Greedy Best-First Search Algorithm

Greedy Best-First Search is a search algorithm that makes use of a heuristic to guide its pathfinding process. The heuristic function estimates the cost from a given node to the nearest goal, allowing the algorithm to prioritize nodes that appear to be closer to the goal. This algorithm is 'greedy' because it always selects the path that seems the best at the current moment, without considering the long-term consequences of this choice.

Implementation

In the maze navigation context, the Greedy Best-First Search algorithm was implemented with the following key components:

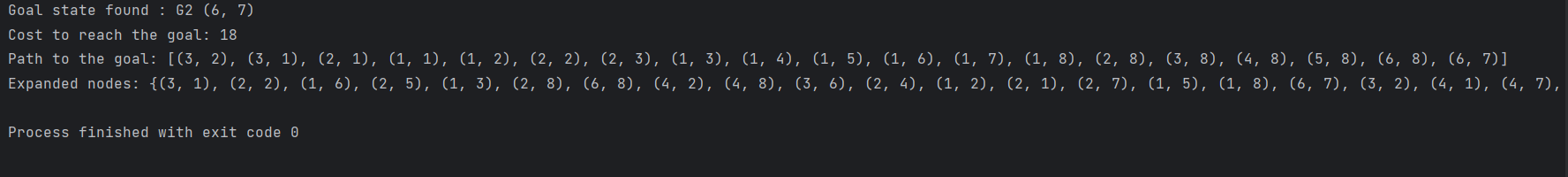
* **Heuristic Function:** This uses Euclidean distance to estimate the shortest path from any node to the closest goal.
* **Node Selection**: At each step, the algorithm selects the neighboring node with the lowest heuristic value.
* **Trap Handling:** When encountering a trap, the algorithm adds an additional cost to the path. However, this cost does not influence the next node selection as it is solely guided by the heuristic.
* **Pathfinding:** The algorithm traces a path from the start node to the goal by continuously moving to the node that appears closest to the goal.

**Fields**

* **graph:** A dictionary or map representing the maze, where keys are nodes (coordinates) and values are lists of neighboring nodes.
* **goals:** A set of tuples representing the goal nodes in the maze.
* **traps:** A set of tuples representing the trap nodes in the maze.

**Methods**

* **greedy\_best\_first\_search(graph, start, goals, traps):** The main function to execute the Greedy Best-First Search algorithm.
* **graph:** The maze graph.
* **start:** Starting node (tuple).
* **goals:** Set of goal nodes.
* **traps:** Set of trap nodes.
* **heuristic(node, goals):** Calculates the heuristic cost from the current node to the closest goal node.
* **node:** The current node.
* **goals:** Set of goal nodes.



A\* Search Algorithm

A\* Search is a popular and versatile pathfinding algorithm that combines features of uniform-cost search and Greedy Best-First Search. It uses a heuristic to estimate the cost to reach the goal from a node, and it also considers the cost to reach that node from the start. This combination helps A\* Search find the most cost-effective path to the goal.

**Implementation**

For navigating the maze, the A\* Search algorithm included these elements:

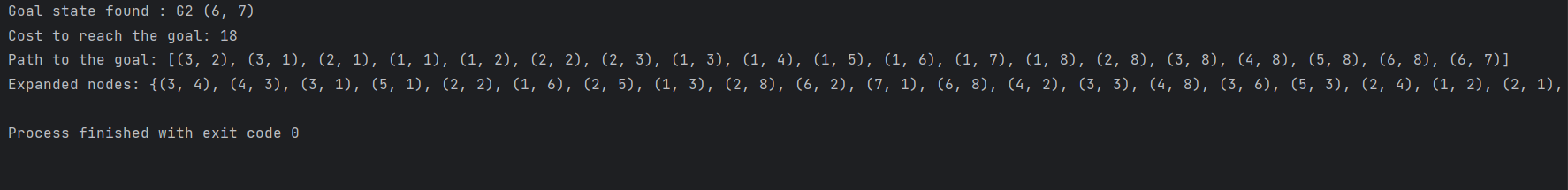
* **Heuristic Function:** Similar to Greedy Best-First Search, it uses the Euclidean distance for heuristic calculation.
* **Cost Calculation:** The total cost of a path is the sum of the distance traveled from the start node and the estimated distance to the goal (heuristic).
* **Node Selection:** The algorithm chooses the node with the lowest total cost (sum of actual cost to reach the node and the heuristic).
* **Trap Handling:** Traps increase the cost of the path. This additional cost influences future node selection, as the total cost is a key factor in the algorithm.
* **Pathfinding:** A\* Search systematically explores paths, balancing between the distance already traveled and the estimated distance to the goal.

Fields

* **graph:** Similar to the Greedy Best-First Search, it's a dictionary or map representing the maze.
* **goals:** A set of tuples indicating the goal nodes.
* **traps:** A set of tuples marking the trap nodes.

Methods

* **a\_star\_search(graph, start, goals, traps):** The main function to perform the A\* Search algorithm.
* **graph:** The maze graph.
* **start:** Starting node (tuple).
* **goals:** Set of goal nodes.
* **traps:** Set of trap nodes.
* **heuristic(node, goals):** Calculates the heuristic cost (Euclidean distance) from the current node to the nearest goal node.
* **node:** The current node.
* **goals:** Set of goal nodes.



Iterative Deepening Search Algorithm

In the context of maze navigation, the Iterative Deepening Search algorithm was implemented with these key components:

* **Trap Management**: The algorithm adds additional cost upon encountering a trap, but this cost is only considered when a successful path is found.
* **Depth-Limited Search**: The algorithm searches for a path at a certain depth and increases the depth for a new search if unsuccessful.
* **Pathfinding**: The algorithm traces a path from the start node to the goal, continuously moving towards the node that appears closest to the goal.

**Fields:**

* **graph:** A dictionary representing the maze, where keys are nodes (coordinates) and values are lists of neighboring nodes.
* **goals:** A set of tuples representing the goal nodes in the maze.
* **traps**: A set of tuples representing the trap nodes in the maze.

**Methods:**

**iterative\_deepening\_search**(graph, start, goals, traps): The main function to execute the Iterative Deepening Search algorithm, taking the maze graph, starting node, set of goal nodes, and set of trap nodes as inputs.

ekran görüntüsü, yazı tipi, siyah içeren bir resim

Açıklama otomatik olarak oluşturuldu

Uniform Cost Search Algorithm

The Uniform Cost Search algorithm, applied in a maze context, is structured as follows:

* **Trap Handling:** Adds an additional cost for traversing traps, yet maintains uniform cost for regular steps.
* **Node Selection:** Chooses the next node based on the lowest cumulative cost from the start node, incorporating additional costs for traps.
* **Pathfinding:** Traces the least expensive path to a goal, considering both regular moves and trap navigations.

**Fields:**

* **graph**: Represents the maze, with nodes (coordinates) as keys and lists of neighboring nodes as values.
* **goals**: Goal nodes in the maze.
* **traps**: Trap nodes in the maze.

**Methods:**

* **uniform\_cost\_search(graph, start, goals, traps)**: Main function executing the algorithm, takes the maze graph, start, goals, and traps as inputs, and returns the cost, path, and expanded nodes.

This algorithm effectively finds the lowest cost path, balancing the need to avoid traps with reaching the goal efficiently.

