1) PEAS and State Description - Automobile Driver Agent

PEAS:

Performance Measure: Safety, reaching destination, fuel efficiency, obeying traffic laws, passenger comfort.

Environment: Roads, traffic signals, other vehicles, pedestrians, weather.

Actuators: Steering, accelerator, brakes, indicators.

Sensors: Cameras, GPS, speedometer, radar, LIDAR, proximity sensors.

State Description:

The Automobile Driver Agent is an AI-based system that mimics a human driver. It perceives the environment through sensors and takes decisions to drive safely using actuators.

2) PEAS and State Description - Online English Tutor

PEAS:

Performance Measure: Learning progress, test scores, student engagement.

Environment: Students, online learning platform, internet.

Actuators: Screen display, audio output, chatbot messages.

Sensors: Keyboard input, quiz results, speech/audio input.

State Description:

The Online English Tutor is an intelligent teaching system that adapts lessons, interacts with students, and tracks learning through quizzes and inputs.

3) Adversarial Search - Min-Max Algorithm

Concept:

Used in two-player games. Max tries to maximize the score, Min tries to minimize it.

Code:

```
def minmax(depth, nodeIndex, isMax, scores, targetDepth):
    if depth == targetDepth:
        return scores[nodeIndex]
    if isMax:
        return max(minmax(depth+1, nodeIndex*2, False, scores, targetDepth),
                   minmax(depth+1, nodeIndex*2 + 1, False, scores, targetDepth))
    else:
        return min(minmax(depth+1, nodeIndex*2, True, scores, targetDepth),
                   minmax(depth+1, nodeIndex*2 + 1, True, scores, targetDepth))


scores = [3, 5, 6, 9, 1, 2, 0, -1]
print("Optimal value:", minmax(0, 0, True, scores, 3))
```

Explanation:

- Recursively selects best move assuming opponent plays optimally.
- Traverses a binary game tree to find optimal value.

4) BFS - Breadth First Search

Concept:

BFS visits nodes level-by-level using a queue.

Code:

```
from collections import deque
def bfs(graph, start):
    visited = set()
    queue = deque([start])
```

```
    while queue:
        vertex = queue.popleft()
        if vertex not in visited:
            print(vertex, end=" ")
            visited.add(vertex)
            queue.extend(set(graph[vertex]) - visited)

graph = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['F'], 'F': []}
bfs(graph, 'A')
```

Explanation:

- Queue stores next nodes to visit.
- Avoids revisiting by checking 'visited' set.

5) DFS - Depth First Search

Concept:

DFS explores one path deeply before backtracking.

Code:

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
```

```
graph = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['F'], 'F': []}
dfs(graph, 'A')
```

Explanation:

- Uses recursion.
- Visited set ensures no node is visited twice.

6) PEAS and State Description - Medical Diagnosis System

PEAS:

Performance Measure: Accuracy, diagnosis time, patient satisfaction.

Environment: Patients, symptoms, medical reports, hospital records.

Actuators: Display diagnosis, suggest treatment, alerts.

Sensors: Input forms, medical tests, patient data.

State Description:

An AI system that analyses symptoms and test results to suggest possible diagnoses, supporting doctors in decision-making.

7) Hill Climbing Algorithm

Concept:

Hill climbing is an optimization algorithm that moves in the direction of increasing value.

Code:

```
def hill_climb(function, x_start, max_iterations, step_size):
    x = x_start
    for _ in range(max_iterations):
```

```
        next_x = x + step_size
        if function(next_x) > function(x):
            x = next_x
        else:
            break
    return x


def f(x):
    return -x**2 + 4


result = hill_climb(f, x_start=0, max_iterations=100, step_size=0.1)
print("Local maximum at x =", result, "with value =", f(result))
```

Explanation:

- Finds peak of the function by comparing next point.
- Stops when no further improvement is found.