分配1：查找最短的路径

1 任务描述

在这个任务中，你必须引导《我的世界》中的玩家代理通过一个迷宫来到达目标块。在这个练习中，我们将假设代理可以看到整个迷宫，即。它是完全可以观察到的。根据这些信息，代理必须以最小的移动次数达到目标。
这个作业主要基于教程7.所以，请遵循教程1到5，熟悉API和Malmo平台（教程6还不相关）。

1.1 提供的源代码

我们提供了两个Python文件。最重要的一个是assignment1.py，它包含使用不同的迷宫设置MalMo环境的完整代码，并让代理运行它。所提供的实现不完整，代理需要您的帮助来最有效地解决迷宫。

1.2 设置和运行代码

假设您已经安装了Malmo并开始使用教程，运行这个任务所需要做的就是将上面的两个文件复制到Python_Examples文件夹，在启动《我的世界》后，运行pythonassignment1.py(Python3或更高版本)。如果所有操作都成功运行，代理应该在10个任务的计时器倒计时时什么都不做。终端中的输出值应如下所示：

```
Size of maze:  6
Waiting for the mission 1 to start
Mission 1 running.
Output (start,end) 1 :  (None, None)
Output (path length) 1 :  0
Output (actions) 1 :  []
Error:  out of actions, but mission has not ended!
Error:  out of actions, but mission has not ended!

Mission 1 ended
Size of maze:  6
Waiting for the mission 2 to start
Mission 2 running.
Output (start,end) 2 :  (None, None)
Output (path length) 2 :  0
Output (actions) 2 :  []
Error:  out of actions, but mission has not ended!
Error:  out of actions, but mission has not ended!
```

1.3 代码概述

如前所述，这项任务的大部分都是基于tutorial_7.py的，所以请参阅Python_Examples目录中的Malmo教程。源代码创建了一系列越来越困难的迷宫，由diamond_block块作为地板组成，每个迷宫都有一个开始(emerald_block)和一个结束(redstone_block)块。任务从玩家在开始块开始，当玩家到达结束块时结束。请确保您至少对整个源代码有些熟悉，但是与您的实现相关的主控制代码如下所述：

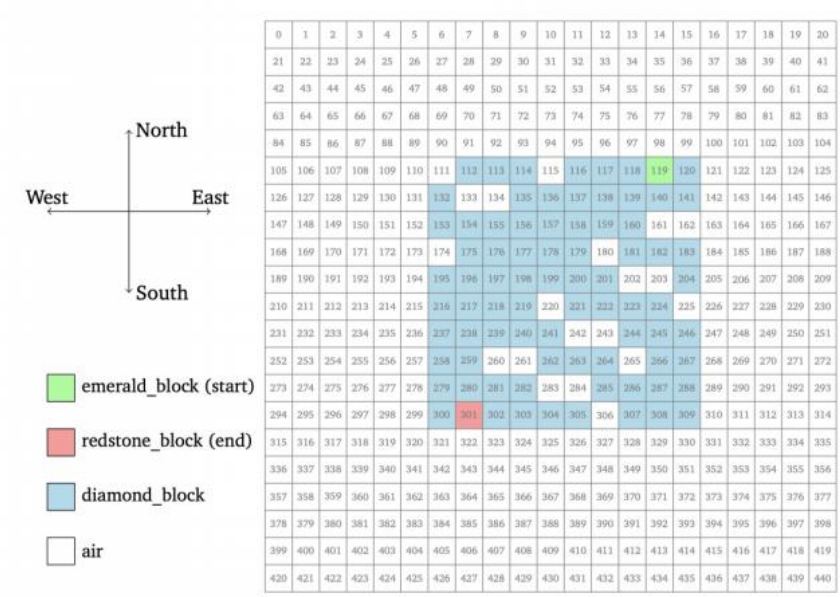

Figure 1: **Grid Layout:** Layout of the two-dimensional grid laid out as a one-dimensional array. The location of the maze, especially the start and end blocks, may be different in the assignment. The array contain string-values representing the four types of possible blocks.

```
grid = load_grid(world_state)
start, end = find_start_end(grid) # implement this
path = dijkstra_shortest_path(grid, start, end) # implement this
action_list = extract_action_list_from_path(path)
```

我们首先从load_grid中的Malmo得到由二维迷宫组成的块数组。然后，我们想要识别开始块和结束块（每个块由网格数组中的一个整数索引表示）。给定开始和结束块，以及整个迷宫，我们想要计算块从开始到结束的最短路径，使用吉布特拉的算法计算。最后，我们将这些块列表转换为由代理执行的Malmo操作。我们提供了第一个和最后一个函数的正确实现，但将第二个和第三个函数的正确实现留给您（稍后将详细介绍）。

1.4 电网布局

对您来说，了解网格是如何布局的是很重要的。我们已经要求了这21个问题×21个来自世界的积木网格，位于玩家的地板上，玩家在它的中心（(10,10)，如果是零索引的话）。该二维坐标以单维串排列（表示块类型）（行表示东西方向），即（i、j）坐标由索引j×21+i表示，其中j为东西坐标，i为南北坐标。有关精确的索引，请参见图1的示。

由于网格由单维数组表示，因此需要得到当前块的相邻块，由整数索引i表示。看看extract_action_list_from_path应该会给你一个提示；紧邻左边（西)和右(东）的块当然用i-1和i+1表示，而北部和南部的块需要跳整整一行，因此分别是i-21和i+21。函数extract_action_list_from_path使用这个来走另一种路：如果块i和j在网格上相邻，从i到j的动作用j-i表示，

i.e.如果j-i是+1，移动向东移动，如果-1，移动向西移动，如果-21，移动，如果+21，移动。

注：玩家面向南，所以如果向南向前移动，不要感到困惑

2 我提交了什么？

在这里，我们将描述您需要提交给任务的确切内容

1. 代码：查找开始和结束块：作为一个简单的练习，实现find_start_end函数，它将网格作为描述块类型的字符串数组，并返回开始块和结束块的索引。这最多应该只需要几行代码。

2. 输出：开始和结束块索引：运行上面执行的代码，并查看从输出（开始、结束）开始的输出行。

3. 代码：最短的路径实现：实现Dijkstra的算法，以找到从源到目的地的最短路径。每个街区可能采取的行动集合是向北、南、东或西的一步，即。不允许执行多个或诊断步骤。您计算的路径应该在移动次数方面是最短的(所有它们的成本相同），应该是块索引列表（整数），应该同时包括开始块和结束块，当然，不应该包含任何空气块。提交您的函数的完整实现。

4. 输出：最短路径的长度：运行使用上面实现的代码，并查看以输出（路径长度）和输出（操作）开始的输出行。

Eg：

```
Mission 2 running.
1
Output (start,end) 2 : (220, 326)
Output (path length) 2 : 13
Output (actions) 2 : ['movewest 1', 'movesouth 1',
 'movesouth 1', 'movewest 1', 'movewest 1', 'moves
outh 1', 'movesouth 1', 'movesouth 1', 'moveeast 1
', 'moveeast 1', 'moveeast 1', 'moveeast 1']

Mission 2 ended
Size of maze: 7
```