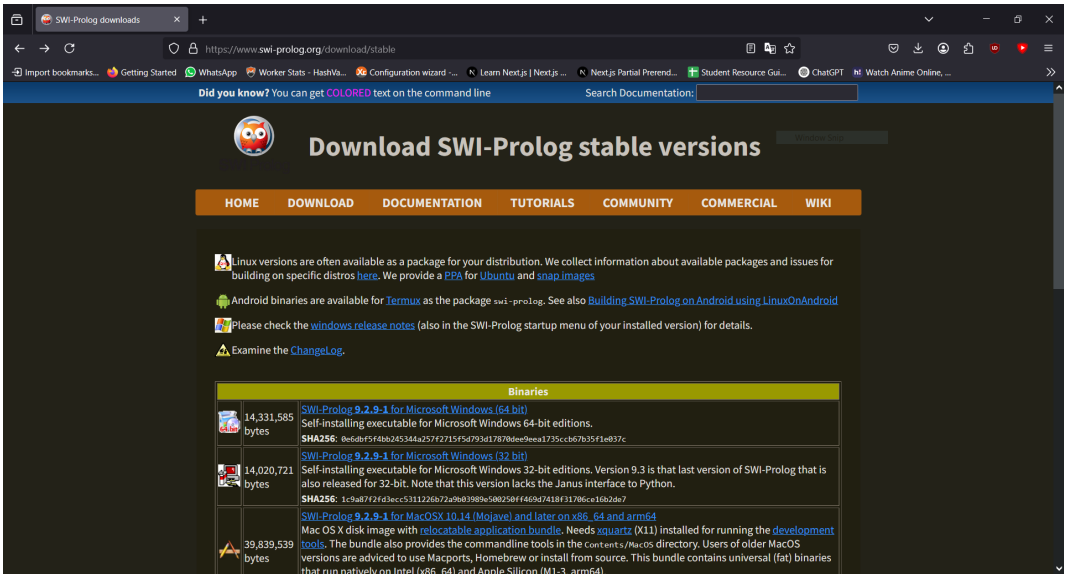
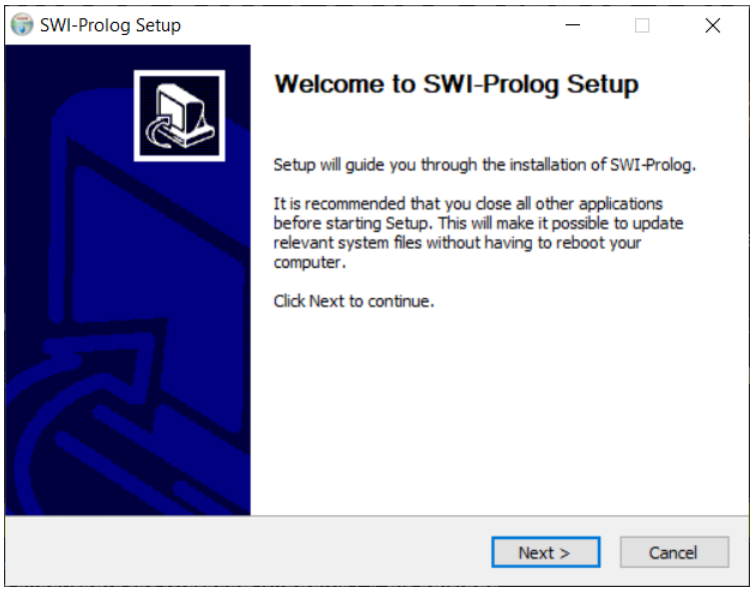
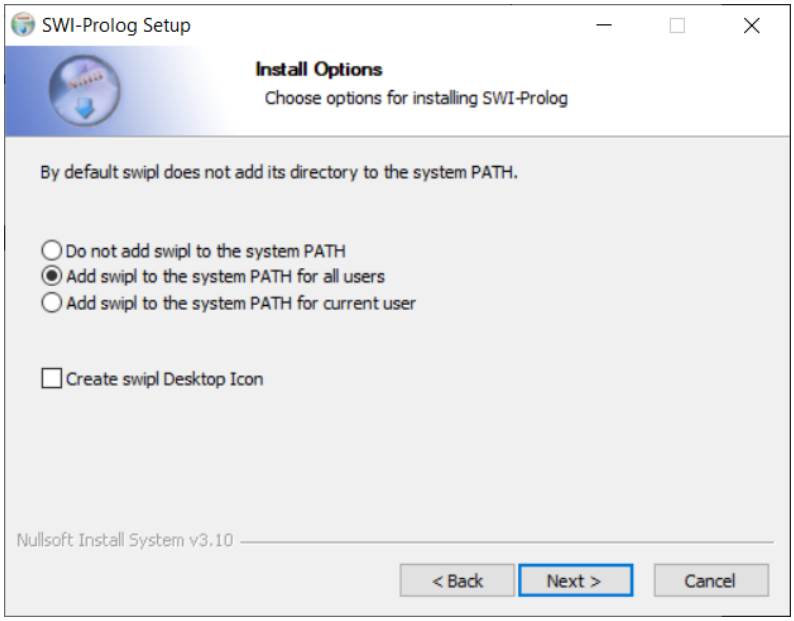
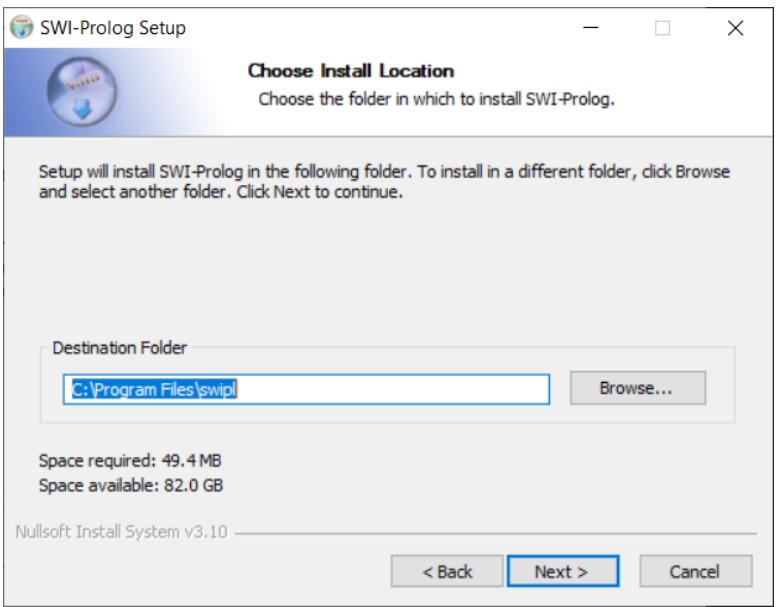
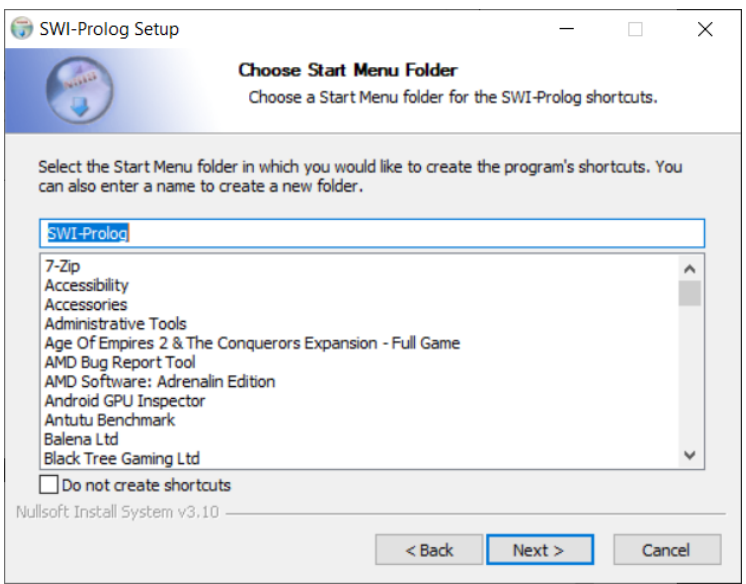
Experiment-1

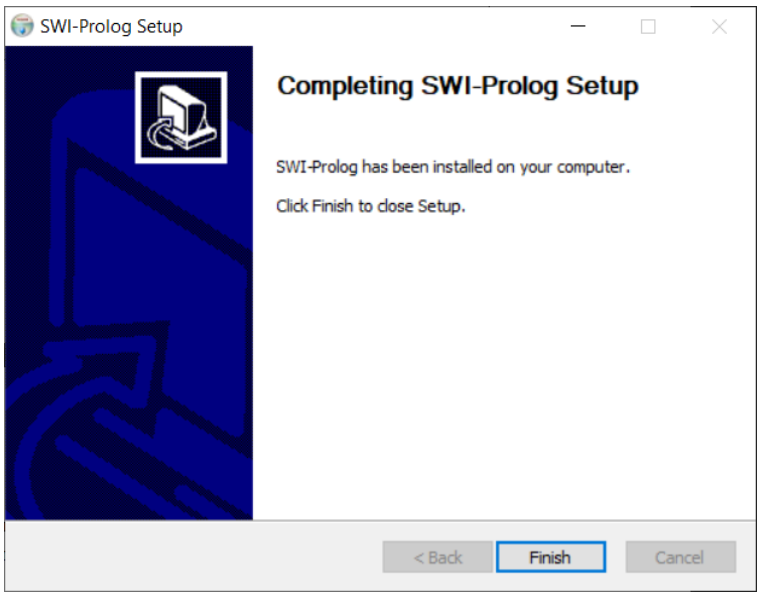


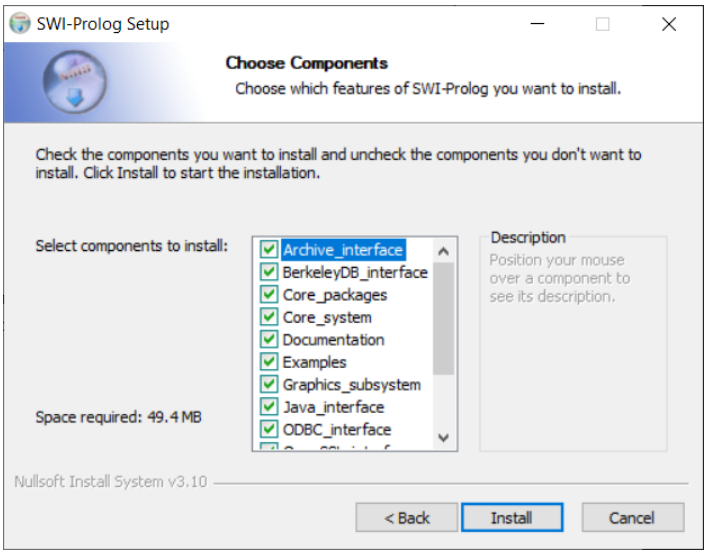


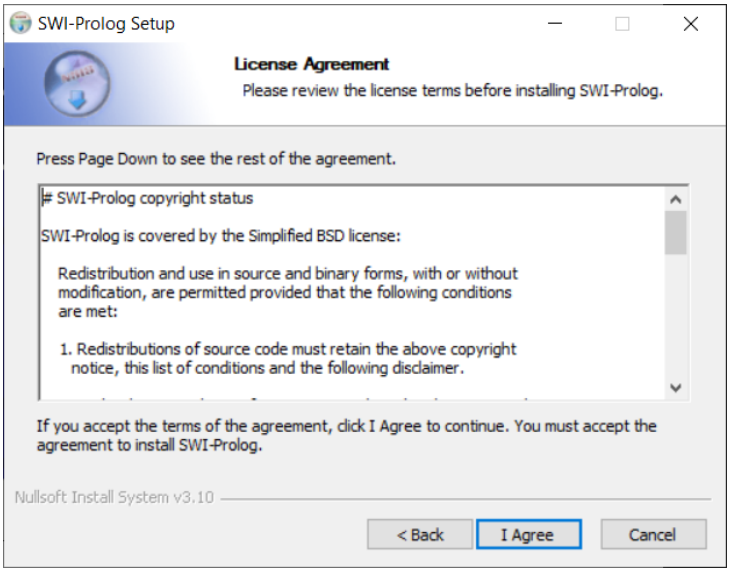


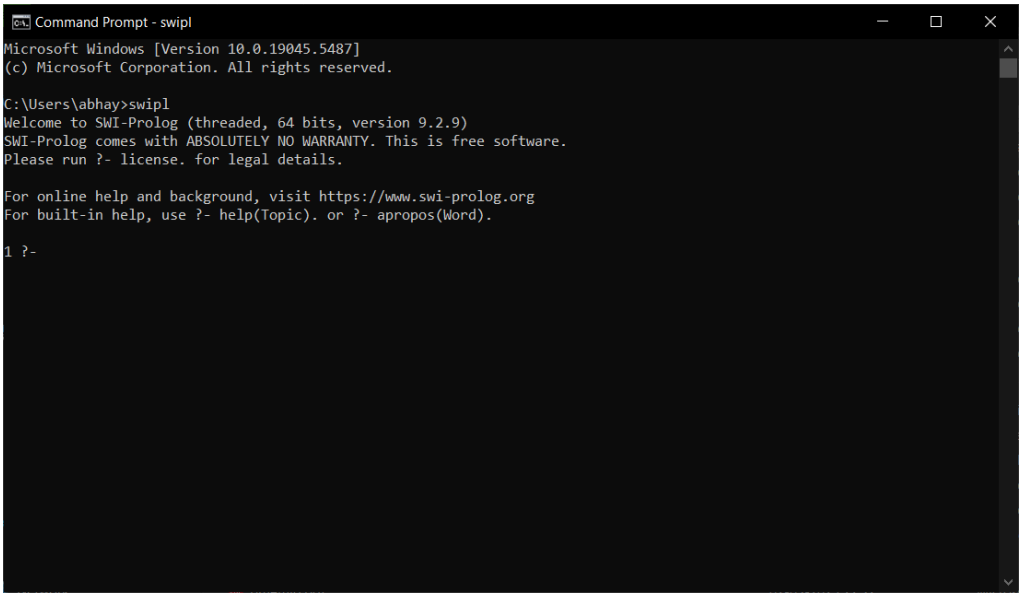





SWI-Prolog Setup
Completing SWI-Prolog Setup
SWI-Prolog has been installed on your computer.
Click Finish to close Setup.
< Back
Finish
Cancel


SWI-Prolog Setup
Choose Components
Choose which features of SWI-Prolog you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Install to start the installation.
Select components to install:
Archive_interface
BerkeleyDB_interface
Core_packages
Core_system
Documentation
Examples
Graphics_subsystem
Java_interface
ODBC_interface
Description
Position your mouse over a component to see its description.
Space required: 49.4 MB
Nullsoft Install System v3.10
< Back
Install
Cancel


SWI-Prolog Setup
License Agreement
Please review the license terms before installing SWI-Prolog.
Press Page Down to see the rest of the agreement.
SWI-Prolog copyright status
SWI-Prolog is covered by the Simplified BSD license:
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install SWI-Prolog.
Nullsoft Install System v3.10
< Back
I Agree
Cancel


Command Prompt - swipl
Microsoft Windows [Version 10.0.19045.5487]
(c) Microsoft Corporation. All rights reserved.
C:\Users\abhay>swipl
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).
1 ?-

Experiment – 2

Code:-

```
square(Number, Result) :-
  Result is Number * Number.
area_circle(Radius, Area) :-
  Area is 3.14159 * Radius * Radius.
area_square(Side, Area) :-
  Area is Side * Side.
area_rectangle(Length, Width, Area) :-
  Area is Length * Width.
simple_interest(Principal, Rate, Time, Interest) :-
  Interest is (Principal * Rate * Time) / 100.
```

```
?- sqaure(4,16).
Correct to: "square(4,16)"? yes
true.

?- area_circle(1,1)
|    .
false.

?- area_rectangle(1,1,1).
true.

?- simple_interest(1, 1, 1, 1).
false.
```

Experiment – 3

Code:-

```
even(Number) :-
  Number mod 2 =:= 0.
odd(Number) :-
  Number mod 2 =\= 0.
max(X, Y, X) :-
  X >= Y.
max(X, Y, Y) :-
  Y > X.
grade(Marks, 'A Grade') :-
  Marks >= 90.
grade(Marks, 'B Grade') :-
  Marks >= 75,
  Marks < 90.
grade(Marks, 'C Grade') :-
  Marks >= 50,
  Marks < 75.
grade(Marks, 'Fail') :-
  Marks < 50.
```

```
?- even(5).
false.

?- odd(3).
true.

?- grade(55,'A Grade').
false.
```

Output:-

Experiment – 4

Code:-

```
% Facts
likes(ram, mango).
girl(seema).
likes(bill, cindy).
color(rose, red).
owns(john, gold).
```

Output:-

```
?- likes(ram, mango).
true.

?- girl(seema).
true.

?- likes(bill, cindy).
true.

?- color(rose, red).
true.

?- owns(john, gold).
true.

?-
```

Experiment – 5

```
c_to_f(C, F) :-
  F is (C * 9 / 5) + 32.
below_freezing(C) :-
  C < 0.
```

```
?- c_to_f(0,32).
true.

?- below_freezing(9).
false.

?-
```

Output:-

Experiment – 6

```
edge(a, b).
edge(b, c).
edge(c, d).
edge(d, a).
edge(b, e).

show_edges :-
  write('Edges of the graph:'), nl,
  edge(X, Y),
  write(X), write(' -> '), write(Y), nl,
  fail.
show_edges.

show_neighbors(Node) :-
  write('Neighbors of '), write(Node), write(':'), nl,
  edge(Node, Neighbor),
  write(Neighbor), nl,
  fail.
show_neighbors(_).

show_graph :-
  show_edges, nl,
  show_neighbors(a),
  show_neighbors(b),
  show_neighbors(c),
  show_neighbors(d).

dfs(Node, Visited) :-
  write('Visiting node: '), write(Node), nl,
  edge(Node, Neighbor),
  not(member(Neighbor, Visited)),
  dfs(Neighbor, [Node | Visited]).
dfs(_, _).

start_dfs(Node) :-
  write('Start DFS from node: '), write(Node), nl,
  dfs(Node, []).
```

```
?- start_dfs(a)
Start DFS from node: a
Visiting node: a
Visiting node: b
Visiting node: c
Visiting node: d
true
```

Experiment – 7

Output:-

```
?- move(0,0,[(0,0)]).
fill 4 jug
fill 3 jug
pour 4 jug
pour 3 jug
fill 4 jug
pour from 4jug to 3jug
pour 3 jug
pour from 4jug to 3jug
fill 4 jug
pour from 4jug to 3jug
pour 3 jug
done
true
```

Code:-

```
member(X,[X|_]).
member(X,[Y|Z]):-member(X,Z).

move(X,Y,_):-X=:=2,Y=:=0,write('done'),!.
move(X,Y,Z):-X<4,\+member((4,Y),Z),write("fill 4 jug"),nl,move(4,Y,[(4,Y)|Z]).
move(X,Y,Z):-Y<3,\+member((X,3),Z),write("fill 3 jug"),nl,move(X,3,[(X,3)|z]).
move(X,Y,Z):-X>0,\+member((0,Y),Z),write("pour 4 jug"),nl,move(0,Y,[(0,Y)|Z]).
move(X,Y,Z):-Y>0,\+member((X,0),Z),write("pour 3 jug"),nl,move(X,0,[(X,0)|Z]).
move(X,Y,Z):-P is X+Y,P>=4,Y>0,K is 4-X,M is Y-K,\+member((4,M),Z),write("pour from 3jug to
4jug"),nl,move(4,M,[(4,M)|Z]).
move(X,Y,Z):-P is X+Y,P>=3,X>0,K is 3-Y,M is X-K,\+member((M,3),Z),write("pour from 4jug to
3jug"),nl,move(M,3,[(M,3)|Z]).
move(X,Y,Z):-K is X+Y,K<4,Y>0,\+member((K,0),Z),write("pour from 3jug to 4jug"),nl,move(K,0,[(K,0)|Z]).
move(X,Y,Z):-K is X+Y,K<3,X>0,\+member((0,K),Z),write("pour from 4jug to 3jug"),nl,move(0,K,[(0,K)|Z]).
```

Experiment – 8 Code:-

```
test(Plan):-
  write('Initial state:'),nl,
  Init= [at(tile4,1), at(tile3,2), at(tile8,3),
at(empty,4), at(tile2,5), at(tile6,6), at(tile5,7), at(tile1,8), at(tile7,9)],
  write_sol(Init),
  Goal= [at(tile1,1), at(tile2,2), at(tile3,3),
at(tile4,4), at(empty,5), at(tile5,6), at(tile6,7), at(tile7,8), at(tile8,9)],
  nl,write('Goal state:'),nl,
  write(Goal),nl,nl,
  solve(Init,Goal,Plan).

solve(State, Goal, Plan):-
  solve(State, Goal, [], Plan).

is_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1 - Y1).

solve(State, Goal, Plan, Plan):-
  is_subset(Goal, State), nl,
  write_sol(Plan).

solve(State, Goal, Sofar, Plan):-
  act(Action, Preconditions, Delete, Add),
  is_subset(Preconditions, State),
  \+ member(Action, Sofar),
  delete_list(Delete, State, Remainder),
  append(Add, Remainder, NewState),
  solve(NewState, Goal, [Action|Sofar], Plan).

act(move(X,Y,Z),
  [at(X,Y), at(empty,Z), is_movable(Y,Z)],
  [at(X,Y), at(empty,Z)],
  [at(X,Z), at(empty,Y)]).
is_subset([H|T], Set):-
  member(H, Set),
  is_subset(T, Set).
is_subset([], _).
delete_list([H|T], Curstate, Newstate):-
  remove(H, Curstate, Remainder),
  delete_list(T, Remainder, Newstate).
delete_list([], Curstate, Curstate).
remove(X, [X|T], T).
remove(X, [H|T], [H|R]):-
  remove(X, T, R).
write_sol([]).
write_sol([H|T]):-
  write_sol(T),
  write(H), nl.
append([H|T], L1, [H|L2]):-
  append(T, L1, L2).
append([], L, L).
member(X, [X|_]).
member(X, [_|T]):-
  member(X, T).
```

Output:-

```
?- test(Plan).
Initial state:
at(tile7,9)
at(tile1,8)
at(tile5,7)
at(tile6,6)
at(tile2,5)
at(empty,4)
at(tile8,3)
at(tile3,2)
at(tile4,1)

Goal state:
[at(tile1,1),at(tile2,2),at(tile3,3),at(tile4,4),at(empty,5),at(tile5,6),at(tile6,7),at(tile7,8),at(tile8,9)]

false.
```

Experiment – 9

Aim:- Write a program to implement a Tic-Tac-Toe game using Prolog.

Code:-

```
win(b, p) :- rowwin(b, p).
win(b, p) :- colwin(b, p).
win(b, p) :- diagwin(b, p).

rowwin(b, p) :- b = [p,p,p,_,_,_,_,_,_].
rowwin(b, p) :- b = [_,_,_,p,p,p,_,_,_].
rowwin(b, p) :- b = [_,_,_,_,_,_,p,p,p].

colwin(b, p) :- b = [p,_,_,p,_,_,p,_,_].
colwin(b, p) :- b = [_,p,_,_,p,_,_,p,_].
colwin(b, p) :- b = [_,_,p,_,_,p,_,_,p].

diagwin(b, p) :- b = [p,_,_,_,p,_,_,_,p].
diagwin(b, p) :- b = [_,_,p,_,p,_,p,_,_].

other(x,o).
other(o,x).

game(b, p) :- win(b, p), !, write([p, p, wins]).
game(b, p) :-
 other(p,Otherp),
 move(b,p,Newb),
 !,
 display(Newb),
 game(Newb,Otherp).

move([b,B,C,D,E,F,G,H,I], p, [p,B,C,D,E,F,G,H,I]).
move([A,b,C,D,E,F,G,H,I], p, [A,p,C,D,E,F,G,H,I]).
move([A,B,b,D,E,F,G,H,I], p, [A,B,p,D,E,F,G,H,I]).
move([A,B,C,b,E,F,G,H,I], p, [A,B,C,p,E,F,G,H,I]).
move([A,B,C,D,b,F,G,H,I], p, [A,B,C,D,p,F,G,H,I]).
move([A,B,C,D,E,b,G,H,I], p, [A,B,C,D,E,p,G,H,I]).
move([A,B,C,D,E,F,b,H,I], p, [A,B,C,D,E,F,p,H,I]).
move([A,B,C,D,E,F,G,b,I], p, [A,B,C,D,E,F,G,p,I]).
move([A,B,C,D,E,F,G,H,b], p, [A,B,C,D,E,F,G,H,p]).

display([A,B,C,D,E,F,G,H,I]) :- write([A,B,C]),nl,write([D,E,F]),nl,
 write([G,H,I]),nl,nl.

selfgame :- game([b,b,b,b,b,b,b,b,b],x).

x_can_win_in_one(b) :- move(b, x, Newb), win(Newb, x).

orespond(b,Newb) :-
 move(b, o, Newb),
 win(Newb, o),
 !.
orespond(b,Newb) :-
 move(b, o, Newb),
 not(x_can_win_in_one(Newb)).
orespond(b,Newb) :-
 move(b, o, Newb).
orespond(b,Newb) :-
 not(member(b,b)),
 !,
 write('Cats game!'), nl,
 Newb = b.

xmove([b,B,C,D,E,F,G,H,I], 1, [x,B,C,D,E,F,G,H,I]).
xmove([A,b,C,D,E,F,G,H,I], 2, [A,x,C,D,E,F,G,H,I]).
xmove([A,B,b,D,E,F,G,H,I], 3, [A,B,x,D,E,F,G,H,I]).
xmove([A,B,C,b,E,F,G,H,I], 4, [A,B,C,x,E,F,G,H,I]).
xmove([A,B,C,D,b,F,G,H,I], 5, [A,B,C,D,x,F,G,H,I]).
xmove([A,B,C,D,E,b,G,H,I], 6, [A,B,C,D,E,x,G,H,I]).
xmove([A,B,C,D,E,F,b,H,I], 7, [A,B,C,D,E,F,x,H,I]).
xmove([A,B,C,D,E,F,G,b,I], 8, [A,B,C,D,E,F,G,x,I]).
xmove([A,B,C,D,E,F,G,H,b], 9, [A,B,C,D,E,F,G,H,x]).
xmove(b, _, b) :- write('Illegal move.'), nl.

playo :- explain, playfrom([b,b,b,b,b,b,b,b,b]).
```

```
explain :-
 write('You play X by entering integer positions followed by a period.'),
 nl,
 display([1,2,3,4,5,6,7,8,9]).

playfrom(b) :- win(b, x), write('You win!').
playfrom(b) :- win(b, o), write('I win!').
playfrom(b) :- read(N),
 xmove(b, N, Newb),
 display(Newb),
 orespond(Newb, Newnewb),
 display(Newnewb),
 playfrom(Newnewb).
```

Output:-

```
?- playo.
You play X by entering integer positions followed by a period.
[1,2,3]
[4,5,6]
[7,8,9]

|: 1.
[x,b,b]
[b,b,b]
[b,b,b]

[x,o,b]
[b,b,b]
[b,b,b]

|: 4.
[x,o,b]
[x,b,b]
[b,b,b]

[x,o,b]
[x,b,b]
[o,b,b]

|: 9.
[x,o,b]
[x,b,b]
[o,b,x]

[x,o,b]
[x,o,b]
[o,b,x]

|: 8.
[x,o,b]
[x,o,b]
[o,x,x]

[x,o,o]
[x,o,b]
[o,x,x]

I win!
true .
```