# MiniZinc Tutorial

## CS 275: Constraint Networks

Annie Raichev Winter 2026

# Recap
## Constraint Network

- A constraint Network, R= (X, D, C)
  - Where X is the variables
  - D is the domain
  - C is the constraints
- Solution: an assignment to all variables that satisfies all constraints
- MiniZinc is a programming language that  allows us to define all parts of a constraint network and solve it

# Example 1:

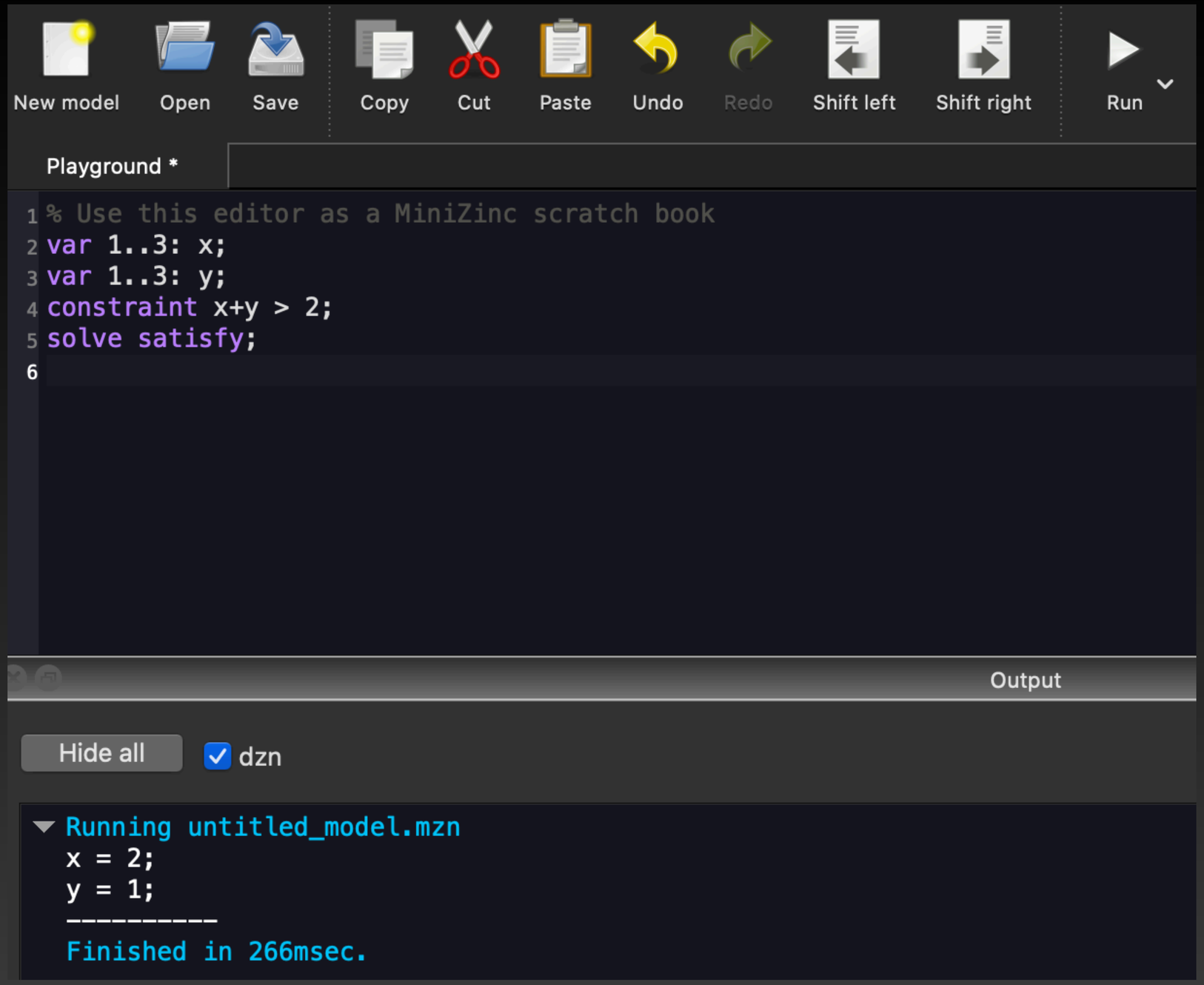
## Variable types

- **Var**: declare decision variables
  - *Var* domain: variable name;
- **Constraint:** declare any boolean expression as a constraint
- **Solve**: solves the problem declared
  - *Solve satisfy;*
- **Output**: a satisfying assignment

## Simple Model

- **Problem:** Consider two integers, x and y, that can take on values 1, 2, or 3. We want to find values such that x +y > 2
- **Variables**: x, y
- **Domains:** $D_x=D_y=\{1,2,3\}$
- **Constraint:** x + y > 2

# Example 1: Code



# Example 2:

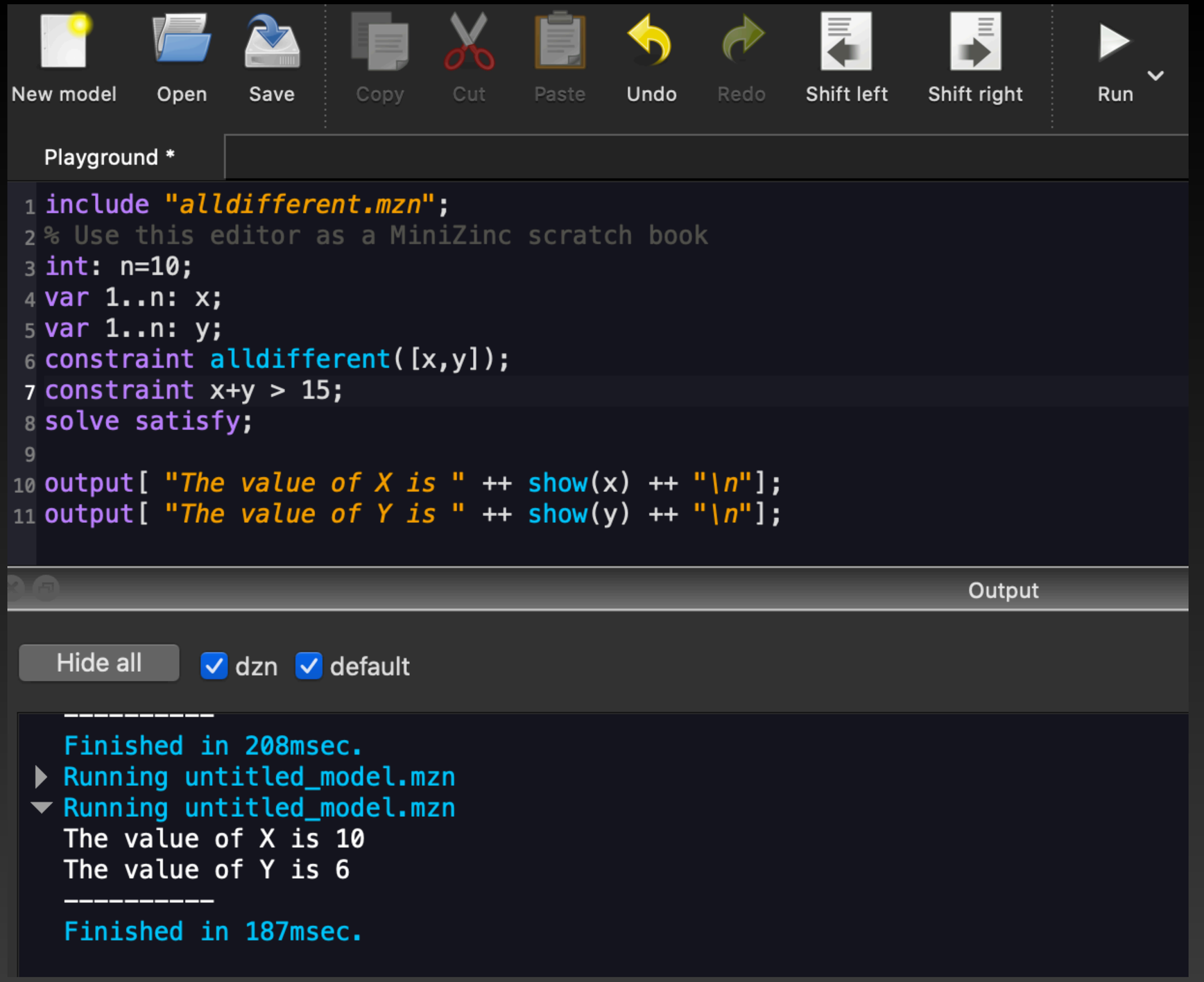
## More variables & functions

- Can also declare variables like integers, floats, strings, etc
  - Need colon after variable type
- Data structures: arrays[] and sets{}
- Output function: can concatenate strings with ++ sign
- Can declare global constraints: AllDifferent makes all variable values unique
  - Need to include the appropriate file
- Every line needs a semicolon ending

## Simple Model pt 2

- **Problem:** Consider two integers, x and y, that can take on values 1,...,10. We want to find values such that x +y > 15 and x and y are different numbers.
- **Variables**: x, y
- **Domains:** $D_x=D_y=\{1,2,...,10\}$
- **Constraint:** $x + y > 15$ and $x \neq y$

# Example 2: Code



# What if the problem is unsatisfiable?

## Simple Model pt 3

- **Problem:** Consider two integers, x and y, that can take on values 1,...,10. We want to find values such that x +y > 21 and x and y are different numbers.
- **Variables**: x, y
- **Domains:** Dx=Dy={1,2,...,10}
- **Constraint:** x + y > 21 and x ≠y

```
1 include "alldifferent.mzn";
2 % Use this editor as a MiniZinc scratch book
3 int: n=10;
4 var 1..n: x;
5 var 1..n: y;
6 constraint alldifferent([x,y]);
7 constraint x+y > 21;
8 solve satisfy;
9
10 output[ "The value of X is " ++ show(x) ++ "\n"];
11 output[ "The value of Y is " ++ show(y) ++ "\n"];
```

Hide all ☑ dzn ☑ default

```
Running untitled_model.mzn
Running untitled_model.mzn
Running untitled_model.mzn
Running untitled_model.mzn
Running untitled_model.mzn
Running untitled_model.mzn
=====UNSATISFIABLE=====
Finished in 203msec.
```

# Example 4:

## Data files & Optimization

- Can include data files as part of the input
- Data files have form *data.dzn*
- Allows us to assign values of parameters in the model
- Can chose to minimize or maximize an objective function in the *solve* command

## Model

- **Problem:** We are raising money for a school bake sale. We have ingredients for a banana cake and choclate cake. They sell for $4.00 and $4.50 respectively. Given a certain amount of ingredients we want to maximize profit.
- **Variables:** b: # of banana cakes c: # of choc. Cakes
- **Banana Cake:** 250g flour, 2 bananas, 75g sugar, 100g butter
- **Chocolate Cake:** 200g flour, 75g cocoa, 150g sugar, 150g butter
- **Maximize:** 400b + 450c

# Example 4:

## Code

- **Input**: 4000g of flour available, 6 bananas, 2000g of sugar, 500g of butter, 500g of cocoa
- **Data file:** pantry.dzn

cakes.mzn * | pantry.dzn

```
%ingredients available
flour = 4000;
banana = 6;
sugar = 2000;
butter = 500;
cocoa = 500;
```

- **To run with data file**: go to Run > and chose the data file

cakes.mzn | pantry.dzn | example.mzn

```
int: flour;  int: banana; int: sugar;  int: butter;
int: cocoa;

constraint assert(flour >= 0,"Invalid datafile: " ++
                  "Amount of flour should be non-negative");
constraint assert(banana >= 0,"Invalid datafile: " ++
                  "Amount of banana should be non-negative");
constraint assert(sugar >= 0,"Invalid datafile: " ++
                  "Amount of sugar should be non-negative");
constraint assert(butter >= 0,"Invalid datafile: " ++
                  "Amount of butter should be non-negative");
constraint assert(cocoa >= 0,"Invalid datafile: " ++
                  "Amount of cocoa should be non-negative");
var 0..100: b; % no. of banana cakes
var 0..100: c; % no. of chocolate cakes
%max amount of ingredients we can use
constraint 250*b + 200*c <= flour;
constraint 2*b  <= banana;
constraint 75*b + 150*c <= sugar;
constraint 100*b + 150*c <= butter;
constraint 75*c <= cocoa;
% maximize our profit
solve maximize 400*b + 450*c;

output ["no. of banana cakes = \(b)\n",
        "no. of chocolate cakes = \(c)\n"];
```

Output

Hide all | dzn | default

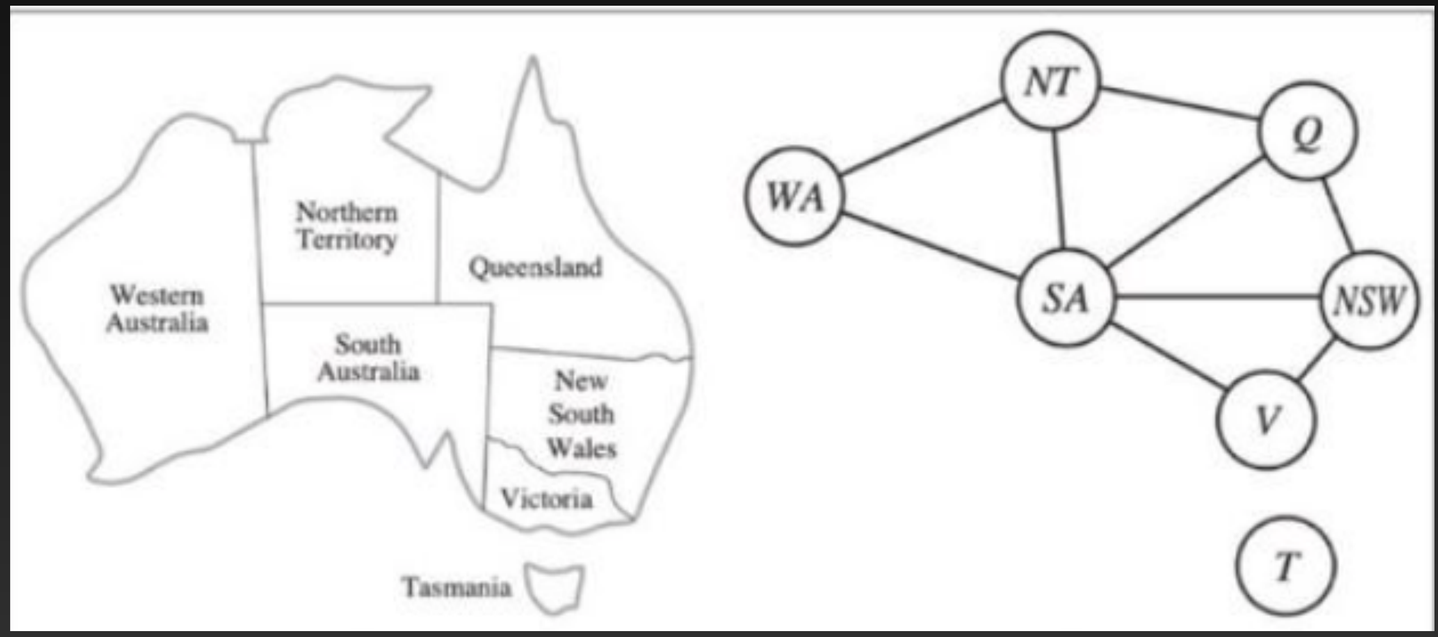
```
no. of banana cakes = 2
no. of chocolate cakes = 2
```

# Example 5:

## Map coloring

- **Problem:** Color the map of Australia such that neighboring territories have different colors
- **Variables**: the different territories of Australia: Western Aus.(WA), Northern Territory(NT), Queensland(Q), South Aus.(SA), New South Wales(NSW), Victoria(V), Tasmania(T)
- **Domains:** number of colors
- **Constraints:** neighboring territories can not have the same color assigned

## Map & Constraint graph



# Example 5: Code

example.mzn | mapColoring.mzn

```
1  int: nColors = 4;
2
3  var 1..nColors: wa;
4  var 1..nColors: nt;
5  var 1..nColors: sa;
6  var 1..nColors: q;
7  var 1..nColors: nsw;
8  var 1..nColors: v;
9  var 1..nColors: t;
10
11 constraint wa != nt;
12 constraint wa != sa;
13 constraint nt != sa;
14 constraint nt != q;
15 constraint sa != q;
16 constraint sa != nsw;
17 constraint sa != v;
18 constraint q != nsw;
19 constraint nsw != v;
20 solve satisfy;
21
22 output ["wa=\(wa)\t nt=\(nt)\t sa=\(sa)\n",
23         "q=\(q)\t nsw=\(nsw)\t v=\(v)\n",
24          "t=", show(t),  "\n"];
25
```

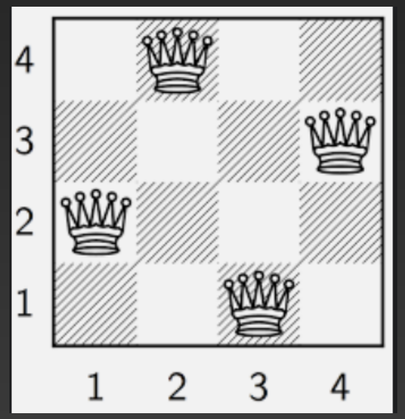
Hide all | default

```
Running mapColoring.mzn
wa=3        nt=2        sa=1
q=3         nsw=2       v=3
t=1
----------
```

# Example 6:

## Search

- Searching for solutions is left to the underlying solver
- We can specify the search method
- Finite domain search propagates constraints to remaining variables




## Model

- **Problem:** 4-Queens: on a 4X4 chessboard we want to place 4 queens such that no pair are attacking each other (queens can attack any other queen in the same row, column, or diagonal)
- **Variables:** 4 queens one in each column
- **Domain:** {1,..,4} indicating which row a queen is placed in
- **Constraints:** all rows must be different, and no two queens placed in same diagonal

# Example 6: Code

- int_search(variable, variable_choice, constraint_choice)
  - first_fail = we chose the variables with smallest domain size
  - indomain_min = assign the variable its smallest domain value
  - Other options available

example.mzn | mapColoring.mzn | 4Queens.mzn

```
1  array [1..4] of var 1..4: q; % queen in column i is in row q[i]
2
3  include "alldifferent.mzn";
4
5  constraint alldifferent(q);                          % distinct rows
6  constraint alldifferent([ q[i] + i | i in 1..4]); % distinct diagonals
7  constraint alldifferent([ q[i] - i | i in 1..4]); % upwards+downwards
8
9  % search
10 solve :: int_search(q, first_fail, indomain_min)
11       satisfy;
12 output [ if fix(q[j]) == i then "Q" else "." endif ++
13          if j == 4 then "\n" else "" endif | i,j in 1..4]
```

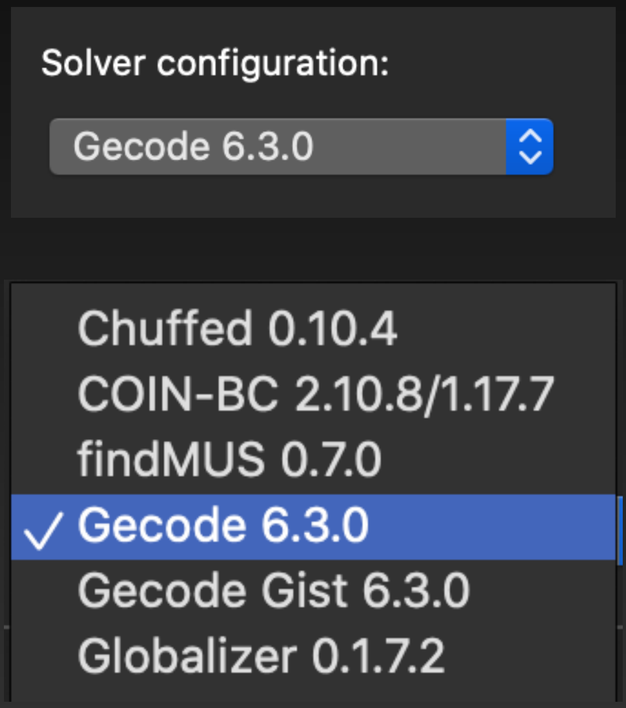
Output

Hide all | default

```
Running 4Queens.mzn
Running 4Queens.mzn
..Q.
Q...
...Q
.Q..
----------
Finished in 243msec.
```

# Solvers

- Mini Zinc is a modeling language
- Can pick different solvers



# References & More Examples

- The minizinc handbook: https://www.minizinc.org/doc-2.5.5/en/index.html
- Lecture 1: https://www.ics.uci.edu/~dechter/courses/ics-275/fall-2022/