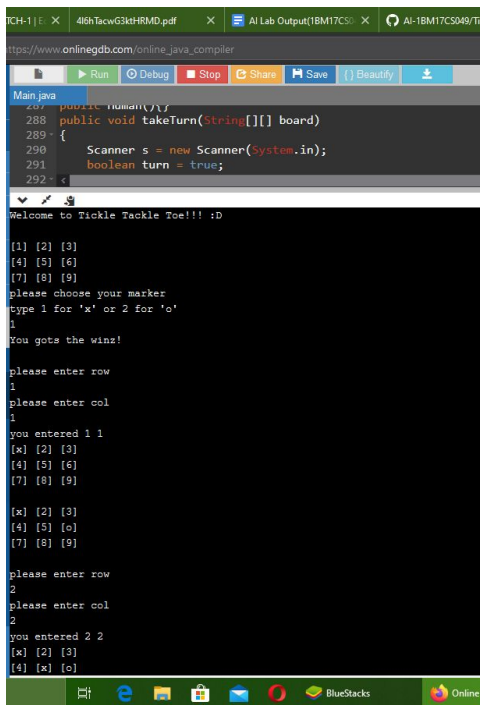


AI LAB PROGRAMS

OUTPUT:

1. Tic Tac Toe:



```
288 public void takeTurn(String[][] board)
289 {
290     Scanner s = new Scanner(System.in);
291     boolean turn = true;
292 }
```

Welcome to Tackle Tackle Toe!!! :D

[1] [2] [3]
[4] [5] [6]
[7] [8] [9]

please choose your marker
type 1 for 'x' or 2 for 'o'
1

You got the wins!

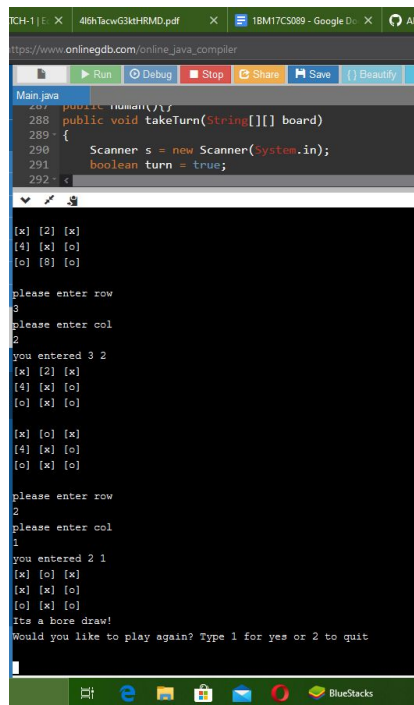
please enter row
1
please enter col
1
you entered 1 1

[x] [2] [3]
[4] [5] [6]
[7] [8] [9]

[x] [2] [3]
[4] [5] [o]
[7] [8] [9]

please enter row
2
please enter col
2
you entered 2 2

[x] [2] [3]
[4] [x] [o]



```
288 public void takeTurn(String[][] board)
289 {
290     Scanner s = new Scanner(System.in);
291     boolean turn = true;
292 }
```

[x] [2] [x]
[4] [x] [o]
[o] [8] [o]

please enter row
3
please enter col
2
you entered 3 2

[x] [2] [x]
[4] [x] [o]
[o] [x] [o]

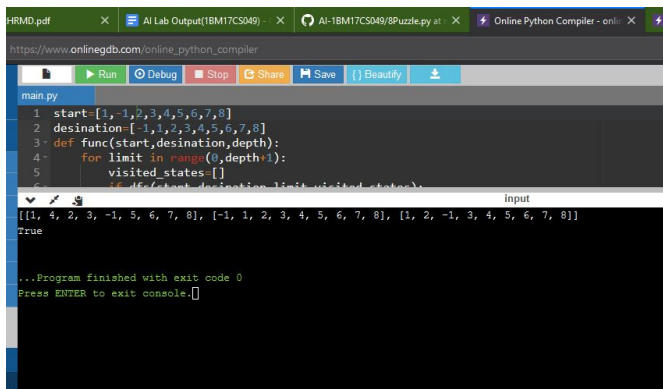
[x] [o] [x]
[4] [x] [o]
[o] [x] [o]

please enter row
2
please enter col
1
you entered 2 1

[x] [o] [x]
[x] [x] [o]
[o] [x] [o]

its a bore draw!
Would you like to play again? Type 1 for yes or 2 to quit

2. Solving 8 Puzzle problem:



```
1 start=[1,-1,2,3,4,5,6,7,8]
2 desination=[-1,1,2,3,4,5,6,7,8]
3 def func(start,desination,depth):
4     for limit in range(0,depth+1):
5         visited_states=[]
6         if (start,desination,limit) not in visited_states:
```

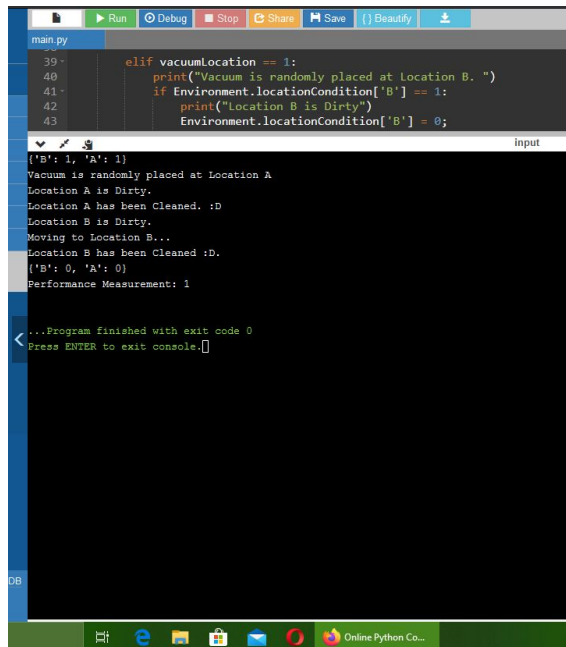
Input

[[1, 4, 2, 3, -1, 5, 6, 7, 8], [-1, 1, 2, 3, 4, 5, 6, 7, 8], [1, 2, -1, 3, 4, 5, 6, 7, 8]]

True

...Program finished with exit code 0
Press ENTER to exit console.

3. Vacuum Cleaner Agent:



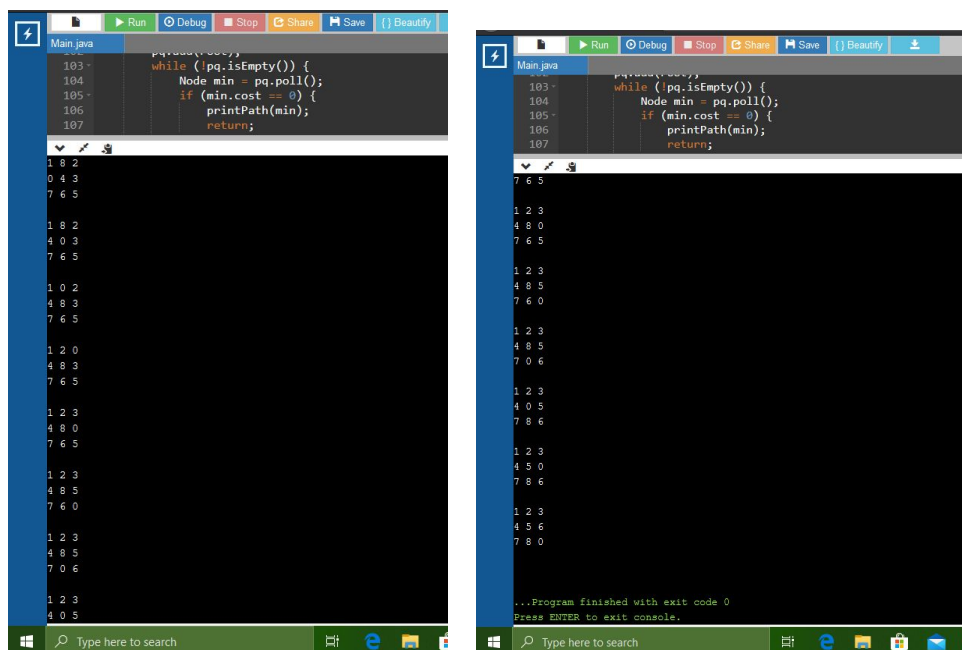
```
main.py
39 elif vacuumLocation == 1:
40     print("Vacuum is randomly placed at Location B. ")
41     if Environment.locationCondition['B'] == 1:
42         print("Location B is Dirty")
43         Environment.locationCondition['B'] = 0;
```

Input

```
{'B': 1, 'A': 1}
Vacuum is randomly placed at Location A
Location A is Dirty.
Location A has been Cleaned. :D
Location B is Dirty.
Moving to Location B...
Location B has been Cleaned :D.
{'B': 0, 'A': 0}
Performance Measurement: 1

...Program finished with exit code 0
Press ENTER to exit console.
```

4. A* search(8 puzzle problem):



```
Main.java
103 while (!pq.isEmpty()) {
104     Node min = pq.poll();
105     if (min.cost == 0) {
106         printPath(min);
107         return;
108     }
```

Output (Left):

```
1 8 2
0 4 3
7 6 5

1 8 2
4 0 3
7 6 5

1 0 2
4 8 3
7 6 5

1 2 0
4 8 3
7 6 5

1 2 3
4 8 0
7 6 5

1 2 3
4 8 5
7 0 6

1 2 3
4 0 5
```

Output (Right):

```
7 6 5

1 2 3
4 8 0
7 6 5

1 2 3
4 8 5
7 6 0

1 2 3
4 8 5
7 0 6

1 2 3
4 0 5
7 8 6

1 2 3
4 5 0
7 8 6

1 2 3
4 5 6
7 8 0

...Program finished with exit code 0
Press ENTER to exit console.
```

5. 8 Puzzle problem(iterative deepening search):

```
main.py
104 def IDS(se1r):
105
106     def DLS(currentNode, depth):
107         if depth == 0:
108             return None
109         if currentNode.isSolved:
```

```
False
False
7 2 4
5 0 6
8 3 1
7 0 4
5 2 6
8 3 1
0 7 4
5 2 6
8 3 1
5 7 4
0 2 6
8 3 1
5 7 4
2 0 6
8 3 1
5 0 4
2 7 6
8 3 1
0 5 4
2 7 6
8 3 1
```

```
main.py
104 def IDS(se1r):
105
106     def DLS(currentNode, depth):
107         if depth == 0:
108             return None
109         if currentNode.isSolved:
```

```
6 3 5
7 8 0
1 4 2
6 3 5
7 0 8
1 4 2
6 3 5
0 7 8
1 4 2
0 3 5
6 7 8
1 4 2
3 0 5
6 7 8
1 0 2
3 4 5
6 7 8
0 1 2
3 4 5
6 7 8

Total number of moves: 42
```

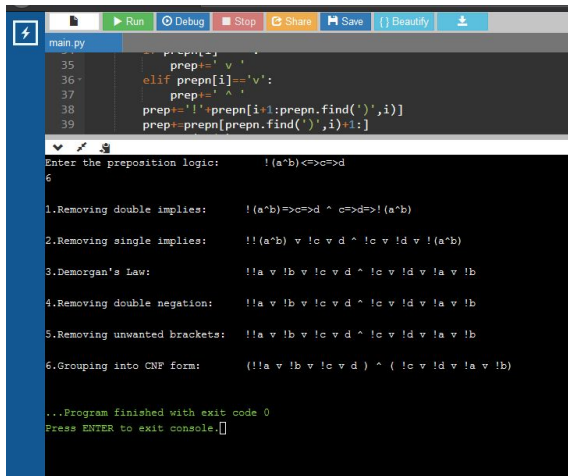
6. Query entailing Knowledge-base:

```
main.py
1 combinations=[(True,True),(False,False),(False,True),(True,False)]#expand this
2 variable={'p':0,'q':1}
3 kb=''
4 q=''
5 priority={'p':3,'q':1,'^':2}
6 def entail_rule(A):

Enter rule : p^q~p
Enter the Query : p^q
*****Truth Table Reference*****
kb alpha
*****
False True
-----
False False
-----
False False
-----
False False
-----
The Knowledge Base entails query

...Program finished with exit code 0
Press ENTER to exit console.
```

7. Conjunctive Normal Form(CNF):



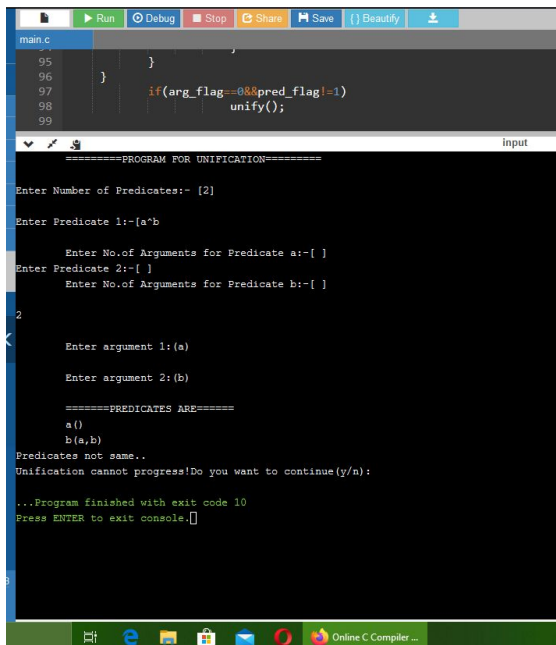
```
main.py
35     prep+=' v '
36     elif prep[i]!='v':
37         prep+=' ^ '
38     prep+='!' +prep[i+1:prep.find(')'),i]]
39     prep=prep[prep.find(')'),i+1:]

Enter the preposition logic:  !(a*b)<=>c>d
6

1.Removing double implies:  !(a*b)=>c=>d ^ c=>d=>!(a*b)
2.Removing single implies:  !!(a*b) v !c v d ^ !c v !d v !(a*b)
3.Demorgan's Law:  !!a v !b v !c v d ^ !c v !d v !a v !b
4.Removing double negation:  !!a v !b v !c v d ^ !c v !d v !a v !b
5.Removing unwanted brackets:  !!a v !b v !c v d ^ !c v !d v !a v !b
6.Grouping into CNF form:  (!!a v !b v !c v d) ^ ( !c v !d v !a v !b)

...Program finished with exit code 0
Press ENTER to exit console.
```

8. Unification in first order logic:



```
main.c
95     }
96 }
97 if(arg_flag==0&&pred_flag!=1)
98     unify();
99

input

=====PROGRAM FOR UNIFICATION=====

Enter Number of Predicates:- [2]

Enter Predicate 1:-[a*b

    Enter No.of Arguments for Predicate a:-[ ]
Enter Predicate 2:-[ ]
    Enter No.of Arguments for Predicate b:-[ ]

2

    Enter argument 1:(a)

    Enter argument 2:(b)

=====PREDICATES ARE=====
a()
b(a,b)
Predicates not same..
Unification cannot progress!Do you want to continue(y/n):

...Program finished with exit code 10
Press ENTER to exit console.
```

9. Prove first order logic query using forward reasoning

```
Enter rule :p^~q
Enter the Query : p
*****Truth Table Reference*****
kb alpha
*****
False True
-----
False False
-----
False False
-----
True True
-----
Proven using forward reasoning
```

10. Decision tree learning:

```
Confusion Matrix: [[ 0  6  7]
                   [ 0 67 18]
                   [ 0 19 71]]
Accuracy : 73.4042553191
Report :
      precision    recall  f1-score   support
B       0.00        0.00        0.00         13
L       0.73        0.79        0.76         85
R       0.74        0.79        0.76         90
avg/total 0.68        0.73        0.71        188

Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[ 0  6  7]
                   [ 0 63 22]
                   [ 0 20 70]]
Accuracy : 70.7446808511
Report :
      precision    recall  f1-score   support
B       0.00        0.00        0.00         13
L       0.71        0.74        0.72         85
R       0.71        0.78        0.74         90
avg / total 0.66        0.71        0.68        188
```