Technical Explanation

1. Advanced Frontier Selection Strategy

My solution is based on a Multi-Criteria, Prioritized Heuristic Frontier Selection Algorithm. I designed a four-level decision model implemented as an efficient scoring function.

Considered Factors

My algorithm comprehensively considers the following three types of key information:

1. **Direction & Efficiency:** To reduce unnecessary turning and energy consumption, the algorithm evaluates the position of each frontier relative to the robot's current heading.

Forward Projection Distance (get_heading_projection): Calculates the projection length of the frontier in the robot's forward direction. A larger value indicates the frontier is more directly "in front" of the robot, promoting straight-line movement.

Heading Alignment (get_heading_alignment): Calculates the cosine of the angle between the robot's heading and the direction to the frontier. A value closer to 1 indicates better alignment, meaning the robot requires less rotation.

2. **Goal Proximity:** To ensure the ultimate purpose of exploration is reaching the endpoint, the algorithm evaluates the relationship between each frontier and the maze's final goal (goal_cell).

Euclidean Distance to Goal (get_dist_to_goal): Calculates the straight-line distance from the frontier to the goal. Selecting frontiers closer to the goal ensures the exploration maintains the correct general direction.

3. **Path Cost:** To select the most easily reachable target, the algorithm uses the path distance information provided by the detect_frontiers function.

Path Distance to Robot (distances[cell]): This is the actual path length from the robot's current position to that frontier (based on the BFS algorithm). This is a key metric for measuring "accessibility".

Scoring Function Design

To integrate the above factors into a unified decision-making framework, I utilized Python's max() function and the characteristics of tuple comparison to design a scoring function (key) with clear priorities.

The decision logic of this scoring tuple is as follows:

1. **Highest Priority: Forward Projection Distance.** The algorithm first selects the frontier farthest directly in front of the robot to maximally encourage efficient

straight-line movement.

2. **Second Priority: Heading Alignment.** If multiple frontiers have similar projection distances, the algorithm selects the one whose direction aligns best with the robot's current heading to minimize the time and energy consumed by rotating in place.

3. **Third Priority: Goal Proximity.** When the first two criteria cannot distinguish between options, the algorithm selects the frontier closer to the final goal, ensuring the exploration remains "goal-oriented". (Note: The negative sign - is used because the max() function selects the largest value, and we want the distance to be minimized).

4. **Lowest Priority: Path Cost.** As the final tie-breaker, the algorithm selects the frontier with the shortest path from the robot's current position, ensuring the chosen target is the "easiest to reach".

2. Testing & Challenges

During the process of writing the function, I primarily encountered issues where **the robot became "indecisive" or "stuck" in specific terrains**:

Cause Analysis: Initially, the algorithm lacked robustness, making it prone to crashes. Also, my initial algorithm logic was too simplistic (e.g., only considering the nearest or farthest frontier). When the robot needed to take a longer path to reach a better exploration area, it would choose a suboptimal target right in front of it, thus falling into a local optimum and even stalling in front of "U"-shaped obstacles.

Solution: Introduced the multi-criteria, prioritized scoring system. By comprehensively considering direction, efficiency, goal, and cost, the robot can make more holistic "trade-offs," enabling correct long-term decisions even if it means temporarily moving away from the goal to bypass large obstacles. Simultaneously, the phased filtering strategy ensures the robot's behavior remains natural and efficient across different environments.

Through repeated testing in both "Random" and "Snake" maze layouts, this algorithm demonstrated stable and efficient exploration behavior. The robot could smoothly navigate narrow passages, intelligently choose exploration branches, and consistently progress steadily towards the final goal, ultimately successfully completing the task.