

State Space Representation and Implementation Choices

In this project, the agent (mouse) must navigate a maze to collect all rewards and find the optimal path using various search algorithms. The maze is represented as a 2D grid where:

- `P` denotes the starting position of the agent.
- `.` represents rewards the agent must collect.
- `%` marks walls that the agent cannot pass.
- Spaces `()` are open paths the agent can traverse.

State Representation

Each state in the search space is defined by:

- `position`: A tuple `(x, y)` representing the agent's location.
- `remaining_rewards`: A frozenset of `(x, y)` coordinates representing prizes that have not been collected yet.

This design, in my opinion, allows efficient tracking of the agent's progress and enables comparison between states for optimal pathfinding.

Transition Model

The agent can move in four directions: **North, East, West, and South**, (best remembered as NEWS :) as long as the next position is not a wall (`%`). The function `get_neighbors(state, maze)` ensures that only valid moves are considered.

Goal Test

A state is considered a goal if there are no remaining rewards:

```
if len(state.remaining_rewards) == 0:  
    return True
```

This ensures the algorithm stops once all rewards have been collected. We don't want the mouse to get after all.

Description of Search Algorithms

Depth-First Search (DFS)

- Uses a **stack (LIFO)** to explore the deepest paths first.
- Does not guarantee the shortest path.
- Can get stuck in deep paths before reaching the goal.
- Works well for **small mazes** but is inefficient for large mazes with multiple rewards.

Breadth-First Search (BFS)

- Uses a **queue (FIFO)** to explore paths layer by layer.
- Guarantees the shortest path in terms of the number of steps.
- Expands many nodes, leading to high memory usage.
- More efficient than DFS for **finding the shortest path**.

Greedy Best-First Search (GBFS)

- Uses a **priority queue** sorted by the heuristic function.
- The heuristic used: **Manhattan distance** to the nearest reward.

- Often faster than BFS but does not guarantee the optimal path.
- Can take suboptimal routes due to its greedy nature.

A Search (A)*

- Uses both the cost so far (**g**) and the heuristic (**h**):

$$f(n) = g(n) + h(n)$$

- Guarantees the optimal path.
- The heuristic used: **Sum of Manhattan distances to all remaining rewards.**
- More efficient than BFS and DFS.

Performance Comparison

After many simulations, and asking AI agents to generate more mazes to test with:

The following table compares the performance of each algorithm on different maze sizes. The metrics used are:

- **Path Cost:** Number of steps taken.
- **Nodes Expanded:** How many states were explored.
- **Runtime:** Execution time in milliseconds.

Algorithm	Path Cost	Nodes Expanded	Runtime (ms)
DFS	Varies	High	High
BFS	Optimal	Very High	Moderate
GBFS	Suboptimal	Low	Fast
A*	Optimal	Low-Medium	Fast

Challenges Faced

1. Reward Collection Bug

- Initially, the agent was unable to recognize when it had collected a reward, causing an infinite loop, making my laptop freeze many times in the process.
- The issue was resolved by correctly updating `remaining_rewards` when reaching a reward position.

Comparing State Objects in `heapq`

- Python's priority queue (`heapq`) requires elements to be comparable.
- A `__lt__()` method was added to the `State` class to enable state comparisons.

Handling Large Mazes Efficiently

- BFS expanded too many nodes, making it impractical for large mazes.
- A* with an effective heuristic significantly reduced search space and computation time.

Final Observations

- **DFS** is unpredictable and inefficient for large mazes.
- **BFS** guarantees the shortest path but is computationally expensive.
- **GBFS** is faster but does not guarantee an optimal solution.
- **A*** is the most efficient approach, balancing **speed and optimality**. I am yet to test random heuristics though, because I'm sure a good heuristic is what made it optimal.

Conclusion

The A algorithm is the *best choice* for solving this problem efficiently, because it guarantees the shortest path while expanding fewer nodes than BFS. However, for smaller mazes where performance is not an issue, BFS may still be an acceptable choice due to its simplicity.

Works Cited

Russell, Stuart J., and Peter Norvig. *Artificial intelligence: a modern approach*. pearson, 2016.