

b) Show the code (complete) and screen shot of prolog code and output

Code:

```
%Reg No:21BCT0402
```

```
% Define the start state
```

```
start_state([  
    [1,2,3],  
    [0,4,6],  
    [7,5,8]  
]).
```

```
% Define the goal state
```

```
goal_state([  
    [1,2,3],  
    [4,5,6],  
    [7,8,0]  
]).
```

```
% Define move operations
```

```
% Move the blank space left
```

```
move_left([Row1, [0, X, Y | Row2], Row3], [Row1, [X, 0, Y | Row2], Row3]).
```

```
% Move the blank space right
```

```
move_right([Row1, [X, Y, 0 | Row2], Row3], [Row1, [X, Y, 0 | Row2], Row3]).
```

% Move the blank space up

move_up([[0, X, Y | Row1], Row2, Row3], [[X, 0, Y | Row1], Row2, Row3]).

% Move the blank space down

move_down([Row1, Row2, [X, 0, Y | Row3]], [Row1, Row2, [X, 0, Y | Row3]]).

% Define valid moves

valid_move(State, NextState) :-

 move_left(State, NextState);

 move_right(State, NextState);

 move_up(State, NextState);

 move_down(State, NextState).

% Define breadth-first search

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

bfs([Path | Paths], State, Solution) :-

 extend(Path, NewPaths),

 append(Paths, NewPaths, Paths1),

 bfs(Paths1, State, Solution).

extend([State | Path], NewPaths) :-

 findall([NextState, State | Path],

 (valid_move(State, NextState),

 \+ member(NextState, Path)),

 NewPaths).

% Define solve predicate

solve(Solution) :-

```
    start_state(InitialState),  
    bfs([[InitialState]], SolutionState, Solution),  
    goal_state(GoalState),  
    reverse(Solution, [GoalState | _]).
```

% Predicate to print solution path

print_solution([]).

print_solution([State | Path]) :-

```
    print_state(State),  
    nl,  
    print_solution(Path).
```

% Predicate to print a state

print_state([]) :- nl.

print_state([Row | Rows]) :-

```
    print_row(Row),  
    print_state(Rows).
```

print_row([]) :- nl.

print_row([X | Xs]) :-

```
    write(X),  
    write(' '),  
    print_row(Xs).
```

main.pl

```
1 %Reg No:21BCT0402
2
3 % Define the start state
4 start_state([
5     [1,2,3],
6     [0,4,6],
7     [7,5,8]
8 ]).
9
10 % Define the goal state
11 goal_state([
12     [1,2,3],
13     [4,5,6],
14     [7,8,0]
15 ]).
16
17 % Define move operations
18
19 % Move the blank space left
20 move_left([Row1, [0, X, Y | Row2], Row3], [Row1, [X, 0, Y | Row2], Row3]).
21
22 % Move the blank space right
23 move_right([Row1, [X, Y, 0 | Row2], Row3], [Row1, [X, Y, 0 | Row2], Row3]).
24
25 % Move the blank space up
26 move_up([[0, X, Y | Row1], Row2, Row3], [[X, 0, Y | Row1], Row2, Row3]).
27
28 % Move the blank space down
29 move_down([Row1, Row2, [X, 0, Y | Row3]], [Row1, Row2, [X, 0, Y | Row3]]).
30
31 % Define valid moves
32 valid_move(State, NextState) :-
33     move_left(State, NextState);
34     move_right(State, NextState);
35     move_up(State, NextState);
36     move_down(State, NextState).
37
38 % Define breadth-first search
39 bfs([[State | Path] | _], State, [State | Path]).
40 bfs([Path | Paths], State, Solution) :-
41     extend(Path, NewPaths),
42     append(Paths, NewPaths, Paths1),
43     bfs(Paths1, State, Solution).
44
45 extend([State | Path], NewPaths) :-
46     findall([NextState, State | Path],
47         (valid_move(State, NextState),
48          \+ member(NextState, Path)),
49         NewPaths).
50
51 % Define solve predicate
52 solve(Solution) :-
53     start_state(InitialState),
54     bfs([[InitialState]], SolutionState, Solution),
55     goal_state(GoalState),
56     reverse(Solution, [GoalState | _]).
57
58 % Predicate to print solution path
59 print_solution([]).
60 print_solution([State | Path]) :-
```

```

61     print_state(State),
62     nl,
63     print_solution(Path).
64
65 % Predicate to print a state
66 print_state([]) :- nl.
67 print_state([Row | Rows]) :-
68     print_row(Row),
69     print_state(Rows).
70 print_row([]) :- nl.
71 print_row([X | Xs]) :-
72     write(X),
73     write(' '),
74     print_row(Xs).

```

Output:

```

:- bfs([[1,2,3],[0,4,6],[7,5,8]], Path), reverse(Path, ReversedPath), print_path(ReversedPath).

right -> down -> left -> up -> up -> right -> right -> down -> down -> left -> up -> left -> up ->

```