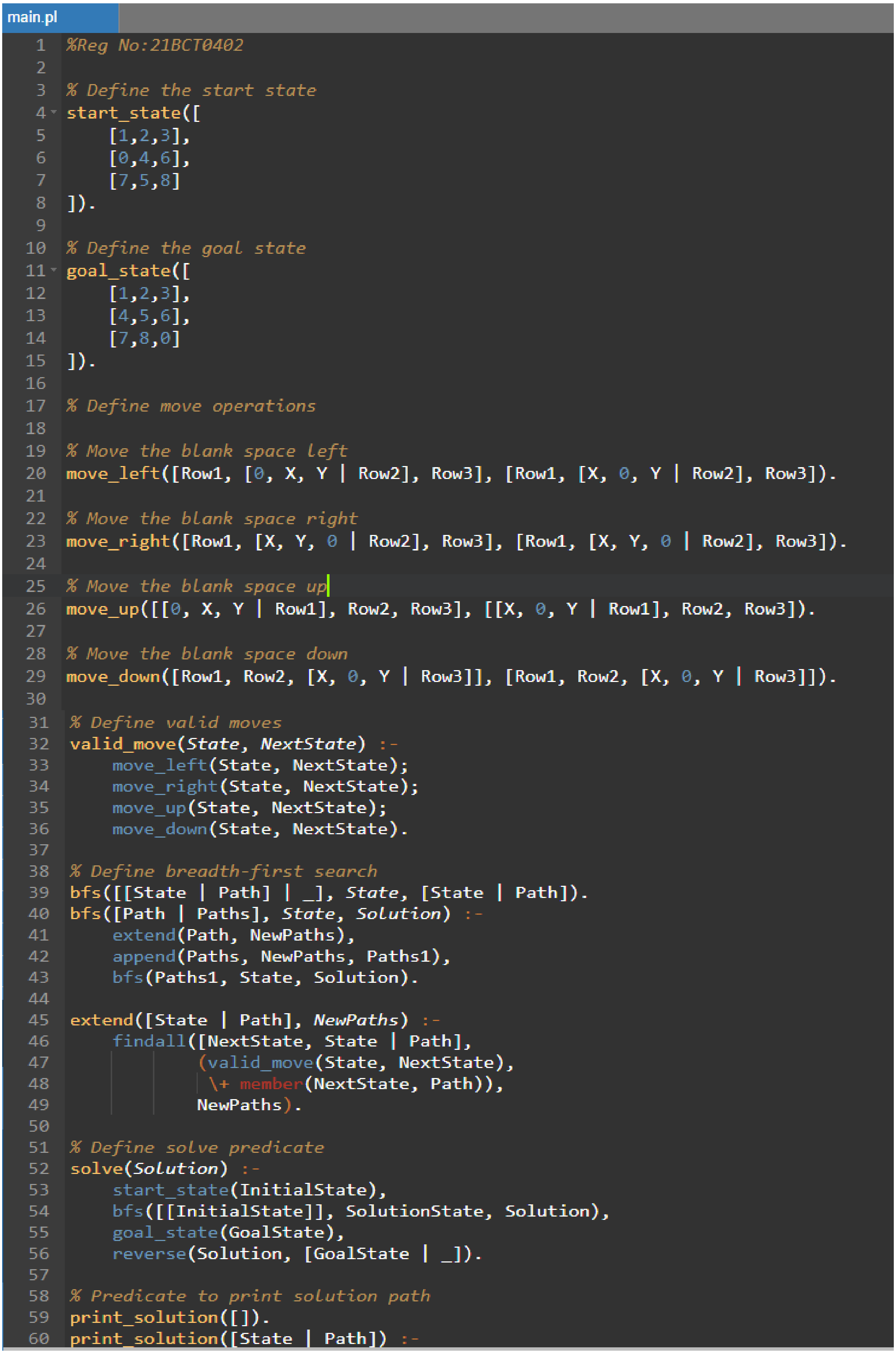
b) Show the code (complete) and screen shot of prolog code and output

Code:

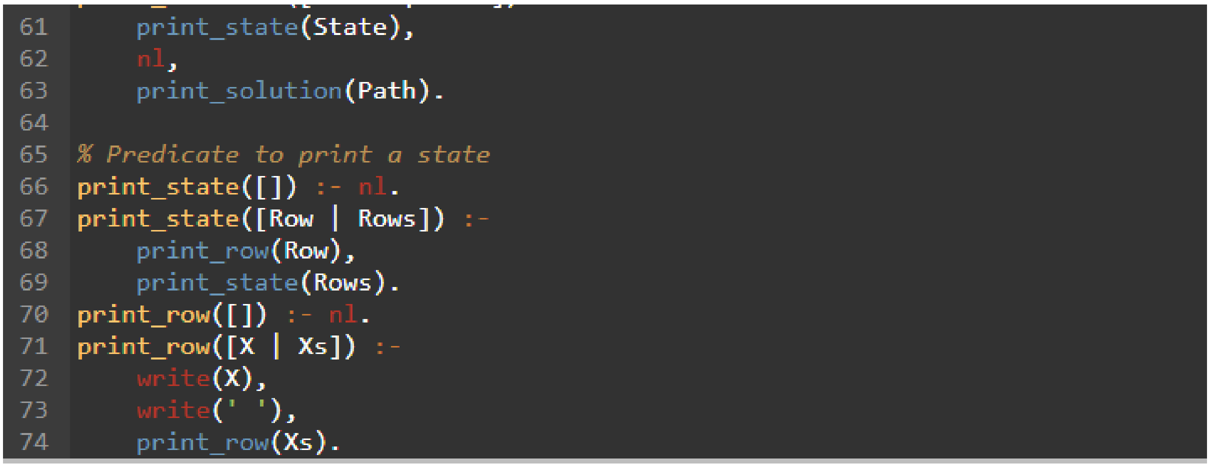
```
%Reg No:21BCT0402

% 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).
```

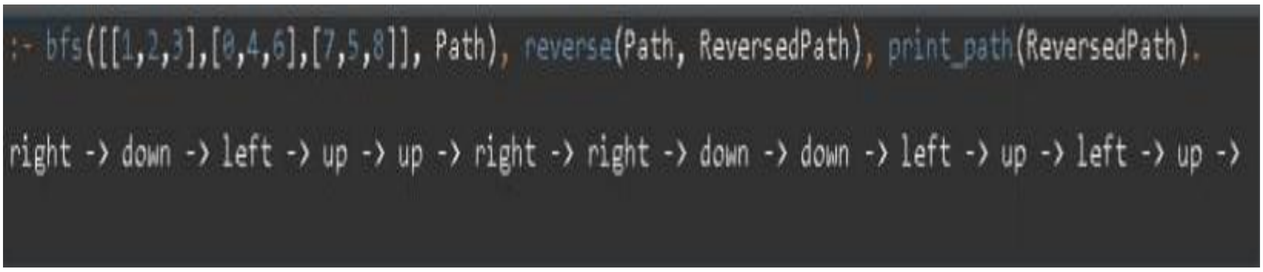


```prolog
%Reg No:21BCT0402

% 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]) :-
```



```prolog
    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).
```

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 ->
```