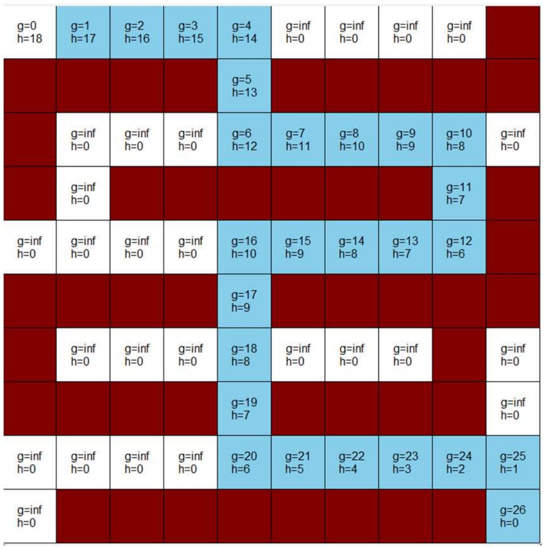
Name: Paul Anderson

Date: 19 October 2025

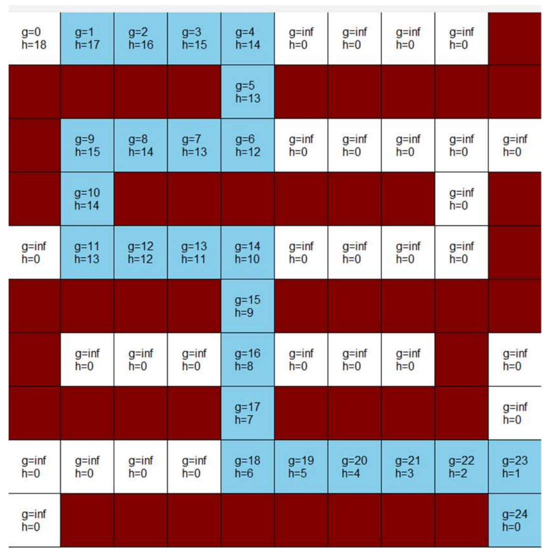
Purpose: *Experiment with Greedy Best-First Search and A* algorithm*

Problem 1:

Greedy Best-First Search (GBFS)



A* Search



GBFS tries to find the path it deems closest to the exit, even if it is not the best available path. Meanwhile A* analyses the path and factors in the g(n). GBFS analyses using f(n) = h(n), while A* is f(n) = g(n) + h(n). Adding the g(n), or the cost from the start to the current node, optimizes the algorithm and ensures the best possible path with a minimum cost is taken.

To Create GBFS algorithm, I simply modified the find_path code in the following way

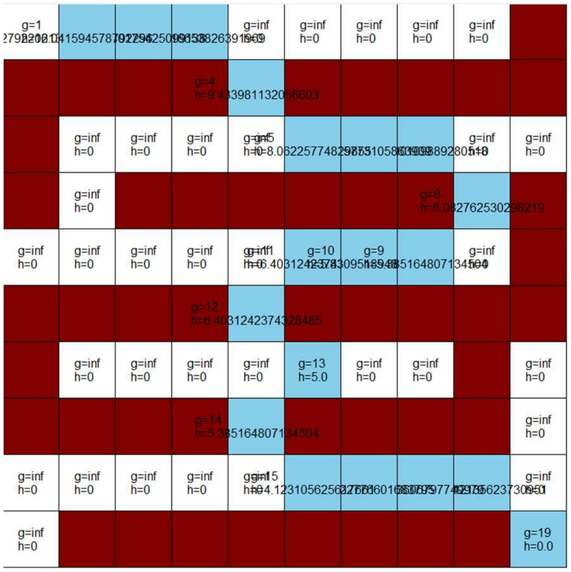
```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
self.cells[new_pos[0]][new_pos[1]].f = new_g + self.cells[new_pos[0]][new_pos[1]].h
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

Changed to, removing the ‘new_g’ section

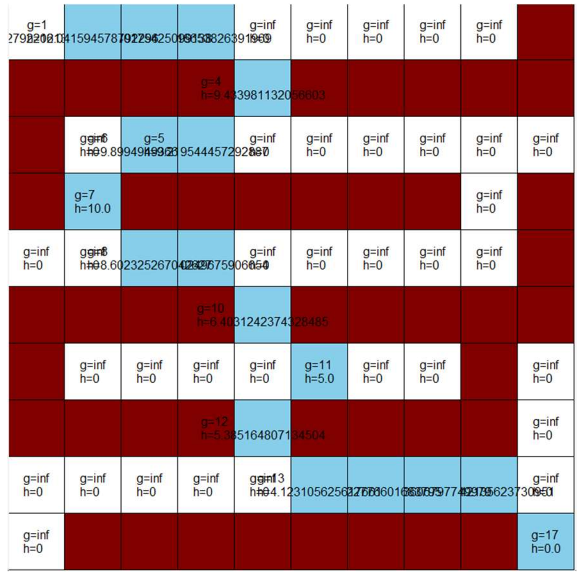
```
# Removing g(n) makes A* Greedy Best-First Search because it is f(n) = h(n)
self.cells[new_pos[0]][new_pos[1]].f = self.cells[new_pos[0]][new_pos[1]].h
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

Problem 2:

Greedy Best-First Search (GBFS)



A*



Similar to problem 1, GBFS tries to get to the end by choosing the closest possible h value to 0, while A* factors in the g value as well. Both g values are considerably smaller. With A* end g value ending at 17, the previous algorithm had a g value of 24. Similarly, GBFS had an ending g value of 19 and previously had a g value of 26.

For this problem, the two algorithm 'find_path' functions remained as they were with one exception

Both GBFS and A* used this modification

```
#### Agent goes E, W, N, and S, whenever possible
# Additionally, nw, ne, sw, and se have been added for Euclidean
for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0), (1,1), (-1,-1), (-1,1), (1,-1)]:
    new_pos = (current_pos[0] + dx, current_pos[1] + dy)
```

The 'heuristic' function was change to

```
def heuristic(self, pos):
    return (m.sqrt(((pos[0] - self.goal_pos[0]) ** 2) + ((pos[1] - self.goal_pos[1]) ** 2)))
    #return (abs(pos[0] - self.goal_pos[0]) + abs(pos[1] - self.goal_pos[1]))
```

From

```
def heuristic(self, pos):
    return (abs(pos[0] - self.goal_pos[0]) + abs(pos[1] - self.goal_pos[1]))
```

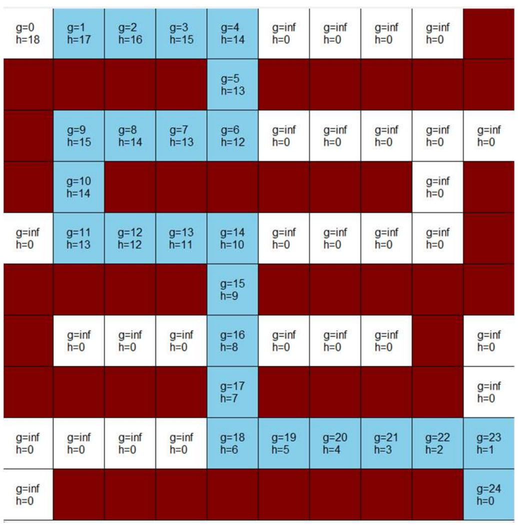
To use the Euclidean distance (first picture) function instead of the Manhattan distance function (second picture)

Problem 3:

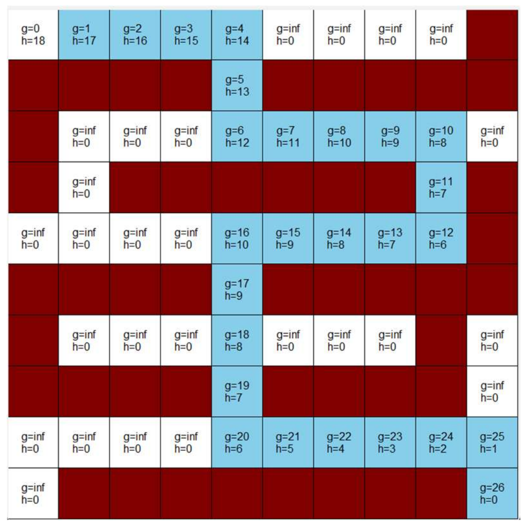
Alpha, the a, affects how much weight goes toward the total cost up to n. Bravo, the B, affects the algorithm in a way that makes it greedier, with B > a resembling GBFS. A > B resembling A* search. If the values are big enough and close enough together, the search will always resemble A* search algorithm

A	B	Observed Behavior
5	1	A* search pattern
5	10	GBFS pattern
2	1	A* search pattern
300	301	A* search pattern

A = 5 B = 1



A = 5 B = 10



See the problem 3 paragraph for an explanation of what is occurring.

The following variable were created for ease of access at the top of code

```
# Alfa and Bravo for the equation
# f(n) = alfa * g(n) + bravo * h(n) where bravo >= 0
alfa = 5
bravo = 10
```

The code in the ‘find_path” function was changed to

```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
# added Bravo and Alfa to equation: f(n) = alfa * g(n) + bravo * h(n) where bravo >= 0
self.cells[new_pos[0]][new_pos[1]].f = (alfa * new_g) + (bravo * self.cells[new_pos[0]][new_pos[1]].h)
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

From

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

In order to add a and b into the mix

Conclusion:

While both A* and GBFS are complete, only A* is optimal. When GBFS is optimized it becomes A*.