AI Project

Title : “Romania Map Route Finding using Prolog ”

Source Code (Prolog)

```
:- use_module(library(pce)).
:- dynamic car_position/1.

% -------------------------
% Cities with coordinates
% -------------------------
city(arab,       50, 100).
city(zerind,     80,  60).
city(oradea,    120,  40).
city(sibiu,     150, 120).
city(timisoara,  40, 150).
city(lugoj,      80, 180).
city(mehadia,   100, 210).
city(drobeta,   120, 240).
city(craiova,   180, 240).
city(rimnicu,   180, 160).
city(pitesti,   230, 190).
city(bucharest, 300, 200).
city(fagaras,   200, 150).
city(giurgiu,   300, 240).
city(urziceni,  330, 160).
city(hirsova,   370, 150).
city(eforie,    390, 180).
city(vaslui,    360, 120).
city(iasi,      340, 100).
city(neamt,     310,  80).

% -------------------------
% Roads with distances
% -------------------------
road(arab, zerind, 75).
road(arab, sibiu, 140).
road(arab, timisoara, 118).
road(zerind, oradea, 71).
road(oradea, sibiu, 151).
```

```
road(timisoara, lugoj, 111).
road(lugoj, mehadia, 70).
road(mehadia, drobeta, 75).
road(drobeta, craiova, 120).
road(craiova, pitesti, 138).
road(craiova, rimnicu, 146).
road(rimnicu, sibiu, 80).
road(rimnicu, pitesti, 97).
road(sibiu, fagaras, 99).
road(fagaras, bucharest, 211).
road(pitesti, bucharest, 101).
road(bucharest, giurgiu, 90).
road(bucharest, urziceni, 85).
road(urziceni, hirsova, 98).
road(hirsova, eforie, 86).
road(urziceni, vaslui, 142).
road(vaslui, iasi, 92).
road(iasi, neamt, 87).

% bidirectional
connected(A,B,C) :- road(A,B,C).
connected(A,B,C) :- road(B,A,C).

% -------------------------
% Heuristic values (straight-line to Bucharest)
% -------------------------
sld(arab,366).  sld(zerind,374).  sld(oradea,380).
sld(timisoara,329). sld(lugoj,244). sld(mehadia,241).
sld(drobeta,242). sld(craiova,160). sld(sibiu,253).
sld(fagaras,176). sld(rimnicu,193). sld(pitesti,100).
sld(bucharest,0). sld(giurgiu,77). sld(urziceni,80).
sld(hirsova,151). sld(eforie,161). sld(vaslui,199).
sld(iasi,226). sld(neamt,234).

% -------------------------
% Path cost
% -------------------------
path_cost([_],0).
path_cost([A,B|Rest],Cost) :-
   connected(A,B,C),
   path_cost([B|Rest],Sub),
   Cost is C+Sub.
```

```
% -------------------------
% BFS
% -------------------------
bfs_with_visited(Start, Goal, Path, Cost, Visited) :-
   bfs_queue([[Start]], Goal, RevPath, [Start], VisRev),
   reverse(RevPath, Path),
   reverse(VisRev, Visited),
   path_cost(Path, Cost).

bfs_queue([[Goal|Rest]|_], Goal, [Goal|Rest], VisAcc, VisAcc).
bfs_queue([[Node|Rest]|Queue], Goal, SolPath, VisAcc, Visited) :-
   findall([Next,Node|Rest],
         (connected(Node,Next,_), \+ member(Next,[Node|Rest])),
         NewPaths),
   append(Queue, NewPaths, Queue1),
   append(VisAcc,[Node], VisAcc1),
   bfs_queue(Queue1, Goal, SolPath, VisAcc1, Visited).

% -------------------------
% DFS
% -------------------------
dfs_with_visited(Start, Goal, Path, Cost, Visited) :-
   dfs([[Start]], Goal, RevPath, [Start], VisRev),
   reverse(RevPath, Path),
   reverse(VisRev, Visited),
   path_cost(Path, Cost).

dfs([[Goal|Rest]|_], Goal, [Goal|Rest], VisAcc, VisAcc).
dfs([[Node|Rest]|Stack], Goal, SolPath, VisAcc, Visited) :-
   findall([Next,Node|Rest],
         (connected(Node,Next,_), \+ member(Next,[Node|Rest])),
         Children),
   append(Children, Stack, NewStack),
   append(VisAcc,[Node], VisAcc1),
   dfs(NewStack, Goal, SolPath, VisAcc1, Visited).

% -------------------------
% Greedy
% -------------------------
greedy_with_visited(Start, Goal, Path, Cost, Visited) :-
   greedy([[Start]], Goal, RevPath, [Start], VisRev),
   reverse(RevPath, Path),
   reverse(VisRev, Visited),
```

```
    path_cost(Path, Cost).

greedy([[Goal|Rest]|_], Goal, [Goal|Rest], VisAcc, VisAcc).
greedy(Frontier, Goal, SolPath, VisAcc, Visited) :-
    map_list_to_pairs(h_of_path(Goal), Frontier, Pairs),
    keysort(Pairs, Sorted),
    pairs_values(Sorted, [BestPath|OtherPaths]),
    BestPath = [Node|RestOfBest],
    findall([Next,Node|RestOfBest],
            (connected(Node,Next,_), \+ member(Next,[Node|RestOfBest])),
            Children),
    append(OtherPaths, Children, NewFrontier),
    append(VisAcc,[Node], VisAcc1),
    greedy(NewFrontier, Goal, SolPath, VisAcc1, Visited).

h_of_path(Goal, [Node|_], H) :-
    ( sld(Node,S) -> H = S
    ; city(Node,X1,Y1), city(Goal,X2,Y2),
      DX is X1-X2, DY is Y1-Y2, H is sqrt(DX*DX+DY*DY)
    ).

% -------------------------
% A*
% -------------------------
astar_with_visited(Start, Goal, Path, Cost, Visited) :-
    astar([node([Start],0,0)], Goal, RevPath, Cost, [Start], VisRev),
    reverse(RevPath, Path),
    reverse(VisRev, Visited).

astar(Open, Goal, Path, Cost, VisAcc, Visited) :-
    select_best(Open, node([Node|Rest],G,_F), Others),
    ( Node = Goal ->
        Path = [Node|Rest],
        Cost = G,
        Visited = VisAcc
    ; findall(node([Next,Node|Rest], G1, F1),
              ( connected(Node,Next,C),
                \+ member(Next,[Node|Rest]),
                G1 is G + C,
                ( sld(Next,H) -> true
                ; city(Next,Xn,Yn), city(Goal,Xg,Yg),
                  DX is Xn-Xg, DY is Yn-Yg, H is sqrt(DX*DX+DY*DY)
                ),
```

```
            F1 is G1 + H
          ),
          Children),
    append(Others, Children, NewOpen),
    append(VisAcc, [Node], VisAcc1),
    astar(NewOpen, Goal, Path, Cost, VisAcc1, Visited)
  ).

select_best([N], N, []) :- !.
select_best([node(P,G,F)|T], Best, Rest) :-
  select_best_helper(T, node(P,G,F), Best, Rest).

select_best_helper([], Best, Best, []).
select_best_helper([node(P,G,F)|T], node(P0,G0,F0), Best, [W|Rest]) :-
  ( F < F0 ->
    W = node(P0,G0,F0),
    Next = node(P,G,F)
  ;
    W = node(P,G,F),
    Next = node(P0,G0,F0)
  ),
  select_best_helper(T, Next, Best, Rest).

% -------------------------
% Algorithm chooser
% -------------------------
choose_algo(bfs, S, G, P, C, V) :- bfs_with_visited(S,G,P,C,V).
choose_algo(dfs, S, G, P, C, V) :- dfs_with_visited(S,G,P,C,V).
choose_algo(greedy, S, G, P, C, V) :- greedy_with_visited(S,G,P,C,V).
choose_algo(astar, S, G, P, C, V) :- astar_with_visited(S,G,P,C,V).

% -------------------------
% GUI Menu
% -------------------------
start_menu :-
  new(D, dialog('Romania Search Game')),
  send(D, size, size(520,300)),
  send(D, background, colour(grey90)),

  send(D, append, new(_, text('Start City:'))),
  send(D, append, new(StartMenu, menu(start, cycle))),
  send(D, append, new(_, text('Goal City:'))),
  send(D, append, new(GoalMenu, menu(goal, cycle))),
```

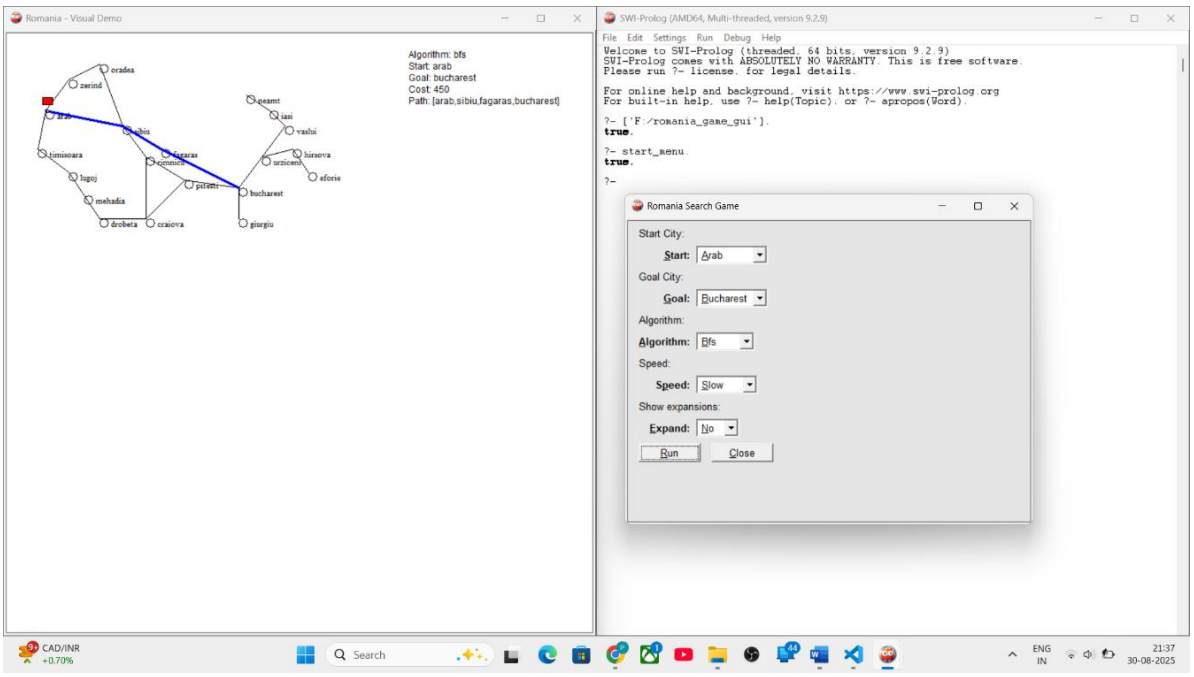
```
    send(D, append, new(_, text('Algorithm:'))),
    send(D, append, new(AlgoMenu, menu(algorithm, cycle))),
    send(D, append, new(_, text('Speed:'))),
    send(D, append, new(SpeedMenu, menu(speed, cycle))),
    send(D, append, new(_, text('Show expansions:'))),
    send(D, append, new(ExpMenu, menu(expand, cycle))),

    forall(city(C,_,_), send(StartMenu, append, C)),
    forall(city(C,_,_), send(GoalMenu, append, C)),
    send_list(AlgoMenu, append, [bfs, dfs, greedy, astar]),
    send_list(SpeedMenu, append, [slow, medium, fast]),
    send_list(ExpMenu, append, [no, yes]),

    send(D, append,
        button(run,
            message(@prolog, run_gui,
                StartMenu?selection,
                GoalMenu?selection,
                AlgoMenu?selection,
                SpeedMenu?selection,
                ExpMenu?selection))),
    send(D, append, button(close, message(D, destroy))),
    send(D, open).

% -------------------------
% GUI Runner
% -------------------------
speed_delay(slow, 300). speed_delay(medium, 120). speed_delay(fast, 40).

run_gui(Start, Goal, Algo, Speed, Exp) :-
    choose_algo(Algo, Start, Goal, Path, Cost, Visited),
    new(W, picture('Romania - Visual Demo', size(820,560))),
    send(W, open),
    draw_map(W),
    format(string(InfoStr),
        "Algorithm: ~w~nStart: ~w~nGoal: ~w~nCost: ~w~nPath: ~w",
        [Algo, Start, Goal, Cost, Path]),
    send(W, display, new(Info, text(InfoStr)), point(520,20)),
    send(Info, font, font(helvetica, plain, 11)),
    ( Exp == yes ->
      speed_delay(Speed, Dexp),
      show_expansions(W, Visited, Dexp)
    ; true ),
```

```
    highlight_path(W, Path),
    speed_delay(Speed, Dmove),
    animate_path(W, Path, Dmove),
    send(W, display, new(M, text('Reached destination! ✅')), point(520,160)),
    send(M, font, font(helvetica, bold, 12)).

% -------------------------
% Drawing
% -------------------------
draw_map(W) :-
    send(W, clear),
    forall((connected(A,B,_), city(A,X1,Y1), city(B,X2,Y2)),
           send(W, display, new(_, line(X1,Y1,X2,Y2)))),
    forall(city(Name,X,Y),
           ( send(W, display, new(_, circle(12)), point(X,Y)),
             send(W, display, new(Label, text(Name)), point(X+15, Y)),
             send(Label, font, font(times, roman, 10))
           )).

show_expansions(_, [], _) :- !.
show_expansions(W, [N|Rest], Delay) :-
    ( city(N,X,Y) ->
        send(W, display, new(M, circle(6)), point(X-3, Y-3)),
        send(M, fill_pattern, colour(green)),
        send(M, pen, 0)
    ; true ),
    sleep(Delay/1000),
    show_expansions(W, Rest, Delay).

highlight_path(_, []) :- !.
highlight_path(_, [_]) :- !.
highlight_path(W, [A,B|Rest]) :-
    city(A,X1,Y1), city(B,X2,Y2),
    send(W, display, new(Line, line(X1,Y1,X2,Y2))),
    send(Line, colour, blue),
    send(Line, pen, 3),
    highlight_path(W, [B|Rest]).

animate_path(_, [], _) :- !.
animate_path(W, [City], _) :-
    draw_car(W, City), !.
animate_path(W, [C1,C2|Rest], Delay) :-
    city(C1,X1,Y1), city(C2,X2,Y2),
```

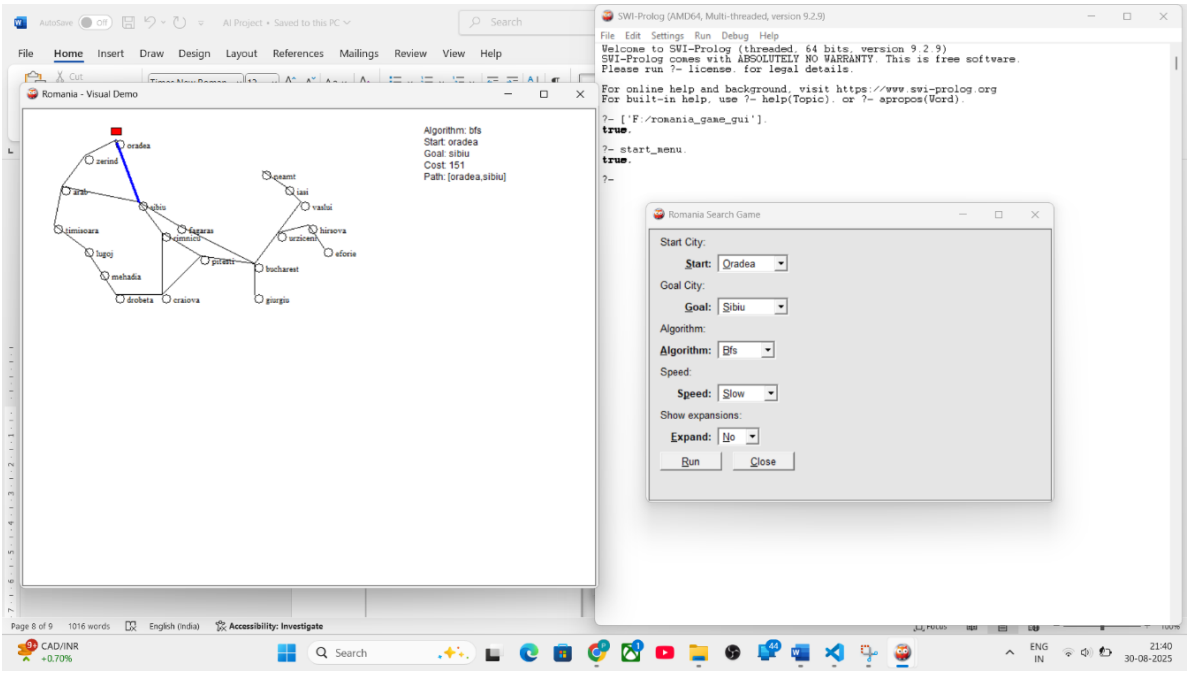
```
    move_car_smooth(W, X1, Y1, X2, Y2, Delay),
    animate_path(W, [C2|Rest], Delay).

draw_car(W, City) :-
    city(City, X, Y),
    send(W, display, new(Car, box(14,10)), point(X-7, Y-18)),
    send(Car, fill_pattern, colour(red)).

move_car_smooth(W, X1, Y1, X2, Y2, Delay) :-
    Steps = 36,
    DX is (X2 - X1) / Steps,
    DY is (Y2 - Y1) / Steps,
    forall(between(1, Steps, I),
          ( NX is X1 + DX * I,
            NY is Y1 + DY * I,
            send(W, display, new(Trail, circle(4)), point(NX-2, NY-2)),
            send(Trail, fill_pattern, colour(lightgrey)),
            send(W, display, new(Car, box(14,10)), point(NX-7, NY-18)),
            send(Car, fill_pattern, colour(red)),
            sleep(Delay/1000)
          )).
```

Output:-

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
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).

?- ['F:/romania_game_gui'].
true.

?- start_menu.
true.
```

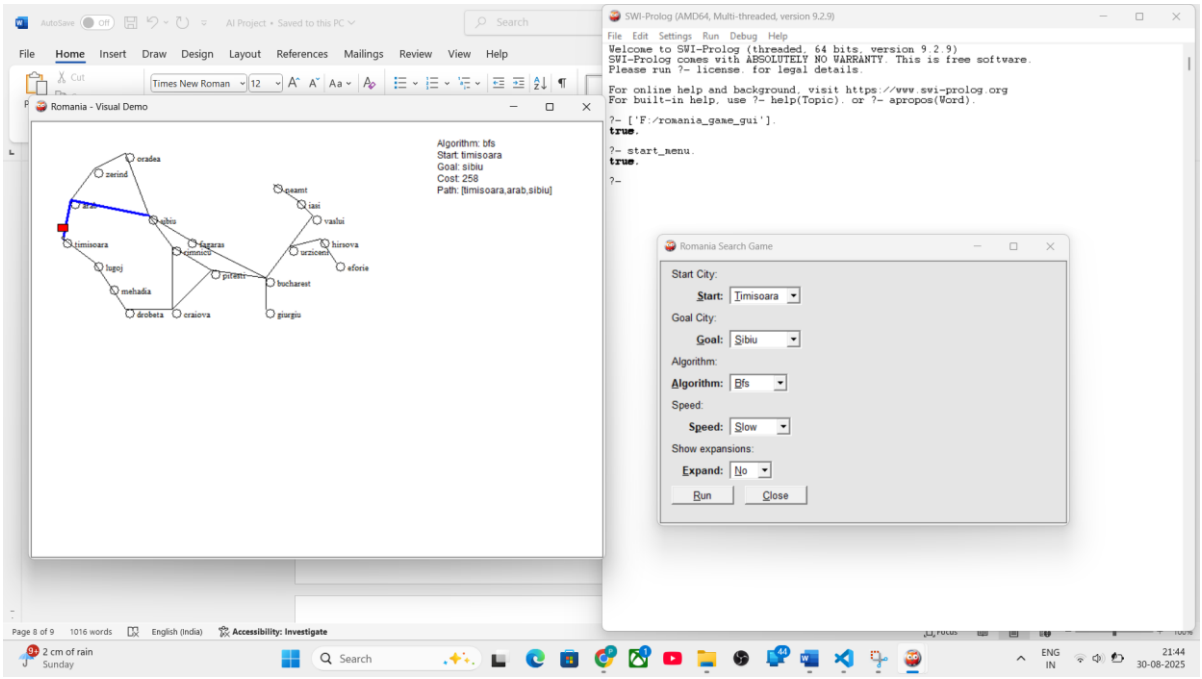


- **Algorithm:** BFS (Breadth-First Search)
- **Start:** Arad
- **Goal:** Bucharest
- **Cost:** 450 (sum of edge weights/distances)
- **Path:** [arad, sibiu, fagaras, bucharest]



- **Algorithm:** BFS (Breadth-First Search)
- **Start:** Oradea
- **Goal:** Sibiu

- **Cost:** 151 (sum of edge weights/distances)
- **Path:** [oradea, sibiu]



- **Algorithm:** BFS (Breadth-First Search)
- **Start:** Timisoara
- **Goal:** Sibiu
- **Cost:** 258 (sum of edge weights/distances)
- **Path:** [timisoara, arad, sibiu]