William Bidle

Final Question 1

At face value this seemed like a very difficult problem. Given that I don't know whether I am at A or B and that my sensors are broken, I needed to be careful with how I approached it. The first thing I realized is that instead of focusing on getting A or B to G, I needed to get A and B to be at the same point. If I know that A and B overlap as well as their location, then it is an easy job to figure out how to get to the goal. For example, consider the following solution:

Move 3 left
Move 4 down
Move 1 right
Move 3 down (now B is at G, but A is not)
Move 8 to the left (now A and B are both in the same location)
Move 8 to the right (now both A and B are at G and we are done)

This sequence of moves guarantees a solution whether you start at A or B. But is it the shortest sequence of moves, or is there a smaller sequence?

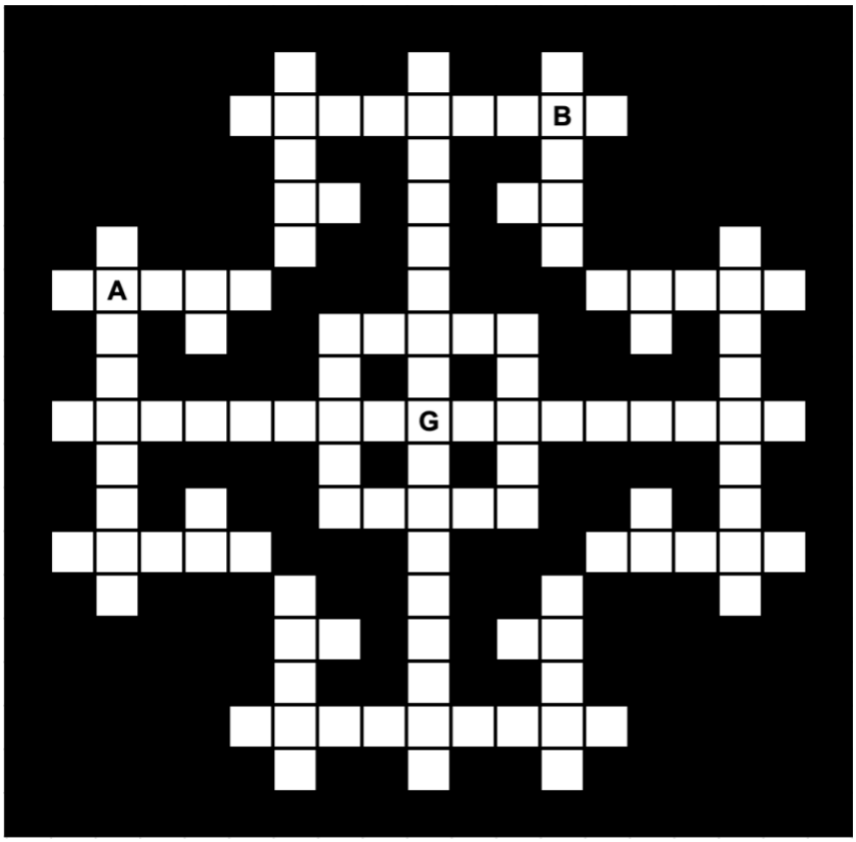


Figure 1. The maze

Finding the Shortest Path

This sort of problem screamed for something like BFS. BFS would search through all of the possibilities in the maze and return the shortest path length every single time[1]. Unlike project 1, however, the points were moving together in tandem with one another, and I had to consider staying put as a move. To keep track of where A and B were in the maze, I used pointers (i, j) for A, and (m, n) for B, which represented their respective coordinates. For each timestep, I analyzed the four possible moves (up, down, left, and right) from where A and B currently were and added them to my general knowledge base/fringe. If A or B were next to a black square and were trying to move into it, nothing would happen, and they would remain stationary. There are situations, however, where one of the two stays place while the other advances in that direction. Having this happen in certain move is necessary to get the solution, as the only way to have A and B coincide is to have some moves where one is moving and the other is stationary. I also made sure to keep track of the order of moves made at what time, by storing the values into a dictionary. Once the points A and B were at the same spot (i.e. i = m and j = n), then I reconstructed the path that was taken to get to this solution using the dictionary. The results can be seen in Figure 2 below.

```
Starting location for A: (6, 2)
Starting location for B: (2, 12)

Optimal moves:
left
left
left
down
down
down
down
right
down
left
left
down
down
left
left
left
left
left
left

Number of moves: 19
Final combined location: (9, 1)
Goal: (9, 9)
```

Figure 2. Calculated Steps to combine A and B

[1] This was also the easiest one to implement. DFS is no good since it does not guarantee shortest path, BiBFS doesn't seem make sense here, and A* is unnecessary.

It seems as though the solution I had found earlier is almost identical to the solution my algorithm finds. The only deviation is when B comes to the intersection in from of G, since I went straight to G then all the way left, and my program went left, down, then all the way left to meet up with A. The length of both solutions is the same, and therefore there is more than one optimal solution (but only these two). Of course, the algorithm only finds where A and B intersect, but the hard part is already done, because now I can just move 8 spaces right directly to G and be done. Just to be sure, I plugged in the new combined starting coordinate for A and B, (9, 1) and coordinate of G, (9, 9) back into my algorithm. This would then give the shortest path between the combined A/B and G. As seen in Figure 3, the results are exactly as I predicted, as A/B only need to move 8 spaces to the right[2]. In the end, it isn't really a problem of figuring out how to get A to G or B to G, it is really about getting A and B together, and based off this location moving directly to the goal.

```
Starting location for combined: (9, 1)
Starting location for G: (9, 9)

Optimal moves:
left
left
left
left
left
left
left
left

Number of moves: 8
Final combined location: (9, 1)
Goal: (9, 9)
```

Figure 3. Calculated Steps from combined A/B to G

See the code attached for an example that it works for a different starting A and B.

[2] Ignore that the output says left 8 times… for some reason I was calculating the moves to get from the goal to the combined A/B point, which is really the same thing just in reverse. Everything is relative.

In [303]:
```
import numpy as np
import matplotlib.pyplot as plt
import queue
```

Algorithm

In [304]:

```
def update(maze, i, j, m, n):
    maze[i][j] = 0.5
    maze[m][n] = 0.5

def mazeSearch(maze, A, B):
    #initialize the solved state of the maze to be false and the pointer
s i, j, m, and n
    #i controls row and j controls column for A
    #m controls row and j controls column for B
    solved = False
    i, j = A
    m, n = B

    #keep track of the steps taken so far
    stepnum = 0

    #create a dictionary called prev to point to previous positions
    prev = {}

    #initialize the fringe (a queue) and store the starting point of the
maze
    fringe = queue.Queue()
    fringe.put((i, j, m, n, stepnum))

    #runs until a solution is found
    while solved == False:

        #gets the current node and update i, j, m, n, and stepnum
        current = fringe.get()
        i, j, m, n, stepnum = current[0], current[1], current[2], curren
t[3], current[4]



        #check if solution has been found, i.e. A and B overlap
        if i == m and j == n:
            location = i,j
            update(maze, i, j, m, n)
            print("SOLUTION FOUND")

            #set solution_length equal to the current number of steps ta
ken

            solution_length = stepnum

            move_list = []
            while i != A[0] or j!= A[1] or m != B[0] or n!= B[1]:

                #find the previous move to this one, and add it to the m
ove list
                x = prev[i,j, m, n, stepnum]
                if x[0] - i == 1 or x[2] - m == 1:
                    move_list.append('up')

                if x[0] - i == -1 or x[2] - m == -1:
                    move_list.append('down')
```

```
                if x[1] - j == 1 or x[3] - n == 1:
                    move_list.append('left')

                if x[1] - j == -1 or x[3] - n == -1:
                    move_list.append('right')

                i, j, m, n, stepnum = x[0], x[1], x[2], x[3], x[4]
            break



        #check down move possibilities
        if maze[i + 1][j] == 1 and maze[m + 1][n] == 1:
            #add to fringe and dictionary if not already in fringe
            if [i, j, m, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j, m, n, stepnum + 1])
                prev[(i, j, m, n, stepnum + 1)] = (i, j, m, n, stepnum)

        elif maze[i + 1][j] == 1 and (maze[m + 1][n] == 0 or maze[m + 1]
[n] == 0.5):
            #add to fringe and dictionary if not already in fringe
            if [i, j, m + 1, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j, m + 1, n, stepnum + 1])
                prev[(i, j, m + 1, n, stepnum + 1)] = (i, j, m, n, stepn
um)

        elif (maze[i + 1][j] == 0 or maze[i + 1][j] == 0.5)  and maze[m
+ 1][n] == 1:
            #add to fringe and dictionary if not already in fringe
            if [i + 1, j, m, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i + 1, j, m, n, stepnum + 1])
                prev[(i + 1, j, m, n, stepnum + 1)] = (i, j, m, n, stepn
um)

        elif (maze[i + 1][j] == 0 or maze[i + 1][j] == 0.5) and (maze[m
+ 1][n] == 0 or maze[m + 1][n] == 0.5):
            #add to fringe and dictionary if not already in fringe
            if [i + 1, j, m + 1, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i + 1, j, m + 1, n, stepnum + 1])
                prev[(i + 1, j, m + 1, n, stepnum + 1)] = (i, j, m, n, s
tepnum)



        #check right move possibilities
        if maze[i][j + 1] == 1 and maze[m][n + 1] == 1:
            #add to fringe and dictionary if not already in fringe
            if [i, j, m, n, stepnum + 1] in fringe.queue:
```

```
                pass
            else:
                fringe.put([i, j, m, n, stepnum + 1])
                prev[(i, j, m, n, stepnum + 1)] = (i, j, m, n, stepnum)

        elif maze[i][j + 1] == 1 and (maze[m][n + 1] == 0 or maze[m][n +
1] == 0.5):
            #add to fringe and dictionary if not already in fringe
            if [i, j, m, n + 1, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j, m, n + 1, stepnum + 1])
                prev[(i, j, m, n + 1, stepnum + 1)] = (i, j, m, n, stepn
um)

        elif (maze[i][j + 1] == 0 or maze[i][j + 1] == 0.5)  and maze[m]
[n + 1] == 1:
            #add to fringe and dictionary if not already in fringe
            if [i, j + 1, m, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j + 1, m, n, stepnum + 1])
                prev[(i, j + 1, m, n, stepnum + 1)] = (i, j, m, n, stepn
um)

        elif (maze[i][j + 1] == 0 or maze[i][j + 1] == 0.5) and (maze[m]
[n + 1] == 0 or maze[m][n + 1] == 0.5):
            #add to fringe and dictionary if not already in fringe
            if [i, j + 1, m, n + 1, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j + 1, m, n + 1, stepnum + 1])
                prev[(i, j + 1, m, n + 1, stepnum + 1)] = (i, j, m, n, s
tepnum)



        #check up move possibilities
        if maze[i - 1][j] == 1 and maze[m - 1][n] == 1:
            #add to fringe and dictionary if not already in fringe
            if [i, j, m, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j, m, n, stepnum + 1])
                prev[(i, j, m, n, stepnum + 1)] = (i, j, m, n, stepnum)


        elif maze[i - 1][j] == 1 and (maze[m - 1][n] == 0 or maze[m - 1]
[n] == 0.5):
            #add to fringe and dictionary if not already in fringe
            if [i, j, m - 1, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j, m - 1, n, stepnum + 1])
                prev[(i, j, m - 1, n, stepnum + 1)] = (i, j, m, n, stepn
um)
```

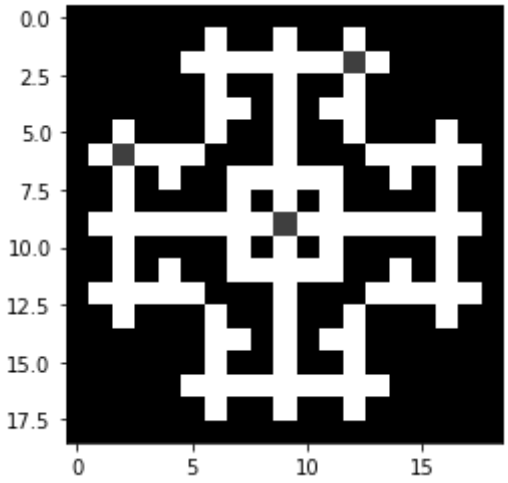
```
        elif (maze[i - 1][j] == 0 or maze[i - 1][j] == 0.5)  and maze[m
- 1][n] == 1:
            #add to fringe and dictionary if not already in fringe
            if [i - 1, j, m, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i - 1, j, m, n, stepnum + 1])
                prev[(i - 1, j, m, n, stepnum + 1)] = (i, j, m, n, stepn
um)


        elif (maze[i - 1][j] == 0 or maze[i - 1][j] == 0.5) and (maze[m
- 1][n] == 0 or maze[m - 1][n] == 0.5):
            #add to fringe and dictionary if not already in fringe
            if [i - 1, j, m - 1, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i - 1, j, m - 1, n, stepnum + 1])
                prev[(i - 1, j, m - 1, n, stepnum + 1)] = (i, j, m, n, s
tepnum)




        #check left move possibilities
        if maze[i][j - 1] == 1 and maze[m][n - 1] == 1:
            #add to fringe and dictionary if not already in fringe
            if [i, j, m, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j, m, n, stepnum + 1])
                prev[(i, j, m, n, stepnum + 1)] = (i, j, m, n, stepnum)


        elif maze[i][j - 1] == 1 and (maze[m][n - 1] == 0 or maze[m][n -
1] == 0.5):
            #add to fringe and dictionary if not already in fringe
            if [i, j, m, n - 1, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j, m, n - 1, stepnum + 1])
                prev[(i, j, m, n - 1, stepnum + 1)] = (i, j, m, n, stepn
um)

        elif (maze[i][j - 1] == 0 or maze[i][j - 1] == 0.5)  and maze[m]
[n - 1] == 1:
            #add to fringe and dictionary if not already in fringe
            if [i, j - 1, m, n, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j - 1, m, n, stepnum + 1])
                prev[(i, j - 1, m, n, stepnum + 1)] = (i, j, m, n, stepn
um)


        elif (maze[i][j - 1] == 0 or maze[i][j - 1] == 0.5) and (maze[m]
[n - 1] == 0 or maze[m][n - 1] == 0.5):
```

```
            #add to fringe and dictionary if not already in fringe
            if [i, j - 1, m, n - 1, stepnum + 1] in fringe.queue:
                pass
            else:
                fringe.put([i, j - 1, m, n - 1, stepnum + 1])
                prev[(i, j - 1, m, n - 1, stepnum + 1)] = (i, j, m, n, s
tepnum)

        #update the maze and keep going
        update(maze, i, j, m, n)

    return solution_length, move_list, location
```

Find a Solution

In [305]:
```
#load in the data
file = np.loadtxt("Map.txt", dtype = str)
for item in range(len(file)):
    for thing in range(len(file[0])):
        if file[item][thing] == 'A':
            A = (item, thing)
            file[item][thing] = '1'
        if file[item][thing] == 'B':
            B = (item, thing)
            file[item][thing] = '1'
        if file[item][thing] == 'G':
            G = (item, thing)
            file[item][thing] = '1'

#this isn't necessary but it looks nicer visually for me while I was doi
ng it
maze = 1 - np.array(file, dtype = float)
maze[A] = 0.75
maze[B] = 0.75
maze[G] = 0.75

print('Calculating Solution...')
plt.imshow(maze, cmap=plt.cm.binary)
plt.show()

solution_length, move_list, location = mazeSearch(maze, A, B)
```

```
Calculating Solution...
```




```
SOLUTION FOUND
```

In [306]:
```
print('Starting location for A:', A)
print('Starting location for B:', B)
print()

print('Optimal moves:')
for i in range(len(move_list)):
    print(move_list[len(move_list) - 1 - i])

print()
print('Number of moves:',solution_length)
print('Final combined location:', location)
print('Goal:', G)
```

```
Starting location for A: (6, 2)
Starting location for B: (2, 12)

Optimal moves:
left
left
left
down
down
down
down
right
down
left
left
down
down
left
left
left
left
left
left

Number of moves: 19
Final combined location: (9, 1)
Goal: (9, 9)
```

In [307]:
```
solution_length, move_list, location2 = mazeSearch(maze, G, location)
```

```
SOLUTION FOUND
```

In [308]:
```
print('Starting location for combined:', location)
print('Starting location for G:', G)
print()

print('Optimal moves:')
for i in range(len(move_list)):
    print(move_list[len(move_list) - 1 - i])

print()
print('Number of moves:',solution_length)
print('Final combined location:', location)
print('Goal:', G)
```

```
Starting location for combined: (9, 1)
Starting location for G: (9, 9)

Optimal moves:
left
left
left
left
left
left
left
left

Number of moves: 8
Final combined location: (9, 1)
Goal: (9, 9)
```

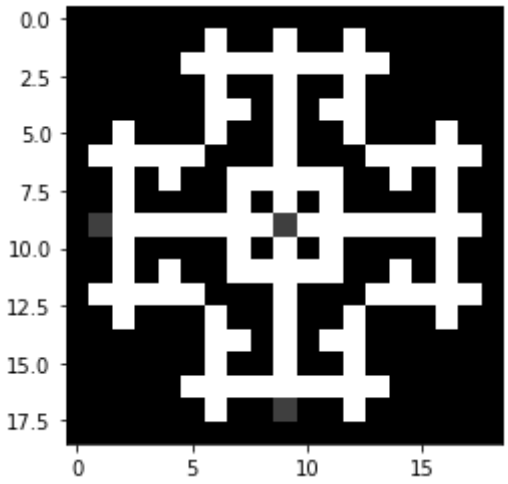
Show That It Works For Different A & B

In [309]:

```
#load in the data
file = np.loadtxt("Map.txt", dtype = str)
for item in range(len(file)):
    for thing in range(len(file[0])):
        if file[item][thing] == 'A':
            A = (item, thing)
            file[item][thing] = '1'
        if file[item][thing] == 'B':
            B = (item, thing)
            file[item][thing] = '1'
        if file[item][thing] == 'G':
            G = (item, thing)
            file[item][thing] = '1'

# set a different A and B!
A = (9, 1)
B = (17,9)

#this isn't necessary but it looks nicer visually for me while I was doi
ng it
maze = 1 - np.array(file, dtype = float)
maze[A] = 0.75
maze[B] = 0.75
maze[G] = 0.75

print('Calculating Solution...')
plt.imshow(maze, cmap=plt.cm.binary)
plt.show()

solution_length, move_list, location = mazeSearch(maze, A, B)


print('Starting location for A:', A)
print('Starting location for B:', B)
print()

print('Optimal moves:')
for i in range(len(move_list)):
    print(move_list[len(move_list) - 1 - i])

print()
print('Number of moves:',solution_length)
print('Final combined location:', location)
print('Goal:', G)

solution_length, move_list, location2 = mazeSearch(maze, G, location)

print('Starting location for combined:', location)
print('Starting location for G:', G)
print()

print('Optimal moves:')
for i in range(len(move_list)):
    print(move_list[len(move_list) - 1 - i])
```

```
print('^ for some reason it calculates the moves that G should make, but
they should all say up ^')
```

```
Calculating Solution...
```



```
SOLUTION FOUND
Starting location for A: (9, 1)
Starting location for B: (17, 9)

Optimal moves:
right
right
right
right
right
right
down
down
right
right
down
down
down
down
down
down

Number of moves: 16
Final combined location: (17, 9)
Goal: (9, 9)
SOLUTION FOUND
Starting location for combined: (17, 9)
Starting location for G: (9, 9)

Optimal moves:
down
down
down
down
down
down
down
down
^ for some reason it calculates the moves that G should make, but they
should all say up ^
```