資料結構

HW3

系級：電機系大三

姓名：林凡皓

學號：B103012002

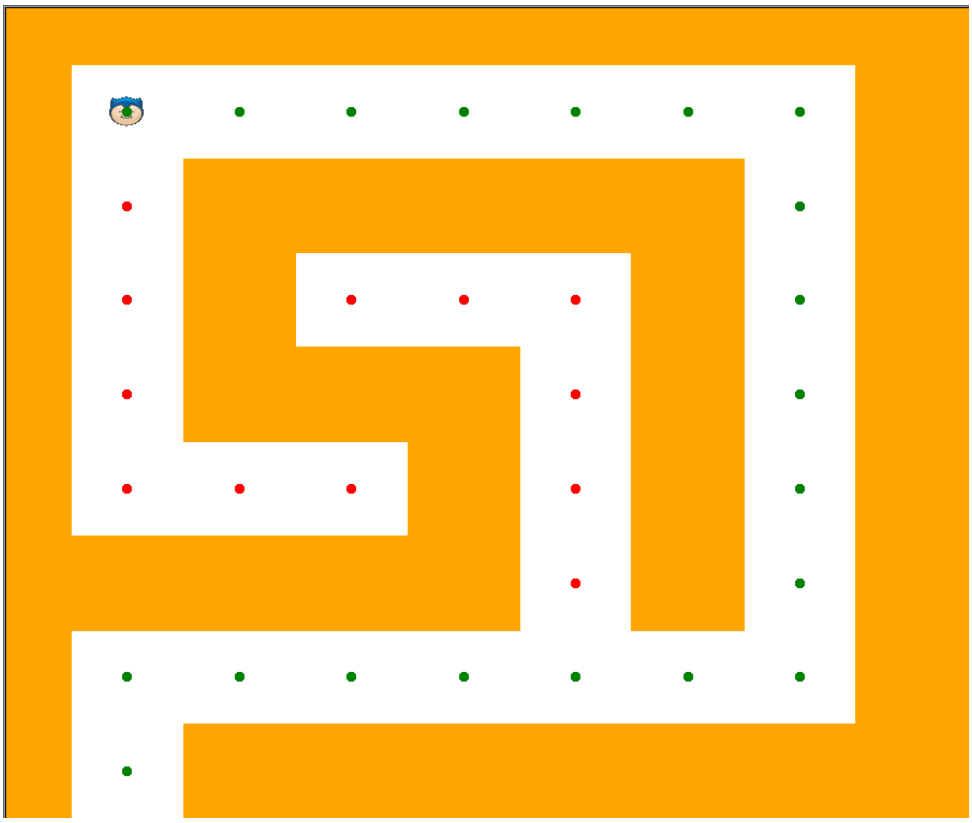
一、 Modify maze.py

我做了五種順序的迷宮，分別為$UP \rightarrow DOWN \rightarrow LEFT \rightarrow RIGHT$、$DOWN \rightarrow UP \rightarrow RIGHT \rightarrow LEFT$、$LEFT \rightarrow RIGHT \rightarrow UP \rightarrow DOWN$、$RIGHT \rightarrow LEFT \rightarrow UP \rightarrow DOWN$。

- $UP \rightarrow DOWN \rightarrow LEFT \rightarrow RIGHT$ ：

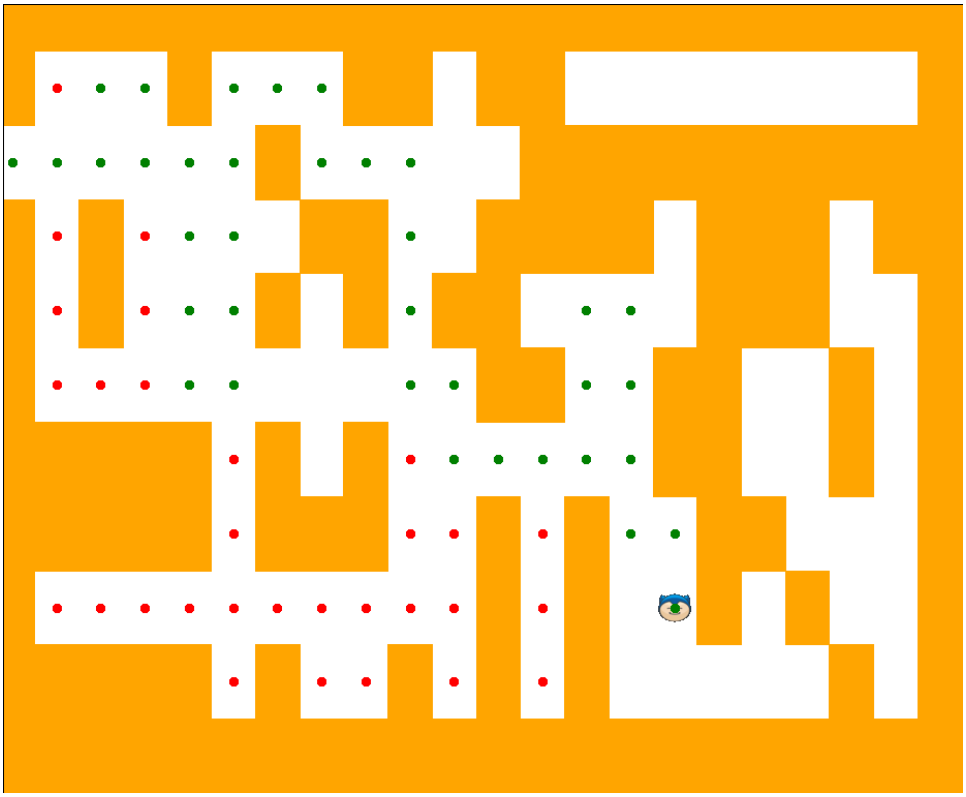
  對於 maze1 的執行結果如下

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze.py .\maze1.txt
Search path: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (3, 7),
(4, 7), (5, 7), (6, 7), (7, 7), (7, 6), (7, 5), (7, 4), (7, 3), (7, 2), (7, 1), (8, 1)
]
```



  對於 maze2 的執行結果如下

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze.py .\maze2.txt
Search path: [(8, 15), (7, 15), (7, 14), (6, 14), (5, 14), (4, 14), (4, 13), (5, 13), (
6, 13), (6, 12), (6, 11), (6, 10), (5, 10), (5, 9), (4, 9), (3, 9), (2, 9), (2, 8), (2,
 7), (1, 7), (1, 6), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (5, 4), (4, 4), (3, 4), (2
, 4), (2, 3), (1, 3), (1, 2), (2, 2), (2, 1), (2, 0)]
```

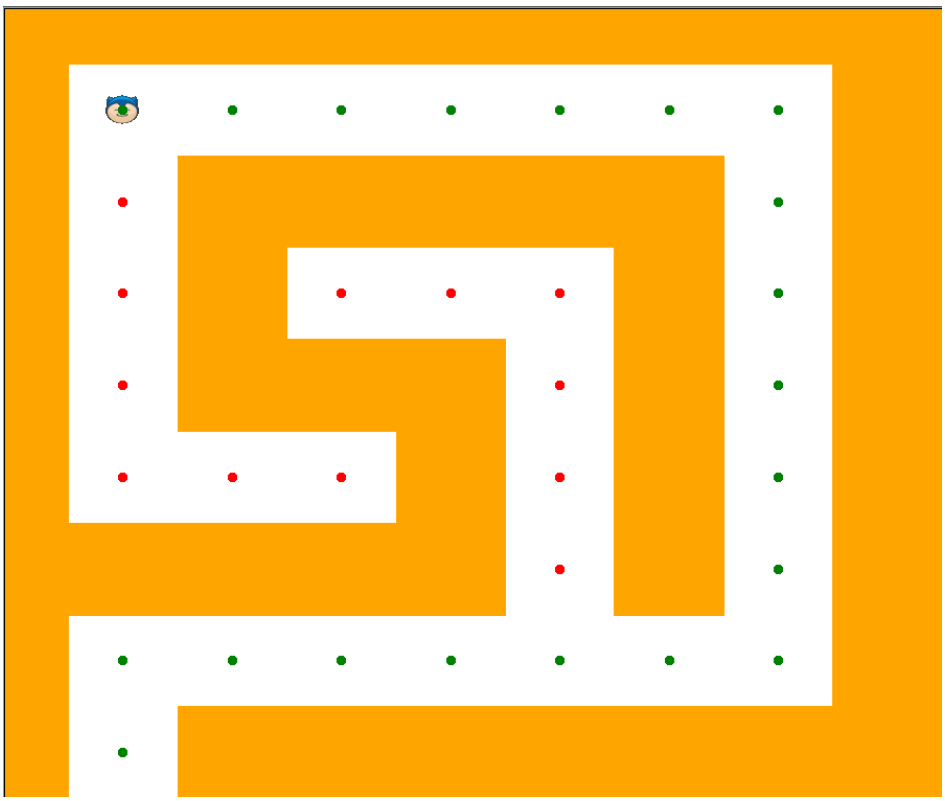


由 maze2 的執行結果可以明顯看出，卡比獸都會優先往上下去走，像是在出口附近，明明繼續往左走就可以成功到達出口，但是卡比獸依然會選擇往上優先做搜索。

- $DOWN \rightarrow UP \rightarrow RIGHT \rightarrow LEFT$ ：

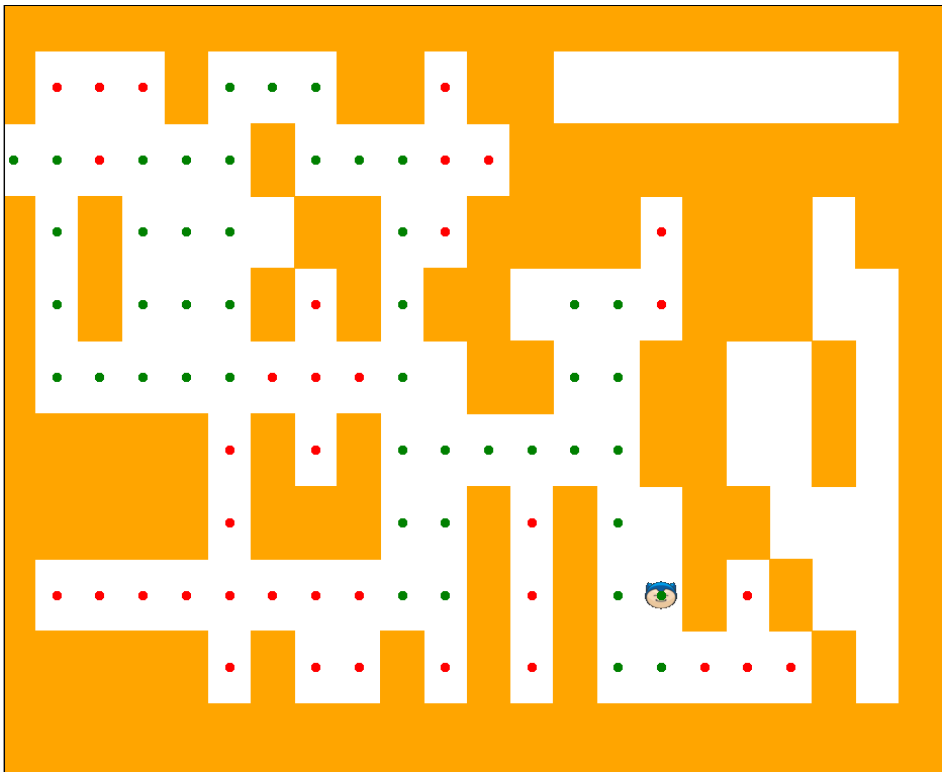
  對於 maze1 的執行結果如下

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze.py .\maze1.txt
Search path: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (3, 7),
(4, 7), (5, 7), (6, 7), (7, 7), (7, 6), (7, 5), (7, 4), (7, 3), (7, 2), (7, 1), (8, 1)
]
```



對於 maze2 的執行結果如下

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze.py .\maze2.txt
Search path: [(8, 15), (9, 15), (9, 14), (8, 14), (7, 14), (6, 14), (5, 14), (4, 14
), (4, 13), (5, 13), (6, 13), (6, 12), (6, 11), (6, 10), (7, 10), (8, 10), (8, 9),
(7, 9), (6, 9), (5, 9), (4, 9), (3, 9), (2, 9), (2, 8), (2, 7), (1, 7), (1, 6), (1,
 5), (2, 5), (3, 5), (4, 5), (5, 5), (5, 4), (4, 4), (3, 4), (2, 4), (2, 3), (3, 3)
, (4, 3), (5, 3), (5, 2), (5, 1), (4, 1), (3, 1), (2, 1), (2, 0)]
```

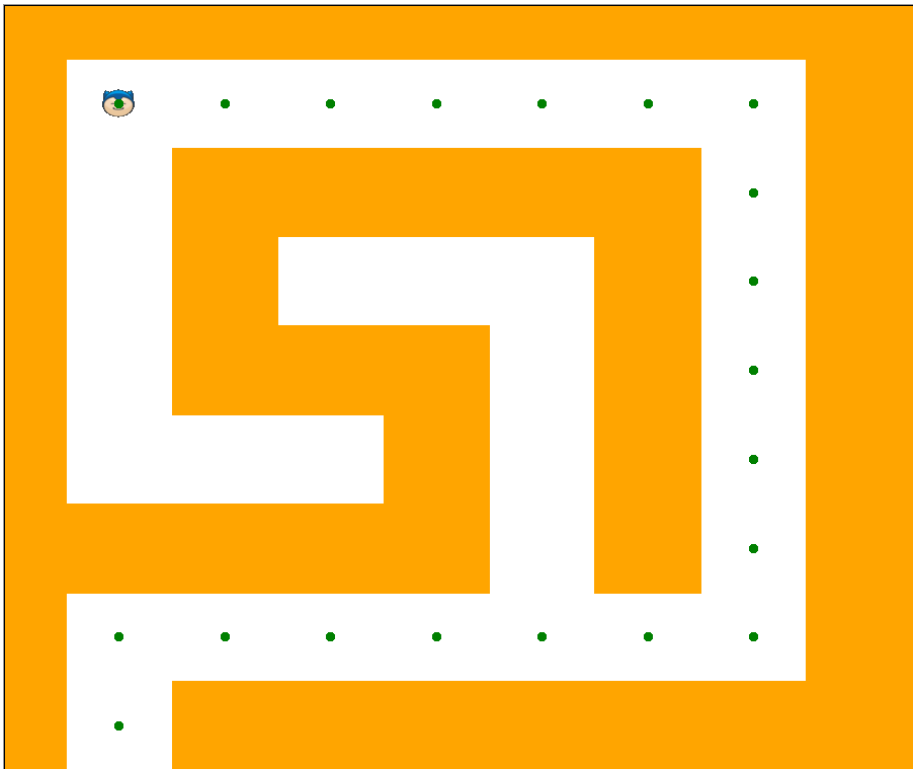


由 maze2 的執行結果可以很明顯的看出，這一次卡比獸會優先往下上搜索，像是在起點的地方，卡比獸就選擇往下走而不是像在$UP \rightarrow DOWN \rightarrow LEFT \rightarrow RIGHT$的時候選擇向上走。

- $LEFT \rightarrow RIGHT \rightarrow UP \rightarrow DOWN$ ：

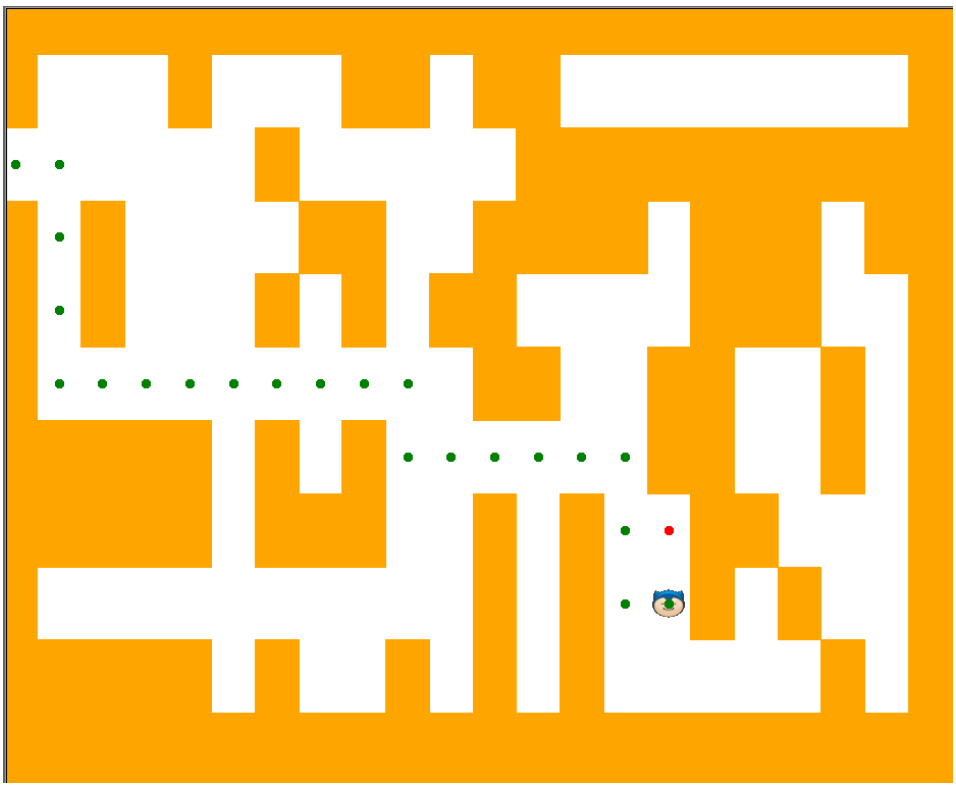
  對於 maze1 的執行結果如下

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze.py .\maze1.txt
Search path: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (3, 7
), (4, 7), (5, 7), (6, 7), (7, 7), (7, 6), (7, 5), (7, 4), (7, 3), (7, 2), (7, 1),
(8, 1)]
```



  對於 maze2 的執行結果如下

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze.py .\maze2.txt
Search path: [(8, 15), (8, 14), (7, 14), (6, 14), (6, 13), (6, 12), (6, 11), (6, 10
), (6, 9), (5, 9), (5, 8), (5, 7), (5, 6), (5, 5), (5, 4), (5, 3), (5, 2), (5, 1),
(4, 1), (3, 1), (2, 1), (2, 0)]
```

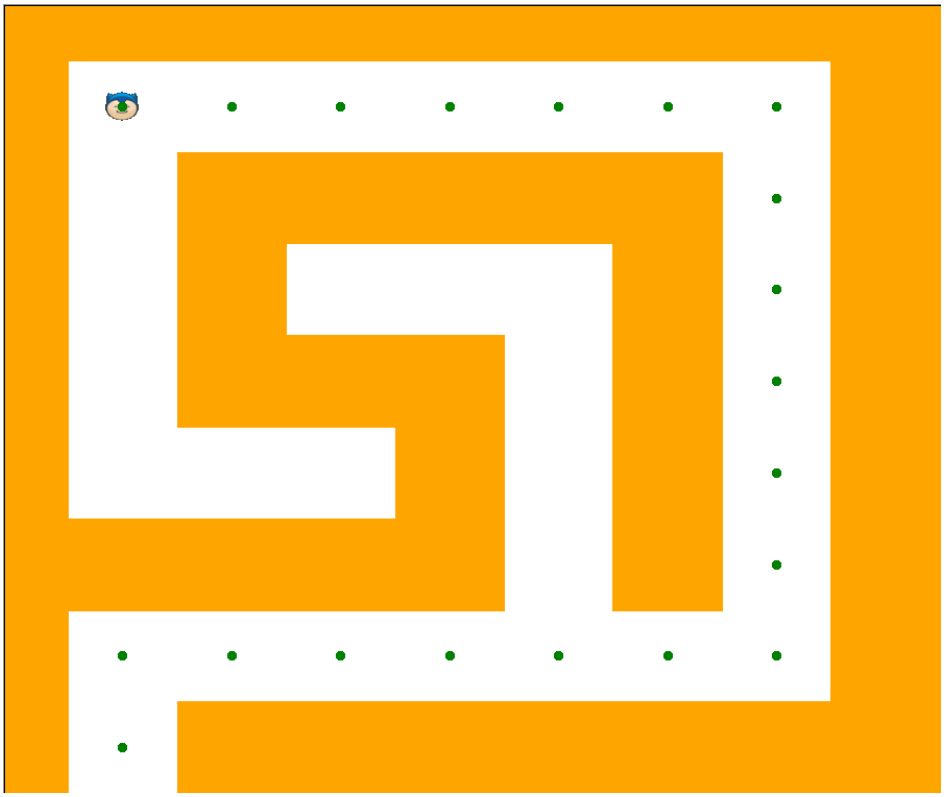


不管是由 maze1 的執行結果或是 maze2 的執行結果都可以看出，這一次卡比獸會優先向左右搜索，像是在 maze1 中，如果是 DFS 演算法的話，起點位置下方的路徑會被搜索過，但是在這一次的結果中，那一條路徑是沒有被搜索的。

- $RIGHT \rightarrow LEFT \rightarrow UP \rightarrow DOWN$ ：

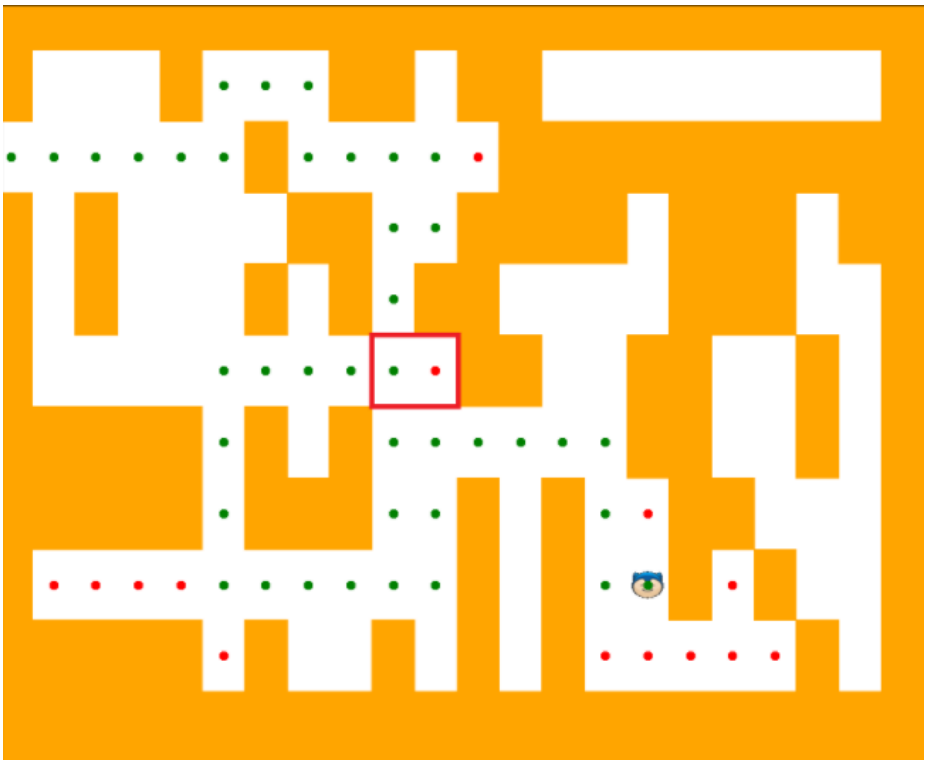
  對於 maze1 的執行結果如下

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze.py .\maze1.txt
Search path: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (3, 7
), (4, 7), (5, 7), (6, 7), (7, 7), (7, 6), (7, 5), (7, 4), (7, 3), (7, 2), (7, 1),
(8, 1)]
```



對於 maze2 的執行結果如下

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze.py .\maze2.txt
Search path: [(8, 15), (8, 14), (7, 14), (6, 14), (6, 13), (6, 12), (6, 11), (6, 10
), (6, 9), (7, 9), (7, 10), (8, 10), (8, 9), (8, 8), (8, 7), (8, 6), (8, 5), (7, 5)
, (6, 5), (5, 5), (5, 6), (5, 7), (5, 8), (5, 9), (4, 9), (3, 9), (3, 10), (2, 10),
 (2, 9), (2, 8), (2, 7), (1, 7), (1, 6), (1, 5), (2, 5), (2, 4), (2, 3), (2, 2), (2
, 1), (2, 0)]
```



由 maze2 的執行結果可以看出，這一次演算法會優先往右左做搜索，像是在紅色框框的地方，如果採用$LEFT \rightarrow RIGHT \rightarrow UP \rightarrow DOWN$的演算法，並不會有那顆紅色點，因為直接往左做搜索的話並不會碰上那條死路。

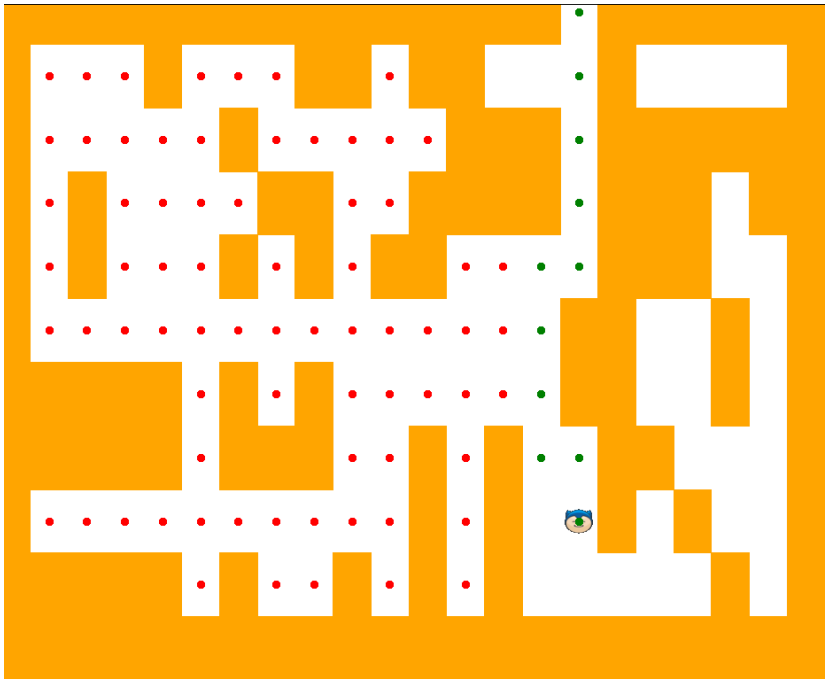
- 討論

1. 為什麼先向左或向右會找到比較短的路徑?

   我認為優先向左右搜索會找到比較好的路徑單純是看迷宮形狀決定。這一次的測試資料只有兩個迷宮是有出口的，剛好這兩個迷宮的出口路徑都是以優先向左或向右可以優先找到的。為了驗證我的想法，我自己建立一個新的迷宮來做測試。測試結果如下

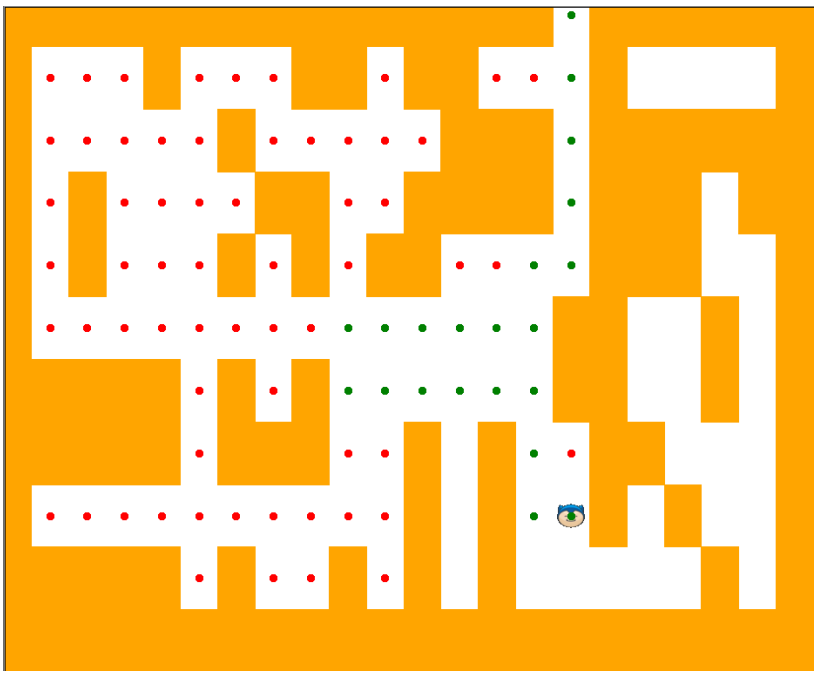
   - 優先向上下搜索：

     執行結果如下



   - 優先向左右搜索

     執行結果如下



由這個小實驗可以看出，對於這個迷宮來說，使用優先向上搜索的演算法會找到最佳路經。因此哪一種搜索順序會得到最佳結果須根據迷宮做選擇。

2. 有沒有辦法確保可以找到最佳路徑?

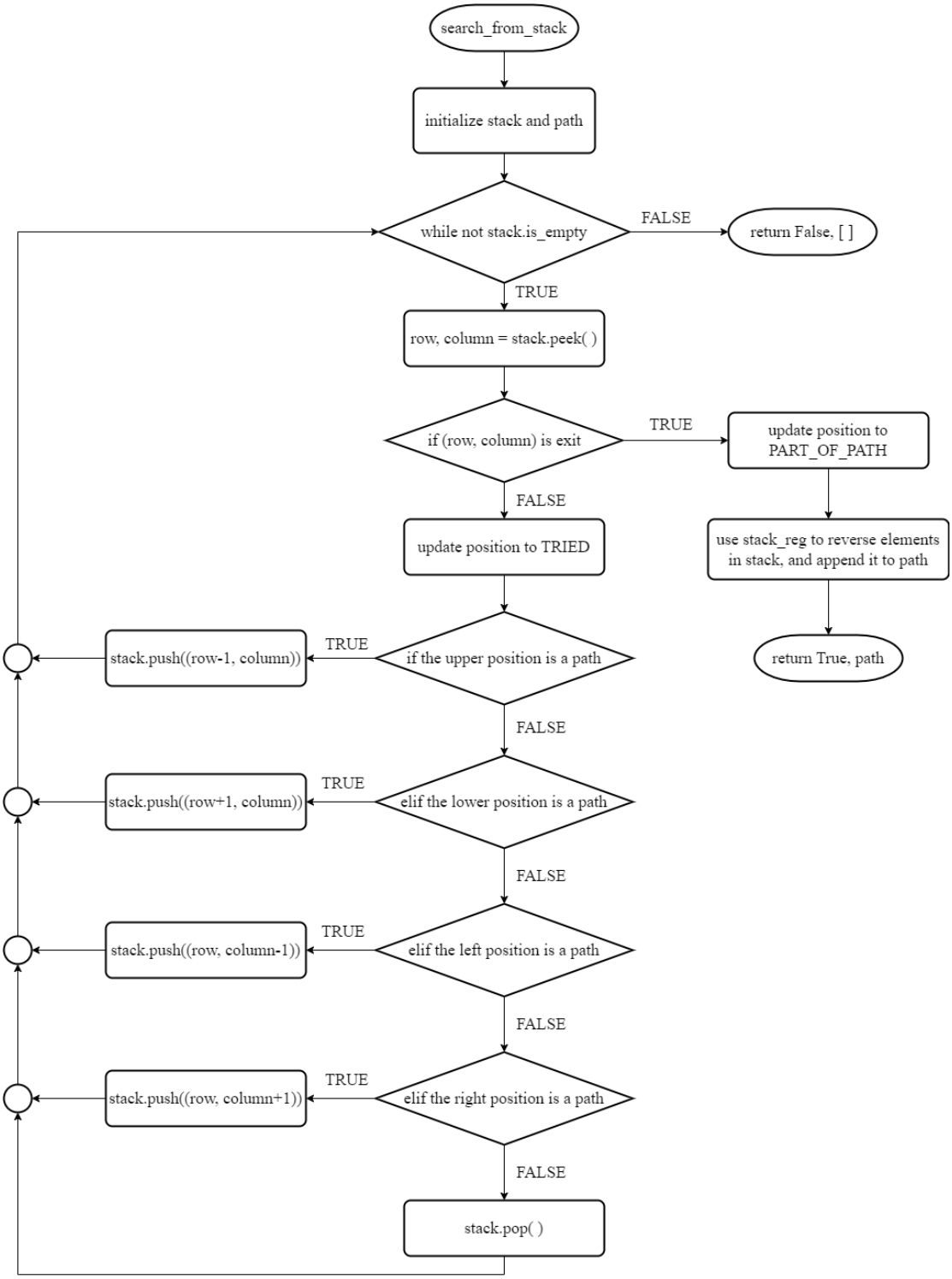
如果希望有一種演算法可以在所有迷宮中都能夠找到最佳路徑，那這個演算法會需要再遇到岔路的時候朝每一題路都逕行搜索，並將最快到達終點的路徑回傳。這就是 BFS (Breadth-First Search) 的概念。在這次作業中所使用的都是 DFS (Depth-First Search)的搜索演算法，這種演算法會根據設定好的優先搜索順序搜索到底。DFS 的好處在於很容易去實現，但是缺點就是找到的路徑並不保證是最佳解。BFS 就跟 DFS 相反，好處是基本上可以確定找到的路徑就是最佳路徑，但是相對於 DFS 難實現很多。

二、 Design of my program

- 設計概念：

這次作業我主要是利用一些 if else 判斷加上 stack 的 push、pop 來完成。程式一開始先創建一個 stack，並將起始位置放入 stack 中。接著使用 while 迴圈來做搜索迷宮。只要 stack 內還有元素，代表說還有路徑可以走，while 迴圈就會繼續執行，直到 stack 內沒有元素。每一次 while 迴圈都會先取得 stack 中 top 的元素，代表說現在要判斷這個位置是否可以走。如果這個位置是出口，就將它加入路徑中，並回傳紀

錄的路徑。否則就繼續搜索。由於有要求搜索順序為上下左右，因此接下來的 if else 會優先去判斷往上的路徑可不可以走，如可以的話，就將它 push 到 stack 中，否則繼續判斷往下的路徑。當所有路徑都判斷完且都不能走，代表說現在是死路，就利用 pop 的功能來實現往回走。程式流程圖如下



- 結果驗證 ：

1. Maze1 :

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze_stack.py .\maze1.txt .\correct_path1.txt
Search path: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7,
 7), (7, 6), (7, 5), (7, 4), (7, 3), (7, 2), (7, 1), (8, 1)]
Correct path: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7
, 7), (7, 6), (7, 5), (7, 4), (7, 3), (7, 2), (7, 1), (8, 1)]
The search path matches the correct path!
```

2. Maze2 :

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze_stack.py .\maze2.txt .\correct_path2.txt
Search path: [(8, 15), (7, 15), (7, 14), (6, 14), (5, 14), (4, 14), (4, 13), (5, 13), (6, 13), (6, 12), (6, 11),
(6, 10), (5, 10), (5, 9), (4, 9), (3, 9), (2, 9), (2, 8), (2, 7), (1, 7), (1, 6), (1, 5), (2, 5), (3, 5), (4, 5),
 (5, 5), (5, 4), (4, 4), (3, 4), (2, 4), (2, 3), (1, 3), (1, 2), (2, 2), (2, 1), (2, 0)]
Correct path: [(8, 15), (7, 15), (7, 14), (6, 14), (5, 14), (4, 14), (4, 13), (5, 13), (6, 13), (6, 12), (6, 11),
 (6, 10), (5, 10), (5, 9), (4, 9), (3, 9), (2, 9), (2, 8), (2, 7), (1, 7), (1, 6), (1, 5), (2, 5), (3, 5), (4, 5)
, (5, 5), (5, 4), (4, 4), (3, 4), (2, 4), (2, 3), (1, 3), (1, 2), (2, 2), (2, 1), (2, 0)]
The search path matches the correct path!
```

3. Maze3 :

```
C:\Users\User\Desktop\大三下\資料結構\HW\HW3>python .\maze_stack.py .\maze3.txt
Search path: []
```

三、 Differences between recursion and stack implementation

- Recursion 深度限制 ：

  使用 recursion 時會因為系統的最大遞迴深度導致說如果深度超過最大深度，會發生 stack overflow 的問題。但是對於 stack 來說，就不會有這個問題。

- 記憶體空間使用不同 ：

  使用 stack 來實作走迷宮問題時，需要額外的 stack 空間來儲存位置資訊，像是可能路徑，因此使用 stack 的話空間複雜度會與迷宮大小還有路徑長短有關。

  如果是使用 recursion 來實作，就不需要額外的空間來儲存可能路徑，但是會有 stack overflow 的問題。

- 程式複雜度 ：

  使用 stack 實作的話，我們需要自己去控制與判斷何時要 pop，何時要 push。這些都會增加程式的複雜度與降低可讀性。

  如果使用 recursion 來實作，我們只需要去定義好 recursion 停止的 base cases，並讓程式不斷呼叫自己即可，這會大幅降低 coding 的難度以及增加程式可讀性。

四、 What I have learned

這次作業主要是學習如何將 stack 實際應用，以及學習如何分析不同演算法之間的優劣。

在調整搜索順序的部分中，實際的看到卡比獸根據不一樣的搜索順序而有不一樣的走訪路徑後，便可以很明確地看到搜索順序是會影響到走迷宮的過程，此外我也透過自己設計的小實驗發現到，搜索順序跟有沒有襖到最佳路徑並沒有絕對的關係，而是會根據迷宮的不同而有不一樣的結果。

在實作走迷宮的部分中，一開始很不知所措，我是透過詳細的閱讀 recursion 版本的 code 以及到網路上搜尋相關演算法才讓卡比獸變聰明的。在實作過程中，因為有詳細閱讀 recursion 版本的 code，所以對於使用 recursion 來實作走迷宮有了更深的了解，像是 base cases 的設置以及如何做到呼叫自己。此外，將 recursion 改成 stack 的過程中，我也深刻的體會到兩個實作起來的難度差異以及 code 可讀性的差異，用 stack 實作的時候會需要加上一些判斷標準來確認要不要將下一個位置列為可能路徑，光是這些判斷就會讓整個演算法複雜許多，code 也會變得比較不直觀。

五、 Reference

[1] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/

[2] Darth-Phoenix “DFS 與 BFS”
https://hackmd.io/@rd2865OAQZSLjri24DYcow/B1zalLE6u

[3] 資工趴趴熊的小天地 “資料結構-stack 堆疊”
https://k3331363.pixnet.net/blog/post/42708058