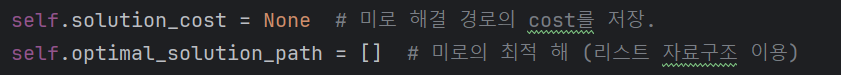
**HW #1**

**Name: 이세형**

**Student Number: 2020312145**

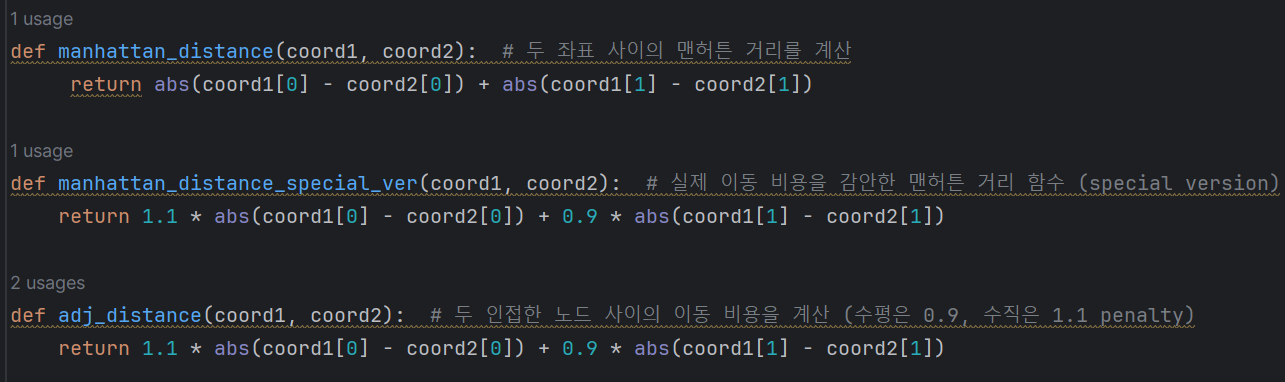
1. **Implementations:**

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

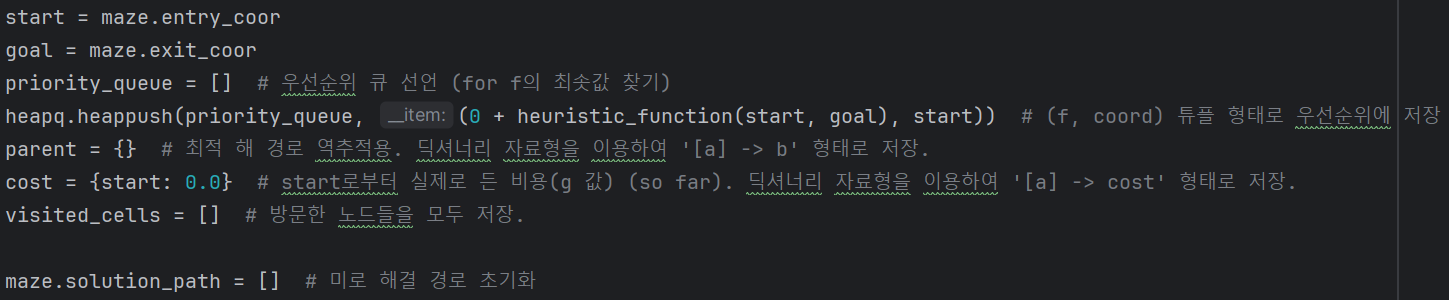
  
The 'manhattan\_distance' function returns the Manhattan distance between the two coordinates passed to it.   
The 'manhattan\_distance\_special\_ver' function returns the Manhattan distance between the two coordinates passed to it, taking into account that the cost of horizontal(0.9) and vertical(1.1)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

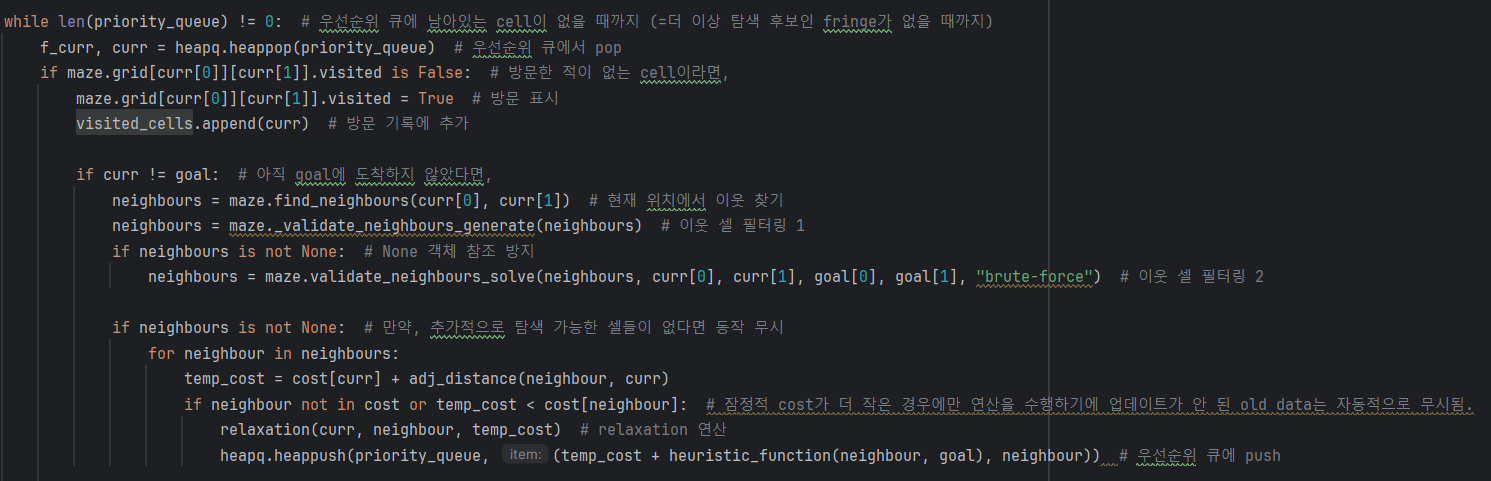
Detailed comments have been added to most lines of this code.

Below are detailed explanation about two main functions implementations.

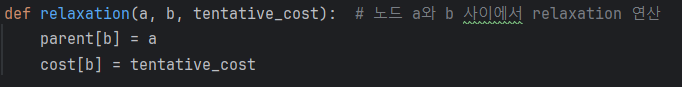
1. **A \* search algorithm:**  
   The 'a\_star\_search' function takes a maze instance and a heuristic function as inputs and returns the path and cost of the optimal solution. Below is a simplified pseudocode of the function.

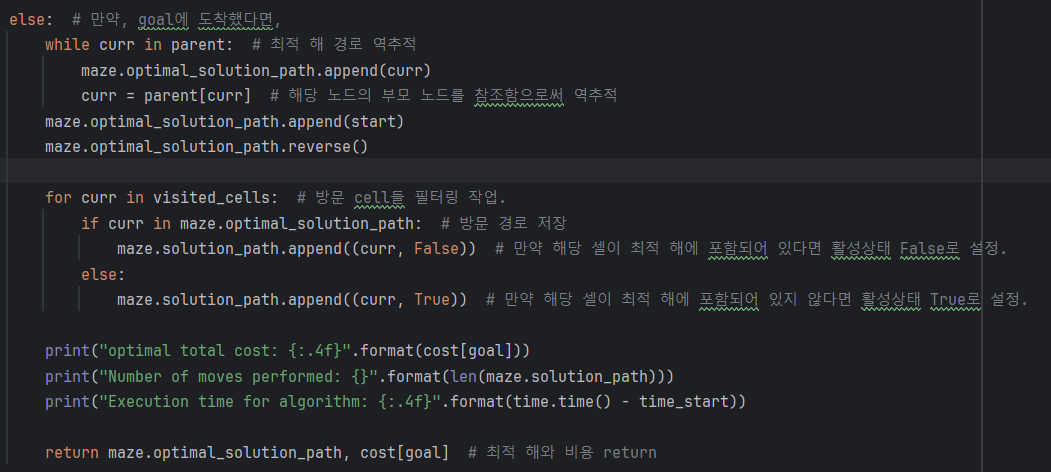
The local variables declared within the 'a\_star\_search' function are as follows.

****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.

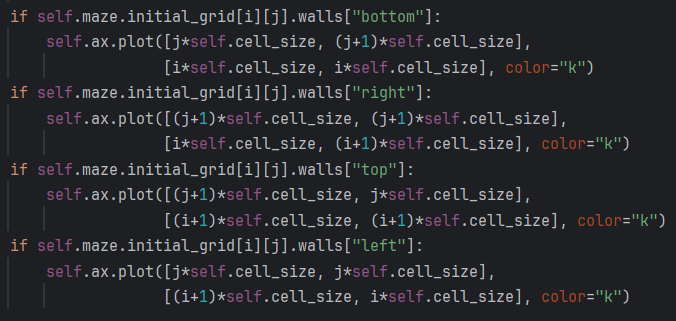
****

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:**