Resolução de um Problema de Decisão usando Programação em Lógica com Restrições: Outside Sudoku

João Pedro Viveiros Franco e Tomás Nuno Fernandes Novo

FEUP-Programação em Lógica, Turma 7, Grupo Outside Sudoku_3

FEUP – Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, 4200-465, Porto, Portugal

Resumo. O projeto concebido foi realizado no âmbito da unidade curricular Programação em Lógica e tem como objetivo a resolução de um problema de decisão usando programação em lógica através da implementação de restrições. O problema abordado designa-se por Outside Sudoku, sendo este um puzzle bastante semelhante ao tradicional Sudoku mas com a notória distinção de que os números que funcionam como dicas se encontram na parte exterior do tabuleiro. A resolução deste problema consiste na descoberta da solução do puzzle através das dicas dadas. O projeto foi desenvolvido em linguagem Prolog usufruindo do Sistema de Desenvolvimento SICStus Prolog. A resolução do problema é analisada pormenorizadamente neste artigo.

Palavras Chave: Outside Sudoku, Decisão, SICStus, Prolog, FEUP.

1 Introdução

No âmbito da unidade curricular Programação em Lógica do 3º ano do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto, foi desenvolvido este projeto que consiste na resolução do problema de decisão utilizando Programação em Lógica com restrições.

O problema de decisão selecionado intitula-se de Outside Sudoku e a sua resolução consiste na determinação da solução do puzzle de Sudoku com base nas dicas exteriores ao tabuleiro.

A estrutura do artigo foi elaborada com o objetivo de auxiliar na interpretação do projeto concebido, sendo que esta se encontra dividida em:

1. **Descrição do Problema –** explicação detalhada do problema.
2. **Abordagem –** descrição da modelação do problema como um PSR, subdividindo-se em:
    - **Variáveis de Decisão –** descrição das variáveis de decisão e seus domínios
    - **Restrições –** explicação das restrições rígidas e flexíveis do problema e respetiva implementação através do SICStus Prolog
    - **Função de Avaliação –** caraterização da avaliação da solução obtida e sua implementação
    - **Estratégia de Pesquisa –** descrição da estratégia de etiquetagem implementada no que diz respeita à ordenação de variáveis e valores
3. **Visualização da Solução** – explicação dos predicados que permitem a observação da solução em modo texto
4. **Resultados** – exemplos de aplicação em instâncias do problema com diferentes dimensões e análise dos resultados obtidos.
5. **Conclusões e Trabalho Futuro** – ilações retiradas da realização do projeto e dos resultados obtidos. Vantagens e limitações da proposta de resolução implementadas. Possibilidades de melhoria do trabalho desenvolvido
6. **Bibliografia –** livros, artigos e páginas Web usados no desenvolvimento do trabalho
7. **Referências –** fontes utilizadas
8. **Anexo** – código fonte, ficheiros de dados e resultados e outros elementos auxiliares à perceção da solução.

2 Descrição do Problema

Como já foi referido neste artigo, o problema escolhido designa-se por Outside Sudoku. A resolução deste puzzle consiste na resolução do puzzle com as regras normais do Sudoku original: as células têm de estar preenchidas com números de 1 a 9, não podendo haver números repetidos na mesma coluna, linha e quadrante. Porém, enquanto que num Sudoku tradicional existem células que contêm números que funcionam como dicas, o Outside Sudoku possui estas dicas fora do tabuleiro, encontrando-se adjacentes a uma linha ou uma coluna. A existência de números adjacentes à linha ou coluna do tabuleiro significa que os números indicados se encontram numa das primeiras três células da direção correspondida, sem ordem especificada.

3 Abordagem

Com o fim de resolver este problema utilizámos a linguagem Prolog e usufruímos do Sistema de Desenvolvimento SICSTus Prolog.

Intencionando a representação do tabuleiro inicial, optámos pela utilização de uma lista de nove listas de tamanho nove, visto que é o tamanho de um tabuleiro de Sudoku. Para representar os números exteriores ao tabuleiro que agem como possibilidades para as três células da direção correspondente, utilizámos uma lista de trinta e seis listas de tamanho um, dois, ou três, visto que podem surgir, respetivamente, um, dois ou três números adjacentes.

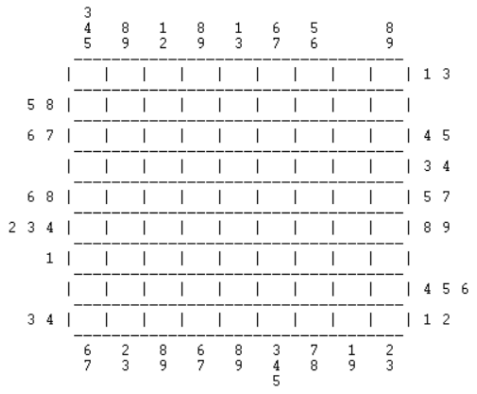

Fig. 1. Exemplo de um tabuleiro de Outside Sudoku.

Se considerarmos a lista de números fora do tabuleiro presente na **figura 1**, a nossa lista de trinta e seis listas a ter em consideração para a resolução do problema é a seguinte:

initialHints([[3,4,5],[8,9],[1,2],[8,9],[1,3],[6,7],[5,6],[],[8,9],
[3,1],[],[5,4],[4,3],[7,5],[9,8],[],[6,5,4],[2,1],
[3,2],[9,1],[8,7],[5,4,3],[9,8],[7,6],[9,8],[3,2],[7,6],
[3,4],[],[1],[2,3,4],[6,8],[],[6,7],[5,8],[]]).

3.1 Variáveis de decisão

A solução do problema consiste no preenchimento das células do tabuleiro considerando que cada linha, coluna e quadrante possui números de um a nove sem estar repetidos e tendo em atenção as dicas. Assim, a variável de decisão é o tabuleiro concebido. Exemplificando o valor desta variável, pode ser por exemplo:

winningBoard([[3, 6, 2, 9, 1, 7, 5, 4, 8], [5, 8, 1, 4, 3, 2, 6, 7, 9], [4, 9, 7, 8, 5, 6, 3, 2, 1], [9, 7, 5, 2, 6, 8, 1, 3 , 4], [8, 1, 6, 3, 4, 9, 2, 5, 7], [2, 4, 3, 5, 7, 1, 9, 8, 6] [1, 5 ,8 ,6 ,2 ,4 ,7 , 9, 3], [7, 2, 9, 1, 8, 3 ,4, 6, 5], [6, 3, 4, 7, 9, 5, 8, 1, 2]]).

Fig. 2. Exemplo da visualização da solução.

Relativamente ao domínio, situa-se entre o intervalo 1 e 9 visto que é o intervalo de valores pelos quais podem ser preenchidas as células.

3.2 Restrições

Em cada linha, coluna e quadrante os números terem de ser **todos distintos**, preenchendo cada linha, coluna e quadrante com números diferentes de 1 a 9. A presença de números que agem dicas no exterior do tabuleiro obriga a que uma das primeiras três células do tabuleiro nessa direção contenha o número correspondente á dica. Com o fim de implementar a solução tendo em conta estas restrições foi usado o predicado de restrição **all_different**. Para as restrições das dicas foi usado o predicado **fillHint(+Board, +[H|T], I)**. Relativamente às retrições do Sudoku, para garantir o preenchimento correto do tabuleiro, foram aplicadas foram usados os predicados fillBoardVertical, fillBoardHorizontal e fillBoardBox.

3.3 Função de avaliação

Com o fim de avaliar a solução obtida, foi construído o predicado **fillBoard(+Hints, -Board),** que gera um tabuleiro tendo em não só conta as restrições do Sudoku como também as dicas definidas. Este predicado devolve um possível tabuleiro para as restrições exigidas. O uso de **backtracking** devolve outros possíveis tabuleiros.

```
fillBoard(Hints, Board):-
  length(Board, 9),
  fillBoardHorizontal(Board),
  fillBoardVertical(Board),
  fillBoardBox(Board),
  fillHints(Board, Hints, 0),
  recursiveLabeling(Board).
```

Fig. 3. Predicado **fillBoard**.

3.4 Estratégia de pesquisa

Relativamente à estratégia de **labeling** implementada, como o nosso tabuleiro é uma lista de listas, foi necessário realizar o **labeling** de forma recursiva, de modo a ser aplicado a cada linha do tabuleiro.

```
recursiveLabeling([]).
recursiveLabeling([H|T]):-
  labeling([], H),
  recursiveLabeling(T).
```

Fig. 4. Predicado **recursiveLabeling**.

Esta estratégia foi utilizada no predicado **fillBoard(+Hints, -Board).**

4 Visualização da Solução

Intencionando visualizar a solução no Sistema de Desenvolvimento SICStus Prolog, implementámos o predicado **printBoard(+Hints, +Board)**, que começa por desenhar as dicas adjacentes à margem superioras do tabuleiro. De seguida, desenha as dicas à esquerda, o tabuleiro e as dicas à direita. Por fim, imprime as dicas por baixo do tabuleiro. Este predicado está preparado para a receção de dicas de diferentes tamanhos.

```
printBoard(Hints, Board):-
  I is 0,
  nl,nl,nl,
  displayTopHints(Hints),
  displayBar,
  printBoard(Board, Hints, I),
  displayBottomHints(Hints).
```

Fig. 5. Predicado **printBoard**.

A solução é obtida através da criação de uma lista de dicas de tamanho trinta e seis, no qual cada elemento é uma lista de tamanho variável entre 0 e 3 e do chamamento dos predicados **fillBoard(+Hints, -Board)** e **printBoard(+Hints, +Board)**, sendo **Hints** as dicas previamente mencionadas.

```
| ?- initialHints(H),fillBoard(H,B), printBoard(H,B).
```

Fig. 6. Exemplo dos comandos a utilizar para a visualização da solução.

5 Resultados

Pela idiossincrasia deste problema, decidimos usar como base um set predefinido de dicas para um tabuleiro específico, aumentando a quantidade de dicas com o fim de avaliar a *performance* do programa desenvolvido.

Intencionando retirar ilações com os resultados obtidos, fizemos variar a quantidade de dicas. Construímos então os gráficos presentes nas **figuras 7, 8** e **9** tendo em conta as dicas presentes na **tabela 1.** Através da interpretação dos dados contidos nos gráficos, podemos concluir que com o aumento do número de dicas, o tempo de execução do programa e o número de retrocessos diminuem enquanto que o número de restrições aumenta. O tempo de execução diminui com a quantidade de dicas visto que o aumento do número de restrições diminui o domínio dos cálculos a realizar pelo programa. O mesmo sucede com o número de retrocessos. Por fim, o número de restrições aumenta com à medida que a quantidade de dicas aumenta visto que há mais condições que o programa tem que verificar.

6 Conclusões e Trabalho Futuro

Em suma, a realização deste projeto foi benéfico para ambos os membros visto que aumentaram o seu conhecimento sobre programação em lógica, nomeadamente na aplicação de restrições e funcionamento do predicado *labelling*. Os objetivos do foram alcançados visto que foi implementada com sucesso a solução para o problema de decisão Outside Sudoku.

O nosso maior entrave na realização do projeto, que retardou o seu término, foi a aplicação das restrições das dicas. As restrições do puzzle foram facilmente implementadas devido à familiarização com o jogo Sudoku.

Apesar da boa implementação da solução que propusemos, esta podia ser melhorada com um método mais eficiente e otimizado.

Concluindo, o projeto foi concretizado com sucesso e a sua realização foi bastante positiva para o aumento dos nossos conhecimentos.

7 References

1. Logic Masters India, http://www.logicmastersindia.com/lmitests/dl.asp?attachmentid=740, last accessed December 2018

8 Anexos

Dados:

Table 1. Table captions should be placed above the tables.

Predicados	Total de dicas	Tempo(s)	Retrocessos	Restrições
initialHints3	52	0.08	1920	505
initialHints4	54	0.06	1584	513
initialHints5	59	0.03	1112	533
initialHints6	62	0.02	522	545
initialHints7	65	0.01	199	565

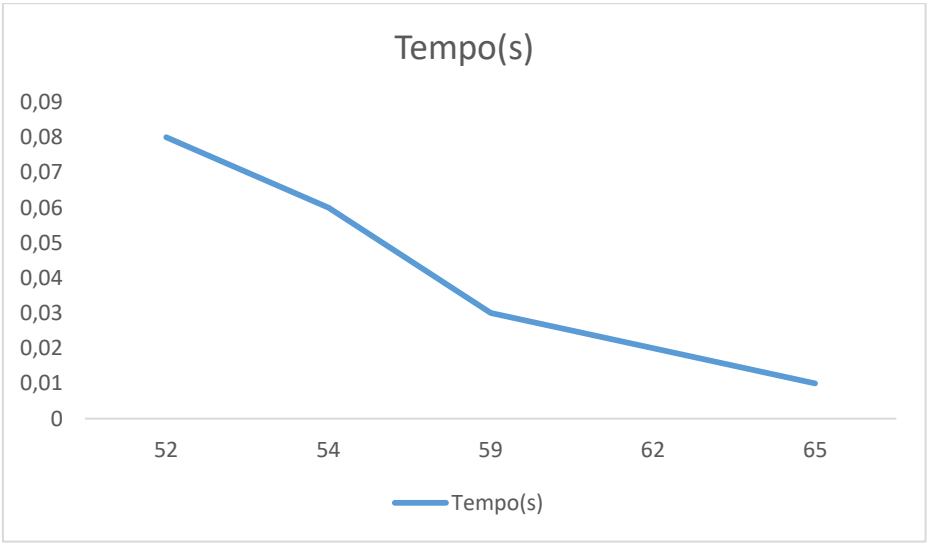

Fig. 7. Gráfico do tempo de processamento em função das dicas.

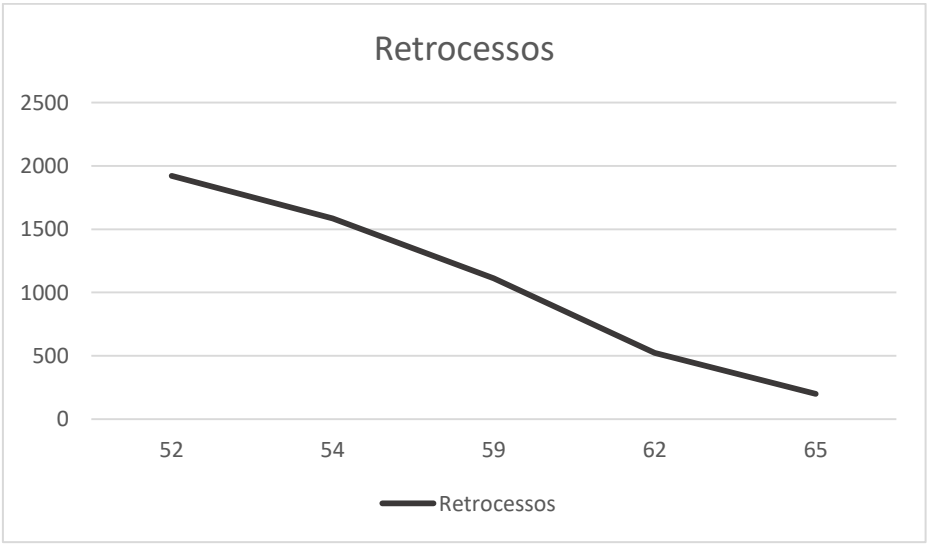

Fig. 8. Gráfico do número de retrocessos em função das dicas.

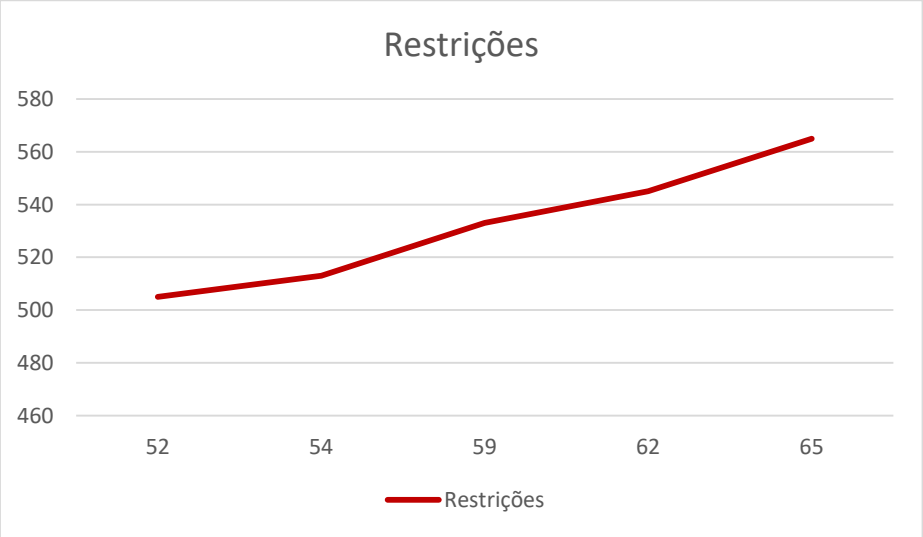

Fig. 9. Gráfico do número de restrições em função das dicas.

Código:

- **logic.pl :**

```
:- consult('utilities.pl'), use_module(library(lists)), use_module(library(clpfd)).

initialBoard( [[ [], [], [], [], [], [], [], [], [] ],
              [ [], [], [], [], [], [], [], [], [] ],
              [ [], [], [], [], [], [], [], [], [] ],
              [ [], [], [], [], [], [], [], [], [] ],
              [ [], [], [], [], [], [], [], [], [] ],
              [ [], [], [], [], [], [], [], [], [] ],
              [ [], [], [], [], [], [], [], [], [] ],
              [ [], [], [], [], [], [], [], [], [] ],
              [ [], [], [], [], [], [], [], [], [] ]]).

testingBoard([ [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],
               [ 4, 5, 6, 7, 8, 9, 1, 2, 3 ],
               [ 7, 8, 9, 1, 2, 3, 4, 5, 6 ],
               [ [], [], [], [], [], [], [], [], [] ],
               [ [], [], [], [], [], [], [], [], [] ],
```

```
                [ [], [], [], [], [], [], [], [], [] ],
                [ [], [], [], [], [], [], [], [], [] ],
                [ [], [], [], [], [], [], [], [], [] ],
               [ [], [], [], [], [], [], [], [], [] ]   ]).


winningBoard([ [ 5, 3, 4, 6, 7, 8, 9, 1, 2 ],
                 [ 6, 7, 2, 1, 9, 5, 3, 4, 8 ],
                 [ 1, 9, 8, 3, 4, 2, 5, 6, 7 ],
                 [ 8, 5, 9, 7, 6, 1, 4, 2, 3 ],
                 [ 4, 2, 6, 8, 5, 3, 7, 9, 1 ],
                 [ 7, 1, 3, 9, 2, 4, 8, 5, 6 ],
                 [ 9, 6, 1, 5, 3, 7, 2, 8, 4 ],
                 [ 2, 8, 7, 4, 1, 9, 6, 3, 5 ],
                 [ 3, 4, 5, 2, 8, 6, 1, 7, 9 ]
                 ]).

initialHints([
        [3,4,5],[8,9],[1,2],[8,9],[1,3],[6,7],[5,6],[],[8,9],
        [3,1],[],[5,4],[4,3],[7,5],[9,8],[],[6,5,4],[2,1],
        [3,2],[9,1],[8,7],[5,4,3],[9,8],[7,6],[9,8],[3,2],[7,6],
        [3,4],[],[1],[2,3,4],[6,8],[],[6,7],[5,8],[]
        ] ).

initialHints2([
        [1,4],[2,5],[3,6],[4,7],[5,8],[6,9],[],[],[],
        [9,6],[8,2],[7,5],[6,4],[3,2],[8],[6],[],[],
        [8,5],[7,4],[3,2],[8,2],[4,1],[5,3],[],[],[],
        [3,7],[5,6],[4,8],[1,3],[7,9],[2],[8],[],[]
        ] ).


initialSelection(X):-
        fillList(3, [], Y),
        fillList(3, Y, X).

getElement(Board, X, Y, Element) :-
        nth1(Y, Board, Line),
        element(X, Line, Element).


getBoardSize([H|T], Width, Height):-
        length([H|T], Height),
        length(H, Width).
```

```
reset_timer :- statistics(walltime,_).
print_time :-
        statistics(walltime,[_,T]),
        TS is ((T//10)*10)/1000,
        nl, write('Time: '), write(TS), write('s'), nl, nl.



fillBoard(Hints, Board):-
        length(Board, 9),
        fillBoardHorizontal(Board),
        fillBoardVertical(Board),
        fillBoardBox(Board),

        fillHints(Board, Hints, 0),

        reset_timer,
        recursiveLabeling(Board),
        print_time,
        fd_statistics.


fillBoardHorizontal([]).
fillBoardHorizontal([H|T]):-
        length(H, 9),
        domain(H, 1, 9),
        all_different(H),
        fillBoardHorizontal(T).

fillBoardVertical(Board):-
        fillBoardVertical(Board, 9).


fillBoardVertical(_, 0).
fillBoardVertical(Board, I):-

        fillBoardVertical(Board, I, [], List),
        domain(List, 1, 9),
        all_different(List),

        I1 is I-1,
        fillBoardVertical(Board, I1).
```

```
fillBoardVertical([], _, List, List).
fillBoardVertical([H|T], I, List, List2):-
        element(I,H,A),
        fillBoardVertical(T, I, [A|List], List2).

fillBoardBox(Board):-
        fillBoardBox(Board, 9).


fillBoardBox(_, 0).
fillBoardBox(Board, I):-
        I1 is I-1,
        X is mod(I1,3)*3,
        Y is div(I1,3)*3,

        fillBoardBox(Board, X, Y, 9, [], List),
        domain(List, 1, 9),
        all_different(List),

        fillBoardBox(Board, I1).

fillBoardBox(_, _, _, 0, List, List).
fillBoardBox(Board, X, Y, I, List, List2):-
        I1 is I-1,
        X1 is X+mod(I1,3)+1,
        Y1 is Y+div(I1,3),

        nth0(Y1, Board, Row),
        element(X1, Row, A),
        fillBoardBox(Board, X, Y, I1, [A|List], List2).


fillHints(_, [], 36).
fillHints(Board, [H|T], I):-
        getTriplet(Board, I, Triplet),
        nth0(0, Triplet, A),
        nth0(1, Triplet, C),
        nth0(2, Triplet, B),
        fillHint(Board, H, I, A, B, C),

        I1 is I+1,
        fillHints(Board, T, I1).
```

```
fillHint(_, [], _, _, _, _).
fillHint(Board, [H|T], I, A, B, C):-

        A #= H #\/ B #= H #\/ C #= H,

        fillHint(Board, T, I, A, B, C).


recursiveLabeling([]).
recursiveLabeling([H|T]):-
        labeling([], H),
        recursiveLabeling(T).


getTriplet(Board, I, Triplet):-
        A is mod(I,9),
        B is div(I,9),

        getTriplet(Board, A, B, Triplet).

getTriplet(Board, A, 0, Triplet):-
        X is A+1,

        getElement(Board, X, 1, P1),
        getElement(Board, X, 2, P2),
        getElement(Board, X, 3, P3),
        Triplet = [P1,P2,P3].

getTriplet(Board, A, 1, Triplet):-
        Y is A+1,

        getElement(Board, 9, Y, P1),
        getElement(Board, 8, Y, P2),
        getElement(Board, 7, Y, P3),
        Triplet = [P1,P2,P3].

getTriplet(Board, A, 2, Triplet):-
        X is 9-A,

        getElement(Board, X, 9, P1),
        getElement(Board, X, 8, P2),
        getElement(Board, X, 7, P3),
        Triplet = [P1,P2,P3].
```

```
getTriplet(Board, A, 3, Triplet):-
        Y is 9-A,

        getElement(Board, 1, Y, P1),
        getElement(Board, 2, Y, P2),
        getElement(Board, 3, Y, P3),
        Triplet = [P1,P2,P3].
```

- **display.pl**

```
:- consult('logic.pl'), consult('utilities.pl').

draw_piece([]):- write('  ').
draw_piece(H):- write(' '),write(H).


print_line_aux([], Hints, I):-
        write(' | '),
        I1 is I+9,
        nth0(I1, Hints, Hint),
        reverse(Hint, Hint2),
        displayRightHints(Hint2),
        displayBar.
print_line_aux([_|_], _, _):-
        write(' |').


print_line([], _, _).
print_line([H|T], Hints, I):-
        draw_piece(H),
        print_line_aux(T, Hints, I),
        print_line(T, Hints, I).

printBoard([], _, _).
printBoard([H|T], Hints, I) :-
        I2 is 35-I,
        nth0(I2, Hints, Hint3),
        fillHint(Hint3, Hint4),
        displayRightHints(Hint4),
        write('|'),
        print_line(H, Hints, I),
        I1 is I+1,
        printBoard(T, Hints, I1).
```

```
printBoard(Hints, Board):-
        I is 0,
        nl,nl,nl,
        displayTopHints(Hints),
        displayBar,
        printBoard(Board, Hints, I),!,
        displayBottomHints(Hints).

printColumnNumber(N):-
        N < 10,
        write('| |').

 printColumnNumber(N):-
        N >= 10,
        write(N),
        write('|').


clear_console :-
        clear_console(40), !.

clear_console(0).

clear_console(N) :-
        nl, N1 is N - 1, clear_console(N1).


displayBar:-
        nl,write('      ----------------------------------'),nl.

displayTopHints(Hints):-
        fillHints(Hints, Hints2),
        displayTopHints(Hints2, 0),
        !.

displayTopHints(_, 3):-!.
displayTopHints(Hints, 2):-
        write('       '),
        displayTopHints(Hints, 2, 0).
```

```
displayTopHints(Hints, I):-
        write('       '),
        displayTopHints(Hints, I, 0),
        nl,
        I1 is I+1,
        displayTopHints(Hints, I1).

displayTopHints(_, _, 9):-!.
displayTopHints(Hints, I, I2):-
        nth0(I2, Hints, Hint),
        nth0(I, Hint, Element),

        write(Element), write('   '),

        I3 is I2+1,
        displayTopHints(Hints, I, I3).


displayBottomHints(Hints):-
        fillHints(Hints, Hints2),
        displayBottomHints(Hints2, 0),
        !.

displayBottomHints(_, 3):-!.
displayBottomHints(Hints, I):-
        write('       '),
        displayBottomHints(Hints, I, 0),
        nl,

        I1 is I+1,
        displayBottomHints(Hints, I1).


displayBottomHints(_, _, 9):-!.
displayBottomHints(Hints, I, I2):-
        I4 is 26-I2,
        nth0(I4, Hints, Hint),
        I5 is 2-I,
        nth0(I5, Hint, Element),

        write(Element), write('   '),

        I3 is I2+1,
        displayBottomHints(Hints, I, I3).
```

```
fillHints([], []):-!.
fillHints([H|T], [H2|T2]):-
        fillHint(H, H2),

        fillHints(T, T2).


fillHint([A,B,C], [A,B,C]):-!.
fillHint(List, Return):-
        append([' '], List, List2),
        fillHint(List2, Return).


displayleHints([]).
displayLeftHints([H|T]):-
        write(H),write(' '),
        displayRightHints(T).

displayRightHints([]).
displayRightHints([H|T]):-
        write(H),write(' '),
        displayRightHints(T).
```

- **utilities.pl**

```
replace([_|T], 0, New, [New|T]).
replace([H|T], Index, New, [H|R]) :-
        I1 is Index - 1,
        replace(T, I1, New, R), !.

replaceByCoords(List1, X, Y, New, List2) :-
   nth0(Y, List1, Line),
   replace(Line, X, New, L),
   replace(List1, Y, L, List2), !.

addHead([H|T],A,Zs) :-
        append([A],[H|T],Zs).

addTail([H|T],A,Zs) :-
        append([H|T],[A],Zs).
```

```
maximum(A, B, C):-
        A > B,
        C = A.

maximum(A, B, C):-
        A =< B,
        C = B.

printList([]).
printList([H|T]):-
        write(H),nl,
        printList(T).


createLine([], 0).
createLine([H|T], I):-
        I > 0,
        append([], [], H),
        I1 is I-1,
        createLine(T, I1), !.

fillList(I, Row, Y):-
        fillList([], I, Row, Y).

fillList(X, 0, _, X):-!.
fillList(X, I, Row, Y):-
        append(X, [Row], NewX),
        I1 is I-1,
        fillList(NewX, I1, Row, Y).



cutLeftSide(List, X, List2):-
        length(List, L),
        L2 is L-X,
        length(List2, L2),
        length(List3, X),

        append(List3, List2, List).
```

- **outside.pl**

```
:-consult('display.pl'), consult('logic.pl'), consult('utilities.pl').
```