (\forall(k_1, k_2, k_3)), \{\forall(i, j) \in k_2 | i \neq i' \pm 1 \vee j \neq j' \pm 1\}$$

- Aucune pièce ne peut occuper les cases menacées par tous les cavaliers. Voir (2.3)

## 3 Question 2 : Le problème de domination

### 3.1 Définition du problème

Ce problème cherche à ce que chaque case de l'échiquier doive être soit occupée soit menacée par au moins une pièce.

---

1. <http://www.choco-solver.org>

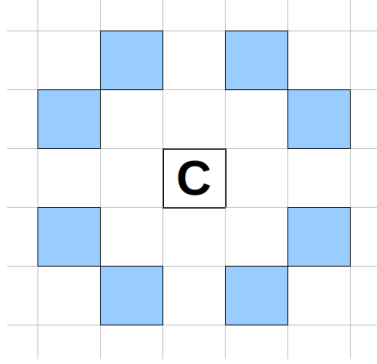


FIGURE 1 – Cases menacées par un cavalier



FIGURE 2 – Commande : `java -jar jarfile -i -n 6`

### 3.2 Définition des variables

- $n$  la taille de l'échiquier ;
- $k_1$  tours ;
- $k_2$  fous ;
- $k_3$  cavalier ;

### 3.3 Contraintes

- Chaque pièce doit occuper une case différente.
- Toute pièce doit occuper soit :
  - les cases situées à gauche, à droite, en haut et en bas des toutes les tours.
  - les cases situées en diagonale des tous les fous.
  - les cases menacées par tous les cavaliers (2.3).

## 4 Question 4 : Les chevaliers minimum

### 4.1 Définition du problème

Ce problème calcule le nombre minimal de cavaliers permettant de dominer un échiquier de taille donné.

Pour résoudre ce problème, il faut changer de tactique avec ChocoSolver. On ne cherche pas à savoir si une solution existe, mais plutôt à connaître la quantité minimale de chevaliers nécessaires pour menacer tout l'échiquier :



FIGURE 3 – Commande : `java -jar jarfile -d -n 6 -t 1 -f 4 -c 3`

```
1 // to minimise X
2 model.setObjectives(Model.MINIMIZE, X);
```

Listing 1 –

## 4.2 Définition des variables

- $n$  la taille de l'échiquier ;

## 4.3 Contraintes

- Chaque pièce doit occuper une case différente. Chaque cavalier doit occuper une et une seule case libre de l'échiquier.
- Toutes les cases de l'échiquier doivent menacées par au moins un cavalier. C'est à dire, toutes les cases devront être bleues dans la figure (2.3), sauf les cases déjà occupées par les cavaliers.



FIGURE 4 – Commande : `java -jar jarfile -mk -n 6`

## 5 Question 5 : La surveillance de musée

Ici le problème est essentiellement le même, appliqué à la surveillance d'un musée. Nos quatre pièces peuvent s'orienter Nord, Sud, Est et Ouest.

## A Code Listing

```
1 package chocolate;
2
3 import chocolate.Domination;
4 import chocolate.Museum;
5
6 import org.chocosolver.solver.Model;
7 import org.chocosolver.solver.Solver;
8 import org.chocosolver.solver.Solution;
9 import org.chocosolver.solver.search.strategy.Search;
10 import org.chocosolver.solver.variables.IntVar;
11
12 import com.google.common.collect.HashBasedTable;
13 import com.google.common.collect.Table;
14
15 import java.io.*;
16
17
18 // Argparse
19 import net.sourceforge.argparse4j.ArgumentParsers;
20 import net.sourceforge.argparse4j.inf.ArgumentParser;
21 import net.sourceforge.argparse4j.inf.ArgumentParserException;
22 import net.sourceforge.argparse4j.inf.Namespace;
23 import net.sourceforge.argparse4j.inf.MutuallyExclusiveGroup;
24 import net.sourceforge.argparse4j.impl.Arguments;
25
26
27 public class App {
28
29     public static void main(String[] args) {
30         ArgumentParser parser = ArgumentParsers.newArgumentParser("Chess pieces parser")
31             .defaultHelp(true)
32             .description("Returns a solution for the independence or domination problem for
the given pieces (only rooks, knights and bishops)");
33         MutuallyExclusiveGroup indom = parser.addMutuallyExclusiveGroup("Independence/
Domination");
34         indom.addArgument("-i").action(Arguments.storeTrue())
35             .help("Solve the independence problem");
36         indom.addArgument("-d").action(Arguments.storeTrue())
37             .help("Solve the domination problem");
38         indom.addArgument("-mk").action(Arguments.storeTrue())
39             .help("Solve the minimum knights problem");
40         indom.addArgument("-m").action(Arguments.storeTrue())
41             .help("Solve the museum problem");
42         parser.addArgument("-n")
43             .type(Integer.class)
44             .help("Generate a chess board NxN")
45             .setDefault(5);
46         parser.addArgument("-t")
47             .type(Integer.class)
48             .help("Number of rooks to place on board")
49             .setDefault(2);
50         parser.addArgument("-f")
51             .type(Integer.class)
52             .help("Number of bishops to place on board")
53             .setDefault(2);
54         parser.addArgument("-c")
55             .type(Integer.class)
56             .help("Number of knights to place on board")
57             .setDefault(2);
58         parser.addArgument("--file")
59             .help("File to read the museum from");
```

```

60     parser.epilog("Usage with Gradle: \n$ gradle run -PappArgs=\"['-flag1', '-arg2 x', '
option3']\"");
61     Namespace ns = null;
62     try {
63         ns = parser.parseArgs(args); // Fill the namespace with pass arguments
64     } catch (ArgumentParserException e) {
65         parser.handleError(e);
66         System.exit(1);
67     }
68
69     App app = new App();
70
71     boolean domination = ns.getBoolean("d");
72     boolean minimum_knights = ns.getBoolean("mk");
73     int boardSize = ns.getInt("n");
74     int rook = ns.getInt("t");
75     int bishop = ns.getInt("f");
76     int knight = ns.getInt("c");
77     boolean museum = ns.getBoolean("m");
78     Board chessBoard = new Board(boardSize, rook, bishop, knight);
79     Solution sol = null;
80
81
82     if (museum) {
83         try {
84             FileInputStream input;
85             input = new FileInputStream(ns.getString("file"));
86             chessBoard.createMuseum(input);
87         } catch (FileNotFoundException s) {
88             System.out.println("File does Not Exist Please Try Again: ");
89         }
90         Museum min = new Museum(chessBoard);
91         sol = min.exec();
92         chessBoard.printMuseum(sol);
93     } else if (minimum_knights) {
94         chessBoard.createPotentialKnights();
95         MinimalKnights min = new MinimalKnights(chessBoard);
96         sol = min.exec();
97         chessBoard.printKnightsBoard(sol);
98     } else if (domination) {
99         chessBoard.createPieces();
100        Domination dom = new Domination(chessBoard);
101        sol = dom.exec();
102        chessBoard.printSolutionBoard(sol);
103    } else {
104        chessBoard.createPieces();
105        Independence ind = new Independence(chessBoard);
106        sol = ind.exec();
107        chessBoard.printSolutionBoard(sol);
108    }
109 }
110 }

```

Listing 2 – ../src/main/java/chocolatte/App.java

```

1 package chocolate;
2
3 import java.util.Arrays;
4 import java.util.stream.Stream;
5 import java.util.stream.Collectors;
6 import org.chocosolver.solver.Model;
7 import org.chocosolver.solver.constraints.Constraint;
8 import org.chocosolver.solver.Solver;
9 import org.chocosolver.solver.Solution;
10 import org.chocosolver.solver.search.strategy.Search;
11 import org.chocosolver.solver.variables.IntVar;
12 import org.chocosolver.solver.variables.BoolVar;
13
14 import com.google.common.collect.HashBasedTable;
15 import com.google.common.collect.Table;
16
17 import java.io.*;
18
19
20 public class Board {
21     /* Public attributes, accessible to our solvers */
22     public Model model;
23     public int boardSize;
24     // Number of each piece to place
25     public int rook;
26     public int bishop;
27     public int knight;
28     // Actual problem variables (chess pieces)
29     public IntVar[][] rooks;
30     public IntVar[][] bishops;
31     public IntVar[][] knights;
32     // Knights-to-place
33     public IntVar totalKnights;
34     public BoolVar[][] knightsLocation;
35     // Museum
36     public boolean[][] museumModel;
37     public IntVar[][] museum;
38
39     public void createPieces() {
40         // Creation of rooks
41         this.rooks = new IntVar[rook][2];
42         for (int r = 0; r < rook; r++) {
43             this.rooks[r][0] = this.model.intVar("R_" + r + "_x", 0, boardSize - 1);
44             this.rooks[r][1] = this.model.intVar("R_" + r + "_y", 0, boardSize - 1);
45         }
46
47         // Creation of bishops
48         this.bishops = new IntVar[bishop][2];
49         for (int b = 0; b < bishop; b++) {
50             this.bishops[b][0] = this.model.intVar("B_" + b + "_x", 0, boardSize - 1);
51             this.bishops[b][1] = this.model.intVar("B_" + b + "_y", 0, boardSize - 1);
52         }
53
54         // Creation of knights
55         this.knights = new IntVar[knight][2];
56         for (int k = 0; k < knight; k++) {
57             this.knights[k][0] = this.model.intVar("K_" + k + "_x", 0, boardSize - 1);
58             this.knights[k][1] = this.model.intVar("K_" + k + "_y", 0, boardSize - 1);
59         }
60     }
61
62     public void createPotentialKnights() {
63         this.totalKnights = model.intVar("total", 0, boardSize * boardSize);
64
65         this.knightsLocation = new BoolVar[boardSize][boardSize];

```



```

66         for (int i = 0; i < boardSize; i++) {
67             for (int j = 0; j < boardSize; j++) {
68                 this.knightsLocation[i][j] = model.boolVar("k_" + i + "_" + j);
69             }
70         }
71     }
72
73     public void printSolutionBoard(Solution chocosolution) {
74         if(chocosolution != null) {
75             System.out.println(chocosolution.toString());
76             System.out.println();
77
78             Table<Integer, Integer, String> chessboard = HashBasedTable.create();
79
80             for (int r = 0; r < rook; r++) {
81                 int column = chocosolution.getIntVal(this.rooks[r][0]);
82                 int row = chocosolution.getIntVal(this.rooks[r][1]);
83                 chessboard.put(row, column, "T ");
84             }
85
86             for (int b = 0; b < bishop; b++) {
87                 int column = chocosolution.getIntVal(this.bishops[b][0]);
88                 int row = chocosolution.getIntVal(this.bishops[b][1]);
89                 chessboard.put(row, column, "F ");
90             }
91
92             for (int k = 0; k < knight; k++) {
93                 int column = chocosolution.getIntVal(this.knights[k][0]);
94                 int row = chocosolution.getIntVal(this.knights[k][1]);
95                 chessboard.put(row, column, "C ");
96             }
97
98             // Print based on Table chessboard
99             for (int i = 0; i < boardSize; i++) {
100                 for (int j = 0; j < boardSize; j++) {
101                     if (chessboard.contains(i, j)) {
102                         System.out.print(chessboard.get(i, j));
103                     } else {
104                         System.out.print("* ");
105                     }
106                     if (j == boardSize - 1) {
107                         System.out.println();
108                     }
109                 }
110             }
111         } else {
112             System.out.println("NO SOLUTION TO THE GIVEN PROBLEM");
113         }
114     }
115 }
116
117
118     public void printKnightsBoard(Solution chocosolution) {
119         if(chocosolution != null) {
120             System.out.println(chocosolution.toString());
121             System.out.println();
122
123             int total_knights = chocosolution.getIntVal(totalKnights);
124             System.out.println(total_knights);
125
126             // Print based on Table chessboard
127             for (int i = 0; i < boardSize; i++) {
128                 for (int j = 0; j < boardSize; j++) {
129                     if (chocosolution.getIntVal(this.knightsLocation[i][j]) == 1) {
130                         System.out.print("C ");

```

```

131         } else {
132             System.out.print("* ");
133         }
134         if (j == boardSize - 1) {
135             System.out.println();
136         }
137         ;
138     }
139 }
140 } else {
141     System.out.println("NO SOLUTION TO THE GIVEN PROBLEM");
142 }
143 }
144
145 public void printMuseum(Solution chocosolution) {
146     if(chocosolution != null) {
147         System.out.println(chocosolution.toString());
148         System.out.println();
149
150         // int total_knights = chocosolution.getIntVal(totalKnights);
151         // System.out.println(total_knights);
152
153         // Print based on Table chessboard
154         for (int i = 0; i < boardSize; i++) {
155             for (int j = 0; j < boardSize; j++) {
156                 if (!museumModel[i][j]) {
157                     System.out.print("* ");
158                 } else if (chocosolution.getIntVal(this.museum[i][j]) > 0) {
159                     System.out.print(chocosolution.getIntVal(this.museum[i][j]));
160                     System.out.print(" ");
161                 } else {
162                     System.out.print(" ");
163                 }
164                 if (j == boardSize - 1) {
165                     System.out.println();
166                 }
167                 ;
168             }
169         }
170     } else {
171         System.out.println("NO SOLUTION TO THE GIVEN PROBLEM");
172     }
173 }
174
175 public void createMuseum(FileInputStream file) {
176     // IMPROVE: read file only once - first time is used here to
177     // know the dimension.
178     int size = 0;
179     try {
180         char current = (char) file.read();
181         while (current != '\n') {
182             size++;
183             current = (char) file.read();
184         }
185
186     } catch (IOException e) {
187         e.printStackTrace();
188     }
189
190     boolean[][] arr = new boolean[size][size];
191
192     try {
193         file.getChannel().position(0);
194     } catch (IOException e) {
195         System.out.println(e);

```

```

196     }
197
198
199     try {
200         char current;
201         int row = 0;
202         int column = 0;
203         while (file.available() > 0) {
204             current = (char) file.read();
205             if (current == ' ') {
206                 arr[row][column] = true;
207             }
208             if (current == '\n') {
209                 row++;
210                 column = 0;
211             } else {
212                 column++;
213             }
214         }
215     } catch (IOException e) {
216         e.printStackTrace();
217     }
218
219     this.museumModel = arr;
220
221     this.museum = new IntVar[this.boardSize][this.boardSize];
222     for (int i = 0; i < this.boardSize; ++i) {
223         for (int j = 0; j < this.boardSize; ++j) {
224             this.museum[i][j] = this.model.intVar("m_" + i + "_" + j, 0, 5);
225         }
226     }
227 }
228
229 public Board (int boardSize, int rook, int bishop, int knight) {
230     this.model = new Model(boardSize + "-size chess problem");
231     this.boardSize = boardSize;
232     this.rook = rook;
233     this.bishop = bishop;
234     this.knight = knight;
235 }
236 }

```

Listing 3 – ../src/main/java/chocolate/Board.java

```

1 package chocolate;
2
3 import chocolate.Domination;
4
5 import org.chocosolver.solver.Model;
6 import org.chocosolver.solver.Solver;
7 import org.chocosolver.solver.Solution;
8 import org.chocosolver.solver.search.strategy.Search;
9 import org.chocosolver.solver.variables.IntVar;
10
11 import chocolate.Board;
12
13 public class Independence {
14     private Board board;
15
16     private void rook_constraints(Model model, IntVar[] a, IntVar[] b) {
17         board.model.arithm(a[0], "!=", b[0]).post();
18         board.model.arithm(a[1], "!=", b[1]).post();
19     }
20
21     private void rooks_constraints(Model model, IntVar[] current, IntVar[][] other) {
22         for (int j = 0; j < other.length; j++) {
23             rook_constraints(board.model, current, other[j]);
24         }
25     }
26
27     private void bishop_constraints(Model model, IntVar[] a, IntVar[] b) {
28         IntVar y = a[1].sub(b[1]).abs().intVar();
29         board.model.not(board.model.distance(a[0], b[0], "=", y)).post();
30     }
31
32     private void bishops_constraints(Model model, IntVar[] current, IntVar[][] other) {
33         for (int j = 0; j < other.length; j++) {
34             bishop_constraints(board.model, current, other[j]);
35         }
36     }
37
38     private void knight_case_constraints(Model model, IntVar[] a, IntVar[] b, String o1, int
d1, String o2, int d2) {
39         board.model.or(
40             board.model.arithm(a[0], "!=", b[0], o1, d1),
41             board.model.arithm(a[1], "!=", b[1], o2, d2)
42         ).post();
43     }
44
45     private void knight_constraints(Model model, IntVar[] a, IntVar[] b) {
46         board.model.or(
47             board.model.arithm(a[0], "!=", b[0]),
48             board.model.arithm(a[1], "!=", b[1])
49         ).post();
50
51         knight_case_constraints(model, a, b, "+", 2, "+", 1);
52         knight_case_constraints(model, a, b, "+", 2, "-", 1);
53         knight_case_constraints(model, a, b, "-", 2, "-", 1);
54         knight_case_constraints(model, a, b, "-", 2, "+", 1);
55
56         knight_case_constraints(model, a, b, "+", 1, "+", 2);
57         knight_case_constraints(model, a, b, "+", 1, "-", 2);
58         knight_case_constraints(model, a, b, "-", 1, "-", 2);
59         knight_case_constraints(model, a, b, "-", 1, "+", 2);
60     }
61
62     private void knights_constraints(Model model, IntVar[] current, IntVar[][] other) {
63         for (int j = 0; j < other.length; j++) {
64             knight_constraints(board.model, current, other[j]);
65         }
66     }

```

```

65     }
66 }
67
68
69 public Solution exec() {
70     int rook = this.board.rook;
71     int bishop = this.board.bishop;
72     int knight = this.board.knight;
73     // Conditions on rooks
74     for (int i = 0; i < rook; i++) {
75         for (int j = i + 1; j < rook; j++) {
76             rook_constraints(board.model, board.rooks[i], board.rooks[j]);
77         }
78
79         rooks_constraints(board.model, board.rooks[i], board.bishops);
80         rooks_constraints(board.model, board.rooks[i], board.knights);
81     }
82
83     // Conditions on bishops
84     for (int i = 0; i < bishop; i++) {
85         for (int j = i + 1; j < bishop; j++) {
86             bishop_constraints(board.model, board.bishops[i], board.bishops[j]);
87         }
88
89         bishops_constraints(board.model, board.bishops[i], board.rooks);
90         bishops_constraints(board.model, board.bishops[i], board.knights);
91     }
92
93     // Conditions on knights
94     for (int i = 0; i < knight; i++) {
95         for (int j = i + 1; j < knight; j++) {
96             knight_constraints(board.model, board.knights[i], board.knights[j]);
97         }
98
99         knights_constraints(board.model, board.knights[i], board.rooks);
100        knights_constraints(board.model, board.knights[i], board.bishops);
101    }
102
103    return board.model.getSolver().findSolution();
104 }
105
106 public Independence (Board board) {
107     this.board = board;
108 }
109
110 }

```

Listing 4 – ../src/main/java/chocolatte/Independence.java

```

1 package chocolate;
2
3 import java.util.Arrays;
4 import java.util.stream.Stream;
5 import java.util.stream.Collectors;
6 import org.chocosolver.solver.Model;
7 import org.chocosolver.solver.constraints.Constraint;
8 import org.chocosolver.solver.Solver;
9 import org.chocosolver.solver.Solution;
10 import org.chocosolver.solver.search.strategy.Search;
11 import org.chocosolver.solver.variables.IntVar;
12
13 import chocolate.Board;
14
15 public class Domination {
16     private Board board;
17
18     private Constraint rook_constraints(IntVar[] a, IntVar[] b) {
19         // Ensure the rook is not on the same case as the other piece.
20         board.model.or(
21             board.model.arithm(a[0], "!=", b[0]),
22             board.model.arithm(a[1], "!=", b[1])
23         ).post();
24
25         return board.model.or(
26             board.model.arithm(a[0], "=", b[0]),
27             board.model.arithm(a[1], "=", b[1])
28         );
29     }
30
31     private Constraint rooks_constraints(IntVar[] current, IntVar[][] other) {
32         if (other.length > 0) {
33             Constraint[] constraints = Arrays.stream(other)
34                 .map(x -> rook_constraints(current, x))
35                 .toArray(size -> new Constraint[size]);
36
37             return board.model.or(constraints);
38         }
39
40         return board.model.falseConstraint();
41     }
42
43     private Constraint bishop_constraints(IntVar[] a, IntVar[] b) {
44         board.model.or(
45             board.model.arithm(a[0], "!=", b[0]),
46             board.model.arithm(a[1], "!=", b[1])
47         ).post();
48
49         IntVar y = a[1].sub(b[1]).abs().intVar();
50         return board.model.distance(a[0], b[0], "=", y);
51     }
52
53     private Constraint bishops_constraints(IntVar[] current, IntVar[][] other) {
54         if (other.length > 0) {
55             Constraint[] constraints = Arrays.stream(other)
56                 .map(x -> bishop_constraints(current, x))
57                 .toArray(size -> new Constraint[size]);
58
59             return board.model.or(constraints);
60         }
61
62         return board.model.falseConstraint();
63     }
64
65     private Constraint knight_case_constraints(IntVar[] a, IntVar[] b, String o1, int d1,

```

```

String o2, int d2) {
66     return board.model.and(
67         board.model.arithm(a[0], "=", b[0], o1, d1),
68         board.model.arithm(a[1], "=", b[1], o2, d2)
69     );
70 }
71
72 private Constraint knight_constraints(IntVar[] a, IntVar[] b) {
73     board.model.or(
74         board.model.arithm(a[0], "!=", b[0]),
75         board.model.arithm(a[1], "!=", b[1])
76     ).post();
77
78     return board.model.or(
79         knight_case_constraints(a, b, "+", 2, "+", 1),
80         knight_case_constraints(a, b, "+", 2, "-", 1),
81         knight_case_constraints(a, b, "-", 2, "-", 1),
82         knight_case_constraints(a, b, "-", 2, "+", 1),
83         knight_case_constraints(a, b, "+", 1, "+", 2),
84         knight_case_constraints(a, b, "+", 1, "-", 2),
85         knight_case_constraints(a, b, "-", 1, "-", 2),
86         knight_case_constraints(a, b, "-", 1, "+", 2)
87     );
88 }
89
90 private Constraint knights_constraints(IntVar[] current, IntVar[][] other) {
91     if (other.length > 0) {
92         Constraint[] constraints = Arrays.stream(other)
93             .map(x -> knight_constraints(current, x))
94             .toArray(size -> new Constraint[size]);
95
96         return board.model.or(constraints);
97     }
98
99     return board.model.falseConstraint();
100 }
101
102 public Solution exec() {
103     for (int i = 0; i < board.rooks.length; i++) {
104         if (i + 1 < board.rooks.length) {
105             board.model.or(
106                 rooks_constraints(board.rooks[i], Arrays.copyOfRange(board.rooks, i + 1,
107                     board.rooks.length)),
108                 rooks_constraints(board.rooks[i], board.bishops),
109                 rooks_constraints(board.rooks[i], board.knights)
110             ).post();
111         } else if (board.knights.length > 0 || board.bishops.length > 0){
112             board.model.or(
113                 rooks_constraints(board.rooks[i], board.bishops),
114                 rooks_constraints(board.rooks[i], board.knights)
115             ).post();
116         }
117
118         for (int i = 0; i < board.bishops.length; i++) {
119             if (i + 1 < board.bishops.length) {
120                 board.model.or(
121                     bishops_constraints(board.bishops[i], Arrays.copyOfRange(board.bishops,
122                         i + 1, board.bishops.length)),
123                     bishops_constraints(board.bishops[i], board.rooks),
124                     bishops_constraints(board.bishops[i], board.knights)
125                 ).post();
126             } else if (board.knights.length > 0 || board.rooks.length > 0){
127                 board.model.or(
128                     bishops_constraints(board.bishops[i], board.rooks),

```

```

128         bishops_constraints(board.bishops[i], board.knights)
129     ).post();
130 }
131 }
132
133     for (int i = 0; i < board.knights.length; i++) {
134         if (i + 1 < board.knights.length) {
135             board.model.or(
136                 knights_constraints(board.knights[i], Arrays.copyOfRange(board.knights,
137 i + 1, board.knights.length)),
138                 knights_constraints(board.knights[i], board.rooks),
139                 knights_constraints(board.knights[i], board.bishops)
140             ).post();
141         } else if (board.bishops.length > 0 || board.rooks.length > 0){
142             board.model.or(
143                 knights_constraints(board.knights[i], board.rooks),
144                 knights_constraints(board.knights[i], board.bishops)
145             ).post();
146         }
147     }
148     return board.model.getSolver().findSolution();
149 }
150
151     public Domination(Board board) {
152         this.board = board;
153     }
154 }

```

Listing 5 – ../src/main/java/chocolatte/Domination.java



```

1 package chocolate;
2
3 import java.util.Arrays;
4 import java.util.stream.Stream;
5 import java.util.stream.Collectors;
6 import org.chocosolver.solver.Model;
7 import org.chocosolver.solver.constraints.Constraint;
8 import org.chocosolver.solver.Solver;
9 import org.chocosolver.solver.Solution;
10 import org.chocosolver.solver.search.strategy.Search;
11 import org.chocosolver.solver.variables.IntVar;
12 import org.chocosolver.solver.variables.BoolVar;
13
14 public class MinimalKnights {
15     private Board board;
16
17     private BoolVar knight_case_constraints(int i, int j, int d1, int d2) {
18         if ((i + d1 >= 0 && i + d1 < board.boardSize)
19             && (j + d2 >= 0 && j + d2 < board.boardSize)) {
20             return board.knightsLocation[i + d1][j + d2];
21         } else {
22             return board.knightsLocation[i][j];
23         }
24     }
25
26     public Solution exec() {
27         for (int i = 0; i < board.boardSize; ++i) {
28             for (int j = 0; j < board.boardSize; ++j) {
29                 board.model.or(
30                     board.knightsLocation[i][j],
31                     knight_case_constraints(i, j, 2, 1),
32                     knight_case_constraints(i, j, 2, -1),
33                     knight_case_constraints(i, j, -2, -1),
34                     knight_case_constraints(i, j, -2, 1),
35                     knight_case_constraints(i, j, 1, 2),
36                     knight_case_constraints(i, j, 1, -2),
37                     knight_case_constraints(i, j, -1, -2),
38                     knight_case_constraints(i, j, -1, 2)
39                 ).post();
40             }
41         }
42
43         // Flatten matrix to list.
44         BoolVar[] vars = Arrays.stream(board.knightsLocation)
45             .flatMap(listContainer -> Arrays.stream(listContainer))
46             .toArray(size -> new BoolVar[size]);
47
48         board.model.sum(vars, "+", board.totalKnights).post();
49         return board.model.getSolver().findOptimalSolution(board.totalKnights, Model.
50             MINIMIZE);
51     }
52
53     public MinimalKnights(Board board) {
54         this.board = board;
55     }
56 }

```

Listing 6 – ../src/main/java/chocolate/MinimalKnights.java

```

1 package chocolate;
2
3 import chocolate.MuseumTypes;
4 import java.util.Arrays;
5 import java.util.ArrayList;
6 import java.util.stream.Stream;
7 import java.util.stream.Collectors;
8 import org.chocosolver.solver.Model;
9 import org.chocosolver.solver.constraints.Constraint;
10 import org.chocosolver.solver.Solver;
11 import org.chocosolver.solver.Solution;
12 import org.chocosolver.solver.search.strategy.Search;
13 import org.chocosolver.solver.variables.IntVar;
14 import org.chocosolver.solver.variables.BoolVar;
15
16 public class Museum {
17     static private MuseumTypes WALL = MuseumTypes.WALL;
18     static private MuseumTypes EMPTY = MuseumTypes.EMPTY;
19     static private MuseumTypes OUEST = MuseumTypes.OUEST;
20     static private MuseumTypes EST = MuseumTypes.EST;
21     static private MuseumTypes NORD = MuseumTypes.NORD;
22     static private MuseumTypes SUD = MuseumTypes.SUD;
23
24     private Board board;
25
26     private IntVar[] get_ouest(int x, int y) {
27         ArrayList<IntVar> result = new ArrayList<IntVar>();
28
29         for (int i = x; i >= 0; --i) {
30             if (!board.museumModel[y][i]) {
31                 break;
32             } else {
33                 result.add(board.museum[y][i]);
34             }
35         }
36
37         System.out.println(result);
38         return result.stream().toArray(size -> new IntVar[size]);
39     }
40
41     private Constraint get_constraint_ouest(int x, int y) {
42         int[] values = {EST.getValue()};
43         IntVar nbVar = board.model.intVar("constraint_ouest_camera_" + x + "_" + y, 1, 1);
44         IntVar[] fields = get_ouest(x, y);
45         return board.model.among(nbVar, fields, values);
46     }
47
48     private IntVar[] get_est(int x, int y) {
49         ArrayList<IntVar> result = new ArrayList<IntVar>();
50
51         for (int i = x; i < board.boardSize; ++i) {
52             if (!board.museumModel[y][i]) {
53                 break;
54             } else {
55                 result.add(board.museum[y][i]);
56             }
57         }
58
59         return result.stream().toArray(size -> new IntVar[size]);
60     }
61
62     private Constraint get_constraint_est(int x, int y) {
63         int[] values = {OUEST.getValue()};
64         IntVar nbVar = board.model.intVar("constraint_est_camera_" + x + "_" + y, 1, 1);
65         IntVar[] fields = get_est(x, y);

```

```

66         return board.model.among(nbVar, fields, values);
67     }
68
69     private IntVar[] get_nord(int x, int y) {
70         ArrayList<IntVar> result = new ArrayList<IntVar>();
71
72         for (int i = x; i >= 0; --i) {
73             if (!board.museumModel[i][x]) {
74                 break;
75             } else {
76                 result.add(board.museum[i][x]);
77             }
78         }
79
80         return result.stream().toArray(size -> new IntVar[size]);
81     }
82
83     private Constraint get_constraint_nord(int x, int y) {
84         int[] values = {SUD.getValue()};
85         IntVar nbVar = board.model.intVar("constraint_nord_camera_" + x + "_" + y, 1, 1);
86         IntVar[] fields = get_nord(x, y);
87         return board.model.among(nbVar, fields, values);
88     }
89
90     private IntVar[] get_sud(int x, int y) {
91         ArrayList<IntVar> result = new ArrayList<IntVar>();
92
93         for (int i = x; i < board.boardSize; ++i) {
94             if (!board.museumModel[i][x]) {
95                 break;
96             } else {
97                 result.add(board.museum[i][x]);
98             }
99         }
100
101         return result.stream().toArray(size -> new IntVar[size]);
102     }
103
104     private Constraint get_constraint_sud(int x, int y) {
105         int[] values = {NORD.getValue()};
106         IntVar nbVar = board.model.intVar("constraint_sud_camera_" + x + "_" + y, 1, 1);
107         IntVar[] fields = get_sud(x, y);
108         return board.model.among(nbVar, fields, values);
109     }
110
111     public Solution exec() {
112         for (int i = 0; i < board.boardSize; ++i) {
113             for (int j = 0; j < board.boardSize; ++j) {
114                 if (board.museumModel[j][i]) {
115                     board.model.or(
116                         get_constraint_ouest(i, j),
117                         get_constraint_est(i, j),
118                         get_constraint_nord(i, j),
119                         get_constraint_sud(i, j)
120                     ).post();
121                 } else {
122                     board.model.arithm(board.museum[j][i], "=", WALL.getValue()).post();
123                 }
124             }
125         }
126
127         // Sum ouest camera + est + nord + sud
128
129         IntVar minimize = board.model.intVar("minimizing", 0, board.boardSize * board.
boardSize);

```

```

130     // Flatten matrix to list.
131     IntVar[] vars = Arrays.stream(board.museum)
132         .flatMap(listContainer -> Arrays.stream(listContainer))
133         .toArray(size -> new IntVar[size]);
134
135     board.model.sum(vars, "+", minimize).post();
136     // return board.model.getSolver().findSolution();
137     return board.model.getSolver().findOptimalSolution(minimize, Model.MINIMIZE);
138 }
139
140 public Museum(Board board) {
141     this.board = board;
142 }
143 }

```

Listing 7 – ../src/main/java/chocolate/Museum.java