```
/**
 * File     : proj2.pl
 * Author   : Yuqiang Zhu <yuqiangz@student.unimelb.edu.au>
 * Purpose  : Solve fill-in puzzle game by a heuristic method
 *
 * A fill-in puzzle is like a crossword puzzle, you are given a list of all
 * the words to place in the puzzle.
 *
 * Call puzzle_solution(Puzzle, WordList) to get solutions if they exist. It
 * will do the following processes:
 *  Firstly, all the slots will be extracted, excluding the slots which only
 *  have one square grid. Then each slot and words which they may fit in this
 *  slot compound the value of the dictionary. The key is the number of words.
 *  Every time, the program will try to fill the word into the slot from the
 *  value whose key is the smallest. When a word is filled into a slot, this
 *  key-value is removed and the rest of key-value will be updated. These two
 *  steps repeat until all the words are used.
 */

% load library
:- ensure_loaded(library(clpfd)).
:- ensure_loaded(library(pairs)).

% ************************************************************************
% ************************ The main entrance *****************************
% ************************************************************************

%%  puzzle_solution(+Puzzle, +WordList)
%   Try to fill all words from WordList into Puzzle.

puzzle_solution(Puzzle, WordList) :-
    % get all the slots, including the horizontal and vertical slots,
    % excluding slots which only have one square grid.
    get_puzzle_slots(Puzzle, Slots),

    % compound slot and suited words as pair
    get_all_pairs(WordList, Slots, Pairs),

    % sort Pairs by key
    keysort(Pairs, SortedPairs),

    % try to fill the slots.
    fill_slots(SortedPairs, WordList).

% ************************************************************************
% ************************ Fill-in work **********************************
% ************************************************************************

%%  fill_slots(+SortedPairs, +WordList)
%   Fill words in slots by heuristic method, always start to fill from the
%   slot which owns the smallest number of suited words.

%   At end: all the pairs are used, and all the words are filled in.
fill_slots([], []).
%   Accumulate first and recur. Try to pair whose the number of possible words
%   is smallest
fill_slots([(_-[Slot, Options])|Pairs], WordList) :-
    member(Slot, Options),
```

```
    % prepare for fill word to next slot
    update_pairs(Pairs, UpdatedPairs, Slot),
    keysort(UpdatedPairs, SortedPairs),
    select(Slot, WordList, RestWordList),

    % fill the rest of slots
    fill_slots(SortedPairs, RestWordList).


%% update_pairs(+Pairs, -UpdatedPairs, +RemoveWord)
%  UpdatedPairs get the updated Pairs which all the RemoveWord is removed
%  from the value.

update_pairs(Pairs, UpdatedPairs, RemoveWord) :-
    update_pairs(Pairs, [], UpdatedPairs, RemoveWord).

%  update_pairs(+Pairs, +Accumulator, -UpdatedPairs, +RemoveWord)

%  At end: unify with accumulator
update_pairs([], A, A, _).
%  Accumulate first and recur. RemoveWord is removed from Options, and all
%  the words in the Options are re-checked whether can fit in this Slot.
update_pairs([(_-[Slot, Options])|Pairs], A, UpdatedPairs, RemoveWord) :-
    delete(Options, RemoveWord, RestOptions),
    get_pair(RestOptions, Slot, Pair),
    append(A, Pair, B),
    update_pairs(Pairs, B, UpdatedPairs, RemoveWord).


%% get_all_pairs(+WordList, +Slots, -Pairs)
%  Pairs is a dictionary. Each slot and words which they may fit in this slot
%  compound the value of the dictionary. The key is the number of words.

get_all_pairs(WordList, Slots, Pairs) :-
    get_all_pairs(WordList, Slots, [], Pairs).

%  get_all_pairs(+WordList, +Slots, +Accumulator, -Pairs)

%  At end: unify with accumulator.
get_all_pairs(_, [], A, A).
%  Accumulate first and recur.
get_all_pairs(WordList, [Slot|Slots], A, Pairs) :-
    get_pair(WordList, Slot, Pair),
    append(A, Pair, B),
    get_all_pairs(WordList, Slots, B, Pairs).


%% get_pair(+WordList, +Slot, -Pair)
%  Pair is the list which contains a key-value pair. value is
%  [Slot,FilteredWords]. key is the number of FilteredWords.

get_pair(WordList, Slot, Pair):-
    filter_words(WordList, Slot, [], FilteredWords),
    length(FilteredWords, N),
    % the fromat of key-value
    Pair = [N-[Slot, FilteredWords]].


%% filter_words(+WordList, +Slot, +Accumulator, -Result)
```

```
%   Unifies Result with the list of words which fit in Slot.

%   At end: unify with accumulator
filter_words([], _, A, A).
%   Accumulate first and recur.
filter_words([Word|Words], Slot, A, Result) :-
    %   Word cannot fit in this Slot
    (   Slot \= Word
    ->  filter_words(Words, Slot, A, Result)
    %   Word fits in this Slot
    ;  append(A, [Word], B),
        filter_words(Words, Slot, B, Result)
    ).

% ************************************************************************
% ********************** Get the puzzle slots ****************************
% ************************************************************************

%%  get_puzzle_slots(+Puzzle, -AllSlots)
%   Extract all slots, excluding 1-square-sized slot, including horizontal
%   line and vertical line.
get_puzzle_slots(Puzzle, AllSlots) :-
    Rows = Puzzle,
    % get slots from horizontal lines
    get_all_slots(Rows, AllRowsSlots),
    transpose(Puzzle, Cols),
    % get slots from vertical lines
    get_all_slots(Cols, AllColsSlots),
    append(AllRowsSlots, AllColsSlots, AllSlots).


%%  get_all_slots(+RowList, -AllSlots)
%   Get all slots from the list of rows.

get_all_slots([], []).
get_all_slots([Row|Rows], AllSlots) :-
    get_slots(Row, Slots),
    get_all_slots(Rows, AllSlots1),
    append(Slots, AllSlots1, AllSlots).


%%  get_slots(+Row, -Slots)
%   Get the slots from a Row.

get_slots(Row, Slots) :-
    get_slots(Row, [[]], Slots).

%   get_slots(+Row, +Accumulator, -Slots)

%   At end: unify with accumulator which removes empty slot and 1-square-sized
%   slot.
get_slots([], A, Slots) :-
    % delete empty slot and 1-square-sized slot
    delete(A, [], B),
    exclude(filter_slot(1), B, Slots).
%   Accumulate first and recur. Deal with square one by one. A square may be
%   solid, or fill-in available, or already filled.
get_slots([Square|Squares], A, Slots) :-
    (
```

```
        % this square is solid
        Square == '#'
    ->  last(A, LastSlot),
        (
            % last square is solid or the head
            LastSlot == []
        ->  get_slots(Squares, A, Slots)
        ;
            % last slot is finished, create a new empty slot
            append(A, [[]], B),
            get_slots(Squares, B, Slots)
        )
    ;
        % this square is fill-in available or already filled, add this square
        % in the slot.
        length(A, N),
        nth1(N, A, LastSlot, RestSlots),
        append(LastSlot, [Square], NewSlot),
        append(RestSlots, [NewSlot], B),
        get_slots(Squares, B, Slots)
    ).


%%  filter_slot(?N, ?Word)
%   It works as length(Word, N) for filtering word with specific length as
%   Grok cannot load library(yall).

filter_slot(N, Word) :-
    length(Word, N).
```