UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

INFO-F-302 Informatique Fondamentale : Rapport de projet

Perale Thomas
Requena Carlos

Année académique 2016 - 2017

Table des matières

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`[1]. Pour celà 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 misent 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, coordonée$_x$, coordonée$_y$) :

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

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

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

2.3 Contraintes

— Chaque pièce doit occuper une case diffèrente.

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

— 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

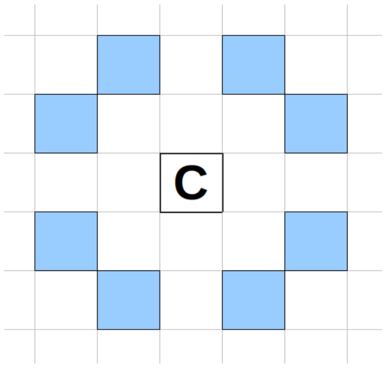
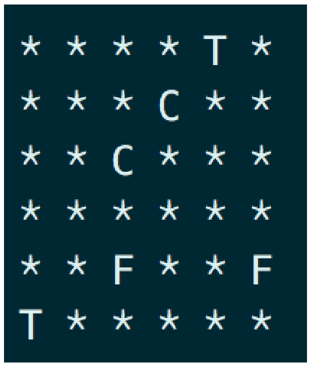


FIGURE 1 – Cases menacées par un cavalier



```
* * * * T *
* * * C * *
* * C * * *
* * * * * *
* * F * * F
T * * * * *
```

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

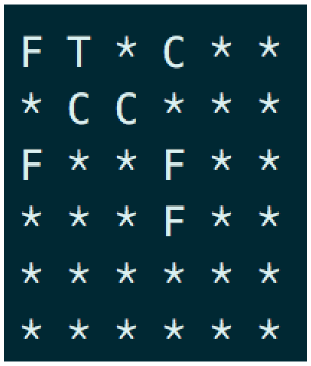
— Chaque pièce doit occuper une case diffèrente.
— Toute pièce doit occuper soit :
  — les cases situées à gauche, à droite, en haut et en bas des toutes les tours.
  — les cases situées en diagonale des tous les fous.
  — les cases menacées par tous les cavaliers (2.3).

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 a savoir si une solution existe, mais plutôt a connaître la quantité minimale de chevaliers nécessaires pour menacer tout l'échiquier :



```
F T * C * *
* C C * * *
F * * F * *
* * * F * *
* * * * * *
* * * * * *
```

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

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

Listing 1 –

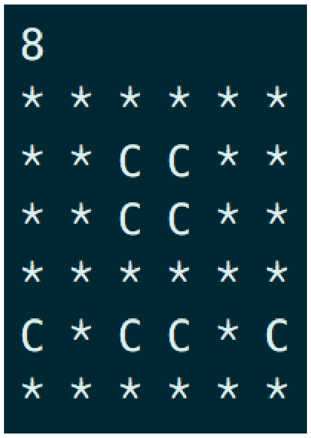
4.2 Définition des variables

— n la taille de l'échiquier ;

4.3 Contraintes

— Chaque pièce doit occuper une case diffèrente. Chaque chevalier 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.



```
8
* * * * * *
* * C C * *
* * C C * *
* * * * * *
C * C C * C
* * * * * *
```

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

```java
package chocolatte;

import chocolatte.Domination;
import chocolatte.Museum;

import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.variables.IntVar;

import com.google.common.collect.HashBasedTable;
import com.google.common.collect.Table;

import java.io.*;


// Argparse
import net.sourceforge.argparse4j.ArgumentParsers;
import net.sourceforge.argparse4j.inf.ArgumentParser;
import net.sourceforge.argparse4j.inf.ArgumentParserException;
import net.sourceforge.argparse4j.inf.Namespace;
import net.sourceforge.argparse4j.inf.MutuallyExclusiveGroup;
import net.sourceforge.argparse4j.impl.Arguments;


public class App {

    public static void main(String[] args) {
        ArgumentParser parser = ArgumentParsers.newArgumentParser("Chess pieces parser")
            .defaultHelp(true)
            .description("Returns a solution for the independence or domination problem for
    the given pieces (only rooks, knights and bishops)");
        MutuallyExclusiveGroup indom = parser.addMutuallyExclusiveGroup("Independence/
    Domination");
        indom.addArgument("-i").action(Arguments.storeTrue())
            .help("Solve the independence problem");
        indom.addArgument("-d").action(Arguments.storeTrue())
            .help("Solve the domination problem");
        indom.addArgument("-mk").action(Arguments.storeTrue())
            .help("Solve the minimum knights problem");
        indom.addArgument("-m").action(Arguments.storeTrue())
            .help("Solve the museum problem");
        parser.addArgument("-n")
            .type(Integer.class)
            .help("Generate a chess board NxN")
            .setDefault(5);
        parser.addArgument("-t")
            .type(Integer.class)
            .help("Number of rooks to place on board")
            .setDefault(2);
        parser.addArgument("-f")
            .type(Integer.class)
            .help("Number of bishops to place on board")
            .setDefault(2);
        parser.addArgument("-c")
            .type(Integer.class)
            .help("Number of knights to place on board")
            .setDefault(2);
        parser.addArgument("--file")
            .help("File to read the museum from");
        parser.epilog("Usage with Gradle: \n$ gradle run -PappArgs=\"['-flag1', '-arg2 x', 'option3']\"");
        Namespace ns = null;
        try {
            ns = parser.parseArgs(args); // Fill the namespace with pass arguments
        } catch (ArgumentParserException e) {
            parser.handleError(e);
            System.exit(1);
        }

        App app = new App();

        boolean domination = ns.getBoolean("d");
        boolean minimum_knights = ns.getBoolean("mk");
        int boardSize = ns.getInt("n");
        int rook = ns.getInt("t");
        int bishop = ns.getInt("f");
        int knight = ns.getInt("c");
        boolean museum = ns.getBoolean("m");
        Board chessBoard = new Board(boardSize, rook, bishop, knight);
        Solution sol = null;


        if (museum) {
            try {
                FileInputStream input;
                input = new FileInputStream(ns.getString("file"));
                chessBoard.createMuseum(input);
            } catch(FileNotFoundException s) {
                System.out.println("File does Not Exist Please Try Again: ");
            }
            Museum min = new Museum(chessBoard);
            sol = min.exec();
            chessBoard.printMuseum(sol);
        } else if (minimum_knights) {
            chessBoard.createPotentialKnights();
            MinimalKnights min = new MinimalKnights(chessBoard);
            sol = min.exec();
            chessBoard.printKnightsBoard(sol);
        } else if (domination) {
            chessBoard.createPieces();
            Domination dom = new Domination(chessBoard);
            sol = dom.exec();
            chessBoard.printSolutionBoard(sol);
        } else {
            chessBoard.createPieces();
            Independence ind = new Independence(chessBoard);
            sol = ind.exec();
            chessBoard.printSolutionBoard(sol);
        }
    }
}
```

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

```java
package chocolatte;

import java.util.Arrays;
import java.util.stream.Stream;
import java.util.stream.Collector;
import org.chocosolver.solver.Model;
import org.chocosolver.solver.constraints.Constraint;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.BoolVar;

import com.google.common.collect.HashBasedTable;
import com.google.common.collect.Table;

import java.io.*;


public class Board {
    /* Public attributes, accessible to our solvers */
    public Model model;
    public int boardSize;
    // Number of each piece to place
    public int rook;
    public int bishop;
    public int knight;
    // Actual problem variables (chess pieces)
    public IntVar[][] rooks;
    public IntVar[][] bishops;
    public IntVar[][] knights;
    // Knights-to-place
    public IntVar totalKnights;
    public BoolVar[][] knightsLocation;
    // Museum
    public boolean[][] museumModel;
    public IntVar[][] museum;

    public void createPieces() {
        // Creation of rooks
        this.rooks = new IntVar[rook][2];
        for (int r = 0; r < rook; r++) {
            this.rooks[r][0] = this.model.intVar("R_" + r + "_x", 0, boardSize - 1);
            this.rooks[r][1] = this.model.intVar("R_" + r + "_y", 0, boardSize - 1);
        }

        // Creation of bishops
        this.bishops = new IntVar[bishop][2];
        for (int b = 0; b < bishop; b++) {
            this.bishops[b][0] = this.model.intVar("B_" + b + "_x", 0, boardSize - 1);
            this.bishops[b][1] = this.model.intVar("B_" + b + "_y", 0, boardSize - 1);
        }

        // Creation of knights
        this.knights = new IntVar[knight][2];
        for (int k = 0; k < knight; k++) {
            this.knights[k][0] = this.model.intVar("K_" + k + "_x", 0, boardSize - 1);
            this.knights[k][1] = this.model.intVar("K_" + k + "_y", 0, boardSize - 1);
        }
    }

    public void createPotentialKnights() {
        this.totalKnights = model.intVar("total", 0, boardSize * boardSize);

        this.knightsLocation = new BoolVar[boardSize][boardSize];
        for (int i = 0; i < boardSize; i++) {
            for (int j = 0; j < boardSize; j++) {
                this.knightsLocation[i][j] = model.boolVar("k_" + i + "_" + j);
            }
        }
    }

    public void printSolutionBoard(Solution chocosolution) {
        if(chocosolution != null) {
            System.out.println(chocosolution.toString());
            System.out.println();

            Table<Integer, Integer, String> chessboard = HashBasedTable.create();

            for (int r = 0; r < rook; r++) {
                int column = chocosolution.getIntVal(this.rooks[r][0]);
                int row = chocosolution.getIntVal(this.rooks[r][1]);
                chessboard.put(row, column, "T ");
            }

            for (int b = 0; b < bishop; b++) {
                int column = chocosolution.getIntVal(this.bishops[b][0]);
                int row = chocosolution.getIntVal(this.bishops[b][1]);
                chessboard.put(row, column, "F ");

            }

            for (int k = 0; k < knight; k++) {
                int column = chocosolution.getIntVal(this.knights[k][0]);
                int row = chocosolution.getIntVal(this.knights[k][1]);
                chessboard.put(row, column, "C ");
            }

            // Print based on Table chessboard
            for (int i = 0; i < boardSize; i++) {
                for (int j = 0; j < boardSize; j++) {

                    if (chessboard.contains(i, j)) {
                        System.out.print(chessboard.get(i, j));
                    } else {
                        System.out.print("* ");
                    }
                    if (j == boardSize - 1) {
                        System.out.println();
                    }
                }
            }
        } else {
            System.out.println("NO SOLUTION TO THE GIVEN PROBLEM");
        }
    }

    public void printKnightsBoard(Solution chocosolution) {
        if(chocosolution != null) {
            System.out.println(chocosolution.toString());
            System.out.println();

            int total_knights = chocosolution.getIntVal(totalKnights);
            System.out.println(total_knights);

            // Print based on Table chessboard
            for (int i = 0; i < boardSize; i++) {
                for (int j = 0; j < boardSize; j++) {
                    if (chocosolution.getIntVal(this.knightsLocation[i][j]) == 1) {
                        System.out.print("C ");
                    } else {
                        System.out.print("* ");
                    }
                    if (j == boardSize - 1) {
                        System.out.println();
                    }
                    ;
                }
            }
        } else {
            System.out.println("NO SOLUTION TO THE GIVEN PROBLEM");
        }
    }

    public void printMuseum(Solution chocosolution) {
        if(chocosolution != null) {
            System.out.println(chocosolution.toString());
            System.out.println();

            // int total_knights = chocosolution.getIntVal(totalKnights);
            // System.out.println(total_knights);

            // Print based on Table chessboard
            for (int i = 0; i < boardSize; i++) {
                for (int j = 0; j < boardSize; j++) {
                    if (!museumModel[i][j]) {
                        System.out.print("* ");
                    } else if (chocosolution.getIntVal(this.museum[i][j]) > 0) {
                        System.out.print(chocosolution.getIntVal(this.museum[i][j]));
                        System.out.print(" ");
                    } else {
                        System.out.print("  ");
                    }
                    if (j == boardSize - 1) {
                        System.out.println();
                    }
                    ;
                }
            }
        } else {
            System.out.println("NO SOLUTION TO THE GIVEN PROBLEM");
        }
    }

    public void createMuseum(FileInputStream file) {
        // IMPROVE: read file only once - first time is used here to
        // know the dimension.
        int size = 0;
        try {
            char current = (char) file.read();
            while (current != '\n') {
                size++;
                current = (char) file.read();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }

        boolean[][] arr = new boolean[size][size];

        try {
            file.getChannel().position(0);
        } catch (IOException e) {
            System.out.println(e);

        }

        try {
            char current;
            int row = 0;
            int column = 0;
            while (file.available() > 0) {
                current = (char) file.read();
                if (current == ' ') {
                    arr[row][column] = true;
                }
                if (current == '\n') {
                    row++;
                    column = 0;
                } else {
                    column++;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        this.museumModel = arr;

        this.museum = new IntVar[this.boardSize][this.boardSize];
        for (int i = 0; i < this.boardSize; ++i) {
            for (int j = 0; j < this.boardSize; ++j) {
                this.museum[i][j] = this.model.intVar("m_" + i + "_" + j, 0, 5);
            }
        }
    }

    public Board (int boardSize, int rook, int bishop, int knight) {
        this.model = new Model(boardSize + "-size chess problem");
        this.boardSize = boardSize;
        this.rook = rook;
        this.bishop = bishop;
        this.knight = knight;
    }
}
```

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

```java
package chocolatte;

import chocolatte.Domination;

import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.variables.IntVar;

import chocolatte.Board;

public class Independence {
    private Board board;

    private void rook_constraints(Model model, IntVar[] a, IntVar[] b) {
        board.model.arithm(a[0], "!=", b[0]).post();
        board.model.arithm(a[1], "!=", b[1]).post();
    }

    private void rooks_constraints(Model model, IntVar[] current, IntVar[][] other) {
        for (int j = 0; j < other.length; j++) {
            rook_constraints(board.model, current, other[j]);
        }
    }

    private void bishop_constraints(Model model, IntVar[] a, IntVar[] b) {
        IntVar y = a[1].sub(b[1]).abs().intVar();
        board.model.not(board.model.distance(a[0], b[0], "=", y)).post();
    }

    private void bishops_constraints(Model model, IntVar[] current, IntVar[][] other) {
        for (int j = 0; j < other.length; j++) {
            bishop_constraints(board.model, current, other[j]);
        }
    }

    private void knight_case_constraints(Model model, IntVar[] a, IntVar[] b, String o1, int
     d1, String o2, int d2) {
        board.model.or(
            board.model.arithm(a[0], "!=", b[0], o1, d1),
            board.model.arithm(a[1], "!=", b[1], o2, d2)
            ).post();
    }

    private void knight_constraints(Model model, IntVar[] a, IntVar[] b) {
        board.model.or(
            board.model.arithm(a[0], "!=", b[0]),
            board.model.arithm(a[1], "!=", b[1])
            ).post();

        knight_case_constraints(model, a, b, "+", 2, "+", 1);
        knight_case_constraints(model, a, b, "+", 2, "-", 1);
        knight_case_constraints(model, a, b, "-", 2, "-", 1);
        knight_case_constraints(model, a, b, "-", 2, "+", 1);

        knight_case_constraints(model, a, b, "+", 1, "+", 2);
        knight_case_constraints(model, a, b, "+", 1, "-", 2);
        knight_case_constraints(model, a, b, "-", 1, "-", 2);
        knight_case_constraints(model, a, b, "-", 1, "+", 2);
    }

    private void knights_constraints(Model model, IntVar[] current, IntVar[][] other) {
        for (int j = 0; j < other.length; j++) {
            knight_constraints(board.model, current, other[j]);
        }
    }


    public Solution exec() {
        int rook = this.board.rook;
        int bishop = this.board.bishop;
        int knight = this.board.knight;
        // Conditions on rooks
        for (int i = 0; i < rook; i++) {
            for (int j = i + 1; j < rook; j++) {
                rook_constraints(board.model, board.rooks[i], board.rooks[j]);
            }

            rooks_constraints(board.model, board.rooks[i], board.bishops);
            rooks_constraints(board.model, board.rooks[i], board.knights);
        }

        // Conditions on bishops
        for (int i = 0; i < bishop; i++) {
            for (int j = i + 1; j < bishop; j++) {
                bishop_constraints(board.model, board.bishops[i], board.bishops[j]);
            }

            bishops_constraints(board.model, board.bishops[i], board.rooks);
            bishops_constraints(board.model, board.bishops[i], board.knights);
        }

        // Conditions on knights
        for (int i = 0; i < knight; i++) {
            for (int j = i + 1; j < knight; j++) {
                knight_constraints(board.model, board.knights[i], board.knights[j]);
            }

            knights_constraints(board.model, board.knights[i], board.rooks);
            knights_constraints(board.model, board.knights[i], board.bishops);
        }

        return board.model.getSolver().findSolution();
    }

    public Independence (Board board) {
        this.board = board;
    }

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

```java
package chocolatte;

import java.util.Arrays;
import java.util.stream.Stream;
import java.util.stream.Collector;
import org.chocosolver.solver.Model;
import org.chocosolver.solver.constraints.Constraint;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.variables.IntVar;

import chocolatte.Board;

public class Domination {
    private Board board;

    private Constraint rook_constraints(IntVar[] a, IntVar[] b) {
        // Ensure the rook is not on the same case as the other piece.
        board.model.or(
            board.model.arithm(a[0], "!=", b[0]),
            board.model.arithm(a[1], "!=", b[1])
            ).post();

        return board.model.or(
            board.model.arithm(a[0], "=", b[0]),
            board.model.arithm(a[1], "=", b[1])
            );
    }

    private Constraint rooks_constraints(IntVar[] current, IntVar[][] other) {
        if (other.length > 0) {
            Constraint[] constraints = Arrays.stream(other)
                .map(x -> rook_constraints(current, x))
                .toArray(size -> new Constraint[size]);

            return board.model.or(constraints);
        }

        return board.model.falseConstraint();
    }

    private Constraint bishop_constraints(IntVar[] a, IntVar[] b) {
        board.model.or(
            board.model.arithm(a[0], "!=", b[0]),
            board.model.arithm(a[1], "!=", b[1])
            ).post();

        IntVar y = a[1].sub(b[1]).abs().intVar();
        return board.model.distance(a[0], b[0], "=", y);
    }

    private Constraint bishops_constraints(IntVar[] current, IntVar[][] other) {
        if (other.length > 0) {
            Constraint[] constraints = Arrays.stream(other)
                .map(x -> bishop_constraints(current, x))
                .toArray(size -> new Constraint[size]);

            return board.model.or(constraints);
        }

        return board.model.falseConstraint();
    }

    private Constraint knight_case_constraints(IntVar[] a, IntVar[] b, String o1, int d1,
    String o2, int d2) {
        return board.model.and(
            board.model.arithm(a[0], "=", b[0], o1, d1),
            board.model.arithm(a[1], "=", b[1], o2, d2)
            );
    }

    private Constraint knight_constraints(IntVar[] a, IntVar[] b) {
        board.model.or(
            board.model.arithm(a[0], "!=", b[0]),
            board.model.arithm(a[1], "!=", b[1])
            ).post();

        return board.model.or(
            knight_case_constraints(a, b, "+", 2, "+", 1),
            knight_case_constraints(a, b, "+", 2, "-", 1),
            knight_case_constraints(a, b, "-", 2, "-", 1),
            knight_case_constraints(a, b, "-", 2, "+", 1),
            knight_case_constraints(a, b, "+", 1, "+", 2),
            knight_case_constraints(a, b, "+", 1, "-", 2),
            knight_case_constraints(a, b, "-", 1, "-", 2),
            knight_case_constraints(a, b, "-", 1, "+", 2)
            );
    }

    private Constraint knights_constraints(IntVar[] current, IntVar[][] other) {
        if (other.length > 0) {
            Constraint[] constraints = Arrays.stream(other)
                .map(x -> knight_constraints(current, x))
                .toArray(size -> new Constraint[size]);

            return board.model.or(constraints);
        }

        return board.model.falseConstraint();
    }

    public Solution exec() {
        for (int i = 0; i < board.rooks.length; i++) {
            if (i + 1 < board.rooks.length) {
                board.model.or(
                    rooks_constraints(board.rooks[i], Arrays.copyOfRange(board.rooks, i + 1,
     board.rooks.length)),
                    rooks_constraints(board.rooks[i], board.bishops),
                    rooks_constraints(board.rooks[i], board.knights)
                    ).post();
            } else if (board.knights.length > 0 || board.bishops.length > 0){
                board.model.or(
                    rooks_constraints(board.rooks[i], board.bishops),
                    rooks_constraints(board.rooks[i], board.knights)
                    ).post();
            }
        }

        for (int i = 0; i < board.bishops.length; i++) {
            if (i + 1 < board.bishops.length) {
                board.model.or(
                    bishops_constraints(board.bishops[i], Arrays.copyOfRange(board.bishops,
    i + 1, board.bishops.length)),
                    bishops_constraints(board.bishops[i], board.rooks),
                    bishops_constraints(board.bishops[i], board.knights)
                    ).post();
            } else if (board.knights.length > 0 || board.rooks.length > 0){
                board.model.or(
                    bishops_constraints(board.bishops[i], board.rooks),
                    bishops_constraints(board.bishops[i], board.knights)
                    ).post();
            }
        }

        for (int i = 0; i < board.knights.length; i++) {
            if (i + 1 < board.knights.length) {
                board.model.or(
                    knights_constraints(board.knights[i], Arrays.copyOfRange(board.knights,
    i + 1, board.knights.length)),
                    knights_constraints(board.knights[i], board.rooks),
                    knights_constraints(board.knights[i], board.bishops)
                    ).post();
            } else if (board.bishops.length > 0 || board.rooks.length > 0){
                board.model.or(
                    knights_constraints(board.knights[i], board.rooks),
                    knights_constraints(board.knights[i], board.bishops)
                    ).post();
            }
        }

        return board.model.getSolver().findSolution();
    }

    public Domination(Board board) {
        this.board = board;
    }
}
```

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

```java
package chocolatte;

import java.util.Arrays;
import java.util.stream.Stream;
import java.util.stream.Collector;
import org.chocosolver.solver.Model;
import org.chocosolver.solver.constraints.Constraint;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.BoolVar;

public class MinimalKnights {
    private Board board;

    private BoolVar knight_case_constraints(int i, int j, int d1, int d2) {
        if ((i + d1 >= 0 && i + d1 < board.boardSize)
                && (j + d2 >= 0 && j + d2 < board.boardSize)) {
            return board.knightsLocation[i + d1][j + d2];
        } else {
            return board.knightsLocation[i][j];
        }
    }

    public Solution exec() {
        for (int i = 0; i < board.boardSize; ++i) {
            for (int j = 0; j < board.boardSize; ++j) {
                board.model.or(
                    board.knightsLocation[i][j],
                    knight_case_constraints(i, j, 2, 1),
                    knight_case_constraints(i, j, 2, -1),
                    knight_case_constraints(i, j, -2, -1),
                    knight_case_constraints(i, j, -2, 1),
                    knight_case_constraints(i, j, 1, 2),
                    knight_case_constraints(i, j, 1, -2),
                    knight_case_constraints(i, j, -1, -2),
                    knight_case_constraints(i, j, -1, 2)
                ).post();
            }
        }

        // Flatten matrix to list.
        BoolVar[] vars = Arrays.stream(board.knightsLocation)
            .flatMap(listContainer -> Arrays.stream(listContainer))
            .toArray(size -> new BoolVar[size]);

        board.model.sum(vars, "+", board.totalKnights).post();
        return board.model.getSolver().findOptimalSolution(board.totalKnights, Model.
    MINIMIZE);
    }

    public MinimalKnights(Board board) {
        this.board = board;
    }
}
```

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

```java
package chocolatte;

import chocolatte.MuseumTypes;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.stream.Stream;
import java.util.stream.Collector;
import org.chocosolver.solver.Model;
import org.chocosolver.solver.constraints.Constraint;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.BoolVar;

public class Museum {
    static private MuseumTypes WALL = MuseumTypes.WALL;
    static private MuseumTypes EMPTY = MuseumTypes.EMPTY;
    static private MuseumTypes OUEST = MuseumTypes.OUEST;
    static private MuseumTypes EST = MuseumTypes.EST;
    static private MuseumTypes NORD = MuseumTypes.NORD;
    static private MuseumTypes SUD = MuseumTypes.SUD;

    private Board board;

    private IntVar[] get_ouest(int x, int y) {
        ArrayList<IntVar> result = new ArrayList<IntVar>();

        for (int i = x; i >= 0; --i) {
            if (!board.museumModel[y][i]) {
                break;
            } else {
                result.add(board.museum[y][i]);
            }
        }

        System.out.println(result);
        return result.stream().toArray(size -> new IntVar[size]);
    }

    private Constraint get_constraint_ouest(int x, int y) {
        int[] values = {EST.getValue()};
        IntVar nbVar = board.model.intVar("constraint_ouest_camera_" + x + "_" + y, 1, 1);
        IntVar[] fields = get_ouest(x, y);
        return board.model.among(nbVar, fields, values);
    }

    private IntVar[] get_est(int x, int y) {
        ArrayList<IntVar> result = new ArrayList<IntVar>();

        for (int i = x; i < board.boardSize; ++i) {
            if (!board.museumModel[y][i]) {
                break;
            } else {
                result.add(board.museum[y][i]);
            }
        }

        return result.stream().toArray(size -> new IntVar[size]);
    }

    private Constraint get_constraint_est(int x, int y) {
        int[] values = {OUEST.getValue()};
        IntVar nbVar = board.model.intVar("constraint_est_camera_" + x + "_" + y, 1, 1);
        IntVar[] fields = get_est(x, y);
        return board.model.among(nbVar, fields, values);
    }

    private IntVar[] get_nord(int x, int y) {
        ArrayList<IntVar> result = new ArrayList<IntVar>();

        for (int i = x; i >= 0; --i) {
            if (!board.museumModel[i][x]) {
                break;
            } else {
                result.add(board.museum[i][x]);
            }
        }

        return result.stream().toArray(size -> new IntVar[size]);
    }

    private Constraint get_constraint_nord(int x, int y) {
        int[] values = {SUD.getValue()};
        IntVar nbVar = board.model.intVar("constraint_nord_camera_" + x + "_" + y, 1, 1);
        IntVar[] fields = get_nord(x, y);
        return board.model.among(nbVar, fields, values);
    }

    private IntVar[] get_sud(int x, int y) {
        ArrayList<IntVar> result = new ArrayList<IntVar>();

        for (int i = x; i < board.boardSize; ++i) {
            if (!board.museumModel[i][x]) {
                break;
            } else {
                result.add(board.museum[i][x]);
            }
        }

        return result.stream().toArray(size -> new IntVar[size]);
    }

    private Constraint get_constraint_sud(int x, int y) {
        int[] values = {NORD.getValue()};
        IntVar nbVar = board.model.intVar("constraint_sud_camera_" + x + "_" + y, 1, 1);
        IntVar[] fields = get_sud(x, y);
        return board.model.among(nbVar, fields, values);
    }

    public Solution exec() {
        for (int i = 0; i < board.boardSize; ++i) {
            for (int j = 0; j < board.boardSize; ++j) {
                if (board.museumModel[j][i]) {
                    board.model.or(
                        get_constraint_ouest(i, j),
                        get_constraint_est(i, j),
                        get_constraint_nord(i, j),
                        get_constraint_sud(i, j)
                    ).post();
                } else {
                    board.model.arithm(board.museum[j][i], "=", WALL.getValue()).post();
                }
            }
        }

        // Sum ouest camera + est + nord + sud

        IntVar minimize = board.model.intVar("minimizing", 0, board.boardSize * board.
    boardSize);
```

```java
        // Flatten matrix to list.
        IntVar[] vars = Arrays.stream(board.museum)
            .flatMap(listContainer -> Arrays.stream(listContainer))
            .toArray(size -> new IntVar[size]);

        board.model.sum(vars, "+", minimize).post();
        // return board.model.getSolver().findSolution();
        return board.model.getSolver().findOptimalSolution(minimize, Model.MINIMIZE);
    }

    public Museum(Board board) {
        this.board = board;
    }
}
```
Listing 7 – ../src/main/java/chocolatte/Museum.java