



Department of Computer Engineering

Vision:

To be recognized globally as a department provides quality technical education that eventually caters to helping and serving the community.

Mission:

- To develop human resources with sound knowledge in theory and practice of computer science and engineering.
- To motivate the students to solve real-world problems to help the society grow.
- To provide a learning ambience to enhance innovations, team spirit and leadership qualities for students.



LAB MANUAL
(CSL 605)
Artificial Intelligence (AI)
Computer Engineering

Name: **VIVEK.SHIVAKUMAR. HOTTI**

Class: **T. E-Computer Engineering**

Roll No.: **37**

Batch: **A2**

Semester: **VI**

Faculty Name: **Mr. Anas Dange**

2021 – 2022

Mr Anas Dange



Lab Code	Lab Name	Credits
CSL604	Artificial Intelligence Lab	1

Pre-requisites: Discrete Mathematics, Data Structure Lab

Objectives:

1. To realize the basic techniques to build intelligent systems
2. To apply appropriate search techniques used in problem solving
3. To create knowledge base for uncertain data

Lab Outcomes:

At the end of the course, the students will be able to

1. Identify languages and technologies for Artificial Intelligence
2. Understand and implement uninformed and informed searching techniques for real world problems.
3. Create a knowledge base using any AI language.
4. Design and implement expert systems for real world problems.

Term work:

1. Term work should consist of a minimum of 8 experiments.
2. Journal must include at least 2 assignments on content of theory and practical of “Artificial Intelligence”
3. The final certification and acceptance of term work ensures that satisfactory performance of laboratory work and minimum passing marks in term work.
4. Total 25 Marks (Experiments: 15-marks, Attendance Theory & Practical: 05-marks, Assignments: 05-marks)



INDEX

Name: Vivek. Hotti

Roll: 37

Class: TE- Computers

Division: A

Semester: 6

Sr No.	Experiment No.	Experiment Name	Page From	Page To
1.	EXPERIMENT 01	Application of Artificial Intelligence.	5	8
2.	EXPERIMENT 02	Program to implement Lexical Analyzer for C code.	9	12
3.	EXPERIMENT 03	BFS Search Algorithm.	13	16
4.	EXPERIMENT 04	DFS Search Algorithm.	17	19
5.	EXPERIMENT 05	A* Search Algorithm.	20	23
6.	EXPERIMENT 06	Minimax Algorithm for Game playing.	24	26
7.	EXPERIMENT 07	Minimax Algorithm for Tic Tac Toe Game.	27	32
8.	EXPERIMENT 08	Case Study on Prolog Software.	33	35
9.	EXPERIMENT 09	Case Study on AI Planning Projects in IBM.	36	38
10.	EXPERIMENT 10	Case Study on Implementation of BBN.	39	42



EXPERIMENT – 01



Name :	Vivek.Shivakumar. Hotti
Roll Number :	37
Division :	TE-A-Comps Dept
Email :	vivek.hotti@universal.edu.in
Subject :	Artificial Intelligence (AI)

EXPERIMENT – 01

AIM: Case study on AI Applications published in IEEE or Springer Journal.

CASE STUDY (Theory):

Research Paper Details:

- **Domain :** Use of Artificial Intelligence in Autonomous Vehicles.
- **Title :** Artificial Intelligence Applications in the Development of Autonomous Vehicles: A Survey.
- **Published In :** IEEE/CAA JOURNAL OF AUTOMATICA SINICA, VOL. 7, NO. 2.
- **Published On :** March 2020.
- **Authors :** Yifang Ma, Zhenyu Wang, Hong Yang, and Lin Yang.
- **Link :** <https://iee-jas.net/fileZDIXHEN-journal/article/zdixhywb/2020/2/PDF/JAS-2019-0401.pdf>

Case Study:

The advancement of computer science (AI) has truly stirred the event and readying of autonomous vehicles (AVs) within the transportation business. oxyacetylene by huge information from numerous sensing devices and advanced computing resources, AI has become an important part of AVs for perceiving the surrounding surroundings and creating acceptable call in motion. to attain goal of full automation (i.e., self-driving), it is important to grasp however AI works in Av systems. Existing analysis have created nice efforts in investigation completely different aspects of applying AI in Av



Vidya Vikas Education Trust's
Universal College of Engineering

(Permanently unaided | Approved by AICTE, DTE & Affiliated to University of Mumbai)

development. However, few studies have offered the analysis community an intensive examination of current practices in implementing AI in AVs.

Thus, after reading this paper, what I perceive is that this paper aims to shorten the gap by providing a comprehensive survey of key studies during this analysis avenue.

Specifically, it intends to investigate their use of AIs in supporting the first applications in AVs:

- 1) perception;
- 2) localization and mapping; and
- 3) deciding.

This paper truly investigates these practices to grasp the concept of AI, however AI is often used and what square measure the challenges and problems related to their implementation. supported the exploration of current practices and technology advances, this paper additional provides insights into potential opportunities concerning the employment of AI in conjunction with alternative rising technologies like: H-D maps, big data, high-performance computing, increased reality (AR)/virtual reality (VR) increased simulation platform, 5G communication for connected AVs.

The paper starts by giving a brief introduction of the current AI market. It states that many AV technologies in laboratory tests, closed-track tests, and public road tests have witnessed considerable advancements in bringing AVs into real-world applications. Further it goes on to state that these advancements have been greatly benefited from significant investments and promotions by numerous stakeholders such as transportation agencies, information technology (IT) giants (e.g., Google, Baidu, etc.), transportation networking companies (e.g., Uber, DiDi, etc.), automobile manufacturers (e.g., Tesla, General Motors, Volvo, etc.), chip/semiconductor makers (e.g., Intel, Nvidia, Qualcomm, etc.), and so forth.

The paper then goes forth to shed light on prevailing wisdom of earlier (semi-) automated driving concept that was highly related to advanced driver assistance systems (ADAS) that assisted drivers in the driving process. These systems are more related to applications such as lane departure warning, and blind spot



Vidya Vikas Education Trust's
Universal College of Engineering

(Permanently unaided | Approved by AICTE, DTE & Affiliated to University of Mumbai)

alerting. They aimed to automate, adapt, and improve some of the vehicle systems for enhanced safety by reducing errors associated with human drivers.

Further on, the paper goes on to comment that AI approaches can provide promising solutions for AVs in recognizing the environment and propelling the vehicle with appropriate decision making. A few studies have specifically reviewed the applications of AI in a specific component associated AV development, for example, perception, motion planning, decision making, and safety validation. Nonetheless, there still lacks a comprehensive review of the state of the art of the progress and lessons learned from AI applications in supporting AVs, especially in latest years after AlphaGo defeating human go masters.

The paper then proceeds to discuss current practices of using artificial intelligence for Autonomous Vehicles. It states that the generalized structure of the AV driving system to recognize objects on roads includes two primary components:

- **Perception Algorithms** : In this paper, perception is considered as an AV's action using sensors to continuously scan and monitor the environment, which is similar to human vision and other senses. Based on the needed output and goal, existing perception algorithms can be grouped into two categories:
 - a) mediated perception that develops detailed maps of the AV's surroundings through analysing distances to vehicles, pedestrians, trees, road markings, etc.; and
 - b) direct perception that provides integrated scene understanding and decision making.

Mediated perception uses AI approaches such as CNN to detect the single or multiple objects.

- **Training AI for Decision Making** : Given the learned information such as surroundings, vehicle states (velocity and steering angle), decision making related applications such as automatic parking, path planning, and car following have been investigated. Automated parking is automated driving in a restricted scenario of parking with low-speed manoeuvring. Other than semi-automated parking using ultrasonic sensors or radars, automated parking using camera, radar, or LIDAR sensors have been investigated. Potential applications including 3D point cloud-based



Vidya Vikas Education Trust's
Universal College of Engineering

(Permanently unaided | Approved by AICTE, DTE & Affiliated to University of Mumbai)

vehicle/pedestrian detection, parking slot marking recognition, and free space identification are discussed.

The paper then comes to a partial conclusion by stating the major challenges for Artificial Intelligence driven Autonomous Vehicles.

- *Sensor Issues Affecting the Input of AI Approaches*
- *Complexity and Uncertainty*
- *Complex Model Tuning Issues*
- *Solving Hardware Problems*

CONCLUSION:

Hence, we have done a case-study on an AI Application by doing an in-detail analysis of a Research Paper published in IEEE journal.

X-X-X

Experiment 01 Over

Author



EXPERIMENT – 02



Name :	Vivek.Shivakumar. Hotti
Roll Number :	37
Division :	TE-A-Comps Dept
Email :	vivek.hotti@universal.edu.in
Subject :	Artificial Intelligence (AI)

EXPERIMENT - 02

AIM: Assignment for PEAS representation & state space formulation for AI applications in selected domain.

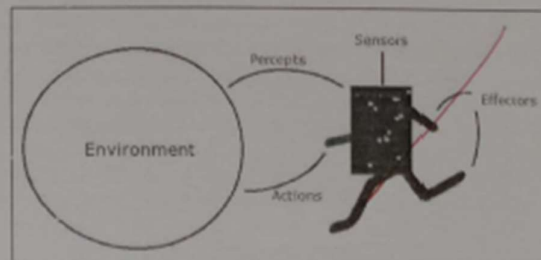
THEORY:

Research Paper Details:

- **Selected Domain :** Use of Artificial Intelligence in Autonomous Vehicles.
- **Paper Title :** Artificial Intelligence Applications in the Development of Autonomous Vehicles: A Survey.
- **Published In :** IEEE/CAA JOURNAL OF AUTOMATICA SINICA, VOL. 7, NO. 2.
- **Published On :** March 2020.
- **Authors :** Yifang Ma, Zhenyu Wang, Hong Yang, and Lin Yang.
- **Link :** <https://scse-jas.net/FileZ1HX8EN/journal/article/rdxhbywh/2020/2/PDF/JAS-2019-0401.pdf>

What is PEAS representation ?

PEAS stands for Performance Measure, Environment, Actuators, and Sensors. It is the short form used for performance issues grouped under Task Environment.





Let's understand PEAS deeply:

1. **Performance Measure:** Performance measure is the unit to define the success of an agent. Performance varies with agents based on their different precepts.
2. **Environment:** Environment is the surrounding of an agent at every instant. It keeps changing with time if the agent is set in motion. There are 5 major types of environments:
 - Fully Observable & Partially Observable
 - Episodic & Sequential
 - Static & Dynamic
 - Discrete & Continuous
 - Deterministic & Stochastic
3. **Actuator:** An actuator is a part of the agent that delivers the output of action to the environment.
4. **Sensor:** Sensors are the receptive parts of an agent that takes in the input for the agent.

PEAS For Automated Car Drive

Agent	Automated Car Drive
Performance Measure	The comfortable trip, Safety, Maximum Distance
Environment	Roads, Traffic, Vehicles
Actuator	Steering wheel, Accelerator, Brake, Mirror
Sensor	Camera, GPS, Odometer

Properties of environment

1) Fully observable vs. partially observable:

- a) If an agent's sensors give it access to the complete state of the environment at each point in time, then the environment is effectively and fully observable.
- b) Fully observable environments are convenient because agents need not maintain any internal state to keep track of the world.
- c) For example, a chessboard or crossword puzzle.
- d) An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.
- e) For example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares, and an automated taxi cannot see what other drivers are thinking.



Vidya Vikas Education Trust's
Universal College of Engineering

(Permanently unaided | Approved by AICTE, DTE & Affiliated to University of Mumbai)

2) Deterministic vs. nondeterministic (stochastic):

- a) If the next state of the environment is completely determined by the current state and the actions selected by the agents, then we say the environment is deterministic.
- b) If the environment is inaccessible, however, then it may appear to be nondeterministic
- c) E.g., Taxi driving (non-deterministic), Teacher's timetable (deterministic)

3) Episodic vs. Sequential

- a) In an episodic environment, the agent's experience is divided into "episodes. Each episode consists of the agent perceiving and then acting.
- b) The quality of its action depends just on the episode itself, because subsequent episodes do not depend on what actions occur in previous episodes.
- c) For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions
- d) In a sequential environment current decision could affect all future decisions. E.g. chess

4) Static vs. dynamic:

- a) If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static.
- b) Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it to worry about the passage of time.
- c) If the environment does not change with the passage of time but the agent's performance score does, then we say the environment is semi-dynamic.
- d) Crossword puzzles are static whereas taxi driving is clearly dynamic.

5) Discrete vs. continuous:

- a) If there are a limited number of distinct, clearly defined percepts and actions our environment is discrete, say that the environment is discrete.
- b) Chess is discrete-there are a fixed number of possible moves on each turn.
- c) Taxi driving is continuous-the speed and location of the taxi and the other vehicles sweep through a range of continuous values.

6) Single-agent vs. Multi agent:

- a) When there is only one agent in a defined environment, it is named the Single-Agent System (SAS) This agent acts and interacts only with its environment.
- b) If there is more than one agent and they interact with each other and their environment, the system is called the Multi-Agent System.
- c) For example, an agent solving a crossword puzzle by itself is clearly in a single-agent environment.
- d) Chess is a multi-agent environment.

7) Competitive vs. Collaborative:

- a) An agent is said to be in a competitive environment when it competes against another agent to optimize the output. For example, Chess game
- b) An agent is said to be in a collaborative environment when multiple agents cooperate to produce the desired output.
- c) For example, when multiple self-driving cars are found on the roads, they cooperate with each other to avoid collisions and reach their destination.



Environment Properties for Automated Car Drive:

Partially Observable
Deterministic
Sequential
Dynamic
Continuous
Multi Agent
Collaborative

Problem-solving agent & State space of a problem

The problem-solving agent is a type of goal-based agent which uses atomic representation with no internal states visible to the problem-solving algorithms and performs precisely by defining problems and its several solutions. "A problem-solving refers to a state where we wish to reach a definite goal from a present state or condition"

Once a solution is found, the actions it recommends can be carried out. This is called the execution phase. Thus, we have a simple "formulate, search, execute" design for the agent.

A problem is defined by the following items:

- a. *Initial State*: It is the starting state or initial step of the agent towards its goal.
- b. *Actions*: It is the description of the possible actions available to the agent.
- c. *Transition Model/successor function*: It describes what each action does. This function returns a set of ordered pairs where each action is one of the legal actions in state x and each successor is a state that can be reached from x by applying the action
- d. *Goal Test*: It determines if the given state is a goal state.
- e. *Path cost*: It assigns a numeric cost to each path that follows the goal.

State Space for AI Car Drive

- 1) **States**: Navigating and driving through real roads without driver/user interference.
- 2) **Initial State**: Off car.
- 3) **Actions**: Changing course based on real time data getting in from sensors, applying breaks providing adequate accelerator when needed, changing gears when needed.
- 4) **Goal Test**: Check whether the car has reached its destination safely without any physical damage to itself or otherwise & with minimal user interference.
- 5) **Path Cost**: Number of actions required; fuel required for the car to reach its destination.

CONCLUSION:

Hence, we have done a detailed analysis of PEAS representation & did State Space Formulation for selected AI Application (Self Driving Automated Cars).

X-X-X

Experiment 02 Over



EXPERIMENT – 03

AIM: To implement BFS algorithm.

THEORY:

Breadth First Search or BFS for a Graph:

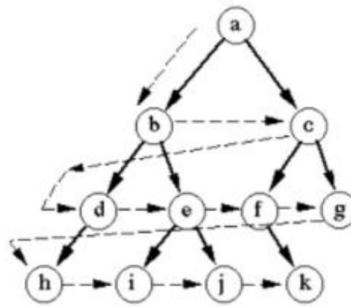
Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a Boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex. It uses queue for storing visited node.

Advantages of BFS:

1. The solution will definitely find out by BFS If there is some solution.
2. BFS will never get trapped in a blind alley, which means unwanted nodes.
3. If there is more than one solution then it will find a solution with minimal steps.

Disadvantages Of BFS:

1. Memory Constraints As it stores all the nodes of the present level to go for the next level.
2. If a solution is far away then it consumes time.



Breadth-first search

Application Of BFS:

1. Finding the Shortest Path.
2. Checking graph with petiteness.
3. Copying Cheney's Algorithm.

INPUT CODE TO COMPUTE first():

```
import java.io.*;
import java.util.*;
class Main
{
    private int V; //number of nodes in the graph
    private LinkedList<Integer> adj[]; //adjacency list
    private Queue<Integer> queue; //maintaining a queue
    Main(int v)
    {
        V = v;
        adj = new LinkedList[V];
        for (int i=0; i<V; i++)
        {
            adj[i] = new LinkedList<>();
        }
        queue = new LinkedList<Integer>();
    }
    void addEdge(int v,int w)
    {
        adj[v].add(w); //adding an edge to the adjacency list (edges are bidirectional in this example)
    }
    void BFS(int n)
    {
        boolean nodes[] = new boolean[V]; //initialize boolean array for holding the data
        int a = 0;
        nodes[n]=true;
        queue.add(n); //root node is added to the top of the queue
        while (queue.size() != 0)
```




```
{
n = queue.poll(); //remove the top element of the queue
System.out.print(n+" "); //print the top element of the queue
for (int i = 0; i < adj[n].size(); i++) //iterate through the linked list and push all
neighbors into queue
{
a = adj[n].get(i);
if (!nodes[a]) //only insert nodes into queue if they have not been explored already
{
nodes[a] = true;
queue.add(a);
}
}
}
}
public static void main(String args[])
{
Main graph = new Main(6);
graph.addEdge(0, 1);
graph.addEdge(0, 3);
graph.addEdge(0, 4);
graph.addEdge(4, 5);
graph.addEdge(3, 5);
graph.addEdge(1, 2);
graph.addEdge(1, 0);
graph.addEdge(2, 1);
graph.addEdge(4, 1);
graph.addEdge(3, 1);
graph.addEdge(5, 4);
graph.addEdge(5, 3);
System.out.println("Vivek Hotti, roll:37");
System.out.println("The Breadth First Traversal of the graph is as follows :");
graph.BFS(2);
}
}
```

OUTPUT

```
Vivek Hotti, roll:37
The Breadth First Traversal of the graph is as follows :
2 1 0 3 4 5

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

(BFS starting from 2)



CONCLUSION:

Thus, we learned about Breath First Search and implemented it..

X-X-X

Experiment 03 Over



EXPERIMENT – 04

AIM: To implement Depth First Search.

THEORY:

Depth First Search on a Graph:

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a Boolean visited array. It uses stack for storing visited node.

Advantages of DFS:

1. The memory requirement is Linear WRT Nodes.
2. Less time and space complexity rather than BFS.
3. The solution can be found out without much more search.

Disadvantages of DFS:

1. Not Guaranteed that it will give you a solution.
2. Cut-off depth is smaller so time complexity is more.
3. Determination of depth until the search has proceeded.

Applications of DFS:

1. Finding Connected components.
2. Topological sorting.
3. Finding Bridges of the graph



INPUT:

```
import java.io.*;
import java.util.*;
class Graph {
private int V;
private LinkedList<Integer> adj[]; //adjacency list
public Graph(int v)
{
V = v;
adj = new LinkedList[v];
for (int i = 0; i < v; ++i)
{
adj[i] = new LinkedList();
}
void addEdge(int v, int w)m
{
adj[v].add(w); //adding an edge to the adjacency
list (edges are bidirectional in this example)
}
void DFSUtil(int vertex, boolean nodes[])
{
nodes[vertex] = true; //mark the node as explored
System.out.print(vertex + " ");
int a = 0;
for (int i = 0; i < adj[vertex].size(); i++) //iterate through the linked
list and then propagate to the next few nodes
{
a = adj[vertex].get(i);
if (!nodes[a]) //only propagate to next nodes
which haven't been explored
{
DFSUtil(a, nodes);
}
}
}
void DFS(int v)
{
boolean already[] = new boolean[V]; //initialize a new boolean
array to store the details of explored nodes
DFSUtil(v, already);
}
public static void main(String args[])
{
Graph g = new Graph(6);g.addEdge(0, 1);
g.addEdge(0, 2);
g.addEdge(1, 0);
g.addEdge(1, 3);
g.addEdge(2, 0);
g.addEdge(2, 3);
```



```
g.addEdge(3, 4);
g.addEdge(3, 5);
g.addEdge(4, 3);
g.addEdge(5, 3);
System.out.println(
"Following is Depth First Traversal: ");
g.DFS(0);
}
```

OUTPUT To Above Program

```
Graphs
Enter no. of edges: 4
Enter no. of vertices: 4
Enter the edges in V1 V2 : 1 1
Enter the edges in V1 V2 : 1 3
Enter the edges in V1 V2 : 2 2
Enter the edges in V1 V2 : 4 3
1 0 1 0
0 1 0 0
0 0 0 0
0 0 1 0
Enter source Vertex: 1
0-> 1->
```

CONCLUSION:

Thus, we learned about Depth First Search and implemented it.

X-X-X

Experiment 04 Over



EXPERIMENT – 05

AIM: To implement A* Algorithm

THEORY:

A* Search Algorithm

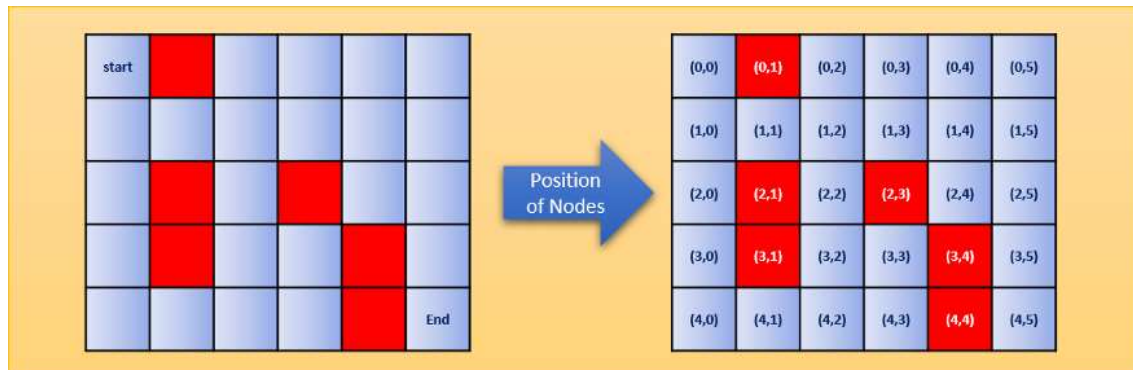
A* is based on using heuristic methods to achieve *optimality* and *completeness*, and is a variant of the best-first algorithm. It is one of the most successful search algorithms to find the shortest path between nodes or graphs. It is an informed search algorithm, as it uses information about path cost and also uses heuristics to find the solution. Each time A* enters a node, it calculates the cost, $f(n)$ (n being the neighbouring node), to travel to all of the neighbouring nodes, and then enters the node with the lowest value of $f(n)$.

Advantages:

1. It is optimal search algorithm in terms of heuristics.
2. It is one of the best heuristic search techniques.
3. It is used to solve complex search problems.
4. There is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages:

1. This algorithm is complete if the branching factor is finite and every action has fixed cost.
2. The performance of A* search is dependent on accuracy of heuristic algorithm used to compute the function $h(n)$.



INPUT:

```
from queue import PriorityQueue
#Creating Base Class
class State(object):
def __init__(self, value, parent, start = 0, goal = 0):
self.children = []
self.parent = parent
self.value = value
self.dist = 0
if parent:
self.start = parent.start
self.goal = parent.goal
self.path = parent.path[:]
self.path.append(value)
else:
self.path = [value]
self.start = start
self.goal = goal
def GetDistance(self):
pass
def CreateChildren(self):
pass
# Creating subclass
class State_String(State):
def __init__(self, value, parent, start = 0, goal = 0 ):
super(State_String, self).__init__(value, parent, start, goal)
self.dist = self.GetDistance()
def GetDistance(self):
if self.value == self.goal:
return 0
```



```
dist = 0
for i in range(len(self.goal)):
    letter = self.goal[i]
    dist += abs(i - self.value.index(letter))
return dist
def CreateChildren(self):
    if not self.children:
        for i in range(len(self.goal)-1):
            val = self.value
            val = val[:i] + val[i+1] + val[i] + val[i+2:]
            child = State_String(val, self)
            self.children.append(child)
# Creating a class that hold the final magic
class A_Star_Solver:
    def __init__(self, start, goal):
        self.path = []
        self.vistedQueue = []
        self.priorityQueue = PriorityQueue()
        self.start = start
        self.goal = goal
    def Solve(self):
        startState = State_String(self.start,0,self.start,self.goal)
        count = 0
        self.priorityQueue.put((0,count, startState))
        while(not self.path and self.priorityQueue.qsize()):
            closesetChild = self.priorityQueue.get()[2]
            closesetChild.CreateChildren()
            self.vistedQueue.append(closesetChild.value)
            for child in closesetChild.children:
                if child.value not in self.vistedQueue:
                    count += 1
                    if not child.dist:
                        self.path = child.path
                        break
            self.priorityQueue.put((child.dist,count,child))
        if not self.path:
            print("Goal Of is not possible !" + self.goal )
        return self.path
# Calling all the existing stuffs
if __name__ == "__main__":
    start1 = "hema"
    goal1 = "mahe"
    print("Starting....")
    a = A_Star_Solver(start1,goal1)
    a.Solve()
    for i in range(len(a.path)):
        print("{0}{1}".format(i,a.path[i]))
```



OUTPUT To Above Program

```
Starting....  
0) hema  
1) hmea  
2) mhea  
3) mhae  
4) mahe  
  
...Program finished with exit code 0  
Press ENTER to exit console. █
```

CONCLUSION:

Thus, we learned about A* Search Algorithm and implemented it.

X-X-X

Experiment 05 Over



EXPERIMENT – 06

AIM: To implement Minimax Algorithm for Game Playing.

THEORY:

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc. In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible. Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

Advantages:

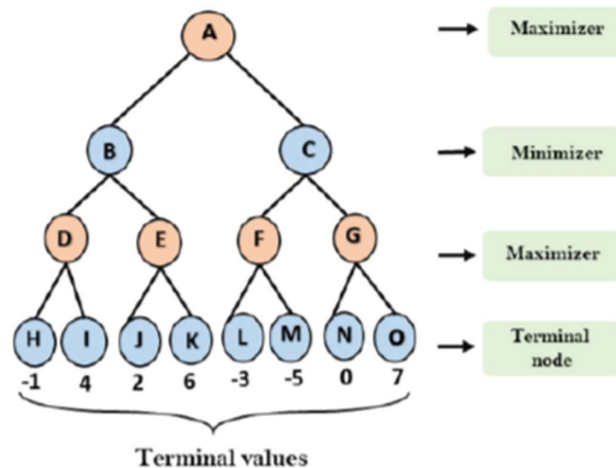
1. Minimax algorithm is a beneficial problem-solving algorithm that helps perform a thorough assessment of the search space.
2. It makes it possible to implement decision making in Artificial Intelligence, which has further given way to the development of new and smart machines, systems, and computers.

Disadvantages:

1. It has a huge branching factor, which makes the process of reaching the goal state slow.



2. Search and evaluation of unnecessary nodes or branches of the game tree degrades the overall performance and efficiency of the engine.
3. Both min and max players have lots of choices to decide from.
4. Exploring the entire tree is not possible as there is a restriction of time and space.



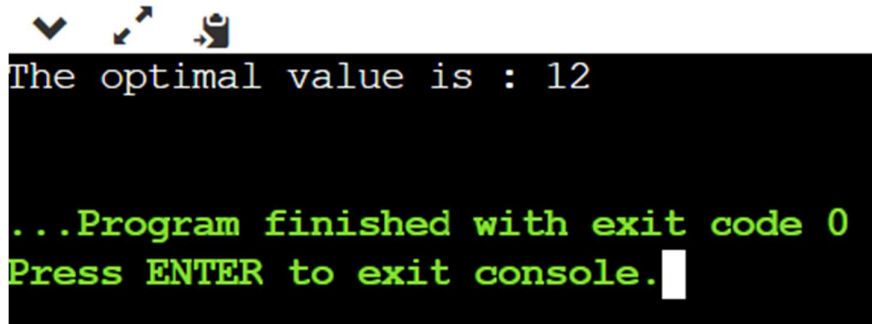
INPUT for DEAD CODE ELIMINATION:

```
int minimax(int depth, int nodeIndex, bool isMax, int scores[], int h)
{
    // Terminating condition. i.e
    // leaf node is reached
    if (depth == h)
        return scores[nodeIndex];
    // If current move is maximizer,
    // find the maximum attainable
    // value
    if (isMax)
        return max(minimax(depth+1, nodeIndex*2, false, scores, h),
                    minimax(depth+1, nodeIndex*2 + 1, false, scores, h));
    // Else (If current move is Minimizer), find the minimum
    // attainable value
    else
        return min(minimax(depth+1, nodeIndex*2, true, scores, h),
                    minimax(depth+1, nodeIndex*2 + 1, true, scores, h));
}
// A utility function to find Log n in base 2
int log2(int n)
{
    return (n==1)? 0 : 1 + log2(n/2);
}
```



```
}  
// Driver code  
int main()  
{  
    // The number of elements in scores must be  
    // a power of 2.  
    int scores[] = {3, 5, 2, 9, 12, 5, 23, 23};  
    int n = sizeof(scores)/sizeof(scores[0]);  
    int h = log2(n);  
    int res = minimax(0, 0, true, scores, h);  
    cout << "The optimal value is : " << res << endl;  
    return 0;  
}
```

OUTPUT:



```
The optimal value is : 12  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

CONCLUSION:

Thus, we learned about Minimax Algorithm in Game Playing and implemented it.

X-X-X

Experiment 06 Over



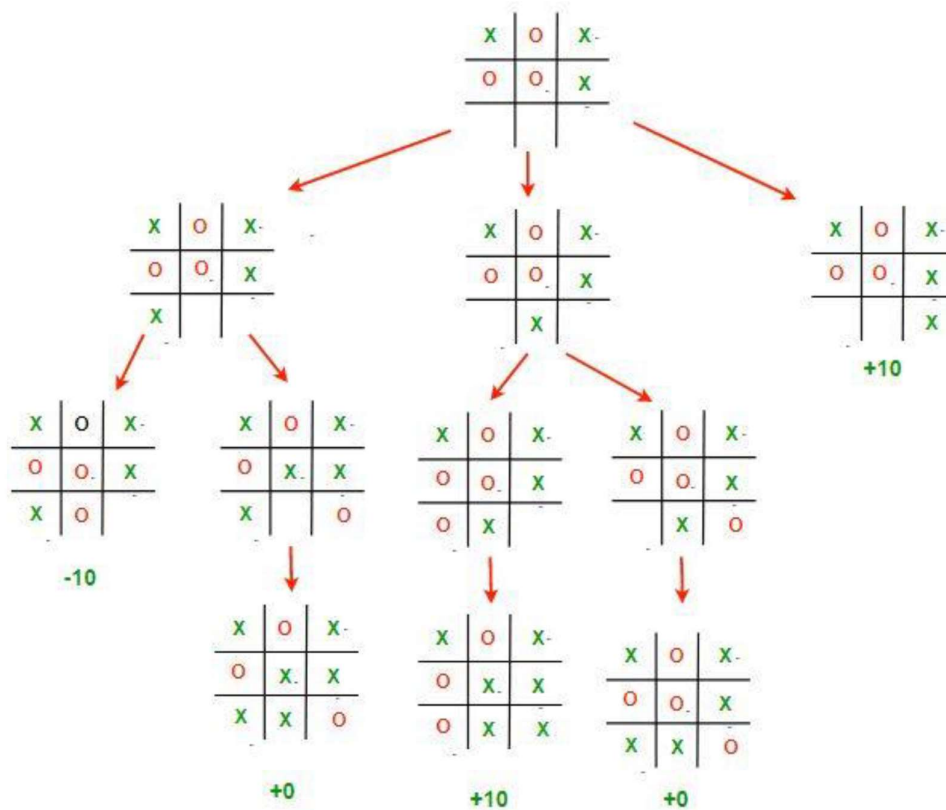
EXPERIMENT – 07

AIM: To implement Minimax Algorithm in Tic Tac Toe.

THEORY:

Two players alternately put Xs and Os in compartments of a figure formed by two vertical lines crossing two horizontal lines and each tries to get a row of three Xs or three Os before the opponent does.

Game Tree:



Pseudocode:



```
function minimax(board, depth, isMaximizingPlayer):
    if current board state is a terminal state :
        return value of the board
    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each move in board :
            value = minimax(board, depth+1, false)
            bestVal = max( bestVal, value)
        return bestVal
    else :
        bestVal = +INFINITY
        for each move in board :
            value = minimax(board, depth+1, true)
            bestVal = min( bestVal, value)
    return bestVal
```

INPUT

```
// Java program to find the
// next optimal move for a player
class GFG
{
    static class Move
    {
        int row, col;
    };
    static char player = 'x', opponent = 'o';
    // This function returns true if there are moves
    // remaining on the board. It returns false if
    // there are no moves left to play.
    static Boolean isMovesLeft(char board[][] )
    {
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                if (board[i][j] == '_')
                    return true;
        return false;
    }
    // This is the evaluation function as discussed
    // in the previous article ( http://goo.gl/sJgv68 )
    static int evaluate(char b[][] )
    {
        // Checking for Rows for X or O victory.
        for (int row = 0; row < 3; row++)
        {
            if (b[row][0] == b[row][1] &&
                b[row][1] == b[row][2])
            {
                if (b[row][0] == player)
```



```
return +10;
else if (b[row][0] == opponent)
return -10;
}
}
// Checking for Columns for X or O victory.
for (int col = 0; col < 3; col++)
{
if (b[0][col] == b[1][col] &&
b[1][col] == b[2][col])
{
if (b[0][col] == player)
return +10;
else if (b[0][col] == opponent)
return -10;
}
}
// Checking for Diagonals for X or O victory.
if (b[0][0] == b[1][1] && b[1][1] == b[2][2])
{
if (b[0][0] == player)
return +10;
else if (b[0][0] == opponent)
return -10;
}
if (b[0][2] == b[1][1] && b[1][1] == b[2][0])
{
if (b[0][2] == player)
return +10;
else if (b[0][2] == opponent)
return -10;
}
// Else if none of them have won then return 0
return 0;
}
// This is the minimax function. It considers all
// the possible ways the game can go and returns
// the value of the board
static int minimax(char board[][],
int depth, Boolean isMax)
{
int score = evaluate(board);
// If Maximizer has won the game
// return his/her evaluated score
if (score == 10)
return score;
// If Minimizer has won the game
// return his/her evaluated score
if (score == -10)
return score;
// If there are no more moves and
```



```
// no winner then it is a tie
if (isMovesLeft(board) == false)
return 0;
// If this maximizer's move
if (isMax)
{
int best = -1000;
// Traverse all cells
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 3; j++)
{
// Check if cell is empty
if (board[i][j] == '_')
{
// Make the move
board[i][j] = player;
// Call minimax recursively and choose
// the maximum value
best = Math.max(best, minimax(board,
depth + 1, !isMax));
// Undo the move
board[i][j] = '_';
}
}
}
return best;
}
// If this minimizer's move
else
{
int best = 1000;
// Traverse all cells
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 3; j++)
{
// Check if cell is empty
if (board[i][j] == '_')
{
// Make the move
board[i][j] = opponent;
// Call minimax recursively and choose
// the minimum value
best = Math.min(best, minimax(board,
depth + 1, !isMax));
// Undo the move
board[i][j] = '_';
}
}
}
}
```



```
return best;
}
}
// This will return the best possible
// move for the player
static Move findBestMove(char board[][])
{
    int bestVal = -1000;
    Move bestMove = new Move();
    bestMove.row = -1;
    bestMove.col = -1;
    // Traverse all cells, evaluate minimax function
    // for all empty cells. And return the cell
    // with optimal value.
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            // Check if cell is empty
            if (board[i][j] == '_')
            {
                // Make the move
                board[i][j] = player;
                // compute evaluation function for this
                // move.
                int moveVal = minimax(board, 0, false);
                // Undo the move
                board[i][j] = '_';
                // If the value of the current move is
                // more than the best value, then update
                // best/
                if (moveVal > bestVal)
                {
                    bestMove.row = i;
                    bestMove.col = j;
                    bestVal = moveVal;
                }
            }
        }
    }
    System.out.printf("The value of the best Move " +
        "is : %d\n\n", bestVal);
    return bestMove;
}
// Driver code
public static void main(String[] args)
{
    char board[][] = {{ 'x', 'o', 'x' },
        { 'o', 'o', 'x' },
        { '_', '_', '_' }};
    Move bestMove = findBestMove(board);
```



```
System.out.printf("The Optimal Move is :\n");  
System.out.printf("ROW: %d COL: %d\n\n",  
bestMove.row, bestMove.col );  
}  
}
```

OUTPUT:

Run	Output
	<pre>java -cp /tmp/mGvNT07UoE GFG The value of the best Move is : 10 The Optimal Move is : ROW: 2 COL: 2 </pre>

CONCLUSION:

Thus, we learned about Minimax Algorithm in Tic Tac Toe and implemented it.

X-X-X

Experiment 07 Over



EXPERIMENT – 08

AIM: To perform a case study on Prolog Software.

THEORY:

Description of Prolog:

Prolog (PROgramming in LOGic) is a 5th generation programming language (5GL). Similar to 4GL but only difference is 4GL is used for specific problems/purposes like SQL for DBMS only while 5GL are used to make computers solve problems for you. In 3GL certain algorithms are used to solve problems but in 5GL only conditions and break-points are needed to solve the problems. Specify what you want and conditions to be met and computer will solve the problems for you. Some famous examples are Prolog, OPS5 and Mercury

Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is intended primarily as a declarative programming language: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations.

Syntax: relation(entity1, entity2,kth entity).

Example:

```
friends(raju, mahesh).  
singer(sonu).  
odd_number(5).
```

Explanation:

These facts can be interpreted as :

raju and mahesh are friends.

sonu is a singer.

5 is an odd number.

Query:



Query 1: ?- singer(sonu).

Output: Yes.

Explanation: As our knowledge base contains the above fact, so output was 'Yes', otherwise it would have been 'No'.

Query 2: ?- odd_number(7).

Output: No.

Explanation: As our knowledge base does not contain the above fact, so output was 'No'.

Key Features:

1. Unification: The basic idea is, can the given terms be made to represent the same structure.
2. Backtracking: When a task fails, prolog traces backwards and tries to satisfy previous task.
3. Recursion: Recursion is the basis for any search in program.

Advantages:

1. Easy to build database. Doesn't need a lot of programming effort.
2. Pattern matching is easy. Search is recursion based.
3. It has built in list handling. Makes it easier to play with any algorithm involving lists.

Disadvantages:

1. LISP (another logic programming language) dominates over prolog with respect to I/O features.
2. Sometimes input and output is not easy.

The applications of prolog are as follows:

1. Specification Language
2. Robot Planning
3. Natural language understanding
4. Machine Learning
5. Problem Solving
6. Intelligent Database retrieval
7. Expert System
8. Automated Reasoning



CONCLUSION:

Hence, we understood what prolog is and why to use it

X-X-X

Experiment 08 Over



EXPERIMENT – 09

AIM: To perform a case study on AI Planning Projects of IBM.

THEORY:

What is AI Planning:

Planning is a long-standing sub-area of Artificial Intelligence (AI). Planning is the task of finding a procedural course of action for a declaratively described system to reach its **goals** while **optimizing overall performance** measures. Automated planners find the transformations to apply in each given state out of the possible transformations for that state. In contrast to the classification problem, planners provide guarantees on the solution quality.

Why is it Important: Planning Applications in Industry?

1. Automation is an emerging trend that requires efficient automated planning
2. Many applications of planning in industry (e.g. robots and autonomous systems, cognitive assistants, cyber security, service composition)

How to Spot a Planning Problem

1. Declarative
 - o You want to find a procedural course of action for a declaratively described system to reach its goals while optimizing overall performance measures.
2. Domain Knowledge can be elicited or learned over time
 - o Existing domain knowledge can/should be exploited for building the model
 - o Human involvement controllable. Humans build the model and can contribute to the solution by introducing knowledge.
3. Favor consistency over learning transient behaviors
 - o There is a structure of the problem that cannot be learned just training



- o When no large training data is available
- o Changes in the problem can make previous data irrelevant

Advantages of AI Planning Techniques

1. When **explainability** is desired
 - a. When you want to be able to explain why a particular course of action was chosen
 - b. Assignment of responsibility/blame is essential for automation of processes (e.g., autonomous driving, medical expert systems)
2. Rapid prototyping: **short time to solution**
3. Variety of of-the-shelf planners available both **IBM proprietary** and **open-source**
4. Your problem is frequently changing, even small changes.
5. No need to change the solution, only tweak the model

Success Stories: When Planning Meets DL

In many real life applications, there is a structure of the problem that cannot be learned with DL (there are just not enough examples). Solving optimization problems with learning is hard, but integrating planning techniques with heuristic guidance learned by DL will result in the most famous success stories of AI to day.

1. GO player AlphaGO uses planning (monte-carlo tree search) with deep learning (heuristic guidance) to select the next move
2. Cognitive assistant Viv (Samsung) uses knowledge graph, planning, and deep learning to answer complicated queries

Example AI Planning Projects in IBM:-

Demo: Causal Knowledge Extraction Through Large-Scale Text Mining,
Oktie Hassanzadeh,
Debarun Bhattacharjya, Mark Feblowitz, Kavitha Srinivas, Michael
Perrone, Shirin Sohrabi, Michael Katz. Poster / Demo Reception 1,
Americas Hall I / II: Sunday, February 9 19:30
– 21:30.



IBM also has a rich history of work at the intersection of automated planning and business processes. During AAAI 2020 Workshops, we will highlight challenges in the model acquisition task for the management of business processes and the design of goal-oriented conversational agents. These works will focus on acquiring domain models from experts in the loop, as opposed to learning from data. However, we will again observe instances of synergy between learning and planning approaches.

Paper info: Planning for Goal-Oriented Dialogue Systems. Christian Muise, Tathagata Chakraborti, Shubham Agarwal, Ondrej Bajgar, Arunima Chaudhary, Luis Lastras, Josef Ondrej, Miroslav Vodol and Charlie Wiecha. AAAI 2020 Workshop on Interactive and Conversational Recommendation Systems (WICRS). Presentation: Saturday February 8th 11:40 – 11:52.

Paper info: D3BA: A Tool for Optimizing Business Processes Using Non-Deterministic Planning. Tathagata Chakraborti and Yasaman Khazaeni. AAAI-20 Workshop on Intelligent Process Automation (IPA-20). Presentation: Friday February 7th 15:10- 15:30.

CONCLUSION:

To prepare for the future society in which artificial intelligences (AI) will have much more pervasive influence on our lives, a better understanding of the difference between AI and human intelligence is necessary.

X-X-X

Experiment 09 Over



EXPERIMENT – 10

AIM: To perform case study on implementation of BBN.

THEORY:

What is AI Planning:

1. What is BBN?

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph." It is also called a Bayes network, belief network, decision network, or Bayesian model.

Bayesian networks are probabilistic, because these networks are built from a probability distribution, and also use probability theory for prediction and anomaly detection.

Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction, and decision making under uncertainty.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- o Directed Acyclic Graph
- o Table of conditional probabilities.

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an Influence diagram.

2. BBN python example with real life data.



Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

o The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.

o The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.

o The conditional distributions for each node are given as conditional probabilities table or CPT.

o Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.

o In CPT, a boolean variable with k boolean parents contains 2^k probabilities. Hence, if there are two parents, then CPT will contain 4 probability values.

List of all events occurring in this network:

o Burglary (B) o Earthquake(E) o Alarm(A) o David Calls(D) o Sophia calls(S)

We can write the events of problem statement in the form of probability: $P[D, S, A, B, E]$, can rewrite the above probability statement using joint probability distribution:

$$\begin{aligned} P[D, S, A, B, E] &= P[D | S, A, B, E]. P[S, A, B, E] \\ &= P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E] \\ &= P[D | A]. P[S | A, B, E]. P[A, B, E] \\ &= P[D | A]. P[S | A]. P[A | B, E]. P[B, E] \\ &= P[D | A]. P[S | A]. P[A | B, E]. P[B | E]. P[E] \end{aligned}$$

Let's take the observed probability for the Burglary and earthquake component:

$P(B = \text{True}) = 0.002$, which is the probability of burglary.

$P(B = \text{False}) = 0.998$, which is the probability of no burglary.



$P(E = \text{True}) = 0.001$, which is the probability of a minor earthquake

$P(E = \text{False}) = 0.999$, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

B	E	$P(A = \text{True})$	$P(A = \text{False})$
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	$P(D = \text{True})$	$P(D = \text{False})$
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	$P(S = \text{True})$	$P(S = \text{False})$
True	0.75	0.25
False	0.02	0.98

The Conditional probability of Alarm A depends on Burglar and earthquake:

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

$$\begin{aligned} P(S, D, A, \neg B, \neg E) &= P(S|A) * P(D|A) * P(A|\neg B \wedge \neg E) * P(\neg B) * P(\neg E). \\ &= 0.75 * 0.91 * 0.001 * 0.998 * 0.999 \\ &= 0.00068045. \end{aligned}$$

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.



The semantics of Bayesian Network:

There are two ways to understand the semantics of the Bayesian network, which is given below:

1. To understand the network as the representation of the Joint probability distribution. It is helpful to understand how to construct the network.
2. To understand the network as an encoding of a collection of conditional independence statements

It is helpful in designing inference procedure.

CONCLUSION:

A Bayesian inference network (BIN) provides an interesting alternative to existing tools for similarity-based virtual screening. The BIN is particularly effective when the active molecules being sought have a high degree of structural homogeneity but has been found to perform less well with structurally heterogeneous sets of actives.

X-X-X

Experiment 10 Over

(((((())))))

(You have reached the end of the AI Lab Manual)