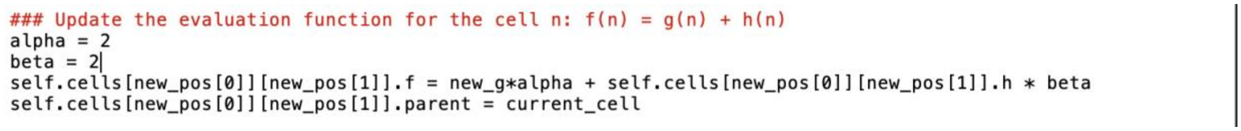
Question 3:

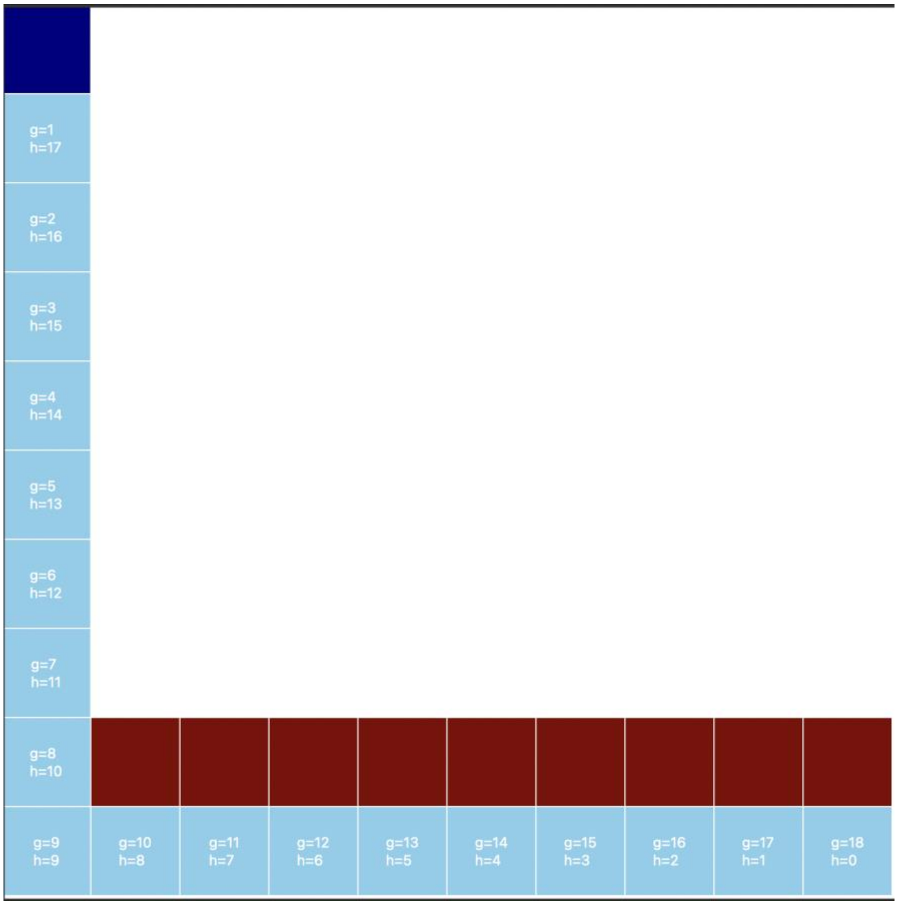
Table:

α	β	Observed observation
2	2	Moves like a balanced A* algorithm
0.5	1	Moves more greedily
0.5	10	Moves even more greedily
0	1	Greedy-best first
1	0.5	Tries to minimize true cost

Alpha = 2, Beta = 2

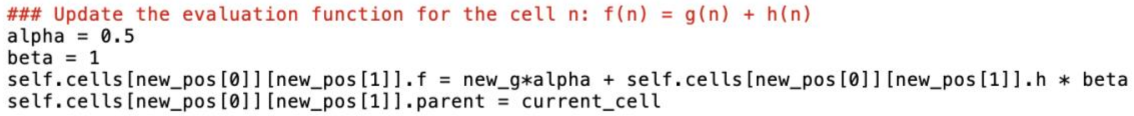


```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
alpha = 2
beta = 2
self.cells[new_pos[0]][new_pos[1]].f = new_g*alpha + self.cells[new_pos[0]][new_pos[1]].h * beta
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

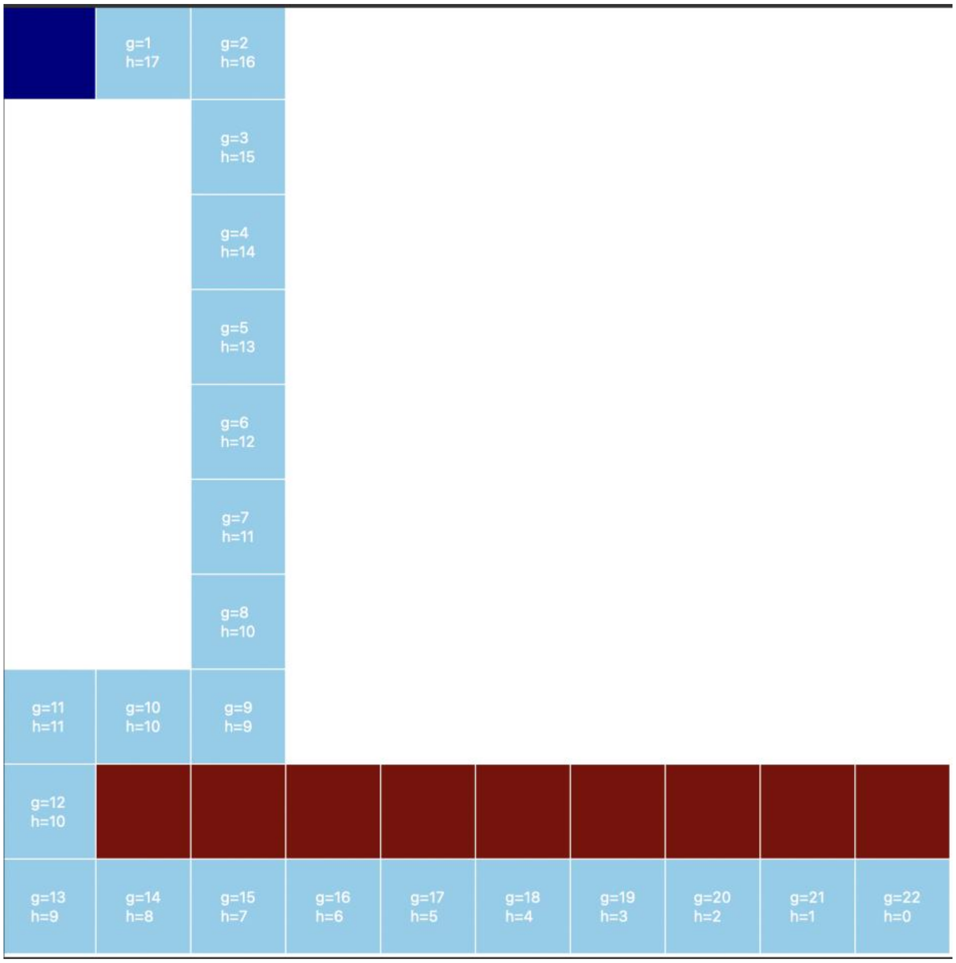


Explanation: Standard A* algorithm. When alpha and beta are equal to each other, A* does not change.

Alpha = 0.5, Beta = 1

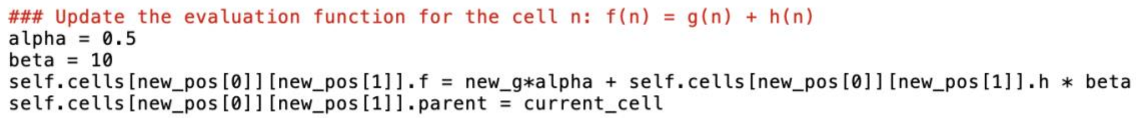


```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
alpha = 0.5
beta = 1
self.cells[new_pos[0]][new_pos[1]].f = new_g*alpha + self.cells[new_pos[0]][new_pos[1]].h * beta
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

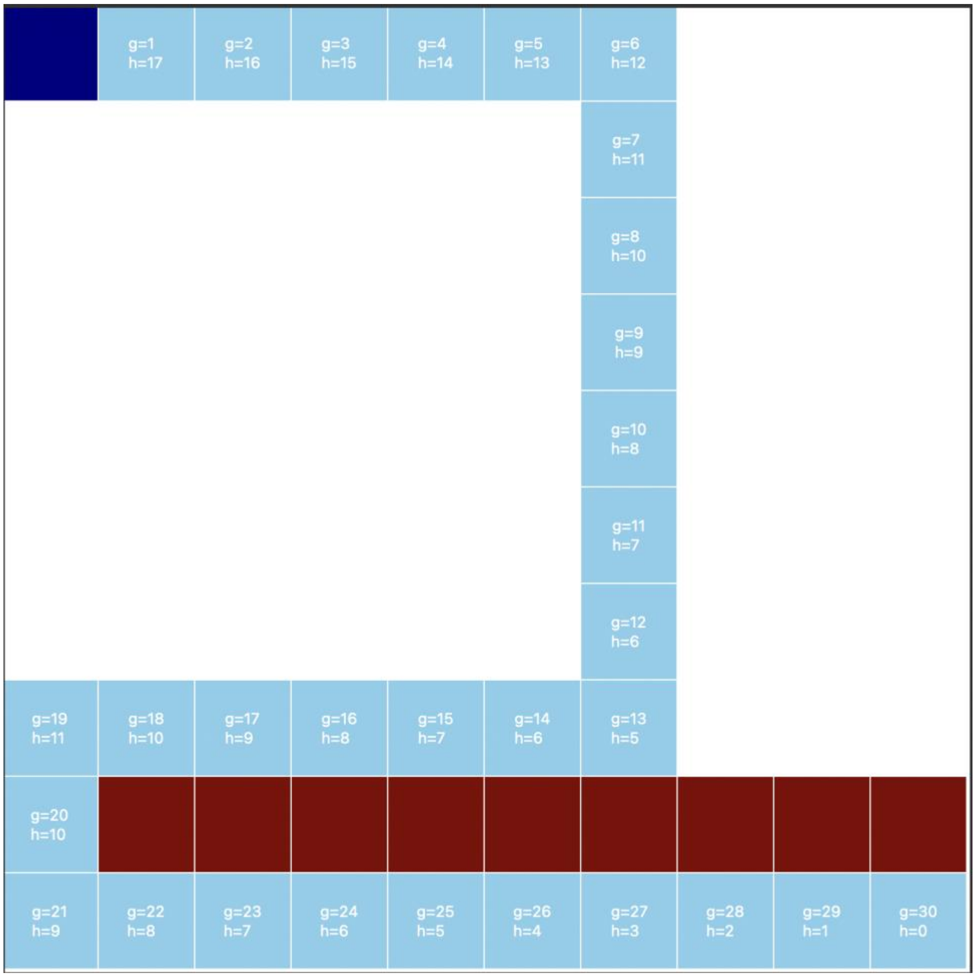


Explanation: When alpha is less than beta, the algorithm will move more greedily towards the goal. h() is weighted heavier, so it is favored in the search (aka moves more "greedy"). But as the search continues, g() accumulates and influences selection of nodes more, balancing out A* towards a less greedy algorithm.

Alpha = 0.5, Beta = 10



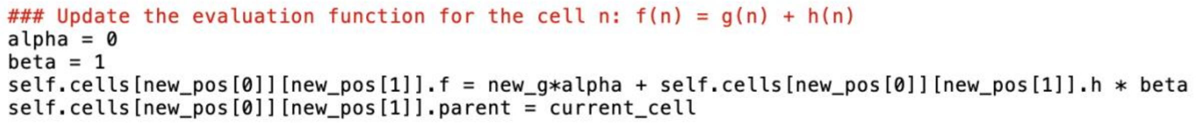
```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
alpha = 0.5
beta = 10
self.cells[new_pos[0]][new_pos[1]].f = new_g*alpha + self.cells[new_pos[0]][new_pos[1]].h * beta
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```



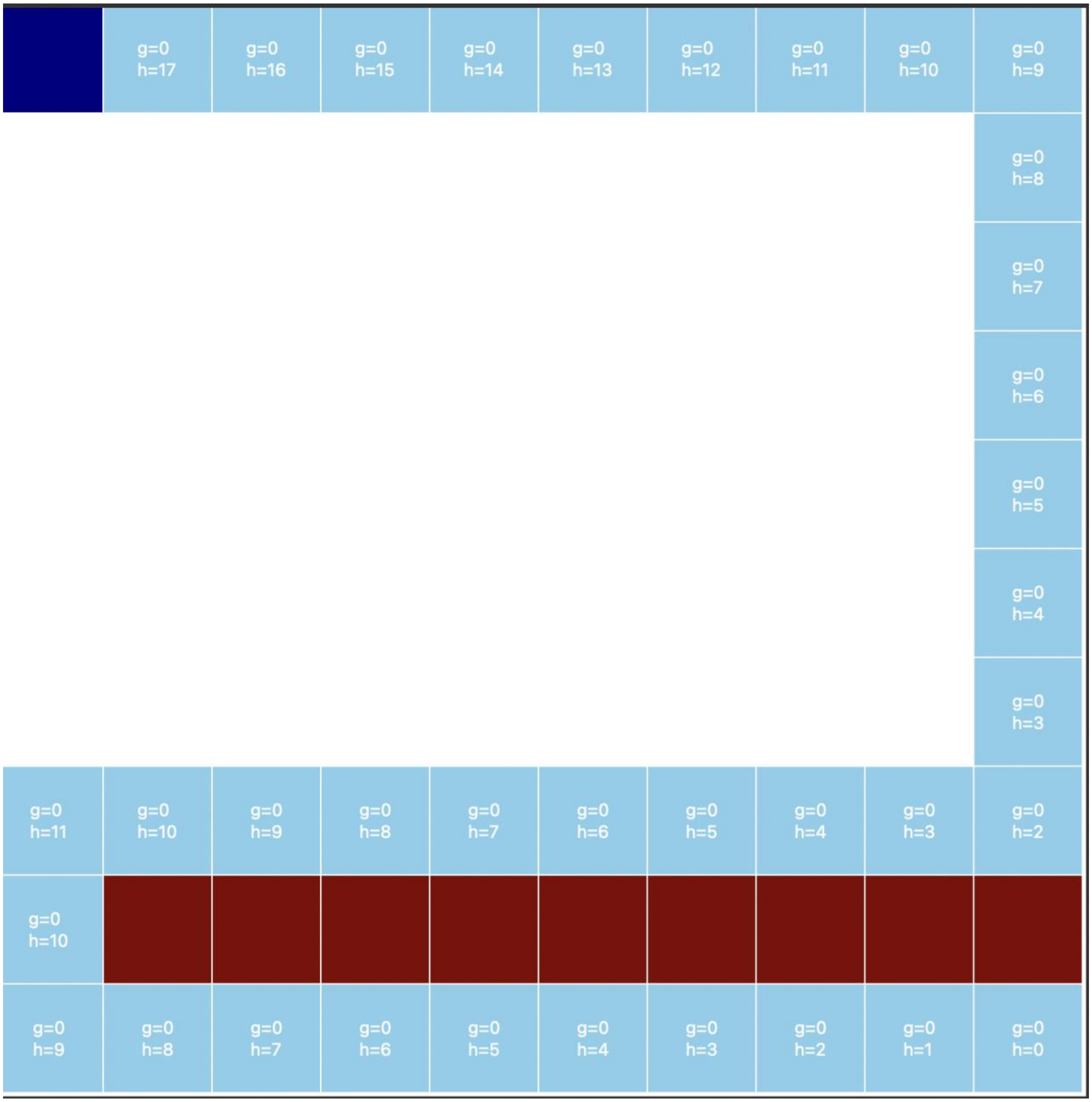
🟦	g=1 h=17	g=2 h=16	g=3 h=15	g=4 h=14	g=5 h=13	g=6 h=12			
						g=7 h=11			
						g=8 h=10			
						g=9 h=9			
						g=10 h=8			
						g=11 h=7			
						g=12 h=6			
g=19 h=11	g=18 h=10	g=17 h=9	g=16 h=8	g=15 h=7	g=14 h=6	g=13 h=5			
g=20 h=10	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
g=21 h=9	g=22 h=8	g=23 h=7	g=24 h=6	g=25 h=5	g=26 h=4	g=27 h=3	g=28 h=2	g=29 h=1	g=30 h=0

Explanation: Similar to the last observation, a heavier weighted h() makes the algorithm greedier. The greater the weight, the greedier is becomes (seen between the path between the previous graph and the current). But eventually, g() accumulates and balances out the algorithm towards being less greedy.

Alpha = 0, Beta = 1



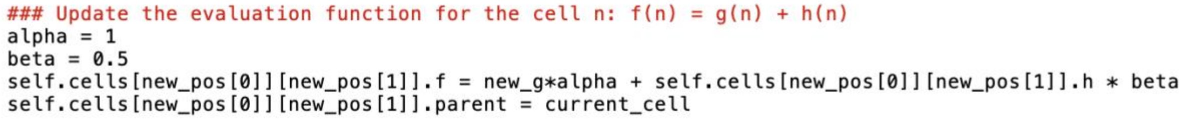
```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
alpha = 0
beta = 1
self.cells[new_pos[0]][new_pos[1]].f = new_g*alpha + self.cells[new_pos[0]][new_pos[1]].h * beta
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```



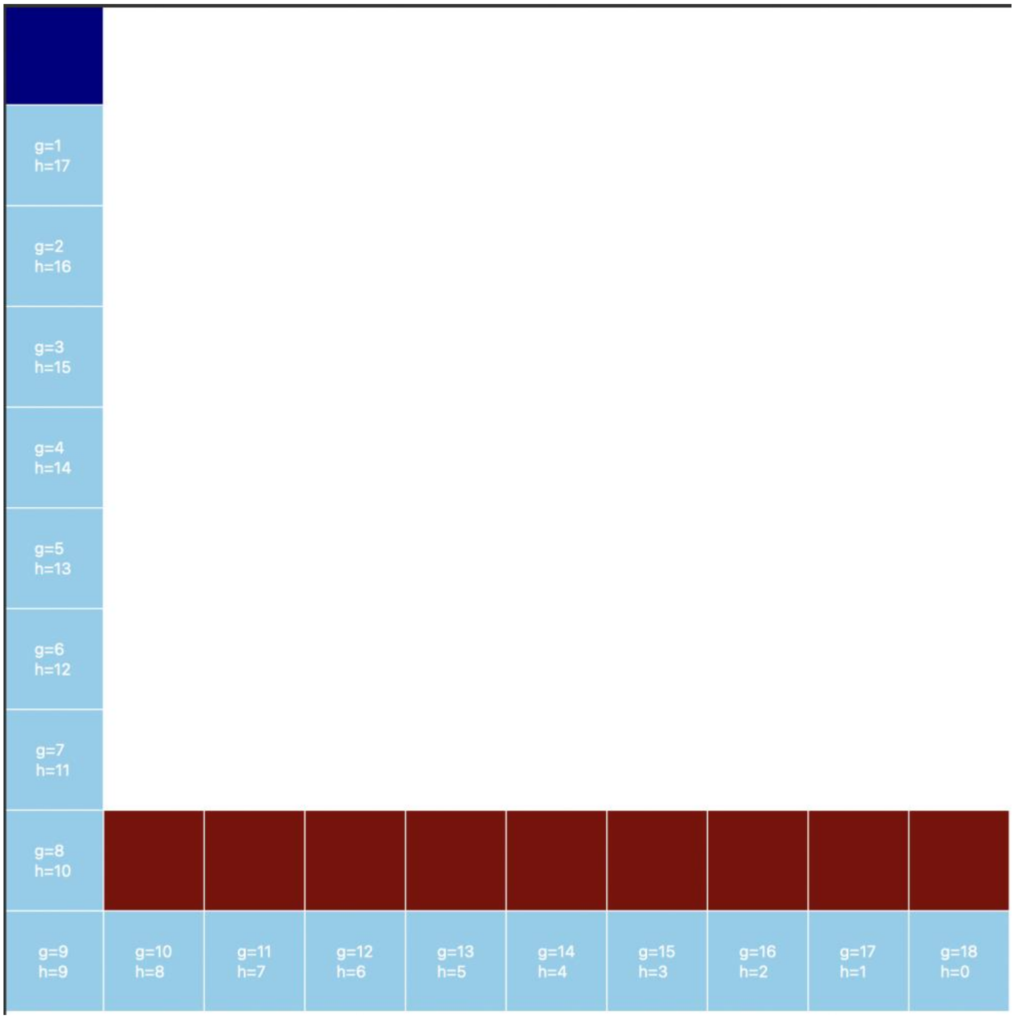
🟦	g=0 h=17	g=0 h=16	g=0 h=15	g=0 h=14	g=0 h=13	g=0 h=12	g=0 h=11	g=0 h=10	g=0 h=9
									g=0 h=8
									g=0 h=7
									g=0 h=6
									g=0 h=5
									g=0 h=4
									g=0 h=3
g=0 h=11	g=0 h=10	g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4	g=0 h=3	g=0 h=2
g=0 h=10	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4	g=0 h=3	g=0 h=2	g=0 h=1	g=0 h=0

Explanation: When alpha is equal to 0, the algorithm becomes a greedy best-first search. In this case, the algorithm only cares about the heuristic and not the path cost.

Alpha = 1, Beta = 0.5



```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
alpha = 1
beta = 0.5
self.cells[new_pos[0]][new_pos[1]].f = new_g*alpha + self.cells[new_pos[0]][new_pos[1]].h * beta
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```



Explanation: When alpha is weighted heavier than the beta in a weighted A* algorithm, the search will prioritize nodes that are the cheapest (similar to a Dijkstra algorithm). As the search continues, g() grows and dominates the evaluation making it more cost-driven.