Bachelor of Information Technology (Hons)
Assignment Cover Sheet

Course Code: EC3272 Course Title: Artificial Intelligence

Assignment Title: Prolog Due Date: 7th, September

Date submitted: 7th, September Lecturer Name: Chiranjibi Pandey

To be completed if this is an individual assignment

I declare that this assignment is my individual work. I have not worked collaboratively nor have I copied from another student's work or from any other source except where due acknowledgement is made explicitly in the text, nor has any part been written for me by another person.

Student Name: _______________ Student ID:

Student Signature: ________________

To be completed if this is a group assignment

We declare that this is a group assignment and that no part of this submission has been copied from any other student's work or from any other source except where due acknowledgement is made explicitly in the text, nor has any part been written for us by another person.

Student ID	Student Name	Signature
00021595	Sneha Tamang	
00021592	Prince Thapa	
00021547	Akriti Gautam	
00021534	Isha Khadka	Isha Khadka

Lecturer's Comment: - ____.

Total Marks: ______________________ Lecture's Signature: -______________________

Feedback to Student:

I/We acknowledged receiving feedback from the lecturer on this assignment.

Student's Signature: ____

Extension certification:

This assignment has been given an extension and is now due on .

Lecturer's Signature: ____.

Table of Contents

Introduction

Sudoku is one of the most popular and widely recognized puzzle games in the world. It is a number placement game that requires logical thinking, patience, and careful observation. The standard version of Sudoku is played on a 9x9 grid, which is further divided into nine smaller 3x3 sub grids. The main challenge for the player is to fill every empty cell of the grid with a number between 1 and 9, while strictly following the rule that no number should be repeated in any row, column, or sub grid. Unlike many other games, Sudoku does not rely on guessing or mathematical calculations. instead, it is purely based on logic and reasoning.

The purpose of this project is to design and implement a simple Sudoku game that demonstrates the interaction between a logical problem and computational problem-solving techniques. The project builds an environment where the player can interact with the puzzle using basic commands to view the grid, fill in numbers, request hints, or check their progress. This interactive design makes the game not only entertaining but also educational, as it sharpens the player's problem-solving ability.

In addition, the project explores different levels of difficulty by varying the number of pre-filled clues in the grid. Easier puzzles provide more initial numbers, giving the player straightforward logical paths, while harder puzzles provide fewer clues, demanding deeper reasoning and strategy. Optional features such as hints, limited attempts, or timers can also be integrated to increase the challenge and engagement level. This project does not just focus on creating a game for entertainment but also emphasizes the importance of logical reasoning and the application of programming concepts in designing interactive applications. The Sudoku game acts as an excellent example of how rules and constraints can be programmed to ensure that the game operates fairly and provides a meaningful challenge to the player.

Objectives of Project

- To design and implement a complete 9x9 Sudoku grid, divided into nine 3x3 sub grids, with a balanced distribution of pre-filled numbers that serve as clues for the player.
- To create interactive commands such as look (Grid) for displaying the current grid, fill (Row, Column, Number) for inserting values into empty cells, hint for providing suggestions, and check for verifying the validity of the current configuration.
- To strictly enforce Sudoku rules, ensuring that no duplicate numbers appear in any row, column, or sub grid, and to provide feedback when the player attempts an invalid move.
- To develop multiple levels of difficulty by controlling the number of pre-filled cells, allowing the game to be accessible for beginners and challenging for experienced players.
- To enhance the user experience with additional features such as hints, warnings for invalid attempts, or a timer system that encourages strategic and timely thinking.
- To clearly define winning and losing conditions, where the player wins upon correctly completing the grid and loses or receives penalties if they consistently break the rules.
- To use this project as a practical demonstration of logical problem-solving in programming, showing how a rule-based puzzle can be implemented in an interactive and educational format.

Prolog Code

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

:- use_module(library(random)).

:- dynamic current_grid/1.

:- dynamic hints_left/1.

%Puzzle Definition

puzzle([

   [5,3,_,_,7,_,_,_,_],

   [6,_,_,1,9,5,_,_,_],

   [_,9,8,_,_,_,_,6,_],

   [8,_,_,_,6,_,_,_,3],

   [4,_,_,8,_,3,_,_,1],

   [7,_,_,_,2,_,_,_,6],

   [_,6,_,_,_,_,2,8,_],

   [_,_,_,4,1,9,_,_,5],

   [_,_,_,_,8,_,_,7,9]

]).

% Difficulty Levels

difficulty(1, 50, 9). % Easy: filled, hints

difficulty(2, 40, 6). % Normal

difficulty(3, 30, 4). % Hard

difficulty(4, 20, 1). % Evil
```

```
% -------- Start Game --------

start :-

    write('Choose difficulty:'), nl,

    write('1 - Easy (50 filled, 9 hints)'), nl,

    write('2 - Normal (40 filled, 6 hints)'), nl,

    write('3 - Hard (30 filled, 4 hints)'), nl,

    write('4 - Evil (20 filled, 1 hint)'), nl,

    write('Enter your choice: '),

    read(Diff),

    difficulty(Diff, Filled, Hints),

    retractall(hints_left(_)),

    asserta(hints_left(Hints)),

retractall(current_difficulty(_)),

    asserta(current_difficulty(Diff)),

    generate_solved_grid(Solved),

    create_puzzle(Solved, Filled, Puzzle),

    retractall(current_grid(_)),

    asserta(current_grid(Puzzle)),

    retractall(initial_grid(_)),

    asserta(initial_grid(Puzzle)),

    look.
```

```
% -------- Display Grid --------

look :-

   current_grid(Grid),

   write('    1 2 3 4 5 6 7 8 9'), nl,

   write('  +------------------+'), nl,

   print_rows_with_numbers(Grid, 1), !.



% -------- Show Initial Puzzle --------

show_initial :-

   initial_grid(Grid),

   write('Initial Puzzle:'), nl,

   write('    1 2 3 4 5 6 7 8 9'), nl,

   write('  +------------------+'), nl,

   print_rows_with_numbers(Grid, 1), !.

print_rows_with_numbers([], _).

print_rows_with_numbers([Row|Rest], RowNum) :-

   write(RowNum), write(' |'),

   maplist(print_cell, Row),

   write('|'), nl,

   NextRowNum is RowNum + 1,

   print_rows_with_numbers(Rest, NextRowNum).

print_cell(Cell) :-

   ( var(Cell) -> write(' _') ; format(' ~w', [Cell]) ).
```

```
% -------- Fill a Cell --------

fill(Row, Col, Num) :-

    current_grid(Grid),

    initial_grid(InitialGrid),

    nth1(Row, InitialGrid, IR),

    nth1(Col, IR, InitialCell),

    var(InitialCell),        % only fill cells that were initially blank

    nth1(Row, Grid, R),

    nth1(Col, R, Cell),

    var(Cell),              % only fill empty cells

    Cell = Num,

    retractall(current_grid(_)),

    asserta(current_grid(Grid)),

    look,

    % Check if puzzle is solved

    append(Grid, Vars),

    \+ (member(Var, Vars), var(Var)),  % all cells filled

    Vars ins 1..9,

    valid_rows(Grid),

    valid_columns(Grid),

    valid_blocks(Grid),

    current_difficulty(Diff),

    difficulty_score(Diff, Base),

    hints_left(H),
    UsedHints = 9 - H,
    Deduct = UsedHints * 10,
    Score = Base - Deduct,
    format('Puzzle completed! Score: ~w~n', [Score]).

% -------- Check Validity --------
check :-
    current_grid(Grid),
    check_rows(Grid, RowErrors),
    check_columns(Grid, ColErrors),
    check_blocks(Grid, BlockErrors),
( RowErrors = [], ColErrors = [], BlockErrors = [] ->
        write('Grid is valid!'), nl
    ;   write('Grid is invalid:'), nl,
        report_errors('Row', RowErrors),
        report_errors('Column', ColErrors),
        report_errors('Block', BlockErrors)
    ).

check_rows(Grid, Errors) :-
    check_rows(Grid, 1, Errors).

check_rows([], _, []).
check_rows([Row|Rest], RowNum, Errors) :-
```

```
  ( all_distinct(Row) ->
    check_rows(Rest, RowNum + 1, Errors)
  ; find_duplicates(Row, Dups),
    check_rows(Rest, RowNum + 1, RestErrors),
    Errors = [row(RowNum, Dups)|RestErrors]
  ).

check_columns(Grid, Errors) :-
  transpose(Grid, Columns),
  check_columns(Columns, 1, Errors).

check_columns([], _, []).
check_columns([Col|Rest], ColNum, Errors) :-
  ( all_distinct(Col) ->
    check_columns(Rest, ColNum + 1, Errors)
  ; find_duplicates(Col, Dups),
    check_columns(Rest, ColNum + 1, RestErrors),
    Errors = [column(ColNum, Dups)|RestErrors]
  ).
check_blocks(Grid, Errors) :-
  findall(block(BlockNum, Dups), (between(1,9,BlockNum), get_block(Grid, BlockNum, Block), \+ all_distinct(Block), find_duplicates(Block, Dups)), Errors).

get_block(Grid, BlockNum, Block) :-
```

```
  RowStart is ((BlockNum - 1) // 3) * 3 + 1,

  ColStart is ((BlockNum - 1) mod 3) * 3 + 1,

  findall(Value, (between(0,2,RI), between(0,2,CI), Row is RowStart + RI, Col is ColStart + CI,
nth1(Row, Grid, R), nth1(Col, R, Value)), Block).


report_errors(_, []).

report_errors(Type, [Error|Rest]) :-

  ( Error =.. [TypeName, Num, Dups] ->

    format('~w ~w has duplicates: ~w~n', [TypeName, Num, Dups])

  ;  format('~w~n', [Error])

  ),

  report_errors(Type, Rest).


% -------- Find Duplicates in List --------

find_duplicates(List, Dups) :-

  exclude(var, List, Values),

  msort(Values, Sorted),

  findall(X, (append(_, [X,X|_], Sorted)), DupsUnsorted),

  sort(DupsUnsorted, Dups).


valid_rows([]).

valid_rows([Row|Rest]) :-

  all_distinct(Row),

  valid_rows(Rest).
```

```
valid_columns(Grid) :-
   transpose(Grid, Columns),
   valid_rows(Columns).

valid_blocks([]).
valid_blocks([A,B,C|Rest]) :-
blocks(A,B,C),
   valid_blocks(Rest).

blocks([], [], []) .
blocks([A,B,C|R1], [D,E,F|R2], [G,H,I|R3]) :-
   all_distinct([A,B,C,D,E,F,G,H,I]),
   blocks(R1,R2,R3).

% -------- Transpose --------
transpose([], []).
transpose([[]|_], []) :- !.
transpose(Matrix, [Row|Rows]) :-
   maplist(head, Matrix, Row),
   maplist(tail, Matrix, Rest),
   transpose(Rest, Rows).

head([H|_], H).
```

```
tail([_|T], T).

% -------- Hint System --------
hint :-
   hints_left(H),
   H > 0,
   NewH is H - 1,
   retractall(hints_left(_)),
   asserta(hints_left(NewH)),
   current_grid(Grid),
   % find first empty cell
   nth1(Row, Grid, R),
   nth1(Col, R, Cell),
   var(Cell),
   append(Grid, Vars), Vars ins 1..9,
   valid_rows(Grid),
   valid_columns(Grid),
valid_blocks(Grid),
   % assign a valid number to this cell
   label([Cell]),
   format('row=~w, col=~w, num=~w~n', [Row, Col, Cell]),
   format('Hints remaining: ~w~n', [NewH]),
   Cell = _,   % undo assignment
   !.
```

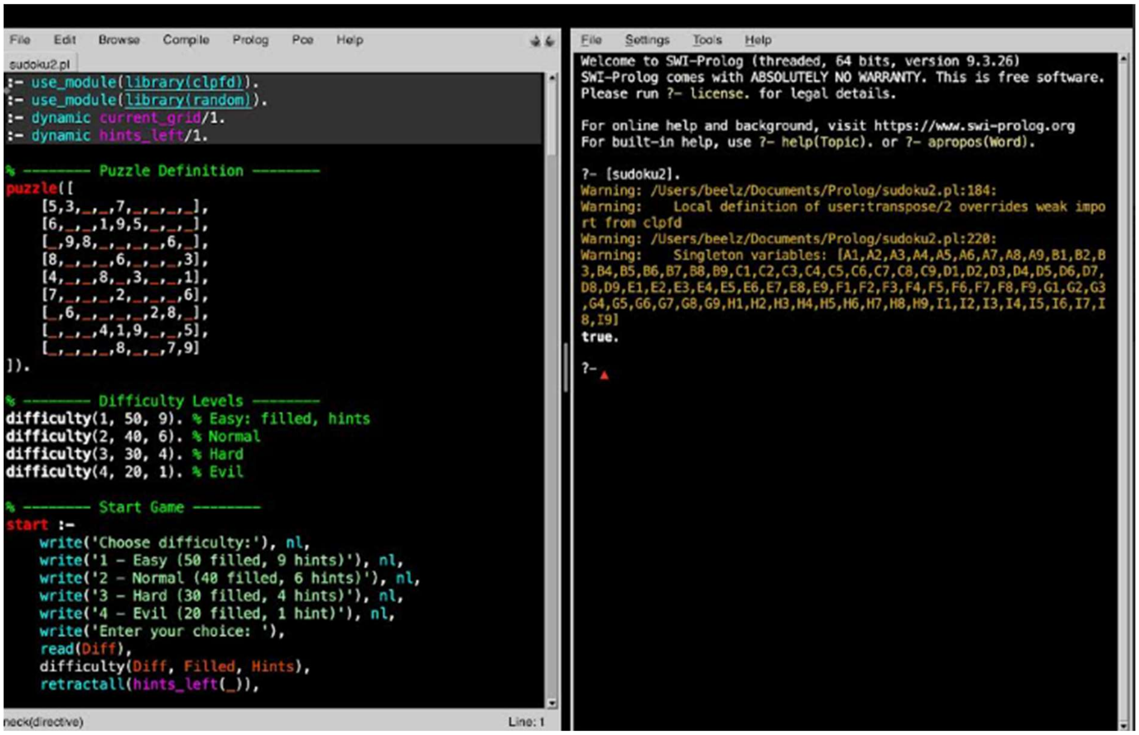
```
hint :-
    write('No more hints available!'), nl.


% -------- Generate Solved Grid --------
generate_solved_grid(Grid) :-
    Grid = [
        [A1,A2,A3,A4,A5,A6,A7,A8,A9],
        [B1,B2,B3,B4,B5,B6,B7,B8,B9],
        [C1,C2,C3,C4,C5,C6,C7,C8,C9],
        [D1,D2,D3,D4,D5,D6,D7,D8,D9],
        [E1,E2,E3,E4,E5,E6,E7,E8,E9],
        [F1,F2,F3,F4,F5,F6,F7,F8,F9],
        [G1,G2,G3,G4,G5,G6,G7,G8,G9],
        [H1,H2,H3,H4,H5,H6,H7,H8,H9],
        [I1,I2,I3,I4,I5,I6,I7,I8,I9]
    ],
    append(Grid, Vars), Vars ins 1..9,
    valid_rows(Grid),
    valid_columns(Grid),
    valid_blocks(Grid),
    random_permutation(Vars, ShuffledVars),
    label(ShuffledVars).
```

```
% -------- Create Puzzle --------
create_puzzle(Solved, Filled, Puzzle) :-
    length(Solved, 9), % Assume 9x9
    Total = 81,
    ToRemove is Total - Filled,
numlist(1, Total, Indices),
    random_select_indices(Indices, ToRemove, ToRemoveIndices),
    remove_by_indices(Solved, ToRemoveIndices, Puzzle).


random_select_indices(_, 0, []) :- !.
random_select_indices(List, N, [Index|Rest]) :-
    random_member(Index, List),
    select(Index, List, NewList),
    N1 is N - 1,
    random_select_indices(NewList, N1, Rest).


% -------- Remove by Indices --------
remove_by_indices(Grid, [], Grid).
remove_by_indices(Grid, [Index|Rest], NewGrid) :-
    replace_by_index(Grid, Index, TempGrid),
    remove_by_indices(TempGrid, Rest, NewGrid).


% -------- Replace by Index --------
replace_by_index(Grid, Index, NewGrid) :-
```

```
   Row is (Index - 1) // 9 + 1,
   Col is (Index - 1) mod 9 + 1,
   nth1(Row, Grid, R),
   nth1(Col, R, _),
   replace_in_row(R, Col, NewR),
   replace_in_grid(Grid, Row, NewR, NewGrid).


replace_in_row([_|Rest], 1, [_|Rest]).
replace_in_row([X|Rest], N, [X|NewRest]) :-
   N > 1,
   N1 is N - 1,
   replace_in_row(Rest, N1, NewRest).


replace_in_grid([_|Rows], 1, NewRow, [NewRow|Rows]).
replace_in_grid([Row|Rows], N, NewRow, [Row|NewRows]) :-
   N > 1,
N1 is N - 1,
   replace_in_grid(Rows, N1, NewRow, NewRows).


% -------- Check if Won --------
win :-
   current_grid(Grid),
   append(Grid, Vars),
   \+ (member(Var, Vars), var(Var)),  % all cells filled
```

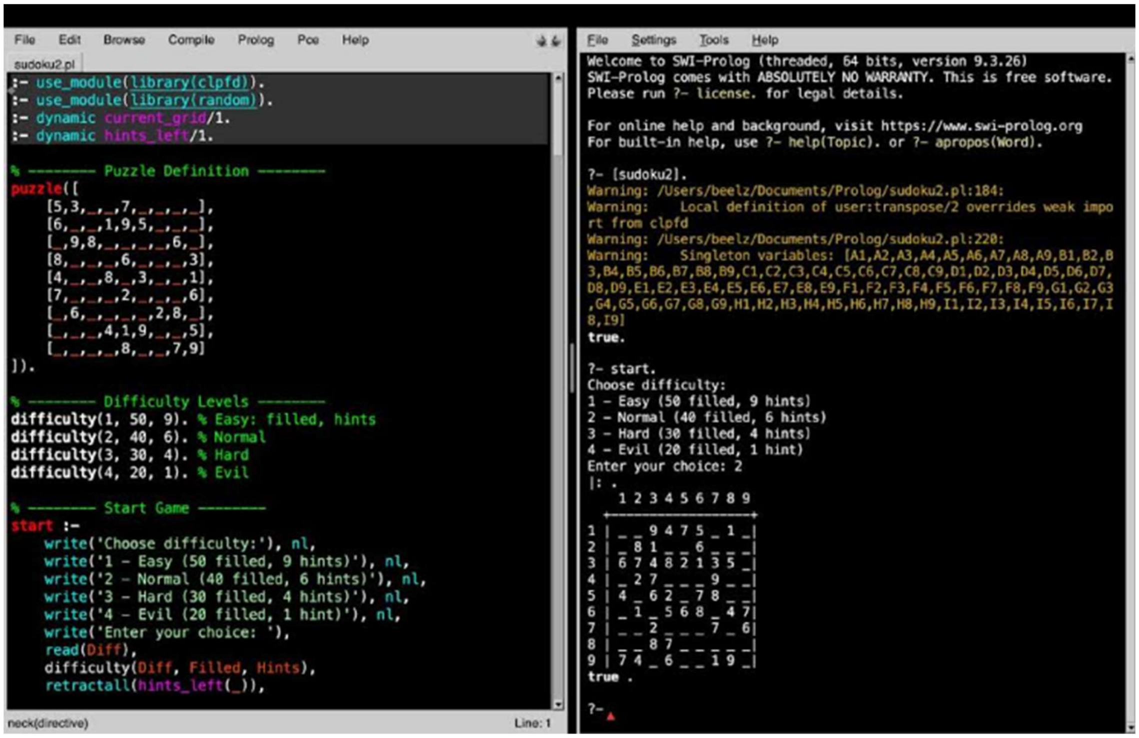
```
    Vars ins 1..9,
    valid_rows(Grid),
    valid_columns(Grid),
    valid_blocks(Grid),
    current_difficulty(Diff),
    difficulty_score(Diff, Base),
    hints_left(H),
    UsedHints = 9 - H,
    Deduct = UsedHints * 10,
    Score = Base - Deduct,
    format('Congratulations! You solved the puzzle. Score: ~w~n', [Score]).

difficulty_score(1, 100).
difficulty_score(2, 200).
difficulty_score(3, 300).
difficulty_score(4, 500).

%  Solve Sudoku
solve :-
    current_grid(Grid),
    append(Grid, Vars), Vars ins 1..9,
    valid_rows(Grid),
    valid_columns(Grid),
    valid_blocks(Grid),
    label(Vars),

    retractall(current_grid(_)),

    asserta(current_grid(Grid)),

    look.
```

Output



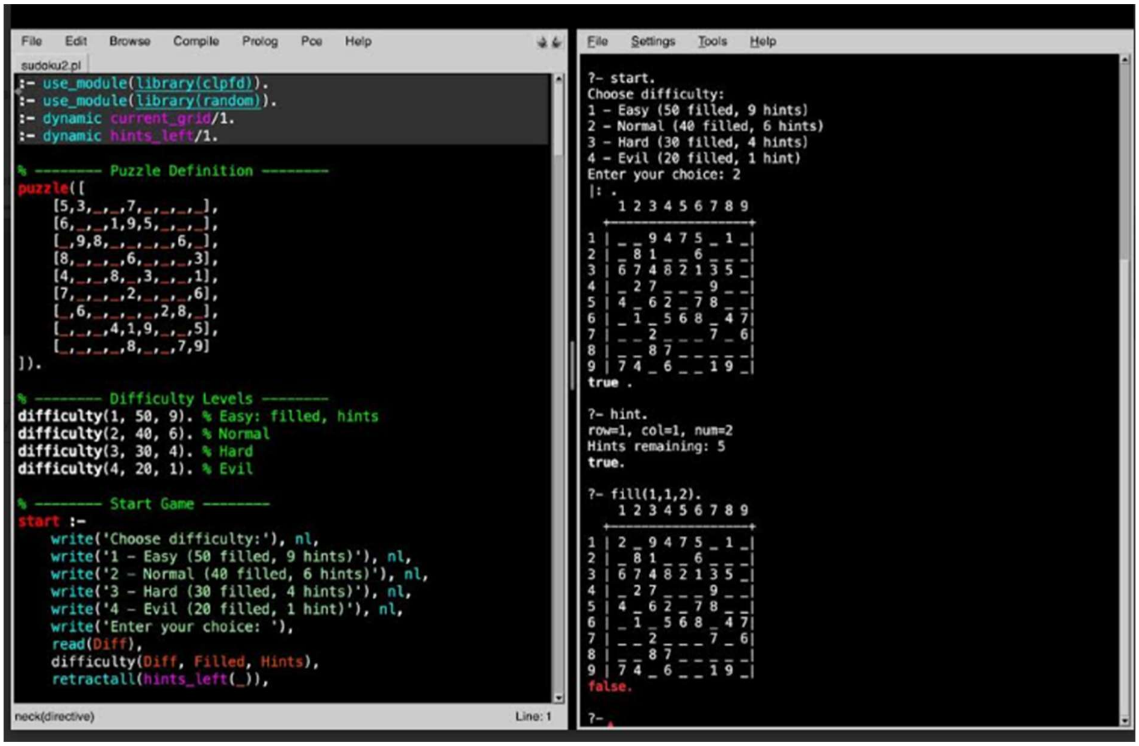
```
:- use_module(library(clpfd)).
:- use_module(library(random)).
:- dynamic current_grid/1.
:- dynamic hints_left/1.

% -------- Puzzle Definition --------
puzzle([
    [5,3,_,_,7,_,_,_,_],
    [6,_,_,1,9,5,_,_,_],
    [_,9,8,_,_,_,_,6,_],
    [8,_,_,_,6,_,_,_,3],
    [4,_,_,8,_,3,_,_,1],
    [7,_,_,_,2,_,_,_,6],
    [_,6,_,_,_,_,2,8,_],
    [_,_,_,4,1,9,_,_,5],
    [_,_,_,_,8,_,_,7,9]
]).

% -------- Difficulty Levels --------
difficulty(1, 50, 9). % Easy: filled, hints
difficulty(2, 40, 6). % Normal
difficulty(3, 30, 4). % Hard
difficulty(4, 20, 1). % Evil

% -------- Start Game --------
start :-
    write('Choose difficulty:'), nl,
    write('1 - Easy (50 filled, 9 hints)'), nl,
    write('2 - Normal (40 filled, 6 hints)'), nl,
    write('3 - Hard (30 filled, 4 hints)'), nl,
    write('4 - Evil (20 filled, 1 hint)'), nl,
    write('Enter your choice: '),
    read(Diff),
    difficulty(Diff, Filled, Hints),
    retractall(hints_left(_)),
```

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.26)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [sudoku2].
Warning: /Users/beelz/Documents/Prolog/sudoku2.pl:184:
Warning:    Local definition of user:transpose/2 overrides weak impo
rt from clpfd
Warning: /Users/beelz/Documents/Prolog/sudoku2.pl:220:
Warning:    Singleton variables: [A1,A2,A3,A4,A5,A6,A7,A8,A9,B1,B2,B
3,B4,B5,B6,B7,B8,B9,C1,C2,C3,C4,C5,C6,C7,C8,C9,D1,D2,D3,D4,D5,D6,D7,
D8,D9,E1,E2,E3,E4,E5,E6,E7,E8,E9,F1,F2,F3,F4,F5,F6,F7,F8,F9,G1,G2,G3
,G4,G5,G6,G7,G8,G9,H1,H2,H3,H4,H5,H6,H7,H8,H9,I1,I2,I3,I4,I5,I6,I7,I
8,I9]
true.

?-
```



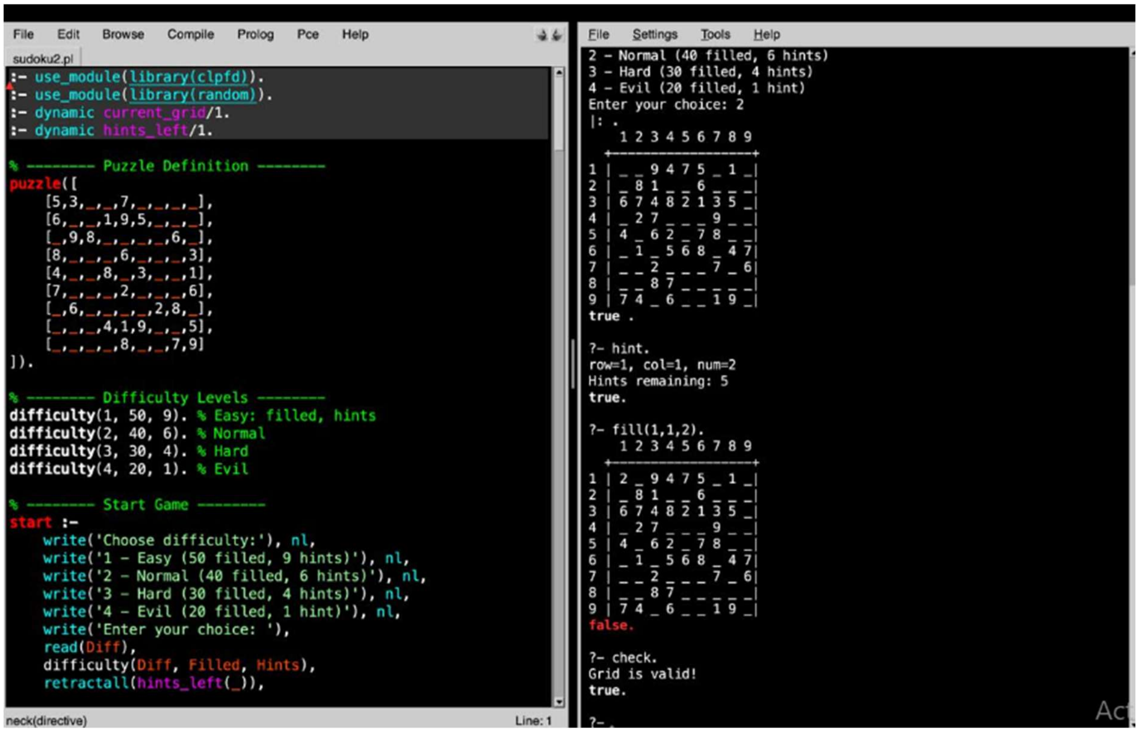
```
:- use_module(library(clpfd)).
:- use_module(library(random)).
:- dynamic current_grid/1.
:- dynamic hints_left/1.

% -------- Puzzle Definition --------
puzzle([
    [5,3,_,_,7,_,_,_,_],
    [6,_,_,1,9,5,_,_,_],
    [_,9,8,_,_,_,_,6,_],
    [8,_,_,_,6,_,_,_,3],
    [4,_,_,8,_,3,_,_,1],
    [7,_,_,_,2,_,_,_,6],
    [_,6,_,_,_,_,2,8,_],
    [_,_,_,4,1,9,_,_,5],
    [_,_,_,_,8,_,_,7,9]
]).

% -------- Difficulty Levels --------
difficulty(1, 50, 9). % Easy: filled, hints
difficulty(2, 40, 6). % Normal
difficulty(3, 30, 4). % Hard
difficulty(4, 20, 1). % Evil

% -------- Start Game --------
start :-
    write('Choose difficulty:'), nl,
    write('1 - Easy (50 filled, 9 hints)'), nl,
    write('2 - Normal (40 filled, 6 hints)'), nl,
    write('3 - Hard (30 filled, 4 hints)'), nl,
    write('4 - Evil (20 filled, 1 hint)'), nl,
    write('Enter your choice: '),
    read(Diff),
    difficulty(Diff, Filled, Hints),
    retractall(hints_left(_)),
```

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.26)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [sudoku2].
Warning: /Users/beelz/Documents/Prolog/sudoku2.pl:184:
Warning:    Local definition of user:transpose/2 overrides weak impo
rt from clpfd
Warning: /Users/beelz/Documents/Prolog/sudoku2.pl:220:
Warning:    Singleton variables: [A1,A2,A3,A4,A5,A6,A7,A8,A9,B1,B2,B
3,B4,B5,B6,B7,B8,B9,C1,C2,C3,C4,C5,C6,C7,C8,C9,D1,D2,D3,D4,D5,D6,D7,
D8,D9,E1,E2,E3,E4,E5,E6,E7,E8,E9,F1,F2,F3,F4,F5,F6,F7,F8,F9,G1,G2,G3
,G4,G5,G6,G7,G8,G9,H1,H2,H3,H4,H5,H6,H7,H8,H9,I1,I2,I3,I4,I5,I6,I7,I
8,I9]
true.

?- start.
Choose difficulty:
1 - Easy (50 filled, 9 hints)
2 - Normal (40 filled, 6 hints)
3 - Hard (30 filled, 4 hints)
4 - Evil (20 filled, 1 hint)
Enter your choice: 2
|: .
    1 2 3 4 5 6 7 8 9
  +-------------------+
1 | _ _ 9 4 7 5 _ 1 _|
2 | _ 8 1 _ _ 6 _ _ _|
3 | 6 7 4 8 2 1 3 5 _|
4 | _ 2 7 _ _ _ 9 _ _|
5 | 4 _ 6 2 _ 7 8 _ _|
6 | _ 1 _ 5 6 8 _ 4 7|
7 | _ _ 2 _ _ _ 7 _ 6|
8 | _ _ 8 7 _ _ _ _ _|
9 | 7 4 _ 6 _ _ 1 9 _|
true .

?-
```



```prolog
:- use_module(library(clpfd)).
:- use_module(library(random)).
:- dynamic current_grid/1.
:- dynamic hints_left/1.

% -------- Puzzle Definition --------
puzzle([
    [5,3,_,_,7,_,_,_,_],
    [6,_,_,1,9,5,_,_,_],
    [_,9,8,_,_,_,_,6,_],
    [8,_,_,_,6,_,_,_,3],
    [4,_,_,8,_,3,_,_,1],
    [7,_,_,_,2,_,_,_,6],
    [_,6,_,_,_,_,2,8,_],
    [_,_,_,4,1,9,_,_,5],
    [_,_,_,_,8,_,_,7,9]
]).

% -------- Difficulty Levels --------
difficulty(1, 50, 9). % Easy: filled, hints
difficulty(2, 40, 6). % Normal
difficulty(3, 30, 4). % Hard
difficulty(4, 20, 1). % Evil

% -------- Start Game --------
start :-
    write('Choose difficulty:'), nl,
    write('1 - Easy (50 filled, 9 hints)'), nl,
    write('2 - Normal (40 filled, 6 hints)'), nl,
    write('3 - Hard (30 filled, 4 hints)'), nl,
    write('4 - Evil (20 filled, 1 hint)'), nl,
    write('Enter your choice: '),
    read(Diff),
    difficulty(Diff, Filled, Hints),
    retractall(hints_left(_)),
```

```
?- start.
Choose difficulty:
1 - Easy (50 filled, 9 hints)
2 - Normal (40 filled, 6 hints)
3 - Hard (30 filled, 4 hints)
4 - Evil (20 filled, 1 hint)
Enter your choice: 2
|: .
    1 2 3 4 5 6 7 8 9
  +-------------------+
1 | _ _ 9 4 7 5 _ 1 _|
2 | _ 8 1 _ _ 6 _ _ _|
3 | 6 7 4 8 2 1 3 5 _|
4 | _ 2 7 _ _ _ 9 _ _|
5 | 4 _ 6 2 _ 7 8 _ _|
6 | _ 1 _ 5 6 8 _ 4 7|
7 | _ _ 2 _ _ _ 7 _ 6|
8 | _ _ 8 7 _ _ _ _ _|
9 | 7 4 _ 6 _ _ 1 9 _|
true .

?- hint.
row=1, col=1, num=2
Hints remaining: 5
true.

?- fill(1,1,2).
    1 2 3 4 5 6 7 8 9
  +-------------------+
1 | 2 _ 9 4 7 5 _ 1 _|
2 | _ 8 1 _ _ 6 _ _ _|
3 | 6 7 4 8 2 1 3 5 _|
4 | _ 2 7 _ _ _ 9 _ _|
5 | 4 _ 6 2 _ 7 8 _ _|
6 | _ 1 _ 5 6 8 _ 4 7|
7 | _ _ 2 _ _ _ 7 _ 6|
8 | _ _ 8 7 _ _ _ _ _|
9 | 7 4 _ 6 _ _ 1 9 _|
false.

?-
```



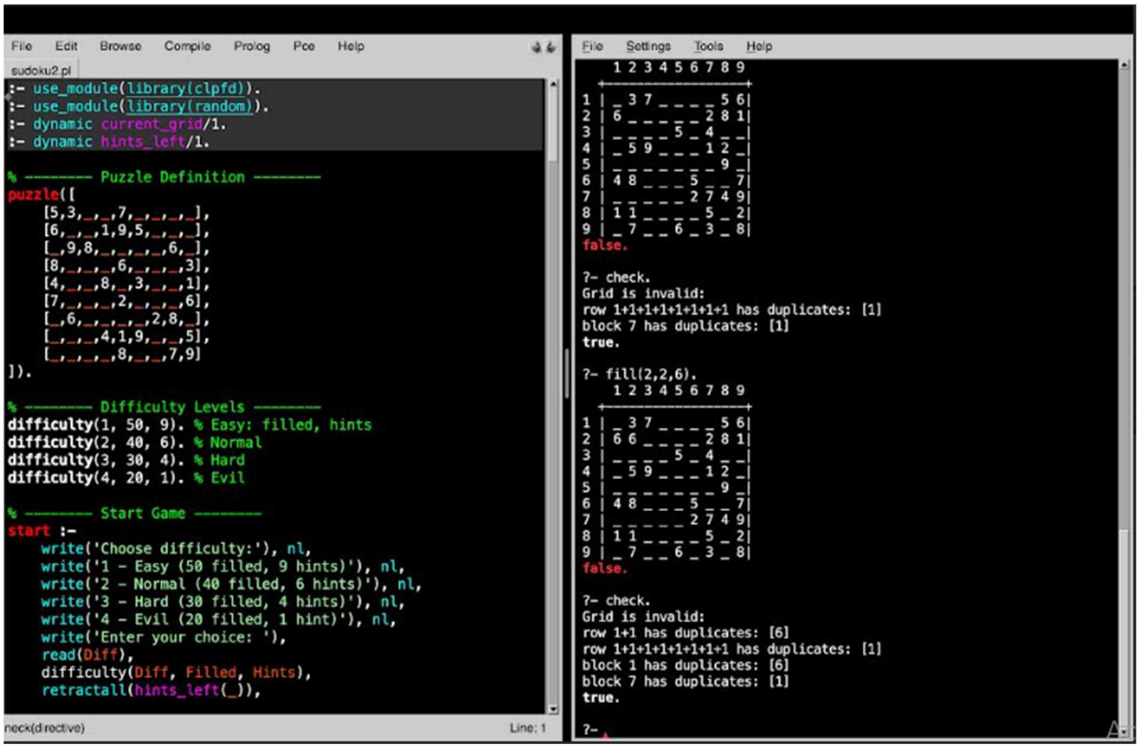
```prolog
:- use_module(library(clpfd)).
:- use_module(library(random)).
:- dynamic current_grid/1.
:- dynamic hints_left/1.

% -------- Puzzle Definition --------
puzzle([
    [5,3,_,_,7,_,_,_,_],
    [6,_,_,1,9,5,_,_,_],
    [_,9,8,_,_,_,_,6,_],
    [8,_,_,_,6,_,_,_,3],
    [4,_,_,8,_,3,_,_,1],
    [7,_,_,_,2,_,_,_,6],
    [_,6,_,_,_,_,2,8,_],
    [_,_,_,4,1,9,_,_,5],
    [_,_,_,_,8,_,_,7,9]
]).

% -------- Difficulty Levels --------
difficulty(1, 50, 9). % Easy: filled, hints
difficulty(2, 40, 6). % Normal
difficulty(3, 30, 4). % Hard
difficulty(4, 20, 1). % Evil

% -------- Start Game --------
start :-
    write('Choose difficulty:'), nl,
    write('1 - Easy (50 filled, 9 hints)'), nl,
    write('2 - Normal (40 filled, 6 hints)'), nl,
    write('3 - Hard (30 filled, 4 hints)'), nl,
    write('4 - Evil (20 filled, 1 hint)'), nl,
    write('Enter your choice: '),
    read(Diff),
    difficulty(Diff, Filled, Hints),
    retractall(hints_left(_)),
```

```
2 - Normal (40 filled, 6 hints)
3 - Hard (30 filled, 4 hints)
4 - Evil (20 filled, 1 hint)
Enter your choice: 2
|: .
    1 2 3 4 5 6 7 8 9
  +-------------------+
1 | _ _ 9 4 7 5 _ 1 _|
2 | _ 8 1 _ _ 6 _ _ _|
3 | 6 7 4 8 2 1 3 5 _|
4 | _ 2 7 _ _ _ 9 _ _|
5 | 4 _ 6 2 _ 7 8 _ _|
6 | _ 1 _ 5 6 8 _ 4 7|
7 | _ _ 2 _ _ _ 7 _ 6|
8 | _ _ 8 7 _ _ _ _ _|
9 | 7 4 _ 6 _ _ 1 9 _|
true .

?- hint.
row=1, col=1, num=2
Hints remaining: 5
true.

?- fill(1,1,2).
    1 2 3 4 5 6 7 8 9
  +-------------------+
1 | 2 _ 9 4 7 5 _ 1 _|
2 | _ 8 1 _ _ 6 _ _ _|
3 | 6 7 4 8 2 1 3 5 _|
4 | _ 2 7 _ _ _ 9 _ _|
5 | 4 _ 6 2 _ 7 8 _ _|
6 | _ 1 _ 5 6 8 _ 4 7|
7 | _ _ 2 _ _ _ 7 _ 6|
8 | _ _ 8 7 _ _ _ _ _|
9 | 7 4 _ 6 _ _ 1 9 _|
false.

?- check.
Grid is valid!
true.

?-
```

Here we can see that we can check if there are any mistakes that we have made by using the check. Keyword and it shows us if it’s valid or not.if not valid, it will show the reason too.

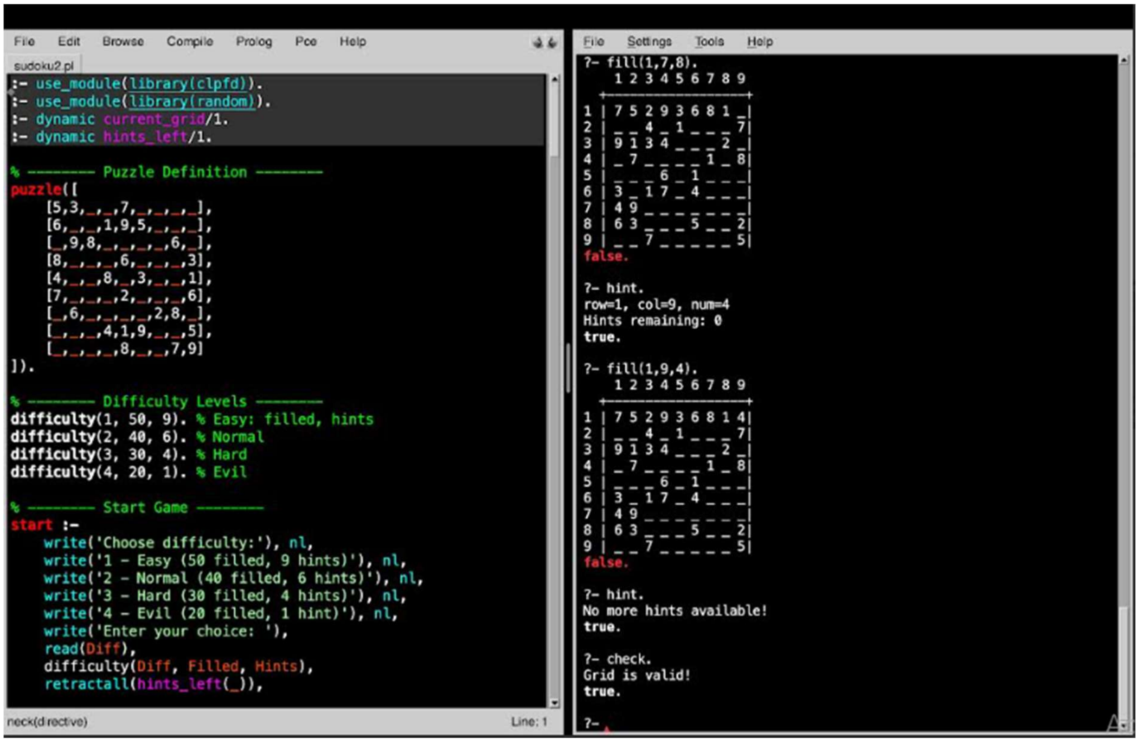
for example, I filled the row 8 with two 1 and it showed there are two duplicates in that row

1+1+1+1+1+1+1+1= 8and also, the block 7 has duplicate (1)



```
:- use_module(library(clpfd)).
:- use_module(library(random)).
:- dynamic current_grid/1.
:- dynamic hints_left/1.

% -------- Puzzle Definition --------
puzzle([
    [5,3,_,_,7,_,_,_,_],
    [6,_,_,1,9,5,_,_,_],
    [_,9,8,_,_,_,_,6,_],
    [8,_,_,_,6,_,_,_,3],
    [4,_,_,8,_,3,_,_,1],
    [7,_,_,_,2,_,_,_,6],
    [_,6,_,_,_,_,2,8,_],
    [_,_,_,4,1,9,_,_,5],
    [_,_,_,_,8,_,_,7,9]
]).

% -------- Difficulty Levels --------
difficulty(1, 50, 9). % Easy: filled, hints
difficulty(2, 40, 6). % Normal
difficulty(3, 30, 4). % Hard
difficulty(4, 20, 1). % Evil

% -------- Start Game --------
start :-
    write('Choose difficulty:'), nl,
    write('1 - Easy (50 filled, 9 hints)'), nl,
    write('2 - Normal (40 filled, 6 hints)'), nl,
    write('3 - Hard (30 filled, 4 hints)'), nl,
    write('4 - Evil (20 filled, 1 hint)'), nl,
    write('Enter your choice: '),
    read(Diff),
    difficulty(Diff, Filled, Hints),
    retractall(hints_left(_)),
```

```
    1 2 3 4 5 6 7 8 9
  +-------------------+
1 | _ 3 7 _ _ _ _ 5 6|
2 | 6 _ _ _ _ _ 2 8 1|
3 | _ _ _ _ 5 _ 4 _ _|
4 | _ 5 9 _ _ _ 1 2 _|
5 | _ _ _ _ _ _ _ 9 _|
6 | 4 8 _ _ _ 5 _ _ 7|
7 | _ _ _ _ _ 2 7 4 9|
8 | 1 1 _ _ _ _ 5 _ 2|
9 | _ 7 _ _ 6 _ 3 _ 8|
false.

?- check.
Grid is invalid:
row 1+1+1+1+1+1+1+1 has duplicates: [1]
block 7 has duplicates: [1]
true.

?- fill(2,2,6).
    1 2 3 4 5 6 7 8 9
  +-------------------+
1 | _ 3 7 _ _ _ _ 5 6|
2 | 6 6 _ _ _ _ 2 8 1|
3 | _ _ _ _ 5 _ 4 _ _|
4 | _ 5 9 _ _ _ 1 2 _|
5 | _ _ _ _ _ _ _ 9 _|
6 | 4 8 _ _ _ 5 _ _ 7|
7 | _ _ _ _ _ 2 7 4 9|
8 | 1 1 _ _ _ _ 5 _ 2|
9 | _ 7 _ _ 6 _ 3 _ 8|
false.

?- check.
Grid is invalid:
row 1+1 has duplicates: [6]
row 1+1+1+1+1+1+1+1 has duplicates: [1]
block 1 has duplicates: [6]
block 7 has duplicates: [1]
true.

?-
```



```
:- use_module(library(clpfd)).
:- use_module(library(random)).
:- dynamic current_grid/1.
:- dynamic hints_left/1.

% -------- Puzzle Definition --------
puzzle([
    [5,3,_,_,7,_,_,_,_],
    [6,_,_,1,9,5,_,_,_],
    [_,9,8,_,_,_,_,6,_],
    [8,_,_,_,6,_,_,_,3],
    [4,_,_,8,_,3,_,_,1],
    [7,_,_,_,2,_,_,_,6],
    [_,6,_,_,_,_,2,8,_],
    [_,_,_,4,1,9,_,_,5],
    [_,_,_,_,8,_,_,7,9]
]).

% -------- Difficulty Levels --------
difficulty(1, 50, 9). % Easy: filled, hints
difficulty(2, 40, 6). % Normal
difficulty(3, 30, 4). % Hard
difficulty(4, 20, 1). % Evil

% -------- Start Game --------
start :-
    write('Choose difficulty:'), nl,
    write('1 - Easy (50 filled, 9 hints)'), nl,
    write('2 - Normal (40 filled, 6 hints)'), nl,
    write('3 - Hard (30 filled, 4 hints)'), nl,
    write('4 - Evil (20 filled, 1 hint)'), nl,
    write('Enter your choice: '),
    read(Diff),
    difficulty(Diff, Filled, Hints),
    retractall(hints_left(_)),
```

```
?- fill(1,7,8).
    1 2 3 4 5 6 7 8 9
  +-----------------+
1 | 7 5 2 9 3 6 8 1 _|
2 | _ _ 4 _ 1 _ _ _ 7|
3 | 9 1 3 4 _ _ _ 2 _|
4 | _ 7 _ _ _ _ 1 _ 8|
5 | _ _ _ 6 _ 1 _ _ _|
6 | 3 _ 1 7 _ 4 _ _ _|
7 | 4 9 _ _ _ _ _ _ _|
8 | 6 3 _ _ _ 5 _ _ 2|
9 | _ _ 7 _ _ _ _ _ 5|
false.

?- hint.
row=1, col=9, num=4
Hints remaining: 0
true.

?- fill(1,9,4).
    1 2 3 4 5 6 7 8 9
  +-----------------+
1 | 7 5 2 9 3 6 8 1 4|
2 | _ _ 4 _ 1 _ _ _ 7|
3 | 9 1 3 4 _ _ _ 2 _|
4 | _ 7 _ _ _ _ 1 _ 8|
5 | _ _ _ 6 _ 1 _ _ _|
6 | 3 _ 1 7 _ 4 _ _ _|
7 | 4 9 _ _ _ _ _ _ _|
8 | 6 3 _ _ _ 5 _ _ 2|
9 | _ _ 7 _ _ _ _ _ 5|
false.

?- hint.
No more hints available!
true.

?- check.
Grid is valid!
true.

?-
```

It can show all the places where there are invalid fills, also which number is the one that is making this invalid is also shown on the box. It has limit feature to hint for every level and every hint is valid and correct.

Conclusion

This project clearly shows the strong connection between a logical problem and a programming language designed for logical reasoning. By building a complete Sudoku game in Prolog, it became possible to see how declarative programming can handle complex tasks in a very clean and elegant way.

One of the main strengths of this approach was the use of Constraint Logic Programming over Finite Domains, which made it possible to write the core rules of Sudoku in a simple and clear form. For example, using constraints such as all distinct directly matched the actual rules of the puzzle. This not only made the code easy to understand but also allowed Prolog's built-in backtracking and search features to take care of the heavy lifting in solving and generating puzzles. Another important part of the implementation was the use of dynamic predicates. They provided an effective way of managing the changing state of the game, making it possible to create features like filling cells, giving hints, or checking progress without complicating the overall structure.

The hint system was especially interesting, as it used Prolog's natural ability to test solutions and backtrack. This is something that would be difficult and time-consuming to build in many imperative languages, but it felt natural in Prolog. There is room for improvement. A graphical interface would make the game more user-friendly, and enhancing the puzzle generator so that every puzzle has a guaranteed unique solution would add even more value.

In the end, this project shows that Prolog is not just a theoretical language but a practical tool. It demonstrates how logic programming can be used to create software that is both intelligent and well-structured, with Sudoku serving as a fun and clear example of its power.

References

https://www.cl.cam.ac.uk/teaching/0809/Prolog/Prolog08M4upWithExercisesR1.pdf

https://stackoverflow.com/questions/44661508/prolog-solve-sudoku

Marking Scheme

Student name: Sneha Tamang, Prince Thapa, Akriti Gautam

Student id: 00021595, 00021592, 00021547

Criteria	**Allocated Marks**	**Obtained Marks**
Cover page	5 Marks	
Table of content	5 Marks	
Project Introduction	5 Marks	
Objectives of project	5 Marks	
Prolog Code	25 Marks	
Output	10 Marks	
Conclusion	5 Marks	
References	5 Marks	
Report format (Font size, style, formatting)	5 Marks	
Total	70 Marks	