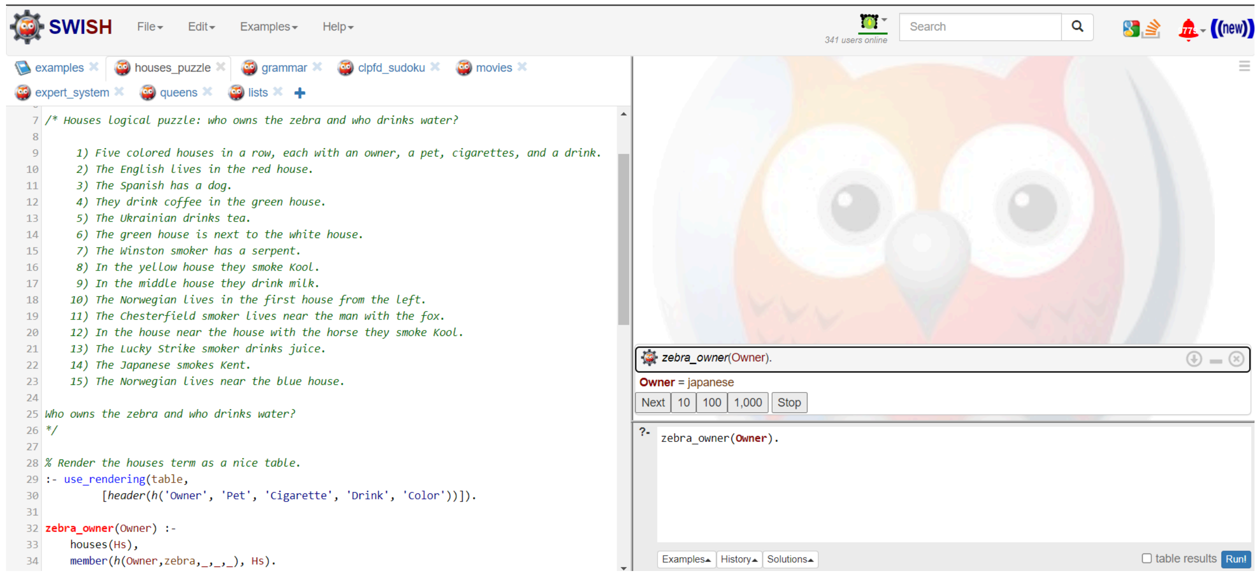
Prolog Programs

Name: Vedant Pathare
R.N: 9564, Comps-A
Batch B

1. Houses.pl

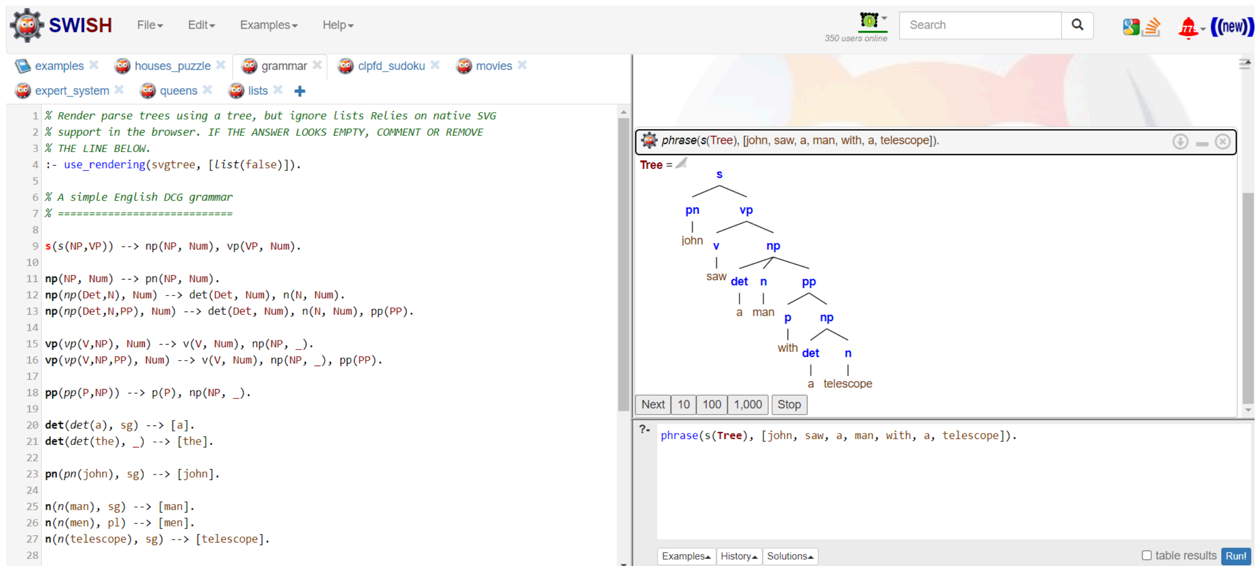


```
/* Houses logical puzzle: who owns the zebra and who drinks water?

     1) Five colored houses in a row, each with an owner, a pet, cigarettes, and a drink.
     2) The English lives in the red house.
     3) The Spanish has a dog.
     4) They drink coffee in the green house.
     5) The Ukrainian drinks tea.
     6) The green house is next to the white house.
     7) The Winston smoker has a serpent.
     8) In the yellow house they smoke Kool.
     9) In the middle house they drink milk.
    10) The Norwegian lives in the first house from the left.
    11) The Chesterfield smoker lives near the man with the fox.
    12) In the house near the house with the horse they smoke Kool.
    13) The Lucky Strike smoker drinks juice.
    14) The Japanese smokes Kent.
    15) The Norwegian lives near the blue house.

Who owns the zebra and who drinks water?
*/

% Render the houses term as a nice table.
:- use_rendering(table,
         [header(h('Owner', 'Pet', 'Cigarette', 'Drink', 'Color'))]).

zebra_owner(Owner) :-
    houses(Hs),
    member(h(Owner,zebra,_,_,_), Hs).
```

```
zebra_owner(Owner).
Owner = japanese
```

```
?- zebra_owner(Owner).
```

2. Grammar.pl

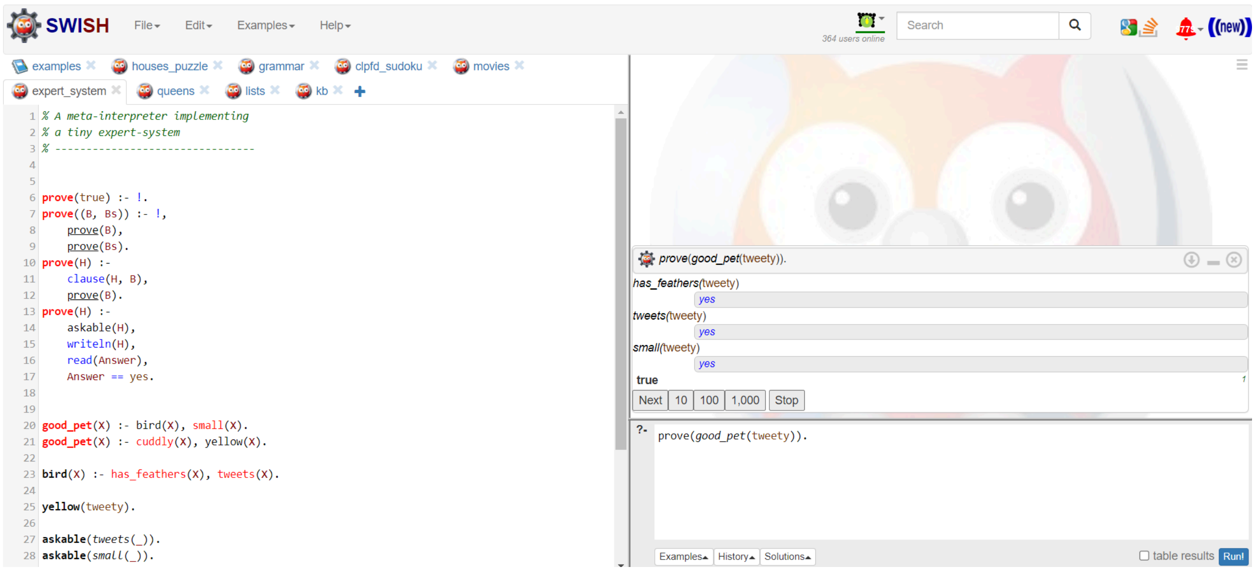


```
% Render parse trees using a tree, but ignore lists Relies on native SVG
% support in the browser. IF THE ANSWER LOOKS EMPTY, COMMENT OR REMOVE
% THE LINE BELOW.
:- use_rendering(svgtree, [list(false)]).

% A simple English DCG grammar
% ===========================

s(s(NP,VP)) --> np(NP, Num), vp(VP, Num).

np(NP, Num) --> pn(NP, Num).
np(np(Det,N), Num) --> det(Det, Num), n(N, Num).
np(np(Det,N,PP), Num) --> det(Det, Num), n(N, Num), pp(PP).

vp(vp(V,NP), Num) --> v(V, Num), np(NP, _).
vp(vp(V,NP,PP), Num) --> v(V, Num), np(NP, _), pp(PP).

pp(pp(P,NP)) --> p(P), np(NP, _).

det(det(a), sg) --> [a].
det(det(the), _) --> [the].

pn(pn(john), sg) --> [john].

n(n(man), sg) --> [man].
n(n(men), pl) --> [men].
n(n(telescope), sg) --> [telescope].

```

```
phrase(s(Tree), [john, saw, a, man, with, a, telescope]).
```

```
?- phrase(s(Tree), [john, saw, a, man, with, a, telescope]).
```

3. ExpertSystem.pl

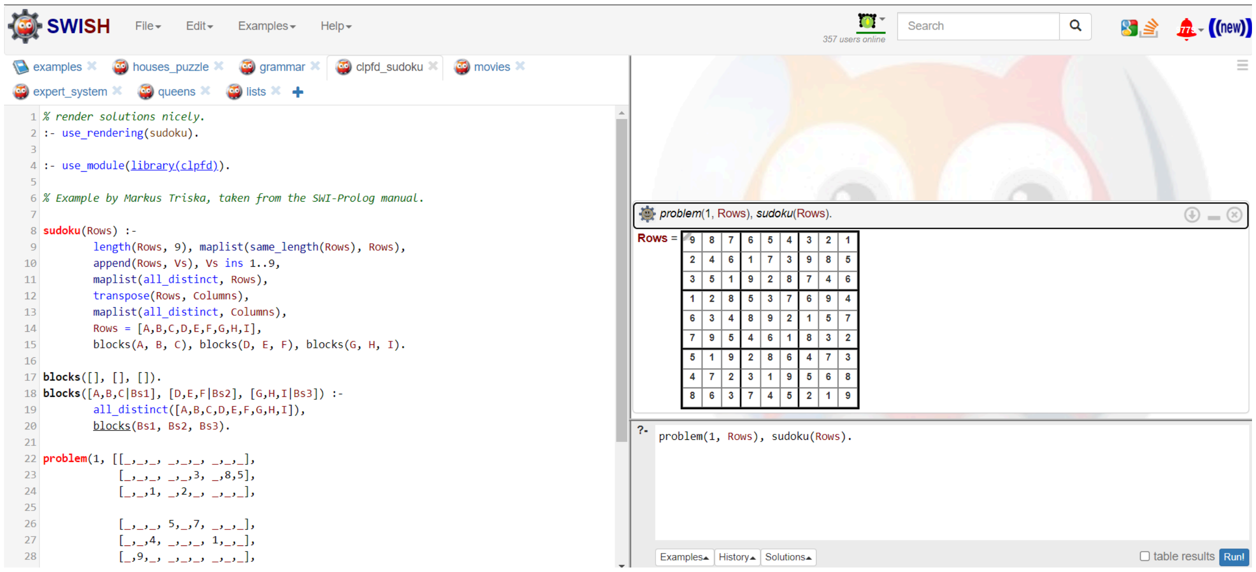


```prolog
% A meta-interpreter implementing
% a tiny expert-system
% -------------------------------


prove(true) :- !.
prove((B, Bs)) :- !,
    prove(B),
    prove(Bs).
prove(H) :-
    clause(H, B),
    prove(B).
prove(H) :-
    askable(H),
    writeln(H),
    read(Answer),
    Answer == yes.


good_pet(X) :- bird(X), small(X).
good_pet(X) :- cuddly(X), yellow(X).

bird(X) :- has_feathers(X), tweets(X).

yellow(tweety).

askable(tweets(_)).
askable(small(_)).
```

```
prove(good_pet(tweety)).
has_feathers(tweety)
    yes
tweets(tweety)
    yes
small(tweety)
    yes
true
```

```
?- prove(good_pet(tweety)).
```

4. Sudoku.pl



```prolog
% render solutions nicely.
:- use_rendering(sudoku).

:- use_module(library(clpfd)).

% Example by Markus Triska, taken from the SWI-Prolog manual.

sudoku(Rows) :-
        length(Rows, 9), maplist(same_length(Rows), Rows),
        append(Rows, Vs), Vs ins 1..9,
        maplist(all_distinct, Rows),
        transpose(Rows, Columns),
        maplist(all_distinct, Columns),
        Rows = [A,B,C,D,E,F,G,H,I],
        blocks(A, B, C), blocks(D, E, F), blocks(G, H, I).

blocks([], [], []).
blocks([A,B,C|Bs1], [D,E,F|Bs2], [G,H,I|Bs3]) :-
        all_distinct([A,B,C,D,E,F,G,H,I]),
        blocks(Bs1, Bs2, Bs3).

problem(1, [[_,_,_, _,_,_, _,_,_],
            [_,_,_, _,_,3, _,8,5],
            [_,_,1, _,2,_, _,_,_],

            [_,_,_, 5,_,7, _,_,_],
            [_,_,4, _,_,_, 1,_,_],
            [_,9,_, _,_,_, _,_,_],
```

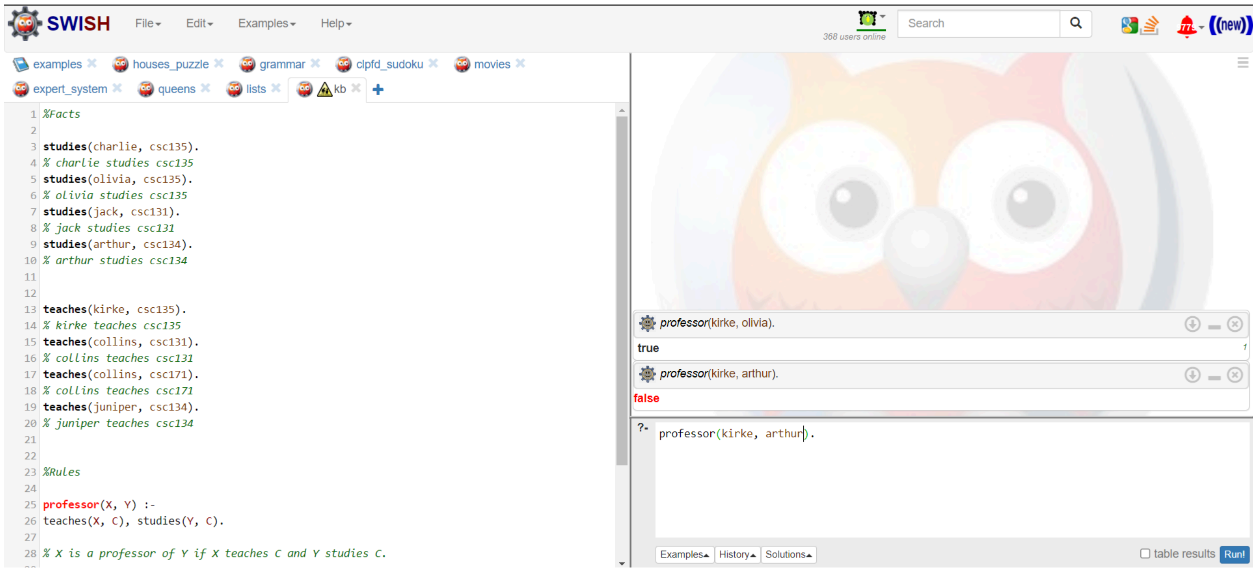
problem(1, Rows), sudoku(Rows).

Rows =

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

```
?- problem(1, Rows), sudoku(Rows).
```

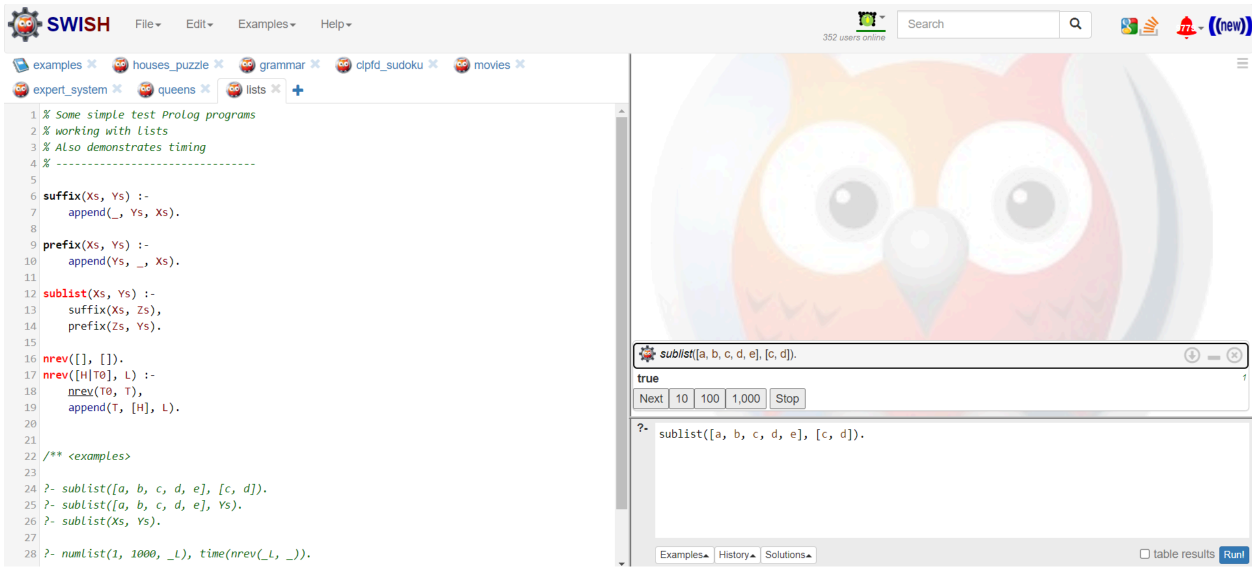
5. Teacher.pl



```
%Facts

studies(charlie, csc135).
% charlie studies csc135
studies(olivia, csc135).
% olivia studies csc135
studies(jack, csc131).
% jack studies csc131
studies(arthur, csc134).
% arthur studies csc134


teaches(kirke, csc135).
% kirke teaches csc135
teaches(collins, csc131).
% collins teaches csc131
teaches(collins, csc171).
% collins teaches csc171
teaches(juniper, csc134).
% juniper teaches csc134


%Rules

professor(X, Y) :-
teaches(X, C), studies(Y, C).

% X is a professor of Y if X teaches C and Y studies C.
```

```
professor(kirke, olivia).
true

professor(kirke, arthur).
false

?- professor(kirke, arthur).
```

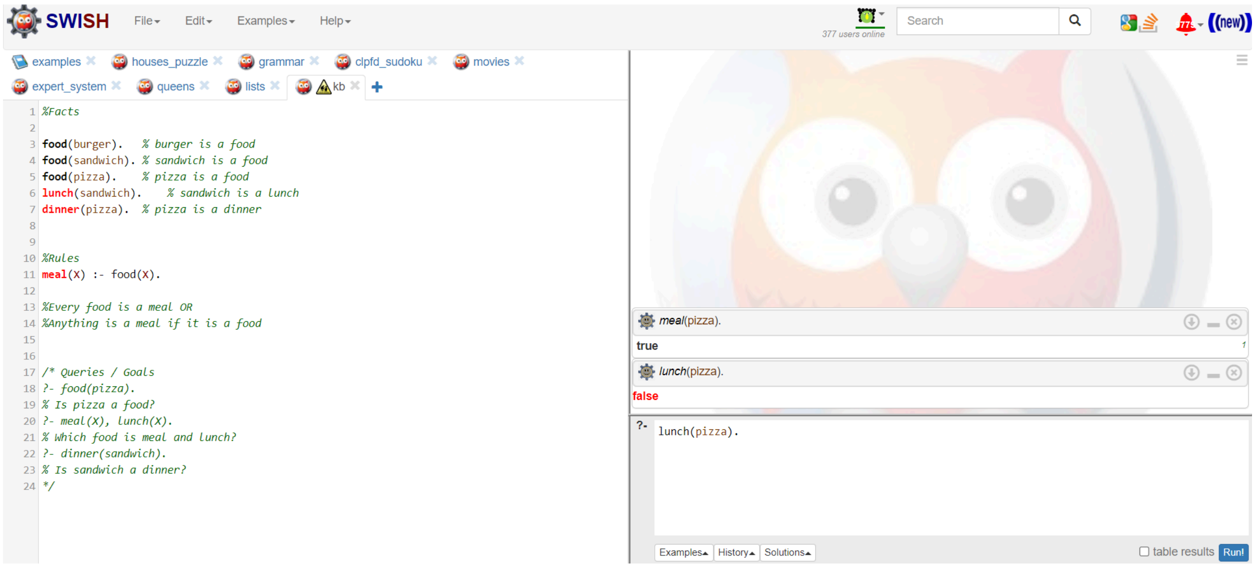
6. Lists.pl



```
% Some simple test Prolog programs
% working with lists
% Also demonstrates timing
% -------------------------------

suffix(Xs, Ys) :-
    append(_, Ys, Xs).

prefix(Xs, Ys) :-
    append(Ys, _, Xs).

sublist(Xs, Ys) :-
    suffix(Xs, Zs),
    prefix(Zs, Ys).

nrev([], []).
nrev([H|T0], L) :-
    nrev(T0, T),
    append(T, [H], L).


/** <examples>

?- sublist([a, b, c, d, e], [c, d]).
?- sublist([a, b, c, d, e], Ys).
?- sublist(Xs, Ys).

?- numlist(1, 1000, _L), time(nrev(_L, _)).
```

```
sublist([a, b, c, d, e], [c, d]).
true

?- sublist([a, b, c, d, e], [c, d]).
```

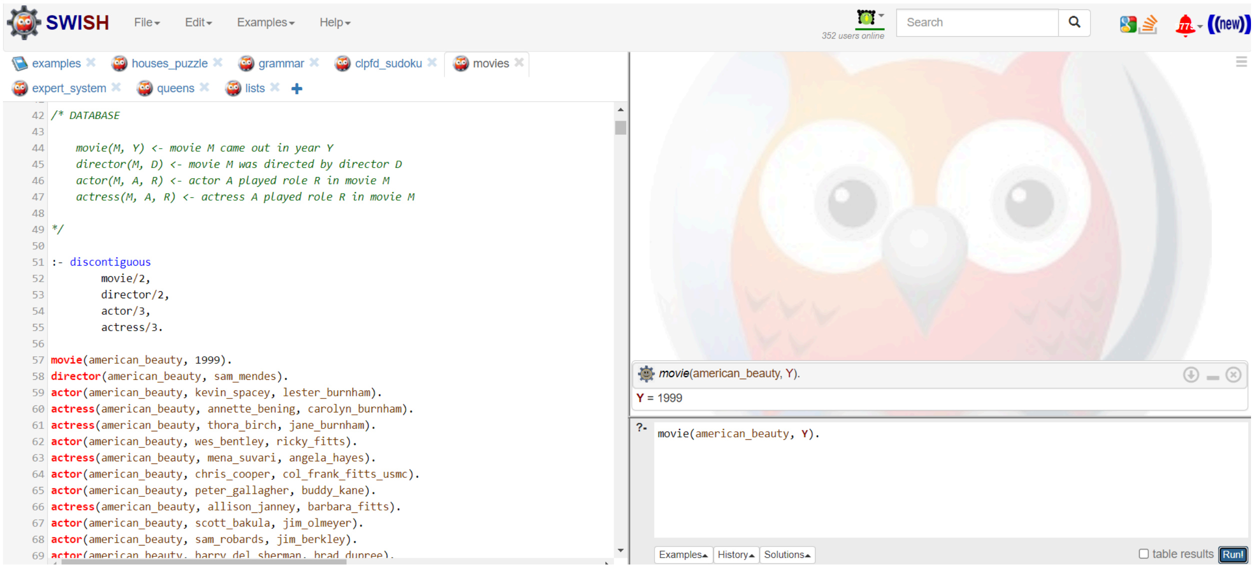
7. Meal.pl



```
%Facts

food(burger).   % burger is a food
food(sandwich). % sandwich is a food
food(pizza).    % pizza is a food
lunch(sandwich).    % sandwich is a lunch
dinner(pizza).  % pizza is a dinner


%Rules
meal(X) :- food(X).

%Every food is a meal OR
%Anything is a meal if it is a food


/* Queries / Goals
?- food(pizza).
% Is pizza a food?
?- meal(X), lunch(X).
% Which food is meal and lunch?
?- dinner(sandwich).
% Is sandwich a dinner?
*/
```

```
meal(pizza).
true

lunch(pizza).
false

?- lunch(pizza).
```

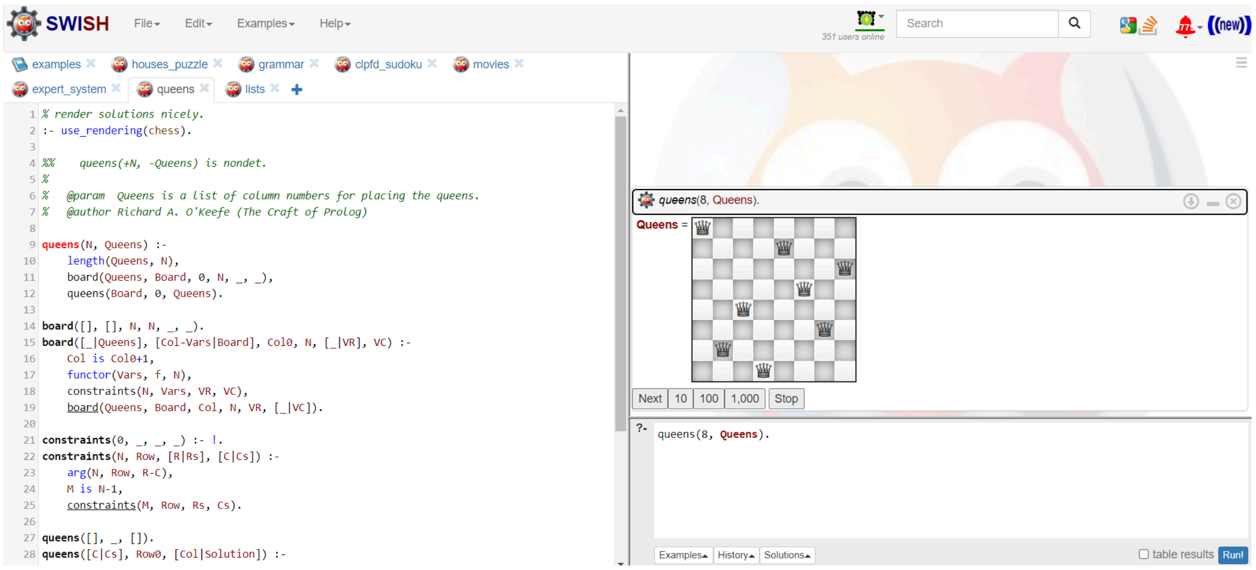
8. Movies.pl



```
/* DATABASE

    movie(M, Y) <- movie M came out in year Y
    director(M, D) <- movie M was directed by director D
    actor(M, A, R) <- actor A played role R in movie M
    actress(M, A, R) <- actress A played role R in movie M

*/

:- discontiguous
        movie/2,
        director/2,
        actor/3,
        actress/3.

movie(american_beauty, 1999).
director(american_beauty, sam_mendes).
actor(american_beauty, kevin_spacey, lester_burnham).
actress(american_beauty, annette_bening, carolyn_burnham).
actress(american_beauty, thora_birch, jane_burnham).
actor(american_beauty, wes_bentley, ricky_fitts).
actress(american_beauty, mena_suvari, angela_hayes).
actor(american_beauty, chris_cooper, col_frank_fitts_usmc).
actor(american_beauty, peter_gallagher, buddy_kane).
actress(american_beauty, allison_janney, barbara_fitts).
actor(american_beauty, scott_bakula, jim_olmeyer).
actor(american_beauty, sam_robards, jim_berkley).
actor(american_beauty, barry_del_sherman, brad_dupree).
```

```
movie(american_beauty, Y).
Y = 1999

?- movie(american_beauty, Y).
```

9. NQueens.pl

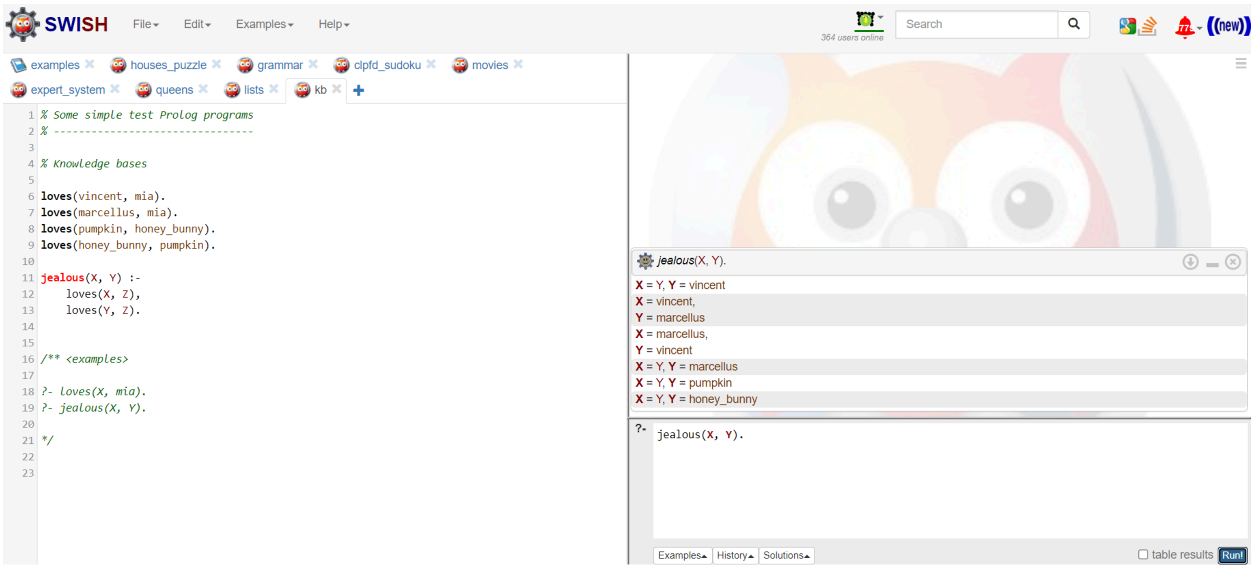


```prolog
% render solutions nicely.
:- use_rendering(chess).

%%    queens(+N, -Queens) is nondet.
%
%   @param  Queens is a list of column numbers for placing the queens.
%   @author Richard A. O'Keefe (The Craft of Prolog)

queens(N, Queens) :-
    length(Queens, N),
    board(Queens, Board, 0, N, _, _),
    queens(Board, 0, Queens).

board([], [], N, N, _, _).
board([_|Queens], [Col-Vars|Board], Col0, N, [_|VR], VC) :-
    Col is Col0+1,
    functor(Vars, f, N),
    constraints(N, Vars, VR, VC),
    board(Queens, Board, Col, N, VR, [_|VC]).

constraints(0, _, _, _) :- !.
constraints(N, Row, [R|Rs], [C|Cs]) :-
    arg(N, Row, R-C),
    M is N-1,
    constraints(M, Row, Rs, Cs).

queens([], _, []).
queens([C|Cs], Row0, [Col|Solution]) :-
```

```
queens(8, Queens).
Queens =
```

```
?- queens(8, Queens).
```

10. Jealous.pl



```prolog
% Some simple test Prolog programs
% ------------------------------

% Knowledge bases

loves(vincent, mia).
loves(marcellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X, Y) :-
    loves(X, Z),
    loves(Y, Z).


/** <examples>

?- loves(X, mia).
?- jealous(X, Y).

*/
```

```
jealous(X, Y).
X = Y, Y = vincent
X = vincent,
Y = marcellus
X = marcellus,
Y = vincent
X = Y, Y = marcellus
X = Y, Y = pumpkin
X = Y, Y = honey_bunny
```

```
?- jealous(X, Y).
```