Middle Sum

Álvaro Francisco Barbosa Miranda

[1] Faculdade de Engenharia da Universidade do Porto, Portugal
[2] Programação Lógica
[3] Turma 2
Grupo 3

Abstract. In this paper we will talk about the resolution of middle sum problems using logic programming and restrictons.

Keywords: Restrictions · Logic Programming · Backtracking · Recurssion.

1 Introduction

Objective The objectives of this work, as proposed by the teacher, is the resolution of a restriction problem, in this case Midle Sum, using Prolog language using the "clpfd" library.

This article will be divided in the following way:
- Introduction
- Problem Description
- Approach
a) Decision Variables
b) Constraints
c) Evaluation Function
d) Search Strategy
- Solution Presentation
- Results
- Conclusions and Future Work
- References
- Annex

2 Problem Description

First the board of Middle Sum is not fixed. He can have any size if the board remains a square and the side is at least 4 or higher. The objective of the game is to have in each row and column 3 numbers different from 0 and the one in the middle is equal to the sum of the other two. There can not be negative numbers or numbers higher than 9. A board to be resolved only have 1 number in each column and line.

3				
				3
		5		
			1	
	6			

Fig. 1. example of a initial board 5x5.

3 Approach

For the resolution of the problem first we used a list of lists containing the cells of the board. If the cell is empty there is a 0 instead. This way we only have to restrict the numbers to the rules previously said and we will obtain the values of the solution. The predicate used to find a solution is **midleSum(+Board, -MatrixFinal)** or **midleSum2(-Board)**, depending if the board is given previously or is randomly generated.

3.1 Decision Variables

In this project the only decision variables existing are the **Boards**.Supposedly there is only one solution for each board. So the only thing we have to choose is the board we want to resolve or in the case of the random generator we can choose the size of the board so it could be random generator for that size.

3.2 Constraints

There is very few constraints in this problem. The principal is that of the numbers different from 0 the one in the middle is equal to the sum of the other 2. These numbers locate in rows and in columns so it has to respect from both directions. The others constraints are consequence o this one, like the number of zeros in a line or column is the equal to the size o the side of the board minus 3(numbers). And the domain of all numbers of the board must be lower or equal than 9 and higher or equal of 0. These are checked in the predicate **restrictLine(+List)**.

3.3 Search Strategy

There is a predicate that applys the constraints to a matrix called **restrictMatrix([+Matrix])**. Because it is needed to check both horizontally and vertically first we check horizontally. After we check this one we transpose the same matrix and after that we check again horizontally that represents the previously columns. After that we transpose again the matrix so it can be in the right

direction. After this we only need to pass the list of lists that represent the board to a single list. To do that we use **flatten(+Matrix, -FlatenedList)**. After this we use labelling and obtain the solution of the board but only in a single list. We then use **list2Matrix(+List, +Columns, -Matrix)** to obtain the final solution in a list of lists. All this is implemented in the predicate **midleSum(+Board, -MatrixFinal)**.

3.4 Random problem generator

It was also created a random problem generator. First it try to create an empty board by putting only one random number in the domain 1-9 in each column and line. This means that independently of the row or column you look for there will always be one and only one number different from 0. After this it will look for the solution by a similar method described in 3.3 section. If no solution is the process starts from the beginning till it found a possible board to solve and in fact be solved.

```
generateTry(Size):-
    generateBoard(Size, G),
    write('Seeking board'),nl,
    reset_timer,
    (
        (midleSum2(G), nl, displayTime,nl);
        generateTry(Size)
    ).
```

Fig. 2. random board generator and solving predicate.

4 Solution Presentation

First it is shown the initial board to see from what board the solution is find. After that is shown the solution of that same board. After that is also shown the amount of the time the problem took to be resolved.

The predicates used are:

displayB(+Board)-¿ To draw the entire board independently of the step

displayBoard(+Board, +Size, +Step)-¿ Depending on the step it will print bot the numbers and seperators

displayLine(+Line)-¿ Draw the numbers

displaySeparator(+Size)-¿ Draw the seperators

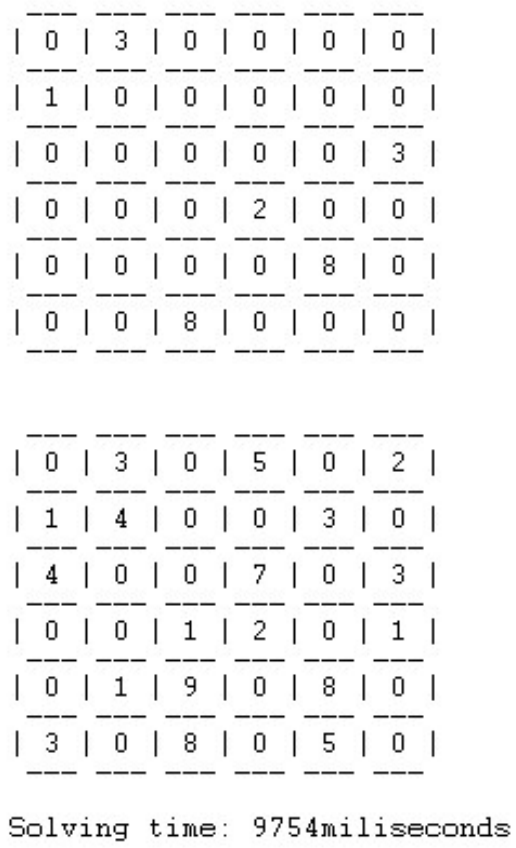


Fig. 3. console display of the output of the program.

5 Results

As we can see from the table below there is not much we can conclude. This happens because the time is not certain and the more we try, more it change experience to experience. But we can conclude that the amount of time is not always proportional as the size of the board as we can see by the last column. We can also conclude that to see which options are better we mustn't use a small board because that way we can't see much difference in the times of execution. Also we can see that the one these are the one that obtains better results uses all the

min The leftmost variable with the smallest lower bound is selected.

step Makes a binary choice between X = B and X $\bar{B}$, where B is the lower or upper bound of X. This is the default.

up The domain is explored in ascending order. This is the default.

Size	4	5	6	7	8	9	10	
min	0,08	0,11	0,15	0,24	4,58	0,67	0,89	(in seconds)
max	0,08	0,12	0,37	0,9	>15	>20	0,54	
min, bisect	0,08	0,1	0,2	0,94	>20	19,99	>27,1	
min, enum	0,08	0,11	0,4	2,77	>25	>25	>25	
min, step	0,08	0,11	0,17	4,84	5,17	0,64	0,86	
min, down	0,08	0,15	3,73	6,65	>15	5,5	>20	
ff	0,08	0,12	0,44	1,1	>15	>20	0,66	
ffc	0,08	0,12	0,45	1,12	>15	>20	0,66	
all default	0,08	0,12	0,34	0,86	>30	18,69	0,52	

Fig. 4. results of experiments with the given boards.

6 Conclusions and Future Work

Reaching the work end we can conclude that using constraints can be very beneficial to solve certain problems and optimization.

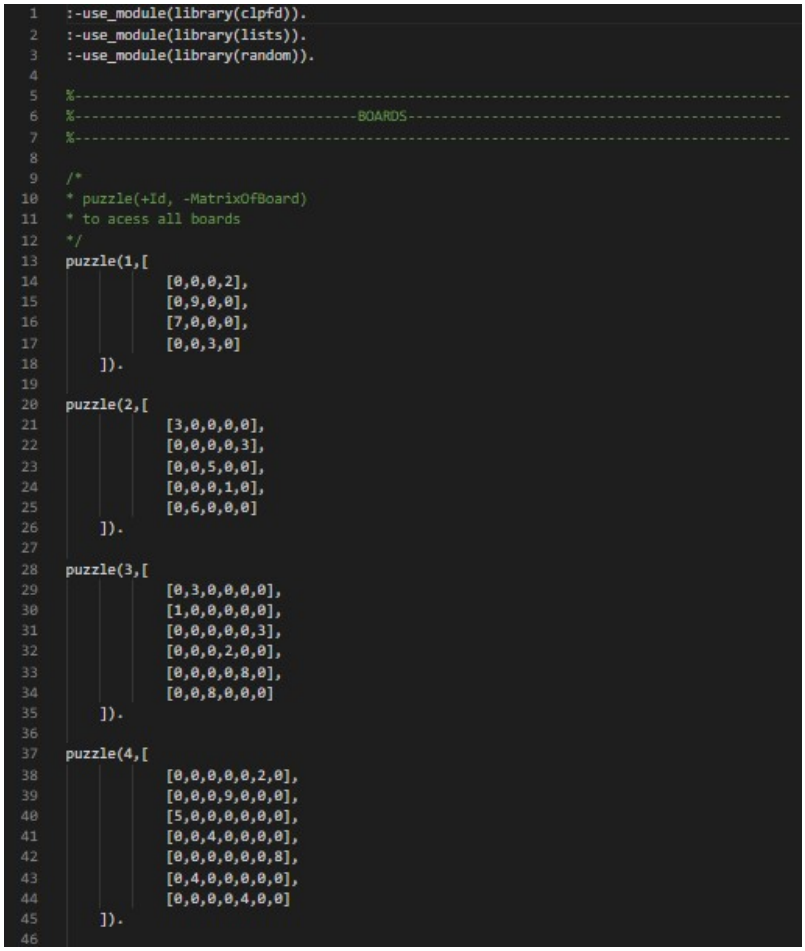
Although I had to do the problem alone, the low complexity of the problem helped. The most difficult part was the random problem generator because when the labelling didn't find a option the program would crash and this created many problems resolving.

As much for conclusions it was also difficult because some option in certain boards took too long and in others they were the fastest which was very hard to get some conclusions.

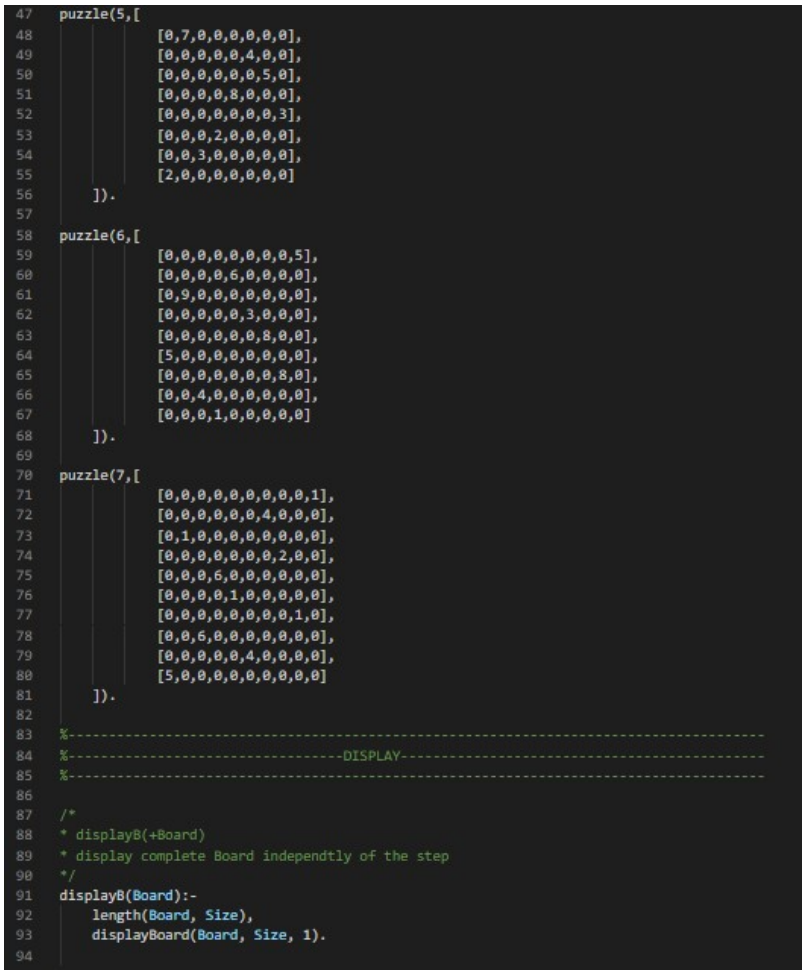
7 References

https://www2.stetson.edu/ efriedma/puzzle/middle/.

8 Annex



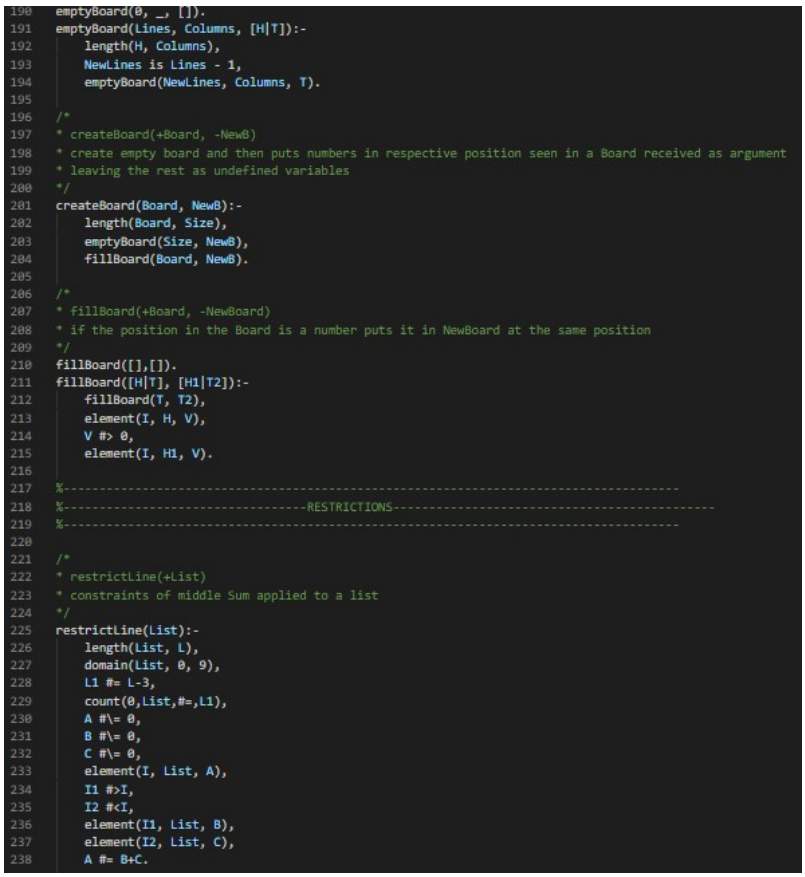
```
:-use_module(library(clpfd)).
:-use_module(library(lists)).
:-use_module(library(random)).

%-------------------------------------------------------------------------------------
%-----------------------------------BOARDS--------------------------------------------
%-------------------------------------------------------------------------------------

/*
* puzzle(+Id, -MatrixOfBoard)
* to acess all boards
*/
puzzle(1,[
            [0,0,0,2],
            [0,9,0,0],
            [7,0,0,0],
            [0,0,3,0]
    ]).

puzzle(2,[
            [3,0,0,0,0],
            [0,0,0,0,3],
            [0,0,5,0,0],
            [0,0,0,1,0],
            [0,6,0,0,0]
    ]).

puzzle(3,[
            [0,3,0,0,0,0],
            [1,0,0,0,0,0],
            [0,0,0,0,0,3],
            [0,0,0,2,0,0],
            [0,0,0,0,8,0],
            [0,0,8,0,0,0]
    ]).

puzzle(4,[
            [0,0,0,0,0,2,0],
            [0,0,0,9,0,0,0],
            [5,0,0,0,0,0,0],
            [0,0,4,0,0,0,0],
            [0,0,0,0,0,0,8],
            [0,4,0,0,0,0,0],
            [0,0,0,0,4,0,0]
    ]).

```



```
puzzle(5,[
            [0,7,0,0,0,0,0,0],
            [0,0,0,0,0,4,0,0],
            [0,0,0,0,0,0,5,0],
            [0,0,0,0,8,0,0,0],
            [0,0,0,0,0,0,0,3],
            [0,0,0,2,0,0,0,0],
            [0,0,3,0,0,0,0,0],
            [2,0,0,0,0,0,0,0]
    ]).

puzzle(6,[
            [0,0,0,0,0,0,0,5],
            [0,0,0,0,6,0,0,0],
            [0,9,0,0,0,0,0,0],
            [0,0,0,0,0,3,0,0],
            [0,0,0,0,0,0,8,0],
            [5,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,8,0],
            [0,0,4,0,0,0,0,0],
            [0,0,0,1,0,0,0,0]
    ]).

puzzle(7,[
            [0,0,0,0,0,0,0,0,1],
            [0,0,0,0,0,0,4,0,0],
            [0,1,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,2,0],
            [0,0,0,6,0,0,0,0,0],
            [0,0,0,0,1,0,0,0,0],
            [0,0,0,0,0,0,0,1,0],
            [0,0,6,0,0,0,0,0,0],
            [0,0,0,0,0,4,0,0,0],
            [5,0,0,0,0,0,0,0,0]
    ]).

%-------------------------------------------------------------------------------------
%----------------------------------DISPLAY--------------------------------------------
%-------------------------------------------------------------------------------------

/*
* displayB(+Board)
* display complete Board independtly of the step
*/
displayB(Board):-
    length(Board, Size),
    displayBoard(Board, Size, 1).

```

```
96  * displayBoard(+Board, +Size, +Step)
97  * display complete Board depending on step
98  */
99  displayBoard([], _, _).
100 displayBoard(Board, Size, 1):-
101     displaySeparator(Size), nl,
102     displayBoard(Board, Size, 2).
103 displayBoard([H|T], Size, 2):-
104     write('| '),
105     displayLine(H), nl,
106     displaySeparator(Size), nl,
107     displayBoard(T,Size, 2).
108
109 /*
110 * displayLine(+List)
111 * display complete List
112 */
113 displayLine([]).
114 displayLine([H | T]) :-
115     write(H), write(' | '),
116     displayLine(T).
117
118 /*
119 * displaySeparator(+Size)
120 * display line separator depending on Size
121 */
122 displaySeparator(0).
123 displaySeparator(Size):-
124     NewSize is Size - 1,
125     write(' ---'),
126     displaySeparator(NewSize).
127
128 %-------------------------------------------------------------------------------
129 %---------------------------------AUXILIAR--------------------------------------
130 %-------------------------------------------------------------------------------
131
132 /*
133 * flatten(+Matrix, -FlatenedList)
134 * transforms a Matrix in a List
135 */
136 flatten([], []).
137 flatten([H|T], List):-
138     flatten(T, List1),
139     append(H, List1, List), !.
140
141 /*
142 * list2Matrix(+List, +Columns, -Matrix)
143 * transforms a Matrix in a List
144 */
```

```
list2Matrix([], _, []).
list2Matrix(List, Size, [Row|Matrix]):-
  list_to_matrix_row(List, Size, Row, Tail),
  list2Matrix(Tail, Size, Matrix).

list_to_matrix_row(Tail, 0, [], Tail).
list_to_matrix_row([Item|List], Size, [Item|Row], Tail):-
  NSize is Size-1,
  list_to_matrix_row(List, NSize, Row, Tail).


/*
* setMatrixAt(+Line, +Column, +NewNumber, +Matrix, -NewBoard)
* set the element at line and column of the board to the newNumber returning the NewBoard
*/
setMatrixNumberAt(1, Column, NewNumber, [HeaderLines|TailLines], [NewHeaderLines|TailLines]):-
    setListNumberAt(Column, NewNumber, HeaderLines, NewHeaderLines).

setMatrixNumberAt(Line, Column, NewNumber, [HeaderLines|TailLines], [HeaderLines|NewTailLines]):-
    Line > 0,
    NewLine is Line-1,
    setMatrixNumberAt(NewLine, Column, NewNumber, TailLines, NewTailLines).

setListNumberAt(1, NewNumber, [_|TailNumbers], [NewNumber|TailNumbers]).

setListNumberAt(Index, NewNumber, [HeaderNumbers|TailNumbers], [HeaderNumbers|NewTailNumbers]):-
    Index > 0,
    NewIndex is Index-1,
    setListNumberAt(NewIndex, NewNumber, TailNumbers, NewTailNumbers).

%-------------------------------------------------------------------------------
%---------------------------------EMPTY BOARD CREATOR-------------------------------------------
%-------------------------------------------------------------------------------

/*
* emptyBoard(+Size, -Board)
* creates a square matrix of undefined variables
*/
emptyBoard(Size, Board):-
    emptyBoard(Size, Size, Board).

/*
* emptyBoard(+Lines, +Columns, -Board)
* creates a Lines X Columns matrix of undefined variables
*/
```

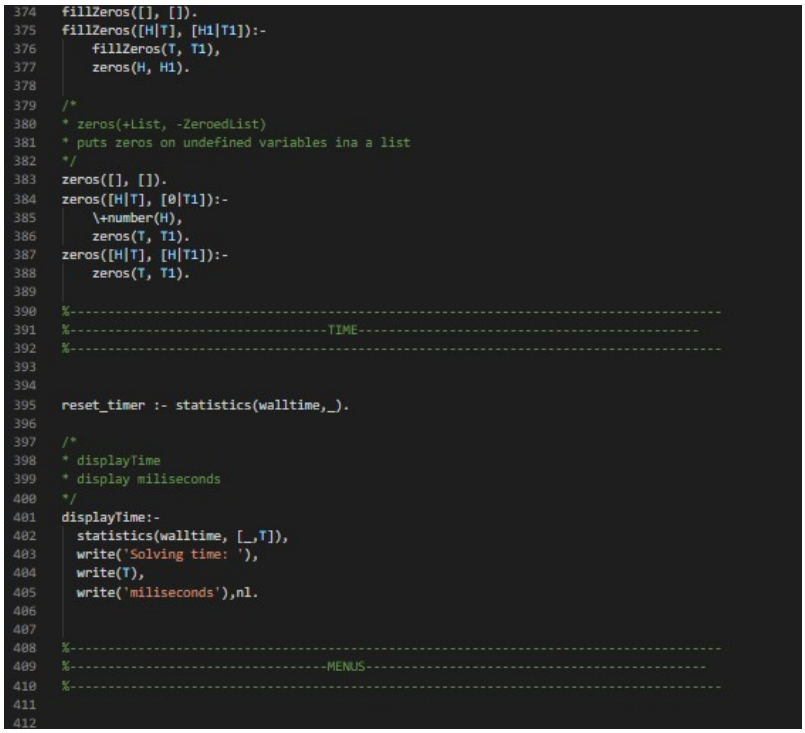


```
emptyBoard(0, _, []).
emptyBoard(Lines, Columns, [H|T]):-
    length(H, Columns),
    NewLines is Lines - 1,
    emptyBoard(NewLines, Columns, T).

/*
* createBoard(+Board, -NewB)
* create empty board and then puts numbers in respective position seen in a Board received as argument
* leaving the rest as undefined variables
*/
createBoard(Board, NewB):-
    length(Board, Size),
    emptyBoard(Size, NewB),
    fillBoard(Board, NewB).

/*
* fillBoard(+Board, -NewBoard)
* if the position in the Board is a number puts it in NewBoard at the same position
*/
fillBoard([],[]).
fillBoard([H|T], [H1|T2]):-
    fillBoard(T, T2),
    element(I, H, V),
    V #> 0,
    element(I, H1, V).

%-------------------------------------------------------------------------------
%----------------------------------RESTRICTIONS---------------------------------
%-------------------------------------------------------------------------------

/*
* restrictLine(+List)
* constraints of middle Sum applied to a list
*/
restrictLine(List):-
    length(List, L),
    domain(List, 0, 9),
    L1 #= L-3,
    count(0,List,#=,L1),
    A #\= 0,
    B #\= 0,
    C #\= 0,
    element(I, List, A),
    I1 #>I,
    I2 #<I,
    element(I1, List, B),
    element(I2, List, C),
    A #= B+C.
```

```
/*
* restrictMatrix(+Matrix)
* constraints of middle Sum applied to a matrix
*/
restrictMatrix([]).
restrictMatrix([H|T]):-
    restrictLine(H),
    restrictMatrix(T).

%-------------------------------------------------------------------------------------
%---------------------------------SOLVER----------------------------------------------
%-------------------------------------------------------------------------------------

/*
* midleSum(+Board, -MatrixFinal)
* applies the constraints and find the solution of Board putting it in MatrixFinal
*/
midleSum(Board, MatrixFinal):-
    length(Board, Size),
    createBoard(Board, NewBoard),
    !,
    restrictMatrix(NewBoard),
    transpose(NewBoard, TransposedBoard),
    !,
    restrictMatrix(TransposedBoard),
    transpose(TransposedBoard, NBoard),

    flatten(NBoard, FinalBoard),
    !,
    labeling([min], FinalBoard),
    list2Matrix(FinalBoard, Size, MatrixFinal).

/*
* midleSum2(+Board)
* used for the random generator, also displays the boards first and solution
*/
```

```
midleSum2(Board):-
    fillZeros(Board, First),
    length(Board, Size),
    !,
    restrictMatrix(Board),
    transpose(Board, TransposedBoard),
    !,
    restrictMatrix(TransposedBoard),
    transpose(TransposedBoard, NBoard),

    flatten(NBoard, FinalBoard),
    !,
    labeling([min], FinalBoard),
    list2Matrix(FinalBoard, Size, MatrixFinal),

    displayB(First),
    displayB(MatrixFinal).


/*
* solvePuzzle(+Number)
* display and solve one of the existing boards
*/
solvePuzzle(Number):-
    puzzle(Number, Board),
    displayB(Board),nl,
    midleSum(Board, Final),nl,
    displayB(Final).

/*
* generateTry(+Size)
* tries to create and solve a board of Size
*/
generateTry(Size):-
    generateBoard(Size, G),
    write('Seeking board'),nl,
    reset_timer,
    (
        (midleSum2(G), nl, displayTime,nl);
        generateTry(Size)
    ).


%-------------------------------------------------------------------------------------
%---------------------------------GENERATOR-------------------------------------------
%-------------------------------------------------------------------------------------

/*
* generateBoard(+Size, -Board)
```

```
325  * generate a board ready to be resolved(with only ine number in each row or column)
326  */
327  generateBoard(Size, Board):-
328      emptyBoard(Size, Bd),
329      generateIndex(Size, Lines),
330      generateIndex(Size, Columns),
331
332      generateNumbers(Bd, Lines, Columns, Board).
333
334  /*
335  * generateNumbers(+Board, +Lines, +Columns, -Final)
336  * return a Final board with only one number in each column or row
337  */
338  generateNumbers(Final, [], [], Final).
339  generateNumbers(Board, Lines, Columns, Final):-
340      length(Lines, Size),
341      random(0, Size, LineIndex),
342      random(0, Size, ColumnIndex),
343      random(1, 10, Number),
344
345      getCoords(Lines, LineIndex, Line, RestLines),
346      getCoords(Columns, ColumnIndex, Column, RestColumns),
347
348      setMatrixNumberAt(Line, Column, Number, Board, NewBoard),
349
350      generateNumbers(NewBoard, RestLines, RestColumns, Final).
351
352  /*
353  * generateIndex(+X, -Index)
354  * create a list from index to later add a number in hat coordinate
355  */
356  generateIndex(0, []).
357  generateIndex(X, Index):-
358      NewX is X - 1,
359      generateIndex(NewX, Append),
360      append(Append, [X], Index).
361
362  /*
363  * getCoords(+List, +Index, -Numb, -Final)
364  * gets one coorinate for row or line and deletes it from list
365  */
366  getCoords(List, Index, Numb, Final):-
367      nth0(Index, List, Numb),
368      delete(List, Numb, Final).
369
370  /*
371  * fillZeros(+Matrix, -ZeroedMatrix)
372  * puts zeros on undefined variables ina a matrix
373  */
```



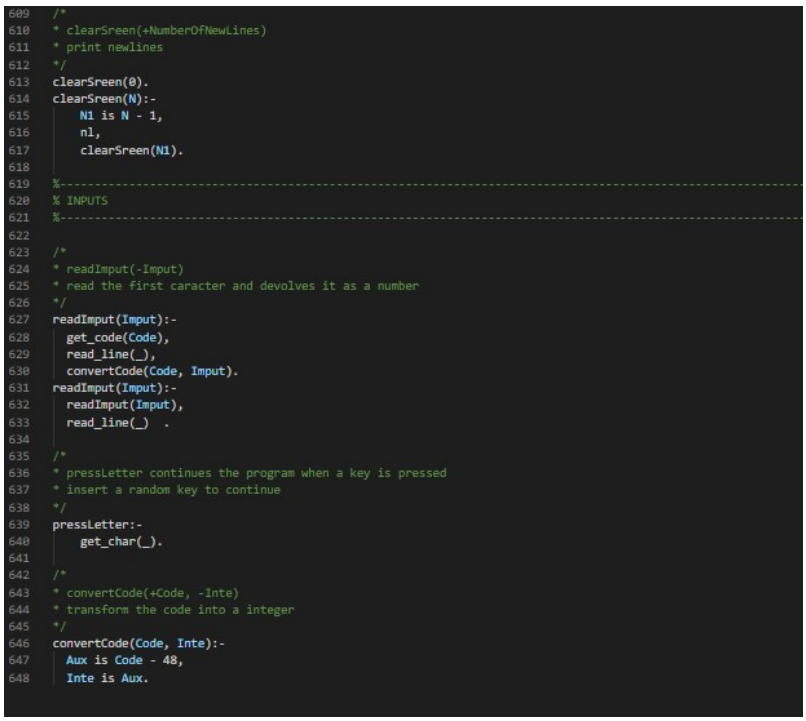
```
374  fillZeros([], []).
375  fillZeros([H|T], [H1|T1]):-
376      fillZeros(T, T1),
377      zeros(H, H1).
378
379  /*
380  * zeros(+List, -ZeroedList)
381  * puts zeros on undefined variables ina a list
382  */
383  zeros([], []).
384  zeros([H|T], [0|T1]):-
385      \+number(H),
386      zeros(T, T1).
387  zeros([H|T], [H|T1]):-
388      zeros(T, T1).
389
390  %--------------------------------------------------------------------------------
391  %---------------------------------TIME-------------------------------------------
392  %--------------------------------------------------------------------------------
393
394
395  reset_timer :- statistics(walltime,_).
396
397  /*
398  * displayTime
399  * display miliseconds
400  */
401  displayTime:-
402    statistics(walltime, [_,T]),
403    write('Solving time: '),
404    write(T),
405    write('miliseconds'),nl.
406
407
408  %--------------------------------------------------------------------------------
409  %---------------------------------MENUS------------------------------------------
410  %--------------------------------------------------------------------------------
411
412
```

```
drawMenu1:-
    repeat,
    write('.......................'), nl,
    write('     MIDLE SUM       '), nl,
    write('.......................'), nl,nl,
    write('1 CHOOSE A PUZZLE'), nl,
    write('2 RANDOM PUZZLE '), nl,
    write('3 QUIT '), nl,
    write('.......................'), nl,nl,
    write('CHOOSE YOUR OPCTION: '), nl,
    readImput(Opction1), nl,
    Opction1 > 0,
    Opction1 =< 3,
    menu(Opction1).



menu(1):-
    clearSreen(40),
    drawMenu2.

menu(2):-
    clearSreen(40),
    drawMenu3.

menu(3):- !.



drawMenu2:-
    repeat,
    write('.......................'), nl,
    write('     DIFFICULTY      '), nl,
    write('.......................'), nl,nl,
    write('1 4x4'), nl,
    write('2 5x5'), nl,
    write('3 6x6 '), nl,
    write('4 7x7 '), nl,
    write('5 8x8 '), nl,
    write('6 9x9 '), nl,
    write('7 10x10 '), nl,
    write('8 BACK '), nl,
    write('9 QUIT '), nl,
    write('.......................'), nl,nl,
    write('CHOOSE YOUR OPCTION: '), nl,
    readImput(Opction2), nl,
    Opction2 > 0,
    Opction2 =< 9,
    menu2(Opction2).
```

```
menu2(1):-
    reset_timer,
    solvePuzzle(1),
nl,displayTime,nl,
    write('Press enter to continue'),nl,
    pressLetter,
    clearSreen(40),
    drawMenu1.

menu2(2):-
    solvePuzzle(2),
    nl,displayTime,nl,
    write('Press enter to continue'),nl,
    pressLetter,
    clearSreen(40),
    drawMenu1.

menu2(3):-
    solvePuzzle(3),
    nl,displayTime,nl,
    write('Press enter to continue'),nl,
    pressLetter,
    clearSreen(40),
    drawMenu1.

menu2(4):-
    solvePuzzle(4),
    nl,displayTime,nl,
    write('Press enter to continue'),nl,
    pressLetter,
    clearSreen(40),
    drawMenu1.

menu2(5):-
    solvePuzzle(5),
    nl,displayTime,nl,
    write('Press enter to continue'),nl,
    pressLetter,
    clearSreen(40),
    drawMenu1.

menu2(6):-
    solvePuzzle(6),
    nl,displayTime,nl,
    write('Press enter to continue'),nl,
    pressLetter,
    clearSreen(40),
    drawMenu1.
```

```
514  menu2(7):-
515      solvePuzzle(7),
516      nl,displayTime,nl,
517      write('Press enter to continue'),nl,
518      pressLetter,
519      clearSreen(40),
520      drawMenu1.
521
522  menu2(8):-
523      drawMenu1.
524
525  menu2(9):- !.
526
527
528  drawMenu3:-
529      repeat,
530      write('.....................'), nl,
531      write('     DIFFICULTY      '), nl,
532      write('.....................'), nl,nl,
533      write('1 4x4'), nl,
534      write('2 5x5'), nl,
535      write('3 6x6 '), nl,
536      write('4 7x7 '), nl,
537      write('5 8x8 '), nl,
538      write('6 9x9 '), nl,
539      write('7 10x10 '), nl,
540      write('8 BACK '), nl,
541      write('9 QUIT '), nl,
542      write('.....................'), nl,nl,
543      write('CHOOSE YOUR OPCTION: '), nl,
544      readImput(Opction3), nl,
545      Opction3 > 0,
546      Opction3 =< 9,
547      menu3(Opction3).
548
549
550  menu3(1):-
551      generateTry(4),
552      write('Press enter to continue'),nl,
553      pressLetter,
554      clearSreen(40),
555      drawMenu1.
556
557  menu3(2):-
558      generateTry(5),
559      write('Press enter to continue'),nl,
560      pressLetter,
561      clearSreen(40),
562      drawMenu1.
```

```
564  menu3(3):-
565      generateTry(6),
566      write('Press enter to continue'),nl,
567      pressLetter,
568      clearSreen(40),
569      drawMenu1.
570
571  menu3(4):-
572      generateTry(7),
573      write('Press enter to continue'),nl,
574      pressLetter,
575      clearSreen(40),
576      drawMenu1.
577
578  menu3(5):-
579      generateTry(8),
580      write('Press enter to continue'),nl,
581      pressLetter,
582      clearSreen(40),
583      drawMenu1.
584
585  menu3(6):-
586      generateTry(9),
587      write('Press enter to continue'),nl,
588      pressLetter,
589      clearSreen(40),
590      drawMenu1.
591
592  menu3(7):-
593      generateTry(10),
594      write('Press enter to continue'),nl,
595      pressLetter,
596      clearSreen(40),
597      drawMenu1.
598
599  menu3(8):-
600      drawMenu1.
601
602  menu3(9):- !.
603
604
605  play:-
606      drawMenu1.
607
608
```



```
/*
* clearSreen(+NumberOfNewLines)
* print newlines
*/
clearSreen(0).
clearSreen(N):-
    N1 is N - 1,
    nl,
    clearSreen(N1).

%-----------------------------------------------------------------------------------------------------------
% INPUTS
%-----------------------------------------------------------------------------------------------------------

/*
* readImput(-Imput)
* read the first caracter and devolves it as a number
*/
readImput(Imput):-
  get_code(Code),
  read_line(_),
  convertCode(Code, Imput).
readImput(Imput):-
  readImput(Imput),
  read_line(_)  .

/*
* pressLetter continues the program when a key is pressed
* insert a random key to continue
*/
pressLetter:-
    get_char(_).

/*
* convertCode(+Code, -Inte)
* transform the code into a integer
*/
convertCode(Code, Inte):-
  Aux is Code - 48,
  Inte is Aux.
```