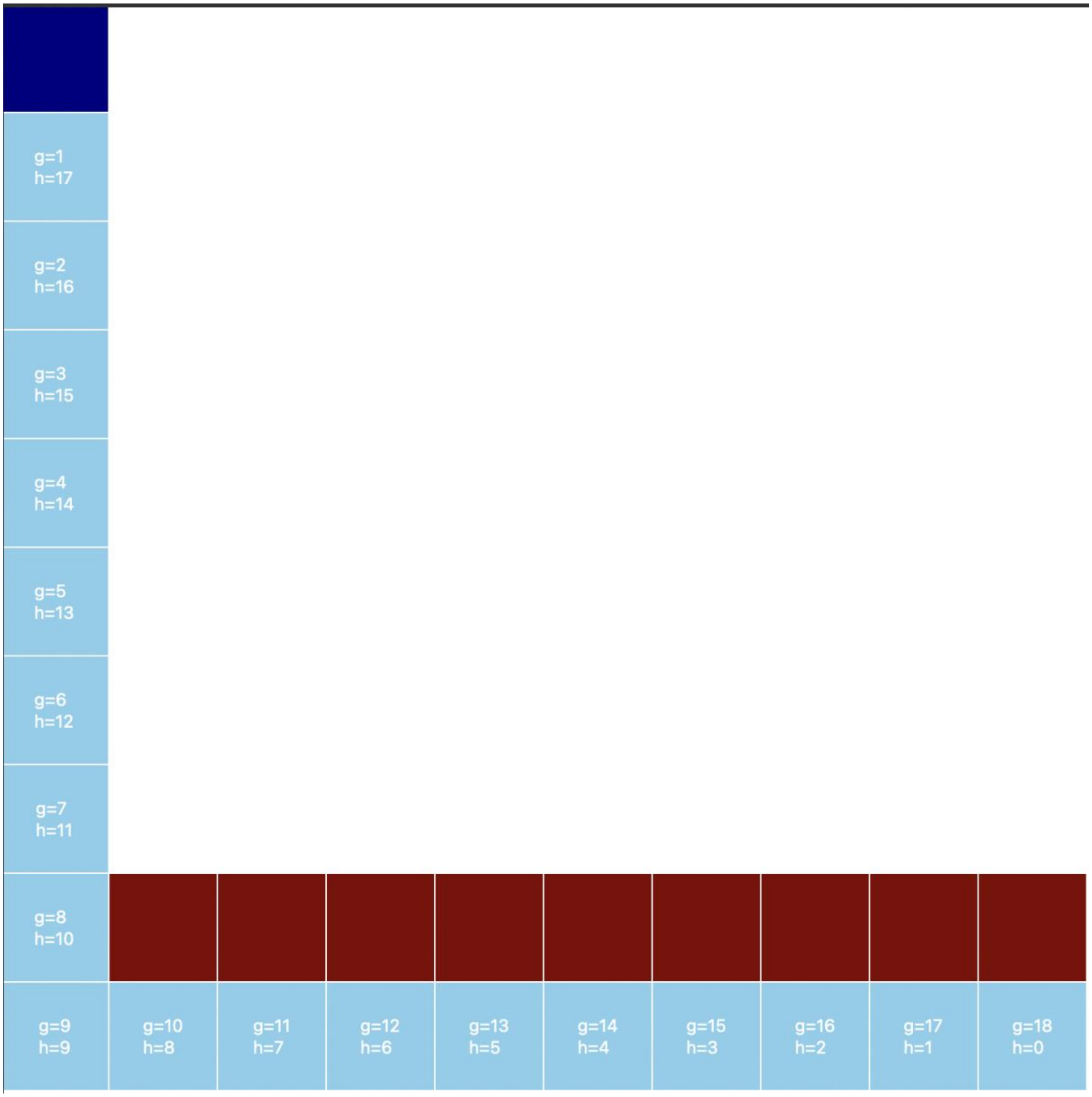
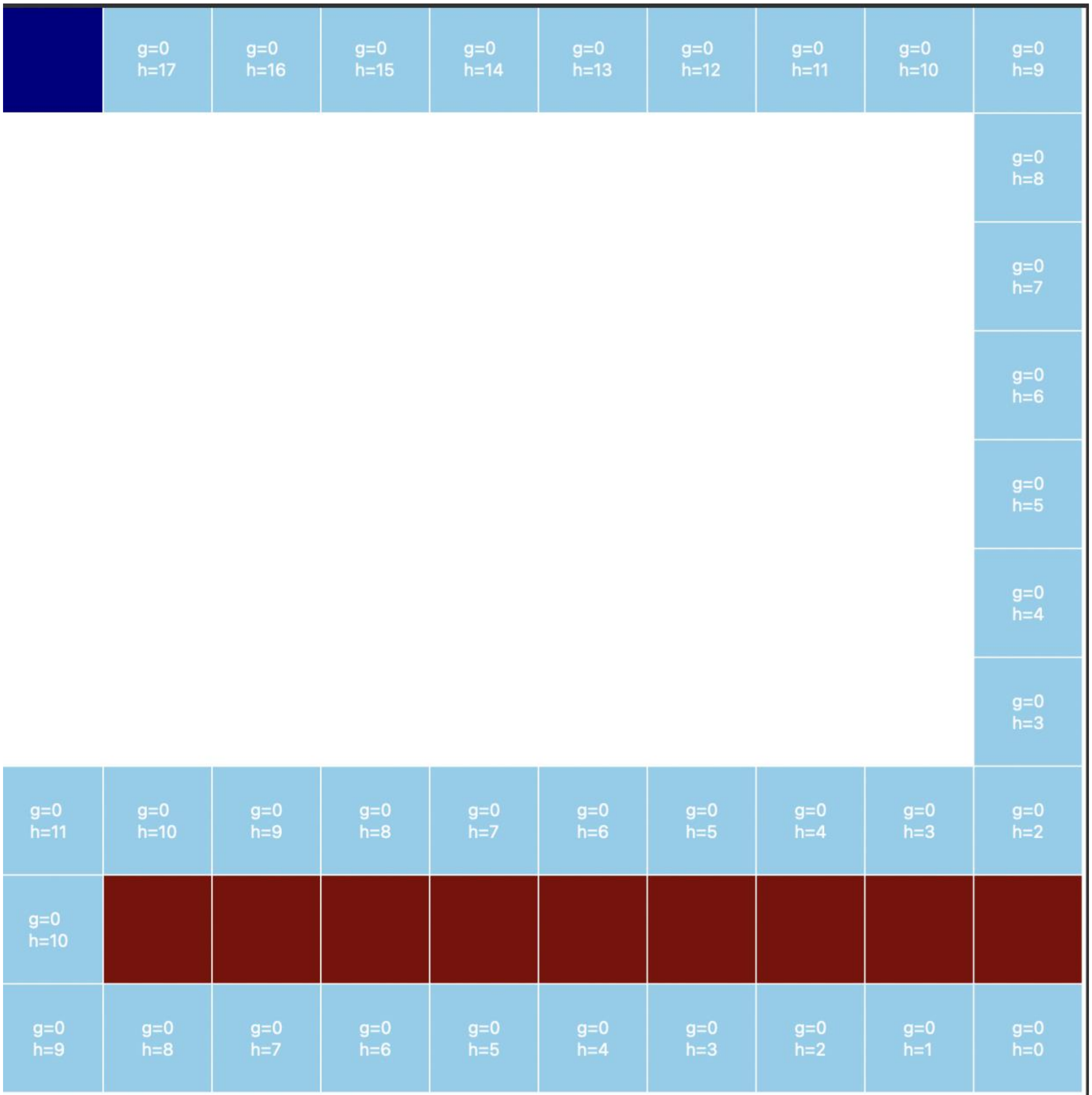
Question 1:

A* Algorithm



Explanation: In a A* search, because it takes into consideration both path cost and future cost, A* takes a more balanced approach. It expands nodes that balance shortest path so far g(n) with closeness to goal h(n).

Greedy Best-First



Explanation: In a Greedy Best-First search, it ignores path cost and only takes into consideration future cost. In doing so, the search attempts to “beeline” to the goal but is blocked by a wall. It is forced to travel along the wall until it isn’t blocked.

Removal of g(n)

```python
###############################################################
#### Greedy Best-First Algorithm
###############################################################
def find_path(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:

                #### Eliminates cost of moving to a new position
                new_g = current_cell.g

                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) = h(n)
                    self.cells[new_pos[0]][new_pos[1]].f = new_g + self.cells[new_pos[0]][new_pos[1]].h ## g(n) = 0 in search
                    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))
```

Explanation: To achieve a Greedy Best-First search, I altered the A* Algorithm in AStarMaze by changing this snippet of code:

new_g = current_cell.g + 1 → new_g = current_cell.g

By doing so, g(n) is constant and does not accumulate path cost and therefore is not calculated into the f(n) function. The function now looks like this:

f(n) = g(n) + h(n) → f(n) = h(n)

The algorithm no longer takes past cost g(n) into consideration, only concerning itself with the estimate of future cost h(n), making it greedy.

Conclusion: A* and Greedy algorithms differ in their best path to the goal. A* will take a balanced approach and consider both future and past costs. Greedy Best-First searches will result in a “beeline” to the goal, often resulting in inference with obstacles and resulting in a longer path. By making a small tweak to the code, by removing accumulation of g(n), will result in a Greedy Best-First algorithm.