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 3rd 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

```
solve_puzzle(Board, Runtime):-
    % --- START STATISTICS ---
    %statistics - runtime calculation (ms)
    statistics(runtime, [Start|_]),
    % --- DECISION VARIABLES ---
    % length is already defined by Board
    apply_solver_domain(Board),
    % --- CONSTRAINTS ---
    apply_solver_occurrences_constraints(Board),
    apply_solver_distance_constraints(Board),
    % --- LABELING ---
    % flatten Board into a 1 dimensional list
    append(Board, FlatBoard),
    labeling([median], FlatBoard),
    % --- END STATISTICS ---
    %statistics - runtime calculation (ms)
    statistics(runtime, [Stop|_]),
    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.

```
puzzle([
    [_, _, _, 2, _, _, _],
    [_, _, _, _, _, _, 2],
    [_, _, _, _, _, 1, _],
    [_, _, 1, _, _, _, _],
    [_, 1, _, _, _, _, _],
    [2, _, _, _, _, _, _],
    [_, _, _, _, 1, _, _]
]).
```

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) Constrains: 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-NumberOfZeros, 1-2, 2-2]*, 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.

```
get_distance_between_elements(List, Element,
    Distance):-
    % find indexes of Element in List
    element(FirstIndex, List, Element),
    element(SecondIndex, List, Element),
    % indexes have to be unique
    FirstIndex #\= SecondIndex,
    % first index has to come first
    FirstIndex #< SecondIndex,
    % calculate distance
    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.

```
generate_random_puzzle(Size, PuzzleBoard, Runtime):-
    % --- START STATISTICS ---
    %statistics - runtime calculation (ms)
    statistics(runtime, [Start|_]),
    % generate random solution
    generate_random_solution(Size, SolutionBoard),
    % find hints from solution board
    find_hints(SolutionBoard, HintColumns,
    HintValues),
    % generate empty board
    generate_board(Size, PuzzleBoard),
    % populate puzzle from hints found
    populate_puzzle(PuzzleBoard, HintColumns,
    HintValues),
    % --- END STATISTICS ---
    %statistics - runtime calculation (ms)
    statistics(runtime, [Stop|_]),
    Runtime is Stop - Start.
```

Listing 4. Puzzle generator predicate

```
find_hints(SolutionBoard, HintColumns, HintValues):-
    % --- DECISION VARIABLES ---
    length(SolutionBoard, NumberOfColumns),
    % hintColumns' domain is [1, Board's number of
    columns]
    length(HintColumns, NumberOfColumns),
    domain(HintColumns, 1, NumberOfColumns),
    % hintValues's domain is 1 or 2 (C or F)
    length(HintValues, NumberOfColumns),
    domain(HintValues, 1, 2),
    % --- CONSTRAINTS ---
    % only one hint per row and column
    all_distinct(HintColumns),
    apply_generator_occurrences_constraints(
    SolutionBoard, HintColumns, HintValues),
    % --- LABELING ---
    % flatten Board into a 1 dimensional list
    append(HintColumns, HintValues, Hints),
    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.

```
HintColumns([4, 7, 6, 3, 2, 1, 5]).
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, 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) Constrains: 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.

```
random_value(Var, _Rest, BB, BB1):-
    % get finite domain set
    fd_set(Var, Set),
    % trasform it to list and choose a random value
    fdset_to_list(Set, List),
    random_member(RandomValue, List),
    (
        first_bound(BB, BB1), Var #= RandomValue
        ;
        later_bound(BB, BB1), Var #\= RandomValue
    ).
```

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.

```
print_board([Line | Board]):-
    length(Line, Width),
    write('|'),
    print_horizontal_separator(Width),
    nl,
    length([Line | Board], Height),
    print_matrix([Line | Board], Height),
    nl.
```

Listing 8. print_board predicate

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

```
print_matrix([], 0).
print_matrix([Line | Board], N):-
    write('| '),
    print_line(Line),
    nl,
    length(Line, Length),
    N1 is N - 1,
    write('|'),
    print_horizontal_separator(Length),
    nl,
    print_matrix(Board, N1).
```

Listing 10. print_matrix predicate

```
print_line([]).
print_line([Cell | Line]):-
    print_cell(Cell),
    write(' | '),
    print_line(Line).

print_cell(Cell):-
    translate(Cell, Char),
    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

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 [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 [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 [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 [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].

[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

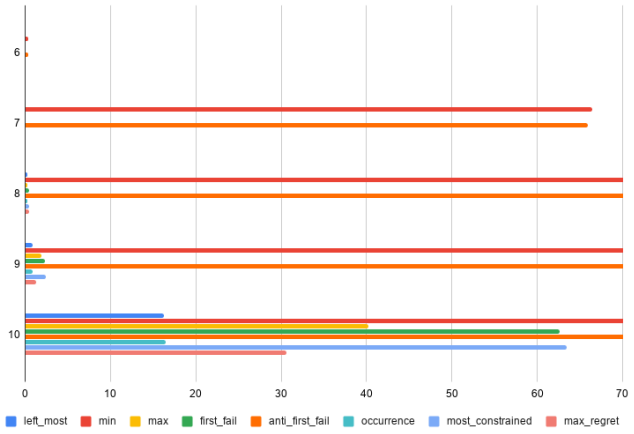

Fig. 3. Variable Choice Heuristic Tests - Time in seconds

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

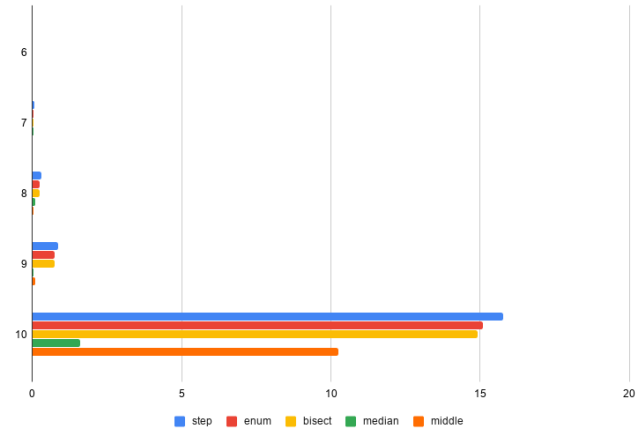

Fig. 4. Value Choice Heuristic Tests - Time in seconds

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

```
:- consult('generator.pl').
:- consult('solver.pl').
:- consult('display.pl').

play(Size):-
    % generate random puzzle with given size
    generate_random_puzzle(Size, PuzzleBoard,
    GeneratorRuntime),
    % print puzzle
    print_board(PuzzleBoard),
    % print generator runtime
    format(' > Generator runtime: ~3d s~n', [
    GeneratorRuntime]),
    % replace 0's with _
    replace_zeros_matrix(PuzzleBoard, SolutionBoard)
    ,
    % solve puzzle
    solve_puzzle(SolutionBoard, SolverRuntime),
    % print solved puzzle
    print_board(SolutionBoard),
    % print solver runtime
    format(' > Solver runtime: ~3d s~n', [
    SolverRuntime]).
```

Listing 13. close_or_far.pl

```
:- use_module(library(clpfd)).
:- consult('utils.pl').

/*
0 -> EMPTY
```

```
1 -> CLOSE
2 -> FAR
*/

% ================================================
% Decision Variables
% ================================================
apply_solver_domain([]).
apply_solver_domain([Row | Board]):-
    % (0 - empty, 1 - close, 2 - far)
    domain(Row, 0, 2),
    apply_solver_domain(Board).

% ================================================
% Constraints
% ================================================
% ====== Occurrences constraints ======
apply_solver_occurrences_constraints(Board):-
    % calculate number of 0's (empty)
    % get number of cells per row (same as per col)
    length(Board, NumberOfCellsPerRow),
    % get number of 0's (number of cells per row - (number of 1's + number of 2's))
    NumberOfZeros #= NumberOfCellsPerRow - 4,
    % apply constraints per row
    solver_occurrences_constraints_per_row(Board, NumberOfZeros),
    % apply constraints per colum (with transposed matrix)
    transpose(Board, TransposedBoard),
    solver_occurrences_constraints_per_row(TransposedBoard, NumberOfZeros).

solver_occurrences_constraints_per_row([], _NumberOfZeros).
solver_occurrences_constraints_per_row([Row | Board], NumberOfZeros):-
    global_cardinality(Row, [
        % NumberOfZeros occurrences of 0 (empty) per row
        0-NumberOfZeros,
        % 2 occurrences of 1 (close) per row
        1-2,
        % 2 occurrences of 2 (far) per row
        2-2
    ]),
    % apply constraints to next rows
    solver_occurrences_constraints_per_row(Board, NumberOfZeros).

% ====== Distances ======
apply_solver_distance_constraints(Board):-
    % apply constraints per row
    solver_distance_constraints_per_row(Board),
    % apply constraints per colum (with transposed board matrix)
    transpose(Board, TransposedBoard),
    solver_distance_constraints_per_row(TransposedBoard).

get_distance_between_elements(List, Element, Distance):-
    % find indexes of Element in List
    element(FirstIndex, List, Element),
    element(SecondIndex, List, Element),
    % indexes have to be unique
    FirstIndex #\= SecondIndex,
    % first index has to come first
    FirstIndex #< SecondIndex,
    % calculate distance
    Distance #= SecondIndex - FirstIndex.

solver_distance_constraints_per_row([]).
solver_distance_constraints_per_row([Row | Board]):-
    % get distance between 1's (close)
    get_distance_between_elements(Row, 1, CloseDistance),
    % get distance between 2's (far)
    get_distance_between_elements(Row, 2, FarDistance),
    % apply MAIN constraints
    CloseDistance #< FarDistance,
    % apply constraints to next rows
    solver_distance_constraints_per_row(Board).

% ================================================
% Puzzle Solver
% ================================================
solve_puzzle(Board, Runtime):-
    % --- START STATISTICS ---
    %statistics - runtime calculation (ms)
    statistics(runtime, [Start|_]),
    % --- DECISION VARIABLES ---
    % length is already defined by Board
    apply_solver_domain(Board),
    % --- CONSTRAINTS ---
    apply_solver_occurrences_constraints(Board),
    apply_solver_distance_constraints(Board),
    % --- LABELING ---
    % flatten Board into a 1 dimensional list
    append(Board, FlatBoard),
    labeling([median], FlatBoard),
    % --- END STATISTICS ---
    %statistics - runtime calculation (ms)
    statistics(runtime, [Stop|_]),
    Runtime is Stop - Start.
```

Listing 14. solver.pl

```
:- consult('utils.pl').
:- consult('solver.pl').
:- use_module(library(random)).

% ================================================
% Generate Empty Board
% ================================================
% generate empty board matrix given its size
generate_board(Size, Board):-
    generate_board_aux(Size, Size, Board).

generate_board_aux(_, 0, []).
generate_board_aux(Size, N, [Row | Board]):-
    generate_row(Size, Row),
    NewN is N - 1,
    generate_board_aux(Size, NewN, Board).

% generate empty board row given its size
generate_row(0, []).
generate_row(Size, [_ | Row]):-
    NewSize is Size - 1,
    generate_row(NewSize, Row).

% ================================================
% Populate Puzzle with hints found from Solution
% ================================================
populate_puzzle([], [], []).
populate_puzzle([PuzzleRow | PuzzleBoard], [HintColumnsCell | HintColumns], [HintValuesCell | HintValues]):-
    % write HintValue in HintColumn of PuzzleRow
    nth1(HintColumnsCell, PuzzleRow, HintValuesCell),
    % repeat for next row
    populate_puzzle(PuzzleBoard, HintColumns, HintValues).

% ================================================
% Find hints from Solution
```

```
% ================================================
apply_generator_occurrences_constraints([], [], []).
apply_generator_occurrences_constraints([SolutionRow | SolutionBoard], [HintColumnsCell | HintColumns], [HintValuesCell | HintValues]):-
    % hint column and value have to be consistent with solution row
    element(HintColumnsCell, SolutionRow, HintValuesCell),
    % repeat for next row
    apply_generator_occurrences_constraints(SolutionBoard, HintColumns, HintValues).

find_hints(SolutionBoard, HintColumns, HintValues):-
    % --- DECISION VARIABLES ---
    length(SolutionBoard, NumberOfColumns),
    % hintColumns' domain is [1, Board's number of columns]
    length(HintColumns, NumberOfColumns),
    domain(HintColumns, 1, NumberOfColumns),
    % hintValues's domain is 1 or 2 (C or F)
    length(HintValues, NumberOfColumns),
    domain(HintValues, 1, 2),
    % --- CONSTRAINTS ---
    % only one hint per row and column
    all_distinct(HintColumns),
    apply_generator_occurrences_constraints(SolutionBoard, HintColumns, HintValues),
    % --- LABELING ---
    % flatten Board into a 1 dimensional list
    append(HintColumns, HintValues, Hints),
    labeling([value(random_value)], Hints).

% ================================================
% Puzzle Generator
% ================================================
generate_random_solution(Size, Board):-
    % --- DECISION VARIABLES ---
    % generate empty board
    generate_board(Size, Board),
    % length is already defined by Board
    apply_solver_domain(Board),
    % --- CONSTRAINTS ---
    apply_solver_occurrences_constraints(Board),
    apply_solver_distance_constraints(Board),
    % --- LABELING ---
    % flatten Board into a 1 dimensional list
    append(Board, FlatBoard),
    labeling([value(random_value)], FlatBoard).

generate_random_puzzle(Size, PuzzleBoard, Runtime):-
    % --- START STATISTICS ---
    %statistics - runtime calculation (ms)
    statistics(runtime, [Start|_]),
    % generate random solution
    generate_random_solution(Size, SolutionBoard),
    % find hints from solution board
    find_hints(SolutionBoard, HintColumns, HintValues),
    % generate empty board
    generate_board(Size, PuzzleBoard),
    % populate puzzle from hints found
    populate_puzzle(PuzzleBoard, HintColumns, HintValues),
    % --- END STATISTICS ---
    %statistics - runtime calculation (ms)
    statistics(runtime, [Stop|_]),
    Runtime is Stop - Start.

% ================================================
% Labeling Options
% ================================================
% select random value
random_value(Var, _Rest, BB, BB1):-
    % get finite domain set
    fd_set(Var, Set),
    % trasform it to list and choose a random value
    fdset_to_list(Set, List),
    random_member(RandomValue, List),
    (
        first_bound(BB, BB1), Var #= RandomValue
        ;
        later_bound(BB, BB1), Var #\= RandomValue
    ).
```

Listing 15. generator.pl

```
:- consult('utils.pl').

% ================================================
% Display Board
% ================================================
print_board([Line | Board]):-
    length(Line, Width),
    write('|'),
    print_horizontal_separator(Width),
    nl,
    length([Line | Board], Height),
    print_matrix([Line | Board], Height),
    nl.

print_horizontal_separator(0).
print_horizontal_separator(N):-
    write('---|'),
    N1 is N - 1,
    print_horizontal_separator(N1).

print_matrix([], 0).
print_matrix([Line | Board], N):-
    write('| '),
    print_line(Line),
    nl,
    length(Line, Length),
    N1 is N - 1,
    write('|'),
    print_horizontal_separator(Length),
    nl,
    print_matrix(Board, N1).

print_line([]).
print_line([Cell | Line]):-
    print_cell(Cell),
    write(' | '),
    print_line(Line).

print_cell(Cell):-
    translate(Cell, Char),
    write(Char).
```

Listing 16. display.pl

```
:- use_module(library(lists)).

% ================================================
% translate board atoms into chars
% ================================================
translate(0, '.').
translate(1, 'C').
translate(2, 'F').

% ================================================
% Replace 0's with _
% ================================================
replace_zeros_matrix([], []).
replace_zeros_matrix([Row | Matrix], [NewRow | Rest]):-
    replace_zeros_matrix(Matrix, Rest),
    replace_zeros_row(Row, NewRow).
```

```

replace_zeros_row([], []).
replace_zeros_row([0 | Tail], [_ | Rest]):-
    replace_zeros_row(Tail, Rest).
replace_zeros_row([Head | Tail], [Head | Rest]):-
    replace_zeros_row(Tail, Rest).
```

Listing 17. utils.pl

```
:- consult('solver.pl').
:- consult('puzzles.pl').

% ================================================
% Testing Purposes
% See puzzles in file 'puzzles.pl'
% ================================================
test_solver_6:-
    puzzle_6_1(Board1),
    solve_puzzle(Board1, Runtime1),
    format(' > Puzzle 1 solver runtime: ~3d s~n', [Runtime1]),
    puzzle_6_2(Board2),
    solve_puzzle(Board2, Runtime2),
    format(' > Puzzle 2 solver runtime: ~3d s~n', [Runtime2]),
    puzzle_6_3(Board3),
    solve_puzzle(Board3, Runtime3),
    format(' > Puzzle 3 solver runtime: ~3d s~n', [Runtime3]),
    puzzle_6_4(Board4),
    solve_puzzle(Board4, Runtime4),
    format(' > Puzzle 4 solver runtime: ~3d s~n', [Runtime4]),
    puzzle_6_5(Board5),
    solve_puzzle(Board5, Runtime5),
    format(' > Puzzle 5 solver runtime: ~3d s~n', [Runtime5]),
    puzzle_6_6(Board6),
    solve_puzzle(Board6, Runtime6),
    format(' > Puzzle 6 solver runtime: ~3d s~n', [Runtime6]),
    puzzle_6_7(Board7),
    solve_puzzle(Board7, Runtime7),
    format(' > Puzzle 7 solver runtime: ~3d s~n', [Runtime7]),
    puzzle_6_8(Board8),
    solve_puzzle(Board8, Runtime8),
    format(' > Puzzle 8 solver runtime: ~3d s~n', [Runtime8]),
    puzzle_6_9(Board9),
    solve_puzzle(Board9, Runtime9),
    format(' > Puzzle 9 solver runtime: ~3d s~n', [Runtime9]),
    puzzle_6_10(Board10),
    solve_puzzle(Board10, Runtime10),
    format(' > Puzzle 10 solver runtime: ~3d s~n', [Runtime10]),
    AvgRuntime is (Runtime1 + Runtime2 + Runtime3 + Runtime4 + Runtime5 + Runtime6 + Runtime7 + Runtime8 + Runtime9 + Runtime10) / 10,
    format(' > Average runtime: ~3d s~n', [AvgRuntime]).

test_solver_7:-
    puzzle_7_1(Board1),
    solve_puzzle(Board1, Runtime1),
    format(' > Puzzle 1 solver runtime: ~3d s~n', [Runtime1]),
    puzzle_7_2(Board2),
    solve_puzzle(Board2, Runtime2),
    format(' > Puzzle 2 solver runtime: ~3d s~n', [Runtime2]),
    puzzle_7_3(Board3),
    solve_puzzle(Board3, Runtime3),
    format(' > Puzzle 3 solver runtime: ~3d s~n', [Runtime3]),
    puzzle_7_4(Board4),
    solve_puzzle(Board4, Runtime4),
    format(' > Puzzle 4 solver runtime: ~3d s~n', [Runtime4]),
    puzzle_7_5(Board5),
    solve_puzzle(Board5, Runtime5),
    format(' > Puzzle 5 solver runtime: ~3d s~n', [Runtime5]),
    puzzle_7_6(Board6),
    solve_puzzle(Board6, Runtime6),
    format(' > Puzzle 6 solver runtime: ~3d s~n', [Runtime6]),
    puzzle_7_7(Board7),
    solve_puzzle(Board7, Runtime7),
    format(' > Puzzle 7 solver runtime: ~3d s~n', [Runtime7]),
    puzzle_7_8(Board8),
    solve_puzzle(Board8, Runtime8),
    format(' > Puzzle 8 solver runtime: ~3d s~n', [Runtime8]),
    puzzle_7_9(Board9),
    solve_puzzle(Board9, Runtime9),
    format(' > Puzzle 9 solver runtime: ~3d s~n', [Runtime9]),
    puzzle_7_10(Board10),
    solve_puzzle(Board10, Runtime10),
    format(' > Puzzle 10 solver runtime: ~3d s~n', [Runtime10]),
    AvgRuntime is (Runtime1 + Runtime2 + Runtime3 + Runtime4 + Runtime5 + Runtime6 + Runtime7 + Runtime8 + Runtime9 + Runtime10) / 10,
    format(' > Average runtime: ~3d s~n', [AvgRuntime]).

test_solver_8:-
    puzzle_8_1(Board1),
    solve_puzzle(Board1, Runtime1),
    format(' > Puzzle 1 solver runtime: ~3d s~n', [Runtime1]),
    puzzle_8_2(Board2),
    solve_puzzle(Board2, Runtime2),
    format(' > Puzzle 2 solver runtime: ~3d s~n', [Runtime2]),
    puzzle_8_3(Board3),
    solve_puzzle(Board3, Runtime3),
    format(' > Puzzle 3 solver runtime: ~3d s~n', [Runtime3]),
    puzzle_8_4(Board4),
    solve_puzzle(Board4, Runtime4),
    format(' > Puzzle 4 solver runtime: ~3d s~n', [Runtime4]),
    puzzle_8_5(Board5),
    solve_puzzle(Board5, Runtime5),
    format(' > Puzzle 5 solver runtime: ~3d s~n', [Runtime5]),
    puzzle_8_6(Board6),
    solve_puzzle(Board6, Runtime6),
    format(' > Puzzle 6 solver runtime: ~3d s~n', [Runtime6]),
    puzzle_8_7(Board7),
    solve_puzzle(Board7, Runtime7),
    format(' > Puzzle 7 solver runtime: ~3d s~n', [Runtime7]),
    puzzle_8_8(Board8),
    solve_puzzle(Board8, Runtime8),
    format(' > Puzzle 8 solver runtime: ~3d s~n', [Runtime8]),
    puzzle_8_9(Board9),
    solve_puzzle(Board9, Runtime9),
    format(' > Puzzle 9 solver runtime: ~3d s~n', [Runtime9]),
    puzzle_8_10(Board10),
```

```
    solve_puzzle(Board10, Runtime10),
    format(' > Puzzle 10 solver runtime: ~3d s~n', [Runtime10]),
    AvgRuntime is (Runtime1 + Runtime2 + Runtime3 + Runtime4 + Runtime5 + Runtime6 + Runtime7 + Runtime8 + Runtime9 + Runtime10) / 10,
    format(' > Average runtime: ~3d s~n', [AvgRuntime]).

test_solver_9:-
    puzzle_9_1(Board1),
    solve_puzzle(Board1, Runtime1),
    format(' > Puzzle 1 solver runtime: ~3d s~n', [Runtime1]),
    puzzle_9_2(Board2),
    solve_puzzle(Board2, Runtime2),
    format(' > Puzzle 2 solver runtime: ~3d s~n', [Runtime2]),
    puzzle_9_3(Board3),
    solve_puzzle(Board3, Runtime3),
    format(' > Puzzle 3 solver runtime: ~3d s~n', [Runtime3]),
    puzzle_9_4(Board4),
    solve_puzzle(Board4, Runtime4),
    format(' > Puzzle 4 solver runtime: ~3d s~n', [Runtime4]),
    puzzle_9_5(Board5),
    solve_puzzle(Board5, Runtime5),
    format(' > Puzzle 5 solver runtime: ~3d s~n', [Runtime5]),
    puzzle_9_6(Board6),
    solve_puzzle(Board6, Runtime6),
    format(' > Puzzle 6 solver runtime: ~3d s~n', [Runtime6]),
    puzzle_9_7(Board7),
    solve_puzzle(Board7, Runtime7),
    format(' > Puzzle 7 solver runtime: ~3d s~n', [Runtime7]),
    puzzle_9_8(Board8),
    solve_puzzle(Board8, Runtime8),
    format(' > Puzzle 8 solver runtime: ~3d s~n', [Runtime8]),
    puzzle_9_9(Board9),
    solve_puzzle(Board9, Runtime9),
    format(' > Puzzle 9 solver runtime: ~3d s~n', [Runtime9]),
    puzzle_9_10(Board10),
    solve_puzzle(Board10, Runtime10),
    format(' > Puzzle 10 solver runtime: ~3d s~n', [Runtime10]),
    AvgRuntime is (Runtime1 + Runtime2 + Runtime3 + Runtime4 + Runtime5 + Runtime6 + Runtime7 + Runtime8 + Runtime9 + Runtime10) / 10,
    format(' > Average runtime: ~3d s~n', [AvgRuntime]).

test_solver_10:-
    puzzle_10_1(Board1),
    solve_puzzle(Board1, Runtime1),
    format(' > Puzzle 1 solver runtime: ~3d s~n', [Runtime1]),
    puzzle_10_2(Board2),
    solve_puzzle(Board2, Runtime2),
    format(' > Puzzle 2 solver runtime: ~3d s~n', [Runtime2]),
    puzzle_10_3(Board3),
    solve_puzzle(Board3, Runtime3),
    format(' > Puzzle 3 solver runtime: ~3d s~n', [Runtime3]),
    puzzle_10_4(Board4),
    solve_puzzle(Board4, Runtime4),
    format(' > Puzzle 4 solver runtime: ~3d s~n', [Runtime4]),
    puzzle_10_5(Board5),
    solve_puzzle(Board5, Runtime5),
    format(' > Puzzle 5 solver runtime: ~3d s~n', [Runtime5]),
    puzzle_10_6(Board6),
    solve_puzzle(Board6, Runtime6),
    format(' > Puzzle 6 solver runtime: ~3d s~n', [Runtime6]),
    puzzle_10_7(Board7),
    solve_puzzle(Board7, Runtime7),
    format(' > Puzzle 7 solver runtime: ~3d s~n', [Runtime7]),
    puzzle_10_8(Board8),
    solve_puzzle(Board8, Runtime8),
    format(' > Puzzle 8 solver runtime: ~3d s~n', [Runtime8]),
    puzzle_10_9(Board9),
    solve_puzzle(Board9, Runtime9),
    format(' > Puzzle 9 solver runtime: ~3d s~n', [Runtime9]),
    puzzle_10_10(Board10),
    solve_puzzle(Board10, Runtime10),
    format(' > Puzzle 10 solver runtime: ~3d s~n', [Runtime10]),
    AvgRuntime is (Runtime1 + Runtime2 + Runtime3 + Runtime4 + Runtime5 + Runtime6 + Runtime7 + Runtime8 + Runtime9 + Runtime10) / 10,
    format(' > Average runtime: ~3d s~n', [AvgRuntime]).
```

Listing 18. labeling_tests.pl

```
:-consult('solver.pl').
:-consult('puzzles.pl').

% ================================================
% Testing Purposes
% See puzzles in file 'puzzles.pl'
% ================================================
% test for puzzle 1
test_puzzle_1:-
    puzzle_1(Board),
    solve_puzzle(Board),
    Solution = [
        [2,1,1,2,0,0],
        [1,1,2,0,2,0],
        [1,2,0,1,0,2],
        [2,0,1,0,2,1],
        [0,2,0,2,1,1],
        [0,0,2,1,1,2]
    ],
    Board == Solution.

% test for puzzle 2
test_puzzle_2:-
    puzzle_2(Board),
    solve_puzzle(Board),
    Solution = [
        [1,1,2,0,2,0],
        [1,2,1,0,0,2],
        [2,1,1,2,0,0],
        [0,0,2,1,1,2],
        [0,2,0,1,2,1],
        [2,0,0,2,1,1]
    ],
    Board == Solution.

% test for puzzle 3
test_puzzle_3:-
    puzzle_3(Board),
    solve_puzzle(Board),
    Solution = [
        [1,1,2,0,2,0],
        [1,1,0,2,0,2],
        [2,0,2,1,1,0],
```

```
        [0,2,1,1,0,2],
        [2,0,1,0,2,1],
        [0,2,0,2,1,1]
    ],
    Board == Solution.

% test for puzzle 4
test_puzzle_4:-
    puzzle_4(Board),
    solve_puzzle(Board),
    Solution = [
        [0,2,0,2,1,1],
        [2,0,2,0,1,1],
        [1,2,0,1,0,2],
        [1,1,2,0,2,0],
        [0,1,1,2,0,2],
        [2,0,1,1,2,0]
    ],
    Board == Solution.

% test for puzzle 5
test_puzzle_5:-
    puzzle_5(Board),
    solve_puzzle(Board),
    Solution =  [
        [1,1,0,2,0,2],
        [1,2,1,0,2,0],
        [2,1,1,2,0,0],
        [0,0,2,1,1,2],
        [0,2,0,1,2,1],
        [2,0,2,0,1,1]
    ],
    Board == Solution.

% test for puzzle 6
test_puzzle_6:-
    puzzle_6(Board),
    solve_puzzle(Board),
    Solution = [
        [0,2,0,0,1,2,1],
        [2,0,2,0,0,1,1],
        [0,2,0,2,1,1,0],
        [1,1,0,0,2,0,2],
        [0,1,2,1,0,2,0],
        [1,0,1,2,0,0,2],
        [2,0,1,1,2,0,0]
    ],
    Board == Solution.

% test for puzzle 7
test_puzzle_7:-
    puzzle_7(Board),
    solve_puzzle(Board),
    Solution = [
        [0,2,0,1,1,2,0],
        [2,1,0,1,2,0,0],
        [0,1,2,0,1,0,2],
        [2,0,0,2,0,1,1],
        [0,0,2,0,2,1,1],
        [1,2,1,0,0,2,0],
        [1,0,1,2,0,0,2]
    ],
    Board == Solution.

% test for puzzle 8
test_puzzle_8:-
    puzzle_8(Board),
    solve_puzzle(Board),
    Solution = [
        [1,0,1,2,0,0,2],
        [1,2,1,0,0,2,0],
        [0,1,2,0,1,0,2],
        [2,0,0,2,1,0,1],
        [0,0,2,0,2,1,1],
        [2,1,0,1,0,2,0],
        [0,2,0,1,2,1,0]
    ],
    Board == Solution.

% test for puzzle 9
test_puzzle_9:-
    puzzle_9(Board),
    solve_puzzle(Board),
    Solution = [
        [0,2,1,0,1,2,0],
        [1,1,2,0,2,0,0],
        [2,0,1,0,1,0,2],
        [1,1,0,2,0,2,0],
        [0,0,2,1,0,1,2],
        [2,0,0,1,2,0,1],
        [0,2,0,2,0,1,1]
    ],
    Board == Solution.

% test for puzzle 10
test_puzzle_10:-
    puzzle_10(Board),
    solve_puzzle(Board),
    Solution = [
        [1,2,0,1,0,0,2],
        [2,0,0,1,2,1,0],
        [0,0,2,0,1,2,1],
        [0,2,0,2,1,1,0],
        [1,1,2,0,2,0,0],
        [0,1,1,2,0,0,2],
        [2,0,1,0,0,2,1]
    ],
    Board == Solution.

% test for puzzle 11
test_puzzle_11:-
    puzzle_11(Board),
    solve_puzzle(Board),
    Solution = [
        [1,2,0,0,1,0,2],
        [0,0,2,0,1,2,1],
        [2,0,0,2,0,1,1],
        [1,1,2,0,2,0,0],
        [0,2,0,1,0,1,2],
        [0,1,1,2,0,2,0],
        [2,0,1,1,2,0,0]
    ],
    Board == Solution.

% test for puzzle 12
test_puzzle_12:-
    puzzle_12(Board),
    solve_puzzle(Board),
    Solution = [
        [0,0,2,1,0,1,2],
        [0,2,0,2,0,1,1],
        [2,0,0,1,2,0,1],
        [0,2,1,0,1,2,0],
        [1,0,1,2,0,0,2],
        [2,1,0,0,1,2,0],
        [1,1,2,0,2,0,0]
    ],
    Board == Solution.

% test for puzzle 13
test_puzzle_13:-
    puzzle_13(Board),
    solve_puzzle(Board),
    Solution = [
        [0,1,2,0,1,0,2],
        [2,1,0,0,1,2,0],
        [0,2,0,2,0,1,1],
        [0,0,2,0,2,1,1],
```

```
        [1,0,1,2,0,0,2],
        [2,0,1,1,2,0,0],
        [1,2,0,1,0,2,0]
    ],
    Board == Solution.
```

Listing 19. solver_tests.pl

```
% ================================================
% Puzzles available @ https://www2.stetson.edu/~efriedma/puzzle/closefar/
% 0 - EMPTY
% 1 - CLOSE
% 2 - FAR
% ================================================

% ================================================
% 6x6 puzzles
% ================================================
puzzle_1([
    [2, _, _, _, _, _],
    [_, _, _, _, 2, _],
    [_, _, _, 1, _, _],
    [_, _, _, _, _, 1],
    [_, 2, _, _, _, _],
    [_, _, 2, _, _, _]
]).

puzzle_2([
    [_, _, _, _, 2, _],
    [_, _, 1, _, _, _],
    [2, _, _, _, _, _],
    [_, _, _, 1, _, _],
    [_, 2, _, _, _, _],
    [_, _, _, _, _, 1]
]).

puzzle_3([
    [_, 1, _, _, _, _],
    [_, _, _, _, _, 2],
    [_, _, 2, _, _, _],
    [_, _, _, 1, _, _],
    [2, _, _, _, _, _],
    [_, _, _, _, 1, _]
]).

puzzle_4([
    [_, 2, _, _, _, _],
    [_, _, 2, _, _, _],
    [_, _, _, _, _, 2],
    [_, _, _, _, 2, _],
    [_, _, _, 2, _, _],
    [2, _, _, _, _, _]
]).

puzzle_5([
    [_, 1, _, _, _, _],
    [1, _, _, _, _, _],
    [_, _, _, 2, _, _],
    [_, _, _, _, _, 2],
    [_, _, _, _, 2, _],
    [_, _, 2, _, _, _]
]).

% ================================================
% 7x7 puzzles
% ================================================
puzzle_6([
    [_, _, _, _, 1, _, _],
    [_, _, _, _, _, 1, _],
    [_, _, _, 2, _, _, _],
    [_, 1, _, _, _, _, _],
    [_, _, 2, _, _, _, _],
    [_, _, _, _, _, _, 2],
    [2, _, _, _, _, _, _]
]).

puzzle_7([
    [_, _, _, _, 1, _, _],
    [_, 1, _, _, _, _, _],
    [_, _, 2, _, _, _, _],
    [2, _, _, _, _, _, _],
    [_, _, _, _, _, _, 1],
    [_, _, _, _, _, 2, _],
    [_, _, _, 2, _, _, _]
]).

puzzle_8([
    [1, _, _, _, _, _, _],
    [_, 2, _, _, _, _, _],
    [_, _, _, _, 1, _, _],
    [_, _, _, _, _, _, 1],
    [_, _, 2, _, _, _, _],
    [_, _, _, _, _, 2, _],
    [_, _, _, 1, _, _, _]
]).

puzzle_9([
    [_, _, _, _, 1, _, _],
    [_, _, 2, _, _, _, _],
    [2, _, _, _, _, _, _],
    [_, 1, _, _, _, _, _],
    [_, _, _, _, _, 1, _],
    [_, _, _, 1, _, _, _],
    [_, _, _, _, _, _, 1]
]).

puzzle_10([
    [1, _, _, _, _, _, _],
    [_, _, _, 1, _, _, _],
    [_, _, _, _, _, 2, _],
    [_, _, _, _, 1, _, _],
    [_, _, 2, _, _, _, _],
    [_, 1, _, _, _, _, _],
    [_, _, _, _, _, _, 1]
]).

puzzle_11([
    [_, 2, _, _, _, _, _],
    [_, _, _, _, _, _, 1],
    [2, _, _, _, _, _, _],
    [_, _, _, _, 2, _, _],
    [_, _, _, _, _, 1, _],
    [_, _, _, 2, _, _, _],
    [_, _, 1, _, _, _, _]
]).

puzzle_12([
    [_, _, 2, _, _, _, _],
    [_, _, _, 2, _, _, _],
    [_, _, _, _, _, _, 1],
    [_, _, _, _, _, 2, _],
    [1, _, _, _, _, _, _],
    [_, _, _, _, 1, _, _],
    [_, 1, _, _, _, _, _]
]).

puzzle_13([
    [_, _, _, _, 1, _, _],
    [2, _, _, _, _, _, _],
    [_, 2, _, _, _, _, _],
    [_, _, _, _, _, 1, _],
    [_, _, _, _, _, _, 2],
    [_, _, 1, _, _, _, _],
    [_, _, _, 1, _, _, _]
]).

% ================================================
```

```
% Randomly Generated Puzzles
% using generate_random_puzzle predicate
% ==================================================
% 6x6
puzzle_6_1([
    [1,_,_,_,_,_],
    [_,_,2,_,_,_],
    [_,_,_,_,_,2],
    [_,2,_,_,_,_],
    [_,_,_,_,1,_],
    [_,_,_,2,_,_]
]).
puzzle_6_2([
    [_,1,_,_,_,_],
    [_,_,2,_,_,_],
    [1,_,_,_,_,_],
    [_,_,_,_,2,_],
    [_,_,_,2,_,_],
    [_,_,_,_,_,1]
]).
puzzle_6_3([
    [_,_,_,_,_,2],
    [_,_,1,_,_,_],
    [_,1,_,_,_,_],
    [_,_,_,_,1,_],
    [_,_,_,1,_,_],
    [2,_,_,_,_,_]
]).
puzzle_6_4([
    [_,_,_,_,_,2],
    [_,2,_,_,_,_],
    [_,_,2,_,_,_],
    [2,_,_,_,_,_],
    [_,_,_,2,_,_],
    [_,_,_,_,1,_]
]).
puzzle_6_5([
    [_,_,1,_,_,_],
    [1,_,_,_,_,_],
    [_,_,_,2,_,_],
    [_,_,_,_,1,_],
    [_,2,_,_,_,_],
    [_,_,_,_,_,2]
]).
puzzle_6_6([
    [2,_,_,_,_,_],
    [_,2,_,_,_,_],
    [_,_,_,_,1,_],
    [_,_,_,2,_,_],
    [_,_,2,_,_,_],
    [_,_,_,_,_,2]
]).
puzzle_6_7([
    [_,1,_,_,_,_],
    [_,_,1,_,_,_],
    [2,_,_,_,_,_],
    [_,_,_,1,_,_],
    [_,_,_,_,2,_],
    [_,_,_,_,_,1]
]).
puzzle_6_8([
    [2,_,_,_,_,_],
    [_,1,_,_,_,_],
    [_,_,1,_,_,_],
    [_,_,_,_,1,_],
    [_,_,_,1,_,_],
    [_,_,_,_,_,1]
]).
puzzle_6_9([
    [_,_,_,1,_,_],
    [_,_,_,_,2,_],
    [_,_,_,_,_,2],
    [1,_,_,_,_,_],
    [_,2,_,_,_,_],
    [_,_,2,_,_,_]
]).
puzzle_6_10([
    [_,_,_,_,_,2],
    [_,_,_,1,_,_],
    [2,_,_,_,_,_],
    [_,1,_,_,_,_],
    [_,_,_,_,2,_],
    [_,_,1,_,_,_]
]).

% 7x7
puzzle_7_1([
    [_,_,_,2,_,_,_],
    [_,_,_,_,_,2,_],
    [2,_,_,_,_,_,_],
    [_,2,_,_,_,_,_],
    [_,_,1,_,_,_,_],
    [_,_,_,_,_,_,2],
    [_,_,_,_,2,_,_]
]).
puzzle_7_2([
    [_,_,_,_,_,2,_],
    [_,_,_,2,_,_,_],
    [_,2,_,_,_,_,_],
    [2,_,_,_,_,_,_],
    [_,_,2,_,_,_,_],
    [_,_,_,_,_,_,1],
    [_,_,_,_,1,_,_]
]).
puzzle_7_3([
    [_,_,_,_,1,_,_],
    [_,_,_,_,_,_,1],
    [2,_,_,_,_,_,_],
    [_,_,2,_,_,_,_],
    [_,_,_,_,_,2,_],
    [_,1,_,_,_,_,_],
    [_,_,_,2,_,_,_]
]).
puzzle_7_4([
    [_,_,_,_,1,_,_],
    [_,_,1,_,_,_,_],
    [_,_,_,_,_,2,_],
    [_,_,_,2,_,_,_],
    [2,_,_,_,_,_,_],
    [_,_,_,_,_,_,1],
    [_,2,_,_,_,_,_]
]).
puzzle_7_5([
    [_,1,_,_,_,_,_],
    [_,_,2,_,_,_,_],
    [_,_,_,_,_,2,_],
    [1,_,_,_,_,_,_],
    [_,_,_,_,1,_,_],
    [_,_,_,_,_,_,1],
    [_,_,_,2,_,_,_]
]).
puzzle_7_6([
    [_,_,_,1,_,_,_],
    [_,_,_,_,_,2,_],
    [_,_,1,_,_,_,_],
    [2,_,_,_,_,_,_],
    [_,_,_,_,1,_,_],
    [_,2,_,_,_,_,_],
    [_,_,_,_,_,_,1]
]).
puzzle_7_7([
    [2,_,_,_,_,_,_],
    [_,_,_,_,_,_,2],
    [_,2,_,_,_,_,_],
    [_,_,1,_,_,_,_],
    [_,_,_,_,2,_,_],
    [_,_,_,2,_,_,_],
    [_,_,_,_,_,2,_]
```

```
]).
puzzle_7_8([
    [_,_,_,2,_,_,_],
    [_,_,_,_,_,_,2],
    [_,_,_,_,_,2,_],
    [_,_,_,_,1,_,_],
    [2,_,_,_,_,_,_],
    [_,2,_,_,_,_,_],
    [_,_,2,_,_,_,_]
]).
puzzle_7_9([
    [2,_,_,_,_,_,_],
    [_,2,_,_,_,_,_],
    [_,_,_,_,_,2,_],
    [_,_,_,_,_,_,2],
    [_,_,1,_,_,_,_],
    [_,_,_,_,2,_,_],
    [_,_,_,2,_,_,_]
]).
puzzle_7_10([
    [1,_,_,_,_,_,_],
    [_,1,_,_,_,_,_],
    [_,_,_,_,2,_,_],
    [_,_,2,_,_,_,_],
    [_,_,_,_,_,_,1],
    [_,_,_,2,_,_,_],
    [_,_,_,_,_,2,_]
]).

% 8x8
puzzle_8_1([
    [_,_,1,_,_,_,_,_],
    [_,_,_,_,_,_,_,1],
    [_,_,_,_,1,_,_,_],
    [2,_,_,_,_,_,_,_],
    [_,_,_,2,_,_,_,_],
    [_,_,_,_,_,_,2,_],
    [_,_,_,_,_,_,1,_],
    [_,1,_,_,_,_,_,_]
]).
puzzle_8_2([
    [_,1,_,_,_,_,_,_],
    [1,_,_,_,_,_,_,_],
    [_,_,_,1,_,_,_,_],
    [_,_,1,_,_,_,_,_],
    [_,_,_,_,_,2,_,_],
    [_,_,_,_,1,_,_,_],
    [_,_,_,_,_,_,_,1],
    [_,_,_,_,_,_,1,_]
]).
puzzle_8_3([
    [2,_,_,_,_,_,_,_],
    [_,2,_,_,_,_,_,_],
    [_,_,_,_,_,_,2,_],
    [_,_,_,_,_,_,_,2],
    [_,_,_,2,_,_,_,_],
    [_,_,_,_,2,_,_,_],
    [_,_,_,_,_,2,_,_],
    [_,_,2,_,_,_,_,_]
]).
puzzle_8_4([
    [_,_,_,_,1,_,_,_],
    [1,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,2,_],
    [_,1,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,2],
    [_,_,_,2,_,_,_,_],
    [_,_,2,_,_,_,_,_],
    [_,_,_,_,_,1,_,_]
]).
puzzle_8_5([
    [_,_,1,_,_,_,_,_],
    [_,_,_,1,_,_,_,_],
    [_,_,_,_,1,_,_,_],
    [_,_,_,_,_,2,_,_],
    [1,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,1,_],
    [_,_,_,_,_,_,_,2],
    [_,2,_,_,_,_,_,_]
]).
puzzle_8_6([
    [_,2,_,_,_,_,_,_],
    [1,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,2,_],
    [_,_,_,_,_,2,_,_],
    [_,_,_,2,_,_,_,_],
    [_,_,_,_,_,_,_,1],
    [_,_,2,_,_,_,_,_],
    [_,_,_,_,1,_,_,_]
]).
puzzle_8_7([
    [2,_,_,_,_,_,_,_],
    [_,_,_,_,_,2,_,_],
    [_,_,_,_,_,_,2,_],
    [_,2,_,_,_,_,_,_],
    [_,_,2,_,_,_,_,_],
    [_,_,_,_,_,_,_,1],
    [_,_,_,2,_,_,_,_],
    [_,_,_,_,2,_,_,_]
]).
puzzle_8_8([
    [2,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,2,_],
    [_,_,_,_,_,2,_,_],
    [_,_,_,_,_,_,_,1],
    [_,_,_,1,_,_,_,_],
    [_,_,_,_,2,_,_,_],
    [_,_,2,_,_,_,_,_],
    [_,2,_,_,_,_,_,_]
]).
puzzle_8_9([
    [_,_,_,2,_,_,_,_],
    [_,_,2,_,_,_,_,_],
    [_,_,_,_,_,_,_,2],
    [_,2,_,_,_,_,_,_],
    [_,_,_,_,1,_,_,_],
    [_,_,_,_,_,1,_,_],
    [_,_,_,_,_,_,1,_],
    [2,_,_,_,_,_,_,_]
]).
puzzle_8_10([
    [1,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,2,_],
    [_,_,_,_,_,1,_,_],
    [_,_,_,1,_,_,_,_],
    [_,1,_,_,_,_,_,_],
    [_,_,_,_,2,_,_,_],
    [_,_,_,_,_,_,_,1],
    [_,_,2,_,_,_,_,_]
]).

% 9x9
puzzle_9_1([
    [2,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,2],
    [_,_,_,1,_,_,_,_,_],
    [_,1,_,_,_,_,_,_,_],
    [_,_,_,_,_,1,_,_,_],
    [_,_,2,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,_,2,_,_],
    [_,_,_,_,2,_,_,_,_]
]).
puzzle_9_2([
    [2,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,2,_,_,_],
    [_,_,_,1,_,_,_,_,_],
    [_,_,1,_,_,_,_,_,_],
```

```
    [_,_,_,_,_,_,_,2,_],
    [_,_,_,_,1,_,_,_,_],
    [_,2,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,1],
    [_,_,_,_,_,_,2,_,_]
]).
puzzle_9_3([
    [_,_,_,1,_,_,_,_,_],
    [_,_,1,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,2],
    [_,_,_,_,1,_,_,_,_],
    [_,_,_,_,_,1,_,_,_],
    [_,2,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,_,_,_,2],
    [2,_,_,_,_,_,_,_,_]
]).
puzzle_9_4([
    [_,_,_,_,1,_,_,_,_],
    [_,_,_,1,_,_,_,_,_],
    [_,1,_,_,_,_,_,_,_],
    [2,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,2,_,_],
    [_,_,1,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,1,_,_,_],
    [_,_,_,_,_,_,_,_,1]
]).
puzzle_9_5([
    [_,_,_,_,1,_,_,_,_],
    [_,2,_,_,_,_,_,_,_],
    [1,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,1,_,_,_],
    [_,_,_,_,_,_,_,1,_],
    [_,_,_,2,_,_,_,_,_],
    [_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,_,_,_,1],
    [_,_,2,_,_,_,_,_,_]
]).
puzzle_9_6([
    [_,1,_,_,_,_,_,_,_],
    [1,_,_,_,_,_,_,_,_],
    [_,_,_,1,_,_,_,_,_],
    [_,_,_,_,_,_,_,2,_],
    [_,_,_,_,1,_,_,_,_],
    [_,_,2,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,1,_,_,_],
    [_,_,_,_,_,_,_,_,1]
]).
puzzle_9_7([
    [1,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,1,_,_,_],
    [_,_,_,_,_,_,_,2,_],
    [_,_,1,_,_,_,_,_,_],
    [_,_,_,_,1,_,_,_,_],
    [_,2,_,_,_,_,_,_,_],
    [_,_,_,1,_,_,_,_,_],
    [_,_,_,_,_,_,_,1,_],
    [_,_,_,_,_,_,_,_,1]
]).
puzzle_9_8([
    [_,_,2,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,2],
    [_,2,_,_,_,_,_,_,_],
    [_,_,_,_,2,_,_,_,_],
    [2,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,2,_,_],
    [_,_,_,_,_,2,_,_,_],
    [_,_,_,2,_,_,_,_,_],
    [_,_,_,_,_,_,_,1,_]
]).
puzzle_9_9([
    [1,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,1,_],
    [_,_,_,_,_,_,_,_,1],
    [_,1,_,_,_,_,_,_,_],
    [_,_,2,_,_,_,_,_,_],
    [_,_,_,_,_,_,1,_,_],
    [_,_,_,_,_,1,_,_,_],
    [_,_,_,_,1,_,_,_,_],
    [_,_,_,2,_,_,_,_,_]
]).
puzzle_9_10([
    [_,_,_,_,2,_,_,_,_],
    [_,1,_,_,_,_,_,_,_],
    [_,_,_,_,_,1,_,_,_],
    [_,_,_,_,_,_,1,_,_],
    [_,_,2,_,_,_,_,_,_],
    [1,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,1],
    [_,_,_,_,_,_,_,1,_],
    [_,_,_,2,_,_,_,_,_]
]).

% 10x10
puzzle_10_1([
    [_,_,_,_,_,_,2,_,_,_],
    [_,_,_,_,_,_,_,2,_,_],
    [_,_,1,_,_,_,_,_,_,_],
    [_,_,_,1,_,_,_,_,_,_],
    [2,_,_,_,_,_,_,_,_,_],
    [_,1,_,_,_,_,_,_,_,_],
    [_,_,_,_,2,_,_,_,_,_],
    [_,_,_,_,_,2,_,_,_,_],
    [_,_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,_,_,_,_,1]
]).
puzzle_10_2([
    [_,_,_,_,_,_,_,2,_,_],
    [_,_,_,_,_,_,2,_,_,_],
    [_,_,_,_,_,_,_,1,_,_],
    [_,_,_,_,2,_,_,_,_,_],
    [_,2,_,_,_,_,_,_,_,_],
    [_,_,2,_,_,_,_,_,_,_],
    [_,_,_,2,_,_,_,_,_,_],
    [2,_,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,_,1],
    [_,_,_,_,_,_,_,_,1,_]
]).
puzzle_10_3([
    [_,_,_,_,_,_,_,1,_,_],
    [_,1,_,_,_,_,_,_,_,_],
    [_,_,2,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,_,_,_,_,1],
    [_,_,_,_,_,_,_,_,_,1],
    [_,_,_,2,_,_,_,_,_,_],
    [_,_,_,_,2,_,_,_,_,_],
    [1,_,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,2,_,_,_,_]
]).
puzzle_10_4([
    [_,1,_,_,_,_,_,_,_,_],
    [_,_,_,2,_,_,_,_,_,_],
    [1,_,_,_,_,_,_,_,_,_],
    [_,_,1,_,_,_,_,_,_,_],
    [_,_,_,_,2,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,_,_,1,_,_],
    [_,_,_,_,_,_,2,_,_,_],
    [_,_,_,_,_,_,_,_,_,2],
    [_,_,_,_,_,_,_,_,1,_]
]).
puzzle_10_5([
    [1,_,_,_,_,_,_,_,_,_],
    [_,_,_,_,1,_,_,_,_,_],
    [_,1,_,_,_,_,_,_,_,_],
```

```
    [_,_,_,_,_,_,_,_,_,2],
    [_,_,_,_,_,2,_,_,_,_],
    [_,_,_,_,_,_,_,2,_,_],
    [_,_,_,_,_,_,2,_,_,_],
    [_,_,_,_,_,_,_,_,2,_],
    [_,_,_,2,_,_,_,_,_,_],
    [_,_,2,_,_,_,_,_,_,_]
]).
puzzle_10_6([
    [_,_,_,1,_,_,_,_,_,_],
    [_,2,_,_,_,_,_,_,_,_],
    [2,_,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,2,_,_],
    [_,_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,1,_,_,_,_],
    [_,_,_,_,1,_,_,_,_,_],
    [_,_,_,_,_,_,1,_,_,_],
    [_,_,_,_,_,_,_,_,_,1],
    [_,_,2,_,_,_,_,_,_,_]
]).
puzzle_10_7([
    [1,_,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,1,_,_],
    [_,_,_,_,_,1,_,_,_,_],
    [_,_,_,_,_,_,_,_,_,2],
    [_,2,_,_,_,_,_,_,_,_],
    [_,_,_,2,_,_,_,_,_,_],
    [_,_,2,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,1,_],
    [_,_,_,_,1,_,_,_,_,_],
    [_,_,_,_,_,_,2,_,_,_]
]).
puzzle_10_8([
    [1,_,_,_,_,_,_,_,_,_],
    [_,_,_,_,1,_,_,_,_,_],
    [_,1,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,_,2],
    [_,_,_,_,_,2,_,_,_,_],
    [_,_,_,_,_,_,_,2,_,_],
    [_,_,_,_,_,_,2,_,_,_],
    [_,_,_,_,_,_,_,_,2,_],
    [_,_,_,2,_,_,_,_,_,_],
    [_,_,2,_,_,_,_,_,_,_]
]).
puzzle_10_9([
    [_,_,_,1,_,_,_,_,_,_],
    [_,2,_,_,_,_,_,_,_,_],
    [2,_,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,_,_,_,2,_],
    [_,_,_,_,_,1,_,_,_,_],
    [_,_,_,_,1,_,_,_,_,_],
    [_,_,_,_,_,_,1,_,_,_],
    [_,_,_,_,_,_,_,_,_,1],
    [_,_,2,_,_,_,_,_,_,_]
]).
puzzle_10_10([
    [1,_,_,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,1,_,_],
    [_,_,_,_,_,1,_,_,_,_],
    [_,_,_,_,_,_,_,_,_,2],
    [_,2,_,_,_,_,_,_,_,_],
    [_,_,_,2,_,_,_,_,_,_],
    [_,_,2,_,_,_,_,_,_,_],
    [_,_,_,_,_,_,_,_,1,_],
    [_,_,_,_,1,_,_,_,_,_],
    [_,_,_,_,_,_,2,_,_,_]
]).
```

Listing 20. puzzles.pl