

# Solving Decision and Optimization Problems using Constraint Logic Programming

FEUP-PLOG 3MIEIC02 - Close or Far\_4

Gonçalo Marantes

Department of Informatics and Computing Engineering  
Faculty of Engineering University of Porto  
up201706917  
goncalo.marantes@ieee.org

Simão Santos

Department of Informatics and Computing Engineering  
Faculty of Engineering University of Porto  
up201504695  
simao.p.m.santos@gmail.com

**Abstract**—This project was developed during the course Logic Programming [1], using *SICStus* [2] as the Prolog Development Environment. The goal of this project is to solve a decision and optimization problem using constraint logic programming. The chosen problem for this project was to solve as well as generate new problems for the *Close or Far* [3] puzzle game. That being said, using Prolog and *SICStus's Constraint Logic Programming over Finite Domains Library* [4], solving and generating puzzles was both possible and simple.

**Index Terms**—decision problems, logic programming, constraints, puzzle, prolog

## I. INTRODUCTION

This project was developed using the *SICStus Prolog Development Environment* during the 3<sup>rd</sup> year course Logic Programming, part of the Integrated Master's Degree in Informatics and Computing Engineering in the Faculty of Engineering of the University of Porto. The main goal of this project is to build a program using Constraint Logic Programming for solving the *Close or Far* puzzle game developed by Mathematics Professor Erich Friedman, as well as generate new puzzles with an arbitrary size and complexity.

This article has the following structure:

- **Problem Description:** detailed puzzle description and rules
- **Approach:** problem description modulation as a Constraint Satisfaction Problem
  - **Decision Variables:** domains and decision variables descriptions.
  - **Constraints:** constraints descriptions and its implementation using *SICStus Prolog*.
  - **Evaluation Function:** evaluation function description and its implementation using *SICStus Prolog*.
  - **Search Strategy:** *labelling* strategy description
- **Solution Presentation:** explanation of predicates used to view solution in text mode.
- **Results:** demonstration of different complexity problems and outcome analysis.

- **Conclusion and Future Work:** conclusions taken from the project.
- **References:** books, articles and web pages consulted to develop this project.
- **Annex:** source code, data files and results.

## II. PROBLEM DESCRIPTION

A *Close or Far* puzzle consists of a grid with the same number of rows as columns. Initially, each row and column have only one letter, *C* or *F*. For example, in *Fig. 1* we can see that the *F* on the first row is the only hint in both its row and column.

			F		
					F
				C	
		C			
	C				
F					
			C		

Fig. 1. Example of an incomplete puzzle

To solve this puzzle the blank squares can be filled with either the letter *C*, *F* or be left blank and for that there are two rules. First, each row and column should contain exactly two occurrences of each letter. And second, in each row and column, the distance between the *C*'s must be less than the distance between the *F*'s. In other words, the *C*'s are closer and the *F*'s are farther, hence the name *Close or Far*.

F			F		C	C
C	F	C				F
		F		F	C	C
	C	C	F		F	
C	C			F		F
F		F	C	C		
	F		C	C	F	

Fig. 2. Example of a complete puzzle

### III. APPROACH

Since our goal was to develop a program to generate both puzzles and solutions, we had two problems to work with: build a puzzle solver and a generator using *Constraint Logic Programming*. Since these two problems are of different complexity and so was our approach.

#### A. Solver

```

1 solve_puzzle(Board, Runtime):-
2   % --- START STATISTICS ---
3   %statistics - runtime calculation (ms)
4   statistics(runtime, [Start|_]),
5   % --- DECISION VARIABLES ---
6   % length is already defined by Board
7   apply_solver_domain(Board),
8   % --- CONSTRAINTS ---
9   apply_solver_occurrences_constraints(Board),
10  apply_solver_distance_constraints(Board),
11  % --- LABELING ---
12  % flatten Board into a 1 dimensional list
13  append(Board, FlatBoard),
14  labeling([median], FlatBoard),
15  % --- END STATISTICS ---
16  %statistics - runtime calculation (ms)
17  statistics(runtime, [Stop|_]),
18  Runtime is Stop - Start.
```

Listing 1. Puzzle solver predicate

1) *Decision Variables*: For our solver we used a matrix representation, i.e a list of lists, for our board. This way we can understand which cells are not yet instantiated and make it easier to apply our constraints.

```

1 puzzle([
2   [_ , _ , _ , 2 , _ , _ , _],
3   [_ , _ , _ , _ , _ , _ , 2],
4   [_ , _ , _ , _ , _ , 1 , _],
5   [_ , _ , 1 , _ , _ , _ , _],
6   [_ , 1 , _ , _ , _ , _ , _],
7   [2 , _ , _ , _ , _ , _ , _],
8   [_ , _ , _ , _ , 1 , _ , _]
9 ]).
```

Listing 2. Fig. 1 Puzzle representation

With the help of the *domain* [5] predicate, *apply\_solver\_domain(+Board)* applies to each row the domain  $\{0, 1, 2\}$ . 0 for blank, 1 for *C* (Close) and 2 for *F* (Far).

2) *Constraints*: As mentioned there are two major rules for solving this puzzle:

- each row and column should contain exactly two occurrences of each letter
- in each row and column, the distance between the *C*'s must be less than the distance between the *F*'s.

For this we use predicates *apply\_solver\_occurrences\_constraints(+Board)* and *apply\_solver\_distance\_constraints(+Board)* respectively.

For the first predicate we use *global\_cardinality* [6] for each row, which pairs are  $[0\text{-NumberOfZeros}, 1\text{-}2, 2\text{-}2]$ , being that *NumberOfZeros* is calculated by subtracting the number

of *C*'s and *F*'s, which is always 4, to row's length. After applying these constraints to each row we need to do the same for each column, so we just use the *transpose* [7] predicate to transpose the Board and repeat the process.

The second predicate is somewhat more difficult due to the fact it is necessary to calculate distances between elements that have not yet been instantiated. In order to calculate this distance we use the *element* [8] predicate of the *Constraint Logic Programming over Finite Domains* Library. This predicate is executed four times, two for each letter, so that per letter we get the first and second (and final) occurrence. Then it is just necessary to subtract each letter's indexes to calculate the distance between them.

```

1 get_distance_between_elements(List, Element,
2   Distance):-
3   % find indexes of Element in List
4   element(FirstIndex, List, Element),
5   element(SecondIndex, List, Element),
6   % indexes have to be unique
7   FirstIndex #\= SecondIndex,
8   % first index has to come first
9   FirstIndex #< SecondIndex,
10  % calculate distance
11  Distance #= SecondIndex - FirstIndex.
```

Listing 3. get\_distance\_between\_elements predicate

After getting the distance between each letter in a single row, we simply assert that the distance between 1's lesser that the distance between 2's.

Similarly to the first predicate, we repeat this process for each row, then transpose the board using the *transpose* predicate and repeat once more per column.

3) *Evaluation Function*: According to *Close or Far*'s web page [3], each puzzle has only one solution, even if a single solution can be applied to multiple puzzles. Because of that, we can conclude that every solution is optimal.

Since every solution is automatically the best, there is no need to evaluate it.

4) *Search Strategy*: In early development, we used the default labeling options [9], which are the *leftmost* option for variable selection and the *step* option for value selection.

After completion, various other options were tried and tested in order to find the best one. A summary of those tests can be found in the *Results* and *Annex* sections.

Looking at the time results for variable choice heuristic we concluded that both *left\_most* and *occurrence* are good variable selection options, and do not change much between them, so we ended up choosing the default option.

It came to us as a surprise when choosing the best value choice heuristic, it is clear that the best option is *median*, taking much less time when compared to the others. And for that reason we decided to use it.

## B. Generator

In order to build a puzzle generator, we use the previous developed puzzle solver to generate a random solution, and then strip that same solution whilst applying constraints.

```

1 generate_random_puzzle(Size, PuzzleBoard, Runtime):-
2     % --- START STATISTICS ---
3     %statistics - runtime calculation (ms)
4     statistics(runtime, [Start|_]),
5     % generate random solution
6     generate_random_solution(Size, SolutionBoard),
7     % find hints from solution board
8     find_hints(SolutionBoard, HintColumns,
9     HintValues),
10    % generate empty board
11    generate_board(Size, PuzzleBoard),
12    % populate puzzle from hints found
13    populate_puzzle(PuzzleBoard, HintColumns,
14    HintValues),
15    % --- END STATISTICS ---
16    %statistics - runtime calculation (ms)
17    statistics(runtime, [Stop|_]),
18    Runtime is Stop - Start.

```

Listing 4. Puzzle generator predicate

```

1 find_hints(SolutionBoard, HintColumns, HintValues):-
2     % --- DECISION VARIABLES ---
3     length(SolutionBoard, NumberOfColumns),
4     % hintColumns' domain is [1, Board's number of
5     columns]
6     length(HintColumns, NumberOfColumns),
7     domain(HintColumns, 1, NumberOfColumns),
8     % hintValues's domain is 1 or 2 (C or F)
9     length(HintValues, NumberOfColumns),
10    domain(HintValues, 1, 2),
11    % --- CONSTRAINTS ---
12    % only one hint per row and column
13    all_distinct(HintColumns),
14    apply_generator_occurrences_constraints(
15    SolutionBoard, HintColumns, HintValues),
16    % --- LABELING ---
17    % flatten Board into a 1 dimensional list
18    append(HintColumns, HintValues, Hints),
19    labeling([value(random_value)], Hints).

```

Listing 5. find\_hints predicate

1) *Decision Variables*: Unlike our solver instead of using a matrix, we used two lists, both of which have the same length as the board's number of columns, since every row and column has only one hint, *C* or *F*.

The first list contains each hint's column and the second one represents their actual value, being 1 or 2.

```

1 HintColumns([4, 7, 6, 3, 2, 1, 5]).
2 HintValues([2, 2, 1, 1, 1, 2, 1]).

```

Listing 6. Fig. 1 lists representation

Using the *domain* predicate, *find\_hints(+SolutionBoard, -HintColumns, -HintValues)* applies to list *HintColumns* the domain  $[1, \text{NumberOfColumns}]$ , being that *NumberOfColumns* is equal to the number of columns of *SolutionBoard*. Moving on to *HintValues*, its domain is equal to  $\{1, 2\}$  since the only values that hints can have is either 1 or 2.

2) *Constraints*: There is only one constraint to generate a puzzle, besides the fact that the puzzle has to be solvable, and that is: each row and column should contain only one occurrence of either a *C* or an *F*.

We have certainty that the puzzle that is going to be generated is solvable, since we are trimming an actual solved puzzle in order to build it.

In order to achieve a single hint in every row and column, we apply the *all\_distinct* [6] predicate to *HintColumns*. Right after that, the *apply\_generator\_occurrences\_constraints(+SolutionBoard, -HintColumns, -HintValues)* takes care of choosing an element per row of *SolutionBoard* and saving its index and value to *HintColumns* and *HintValues* respectively.

3) *Evaluation Function*: Like it is mentioned in Solver's evaluation function, every puzzle has a single solution, but various puzzles can be extracted from the same solution.

However, each puzzle extracted is unique and as long as the constraints are applied there is no optimal puzzle that can be built from a solution.

4) *Search Strategy*: Due to the fact that every puzzle generated from a single solution is as favorable as the others, it is just necessary to choose a random one.

In order to do that, we built our custom value selection labeling option, which is the same one used to generate a random solution.

```

1 random_value(Var, _Rest, BB, BB1):-
2     % get finite domain set
3     fd_set(Var, Set),
4     % transform it to list and choose a random value
5     fdset_to_list(Set, List),
6     random_member(RandomValue, List),
7     (
8         first_bound(BB, BB1, Var #= RandomValue
9         ;
10        later_bound(BB, BB1, Var #\= RandomValue
11        ).

```

Listing 7. random\_value predicate

## IV. SOLUTION PRESENTATION

To display the board we simply use 5 linked predicates:

- 1) *print\_board(+Board)*
- 2) *print\_horizontal\_separator(+N)*
- 3) *print\_matrix(+Matrix)*
- 4) *print\_line(+Row)*
- 5) *print\_cell(+Cell)*

The main one, *print\_board(+Board)* as seen below, starts by figuring out the width of the first line (the number of columns), and printing the character "|", as the start of the board's border. Then prints successive "—" as many times as the number of columns, by use of the *print\_horizontal\_separator(+N)* predicate.

## V. RESULTS

To test this solution, 10 puzzles were randomly generated for each board size (starting from 6 by 6 and finishing in 10 by 10). After running the solver for each board an average was calculated for each strategy and the results can be found in table I and table II in the *Annex*. Note that some tests do not have a final value, this is because the test was lasting far beyond what was desired, because of this there was no need to continue testing.

We can see that a good search strategy for the solver is a combination of *left\_most*, which is the default for *labeling*, or *occurrence* as the variable choice option and *median* as the value choice option. Although it is important to note that the puzzle itself highly influences the choice for the best search strategy. That is, different puzzles with the same size had lower run times for a given search strategy, even if that same search strategy's overall run time is higher. For example, the outcome of *anti\_first\_fail* search strategy applied to 7 by 7 puzzles can also be found in table III in the *Annex* section.

## VI. CONCLUSIONS AND FUTURE WORK

In summary, with this puzzle problem the group learned to work with a constraint programming paradigm, using *SICStus Prolog Development Environment*. Constraint logic programming is widely used in optimisation and scheduling problems and in more languages than Prolog [10].

During the development of this project it became clear the influence and importance the modelling phase has when working with constraint logic programming. As the problem can be modeled and represented in various ways having a big influence on logic and constraints modelling.

In the future the group would like to learn more about constraint programming as its efficiency and utility was above expected. Thinking in a more declarative way and restricting the domain of a seemingly hard problem can quickly turn it into an easily solvable problem for a computer.

## REFERENCES

- [1] sigarra.up.pt. (2020). FEUP - Programação em Lógica. [online] Available at: [https://sigarra.up.pt/feup/pt/ucurr\\_geral.ficha\\_uc\\_view?pv\\_ocorrencia\\_id=436444](https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrencia_id=436444) [Accessed 2 Jan. 2020].
- [2] sicstus.sics.se. (2020). SICStus Prolog Homepage. [online] Available at: <https://sicstus.sics.se/> [Accessed 2 Jan. 2020].
- [3] Friedman, E. (2011). Close Or Far Puzzles. [online] Available at: <https://www2.stetson.edu/~efriedma/puzzle/closefar/> [Accessed 24 Nov. 2019].
- [4] sicstus.sics.se. (2020). SICStus Prolog: lib-clpfd. [online] Available at: [https://sicstus.sics.se/sicstus/docs/4.5.1/html/sicstus/lib\\_002dclpfd.html](https://sicstus.sics.se/sicstus/docs/4.5.1/html/sicstus/lib_002dclpfd.html) [Accessed 2 Jan. 2020].
- [5] sicstus.sics.se. (2020). SICStus Prolog: Membership Constraints. [online] Available at: <https://sicstus.sics.se/sicstus/docs/4.5.1/html/sicstus/Membership-Constraints.html> [Accessed 3 Jan. 2020].
- [6] sicstus.sics.se. (2020). SICStus Prolog: Arithmetic-Logical Constraints. [online] Available at: [https://sicstus.sics.se/sicstus/docs/4.5.1/html/sicstus/Arithmetic\\_002dLogical-Constraints.html](https://sicstus.sics.se/sicstus/docs/4.5.1/html/sicstus/Arithmetic_002dLogical-Constraints.html) [Accessed 3 Jan. 2020].
- [7] sicstus.sics.se. (2020). SICStus Prolog: lib-lists. [online] Available at: [https://sicstus.sics.se/sicstus/docs/4.5.1/html/sicstus/lib\\_002dlists.html](https://sicstus.sics.se/sicstus/docs/4.5.1/html/sicstus/lib_002dlists.html) [Accessed 3 Jan. 2020].
- [8] sicstus.sics.se. (2020). SICStus Prolog: Extensional Constraints. [online] Available at: <https://sicstus.sics.se/sicstus/docs/4.5.1/html/sicstus/Extensional-Constraints.html> [Accessed 3 Jan. 2020].

```
1 print_board([Line | Board]):-
2     length(Line, Width),
3     write('|'),
4     print_horizontal_separator(Width),
5     nl,
6     length([Line | Board], Height),
7     print_matrix([Line | Board], Height),
8     nl.
```

Listing 8. print\_board predicate

```
1 print_horizontal_separator(0).
2 print_horizontal_separator(N):-
3     write('---|'),
4     N1 is N - 1,
5     print_horizontal_separator(N1).
```

Listing 9. print\_horizontal\_separator predicate

On to print the rest of the matrix via *print\_matrix(+Matrix)*. We get the number of rows (the height) and we print row by row the character separator "|", the content of each cell and another "|". All while using the *print\_line(+Row)* predicate, which also translates the number by which C, F or 'empty' are internally represented.

```
1 print_matrix([], 0).
2 print_matrix([Line | Board], N):-
3     write('|'),
4     print_line(Line),
5     nl,
6     length(Line, Length),
7     N1 is N - 1,
8     write('|'),
9     print_horizontal_separator(Length),
10    nl,
11    print_matrix(Board, N1).
```

Listing 10. print\_matrix predicate

```
1 print_line([]).
2 print_line([Cell | Line]):-
3     print_cell(Cell),
4     write(' | '),
5     print_line(Line).
6
7 print_cell(Cell):-
8     translate(Cell, Char),
9     write(Char).
```

Listing 11. print\_line and print\_cell predicates

In the end, the result is something like this:

```

|---|---|---|---|---|---|
| F | . | . | F | . | C | C |
|---|---|---|---|---|---|
| C | F | C | . | . | . | F |
|---|---|---|---|---|---|
| . | F | . | . | F | C | C |
|---|---|---|---|---|---|
| . | C | C | F | . | F | . |
|---|---|---|---|---|---|
| C | C | . | . | F | . | F |
|---|---|---|---|---|---|
| F | . | F | C | C | . | . |
|---|---|---|---|---|---|
| . | F | . | C | C | F | . |
|---|---|---|---|---|---|
```

Listing 12. Fig. 1 complete puzzle with our display method

- [9] sicstus.sics.se. (2020). SICStus Prolog: Enumeration Predicates. [online] Available at: <https://sicstus.sics.se/sicstus/docs/4.5.1/html/sicstus/Enumeration-Predicates.html> [Accessed 4 Jan. 2020].
- [10] PyPI. (2020). python-constraint. [online] Available at: <https://pypi.org/project/python-constraint/> [Accessed 3 Jan. 2020].

## ANNEX

### A. Data

TABLE I  
VARIABLE CHOICE HEURISTIC TESTS  
TIME IN SECONDS

	leftmost	min	max	first_fail
6	0.004	0.364	0.004	0.004
7	0.054	66.407	0.056	0.059
8	0.293	N/A	0.265	0.498
9	0.851	N/A	1.940	2.335
10	16,221	N/A	40,250	62,640
	anti_first_fail	occurrence	most_constrained	max_regret
6	0.361	0.003	0.003	0.004
7	65.898	0.056	0.059	0.084
8	N/A	0.273	0.467	0.506
9	N/A	0.842	2.403	1.295
10	N/A	16.503	63.406	30.629

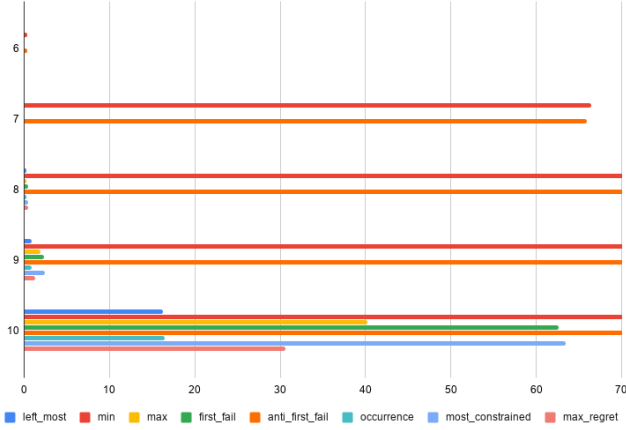


Fig. 3. Variable Choice Heuristic Tests - Time in seconds

TABLE II  
VALUE CHOICE HEURISTIC TESTS  
TIME IN SECONDS

	step	enum	bisect	median	middle
6	0.006	0.001	0.004	0.006	0.003
7	0.059	0.053	0.051	0.035	0.017
8	0.298	0.259	0.251	0.095	0.289
9	0.876	0.762	0.742	0.034	0.089
10	15.782	15.093	14,900	1.606	10.257

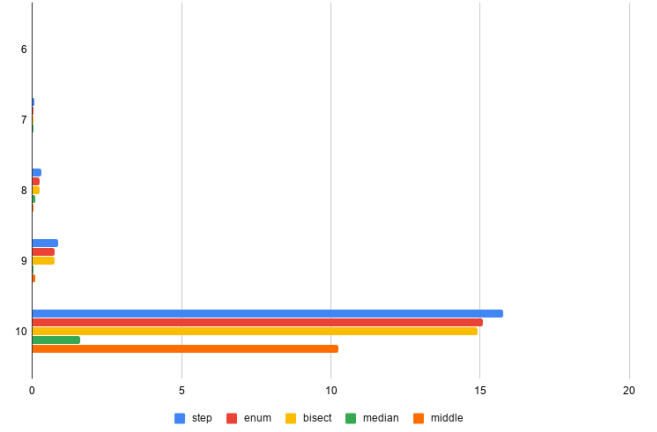


Fig. 4. Value Choice Heuristic Tests - Time in seconds

TABLE III  
ANTI-FIRST\_FAIL SEARCH STRATEGY TEST FOR 7X7 PUZZLES

	Time [s]
Puzzle 1	1.437
Puzzle 2	107.188
Puzzle 3	13.437
Puzzle 4	176.438
Puzzle 5	60.109
Puzzle 6	96.516
Puzzle 7	28.656
Puzzle 8	95.016
Puzzle 9	0.375
Puzzle 10	79.812

### B. Source Code

```

1 :- consult('generator.pl').
2 :- consult('solver.pl').
3 :- consult('display.pl').
4
5 play(Size):-
6     % generate random puzzle with given size
7     generate_random_puzzle(Size, PuzzleBoard,
8                             GeneratorRuntime),
9     % print puzzle
10    print_board(PuzzleBoard),
11    % print generator runtime
12    format('~> Generator runtime: ~3d s~n', [
13        GeneratorRuntime]),
14    % solve puzzle
15    solve_puzzle(SolutionBoard, SolverRuntime),
16    % print solved puzzle
17    print_board(SolutionBoard),
18    % print solver runtime
19    format('~> Solver runtime: ~3d s~n', [
20        SolverRuntime]).

```

Listing 13. close\_or\_far.pl

```

1 :- use_module(library(clpfd)).
2 :- consult('utils.pl').
3
4 /*
5 0 -> EMPTY

```



```

6 1 -> CLOSE
7 2 -> FAR
8 */
9
10 % =====
11 % Decision Variables
12 % =====
13 apply_solver_domain([]).
14 apply_solver_domain([Row | Board]):-
15     % (0 - empty, 1 - close, 2 - far)
16     domain(Row, 0, 2),
17     apply_solver_domain(Board).
18
19 % =====
20 % Constraints
21 % =====
22 % ===== Occurrences constraints =====
23 apply_solver_occurrences_constraints(Board):-
24     % calculate number of 0's (empty)
25     % get number of cells per row (same as per col)
26     length(Board, NumberOfCellsPerRow),
27     % get number of 0's (number of cells per row - (
28     % number of 1's + number of 2's))
29     NumberOfZeros #= NumberOfCellsPerRow - 4,
30     % apply constraints per row
31     solver_occurrences_constraints_per_row(Board,
32     NumberOfZeros),
33     % apply constraints per column (with transposed
34     % matrix)
35     transpose(Board, TransposedBoard),
36     solver_occurrences_constraints_per_row(
37     TransposedBoard, NumberOfZeros).
38
39 solver_occurrences_constraints_per_row([],
40     _NumberOfZeros).
41 solver_occurrences_constraints_per_row([Row | Board
42 ], NumberOfZeros):-
43     global_cardinality(Row, [
44     % NumberOfZeros occurrences of 0 (empty) per
45     % row
46     0-NumberOfZeros,
47     % 2 occurrences of 1 (close) per row
48     1-2,
49     % 2 occurrences of 2 (far) per row
50     2-2
51     ]),
52     % apply constraints to next rows
53     solver_occurrences_constraints_per_row(Board,
54     NumberOfZeros).
55
56 % ===== Distances =====
57 apply_solver_distance_constraints(Board):-
58     % apply constraints per row
59     solver_distance_constraints_per_row(Board),
60     % apply constraints per column (with transposed
61     % board matrix)
62     transpose(Board, TransposedBoard),
63     solver_distance_constraints_per_row(
64     TransposedBoard).
65
66 get_distance_between_elements(List, Element,
67     Distance):-
68     % find indexes of Element in List
69     element(FirstIndex, List, Element),
70     element(SecondIndex, List, Element),
71     % indexes have to be unique
72     FirstIndex #\= SecondIndex,
73     % first index has to come first
74     FirstIndex #< SecondIndex,
75     % calculate distance
76     Distance #= SecondIndex - FirstIndex.
77
78 solver_distance_constraints_per_row([]).
79 solver_distance_constraints_per_row([Row | Board]):-

```

```

69 % get distance between 1's (close)
70 get_distance_between_elements(Row, 1,
71     CloseDistance),
72 % get distance between 2's (far)
73 get_distance_between_elements(Row, 2,
74     FarDistance),
75 % apply MAIN constraints
76 CloseDistance #< FarDistance,
77 % apply constraints to next rows
78 solver_distance_constraints_per_row(Board).
79
80 % =====
81 % Puzzle Solver
82 % =====
83 solve_puzzle(Board, Runtime):-
84     % --- START STATISTICS ---
85     % statistics - runtime calculation (ms)
86     statistics(runtime, [Start|_]),
87     % --- DECISION VARIABLES ---
88     % length is already defined by Board
89     apply_solver_domain(Board),
90     % --- CONSTRAINTS ---
91     apply_solver_occurrences_constraints(Board),
92     apply_solver_distance_constraints(Board),
93     % --- LABELING ---
94     % flatten Board into a 1 dimensional list
95     append(Board, FlatBoard),
96     labeling([median], FlatBoard),
97     % --- END STATISTICS ---
98     % statistics - runtime calculation (ms)
99     statistics(runtime, [Stop|_]),
100     Runtime is Stop - Start.

```

Listing 14. solver.pl

```

1 :- consult('utils.pl').
2 :- consult('solver.pl').
3 :- use_module(library(random)).
4
5 % =====
6 % Generate Empty Board
7 % =====
8 % generate empty board matrix given its size
9 generate_board(Size, Board):-
10     generate_board_aux(Size, Size, Board).
11
12 generate_board_aux(_, 0, []).
13 generate_board_aux(Size, N, [Row | Board]):-
14     generate_row(Size, Row),
15     NewN is N - 1,
16     generate_board_aux(Size, NewN, Board).
17
18 % generate empty board row given its size
19 generate_row(0, []).
20 generate_row(Size, [_ | Row]):-
21     NewSize is Size - 1,
22     generate_row(NewSize, Row).
23
24 % =====
25 % Populate Puzzle with hints found from Solution
26 % =====
27 populate_puzzle([], [], []).
28 populate_puzzle([PuzzleRow | PuzzleBoard], [
29     HintColumnsCell | HintColumns], [HintValuesCell
30     | HintValues]):-
31     % write HintValue in HintColumn of PuzzleRow
32     nth1(HintColumnsCell, PuzzleRow, HintValuesCell),
33     % repeat for next row
34     populate_puzzle(PuzzleBoard, HintColumns,
35     HintValues).
36
37 % =====
38 % Find hints from Solution

```

```

36 % =====
37 apply_generator_occurrences_constraints([], [], []).
38 apply_generator_occurrences_constraints([SolutionRow
    | SolutionBoard], [HintColumnsCell |
    HintColumns], [HintValuesCell | HintValues]):-
39     % hint column and value have to be consistent
    with solution row
40     element(HintColumnsCell, SolutionRow,
    HintValuesCell),
41     % repeat for next row
42     apply_generator_occurrences_constraints(
    SolutionBoard, HintColumns, HintValues).
43
44 find_hints(SolutionBoard, HintColumns, HintValues):-
45     % --- DECISION VARIABLES ---
46     length(SolutionBoard, NumberOfColumns),
47     % hintColumns' domain is [1, Board's number of
    columns]
48     length(HintColumns, NumberOfColumns),
49     domain(HintColumns, 1, NumberOfColumns),
50     % hintValues's domain is 1 or 2 (C or F)
51     length(HintValues, NumberOfColumns),
52     domain(HintValues, 1, 2),
53     % --- CONSTRAINTS ---
54     % only one hint per row and column
55     all_distinct(HintColumns),
56     apply_generator_occurrences_constraints(
    SolutionBoard, HintColumns, HintValues),
57     % --- LABELING ---
58     % flatten Board into a 1 dimensional list
59     append(HintColumns, HintValues, Hints),
60     labeling([value(random_value)], Hints).
61
62 % =====
63 % Puzzle Generator
64 % =====
65 generate_random_solution(Size, Board):-
66     % --- DECISION VARIABLES ---
67     % generate empty board
68     generate_board(Size, Board),
69     % length is already defined by Board
70     apply_solver_domain(Board),
71     % --- CONSTRAINTS ---
72     apply_solver_occurrences_constraints(Board),
73     apply_solver_distance_constraints(Board),
74     % --- LABELING ---
75     % flatten Board into a 1 dimensional list
76     append(Board, FlatBoard),
77     labeling([value(random_value)], FlatBoard).
78
79 generate_random_puzzle(Size, PuzzleBoard, Runtime):-
80     % --- START STATISTICS ---
81     %statistics - runtime calculation (ms)
82     statistics(runtime, [Start|_]),
83     % generate random solution
84     generate_random_solution(Size, SolutionBoard),
85     % find hints from solution board
86     find_hints(SolutionBoard, HintColumns,
    HintValues),
87     % generate empty board
88     generate_board(Size, PuzzleBoard),
89     % populate puzzle from hints found
90     populate_puzzle(PuzzleBoard, HintColumns,
    HintValues),
91     % --- END STATISTICS ---
92     %statistics - runtime calculation (ms)
93     statistics(runtime, [Stop|_]),
94     Runtime is Stop - Start.
95
96 % =====
97 % Labeling Options
98 % =====
99 % select random value
100 random_value(Var, _Rest, BB, BB1):-

```

```

101 % get finite domain set
    fd_set(Var, Set),
102 % trasform it to list and choose a random value
    fdset_to_list(Set, List),
103 random_member(RandomValue, List),
104 (
105     first_bound(BB, BB1, Var #= RandomValue
106     ;
107     later_bound(BB, BB1, Var #\= RandomValue
108     ).

```

Listing 15. generator.pl

```

1 :- consult('utils.pl').
2
3 % =====
4 % Display Board
5 % =====
6 print_board([Line | Board]):-
7     length(Line, Width),
8     write('|'),
9     print_horizontal_separator(Width),
10    nl,
11    length([Line | Board], Height),
12    print_matrix([Line | Board], Height),
13    nl.
14
15 print_horizontal_separator(0).
16 print_horizontal_separator(N):-
17     write('---|'),
18     N1 is N - 1,
19     print_horizontal_separator(N1).
20
21 print_matrix([], 0).
22 print_matrix([Line | Board], N):-
23     write('| '),
24     print_line(Line),
25     nl,
26     length(Line, Length),
27     N1 is N - 1,
28     write('| '),
29     print_horizontal_separator(Length),
30     nl,
31     print_matrix(Board, N1).
32
33 print_line([]).
34 print_line([Cell | Line]):-
35     print_cell(Cell),
36     write(' | '),
37     print_line(Line).
38
39 print_cell(Cell):-
40     translate(Cell, Char),
41     write(Char).

```

Listing 16. display.pl

```

1 :- use_module(library(lists)).
2
3 % =====
4 % translate board atoms into chars
5 % =====
6 translate(0, '.').
7 translate(1, 'C').
8 translate(2, 'F').
9
10 % =====
11 % Replace 0's with _
12 % =====
13 replace_zeros_matrix([], []).
14 replace_zeros_matrix([Row | Matrix], [NewRow | Rest
    ]):-
15     replace_zeros_matrix(Matrix, Rest),
16     replace_zeros_row(Row, NewRow).

```

```

17 replace_zeros_row([], []).
18 replace_zeros_row([0 | Tail], [_ | Rest]):-
19     replace_zeros_row(Tail, Rest).
20 replace_zeros_row([Head | Tail], [Head | Rest]):-
21     replace_zeros_row(Tail, Rest).
22

```

Listing 17. utils.pl

```

1 :- consult('solver.pl').
2 :- consult('puzzles.pl').
3
4 % =====
5 % Testing Purposes
6 % See puzzles in file 'puzzles.pl'
7 % =====
8 test_solver_6:-
9     puzzle_6_1(Board1),
10    solve_puzzle(Board1, Runtime1),
11    format('~> Puzzle 1 solver runtime: ~3d s~n', [
12        Runtime1]),
13    puzzle_6_2(Board2),
14    solve_puzzle(Board2, Runtime2),
15    format('~> Puzzle 2 solver runtime: ~3d s~n', [
16        Runtime2]),
17    puzzle_6_3(Board3),
18    solve_puzzle(Board3, Runtime3),
19    format('~> Puzzle 3 solver runtime: ~3d s~n', [
20        Runtime3]),
21    puzzle_6_4(Board4),
22    solve_puzzle(Board4, Runtime4),
23    format('~> Puzzle 4 solver runtime: ~3d s~n', [
24        Runtime4]),
25    puzzle_6_5(Board5),
26    solve_puzzle(Board5, Runtime5),
27    format('~> Puzzle 5 solver runtime: ~3d s~n', [
28        Runtime5]),
29    puzzle_6_6(Board6),
30    solve_puzzle(Board6, Runtime6),
31    format('~> Puzzle 6 solver runtime: ~3d s~n', [
32        Runtime6]),
33    puzzle_6_7(Board7),
34    solve_puzzle(Board7, Runtime7),
35    format('~> Puzzle 7 solver runtime: ~3d s~n', [
36        Runtime7]),
37    puzzle_6_8(Board8),
38    solve_puzzle(Board8, Runtime8),
39    format('~> Puzzle 8 solver runtime: ~3d s~n', [
40        Runtime8]),
41    puzzle_6_9(Board9),
42    solve_puzzle(Board9, Runtime9),
43    format('~> Puzzle 9 solver runtime: ~3d s~n', [
44        Runtime9]),
45    puzzle_6_10(Board10),
46    solve_puzzle(Board10, Runtime10),
47    format('~> Puzzle 10 solver runtime: ~3d s~n', [
48        Runtime10]),
49    AvgRuntime is (Runtime1 + Runtime2 + Runtime3 +
50        Runtime4 + Runtime5 + Runtime6 + Runtime7 +
51        Runtime8 + Runtime9 + Runtime10) / 10,
52    format('~> Average runtime: ~3d s~n', [
53        AvgRuntime]).
54
55 test_solver_7:-
56     puzzle_7_1(Board1),
57    solve_puzzle(Board1, Runtime1),
58    format('~> Puzzle 1 solver runtime: ~3d s~n', [
59        Runtime1]),
60    puzzle_7_2(Board2),
61    solve_puzzle(Board2, Runtime2),
62    format('~> Puzzle 2 solver runtime: ~3d s~n', [
63        Runtime2]),
64    puzzle_7_3(Board3),
65    solve_puzzle(Board3, Runtime3),
66

```

```

51 format('~> Puzzle 3 solver runtime: ~3d s~n', [
52     Runtime3]),
53 puzzle_7_4(Board4),
54 solve_puzzle(Board4, Runtime4),
55 format('~> Puzzle 4 solver runtime: ~3d s~n', [
56     Runtime4]),
57 puzzle_7_5(Board5),
58 solve_puzzle(Board5, Runtime5),
59 format('~> Puzzle 5 solver runtime: ~3d s~n', [
60     Runtime5]),
61 puzzle_7_6(Board6),
62 solve_puzzle(Board6, Runtime6),
63 format('~> Puzzle 6 solver runtime: ~3d s~n', [
64     Runtime6]),
65 puzzle_7_7(Board7),
66 solve_puzzle(Board7, Runtime7),
67 format('~> Puzzle 7 solver runtime: ~3d s~n', [
68     Runtime7]),
69 puzzle_7_8(Board8),
70 solve_puzzle(Board8, Runtime8),
71 format('~> Puzzle 8 solver runtime: ~3d s~n', [
72     Runtime8]),
73 puzzle_7_9(Board9),
74 solve_puzzle(Board9, Runtime9),
75 format('~> Puzzle 9 solver runtime: ~3d s~n', [
76     Runtime9]),
77 puzzle_7_10(Board10),
78 solve_puzzle(Board10, Runtime10),
79 format('~> Puzzle 10 solver runtime: ~3d s~n', [
80     Runtime10]),
81 AvgRuntime is (Runtime1 + Runtime2 + Runtime3 +
82     Runtime4 + Runtime5 + Runtime6 + Runtime7 +
83     Runtime8 + Runtime9 + Runtime10) / 10,
84 format('~> Average runtime: ~3d s~n', [
85     AvgRuntime]).
86

```

```

87 test_solver_8:-
88     puzzle_8_1(Board1),
89    solve_puzzle(Board1, Runtime1),
90    format('~> Puzzle 1 solver runtime: ~3d s~n', [
91        Runtime1]),
92    puzzle_8_2(Board2),
93    solve_puzzle(Board2, Runtime2),
94    format('~> Puzzle 2 solver runtime: ~3d s~n', [
95        Runtime2]),
96    puzzle_8_3(Board3),
97    solve_puzzle(Board3, Runtime3),
98    format('~> Puzzle 3 solver runtime: ~3d s~n', [
99        Runtime3]),
100    puzzle_8_4(Board4),
101    solve_puzzle(Board4, Runtime4),
102    format('~> Puzzle 4 solver runtime: ~3d s~n', [
103        Runtime4]),
104    puzzle_8_5(Board5),
105    solve_puzzle(Board5, Runtime5),
106    format('~> Puzzle 5 solver runtime: ~3d s~n', [
107        Runtime5]),
108    puzzle_8_6(Board6),
109    solve_puzzle(Board6, Runtime6),
110    format('~> Puzzle 6 solver runtime: ~3d s~n', [
111        Runtime6]),
112    puzzle_8_7(Board7),
113    solve_puzzle(Board7, Runtime7),
114    format('~> Puzzle 7 solver runtime: ~3d s~n', [
115        Runtime7]),
116    puzzle_8_8(Board8),
117    solve_puzzle(Board8, Runtime8),
118    format('~> Puzzle 8 solver runtime: ~3d s~n', [
119        Runtime8]),
120    puzzle_8_9(Board9),
121    solve_puzzle(Board9, Runtime9),
122    format('~> Puzzle 9 solver runtime: ~3d s~n', [
123        Runtime9]),
124    puzzle_8_10(Board10),

```



```

105 solve_puzzle(Board10, Runtime10),
106 format(' > Puzzle 10 solver runtime: ~3d s~n', [
Runtime10]),
107 AvgRuntime is (Runtime1 + Runtime2 + Runtime3 +
Runtime4 + Runtime5 + Runtime6 + Runtime7 +
Runtime8 + Runtime9 + Runtime10) / 10,
108 format(' > Average runtime: ~3d s~n', [
AvgRuntime]).
109
110 test_solver_9:-
111 puzzle_9_1(Board1),
112 solve_puzzle(Board1, Runtime1),
113 format(' > Puzzle 1 solver runtime: ~3d s~n', [
Runtime1]),
114 puzzle_9_2(Board2),
115 solve_puzzle(Board2, Runtime2),
116 format(' > Puzzle 2 solver runtime: ~3d s~n', [
Runtime2]),
117 puzzle_9_3(Board3),
118 solve_puzzle(Board3, Runtime3),
119 format(' > Puzzle 3 solver runtime: ~3d s~n', [
Runtime3]),
120 puzzle_9_4(Board4),
121 solve_puzzle(Board4, Runtime4),
122 format(' > Puzzle 4 solver runtime: ~3d s~n', [
Runtime4]),
123 puzzle_9_5(Board5),
124 solve_puzzle(Board5, Runtime5),
125 format(' > Puzzle 5 solver runtime: ~3d s~n', [
Runtime5]),
126 puzzle_9_6(Board6),
127 solve_puzzle(Board6, Runtime6),
128 format(' > Puzzle 6 solver runtime: ~3d s~n', [
Runtime6]),
129 puzzle_9_7(Board7),
130 solve_puzzle(Board7, Runtime7),
131 format(' > Puzzle 7 solver runtime: ~3d s~n', [
Runtime7]),
132 puzzle_9_8(Board8),
133 solve_puzzle(Board8, Runtime8),
134 format(' > Puzzle 8 solver runtime: ~3d s~n', [
Runtime8]),
135 puzzle_9_9(Board9),
136 solve_puzzle(Board9, Runtime9),
137 format(' > Puzzle 9 solver runtime: ~3d s~n', [
Runtime9]),
138 puzzle_9_10(Board10),
139 solve_puzzle(Board10, Runtime10),
140 format(' > Puzzle 10 solver runtime: ~3d s~n', [
Runtime10]),
141 AvgRuntime is (Runtime1 + Runtime2 + Runtime3 +
Runtime4 + Runtime5 + Runtime6 + Runtime7 +
Runtime8 + Runtime9 + Runtime10) / 10,
142 format(' > Average runtime: ~3d s~n', [
AvgRuntime]).
143
144 test_solver_10:-
145 puzzle_10_1(Board1),
146 solve_puzzle(Board1, Runtime1),
147 format(' > Puzzle 1 solver runtime: ~3d s~n', [
Runtime1]),
148 puzzle_10_2(Board2),
149 solve_puzzle(Board2, Runtime2),
150 format(' > Puzzle 2 solver runtime: ~3d s~n', [
Runtime2]),
151 puzzle_10_3(Board3),
152 solve_puzzle(Board3, Runtime3),
153 format(' > Puzzle 3 solver runtime: ~3d s~n', [
Runtime3]),
154 puzzle_10_4(Board4),
155 solve_puzzle(Board4, Runtime4),
156 format(' > Puzzle 4 solver runtime: ~3d s~n', [
Runtime4]),
157 puzzle_10_5(Board5),
158 solve_puzzle(Board5, Runtime5),
159 format(' > Puzzle 5 solver runtime: ~3d s~n', [
Runtime5]),
160 puzzle_10_6(Board6),
161 solve_puzzle(Board6, Runtime6),
162 format(' > Puzzle 6 solver runtime: ~3d s~n', [
Runtime6]),
163 puzzle_10_7(Board7),
164 solve_puzzle(Board7, Runtime7),
165 format(' > Puzzle 7 solver runtime: ~3d s~n', [
Runtime7]),
166 puzzle_10_8(Board8),
167 solve_puzzle(Board8, Runtime8),
168 format(' > Puzzle 8 solver runtime: ~3d s~n', [
Runtime8]),
169 puzzle_10_9(Board9),
170 solve_puzzle(Board9, Runtime9),
171 format(' > Puzzle 9 solver runtime: ~3d s~n', [
Runtime9]),
172 puzzle_10_10(Board10),
173 solve_puzzle(Board10, Runtime10),
174 format(' > Puzzle 10 solver runtime: ~3d s~n', [
Runtime10]),
175 AvgRuntime is (Runtime1 + Runtime2 + Runtime3 +
Runtime4 + Runtime5 + Runtime6 + Runtime7 +
Runtime8 + Runtime9 + Runtime10) / 10,
176 format(' > Average runtime: ~3d s~n', [
AvgRuntime]).

```

Listing 18. labeling\_tests.pl

```

1 :-consult('solver.pl').
2 :-consult('puzzles.pl').
3
4 % =====
5 % Testing Purposes
6 % See puzzles in file 'puzzles.pl'
7 % =====
8 % test for puzzle 1
9 test_puzzle_1:-
10 puzzle_1(Board),
11 solve_puzzle(Board),
12 Solution = [
13     [2,1,1,2,0,0],
14     [1,1,2,0,2,0],
15     [1,2,0,1,0,2],
16     [2,0,1,0,2,1],
17     [0,2,0,2,1,1],
18     [0,0,2,1,1,2]
19 ],
20 Board == Solution.
21
22 % test for puzzle 2
23 test_puzzle_2:-
24 puzzle_2(Board),
25 solve_puzzle(Board),
26 Solution = [
27     [1,1,2,0,2,0],
28     [1,2,1,0,0,2],
29     [2,1,1,2,0,0],
30     [0,0,2,1,1,2],
31     [0,2,0,1,2,1],
32     [2,0,0,2,1,1]
33 ],
34 Board == Solution.
35
36 % test for puzzle 3
37 test_puzzle_3:-
38 puzzle_3(Board),
39 solve_puzzle(Board),
40 Solution = [
41     [1,1,2,0,2,0],
42     [1,1,0,2,0,2],
43     [2,0,2,1,1,0],

```

```

44         [0,2,1,1,0,2],
45         [2,0,1,0,2,1],
46         [0,2,0,2,1,1]
47     ],
48     Board == Solution.
49
50 % test for puzzle 4
51 test_puzzle_4:-
52     puzzle_4(Board),
53     solve_puzzle(Board),
54     Solution = [
55         [0,2,0,2,1,1],
56         [2,0,2,0,1,1],
57         [1,2,0,1,0,2],
58         [1,1,2,0,2,0],
59         [0,1,1,2,0,2],
60         [2,0,1,1,2,0]
61     ],
62     Board == Solution.
63
64 % test for puzzle 5
65 test_puzzle_5:-
66     puzzle_5(Board),
67     solve_puzzle(Board),
68     Solution = [
69         [1,1,0,2,0,2],
70         [1,2,1,0,2,0],
71         [2,1,1,2,0,0],
72         [0,0,2,1,1,2],
73         [0,2,0,1,2,1],
74         [2,0,2,0,1,1]
75     ],
76     Board == Solution.
77
78 % test for puzzle 6
79 test_puzzle_6:-
80     puzzle_6(Board),
81     solve_puzzle(Board),
82     Solution = [
83         [0,2,0,0,1,2,1],
84         [2,0,2,0,0,1,1],
85         [0,2,0,2,1,1,0],
86         [1,1,0,0,2,0,2],
87         [0,1,2,1,0,2,0],
88         [1,0,1,2,0,0,2],
89         [2,0,1,1,2,0,0]
90     ],
91     Board == Solution.
92
93 % test for puzzle 7
94 test_puzzle_7:-
95     puzzle_7(Board),
96     solve_puzzle(Board),
97     Solution = [
98         [0,2,0,1,1,2,0],
99         [2,1,0,1,2,0,0],
100        [0,1,2,0,1,0,2],
101        [2,0,0,2,0,1,1],
102        [0,0,2,0,2,1,1],
103        [1,2,1,0,0,2,0],
104        [1,0,1,2,0,0,2]
105    ],
106    Board == Solution.
107
108 % test for puzzle 8
109 test_puzzle_8:-
110     puzzle_8(Board),
111     solve_puzzle(Board),
112     Solution = [
113         [1,0,1,2,0,0,2],
114         [1,2,1,0,0,2,0],
115         [0,1,2,0,1,0,2],
116         [2,0,0,2,1,0,1],
117         [0,0,2,0,2,1,1],
118         [2,1,0,1,0,2,0],
119         [0,2,0,1,2,1,0]
120     ],
121     Board == Solution.
122
123 % test for puzzle 9
124 test_puzzle_9:-
125     puzzle_9(Board),
126     solve_puzzle(Board),
127     Solution = [
128         [0,2,1,0,1,2,0],
129         [1,1,2,0,2,0,0],
130         [2,0,1,0,1,0,2],
131         [1,1,0,2,0,2,0],
132         [0,0,2,1,0,1,2],
133         [2,0,0,1,2,0,1],
134         [0,2,0,2,0,1,1]
135     ],
136     Board == Solution.
137
138 % test for puzzle 10
139 test_puzzle_10:-
140     puzzle_10(Board),
141     solve_puzzle(Board),
142     Solution = [
143         [1,2,0,1,0,0,2],
144         [2,0,0,1,2,1,0],
145         [0,0,2,0,1,2,1],
146         [0,2,0,2,1,1,0],
147         [1,1,2,0,2,0,0],
148         [0,1,1,2,0,0,2],
149         [2,0,1,0,0,2,1]
150     ],
151     Board == Solution.
152
153 % test for puzzle 11
154 test_puzzle_11:-
155     puzzle_11(Board),
156     solve_puzzle(Board),
157     Solution = [
158         [1,2,0,0,1,0,2],
159         [0,0,2,0,1,2,1],
160         [2,0,0,2,0,1,1],
161         [1,1,2,0,2,0,0],
162         [0,2,0,1,0,1,2],
163         [0,1,1,2,0,2,0],
164         [2,0,1,1,2,0,0]
165     ],
166     Board == Solution.
167
168 % test for puzzle 12
169 test_puzzle_12:-
170     puzzle_12(Board),
171     solve_puzzle(Board),
172     Solution = [
173         [0,0,2,1,0,1,2],
174         [0,2,0,2,0,1,1],
175         [2,0,0,1,2,0,1],
176         [0,2,1,0,1,2,0],
177         [1,0,1,2,0,0,2],
178         [2,1,0,0,1,2,0],
179         [1,1,2,0,2,0,0]
180     ],
181     Board == Solution.
182
183 % test for puzzle 13
184 test_puzzle_13:-
185     puzzle_13(Board),
186     solve_puzzle(Board),
187     Solution = [
188         [0,1,2,0,1,0,2],
189         [2,1,0,0,1,2,0],
190         [0,2,0,2,0,1,1],
191         [0,0,2,0,2,1,1],

```

```

192         [1,0,1,2,0,0,2],
193         [2,0,1,1,2,0,0],
194         [1,2,0,1,0,2,0]
195     ],
196     Board == Solution.

```

Listing 19. solver\_tests.pl

```

1  % =====
2  % Puzzles available @ https://www2.stetson.edu/~
   % efriedma/puzzle/closefar/
3  % 0 - EMPTY
4  % 1 - CLOSE
5  % 2 - FAR
6  % =====
7
8  % =====
9  % 6x6 puzzles
10 % =====
11 puzzle_1([
12     [2, _, _, _, _, _],
13     [_, _, _, _, 2, _],
14     [_, _, _, 1, _, _],
15     [_, _, _, _, _, 1],
16     [_, 2, _, _, _, _],
17     [_, _, 2, _, _, _]
18 ]).
19
20 puzzle_2([
21     [_, _, _, _, 2, _],
22     [_, _, 1, _, _, _],
23     [2, _, _, _, _, _],
24     [_, _, _, 1, _, _],
25     [_, 2, _, _, _, _],
26     [_, _, _, _, _, 1]
27 ]).
28
29 puzzle_3([
30     [_, 1, _, _, _, _],
31     [_, _, _, _, 2, _],
32     [_, _, 2, _, _, _],
33     [_, _, _, 1, _, _],
34     [2, _, _, _, _, _],
35     [_, _, _, _, 1, _]
36 ]).
37
38 puzzle_4([
39     [_, 2, _, _, _, _],
40     [_, _, 2, _, _, _],
41     [_, _, _, _, 2, _],
42     [_, _, _, _, 2, _],
43     [_, _, _, 2, _, _],
44     [2, _, _, _, _, _]
45 ]).
46
47 puzzle_5([
48     [_, 1, _, _, _, _],
49     [1, _, _, _, _, _],
50     [_, _, _, 2, _, _],
51     [_, _, _, _, 2, _],
52     [_, _, _, _, 2, _],
53     [_, _, 2, _, _, _]
54 ]).
55
56 % =====
57 % 7x7 puzzles
58 % =====
59 puzzle_6([
60     [_, _, _, _, 1, _, _],
61     [_, _, _, _, 1, _],
62     [_, _, _, 2, _, _],
63     [_, 1, _, _, _, _],
64     [_, _, 2, _, _, _],
65     [_, _, _, _, _, 2],

```

```

66     [2, _, _, _, _, _]
67 ]).
68
69 puzzle_7([
70     [_, _, _, _, 1, _, _],
71     [_, 1, _, _, _, _],
72     [_, _, 2, _, _, _],
73     [2, _, _, _, _, _],
74     [_, _, _, _, _, 1],
75     [_, _, _, _, 2, _],
76     [_, _, _, 2, _, _]
77 ]).
78
79 puzzle_8([
80     [1, _, _, _, _, _],
81     [_, 2, _, _, _, _],
82     [_, _, _, 1, _, _],
83     [_, _, _, _, _, 1],
84     [_, _, 2, _, _, _],
85     [_, _, _, _, 2, _],
86     [_, _, 1, _, _, _]
87 ]).
88
89 puzzle_9([
90     [_, _, _, _, 1, _, _],
91     [_, _, 2, _, _, _],
92     [2, _, _, _, _, _],
93     [_, 1, _, _, _, _],
94     [_, _, _, 1, _, _],
95     [_, _, 1, _, _, _],
96     [_, _, _, _, _, 1]
97 ]).
98
99 puzzle_10([
100     [1, _, _, _, _, _],
101     [_, _, 1, _, _, _],
102     [_, _, _, _, 2, _],
103     [_, _, _, 1, _, _],
104     [_, 2, _, _, _, _],
105     [_, 1, _, _, _, _],
106     [_, _, _, _, _, 1]
107 ]).
108
109 puzzle_11([
110     [_, 2, _, _, _, _],
111     [_, _, _, _, _, 1],
112     [2, _, _, _, _, _],
113     [_, _, _, 2, _, _],
114     [_, _, _, _, 1, _],
115     [_, _, 2, _, _, _],
116     [_, 1, _, _, _, _]
117 ]).
118
119 puzzle_12([
120     [_, _, 2, _, _, _],
121     [_, _, 2, _, _, _],
122     [_, _, _, _, 1, _],
123     [_, _, _, _, 2, _],
124     [1, _, _, _, _, _],
125     [_, _, _, 1, _, _],
126     [_, 1, _, _, _, _]
127 ]).
128
129 puzzle_13([
130     [_, _, _, 1, _, _],
131     [2, _, _, _, _, _],
132     [_, 2, _, _, _, _],
133     [_, _, _, 1, _, _],
134     [_, _, _, _, 2, _],
135     [_, 1, _, _, _, _],
136     [_, _, 1, _, _, _]
137 ]).
138
139 % =====

```

```

140 % Randomly Generated Puzzles
141 % using generate_random_puzzle predicate
142 % =====
143 % 6x6
144 puzzle_6_1([
145     [1,_,_,_,_,_],
146     [_,_,2,_,_,_],
147     [_,_,_,_,2,_,_],
148     [_,2,_,_,_,_],
149     [_,_,_,_,1,_,_],
150     [_,_,_,2,_,_,_]
151 ]).
152 puzzle_6_2([
153     [_,1,_,_,_,_],
154     [_,_,2,_,_,_],
155     [1,_,_,_,_,_],
156     [_,_,_,_,2,_,_],
157     [_,_,2,_,_,_],
158     [_,_,_,_,1,_,_]
159 ]).
160 puzzle_6_3([
161     [_,_,_,_,2,_,_],
162     [_,_,1,_,_,_],
163     [_,1,_,_,_,_],
164     [_,_,_,_,1,_,_],
165     [_,_,_,1,_,_,_],
166     [2,_,_,_,_,_]
167 ]).
168 puzzle_6_4([
169     [_,_,_,_,2,_,_],
170     [_,2,_,_,_,_],
171     [_,_,2,_,_,_],
172     [2,_,_,_,_,_],
173     [_,_,2,_,_,_],
174     [_,_,_,_,1,_,_]
175 ]).
176 puzzle_6_5([
177     [_,_,1,_,_,_],
178     [1,_,_,_,_,_],
179     [_,_,2,_,_,_],
180     [_,_,_,_,1,_,_],
181     [_,2,_,_,_,_],
182     [_,_,_,_,2,_,_]
183 ]).
184 puzzle_6_6([
185     [2,_,_,_,_,_],
186     [_,2,_,_,_,_],
187     [_,_,_,_,1,_,_],
188     [_,_,2,_,_,_],
189     [_,_,2,_,_,_],
190     [_,_,_,_,2,_,_]
191 ]).
192 puzzle_6_7([
193     [_,1,_,_,_,_],
194     [_,_,1,_,_,_],
195     [2,_,_,_,_,_],
196     [_,_,_,1,_,_,_],
197     [_,_,_,2,_,_,_],
198     [_,_,_,_,1,_,_]
199 ]).
200 puzzle_6_8([
201     [2,_,_,_,_,_],
202     [_,1,_,_,_,_],
203     [_,_,1,_,_,_],
204     [_,_,_,_,1,_,_],
205     [_,_,_,1,_,_,_],
206     [_,_,_,_,1,_,_]
207 ]).
208 puzzle_6_9([
209     [_,_,_,1,_,_,_],
210     [_,_,_,2,_,_,_],
211     [_,_,_,_,2,_,_],
212     [1,_,_,_,_,_],
213     [_,2,_,_,_,_,_]
214     [_,_,2,_,_,_,_]
215 ]).
216 puzzle_6_10([
217     [_,_,_,_,2,_,_],
218     [_,_,_,1,_,_,_],
219     [2,_,_,_,_,_,_],
220     [_,1,_,_,_,_,_],
221     [_,_,_,2,_,_,_],
222     [_,_,1,_,_,_,_]
223 ]).
224
225 % 7x7
226 puzzle_7_1([
227     [_,_,_,2,_,_,_,_],
228     [_,_,_,_,2,_,_,_],
229     [2,_,_,_,_,_,_,_],
230     [_,2,_,_,_,_,_,_],
231     [_,_,1,_,_,_,_,_],
232     [_,_,_,_,2,_,_,_],
233     [_,_,_,2,_,_,_,_]
234 ]).
235 puzzle_7_2([
236     [_,_,_,_,2,_,_,_],
237     [_,_,2,_,_,_,_,_],
238     [_,2,_,_,_,_,_,_],
239     [2,_,_,_,_,_,_,_],
240     [_,_,2,_,_,_,_,_],
241     [_,_,_,_,_,1,_,_],
242     [_,_,_,_,1,_,_,_]
243 ]).
244 puzzle_7_3([
245     [_,_,_,_,1,_,_,_],
246     [_,_,_,_,_,1,_,_],
247     [2,_,_,_,_,_,_,_],
248     [_,_,2,_,_,_,_,_],
249     [_,_,_,_,2,_,_,_],
250     [_,1,_,_,_,_,_,_],
251     [_,_,2,_,_,_,_,_]
252 ]).
253 puzzle_7_4([
254     [_,_,_,_,1,_,_,_],
255     [_,_,1,_,_,_,_,_],
256     [_,_,_,_,2,_,_,_],
257     [_,_,_,2,_,_,_,_],
258     [2,_,_,_,_,_,_,_],
259     [_,_,_,_,_,1,_,_],
260     [_,2,_,_,_,_,_,_]
261 ]).
262 puzzle_7_5([
263     [_,1,_,_,_,_,_,_],
264     [_,_,2,_,_,_,_,_],
265     [_,_,_,_,2,_,_,_],
266     [1,_,_,_,_,_,_,_],
267     [_,_,_,_,1,_,_,_],
268     [_,_,_,_,_,1,_,_],
269     [_,_,2,_,_,_,_,_]
270 ]).
271 puzzle_7_6([
272     [_,_,_,1,_,_,_,_],
273     [_,_,_,_,2,_,_,_],
274     [_,_,1,_,_,_,_,_],
275     [2,_,_,_,_,_,_,_],
276     [_,_,_,_,1,_,_,_],
277     [_,2,_,_,_,_,_,_],
278     [_,_,_,_,_,1,_,_]
279 ]).
280 puzzle_7_7([
281     [2,_,_,_,_,_,_,_],
282     [_,_,_,_,_,2,_,_],
283     [_,2,_,_,_,_,_,_],
284     [_,_,1,_,_,_,_,_],
285     [_,_,_,_,2,_,_,_],
286     [_,_,2,_,_,_,_,_],
287     [_,_,_,_,2,_,_,_]

```

```

288 ]).
289 puzzle_7_8([
290     [_,_ ,2,_ ,_ ],
291     [_,_ ,_ ,_ ,2],
292     [_,_ ,_ ,2,_ ],
293     [_,_ ,_ ,1,_ ],
294     [2,_ ,_ ,_ ,_ ],
295     [_ ,2,_ ,_ ,_ ],
296     [_ ,2,_ ,_ ,_ ]
297 ]).
298 puzzle_7_9([
299     [2,_ ,_ ,_ ,_ ,_ ],
300     [_ ,2,_ ,_ ,_ ,_ ],
301     [_ ,_ ,_ ,2,_ ],
302     [_ ,_ ,_ ,_ ,2],
303     [_ ,1,_ ,_ ,_ ,_ ],
304     [_ ,_ ,_ ,2,_ ],
305     [_ ,_ ,2,_ ,_ ,_ ]
306 ]).
307 puzzle_7_10([
308     [1,_ ,_ ,_ ,_ ,_ ],
309     [_ ,1,_ ,_ ,_ ,_ ],
310     [_ ,_ ,_ ,2,_ ],
311     [_ ,2,_ ,_ ,_ ,_ ],
312     [_ ,_ ,_ ,_ ,1],
313     [_ ,_ ,2,_ ,_ ,_ ],
314     [_ ,_ ,_ ,_ ,2_]
315 ]).
316
317 % 8x8
318 puzzle_8_1([
319     [_ ,1,_ ,_ ,_ ,_ ],
320     [_ ,_ ,_ ,_ ,1],
321     [_ ,_ ,_ ,1,_ ,_ ],
322     [2,_ ,_ ,_ ,_ ,_ ],
323     [_ ,_ ,2,_ ,_ ,_ ],
324     [_ ,_ ,_ ,2,_ ],
325     [_ ,_ ,_ ,1,_ ],
326     [_ ,1,_ ,_ ,_ ,_ ]
327 ]).
328 puzzle_8_2([
329     [_ ,1,_ ,_ ,_ ,_ ,_ ],
330     [1,_ ,_ ,_ ,_ ,_ ,_ ],
331     [_ ,_ ,1,_ ,_ ,_ ,_ ],
332     [_ ,1,_ ,_ ,_ ,_ ,_ ],
333     [_ ,_ ,_ ,2,_ ,_ ,_ ],
334     [_ ,_ ,_ ,1,_ ,_ ,_ ],
335     [_ ,_ ,_ ,_ ,1],
336     [_ ,_ ,_ ,_ ,_ ,1]
337 ]).
338 puzzle_8_3([
339     [2,_ ,_ ,_ ,_ ,_ ,_ ],
340     [_ ,2,_ ,_ ,_ ,_ ,_ ],
341     [_ ,_ ,_ ,_ ,2,_ ],
342     [_ ,_ ,_ ,_ ,_ ,2],
343     [_ ,_ ,2,_ ,_ ,_ ,_ ],
344     [_ ,_ ,_ ,2,_ ,_ ,_ ],
345     [_ ,_ ,_ ,_ ,2,_ ,_ ],
346     [_ ,2,_ ,_ ,_ ,_ ,_ ]
347 ]).
348 puzzle_8_4([
349     [_ ,_ ,_ ,1,_ ,_ ,_ ],
350     [1,_ ,_ ,_ ,_ ,_ ,_ ],
351     [_ ,_ ,_ ,_ ,2,_ ],
352     [_ ,1,_ ,_ ,_ ,_ ,_ ],
353     [_ ,_ ,_ ,_ ,_ ,2],
354     [_ ,_ ,2,_ ,_ ,_ ,_ ],
355     [_ ,2,_ ,_ ,_ ,_ ,_ ],
356     [_ ,_ ,_ ,_ ,1,_ ,_ ]
357 ]).
358 puzzle_8_5([
359     [_ ,1,_ ,_ ,_ ,_ ,_ ],
360     [_ ,_ ,1,_ ,_ ,_ ,_ ],
361     [_ ,_ ,_ ,1,_ ,_ ,_ ],
362     [_ ,_ ,_ ,_ ,2,_ ,_ ],
363     [1,_ ,_ ,_ ,_ ,_ ,_ ],
364     [_ ,_ ,_ ,_ ,_ ,1,_ ],
365     [_ ,_ ,_ ,_ ,_ ,2],
366     [_ ,2,_ ,_ ,_ ,_ ,_ ]
367 ]).
368 puzzle_8_6([
369     [_ ,2,_ ,_ ,_ ,_ ,_ ],
370     [1,_ ,_ ,_ ,_ ,_ ,_ ],
371     [_ ,_ ,_ ,_ ,2,_ ],
372     [_ ,_ ,_ ,_ ,2,_ ,_ ],
373     [_ ,_ ,2,_ ,_ ,_ ,_ ],
374     [_ ,_ ,_ ,_ ,_ ,1],
375     [_ ,2,_ ,_ ,_ ,_ ,_ ],
376     [_ ,_ ,_ ,1,_ ,_ ,_ ]
377 ]).
378 puzzle_8_7([
379     [2,_ ,_ ,_ ,_ ,_ ,_ ],
380     [_ ,_ ,_ ,_ ,2,_ ,_ ],
381     [_ ,_ ,_ ,_ ,_ ,2,_ ],
382     [_ ,2,_ ,_ ,_ ,_ ,_ ],
383     [_ ,2,_ ,_ ,_ ,_ ,_ ],
384     [_ ,_ ,_ ,_ ,_ ,1],
385     [_ ,_ ,2,_ ,_ ,_ ,_ ],
386     [_ ,_ ,_ ,2,_ ,_ ,_ ]
387 ]).
388 puzzle_8_8([
389     [2,_ ,_ ,_ ,_ ,_ ,_ ],
390     [_ ,_ ,_ ,_ ,2,_ ],
391     [_ ,_ ,_ ,_ ,2,_ ,_ ],
392     [_ ,_ ,_ ,_ ,_ ,1],
393     [_ ,_ ,1,_ ,_ ,_ ,_ ],
394     [_ ,_ ,_ ,2,_ ,_ ,_ ],
395     [_ ,2,_ ,_ ,_ ,_ ,_ ],
396     [_ ,2,_ ,_ ,_ ,_ ,_ ]
397 ]).
398 puzzle_8_9([
399     [_ ,_ ,2,_ ,_ ,_ ,_ ],
400     [_ ,2,_ ,_ ,_ ,_ ,_ ],
401     [_ ,_ ,_ ,_ ,2],
402     [_ ,2,_ ,_ ,_ ,_ ,_ ],
403     [_ ,_ ,_ ,1,_ ,_ ,_ ],
404     [_ ,_ ,_ ,_ ,1,_ ,_ ],
405     [_ ,_ ,_ ,_ ,_ ,1,_ ],
406     [2,_ ,_ ,_ ,_ ,_ ,_ ]
407 ]).
408 puzzle_8_10([
409     [1,_ ,_ ,_ ,_ ,_ ,_ ],
410     [_ ,_ ,_ ,_ ,2,_ ],
411     [_ ,_ ,_ ,_ ,1,_ ,_ ],
412     [_ ,_ ,1,_ ,_ ,_ ,_ ],
413     [_ ,1,_ ,_ ,_ ,_ ,_ ],
414     [_ ,_ ,_ ,2,_ ,_ ,_ ],
415     [_ ,_ ,_ ,_ ,_ ,1],
416     [_ ,2,_ ,_ ,_ ,_ ,_ ]
417 ]).
418
419 % 9x9
420 puzzle_9_1([
421     [2,_ ,_ ,_ ,_ ,_ ,_ ,_ ],
422     [_ ,_ ,_ ,_ ,_ ,2],
423     [_ ,_ ,1,_ ,_ ,_ ,_ ,_ ],
424     [_ ,1,_ ,_ ,_ ,_ ,_ ,_ ],
425     [_ ,_ ,_ ,_ ,1,_ ,_ ,_ ],
426     [_ ,2,_ ,_ ,_ ,_ ,_ ,_ ],
427     [_ ,_ ,_ ,_ ,_ ,2,_ ],
428     [_ ,_ ,_ ,_ ,_ ,2,_ ,_ ],
429     [_ ,_ ,_ ,2,_ ,_ ,_ ,_ ]
430 ]).
431 puzzle_9_2([
432     [2,_ ,_ ,_ ,_ ,_ ,_ ,_ ],
433     [_ ,_ ,_ ,_ ,2,_ ,_ ,_ ],
434     [_ ,_ ,1,_ ,_ ,_ ,_ ,_ ],
435     [_ ,_ ,1,_ ,_ ,_ ,_ ,_ ],

```

```

436     [_,_,_,_,_,_2,_],
437     [_,_,_,_1,_,_,_],
438     [_2,_,_,_,_,_,_],
439     [_,_,_,_,_,_1],
440     [_,_,_,_,_2,_,_]
441 ]).
442 puzzle_9_3([
443     [_,_,_1,_,_,_,_],
444     [_,_1,_,_,_,_,_],
445     [_,_,_,_,_,_2],
446     [_,_,_,_1,_,_,_],
447     [_,_,_,_1,_,_,_],
448     [_2,_,_,_,_,_,_],
449     [_,_,_,_,_2,_,_],
450     [_,_,_,_,_2,_,_],
451     [2,_,_,_,_,_,_]
452 ]).
453 puzzle_9_4([
454     [_,_,_,_1,_,_,_],
455     [_,_,_1,_,_,_,_],
456     [_1,_,_,_,_,_,_],
457     [2,_,_,_,_,_,_],
458     [_,_,_,_,_2,_,_],
459     [_,_1,_,_,_,_,_],
460     [_,_,_,_,_2,_,_],
461     [_,_,_,_1,_,_,_],
462     [_,_,_,_,_,_1]
463 ]).
464 puzzle_9_5([
465     [_,_,_,_1,_,_,_],
466     [_2,_,_,_,_,_,_],
467     [1,_,_,_,_,_,_],
468     [_,_,_,_1,_,_,_],
469     [_,_,_,_,_1,_,_],
470     [_,_,_2,_,_,_,_],
471     [_,_,_,_,_2,_,_],
472     [_,_,_,_,_1,_],
473     [_,_2,_,_,_,_,_]
474 ]).
475 puzzle_9_6([
476     [_1,_,_,_,_,_,_],
477     [1,_,_,_,_,_,_],
478     [_,_,_1,_,_,_,_],
479     [_,_,_,_2,_,_,_],
480     [_,_,_,_1,_,_,_],
481     [_,_2,_,_,_,_,_],
482     [_,_,_,_,_2,_],
483     [_,_,_,_,_1,_,_],
484     [_,_,_,_,_,_1]
485 ]).
486 puzzle_9_7([
487     [1,_,_,_,_,_,_],
488     [_,_,_,_1,_,_,_],
489     [_,_,_,_2,_,_,_],
490     [_,_1,_,_,_,_,_],
491     [_,_,_,_1,_,_,_],
492     [_2,_,_,_,_,_,_],
493     [_,_,_1,_,_,_,_],
494     [_,_,_,_,_1,_],
495     [_,_,_,_,_,_1]
496 ]).
497 puzzle_9_8([
498     [_,_2,_,_,_,_,_],
499     [_,_,_,_,_2],
500     [_2,_,_,_,_,_,_],
501     [_,_,_,_2,_,_,_],
502     [2,_,_,_,_,_,_],
503     [_,_,_,_2,_,_,_],
504     [_,_,_2,_,_,_,_],
505     [_,_,_2,_,_,_,_],
506     [_,_,_,_,_1,_]
507 ]).
508 puzzle_9_9([
509     [1,_,_,_,_,_,_],
510     [_,_,_,_,_1,_,_],
511     [_,_,_,_,_,_1],
512     [_1,_,_,_,_,_,_],
513     [_,_2,_,_,_,_,_],
514     [_,_,_,_,_1,_],
515     [_,_,_,_1,_,_,_],
516     [_,_,_1,_,_,_,_],
517     [_,_,_2,_,_,_,_]
518 ]).
519 puzzle_9_10([
520     [_,_,_2,_,_,_,_],
521     [_1,_,_,_,_,_,_],
522     [_,_,_,_1,_,_,_],
523     [_,_,_,_,_1,_,_],
524     [_,_2,_,_,_,_,_],
525     [1,_,_,_,_,_,_],
526     [_,_,_,_,_,_1],
527     [_,_,_,_,_1,_],
528     [_,_,_2,_,_,_,_]
529 ]).
530
531 % 10x10
532 puzzle_10_1([
533     [_,_,_,_,_2,_,_],
534     [_,_,_,_,_2,_,_],
535     [_,_1,_,_,_,_,_],
536     [_,_,_1,_,_,_,_],
537     [2,_,_,_,_,_,_],
538     [_1,_,_,_,_,_,_],
539     [_,_,_2,_,_,_,_],
540     [_,_,_2,_,_,_,_],
541     [_,_,_,_,_2,_],
542     [_,_,_,_,_,_1]
543 ]).
544 puzzle_10_2([
545     [_,_,_,_,_2,_,_],
546     [_,_,_,_2,_,_,_],
547     [_,_,_,_1,_,_,_],
548     [_,_,_2,_,_,_,_],
549     [_2,_,_,_,_,_,_],
550     [_,_2,_,_,_,_,_],
551     [_,_2,_,_,_,_,_],
552     [2,_,_,_,_,_,_],
553     [_,_,_,_,_1],
554     [_,_,_,_,_1,_]
555 ]).
556 puzzle_10_3([
557     [_,_,_,_,_1,_,_],
558     [_1,_,_,_,_,_,_],
559     [_,_2,_,_,_,_,_],
560     [_,_,_,_,_2,_],
561     [_,_,_,_,_1,_],
562     [_,_,_,_,_1],
563     [_,_2,_,_,_,_,_],
564     [_,_,_2,_,_,_,_],
565     [1,_,_,_,_,_,_],
566     [_,_,_,_2,_,_,_]
567 ]).
568 puzzle_10_4([
569     [_1,_,_,_,_,_,_],
570     [_,_2,_,_,_,_,_],
571     [1,_,_,_,_,_,_],
572     [_,_1,_,_,_,_,_],
573     [_,_,_2,_,_,_,_],
574     [_,_,_,_2,_,_],
575     [_,_,_,_1,_,_],
576     [_,_,_2,_,_,_],
577     [_,_,_,_,_2],
578     [_,_,_,_,_1,_]
579 ]).
580 puzzle_10_5([
581     [1,_,_,_,_,_,_],
582     [_,_,_1,_,_,_,_],
583     [_1,_,_,_,_,_,_],

```



```

584     [_,_,_,_,_,_,_,_,_],
585     [_,_,_,_,_,2,_,_,_],
586     [_,_,_,_,_,_,_,2,_],
587     [_,_,_,_,_,_,2,_,_],
588     [_,_,_,_,_,_,_,_,2,_],
589     [_,_,_,2,_,_,_,_,_],
590     [_,_,2,_,_,_,_,_,_]
591 ]).
592 puzzle_10_6([
593     [_,_,_,_,1,_,_,_,_],
594     [_,2,_,_,_,_,_,_,_],
595     [2,_,_,_,_,_,_,_],
596     [_,_,_,_,_,_,2,_],
597     [_,_,_,_,_,_,_,2,_],
598     [_,_,_,_,_,1,_,_,_],
599     [_,_,_,_,1,_,_,_,_],
600     [_,_,_,_,_,_,1,_,_],
601     [_,_,_,_,_,_,_,_,1],
602     [_,_,2,_,_,_,_,_,_]
603 ]).
604 puzzle_10_7([
605     [1,_,_,_,_,_,_,_,_],
606     [_,_,_,_,_,_,_,1,_],
607     [_,_,_,_,_,1,_,_,_],
608     [_,_,_,_,_,_,_,_,2],
609     [_,2,_,_,_,_,_,_,_],
610     [_,_,_,2,_,_,_,_,_],
611     [_,_,2,_,_,_,_,_,_],
612     [_,_,_,_,_,_,_,1,_],
613     [_,_,_,_,1,_,_,_,_],
614     [_,_,_,_,_,_,2,_,_]
615 ]).
616 puzzle_10_8([
617     [1,_,_,_,_,_,_,_,_],
618     [_,_,_,_,1,_,_,_,_],
619     [_,1,_,_,_,_,_,_,_],
620     [_,_,_,_,_,_,_,_,2],
621     [_,_,_,_,_,2,_,_,_],
622     [_,_,_,_,_,_,2,_,_],
623     [_,_,_,_,_,_,2,_,_],
624     [_,_,_,_,_,_,_,2,_],
625     [_,_,_,2,_,_,_,_,_],
626     [_,_,2,_,_,_,_,_,_]
627 ]).
628 puzzle_10_9([
629     [_,_,_,_,1,_,_,_,_],
630     [_,2,_,_,_,_,_,_,_],
631     [2,_,_,_,_,_,_,_],
632     [_,_,_,_,_,_,2,_],
633     [_,_,_,_,_,_,_,2,_],
634     [_,_,_,_,_,1,_,_,_],
635     [_,_,_,_,1,_,_,_,_],
636     [_,_,_,_,_,_,1,_,_],
637     [_,_,_,_,_,_,_,_,1],
638     [_,_,2,_,_,_,_,_,_]
639 ]).
640 puzzle_10_10([
641     [1,_,_,_,_,_,_,_,_],
642     [_,_,_,_,_,_,_,1,_],
643     [_,_,_,_,_,1,_,_,_],
644     [_,_,_,_,_,_,_,_,2],
645     [_,2,_,_,_,_,_,_,_],
646     [_,_,_,2,_,_,_,_,_],
647     [_,_,2,_,_,_,_,_,_],
648     [_,_,_,_,_,_,_,1,_],
649     [_,_,_,_,1,_,_,_,_],
650     [_,_,_,_,_,_,2,_,_]
651 ]).

```

Listing 20. puzzles.pl