Module 03

Introduction to Optimal Pathfinding

- Optimal pathfinding involves finding the most efficient route from a starting point to a goal state in a graph or grid.
- Efficiency is often measured in terms of distance, time, cost, etc.
- Different algorithms can be used to find optimal paths, each with its own strengths and weaknesses.

Branch and Bound

- Branch and Bound is a systematic method for finding optimal solutions to combinatorial optimization problems.
- It explores the search space by branching into different paths and uses bounding techniques to prune the search tree.
- In pathfinding, it can be used to find the shortest path by exploring promising paths and discarding those that are not optimal.
- It guarantees finding the optimal solution but may have high time complexity for large graphs.

A* Algorithm

- A* (pronounced "A star") is a popular heuristic search algorithm used for pathfinding and graph traversal.
- It uses both the cost to reach a node and an estimate of the cost from that node to the goal to guide the search.
- A* maintains two lists: open list (nodes to be evaluated) and closed list (nodes already evaluated).
- It selects nodes to expand based on a heuristic function, which estimates the cost from the current node to the goal.
- A* is guaranteed to find the optimal path if certain conditions are met (admissibility and consistency of the heuristic).

Algorithm

```
Initialization:

    Create an open list and a closed list. Initially, the open list contains only the starti
    Assign a cost of 0 to the starting node and an estimated cost (heuristic) to reach the g

Main Loop:

    While the open list is not empty:
        Select the node with the lowest total cost (sum of the path cost from the start node
        If the current node is the goal node, the path has been found. Terminate the search.
        Otherwise, move the current node from the open list to the closed list to mark it as
```

```
Expansion:

Expand the current node by considering all its neighboring nodes (successors).
    For each successor:
        Calculate the cost of reaching that successor from the start node (known as the g-
value).
        Calculate the estimated cost to reach the goal from the successor (known as the heu
value).
        Compute the total cost f(n) = g(n) + h(n) for the successor.
        If the successor is already in the open list and the new f-
value is lower, update its cost and parent.
        If the successor is not in the open list, add it to the open list with the computed

Termination:

    If the open list becomes empty and the goal has not been reached, there is no path from

Backtracking:

Once the goal is reached, reconstruct the path by backtracking from the goal node through th
```

Example

IDA Algorithm:*

- IDA* (Iterative Deepening A*) is a variant of the A* algorithm designed to address memory constraints.
- It avoids storing the entire search tree in memory by performing a depth-first search with iterative deepening.
- At each iteration, it limits the search depth based on a threshold, gradually increasing the threshold until the goal is found.
- IDA* guarantees finding the optimal solution like A* but uses less memory, making it suitable for memory-constrained environments.
- It sacrifices some efficiency for memory savings compared to A*, especially in domains with high branching factors.