Author: A.J. Litchfield

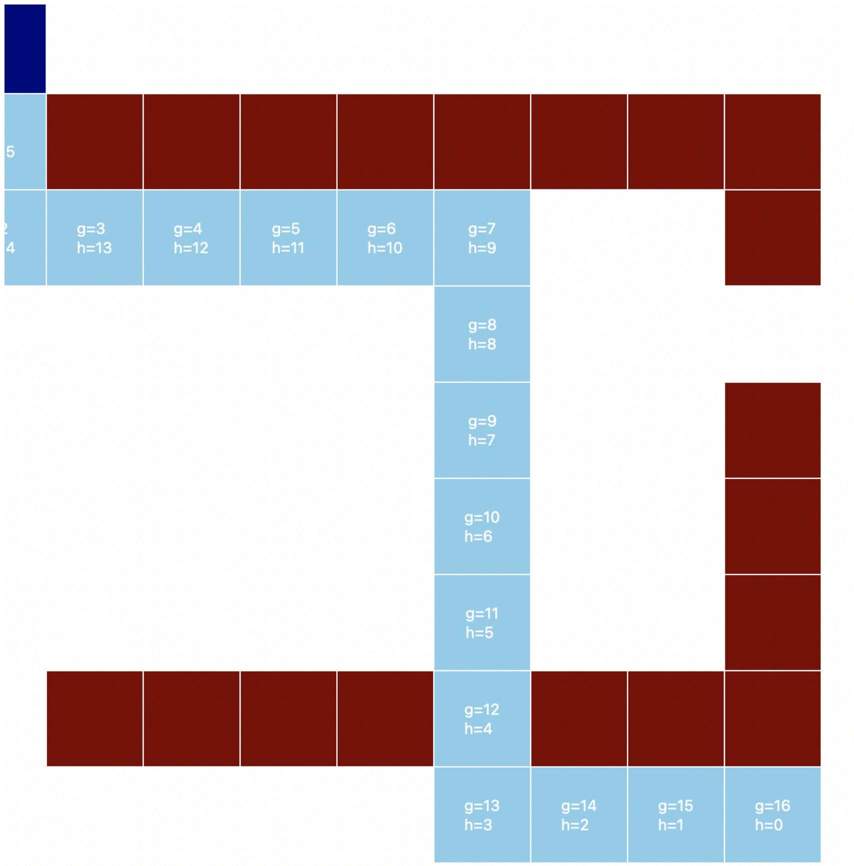
For this assignment, we were asked to compare A* and greedy best first search algorithms when implemented in maze traversals. We were also asked to change the heuristic used, as well as manipulate the evaluation functions used.

Problem 1:
For this problem, I removed the consideration of g(n) from the evaluation function portion of the A* algorithm to change the traversal to a greedy best first approach that is only making it's next moved based on the heuristic h(n). Below is my updated code:

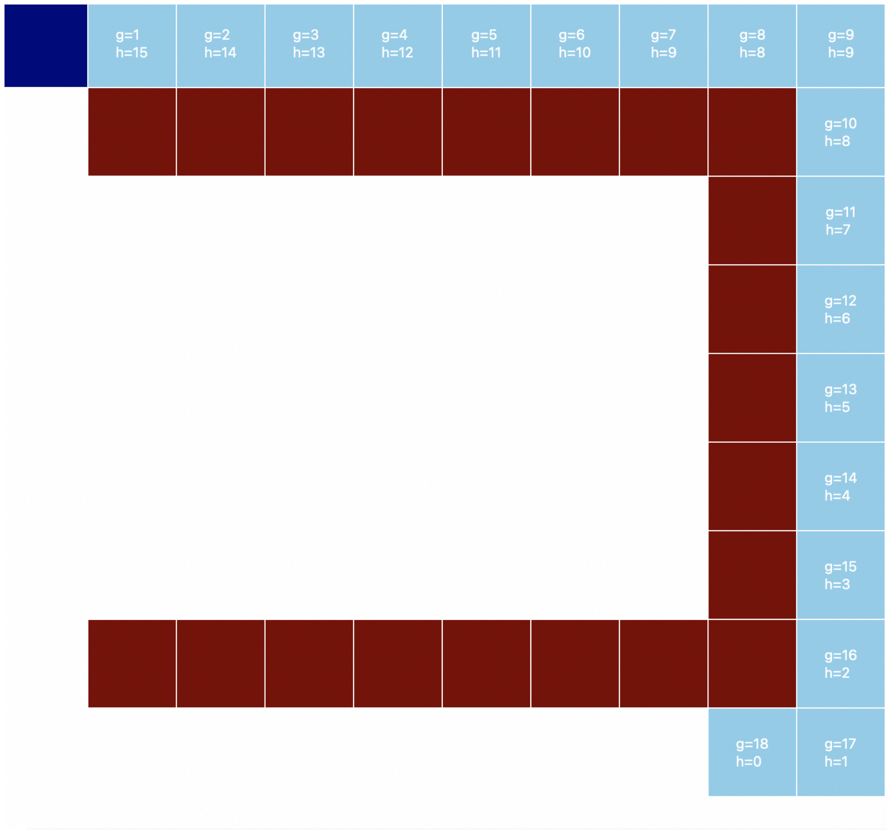
```
############################################################
#### Greedy Best First
############################################################
def find_path_GBF(self):
    open_set = PriorityQueue()

    #### Add the start state to the queue
    open_set.put((0, self.agent_pos))

    #### Continue exploring until the queue is exhausted
    while not open_set.empty():
        current_cost, current_pos = open_set.get()
        current_cell = self.cells[current_pos[0]][current_pos[1]]

        #### Stop if goal is reached
        if current_pos == self.goal_pos:
            self.reconstruct_path()
            break


        #### Agent goes E, W, N, and S, whenever possible
        for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
            new_pos = (current_pos[0] + dx, current_pos[1] + dy)

            if 0 <= new_pos[0] < self.rows and 0 <= new_pos[1] < self.cols and not self.cells[new_pos[0]][new_pos[1]].is_wall:

                #### The cost of moving to a new position is 1 unit
                new_g = current_cell.g + 1


                if new_g < self.cells[new_pos[0]][new_pos[1]].g:
                    ### Update the path cost g()
                    self.cells[new_pos[0]][new_pos[1]].g = new_g

                    ### Update the heurstic h()
                    self.cells[new_pos[0]][new_pos[1]].h = self.heuristic(new_pos)

                    ### Update the evaluation function for the cell n: f(n) to the new heuristic h(n)
                    self.cells[new_pos[0]][new_pos[1]].f = self.cells[new_pos[0]][new_pos[1]].h
                    self.cells[new_pos[0]][new_pos[1]].parent = current_cell

                    #### Add the new cell to the priority queue
                    open_set.put((self.cells[new_pos[0]][new_pos[1]].f, new_pos))
```

This implementation of the greedy best first search algorithm resulted in a true path cost of 18 compared to the 16 true path cost associated with the A* algorithm. That is because though GBFS makes each decision based on the lowest heuristic value, it does not provide an optimal solution.

A* Traversal:


g=3
h=13
g=4
h=12
g=5
h=11
g=6
h=10
g=7
h=9
g=8
h=8
g=9
h=7
g=10
h=6
g=11
h=5
g=12
h=4
g=13
h=3
g=14
h=2
g=15
h=1
g=16
h=0

GBFS Traversal:

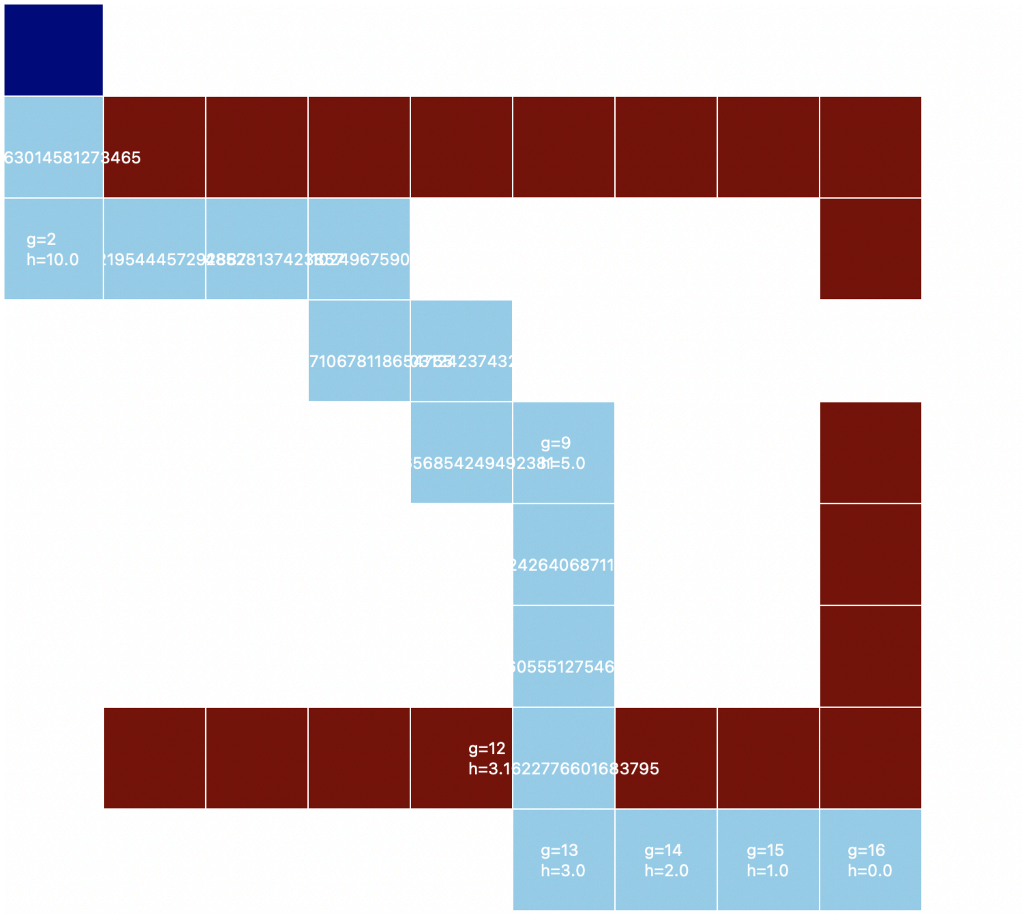


Problem 2:

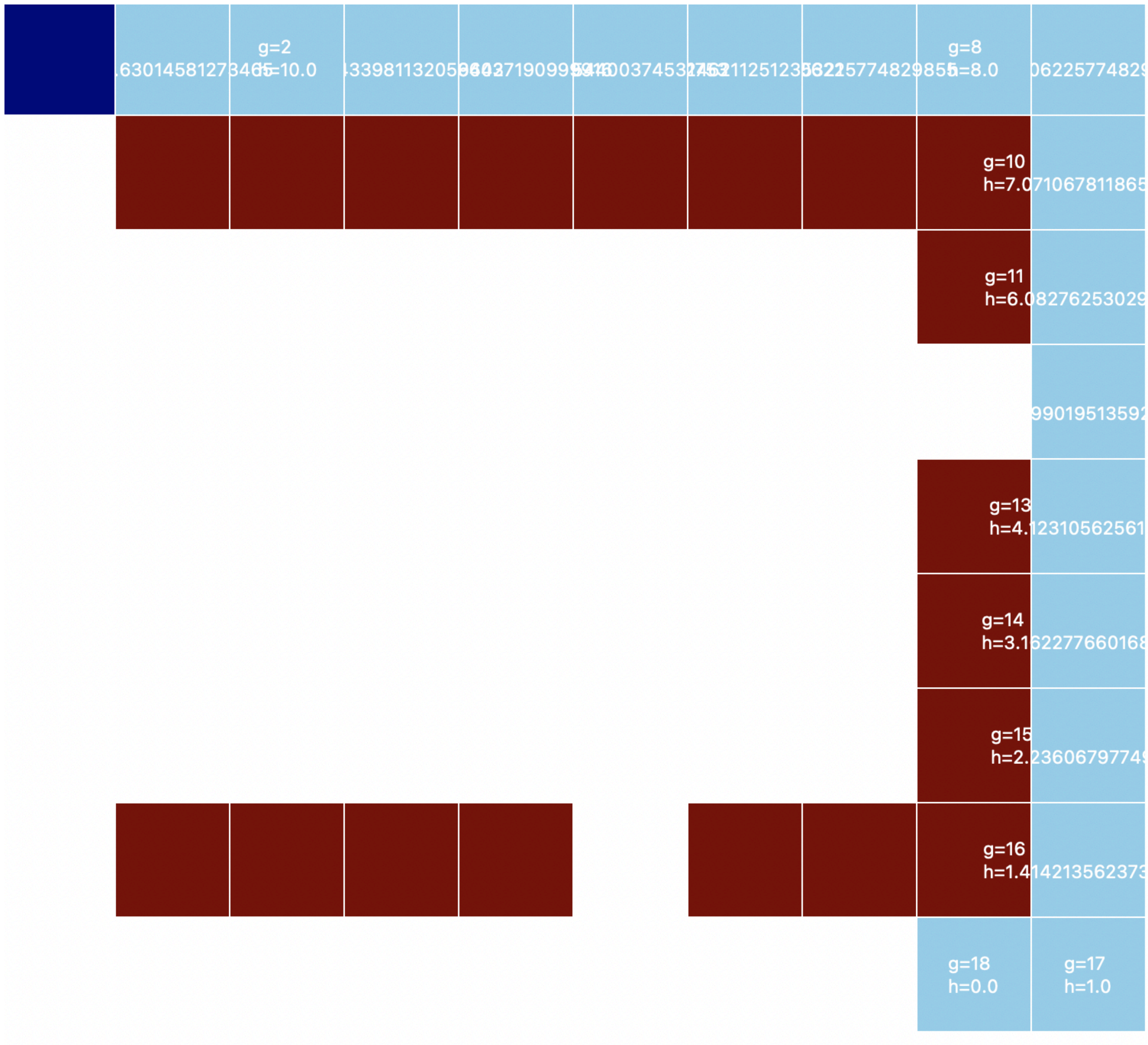
This problem asked us to update the heuristic to implement Euclidean distance rather than Manhattan distance. I wrote the following code to implement that heuristic:

```
############################################################
#### Euclidean distance
############################################################
def heuristic2(self, pos):
    # Calculate the difference in coordinates
    dx = pos[0] - self.goal_pos[0]
    dy = pos[1] - self.goal_pos[1]

    # Calculate the squared Euclidean distance
    squared_dist = dx**2 + dy**2

    # Return the square root of the squared distance (Euclidean distance)
    return math.sqrt(squared_dist)
```

A* with updated heuristic:



Greedy best first search with updated heuristic:



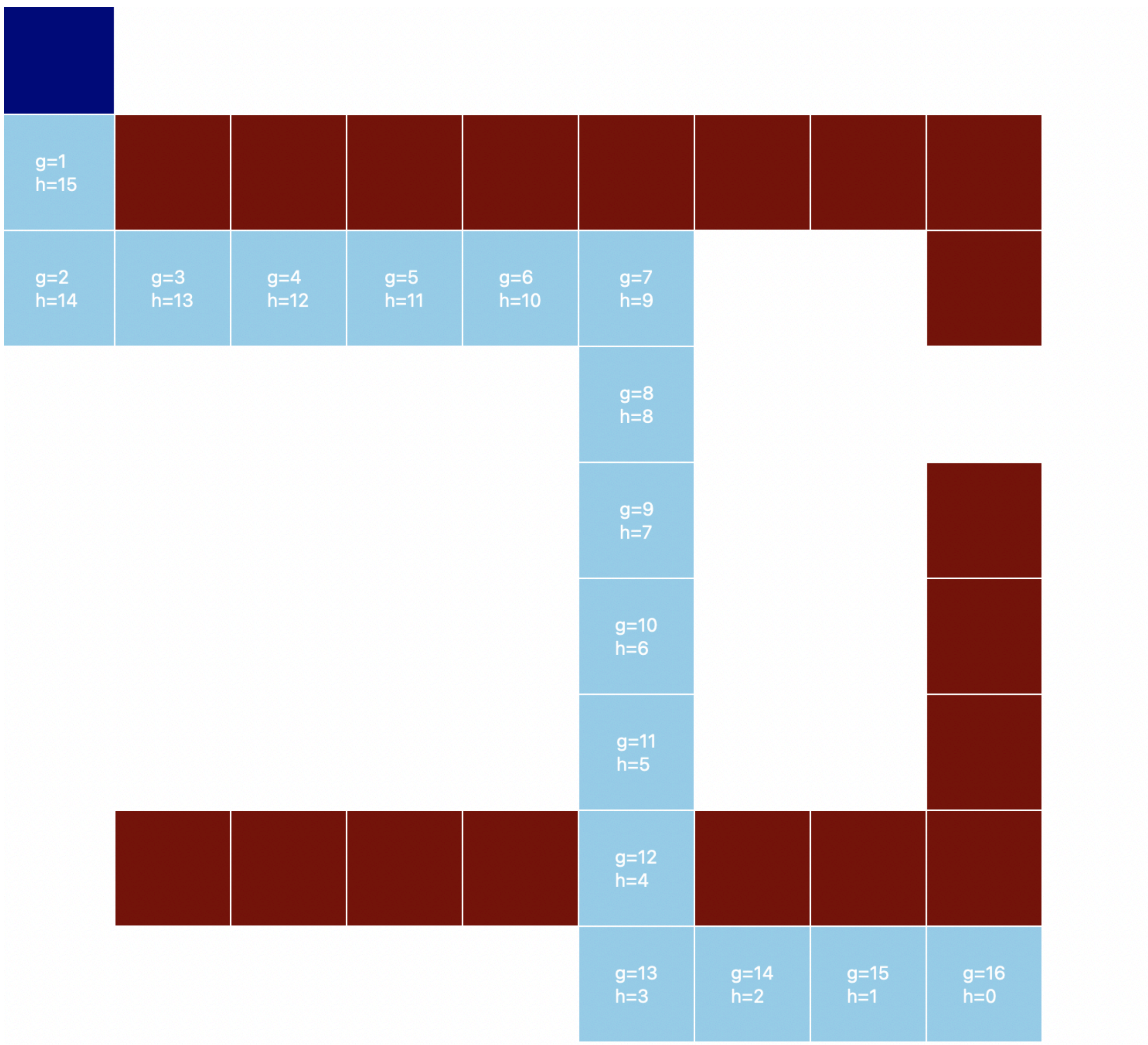
Problem 3

For this problem, we were asked to manipulate the evaluation function through weights on g(n) as denoted by α and on h(n) as denoted in the table below by β. When messing with the values of α and β it became obvious that depending on which variable is weighted heavier, that variable steers the optimum path towards that exhibited by the algorithm which relies on that variable. For instance, when h(n) is weighted more than g(n), that is the greedy best first search algorithm where g(n) is not considered.

When the weight on g(n) is higher, behavior is more in line with that of the A* algorithm. This also makes sense because A* is predicated upon optimum true path cost which would be emphasized in g(n) with a higher weight.

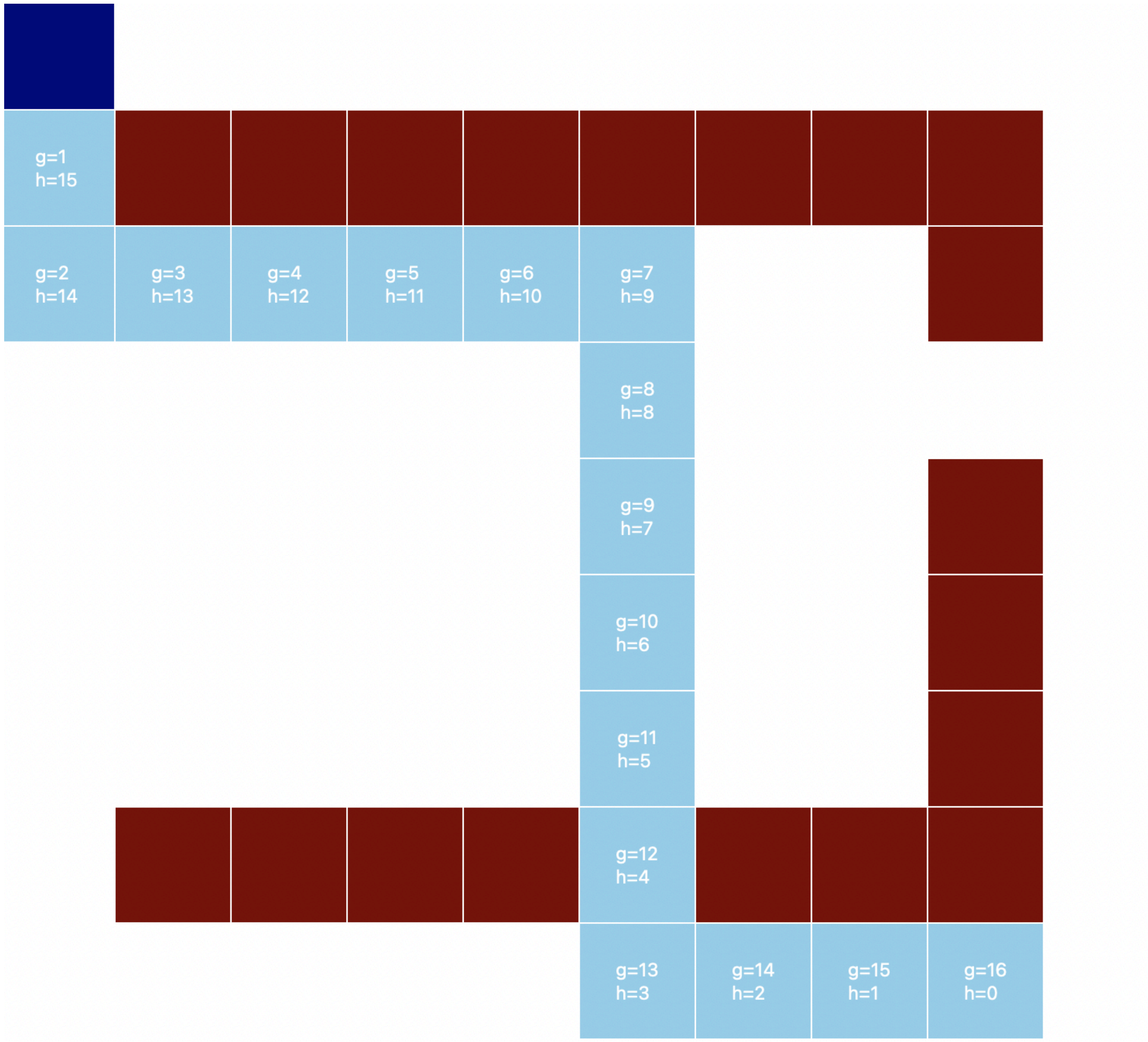
α	β	**Observed Behavior**
0	2	**GBFS**
1	2	**GBFS**
2	1	**A***
2	0	**A***

β = .5
This is closer to the optimum path exhibited by A*



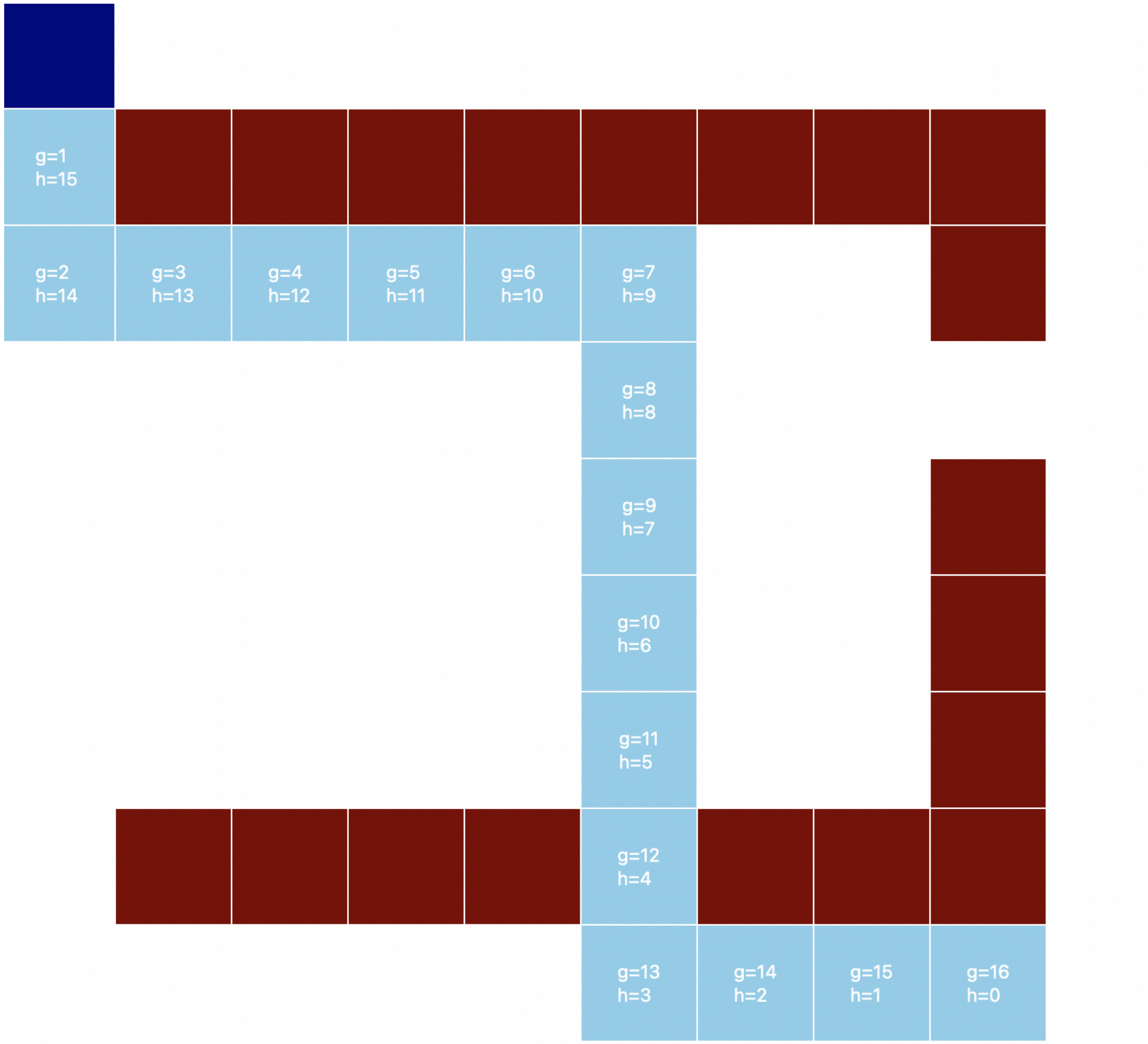
β = .75

This is closer to the optimum path as well.

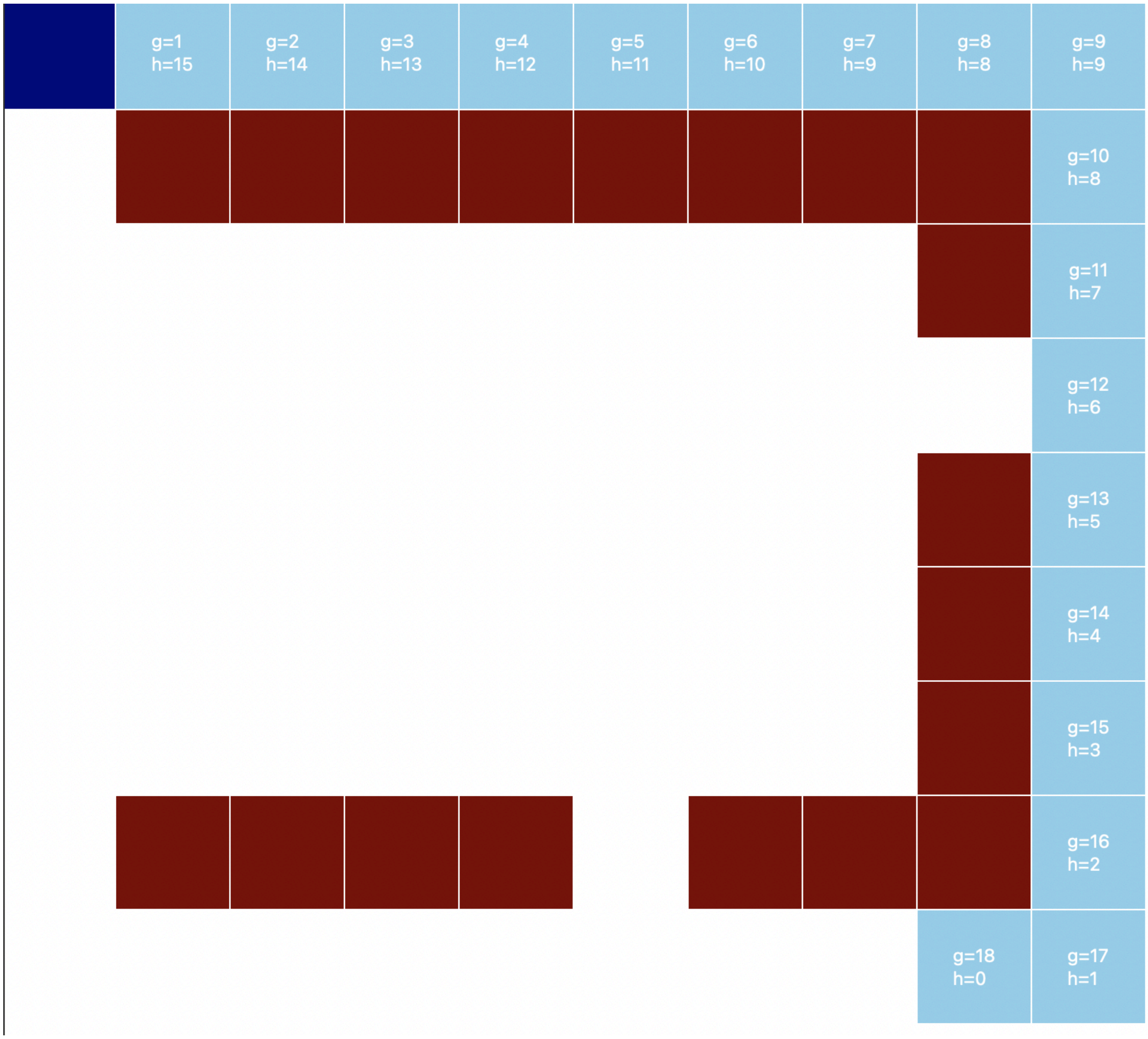


β = 1.25

Again, optimum path behavior



β = 1.50
At this point, the behavior shifts over to GBFS.



When I noticed that between a weight of 1.25 and 1.50 on h(n), the behavior switched, I decided to look into when exactly that switch occurred. Between 1.33 and 1.34 the behavior of the algorithm jumped from A* to GBFS.