

```
% Maze Solver in Prolog  
% wall(X, Y) - defines a wall at position (X, Y)  
% path(X, Y) - defines a valid path at position (X, Y)
```

```
% Sample maze configuration (10x10 grid)  
% Walls are represented as wall(Row, Col)  
maze_size(10).
```

```
% Define walls for a sample maze  
wall(0, X) :- between(0, 9, X).  
wall(9, X) :- between(0, 9, X).  
wall(X, 0) :- between(0, 9, X).  
wall(X, 9) :- between(0, 9, X).
```

```
% Internal walls (example pattern)  
wall(2, 2). wall(2, 3). wall(2, 4).  
wall(3, 6). wall(4, 6). wall(5, 6).  
wall(6, 2). wall(6, 3). wall(6, 4).  
wall(7, 4). wall(7, 5). wall(7, 6).
```

```
% Define start and exit positions  
start_position(1, 1).  
exit_position(8, 8).
```

```
% Check if a position is valid (not a wall and within bounds)  
valid_position(X, Y) :-  
    maze_size(Size),  
    X >= 0, X < Size,  
    Y >= 0, Y < Size,
```

```

\+ wall(X, Y).

% Define possible moves (up, down, left, right)

move(X, Y, X1, Y) :- X1 is X - 1, valid_position(X1, Y). % Up
move(X, Y, X1, Y) :- X1 is X + 1, valid_position(X1, Y). % Down
move(X, Y, X, Y1) :- Y1 is Y - 1, valid_position(X, Y1). % Left
move(X, Y, X, Y1) :- Y1 is Y + 1, valid_position(X, Y1). % Right

```

% Depth-first search to find path from current position to exit

```

solve_maze(Path) :-
    start_position(StartX, StartY),
    exit_position(ExitX, ExitY),
    dfs((StartX, StartY), (ExitX, ExitY), [(StartX, StartY)], ReversePath),
    reverse(ReversePath, Path).

```

% DFS implementation

```

dfs((X, Y), (X, Y), Visited, Visited).
dfs((X, Y), Exit, Visited, Path) :-
    move(X, Y, X1, Y1),
    \+ member((X1, Y1), Visited),
    dfs((X1, Y1), Exit, [(X1, Y1)|Visited], Path).

```

% Breadth-first search (finds shortest path)

```

solve_maze_bfs(Path) :-
    start_position(StartX, StartY),
    exit_position(ExitX, ExitY),
    bfs([[([StartX, StartY])]], (ExitX, ExitY), ReversePath),
    reverse(ReversePath, Path).

```

% BFS implementation

```

bfs([[Position|Path]|_], Position, [Position|Path]).  

bfs([CurrentPath|OtherPaths], Exit, Solution) :-  

    CurrentPath = [Position|_],  

    findall([NextPos|CurrentPath],  

        (move_bfs(Position, NextPos),  

         \+ member(NextPos, CurrentPath)),  

        NewPaths),  

    append(OtherPaths, NewPaths, UpdatedPaths),  

    bfs(UpdatedPaths, Exit, Solution).  
  

% Helper for BFS moves  

move_bfs((X, Y), (X1, Y1)) :-  

    move(X, Y, X1, Y1).  
  

% Calculate path length  

path_length(Path, Length) :-  

    length(Path, Length).  
  

% Print the path  

print_path([]).  

print_path([(X, Y)|Rest]) :-  

    format('Position: (~w, ~w)~n', [X, Y]),  

    print_path(Rest).  
  

% Main query examples:  

% ?- solve_maze(Path), print_path(Path).  

% ?- solve_maze_bfs(Path), path_length(Path, Length), print_path(Path).  
  

% Check if position is on the solution path

```

```
on_solution_path(X, Y, Path) :-  
    member((X, Y), Path).  
  
% Count number of possible moves from a position  
count_moves(X, Y, Count) :-  
    findall((X1, Y1), move(X, Y, X1, Y1), Moves),  
    length(Moves, Count).  
  
% Find all dead ends in the maze (positions with only one exit)  
find_dead_ends(DeadEnds) :-  
    findall((X, Y),  
        (valid_position(X, Y),  
         count_moves(X, Y, 1))),  
    DeadEnds).
```