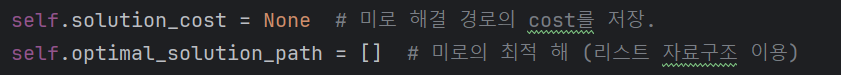
**HW #1**

**Name: 이세형**

**Student Number: 2020312145**

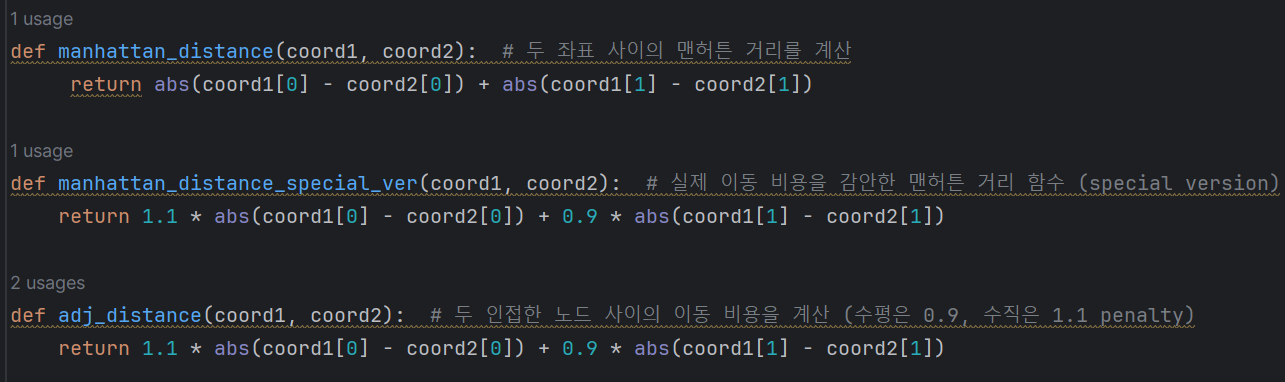
1. **Implementations:**

First, I added new two fields to class maze in maze.py.



‘solution\_cost’ filed stores the cost of optimal solution. And ‘optimal\_solution\_path’ field stores optimal solution path of maze.

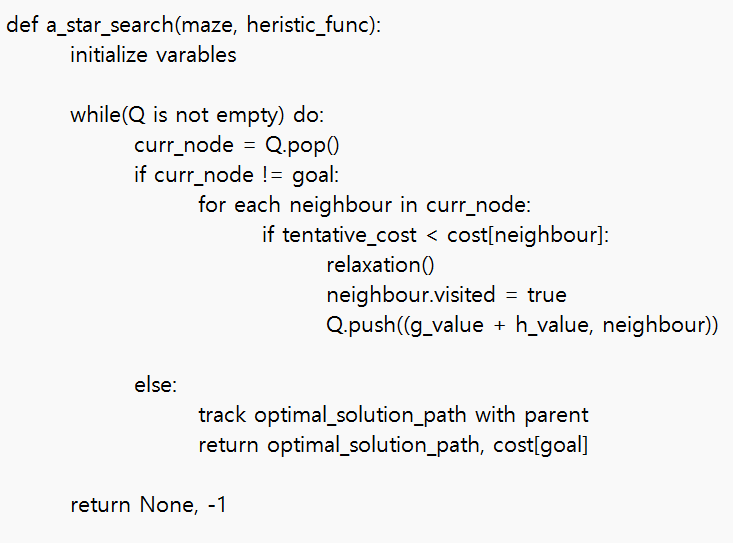
‘manhattan\_distance’, ‘manhattan\_distance\_special\_ver’, ‘adj\_distance’ functions are also implemented in maze\_manager.py

  
The 'manhattan\_distance' function returns the Manhattan distance between the two coordinates passed to it.   
The 'manhattan\_distance\_special\_ver' function returns the Manhattan distance between the two coordinates passed to it, taking into account that the cost of horizontal (0.9) and vertical (1.1) movements are different.  
The 'adj\_distance' function calculates and returns the movement distance cost between two adjacent nodes.

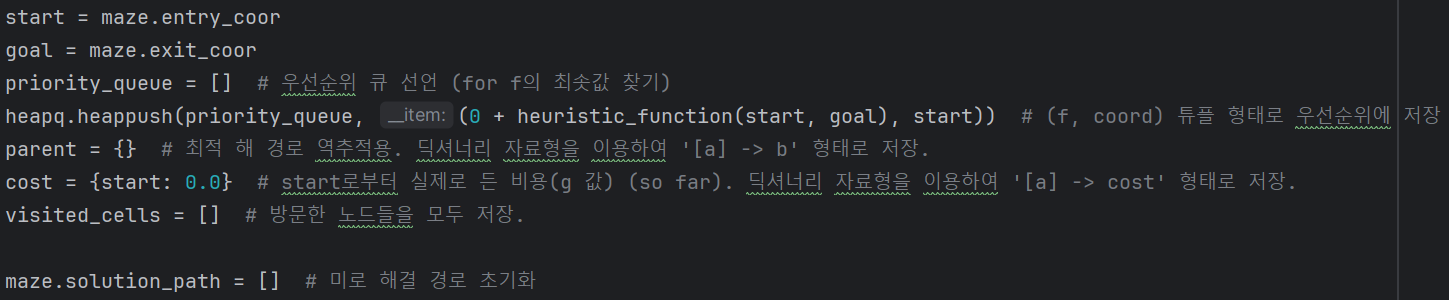
Detailed comments have been added to most lines of this code.

Below is detailed explanation about implementations of two functions.

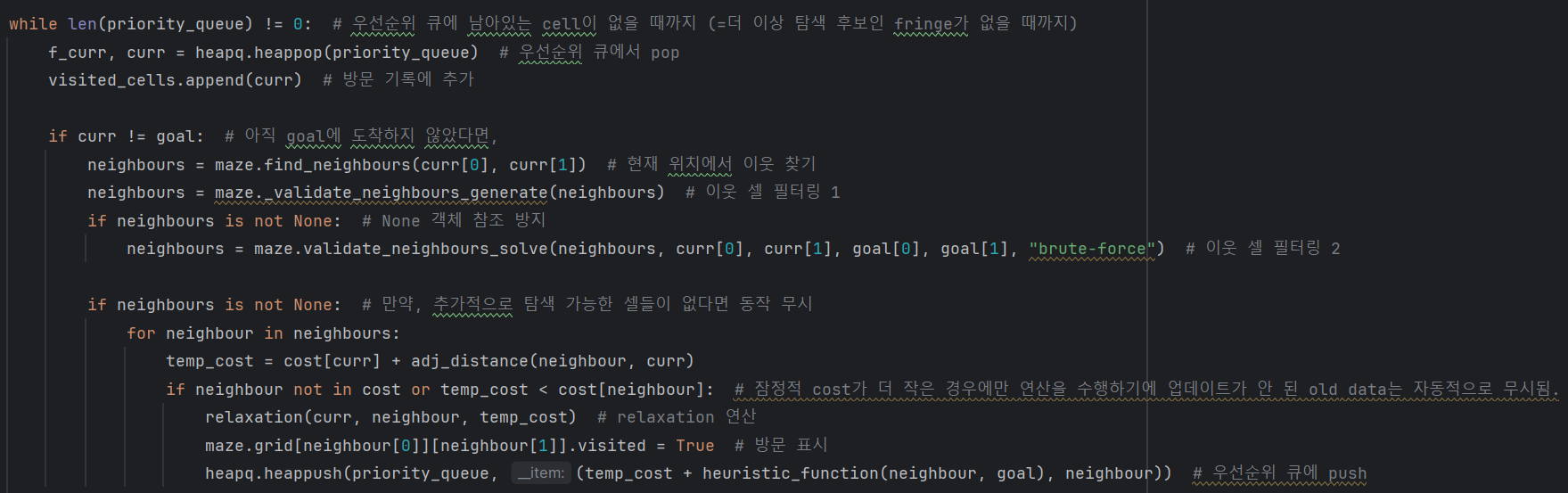
1. **A \* search algorithm:**  
   The 'a\_star\_search' function takes a maze instance and a heuristic function as inputs and returns the path and cost of the optimal solution. Below is a simplified pseudocode of the function.

****

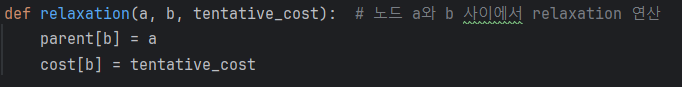
The local variables declared within the 'a\_star\_search' function are as follows.

****

The 'priority\_queue' variable is a priority queue that stores tuples in the form of (f value, coordinate). (cf. f value = g value + h value) The 'parent' variable is a dictionary data structure that stores information about the parent node of a given node. This variable is later used for backtracking the shortest path. 'Cost' is a data structure that stores the shortest path cost calculated so far from the start node to the given node. The 'visited\_cells' variable is a dictionary data structure that records the visit history of nodes.

****

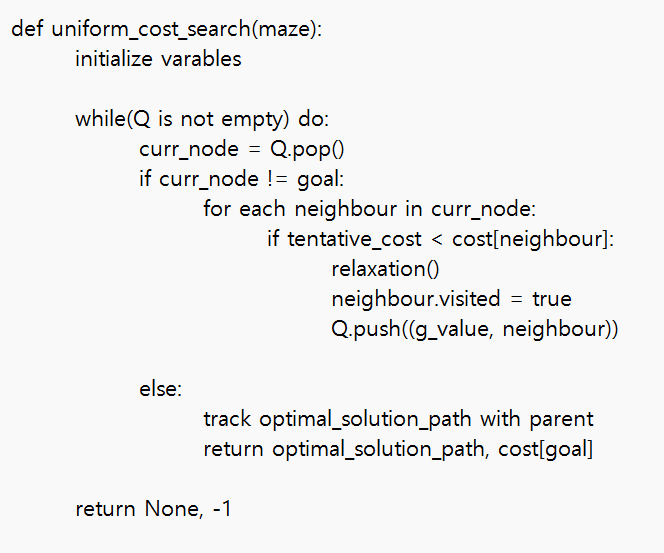
As mentioned in the pseudocode above, the search continues through neighboring nodes and performs relaxation operations until there are no more candidates in the 'priority\_queue'. For the relaxation operation, a separate function was defined within the 'a\_star\_search' function as follows. In this process, the 'manhattan\_distance' function was used as the heuristic function. (It will be explained in detail later, but experiments were also conducted using the 'manhattan\_distance\_special\_ver' function, which reflects the actual movement cost.)

****

  
If the goal node is reached, the search is terminated and the optimal solution is backtracked using the 'parent' and 'visited\_cells' variables. Cells that are included in the optimal solution are stored in the maze's 'solution\_path' member variable with an active state set to 'False', while those not included are stored with an active state set to 'True'. Thus, the 'a\_star\_search' function returns the optimal solution of the maze and its cost. If there is no solution, it returns 'None' and '-1', respectively.

****

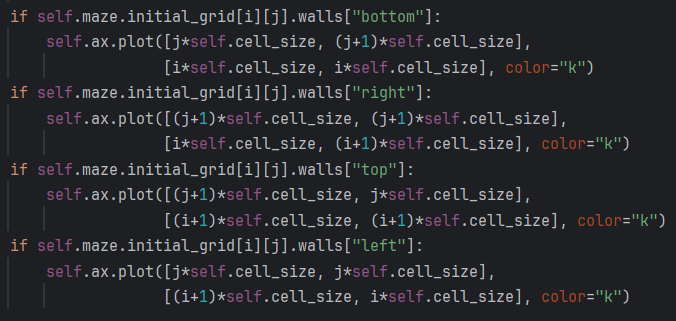
1. **Uniform Cost search algorithm:**  
   The 'uniform\_cost\_search' function takes a maze instance and returns the solution path and cost. During this process, information about the optimal solution path is also stored in the maze's 'optimal\_solution\_cost' member variable. The pseudocode is as follows.



Since the 'uniform\_cost\_search' function, unlike the 'a\_star\_search' function, does not use a heuristic function, the code is almost identical, and thus a detailed explanation is omitted.

1. **Modifications to the existing codebase:**

Due to 'top' and 'bottom' being reversed in the existing code base, slight modifications were made as follows. However, even without these modifications, the experiment results are not affected.

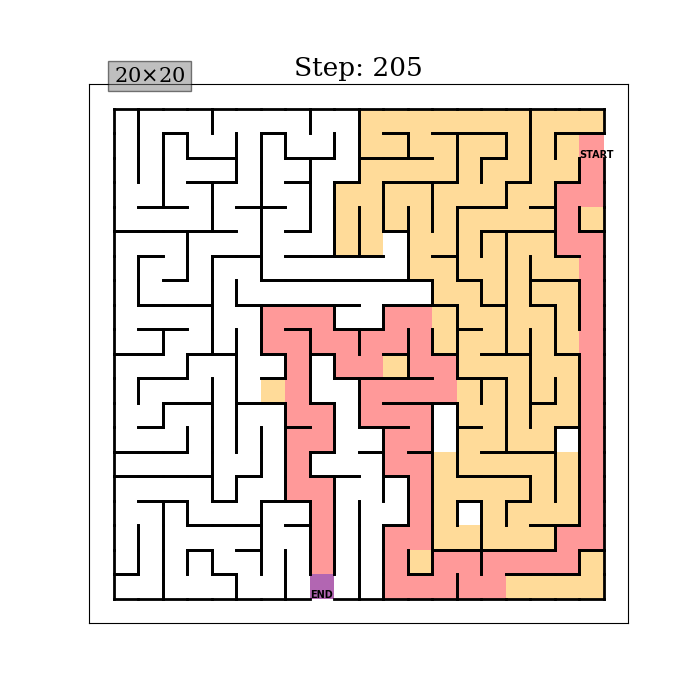
****

1. **Analysis:**

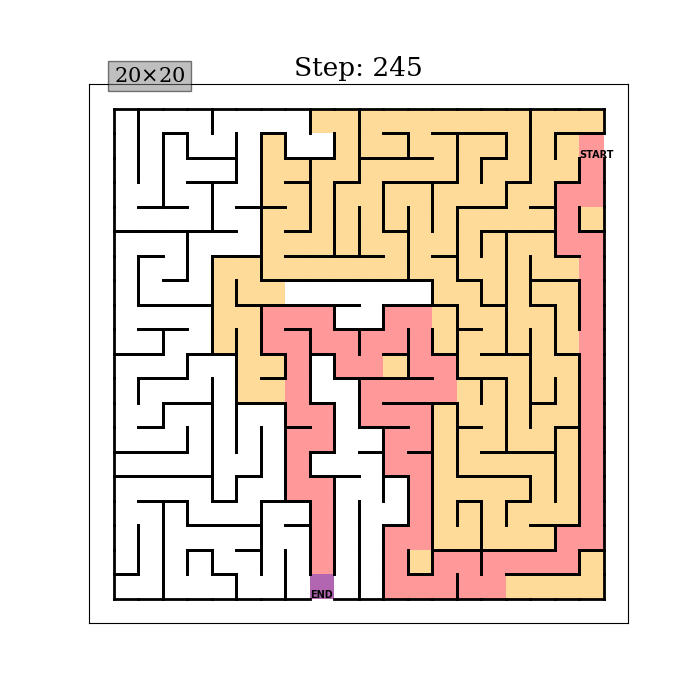
I conducted experiments using a 20 \* 20 maze. For comparison purposes, I fixed the seed value for maze.py to 1 and the seed value for algorithm.py to 0. I tested the A\* search algorithm by running the tests/a\_star\_search\_test.py file and the uniform cost search algorithm by running the tests/uniform\_cost\_search\_test.py file.

(As I will mention later, I also conducted additional experiments by applying an improved heuristic function and running the tests/a\_star\_search\_special\_version\_test.py file.)

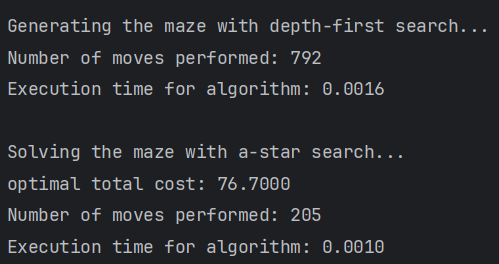
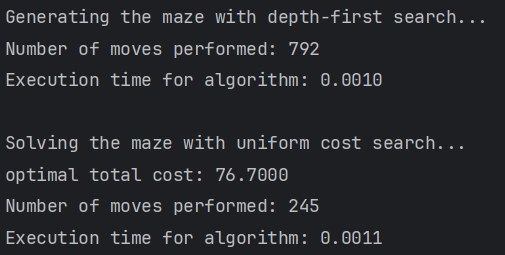
The images below are the results of the experiments.

The red cells indicate the optimal solution path, and the yellow cells represent cells that were visited but are not part of the optimal solution.

**Figure 1. a \* search algorithm result (animation)**

****

**Figure 2. uniform cost search algorithm result (animation)**

****

**Figure 4. uniform cost search algorithm result (terminal)**

**Figure 3. a \* search algorithm result (terminal)**

1. **Strengths:**

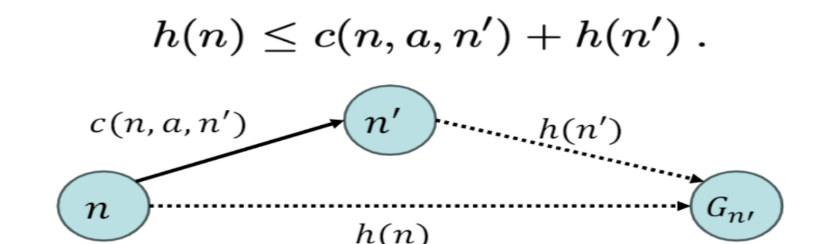
* **A \* search algorithm:**

By using a heuristic function that calculates the forward cost, the search can be conducted at a lower cost than with the uniform cost search algorithm or the BFS algorithm. In the results above, the A\* search algorithm visited 40 fewer cells than the uniform cost search algorithm.

* **Uniform cost search (UCS)algorithm:**  
  Uniform cost search guarantees an optimal solution because it conducts the search through the Dijkstra algorithm, which finds the shortest path. If the moving cost between all cells (or nodes) is the same, using the BFS algorithm with a simple queue would yield better performance.

1. **Weaknesses:**

* **A \* search algorithm:**

The dependency on the heuristic function is high. That is, depending on how the heuristic function is designed, it may not guarantee an optimal solution. The existing 'a\_star\_search' function uses the 'manhattan\_distance' function, which does not satisfy the consistency condition, and may not find the optimal solution for a given maze. "Below is the definition of the consistency condition and a proof that the existing 'manhattan\_distance' function does not satisfy this consistency condition.

Nevertheless, in the experiment, since the maze size was small (20 \* 20) and the actual distance costs (1.1, 0.9) did not significantly differ, it succeeded in finding the optimal solution.

(I conducted additional experiments using a heuristic function that addresses these weaknesses. I will mention this later in the report.)

Additionally, since the heuristic value is calculated based on the distance to the goal, if the solution to the maze involves a detour rather than a straight path to the goal node, it would require a significant search cost.

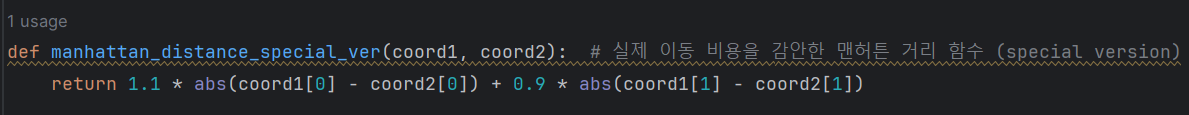
* **Uniform cost search (UCS) algorithm:**

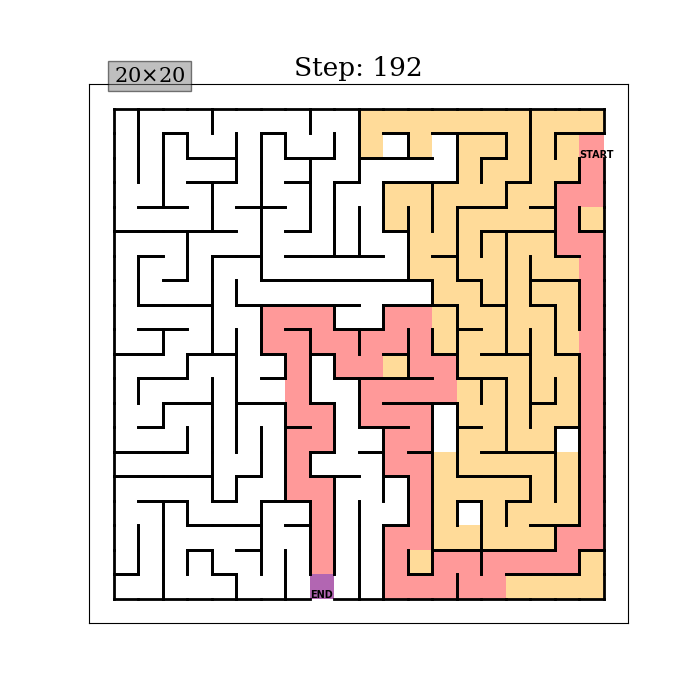
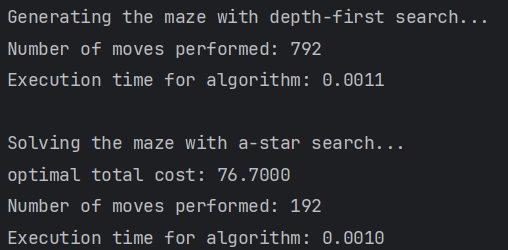
The uniform cost search algorithm guarantees an optimal solution in any situation, but since it only considers the backward cost without taking the forward cost into account, it searches through more cells, thereby incurring higher search costs.

1. **Improvements:**

As I mentioned before, I also implemented and experimented with a Manhattan distance function (‘manhattan\_distance\_special\_ver’) that reflects the actual distance costs (1.1, 0.9). This function satisfies the consistency condition, thereby guaranteeing an optimal solution.

The following are the experimental results of the A\* search algorithm with the improved heuristic applied. The experiment was conducted by running the 'a\_star\_search\_special\_version\_test.py' file.

****

**Figure 5. improved version of heuristic function**

**Figure 6, 7. a \* search algorithm result (with improve heuristic function)**

As seen in the experimental results, the A\* search algorithm using the improved heuristic function derived the optimal solution at a lower search cost. Additionally, this function will guarantee an optimal solution for any given maze.