CSC140003 Introduction to Aritificial Intelligence

Delivery AI Project Report

Group 05

Group members

Phan Hai Minh - 22127273
Mai Duc Duy - 22127084
Nguyen Tran Minh Hoang - 22127130
Tran Thanh Long - 22127250

VIETNAM NATIONAL UNIVERSITY HO CHI MINH
UNIVERSITY OF SCIENCE
Faculty of Information Technology

Contents

I. Overview

Introduction

This document aims to report the algorithms, systems and testing involved in Project 1 of the Introduction to Artificial Intelligence course, also known as the Delivery AI project. The project is a simulation of a delivery agent navigating through a grid to deliver packages. On the grid with toll booths and fuel stations of various waiting time, the agent must search for a path that optimizes travelling distance, time while satisfying the fuel constraint.

The project is divided into four levels, each with increasing complexity. The first level is a simple grid with obstacles and no constraints. The second level introduces waiting toll booths with a time constraint on the agent. The third level further constrains the agent with a fuel limit and introduces fuel stations. The fourth level has multiple agents involved, each aims to optimize its own path.

Project assignment

In this section we will report the tasks involved in the project as well as the person responsible for each task.

Description	**Due Date**	**Responsibility**
Discuss the project requirements and divide the work among the team members.	08/07/2024	22127273
Implement Level 1	21/07/2024	22127273
Implement Level 2	21/07/2024	22127273
Implement Level 3	24/07/2024	22127273, 22127250
Implement Level 4	28/07/2024	
Construct & test GUI	24/07/2024	22127084, 22127130
Perform preliminary testing of the project to ensure the program operates stably, without conflicts or minor errors.	27/07/2024	22127084, 22127130, 22127250, 22127273
Discuss and propose suitable test cases (e.g., special cases, larger input data, etc.).	27/07/2024	22127084, 22127130, 22127273
Prepare the report document and complete it.	27/07/2024	22127250
Create a demo video	28/07/2024	22127084, 22127273
Submission	28/07/2024	22127273

Table 1: Project assignment tasks and responsibilities

Self-evaluation

In this section we will self-evaluate our completion status for each of the required task given in the project.

Description	**Percentage of completion**
Finish Level 1 successfully	100%
Finish Level 2 successfully	100%
Finish Level 3 successfully	100%
Finish Level 4 successfully	0%
Graphical User Interface (GUI)	100%
Generate at least 5 test cases for each level with different attributes. Describe them in the experiment section of your report. Videos to demonstrate each testcase	100%
Report your algorithm, experiment with some reflection or comments.	100%

Table 2: Completion status of tasks

II. Algorithm report

Level 1

This level requires the agent to find a path from the Start node S to the goal G with the minimum amount of cells. The agent can use several strategies to search for the optimal path, 5 of which will be reported are Breadth-first search, Depth-first search, Uniform Cost search, Greedy Best-first search and A* algorithm.

Best-first search scheme

The above 5 searching algorithms uses a same strategy (or scheme) of searching called the Best-first search scheme. The Best-first search is generic search strategy that prioritizes nodes based on an evaluation. Each of the 5 algorithms actually implements the Best-first search with a differnt prioritization function.

Specifically, the Best-first search uses a priority queue structure to arrange better evaluated nodes closer to the front. On each iteration while the queue is not empty, the search examines the foremost node in the queue then add its neighbors to the queue for later exploration. While exploring the algorithm also keeps track of the visited nodes, marking each of the nodes when it is examined. This way, when a node is revisited, it will not be explored again. The search can either check for goal state when a node is examined (a Late Goal Test) or when a node is reached from its neighbor (an Early Goal Test).

The priority queue is sorted based on the evaluation function of each algorithm. Using this approach, the most optimized nodes are explored first, hence when the goal is reached it will be reached from the most optimized path. The Best-first search can be briefly demonstrated with this pseudocode snippet:

```
def best_first_search(graph, start, goal, evaluate):
  frontier = [(evaluate(start), start, [start])]
  visited = set()

  while frontier:
      priority, current, path = frontier.pop()

      if current == goal: return path
      elif current in visited: continue

      visited.add(current)

      for neighbor, cost in graph[current]:
          if neighbor not in visited:
              new_priority = cost(neighbor) + heuristic(neighbor)
              new_path = path + [neighbor]
              frontier.push([(new_priority, neighbor, new_path)])

  return None
```

In this project context, the S and G nodes are a pair of coordinates on the grid. To make the code more concise, a `Node` class can be defined instead of using tuple to represent the nodes on the grid with the coordinates as well as packing in its evaluation value and path:

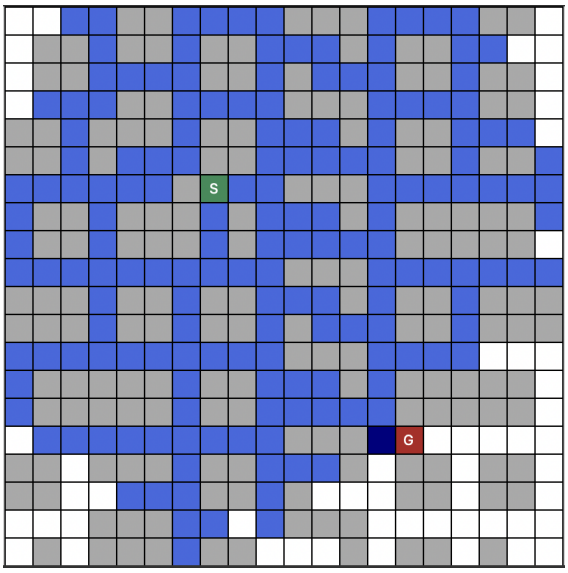
```
class Node:
    def __init__(self, x, y, evaluation, path):
        self.x = x
        self.y = y
        self.evaluation = evaluation
        self.path = path
```

Breadth-first search

The Breadth-first search (BFS) is a search algorithm that explores the nodes in a level-by-level manner. The algorithm explores nodes with the shallowest depth first, then uses an early-goal test to check if the neighbors are the goal. Formally, the BFS algorithm implements the Best-first search with the following function:

$$f(n) = \text{depth}(n)$$

Visually, the algorithm searches for the closer surrounding nodes from the start first then slowly expanding outwards:



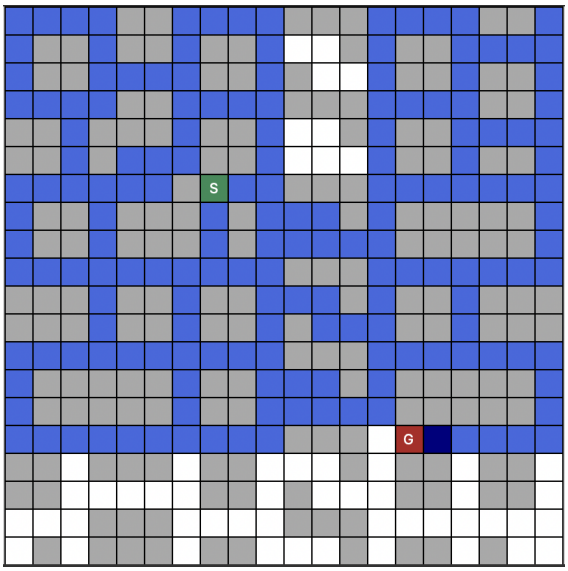
Expanded cells of the BFS.

Depth-first search

The Depth-first search is a search algorithm that explores the nodes in a depth-first manner. The algorithm explores nodes with the deepest depth first, then uses an early-goal test to check if the neighbors are the goal. Formally, the DFS algorithm implements the Best-first search with the following function:

$$f(n) = -\text{depth}(n)$$

Visually, the algorithm seems to wander randomly as it dives into the deepest nodes first:



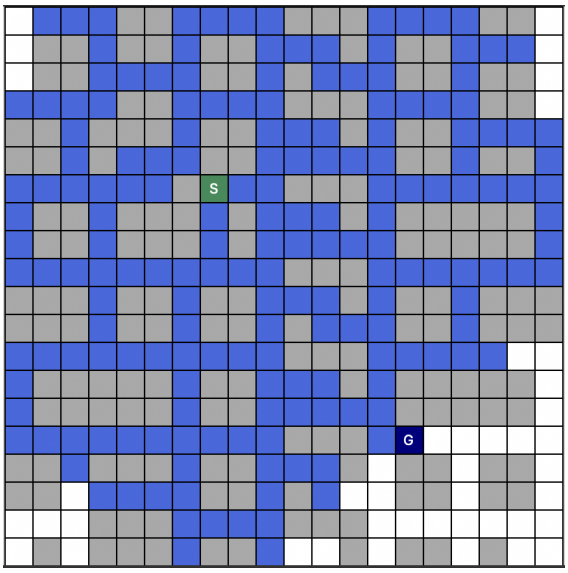
Expanded cells of the DFS.

Uniform Cost search

The Uniform-cost search is a search algorithm that explores the nodes uniformly based on the path cost. Unlike other searches, UCS must use a Late Goal Test since a node reached by its neighbor is not guaranteed to be optimal. Formally, the UCS algorithm implements the Best-first search with the following function:

$$f(n) = \text{cost\_from\_root}(n)$$

As in Level 1 all path across cells have the same cost of 1, the UCS is visually similar to the BFS except for the larger amount of cells expanded:



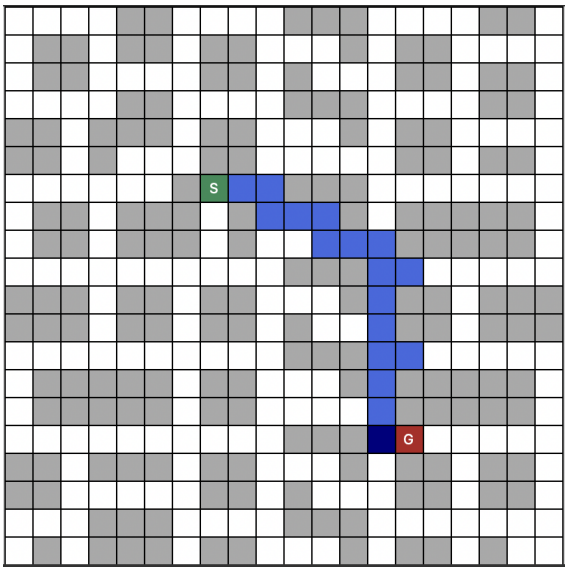
Expanded cells of the UCS. Notice how the Late Goal Test is applied here as G is highlighted.

Greedy Best-first search

Greedy best-first search is a search algorithm that explores the nodes entirely based on the heuristic function. This algorithm belongs to a search class called **Informed search** where the search is guided (or *informed*) by a heuristic function estimating how far the node is from G. GBFS explores nodes with the lowest heuristic value first, then test the node for goal when it is visited (Early Goal Test). Formally, the GBFS algorithm implements the Best-first search with the following function:

$$f(n) = \text{heuristic}(n)$$

Visually, the algorithm seems to magically head towards the path without redundant exploration, since a good heuristic is used here:



Expanded cells of the GBFS.

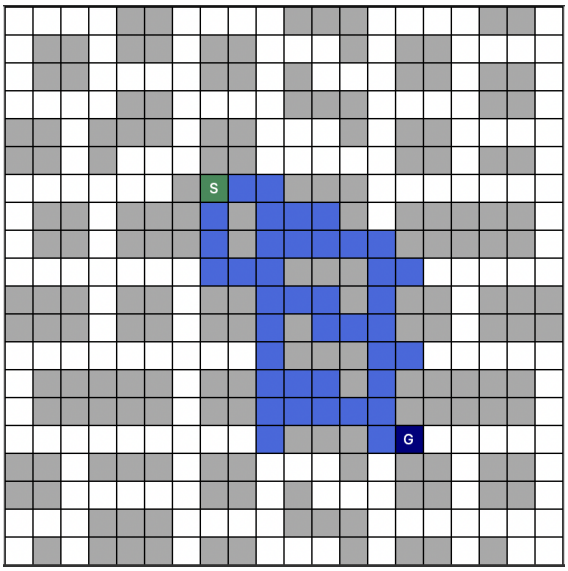
Do note that, while the GBFS is very efficient in this case, it is not guaranteed to find the optimal path. If the heuristic is bad or even misleading, the algorithm may end up with an extremely suboptimal path.

A* search

A* search is a search algorithm that explores the nodes based on the sum of the path cost combined the heuristic value. This algorithm is a combination of UCS and GBFS, where the search is guided by both the path cost and the heuristic function. A* explores nodes with the lowest sum of the path cost and the heuristic value first, then test the node with the Late Goal Test. Formally, the A* algorithm implements the Best-first search with the following function:

$$f(n) = \text{heuristic}(n) + \text{cost\_from\_root}(n)$$

Visually, the algorithm first explores nodes surrounding S then gradually streamlines towards G:



Expanded cells of the A*.

Level 2

Additionally to Level 1 where the agent has to find a path with the minimum distance, Level 2 introduces a new time constraint for the agent and toll booths of various waiting time. The agent must now find a path that both minimizes the distance satisfying the total waiting time constraint (but not necessarily minimizing it).

Realizing the ability to find optimal path with efficiency of the A* algorithm, we decided to use the A* algorithm as the base for further modification in our Level 2 algorithm. We modified the `Node` structure of the search to include the total time passed as well with the evaluation value and path:

```
class Node:
    def __init__(self, x, y, evaluation, path, time):
        self.x = x
        self.y = y
        self.evaluation = evaluation
        self.path = path
        self.time = time
```

The evaluation function of the A* algorithm is then modified to include the time constraint. The function evaluates the nodes based on the original A* evaluation function but when the function gives a tie, the node with the least time passed is prioritized:

```
def compare(u: Node, v: Node):
  u_sum = u._cost + u._heuristic
  v_sum = v._cost + v._heuristic

  if u_sum != v_sum: return u_sum < v_sum
  return u.time < v.time
```

It turns out that due to the Best-first nature of the node expanding process, this modification alone is enough to ensure the optimality of the search. In this problem, the total time is strictly increasing as the agent moves, so the node with lower time passed is more highly prioritized in the queue. Since A* always treats the first expanded instance of a node as optimal, the agent will always find the optimal distance path and among the optimal distance paths, the one with the least time passed.

Level 3

Level 3 introduces yet another new constraint for the agent: a fuel limit. There are also fuel stations scattered across the grid where the agent can refuel with a certain amount of waiting time. The agent must now find a path that minimizes the distance, satisfies the total waiting time constraint and the fuel constraint by making sure that the fuel level never drops below zero throughout the path.

To solve this problem we continue to ultilize the modified A* algorithm from Level 2. We further modify the `Node` structure to include the fuel level as well:

```
class Node:
    def __init__(self, x, y, evaluation, path, time, fuel):
        self.x = x
        self.y = y
        self.evaluation = evaluation
        self.path = path
        self.time = time
        self.fuel = fuel
```

We first tried to naively maximize the fuel level when all other evaluations tied in the prioritization function, but this modification is turned out to be not optimal. Since the fuel level variates throughout the path, a situation failed where a cell with an optimal time but very little fuel left is expanded first. This state then does not have enough fuel to reach the goal, but since it is expanded first the later states of this cell while having more fuel are not explored.

The problem is then solved by instead of treating the fuel level as a value to evaluate, we treat it as a part of the state, then evaluate the remaining values as usual:

```
class Node:
    def __init__(self, x, y, evaluation, path, time, fuel):
        self.state = (x, y, fuel)
        self.evaluation = evaluation
        self.path = path
        self.time = time
        self.fuel = fuel
```

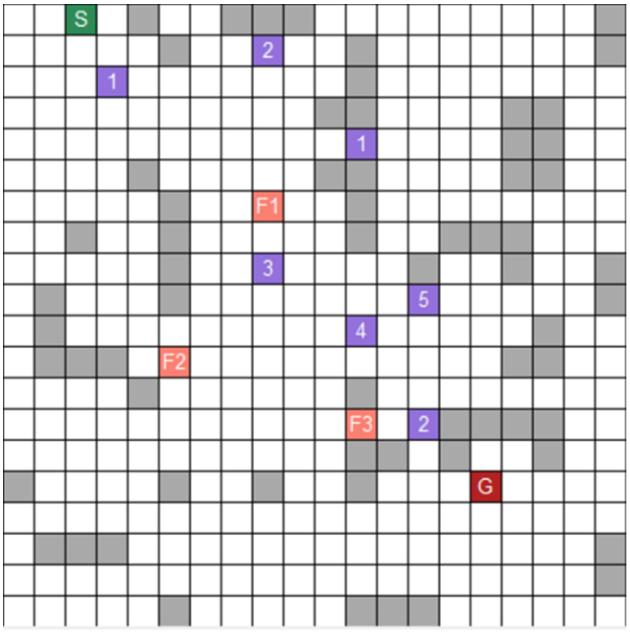
This way, states with the same cell but different levels of fuel are treated as different states, hence all of these states will be explored. Because of this modification, the search will now expand a cell multiple times making the algorithm more computationally expensive, but only to an extra order of magnitude.

Let's define the amount of nodes N and the maximal value of fuel F. The time complexity of the original A* algorithm is $O(N \log N)$. With the modification, for each node all levels of fuel are considered, hence the time complexity is now $O(NF \log NF)$.

III. Testing

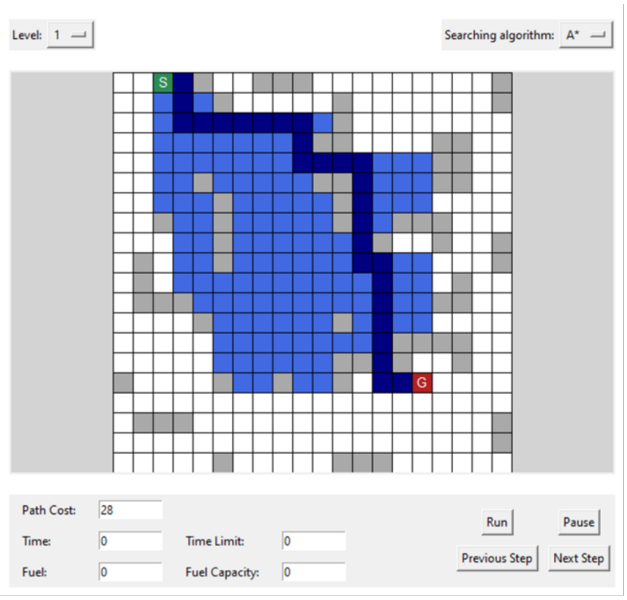
To make sure that the algorithms are implemented correctly, we have prepared a set of test cases for each level. The test cases are designed to cover a wide range of scenarios, from simple cases to more complex ones. The test cases are then run on the implemented algorithms to verify their correctness.

Test 1

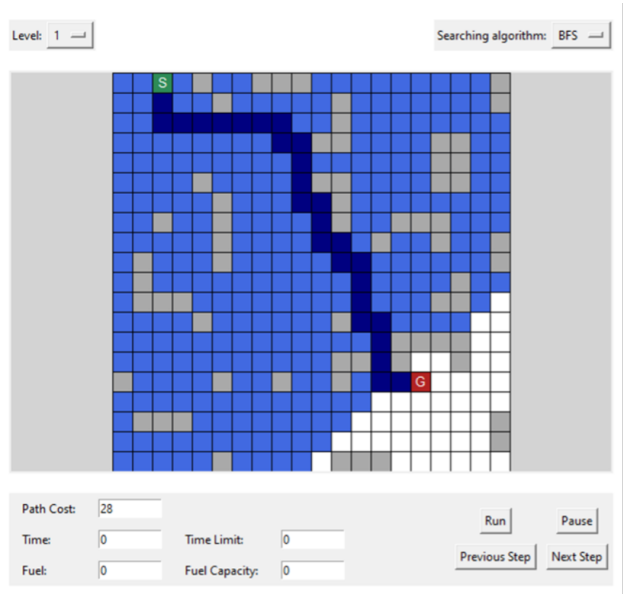


This is the map for this test case at level 3. This test case is saved in the file “input_1.txt”. With this test case, the start node is at cell (0, 2), and the goal node is at cell (15, 15). This case also has 7 toll booths and 3 gas stations. The time limit is 35 and the fuel limit is 15.

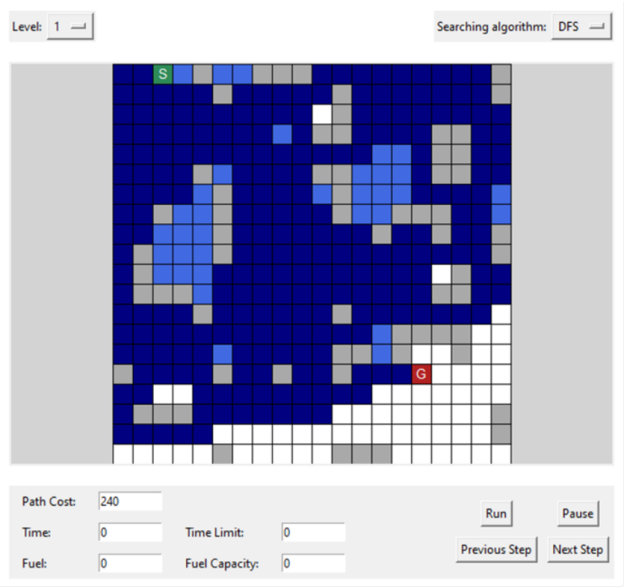
A* search



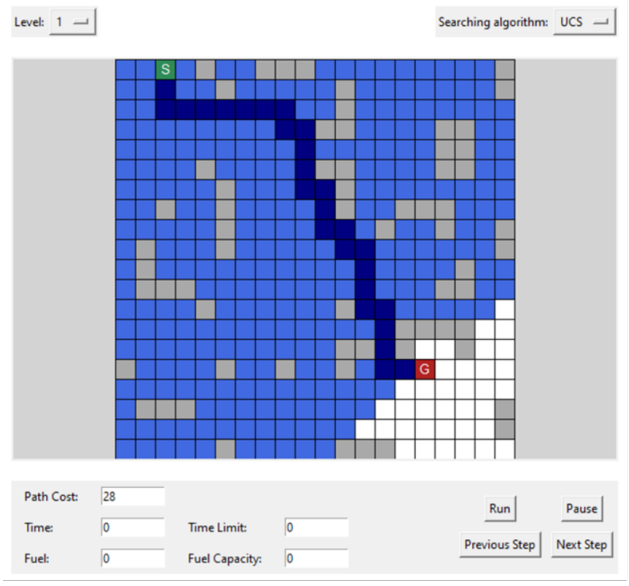
Breadth-first search



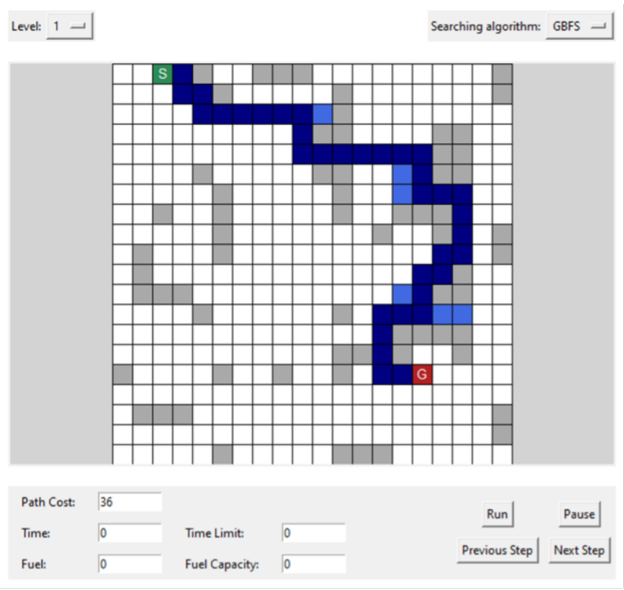
Depth-first search



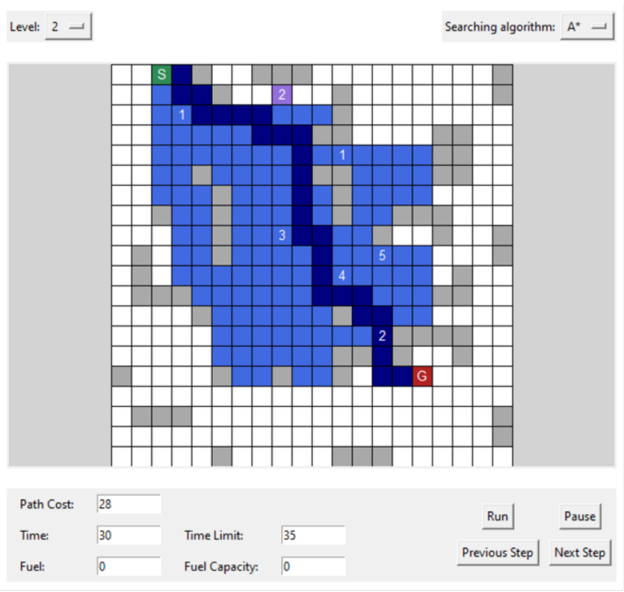
Uniform Cost search



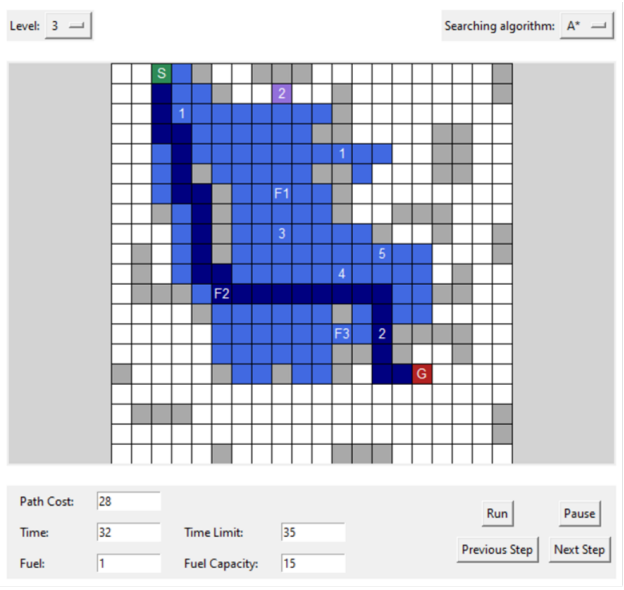
Greedy Best-first search



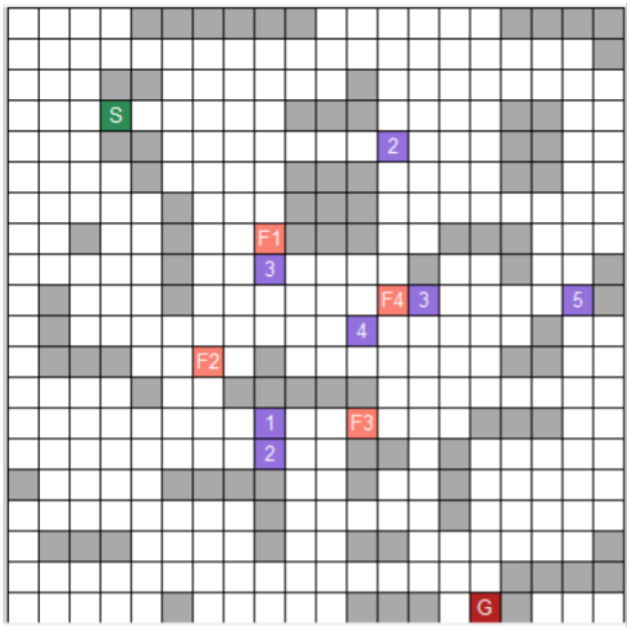
Level 2 search



Level 3 search

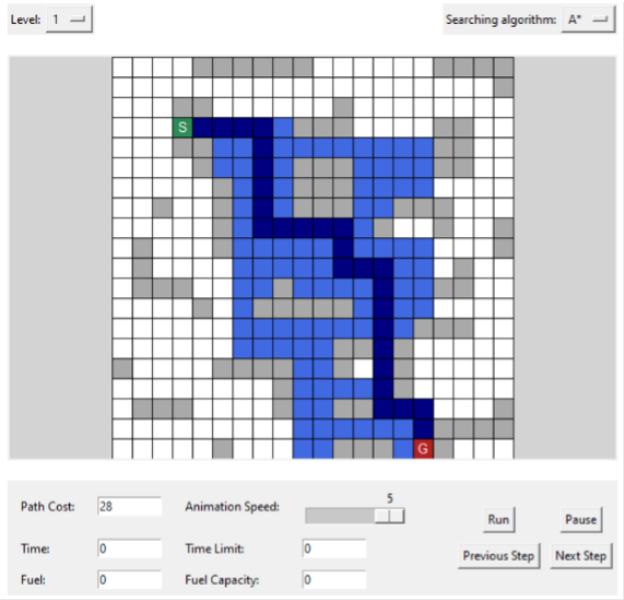


Test 2

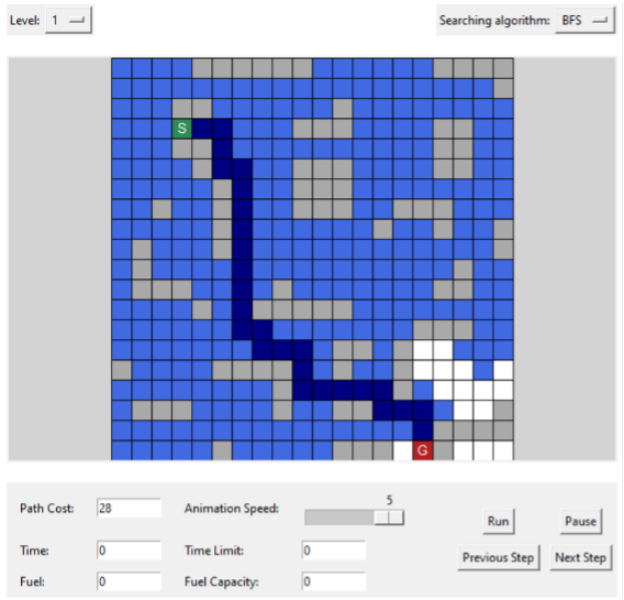


Here is the map at level 3 for test case 2. It is stored in the file "input_2.txt". With this test case, the start node is at cell (3, 3), and the goal node is at cell (19, 15). This case also has 7 toll booths and 4 gas stations. The time limit is 35 and the fuel limit is 18.

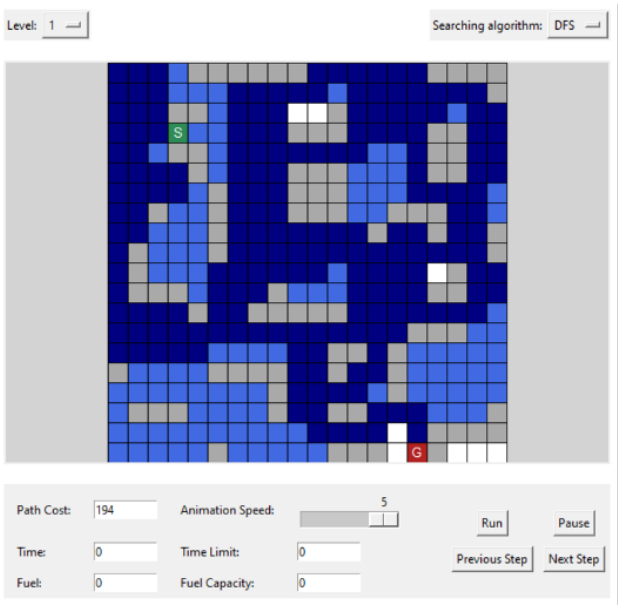
A* search



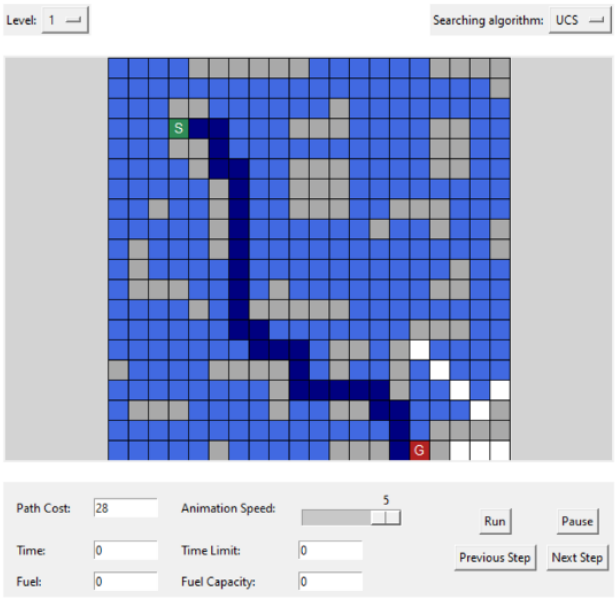
Breadth-first search



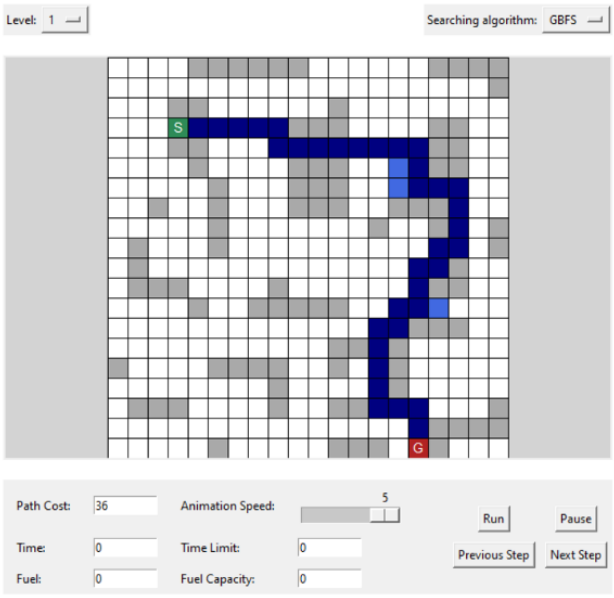
Depth-first search



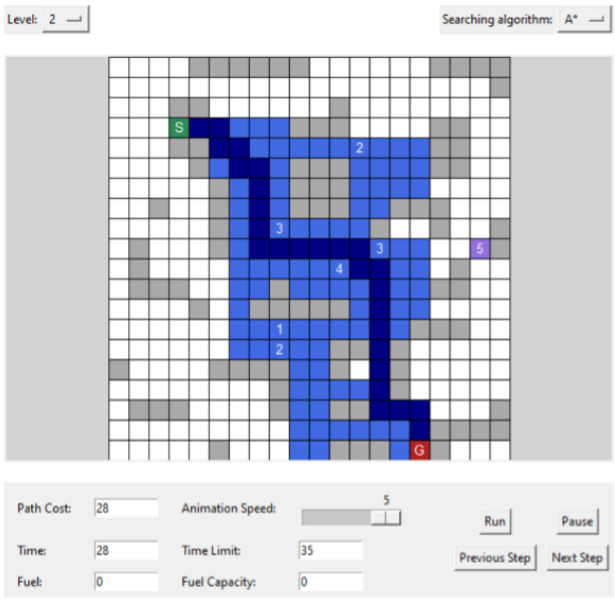
Uniform Cost search



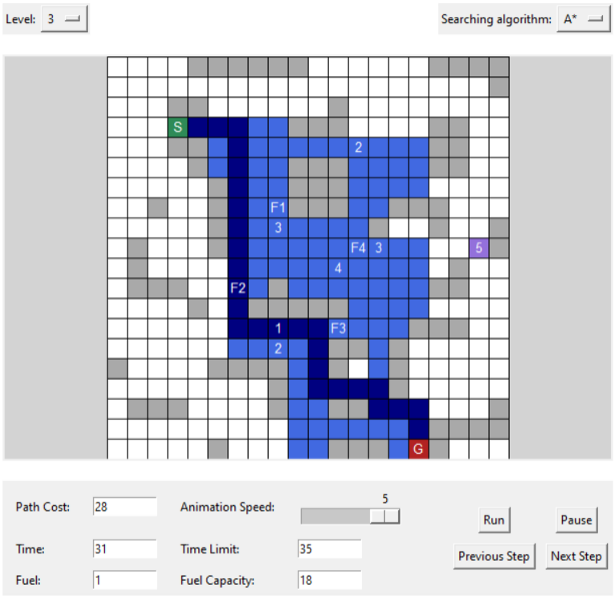
Greedy Best-first search



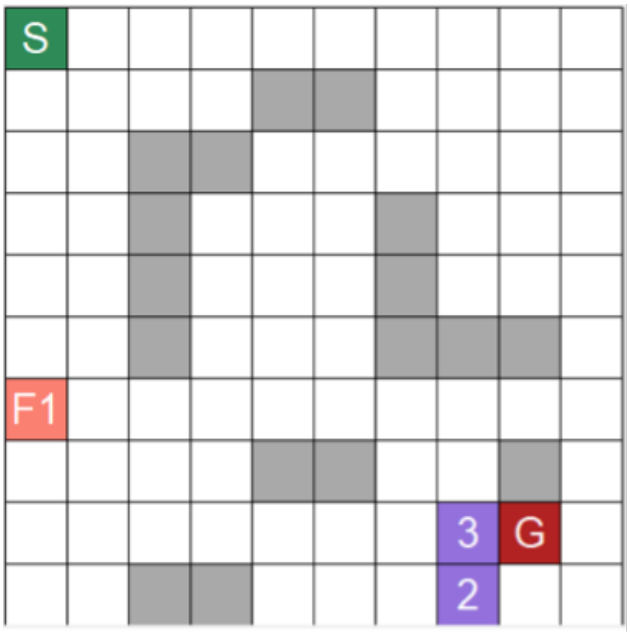
Level 2 search



Level 3 search

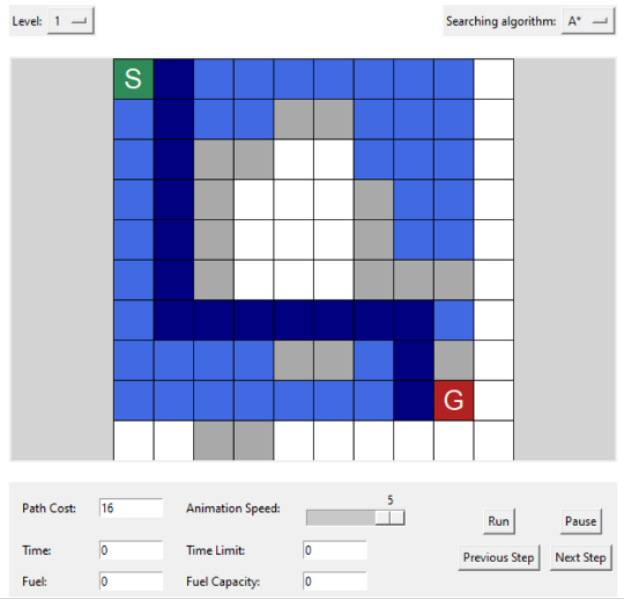


Test 3

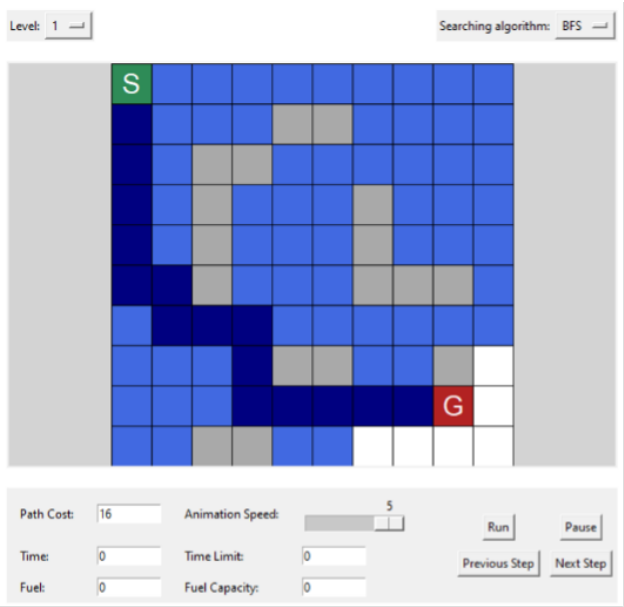


This is the map at level 3 for test case 3. It is stored in the file "input_3.txt". With this test case, the start node is at cell (0, 0), and the goal node is at cell (8, 8). This case also has 2 toll booths and 1 gas station. The time limit is 15 and the fuel limit is 8.

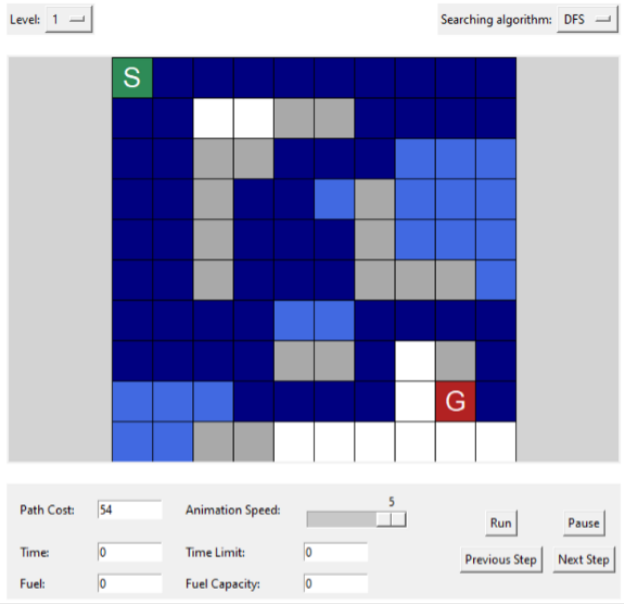
A* search



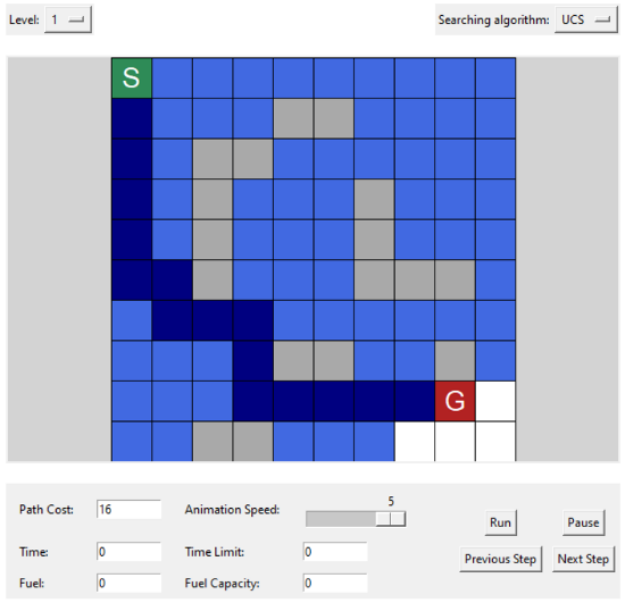
Breadth-first search



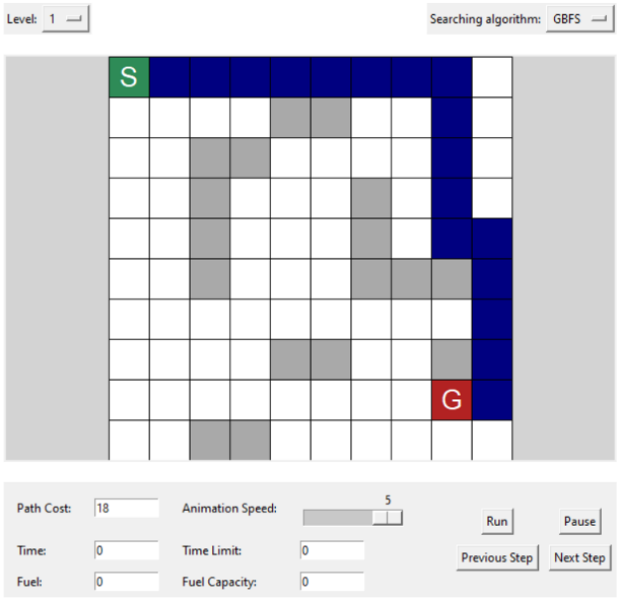
Depth-first search



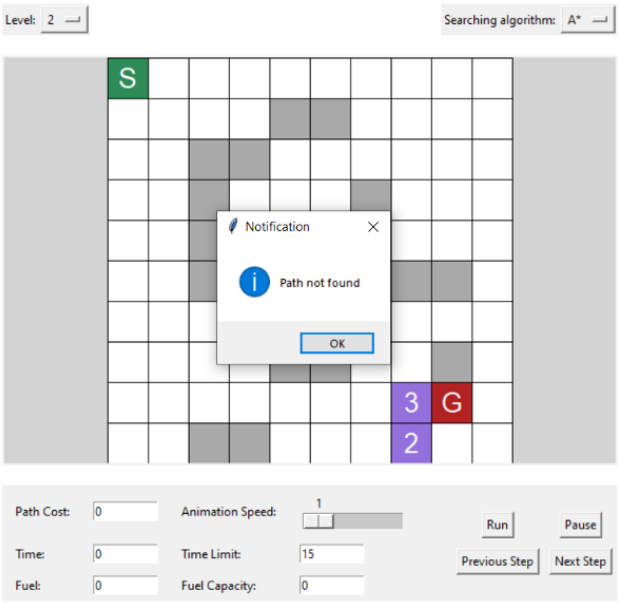
Uniform Cost search



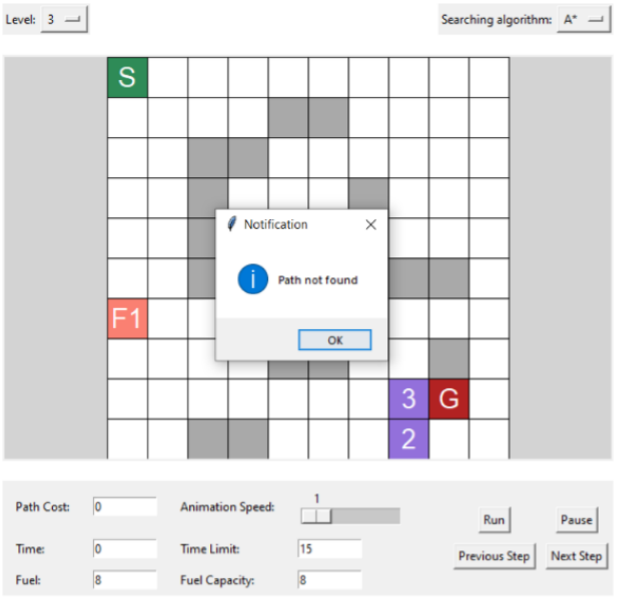
Greedy Best-first search



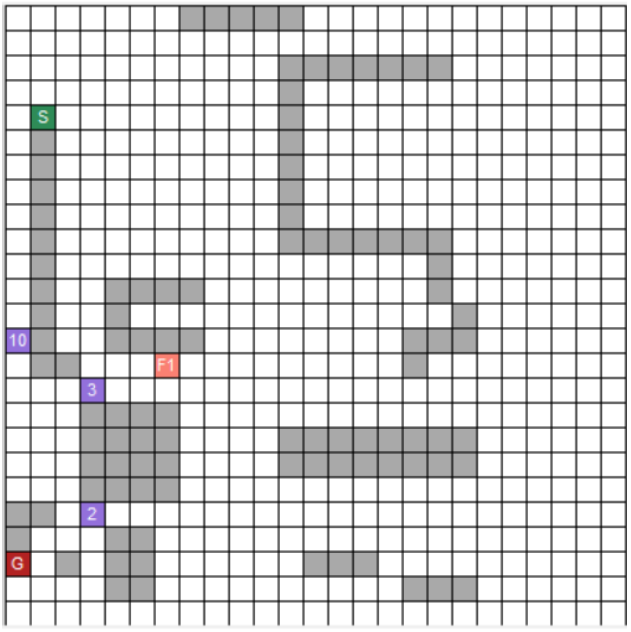
Level 2 search



Level 3 search

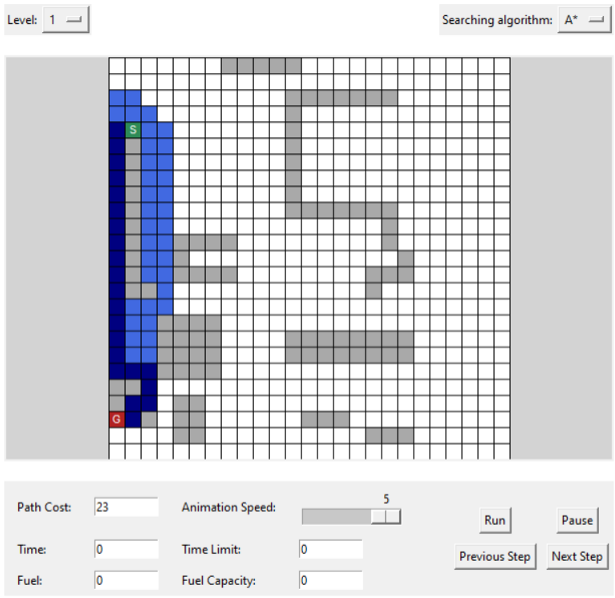


Test 4

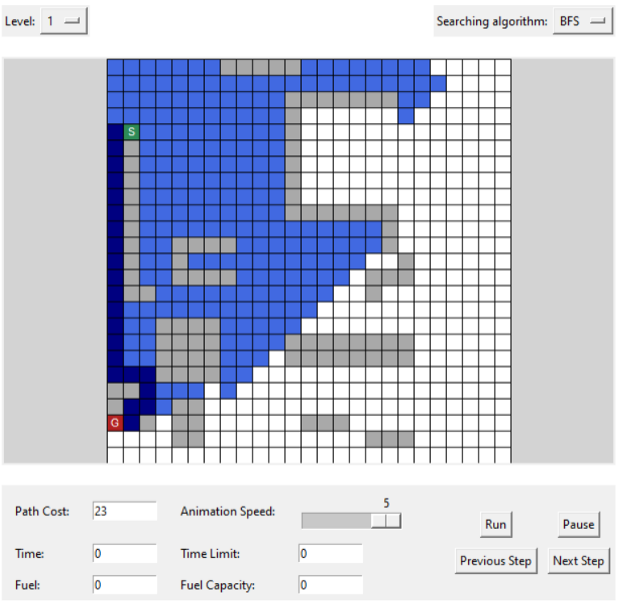


This is the map at level 3 for this test case. It is stored in the file “input_4.txt”. With this test case, the start node is at cell (4, 1), and the goal node is at cell (22, 0). This case also has 3 toll booths and 1 gas station. The time limit is 35 and the fuel limit is 20.

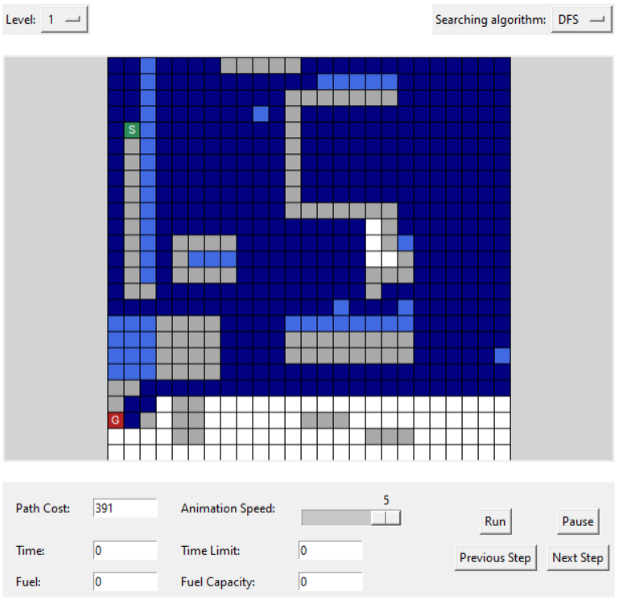
A* search



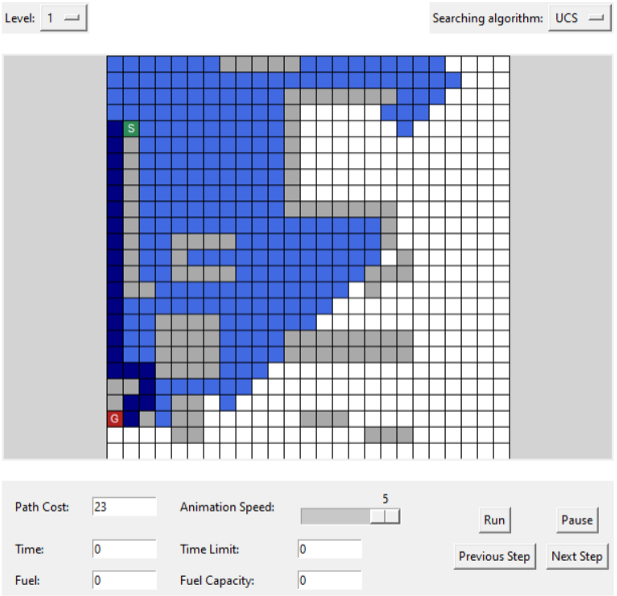
Breadth-first search



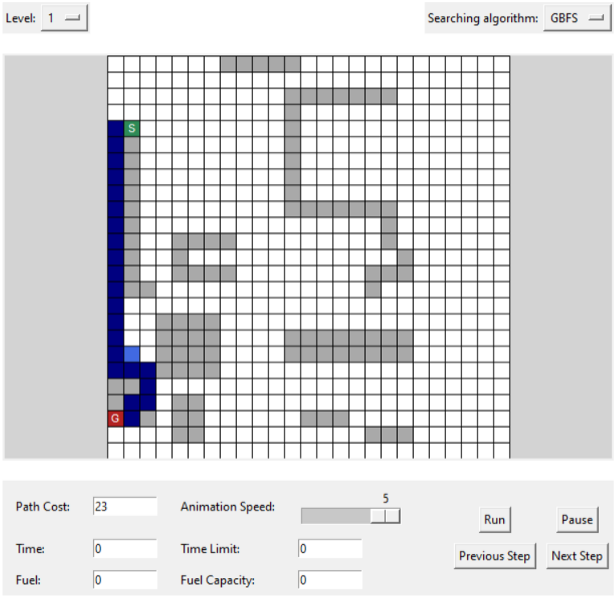
Depth-first search



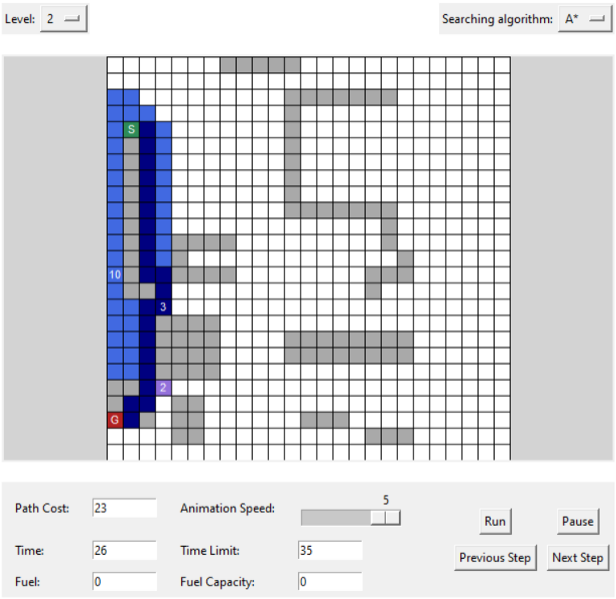
Uniform Cost search



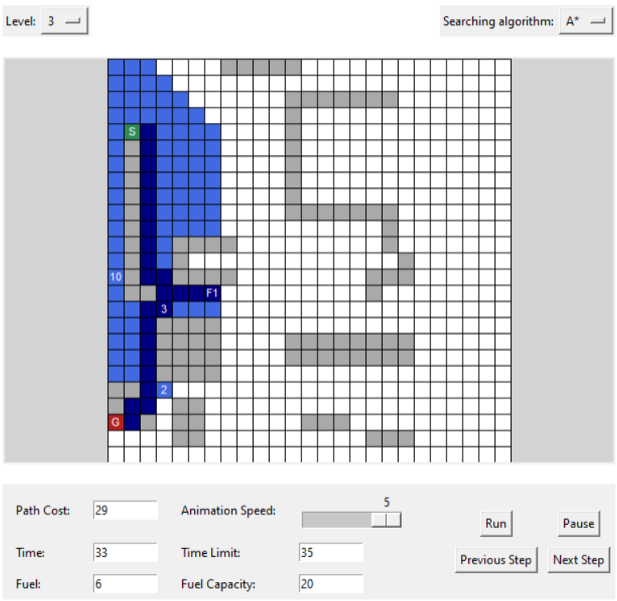
Greedy Best-first search



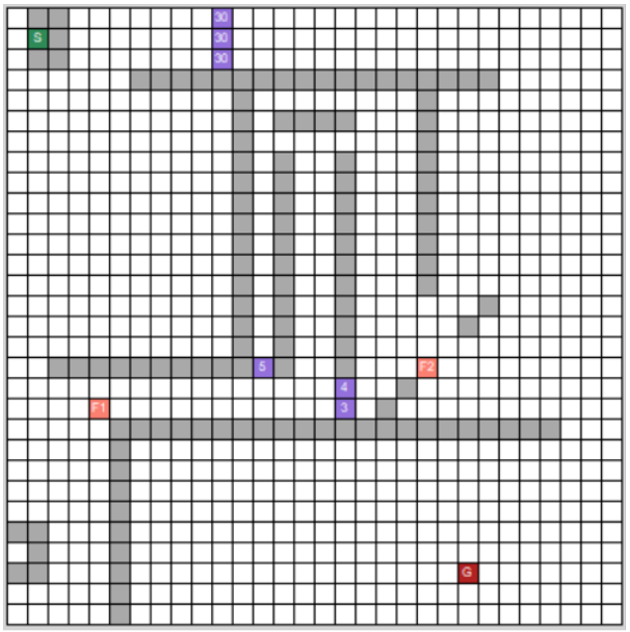
Level 2 search



Level 3 search

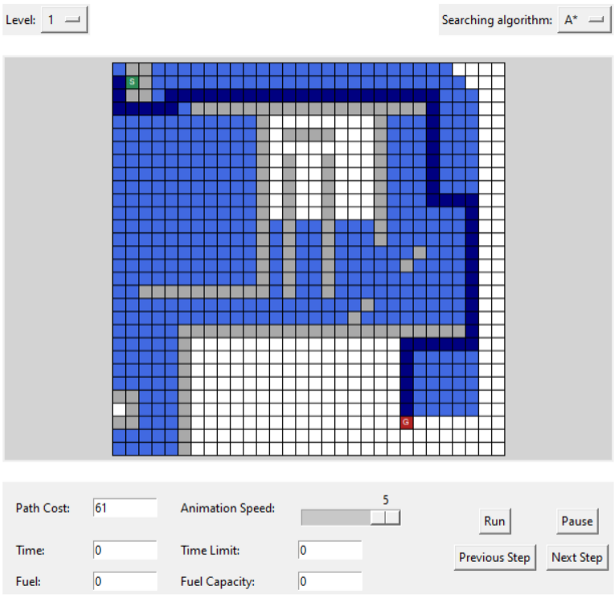


Test 5

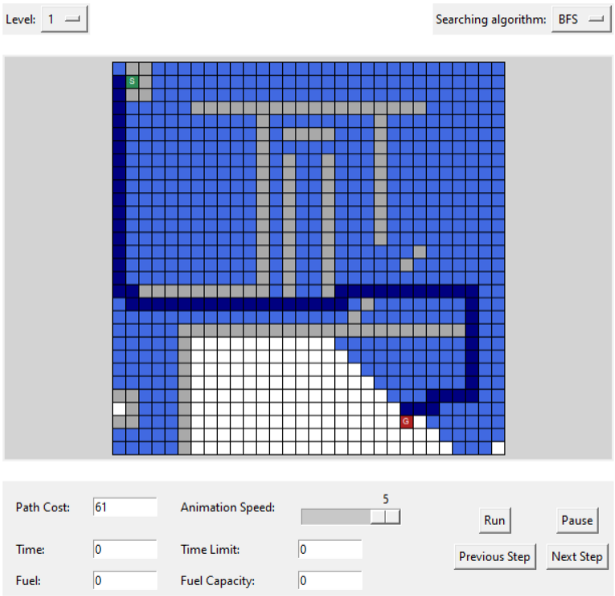


This is the map at level 3 for this test case. It is stored in the file “input_5.txt”. With this test case, the start node is at cell (1, 1), and the goal node is at cell (27, 22). This case also has 6 toll booths and 2 gas stations. The time limit is 80 and the fuel limit is 35.

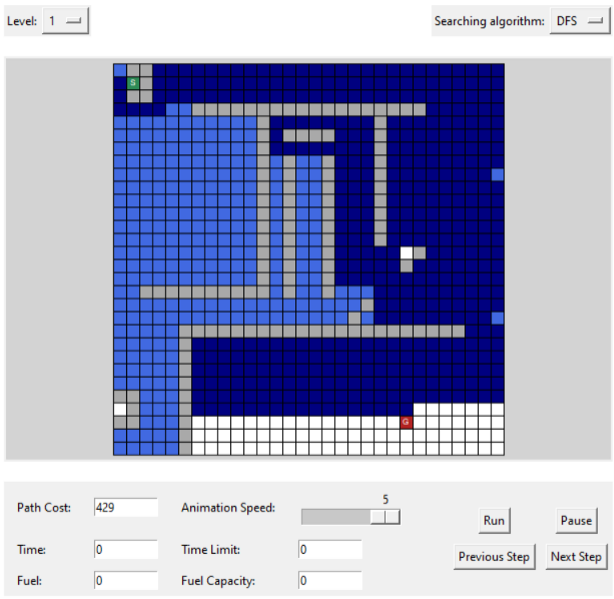
A* search



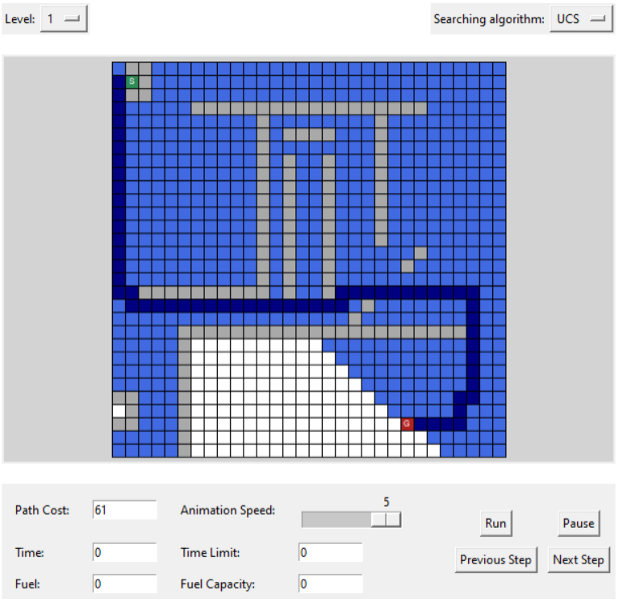
Breadth-first search



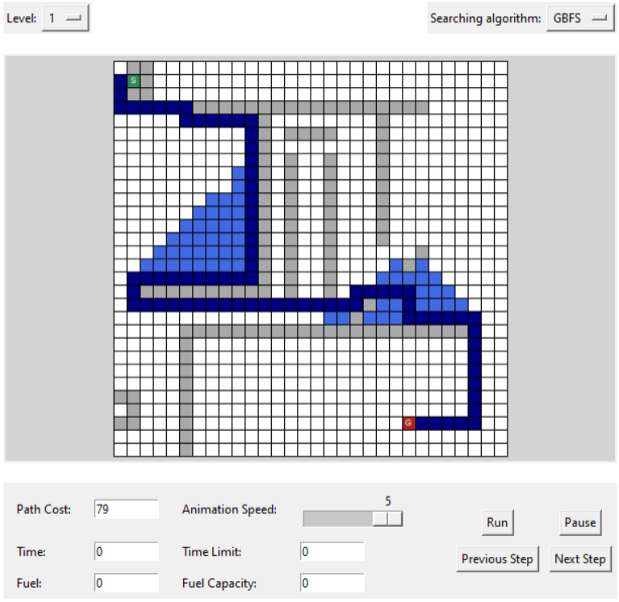
Depth-first search



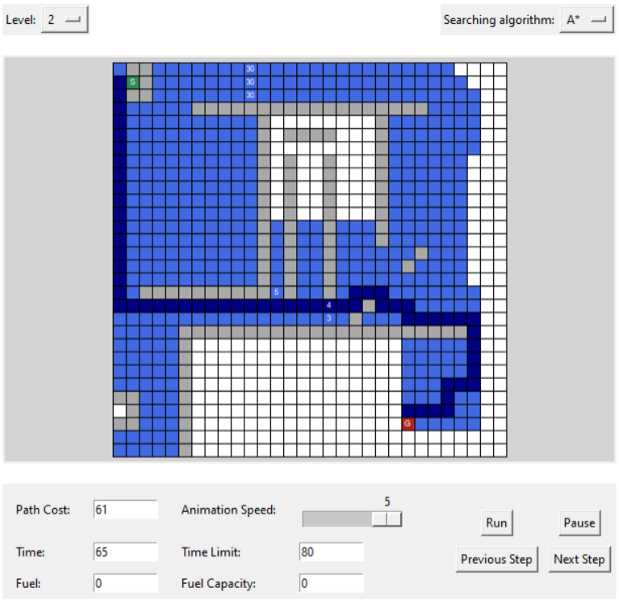
Uniform Cost search



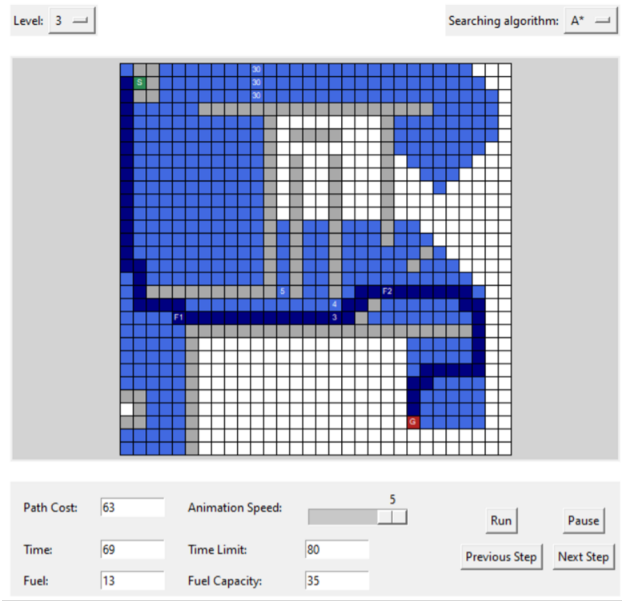
Greedy Best-first search



Level 2 search



Level 3 search



IV. Demonstration

A video of the application demo can be found here: https://www.youtube.com/watch?v=jYYZUnpD_8E