AKARI

AI4PUZZLES

PROJ 201 / Spring 2020

Berna Yıldıran - Mehmet Fatih Baş

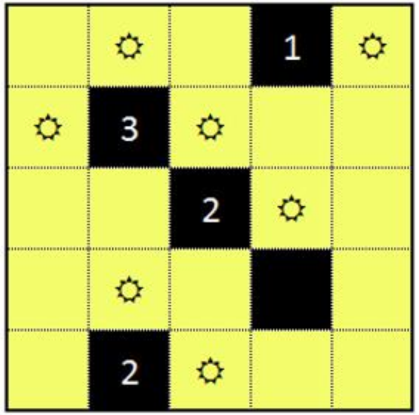
Idil Kapıkıran - Nazlı Gülşah Önen

Supervised by Esra Erdem

Co-Supervisors : Aysu Boğatarkan, Müge Fidan

PURPOSE OF THE PROJECT

To translate the rules and constraints of the Akari puzzle to statements in Answer Set Programming (ASP) and find a solution using answer set solver, Clingo.



INTRODUCTION TO AI & ASP Programming

Artificial intelligence(AI) refers to the intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and animals.

The ASP is Declarative Programming Method that used for solving problems for AI systems.

-Probabilistic Reasoning -Integrating Data -Processing Natural Language -Combinatorial Search Problems

Applications of ASP:

{ Bio-informatics, Software Engineering, Robotics, Automatic Music Composition}

SAMPLE & METHODS :

POTASSCO - RUNNING CLINGO

Clingo is a tool of ASP that based on generating possible answer sets and testing according to constraints.

ASP is used for structuring a problem as a "program" whose answer sets are equivalent to the solutions of the problem.

AI4PUZZLES EXAMPLES

Graph Coloring

N-Queens Puzzle

& our main puzzle AKARI

Steps in Our Project

1-Learning Clingo by solving Graph Coloring and N-Queens puzzles.

2-Study the Constraints and rules of our main puzzle, AKARI.

3-Divide into two groups and develop two different solutions.

4-Test our solutions with different puzzles which has diverse sizes.

AKARI PUZZLE - RULES & CONSTRAINTS

1. Light bulbs should be distributed around black squares according to the number written inside that square.

2. When a light bulb is inserted to the puzzle, that light bulb illuminates the row and column it is placed until light hits a puzzle corner or black cell.

3. Every white square should be illuminated .

4 . Two bulbs cannot illuminate each other.

GENERAL STRATEGY

UNDERSTAND & ANALYZE the problem

DECOMPOSITION to parts

ELIMINATION according to constraints (Clingos progress structure)

FORM a general solution

OUTLINE

1 - Solution#1 - By Berna & Fatih

2 - Solution#2 - By Idil & Gülşah

3 - conclusion

Akari Solution #1

Berna Yıldıran && Mehmet Fatih Baş

STEP 1

INPUTTING DEFINITION OF THE SAMPLE AKARI PUZZLE.

-Determine the size of the puzzle by specifying the range of rows and columns.

-Determine the locations of NUMBERED BLACK (numBlack) squares and ordinary BLACK (black) squares.

-Determine a range for the number inside the numBlack squares (index).

```
#const n = 6.

row(1..n).

column(1..n).

index(0..4).

numBlack(1,2,1). numBlack(3,4,2).
numBlack(4,3,4). numBlack(5,5,0).

black(1,5). black(2,2). black(3,6).
black(4,1). black(6,2). black(6,5).
```

Sample Akari Puzzle

(1,1)	(1,2,1)	(1,3)	(1,4)	(1,5)	(1,6)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,1)	(3,2)	(3,3)	(3,4,2)	(3,5)	(3,6)
(4,1)	(4,2)	(4,3,4)	(4,4)	(4,5)	(4,6)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5,0)	(5,6)
(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)

Step 2

GROUPING AND CREATING ATOMS OF THE PROGRAM

-Group all black type of black squares under black.

-Create another element named white for squares which are not black

```
black(I,J) :- numBlack(I,J,Z).

white(I,J) :- not black(I,J), row(I), column(J).
```

STEP 3

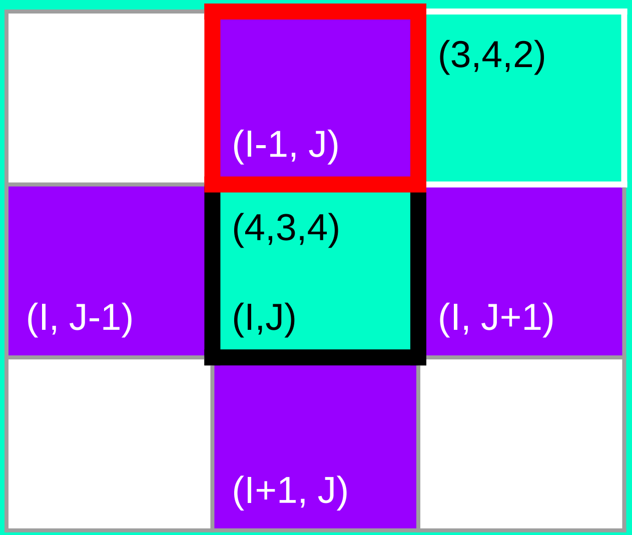
CREATING AN ATOMS FOR CHECKING THE SURROUNDINGS OF THE CELL IN 4 MAIN DIRECTION

-Total of the absolute value of the difference between x and y coordinates among center and the surrounding squares is equal to 1.

For example,

Let (I1, J1) == (I-1, J)
|I-I1|+|J-J1|=|I-(I-1)|+|J-J|=1

```
neighbor(I, J, I1, J1) :- |I-I1| + |J-J1| == 1,
row(I), row(I1), column(J), column(J1).
```

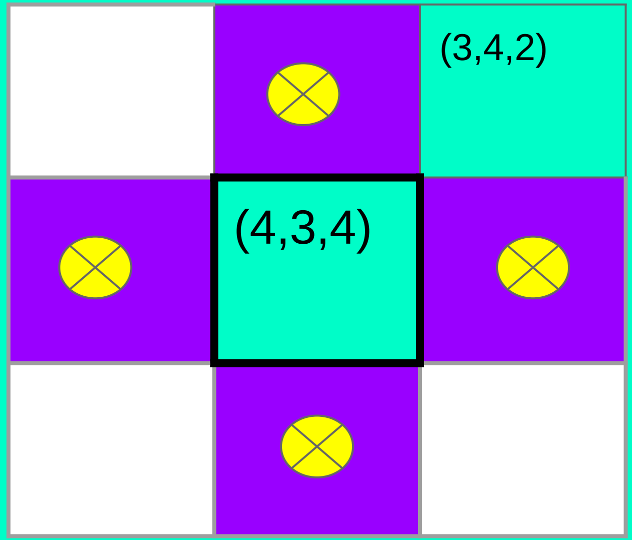


STEP 4

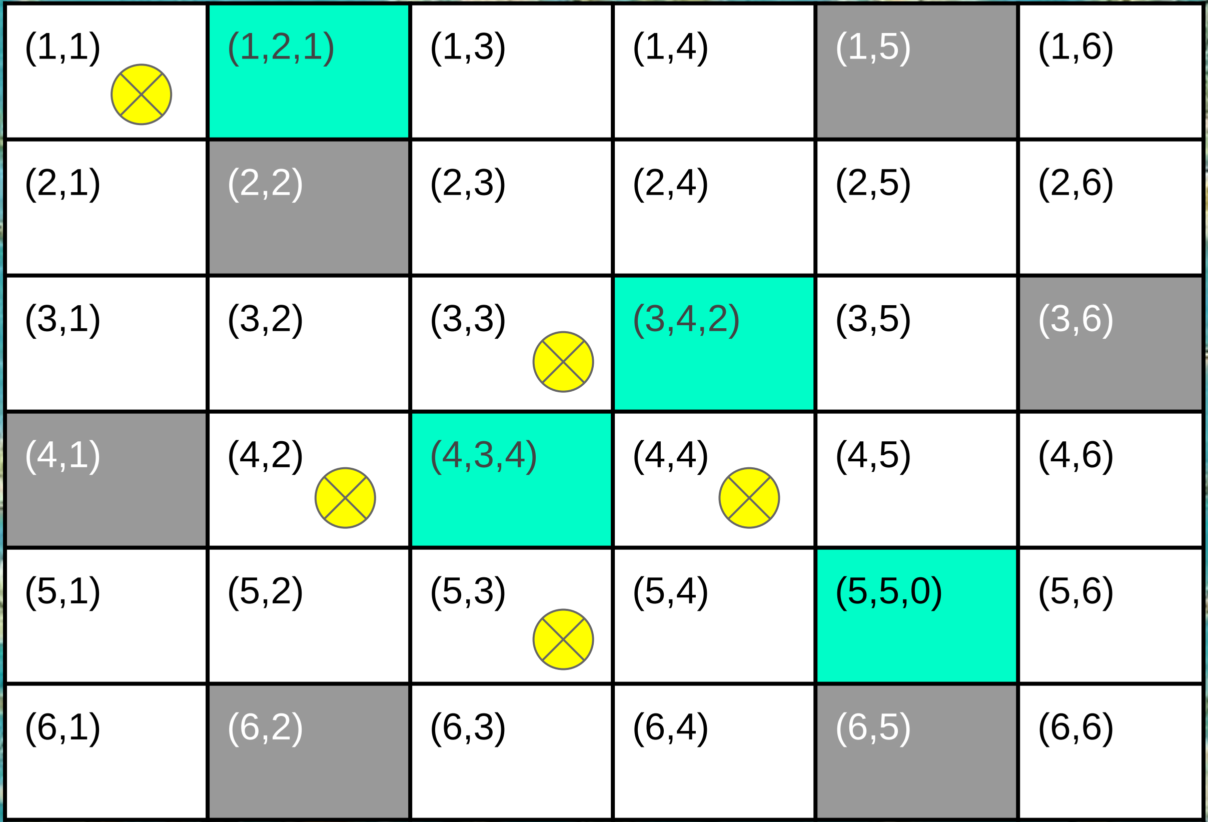
CREATING bulbs AROUND numBlack SQUARES ACCORDING TO THE NUMBER WRITTEN INSIDE THEM.

-For each numBlack square which has index value Z, create Z amount of bulbs from white square such that bulbs will be the neighbor of that numBlack square.

```
Z{bulb(I1,J1) : neighbor(I, J, I1, J1),
white(I1, J1)}Z :- numBlack(I,J,Z),
index(Z).
```



Sample Akari Puzzle

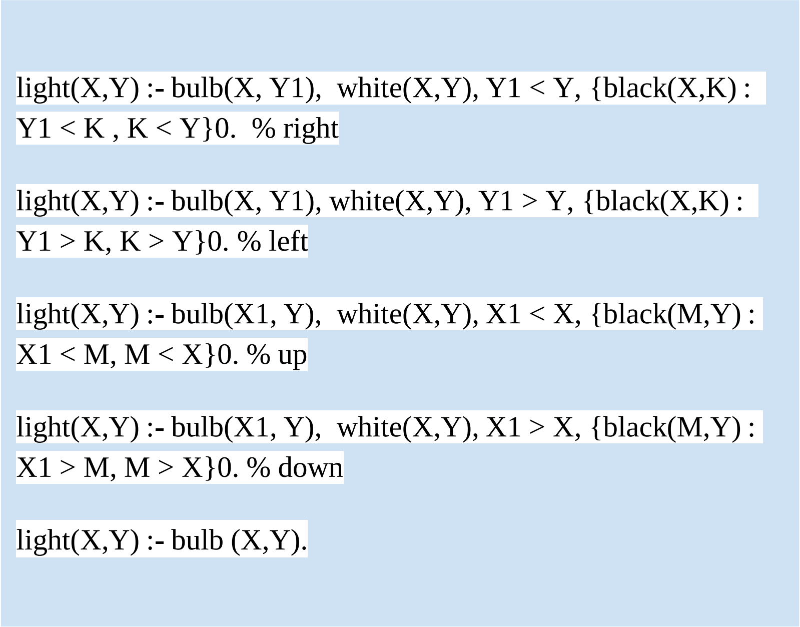


(1,1) ⊗	(1,2,1)	(1,3)	(1,4)	(1,5)	(1,6)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,1)	(3,2)	(3,3) ⊗	(3,4,2)	(3,5)	(3,6)
(4,1)	(4,2) ⊗	(4,3,4)	(4,4) ⊗	(4,5)	(4,6)
(5,1)	(5,2)	(5,3) ⊗	(5,4)	(5,5,0)	(5,6)
(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)

Step 5

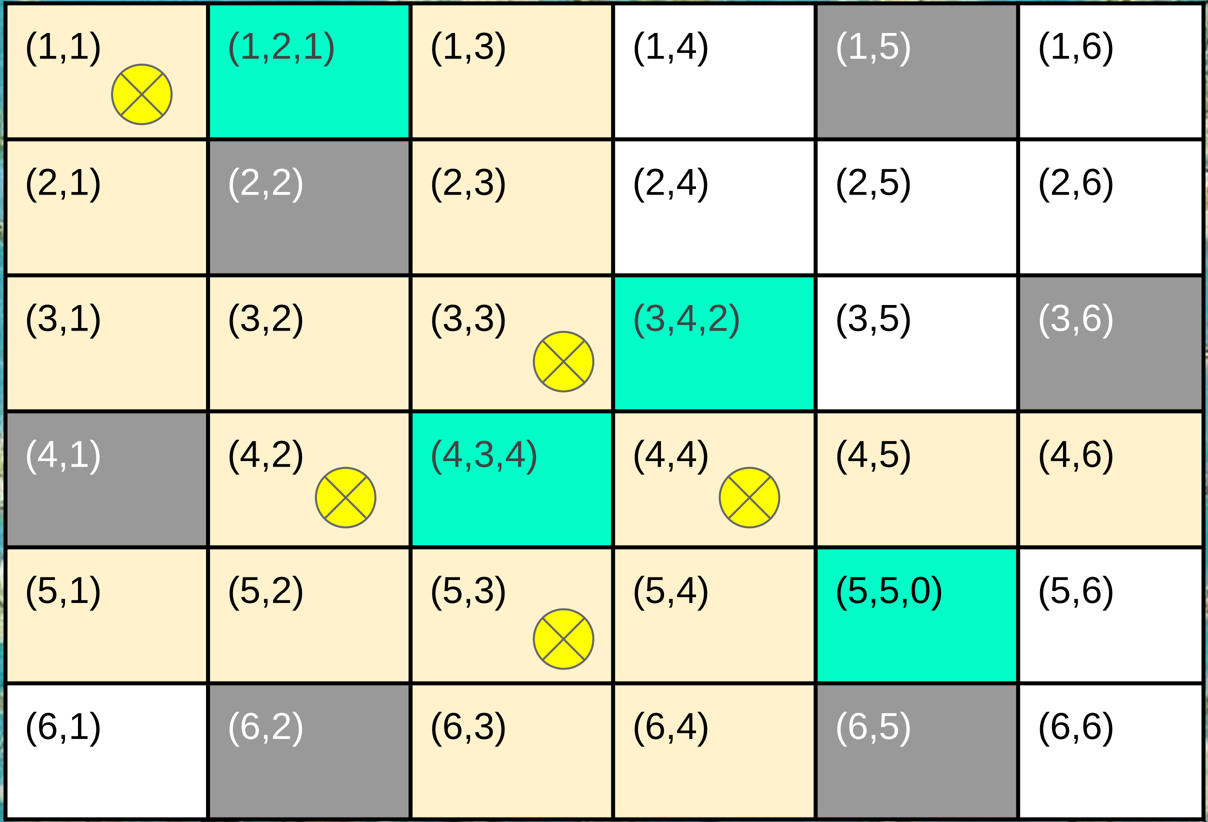
Create an atom, light, for representing the rows/columns that was illuminated in 4 main direction due to the bulbs that generated in previous step.

white squares which share the same row or column with the bulb will illuminated until they face with a black square or the corner of the puzzle.



```
light(X,Y) :- bulb(X, Y1),  white(X,Y), Y1 < Y, {black(X,K) :
Y1 < K , K < Y}0.  % right

light(X,Y) :- bulb(X, Y1), white(X,Y), Y1 > Y, {black(X,K) :
Y1 > K, K > Y}0. % left

light(X,Y) :- bulb(X1, Y),  white(X,Y), X1 < X, {black(M,Y) :
X1 < M, M < X}0. % up

light(X,Y) :- bulb(X1, Y),  white(X,Y), X1 > X, {black(M,Y) :
X1 > M, M > X}0. % down

light(X,Y) :- bulb (X,Y).
```

Sample Akari Puzzle



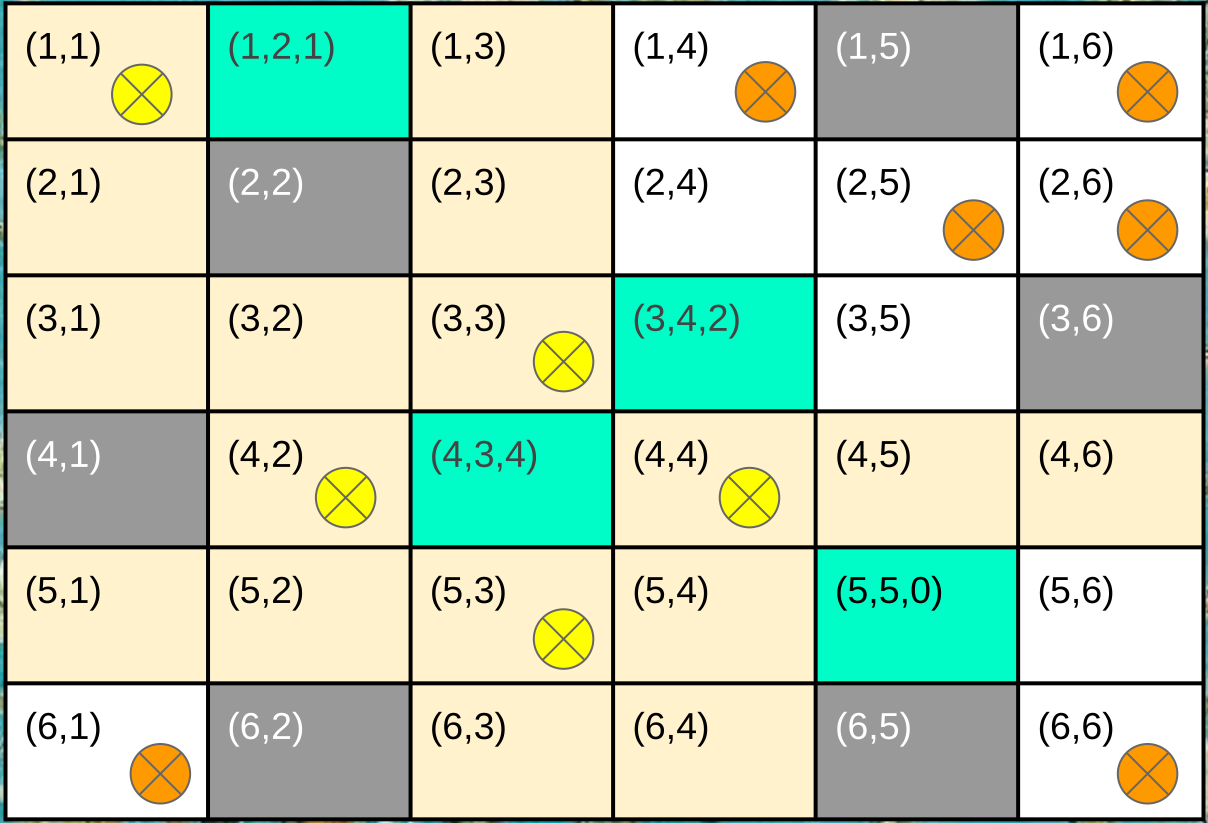
(1,1) ⊗	(1,2,1)	(1,3)	(1,4)	(1,5)	(1,6)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,1)	(3,2)	(3,3) ⊗	(3,4,2)	(3,5)	(3,6)
(4,1)	(4,2) ⊗	(4,3,4)	(4,4) ⊗	(4,5)	(4,6)
(5,1)	(5,2)	(5,3) ⊗	(5,4)	(5,5,0)	(5,6)
(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)

Step 6

For every white square, generate bulb such that it is not neighbor of any numBlack square.

```
{bulb(X,Y): numBlack(X1,Y1,R), not
neighbor(X1,Y,1X,Y)} :- white(X,Y), row(X),
column(Y).

:- white(X,Y), not light(X,Y).
```

Sample Akari Puzzle



(1,1) ⊗	(1,2,1)	(1,3)	(1,4) ⊗	(1,5)	(1,6) ⊗
(2,1)	(2,2)	(2,3)	(2,4)	(2,5) ⊗	(2,6) ⊗
(3,1)	(3,2)	(3,3) ⊗	(3,4,2)	(3,5)	(3,6)
(4,1)	(4,2) ⊗	(4,3,4)	(4,4) ⊗	(4,5)	(4,6)
(5,1)	(5,2)	(5,3) ⊗	(5,4)	(5,5,0)	(5,6)
(6,1) ⊗	(6,2)	(6,3)	(6,4)	(6,5)	(6,6) ⊗

Step 7

If there is no black square between two bulbs that may illumination each other, program will eliminate one of the bulbs.

Checks same rows/columns for such cases in 4 main direction and decides which bulb would be sufficient.

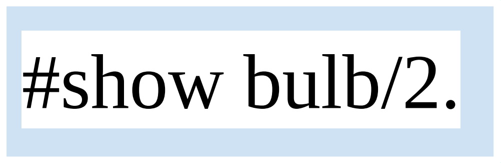
```
:- bulb(X,Y), bulb(X,Y1), Y !=
Y1, Y<Y1,{black(X,Y2): Y < Y2,
Y2 < Y1}0. %right

:- bulb(X,Y), bulb(X,Y1), Y !=
Y1, Y>Y1,{black(X,Y2): Y > Y2,
Y2 > Y1}0. %left

:- bulb(X,Y), bulb(X1,Y), X !=
X1, X<X1,{black(X2,Y): X < X2,
X2 < X1}0. %up

:- bulb(X,Y), bulb(X1,Y), X !=
X1, X>X1,{black(X2,Y): X > X2,
X2 > X1}0. %down
```

Sample Akari Puzzle

(1,1) ⊗	(1,2,1)	(1,3)	(1,4) ⊗	(1,5)	(1,6) ⊗
(2,1)	(2,2)	(2,3)	(2,4)	(2,5) ⊗	(2,6)
(3,1)	(3,2)	(3,3) ⊗	(3,4,2)	(3,5)	(3,6)
(4,1)	(4,2) ⊗	(4,3,4)	(4,4) ⊗	(4,5)	(4,6)
(5,1)	(5,2)	(5,3) ⊗	(5,4)	(5,5,0)	(5,6)
(6,1) ⊗	(6,2)	(6,3)	(6,4)	(6,5)	(6,6) ⊗

Step 8

DISPLAY THE BULBS YOU POSITIONED.



```
#show bulb/2.
```

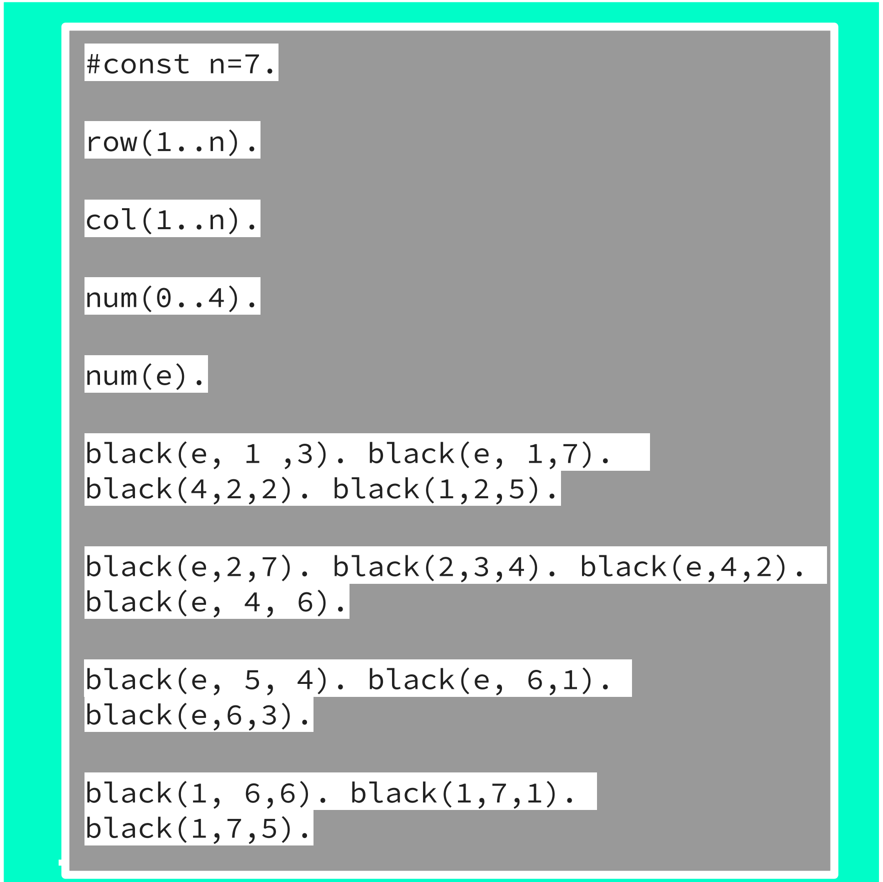
Akari Solution #2

Idil Kapıkıran && Nazlı Gülşah Önen

Step 1

Declaration of inputs.

- Determine the size (n) of the puzzle by specifying the range of rows (row) and columns (col).
- If there is a number inside of black atom the format would be: (num,X,Y). Where range of num is declared. If there is not a number inside of the atom, then the format would be: (e,X,Y).



```
#const n=7.

row(1..n).

col(1..n).

num(0..4).

num(e).

black(e, 1 ,3). black(e, 1,7).
black(4,2,2). black(1,2,5).

black(e,2,7). black(2,3,4). black(e,4,2).
black(e, 4, 6).

black(e, 5, 4). black(e, 6,1).
black(e,6,3).

black(1, 6,6). black(1,7,1).
black(1,7,5).
```

Step 2

Generation of white atoms and lightbulbs.

- Generate white atoms wherever there are no black atoms.

- If the atom is white, there can or cannot be lightbulb inside the white atom.
- Put “_” as num variable to indicate there is a num of either 0..4 or e.

```
white(X, Y):- not black(_,X,Y),row(X),
col(Y).

0{lightbulb(X, Y)}1:-white(X, Y).
```

Step 3

Eliminating lightbulbs in rows and columns.

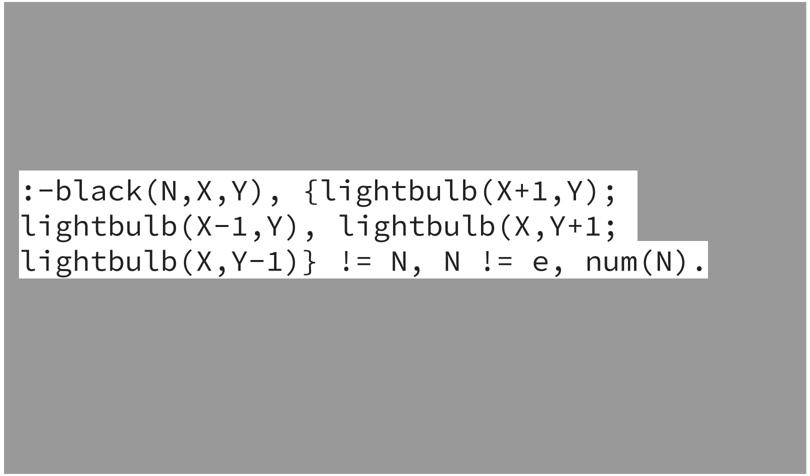
- If there are two lightbulbs in a row or column without a black in between, then eliminate one of the lightbulbs since two of them shouldn’t illuminate each other.

```
:- lightbulb(X,Y),lightbulb(X,Y1),
{black(_,X,Y2): Y< Y2, Y2 < Y1} <= 0, Y< Y1.
```

```
:- lightbulb(X,Y), lightbulb(X1,Y),
{black(_,X2,Y): X< X2, X2 < X1}<= 0, X < X1.
```

Step 4

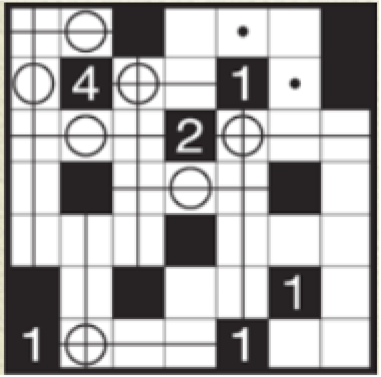
- Place lightbulbs around the black atoms according to the num variable inside their declarations.



```
:-black(N,X,Y), {lightbulb(X+1,Y);
lightbulb(X-1,Y), lightbulb(X,Y+1;
lightbulb(X,Y-1)} != N, N != e, num(N).
```

Step 5

- Assign light atom to the atoms that are in the same row or column with lightbulbs that are previously generated, until there is a black atom or border of the puzzle.



```
light(X,Y1):-lightbulb(X,Y), white(X,Y1),
Y1>=Y, {black(_,X,Y2): Y< Y2, Y2 < Y1} ==0.

light(X,Y1):-lightbulb(X,Y), white(X,Y1),
Y1<=Y, {black(_,X,Y2): Y > Y2, Y2>Y1}== 0.

light(X1,Y):- lightbulb(X,Y), white(X1,Y),
X1 >=X, {black(_,X2,Y): X< X2, X2<X1}==0.}

light(X1,Y):- lightbulb(X,Y), white(X1,Y),
X1 <= X, {black(_,X2,Y): X>X2, X2> X1}== 0.
```

Step 6

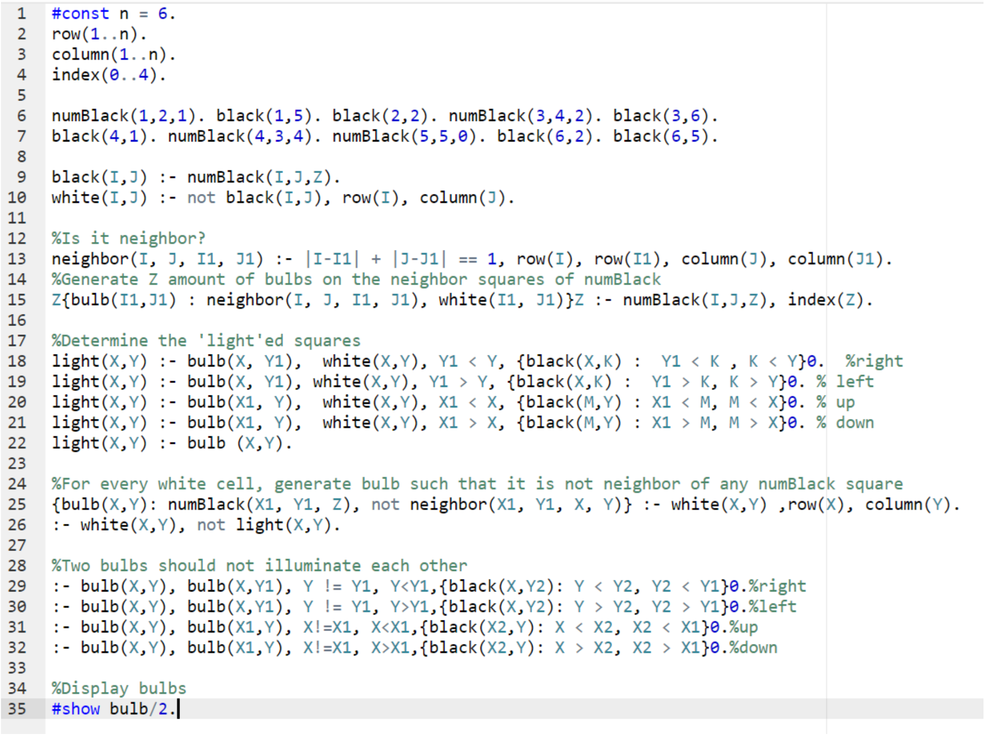
- If there are solutions with white atoms that are not assigned as light atoms, eliminate those solutions.
- Display the the lightbulbs that are placed where “/2” indicates there are two parameters (which are X and Y) inside lightbulbs.

```
:- not light(X,Y), white(X,Y).

#show lightbulb/2.
```

CONCLUSION:

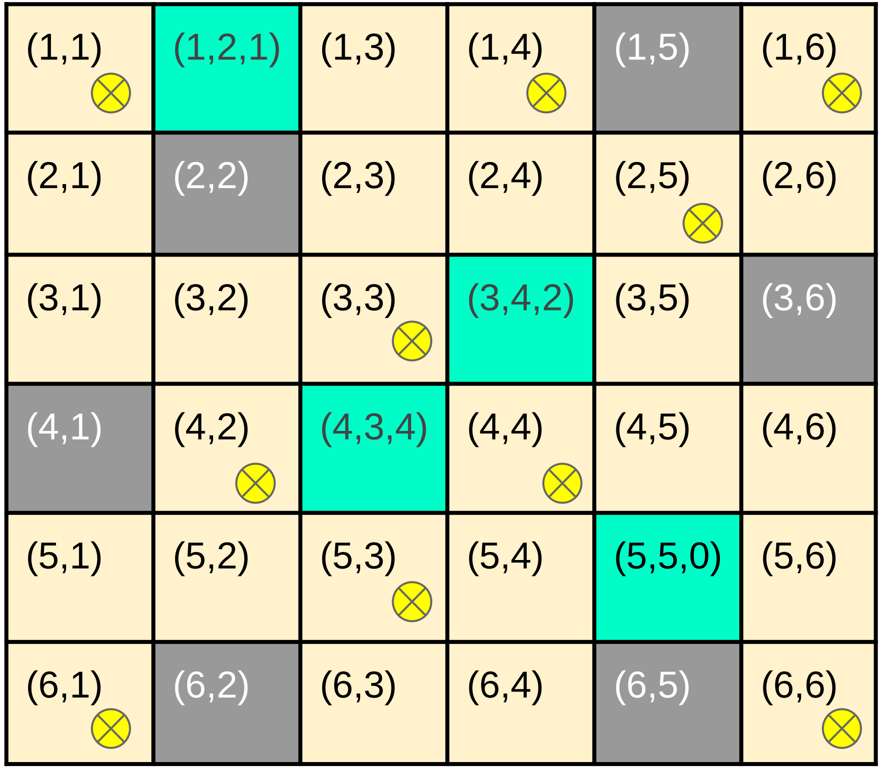
BOTH OF OUR PROGRAMS ABLE TO SOLVES ALL POSSIBLE AKARI PUZZLES.

SOLUTION#1 ON CLINGO



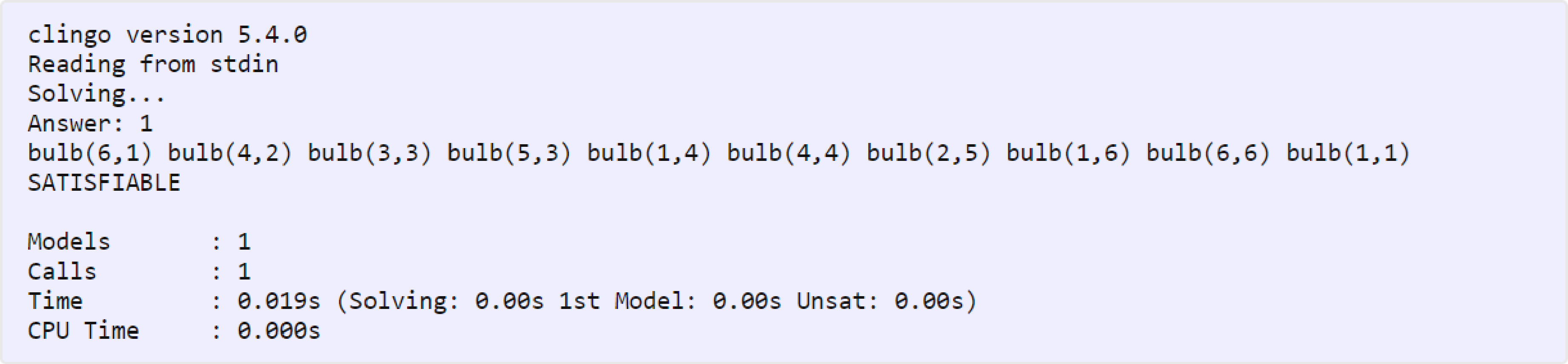
```
#const n = 6.
row(1..n).
column(1..n).
index(0..4).

numBlack(1,2,1). black(1,5). black(2,2). numBlack(3,4,2). black(3,6).
black(4,1). numBlack(4,3,4). numBlack(5,5,0). black(6,2). black(6,5).

black(I,J) :- numBlack(I,J,Z).
white(I,J) :- not black(I,J), row(I), column(J).

%Is it neighbor?
neighbor(I, J, I1, J1) :- |I-I1| + |J-J1| == 1, row(I), row(I1), column(J), column(J1).
%Generate Z amount of bulbs on the neighbor squares of numBlack
Z{bulb(I1,J1) : neighbor(I, J, I1, J1), white(I1, J1)}Z :- numBlack(I,J,Z), index(Z).

%Determine the 'light'ed squares
light(X,Y) :- bulb(X, Y1),  white(X,Y), Y1 < Y, {black(X,K) :  Y1 < K , K < Y}0.  %right
light(X,Y) :- bulb(X, Y1), white(X,Y), Y1 > Y, {black(X,K) :  Y1 > K, K > Y}0. % left
light(X,Y) :- bulb(X1, Y),  white(X,Y), X1 < X, {black(M,Y) : X1 < M, M < X}0. % up
light(X,Y) :- bulb(X1, Y),  white(X,Y), X1 > X, {black(M,Y) : X1 > M, M > X}0. % down
light(X,Y) :- bulb (X,Y).

%For every white cell, generate bulb such that it is not neighbor of any numBlack square
{bulb(X,Y): numBlack(X1, Y1, Z), not neighbor(X1, Y1, X, Y)} :- white(X,Y) ,row(X), column(Y).
:- white(X,Y), not light(X,Y).

%Two bulbs should not illuminate each other
:- bulb(X,Y), bulb(X,Y1), Y != Y1, Y<Y1,{black(X,Y2): Y < Y2, Y2 < Y1}0.%right
:- bulb(X,Y), bulb(X,Y1), Y != Y1, Y>Y1,{black(X,Y2): Y > Y2, Y2 > Y1}0.%left
:- bulb(X,Y), bulb(X1,Y), X!=X1, X<X1,{black(X2,Y): X < X2, X2 < X1}0.%up
:- bulb(X,Y), bulb(X1,Y), X!=X1, X>X1,{black(X2,Y): X > X2, X2 > X1}0.%down

%Display bulbs
#show bulb/2.
```

MODEL OF THE PUZZLE



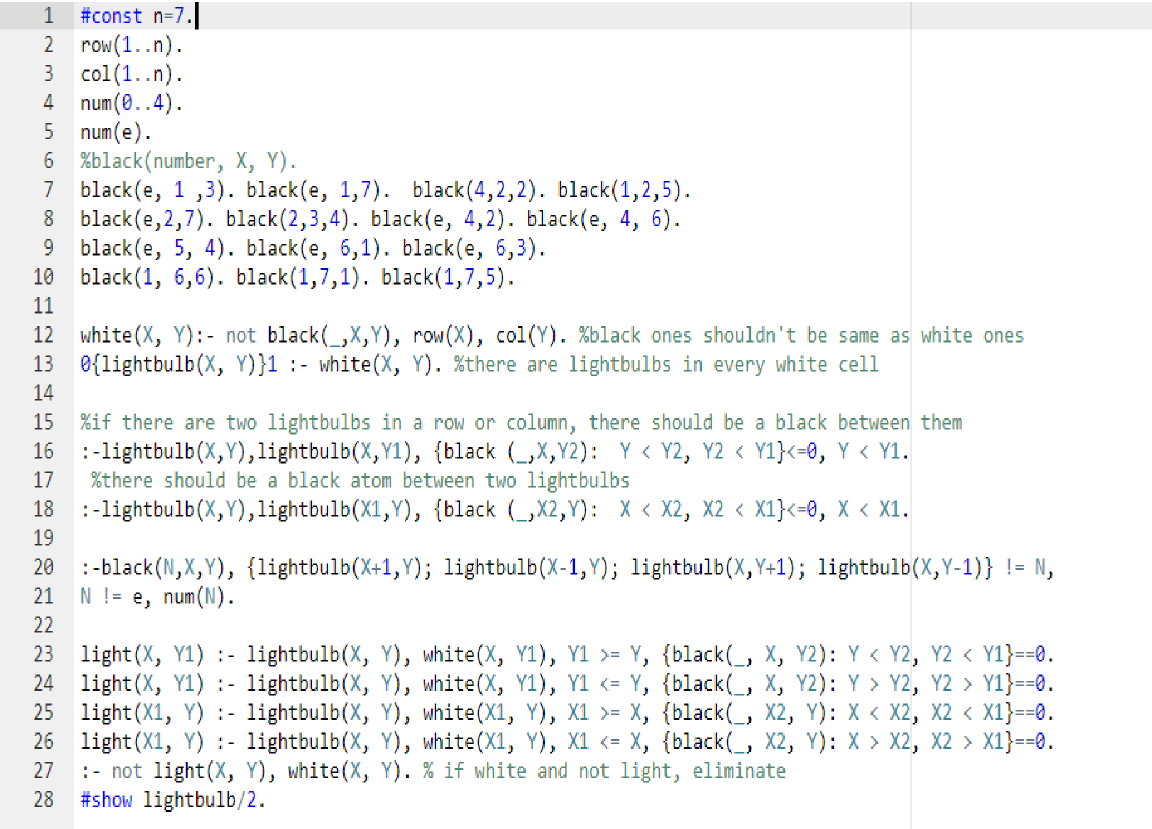
(1,1) ⊗	(1,2,1)	(1,3)	(1,4) ⊗	(1,5)	(1,6) ⊗
(2,1)	(2,2)	(2,3)	(2,4)	(2,5) ⊗	(2,6)
(3,1)	(3,2)	(3,3) ⊗	(3,4,2)	(3,5)	(3,6)
(4,1)	(4,2) ⊗	(4,3,4)	(4,4) ⊗	(4,5)	(4,6)
(5,1)	(5,2)	(5,3) ⊗	(5,4)	(5,5,0)	(5,6)
(6,1) ⊗	(6,2)	(6,3)	(6,4)	(6,5)	(6,6) ⊗

SOLUTION#1 - OUTPUT



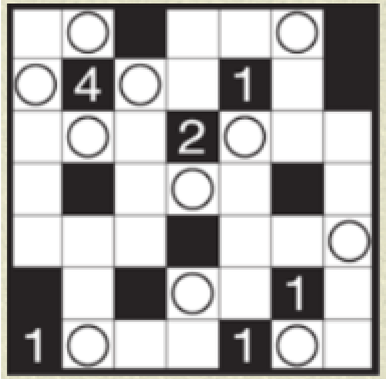
```
clingo version 5.4.0
Reading from stdin
Solving...
Answer: 1
bulb(6,1) bulb(4,2) bulb(3,3) bulb(5,3) bulb(1,4) bulb(4,4) bulb(2,5) bulb(1,6) bulb(6,6) bulb(1,1)
SATISFIABLE

Models       : 1
Calls        : 1
Time         : 0.019s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

SOLUTION#2 ON CLINGO

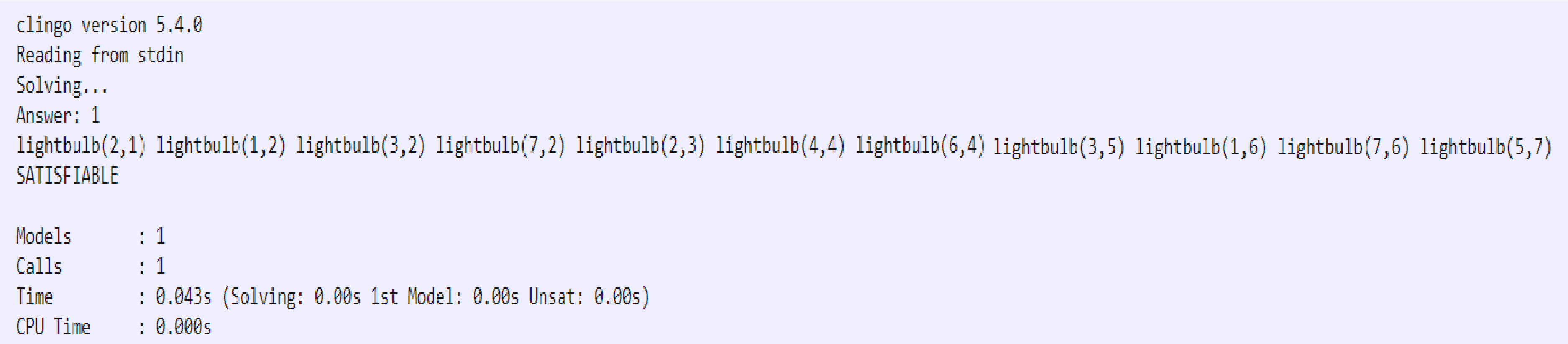


```
#const n=7.
row(1..n).
col(1..n).
num(0..4).
num(e).
%black(number, X, Y).
black(e, 1 ,3). black(e, 1,7).  black(4,2,2). black(1,2,5).
black(e,2,7). black(2,3,4). black(e, 4,2). black(e, 4, 6).
black(e, 5, 4). black(e, 6,1). black(e, 6,3).
black(1, 6,6). black(1,7,1). black(1,7,5).

white(X, Y):- not black(_,X,Y), row(X), col(Y). %black ones shouldn't be same as white ones
0{lightbulb(X, Y)}1 :- white(X, Y). %there are lightbulbs in every white cell

%if there are two lightbulbs in a row or column, there should be a black between them
:-lightbulb(X,Y),lightbulb(X,Y1), {black (_,X,Y2):  Y < Y2, Y2 < Y1}<=0, Y < Y1.
 %there should be a black atom between two lightbulbs
:-lightbulb(X,Y),lightbulb(X1,Y), {black (_,X2,Y):  X < X2, X2 < X1}<=0, X < X1.

:-black(N,X,Y), {lightbulb(X+1,Y); lightbulb(X-1,Y); lightbulb(X,Y+1); lightbulb(X,Y-1)} != N,
N != e, num(N).

light(X, Y1) :- lightbulb(X, Y), white(X, Y1), Y1 >= Y, {black(_, X, Y2): Y < Y2, Y2 < Y1}==0.
light(X, Y1) :- lightbulb(X, Y), white(X, Y1), Y1 <= Y, {black(_, X, Y2): Y > Y2, Y2 > Y1}==0.
light(X1, Y) :- lightbulb(X, Y), white(X1, Y), X1 >= X, {black(_, X2, Y): X < X2, X2 < X1}==0.
light(X1, Y) :- lightbulb(X, Y), white(X1, Y), X1 <= X, {black(_, X2, Y): X > X2, X2 > X1}==0.
:- not light(X, Y), white(X, Y). % if white and not light, eliminate
#show lightbulb/2.
```

MODEL OF THE PUZZLE



SOLUTION#2 - OUTPUT



```
clingo version 5.4.0
Reading from stdin
Solving...
Answer: 1
lightbulb(2,1) lightbulb(1,2) lightbulb(3,2) lightbulb(7,2) lightbulb(2,3) lightbulb(4,4) lightbulb(6,4) lightbulb(3,5) lightbulb(1,6) lightbulb(7,6) lightbulb(5,7)
SATISFIABLE

Models       : 1
Calls        : 1
Time         : 0.043s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Thank you !