Aleppo University

Informatics Engineering Faculty

Build Sudoku

9	1	3				5		
6		7					2	4
	5			8			7	
	7	9						
		2		9			4	3
					4		9	
	4				1	9		
7		6			9			5
		1			6	4		7

Supervised By:

Hadeel Abd Al-Rahman

Worked By:

Mahmoud Bannan

Mahmoud Abd Al-Rhim

Introduction:

Focusing on history and development:

Sudoku is a popular logic-based number-placement puzzle. The objective is to fill a 9 × 9 grid with digits so that each column, each row, and each of the nine 3 × 3 subgrids contains all of the digits from 1 to 9 only once. While its exact origins are uncertain, the modern form of Sudoku is believed to have originated in late 20th-century Japan, where it gained immense popularity and eventually spread worldwide. Beyond being a fun pastime, Sudoku has been shown to enhance cognitive abilities, improve focus, and boost logical reasoning skills.

Sudoku is more than just an entertaining puzzle; it's a powerful educational tool. The game helps develop a wide range of cognitive skills, including:

* Focus and attention: Solving Sudoku puzzles requires intense focus and the ability to switch between different parts of the grid

How To Build:

Creating a Sudoku puzzle involves starting with a completely filled grid, then systematically removing numbers until you reach a specific difficulty level. Here are the basic steps:

* Create a complete grid: Fill a 9x9 grid with numbers from 1 to 9, ensuring that each row, column, and 3x3 subgrid contains each number only once.

* Remove numbers: Start removing numbers one by one. The more numbers you remove, the harder the puzzle becomes.

* Check for uniqueness: Ensure that the puzzle has only one unique solution. You can do this by solving the puzzle yourself or using a computer program.

- Build SUDOKU -

```
% generate a new cell or override an old one
generateCell(X,Y,[V]):-retract(cell(X,Y,_,_)),random(1,10,V),assert(cell(X,Y,[V],0)).
generateCell(X,Y,[V]):-random(1,10,V),assert(cell(X,Y,[V],0)).
%check validation of cell in square (X and Y is the position of the cell to validate its value),
%(MX is the max index of the square horizontally),(SX,SY start of the square horizontally and
vertically)

%It will return a true if the value is valid and false if it's not
validateSquare(X,Y,_,X,Y):-!.
validateSquare(X,Y,MX,SX,SY):- SX=<MX,cell(X,Y,V,_),cell(SX,SY,V1,_),not(isEqual(V,V1)), NX is
SX+1,validateSquare(X,Y,MX,NX,SY),!.
validateSquare(X,Y,MX,SX,SY):- SX=:=MX+1, NY is SY+1, NX is SX-3,validateSquare(X,Y,MX,NX,NY),!.

% check validation of cell in column,return true if it's valid and false if it's not[X,Y is the indexes of
the cell,I is an index]
validateColumn(_,_,10):-!.
validateColumn(X,Y,I):- I=:=Y,NI is I+1,validateColumn(X,Y,NI),!.
validateColumn(X,Y,I):- not(cell(X,I,_,_)),NI is I+1,validateColumn(X,Y,NI),!.
validateColumn(X,Y,I):- cell(X,I,[H|_],_),cell(X,Y,[H1|_],_),H=\=H1,NI is I+1,validateColumn(X,Y,NI),!.
% check validation of cell in row,return true if it's valid and false if it's not[X,Y is the indexes of the
cell,I is an index]

validateRow(_,_,10):-!.
validateRow(X,Y,I):- I=:=X,NI is I+1,validateRow(X,Y,NI),!.
validateRow(X,Y,I):- not(cell(I,Y,_,_)),NI is I+1,validateRow(X,Y,NI),!.
validateRow(X,Y,I):- cell(I,Y,[H|_],_),cell(X,Y,[H1|_],_),H=\=H1,NI is I+1,validateRow(X,Y,NI),!.

% fill a square with unique numbers(X,Y are the start indexes of the square),(MX,MY are the end
indexes of the square)
% I is count of times the cell has been build ,J is count of times the square has been build
% refill the square if the cell has been built 20 times
% refill the the last square or the above square if the square has been built 20 times

fillSquare(_,_,MX,MY,_,10):-MX>3,NSX is MX -5,NSY is MY-2,NMX is MX-
3,clearSquare(NSX,NMX,NSY,MY),SX is MX -5,SY is MY-2,NNMX is MX-
3,fillSquare(SX,SY,NNMX,MY,0,0),
                TSX is MX -2,TSY is MY -2,clearSquare(TSX,MX,TSY,MY),TNSX is MX-2,TNSY is MY-
2,fillSquare(TNSX,TNSY,MX,MY,0,0).
fillSquare(_,_,MX,MY,_,10):-MY>3,NSX is MX -2,NSY is MY-5,NMY is MY-
3,clearSquare(NSX,MX,NSY,NMY),SX is MX -2,SY is MY-5,NNMY is MY-
3,fillSquare(SX,SY,MX,NNMY,0,0),
                TSX is MX -2,TSY is MY -2,clearSquare(TSX,MX,TSY,MY),TNSX is MX-2,TNSY is MY-
2,fillSquare(TNSX,TNSY,MX,MY,0,0).- - - - - - - - - - -
```

```
fillSquare(_,_,MX,MY,20,J):- SX is MX -2,SY is MY -2,clearSquare(SX,MX,SY,MY),NSX is MX-2,NSY is
MY-2,NJ is J+1,fillSquare(NSX,NSY,MX,MY,0,NJ).
fillSquare(_,Y,_,MY,_,_) :- Y > MY.
fillSquare(X,Y,MX,MY,_,J):-X=<MX,generateCell(X,Y,_),SX is (MX-2),SY is (MY-
2),validateSquare(X,Y,MX,SX,SY),
                    validateRow(X,Y,1),validateColumn(X,Y,1),NX is X+1,fillSquare(NX,Y,MX,MY,0,J).
fillSquare(X,Y,MX,MY,I,J):-X=<MX,retract(cell(X,Y,_,_)),NI is I+1,fillSquare(X,Y,MX,MY,NI,J).
fillSquare(X,Y,MX,MY,_,J):-X>MX, NY is Y+1, NX is MX-2,fillSquare(NX,NY,MX,MY,0,J).

% clear all cells in square
clearSquare(_,_,SY,EY):-SY>EY.
clearSquare(SX,EX,SY,EY):-SX=<EX,retract(cell(SX,SY,_,_)),NX is SX +1 ,clearSquare(NX,EX,SY,EY).
clearSquare(SX,EX,SY,EY):-SX=<EX,NX is SX +1 ,clearSquare(NX,EX,SY,EY).
clearSquare(SX,EX,SY,EY):- SX>EX,NY is SY +1,NX is EX -2,clearSquare(NX,EX,NY,EY).
% build solved puzzle
fillPuzzle():-  fillSquare(1,1,3,3,0,0),fillSquare(4,1,6,3,0,0),fillSquare(7,1,9,3,0,0),
          fillSquare(1,4,3,6,0,0),fillSquare(4,4,6,6,0,0),fillSquare(7,4,9,6,0,0),
          fillSquare(1,7,3,9,0,0),fillSquare(4,7,6,9,0,0),fillSquare(7,7,9,9,0,0).

% reset all cell to default value (make the fourth parameter 0)
resetCell(_,10):-!.
resetCell(X,Y):-X<10,cell(X,Y,R,_),retract(cell(X,Y,_,_)),assert(cell(X,Y,R,0)),NX is X+1,resetCell(NX,Y),!.
resetCell(X,Y):-X<10,NX is X+1,resetCell(NX,Y),!.
resetCell(X,Y):-X>9,NY is Y+1,NX is 1,resetCell(NX,NY),!.

% remove some cells from puzzle according to indexes from lists[first list is the Xs, and the second is
Ys]
removeCells([],[]):-!.
removeCells([H1|T1],[H2|T2]):-retract(cell(H1,H2,_,_)),removeCells(T1,T2),!.
removeCells([_|T1],[_|T2]):-removeCells(T1,T2),!.

% remove numbers from the puzzle
clearProb(_,10,_,_).
clearProb(X,Y,LX,LY):-X<10,cell(X,Y,R,_),assert(cell(-1,-
1,R,0)),retract(cell(X,Y,_,_)),solvePuzzle(Res),Res=:=1,append([X],LX,NLX),append([Y],LY,NLY),
          retract(cell(-1,-1,_,_)),resetCell(1,1),removeCells(NLX,NLY),NX is
X+1,clearProb(NX,Y,NLX,NLY).
clearProb(X,Y,LX,LY):-X<10,cell(-1,-1,R,_),retract(cell(X,Y,_,_)),assert(cell(X,Y,R,0)),retract(cell(-1,-
1,_,_)),resetCell(1,1),removeCells(LX,LY),NX is X+1,clearProb(NX,Y,LX,LY).
clearProb(X,Y,LX,LY):-X<10,cell(-1,-1,R,_),assert(cell(X,Y,R,0)),retract(cell(-1,-
1,_,_)),resetCell(1,1),removeCells(LX,LY),NX is X+1,clearProb(NX,Y,LX,LY).
clearProb(X,Y,LX,LY):-X>9,NY is Y+1,NX is 1,clearProb(NX,NY,LX,LY).
% create the puzzle
createPuzzle():-fillPuzzle(),clearProb(1,1,[],[]).
```

- Solve SUDOKU -

```
% merge two list into one
append([],L2,L2).
append([H|T],L2,[H|L3]):-append(T,L2,L3).
% remove element from list (v is the element,L should be empty in the call, [H|T] is the list that we want to remove the element from ,F the list after the delete)
removeElement(_,L,[],F):- append(L,[],F),!.
removeElement(V,L,[H|T],F):-H =\= V ,append(L,[H],NL), removeElement(V,NL,T,F).
removeElement(_,L,[_|T],F):- append(L,T,F),!.
% remove elements from the list (the first parameters is the list of element to remove, the second one the the list to remove from, the third is the result)
removeElements([],R,F):-append(R,[],F).
removeElements([H],R,F):-removeElement(H,[],R,F).
removeElements([H|T],R,F):-removeElement(H,[],R,F1),removeElements(T,F1,F).
% check if the two list are equal
isEqual([],[]).
isEqual([H|T],[H1|T1]):-H=:=H1,isEqual(T,T1).
% check if the list is contains V
contain(V,[H]):-H=:=V.
contain(V,[H|_]):-H=:=V.
contain(V,[_|T]):-contain(V,T).
% search for list of items if contain in other list and return true if on of this items is found ,[first argument is the list to search for the second argument is the list to search in]
containList([],_):-1=:=2.
containList([H|_],L):-contain(H,L).
containList([_|T],L):-containList(T,L).
% fill the cell with all probability(1 -> 9) (X,Y the position of the cell,N is an index should be one , L is list of all probability)
fillProb(X,Y,10,L):-assert(cell(X,Y,L,0)).
fillProb(X,Y,N,L) :- append(L,[N],R), NN is N+1, fillProb(X,Y,NN,R).
% fill all cells with all probability (1->9),(X,Y the first cell position)
fillPuzzleProb(_,10).
fillPuzzleProb(X,Y):- cell(X,Y,_,_),NX is X+1, fillPuzzleProb(NX,Y).
fillPuzzleProb(X,Y):- X<10, fillProb(X,Y,1,[]),NX is X+1, fillPuzzleProb(NX,Y).
fillPuzzleProb(_,Y):- NY is Y+1, fillPuzzleProb(1,NY).
% remove V from probability list in the selected row (V is the value want to remove,[X,Y] position of the cell to ignore,I is an index to start from)
clearRowProb(_,_,_,10,0,0).
clearRowProb(_,_,_,10,1,1).
clearRowProb(V,X,Y,I,S,C):- I =:= X,NI is I+1,clearRowProb(V,X,Y,NI,S,C).
clearRowProb(V,X,Y,I,_,C):-
cell(I,Y,R,_),contain(V,R),removeElement(V,[],R,Res),retract(cell(I,Y,_,_)),assert(cell(I,Y,Res,0)),NI is I+1,clearRowProb(V,X,Y,NI,1,C).
clearRowProb(V,X,Y,I,S,C):- NI is I+1,clearRowProb(V,X,Y,NI,S,C).
```

```
% remove V from probability list in the selected column (V is the value want to remove,[X,Y]
position of the cell to ignore,I is an index to start from)
clearColumnProb(_,_,_,10,1,1).
clearColumnProb(_,_,_,10,0,0).
clearColumnProb(V,X,Y,I,S,C):- I =:=Y,NI is I+1,clearColumnProb(V,X,Y,NI,S,C).
clearColumnProb(V,X,Y,I,_,C):-
cell(X,I,R,_),contain(V,R),removeElement(V,[],R,Res),retract(cell(X,I,_,_)),assert(cell(X,I,Res,0)),NI is
I+1,clearColumnProb(V,X,Y,NI,1,C).
clearColumnProb(V,X,Y,I,S,C):- NI is I+1,clearColumnProb(V,X,Y,NI,S,C).
% remove V from probability list in the selected Square (V is the value want to remove,[X,Y]
position of the cell to ignore,
%[SX,EX,SY,EY] is the boundary of the square indexes)
clearSquareProb(_,_,_,_,_,SY,EY,1,1):-SY>EY.
clearSquareProb(_,_,_,_,_,SY,EY,0,0):-SY>EY.
clearSquareProb(V,X,Y,SX,EX,SY,EY,S,C):-SX=<EX,SX=:=X,SY=:=Y,NSX is SX + 1
,clearSquareProb(V,X,Y,NSX,EX,SY,EY,S,C).
clearSquareProb(V,X,Y,SX,EX,SY,EY,_,C):-
SX=<EX,cell(SX,SY,R,_),contain(V,R),removeElement(V,[],R,Res),retract(cell(SX,SY,_,_)),assert(cell(SX,
SY,Res,0)),
                    NSX is SX+1,clearSquareProb(V,X,Y,NSX,EX,SY,EY,1,C).
clearSquareProb(V,X,Y,SX,EX,SY,EY,S,C):-SX=<EX,NSX is SX + 1
,clearSquareProb(V,X,Y,NSX,EX,SY,EY,S,C).
clearSquareProb(V,X,Y,SX,EX,SY,EY ,S,C):-SX>EX, NSX is EX-2,NSY is
SY+1,clearSquareProb(V,X,Y,NSX,EX,NSY,EY,S,C).
% clear the value of the cell from the probability in the column, row and square of the selected
cell(X,Y is the position of the selected cell
%[SX,EX,SY,EY] is the boundary of the square indexes that the cell in located in)
%! the cell should contain one probability
clearOneProbCell(X,Y,SX,EX,SY,EY,C):-
cell(X,Y,[H|_],0),clearColumnProb(H,X,Y,1,0,CC),clearRowProb(H,X,Y,1,0,CR),clearSquareProb(H,X,Y,S
X,EX,SY,EY,0,CS),
                    retract(cell(X,Y,_,_)),assert(cell(X,Y,[H],1)),C is CC+CR+CS.
clearOneProbCell(X,Y,_,_,_,_,0):-cell(X,Y,_,1).
% remove list of probability from cell in selected row (L is the list to remove,[X1,X2] indexes of cells
to ignore,y is the number of line,I is an index)
clearRowTwoProb(_,_,_,_,10,0,0).
clearRowTwoProb(_,_,_,_,10,1,1).
clearRowTwoProb(L,X1,X2,Y,I,S,C):- (I =:= X1;I =:= X2),NI is I+1,clearRowTwoProb(L,X1,X2,Y,NI,S,C).
clearRowTwoProb(L,X1,X2,Y,I,_,C):-
cell(I,Y,R,_),containList(L,R),removeElements(L,R,Res),retract(cell(I,Y,_,_)),
                    assert(cell(I,Y,Res,0)),NI is I+1,clearRowTwoProb(L,X1,X2,Y,NI,1,C).
clearRowTwoProb(L,X1,X2,Y,I,S,C):- NI is I+1,clearRowTwoProb(L,X1,X2,Y,NI,S,C).
% remove list of probability from cell in selected column (L is the list to remove,[Y1,Y2] indexes of
cells to ignore,X is the number of column,I is an index)
clearColumnTwoProb(_,_,_,_,10,0,0).
```

```
clearColumnTwoProb(_,_,_,_,10,1,1).
clearColumnTwoProb(L,Y1,Y2,X,I,S,C):- (I =:= Y1;I =:= Y2),NI is
I+1,clearColumnTwoProb(L,Y1,Y2,X,NI,S,C).
clearColumnTwoProb(L,Y1,Y2,X,I,_,C):-
cell(X,I,R,_),containList(L,R),removeElements(L,R,Res),retract(cell(X,I,_,_)),
                        assert(cell(X,I,Res,0)),NI is I+1,clearColumnTwoProb(L,Y1,Y2,X,NI,1,C).
clearColumnTwoProb(L,Y1,Y2,X,I,S,C):- NI is I+1,clearColumnTwoProb(L,Y1,Y2,X,NI,S,C).
% remove list of probability from cell in selected Square (L is the list to remove,[X1,X2,Y1,Y2]
indexes of cells to ignore,[SX,EX,SY,EY] is the boundary of the square )
clearSquareTwoProb(_,_,_,_,_,_,_,_,SY,EY,0,0)     :- SY > EY.
clearSquareTwoProb(_,_,_,_,_,_,_,_,SY,EY,1,1)     :- SY > EY.
clearSquareTwoProb(L,X1,Y1,X2,Y2,SX,EX,SY,EY,S,C):- (SX=:=X1,SY=:=Y1;SX=:=X2,SY=:=Y2),NSX is
SX+1,clearSquareTwoProb(L,X1,Y1,X2,Y2,NSX,EX,SY,EY,S,C).
clearSquareTwoProb(L,X1,Y1,X2,Y2,SX,EX,SY,EY,_,C):-
SX=<EX,cell(SX,SY,R,_),containList(L,R),removeElements(L,R,Res),retract(cell(SX,SY,_,_)),
                        assert(cell(SX,SY,Res,0)),NSX is
SX+1,clearSquareTwoProb(L,X1,Y1,X2,Y2,NSX,EX,SY,EY,1,C).
clearSquareTwoProb(L,X1,Y1,X2,Y2,SX,EX,SY,EY,S,C):- SX=<EX,NSX is
SX+1,clearSquareTwoProb(L,X1,Y1,X2,Y2,NSX,EX,SY,EY,S,C).
clearSquareTwoProb(L,X1,Y1,X2,Y2,SX,EX,SY,EY,S,C):- SX>EX,NSX is SX-3,NSY is
SY+1,clearSquareTwoProb(L,X1,Y1,X2,Y2,NSX,EX,NSY,EY,S,C).
% scan row to check if there any cell contain the same two probability of the cell in index [X,Y] and
if there any cell remove this two probability from the other cells in this row
%[X,Y] is the index of the cell to compare with its probability , I is an index
%! this cell should contain just two probability
scanHorizontallyForTwoProbCell(_,_,10,0).
scanHorizontallyForTwoProbCell(X,Y,I,C):-
I=\=X,cell(X,Y,R,_),cell(I,Y,TR,_),isEqual(R,TR),clearRowTwoProb(R,X,I,Y,1,0,C).
scanHorizontallyForTwoProbCell(X,Y,I,C):-NI is I+1,scanHorizontallyForTwoProbCell(X,Y,NI,C).
% scan column to check if there any cell contain the same two probability of the cell in index [X,Y]
and if there any cell remove this two probability from the other cells in this column
%[X,Y] is the index of the cell to compare with its probability , I is an index
%! this cell should contain just two probability
scanVerticallyForTwoProbCell(_,_,10,0).
scanVerticallyForTwoProbCell(X,Y,I,C):-
I=\=Y,cell(X,Y,R,_),cell(X,I,TR,_),isEqual(R,TR),clearColumnTwoProb(R,Y,I,X,1,0,C).
scanVerticallyForTwoProbCell(X,Y,I,C):-NI is I+1,scanVerticallyForTwoProbCell(X,Y,NI,C).
% scan square to check if there any cell contain the same two probability of the cell in index [X,Y]
and if there any cell remove this two probability from the other cells in this column
%[X,Y] is the index of the cell to compare with its probability ,[SX,EX,SY,EY] is the boundary of the
square
scanSquarelyForTwoProbCell(_,_,_,_,SY,EY,0):-SY>EY.
scanSquarelyForTwoProbCell(X,Y,SX,EX,SY,EY,C):-SX=<EX,SX=:=X,SY=:=Y,NSX is
SX+1,scanSquarelyForTwoProbCell(X,Y,NSX,EX,SY,EY,C).
```

```
scanSquarelyForTwoProbCell(X,Y,SX,EX,SY,EY,C):-
SX=<EX,cell(X,Y,R,_),cell(SX,SY,TR,_),isEqual(R,TR),TSX is EX-2,TSY is EY-2,
                                clearSquareTwoProb(R,X,Y,SX,SY,TSX,EX,TSY,EY,0,C).
scanSquarelyForTwoProbCell(X,Y,SX,EX,SY,EY,C):-SX=<EX,NSX is
SX+1,scanSquarelyForTwoProbCell(X,Y,NSX,EX,SY,EY,C).
scanSquarelyForTwoProbCell(X,Y,SX,EX,SY,EY,C):-SX>EX,NSX is EX-2,NSY is
SY+1,scanSquarelyForTwoProbCell(X,Y,NSX,EX,NSY,EY,C).
% clear the two of the cell probability from the probability in the column, row and square of the
selected cell(X,Y is the position of the selected cell
%[SX,EX,SY,EY] is the boundary of the square indexes that the cell in located in)
%! the cell should contain just two probability
clearTwoCellProb(X,Y,SX,EX,SY,EY,C):-
scanVerticallyForTwoProbCell(X,Y,1,CV),scanHorizontallyForTwoProbCell(X,Y,1,CH),scanSquarelyForT
woProbCell(X,Y,SX,EX,SY,EY,CS),C is CV+CH+CS.
% check if v is founded in the other cells in row prob and return false if founded and true if not [X,Y
index of the cell to ignore , I is an index]
checkIfValueIsUniqHorizontally(_,_,_,10).
checkIfValueIsUniqHorizontally(X,Y,V,I):-I=:=X,NI is I+1,checkIfValueIsUniqHorizontally(X,Y,V,NI).
checkIfValueIsUniqHorizontally(_,Y,V,I):-cell(I,Y,R,_),contain(V,R),1=:=2.
checkIfValueIsUniqHorizontally(X,Y,V,I):-cell(I,Y,R,_),not(contain(V,R)),NI is
I+1,checkIfValueIsUniqHorizontally(X,Y,V,NI).
% check if v is founded in the other cells in column prob and return false if founded and true if not
[X,Y index of the cell to ignore , I is an index]
checkIfValueIsUniqVertically(_,_,_,10).
checkIfValueIsUniqVertically(X,Y,V,I):-I=:=Y,NI is I+1,checkIfValueIsUniqVertically(X,Y,V,NI).
checkIfValueIsUniqVertically(X,_,V,I):-cell(X,I,R,_),contain(V,R),1=:=2.
checkIfValueIsUniqVertically(X,Y,V,I):-cell(X,I,R,_),not(contain(V,R)),NI is
I+1,checkIfValueIsUniqVertically(X,Y,V,NI).
% check if v is founded in the other cells in square prob and return false if founded and true if not
[X,Y index of the cell to ignore , I is an index,(SX,SY,EX,EY) is the boundary of the square]
checkIfValueIsUniqSquarely(_,_,_,_,_,SY,EY):-SY>EY.
checkIfValueIsUniqSquarely(X,Y,V,SX,EX,SY,EY):-SX=<EX,X=:=SX,Y=:=SY,NSX is
SX+1,checkIfValueIsUniqSquarely(X,Y,V,NSX,EX,SY,EY).
checkIfValueIsUniqSquarely(_,_,V,SX,EX,SY,_):-SX=<EX,cell(SX,SY,R,_),contain(V,R),1=:=2.
checkIfValueIsUniqSquarely(X,Y,V,SX,EX,SY,EY):-SX=<EX,cell(SX,SY,R,_),not(contain(V,R)),NSX is
SX+1,checkIfValueIsUniqSquarely(X,Y,V,NSX,EX,SY,EY).
checkIfValueIsUniqSquarely(X,Y,V,SX,EX,SY,EY):-SX>EX,NSY is SY+1,NSX is EX-
2,checkIfValueIsUniqSquarely(X,Y,V,NSX,EX,NSY,EY).
%scan to check if the any uniq prob in the cell and delete the other probabilities
scanForUniqCell(_,_,_,_,_,_,[],0).
scanForUniqCell(X,Y,SX,EX,SY,EY,[V|_],1):-
checkIfValueIsUniqSquarely(X,Y,V,SX,EX,SY,EY),retract(cell(X,Y,_,_)),assert(cell(X,Y,[V],0)).
scanForUniqCell(X,Y,_,_,_,_,[V|_],1):-
checkIfValueIsUniqVertically(X,Y,V,1),retract(cell(X,Y,_,_)),assert(cell(X,Y,[V],0)).
```

```
scanForUniqCell(X,Y,_,_,_,_,[V|_],1):-
checkIfValueIsUniqHorizontally(X,Y,V,1),retract(cell(X,Y,_,_)),assert(cell(X,Y,[V],0)).
scanForUniqCell(X,Y,SX,EX,SY,EY,[_|T],S):-scanForUniqCell(X,Y,SX,EX,SY,EY,T,S).
% solve the cells that contain uniq probability
solveUniqCellProb(X,Y,SX,EX,SY,EY,C):- cell(X,Y,R,_),scanForUniqCell(X,Y,SX,EX,SY,EY,R,C).
% scan square and solve cells that can be solved [SX,EX,SY,EY] is the boundary of the square
solveSquareInPuzzle(_,_,SY,EY,0,0):-SY>EY.
solveSquareInPuzzle(_,_,SY,EY,S,1):-SY>EY,S>0.
solveSquareInPuzzle(SX,EX,SY,EY,S,C):-SX=<EX,cell(SX,SY,_,1),NSX is
SX+1,solveSquareInPuzzle(NSX,EX,SY,EY,S,C).
solveSquareInPuzzle(SX,EX,SY,EY,_,C):-SX=<EX,cell(SX,SY,R,0),length(R,L),L=:=1,TSX is EX-2,TSY is EY-
2,
                clearOneProbCell(SX,SY,TSX,EX,TSY,EY,CO),NSX is
SX+1,solveSquareInPuzzle(NSX,EX,SY,EY,CO,C).
solveSquareInPuzzle(SX,EX,SY,EY,_,C):-SX=<EX,cell(SX,SY,R,0),length(R,L),L=:=2,TSX is EX-2,TSY is EY-
2,NSX is SX+1,
                clearTwoCellProb(SX,SY,TSX,EX,TSY,EY,CT),TTSX is EX-2,TTSY is EY-
2,solveUniqCellProb(SX,SY,TTSX,EX,TTSY,EY,CU),NS is
CT+CU,solveSquareInPuzzle(NSX,EX,SY,EY,NS,C).
solveSquareInPuzzle(SX,EX,SY,EY,_,C):-SX=<EX,NSX is SX+1,TTSX is EX-2,TTSY is EY-
2,solveUniqCellProb(SX,SY,TTSX,EX,TTSY,EY,SU),solveSquareInPuzzle(NSX,EX,SY,EY,SU,C).
solveSquareInPuzzle(SX,EX,SY,EY,S,C):-SX>EX,NSX is EX-2,NSY is
SY+1,solveSquareInPuzzle(NSX,EX,NSY,EY,S,C).
% check if the puzzle is solved or not
isSolved(_,10).
isSolved(X,Y):- X=<9,cell(X,Y,R,_),length(R,L),L=:=1,NX is X+1,isSolved(NX,Y).
isSolved(X,Y):- X>9,NX is 1, NY is Y+1,isSolved(NX,NY).
%scan tha puzzle to search for solve
scan(0,R):-     isSolved(1,1),R is 1,!.
scan(0,0).
scan(_,R):-
solveSquareInPuzzle(1,3,1,3,0,C1),solveSquareInPuzzle(4,6,1,3,0,C2),solveSquareInPuzzle(7,9,1,3,0,
C3),
        solveSquareInPuzzle(1,3,4,6,0,C4),solveSquareInPuzzle(4,6,4,6,0,C5),solveSquareInPuzzle
(7,9,4,6,0,C6),
        solveSquareInPuzzle(1,3,7,9,0,C7),solveSquareInPuzzle(4,6,7,9,0,C8),solveSquareInPuzzle
(7,9,7,9,0,C9),
        NC is (C1+C2+C3+C4+C5+C6+C7+C8+C9),scan(NC,R),!.
% solve the puzzle
solvePuzzle(R):-fillPuzzleProb(1,1),scan(1,R),!.
```