Path finding, Policies, and A* Algorithms

- Setting: Given a 2D Grid representing the World
  - Value of 1 Represents a Wall
  - Value of 0 Represents an open space
  - A point on Grid is designated as “Goal”
- Problem:
  - Determine the number of steps to reach the goal from any point on the Grid
  - Determine the shortest path to the goal given a starting point
  - Alter paths according to cost settings and policies

Definitions

- **Path Finding:** Determining the distance to goal from any open point on the Grid
- **Shortest Path:** Route that takes the least steps and has lowest cost
  - For our experiment – the cost will be set to 1 for all steps
- **Heuristic Estimate:** Determining base distance for each open point on the Grid. “Sufficient for Immediate Goal”
- **Cost Function:**
  - f(n) = g(n) + h(n)
  - Where f(n) is the cost, g(n) is the cost for moving one step, h(n) is the heuristic of (n)

A*: The process of finding the shortest route to the goal in a Grid or Graph

- Route Finding (Mapping Software)
- AI for games (Like PacMan!)
- Logistics Problems

Process for Finding Shortest Path (A*)

- Compute the Heuristic Model of the Grid
- Determine Policies for each point based on Cost

Our Grid (World)

- 2D Array in Python (Not Numpy, regular 2D List object)
- Value of 0 represents open space (Can be traversed)
- Value of 1 represents Wall or obstacle
- Goal is an [y][x] point on 2D Array.

gridA

```
gridA = [[0, 1, 0, 1, 0, 0],
         [0, 1, 0, 1, 0, 0],
         [0, 1, 0, 0, 0, 0],
         [0, 1, 0, 1, 0, 0],
         [0, 0, 0, 1, 0, 0]]

goalA = [len(gridA)-1, len(gridA[0])-1]
```

gridB

```
gridB = [[0, 1, 0, 1, 0, 1, 0],
         [0, 1, 0, 1, 0, 1, 0],
         [0, 0, 0, 1, 0, 1, 0],
         [0, 1, 0, 1, 0, 1, 0],
         [0, 1, 0, 0, 0, 0, 0],
         [0, 1, 0, 1, 0, 1, 0]]

goalB = [0, len(gridB[0])-1]
```

Goal: Build a Heuristic Model

```
gridA = [[0, 1, 0, 1, 0, 0],
         [0, 1, 0, 1, 0, 0],
         [0, 1, 0, 0, 0, 0],
         [0, 1, 0, 1, 0, 0],
         [0, 0, 0, 1, 0, 0]]

goalA = [len(gridA)-1, len(gridA[0])-1]
```

```
[13, 99, 7, 99, 5, 4]
[12, 99, 6, 99, 4, 3]
[11, 99, 5, 4, 3, 2]
[10, 99, 6, 99, 2, 1]
[9, 8, 7, 99, 1, 0]
```

Heuristic Model for gridB

```
gridB = [[0, 1, 0, 1, 0, 1, 0],
         [0, 1, 0, 1, 0, 1, 0],
         [0, 0, 0, 1, 0, 1, 0],
         [0, 1, 0, 1, 0, 1, 0],
         [0, 1, 0, 0, 0, 0, 0],
         [0, 1, 0, 1, 0, 1, 0]]

goalB = [0, len(gridB[0])-1]
```

```
[14, 99, 12, 99, 10, 99, 0]
[13, 99, 11, 99, 9, 99, 1]
[12, 11, 10, 99, 8, 99, 2]
[13, 99, 9, 99, 7, 99, 3]
[14, 99, 8, 7, 6, 5, 4]
[15, 99, 9, 99, 7, 99, 5]
```

Building a Heuristic Model

- What you are given:
  - Grid: A 2D Array to solve (0 -> Open Space)(1 -> Wall or barrier)
  - Goal: [y][x] point on Grid that represents the target
  - Cost: For this example the cost for all moves will be 1
  - Delta: A 2D array representing possible moves (y, x)

```
delta = [[-1, 0 ], # go up
         [ 0, -1], # go left
         [ 1, 0 ], # go down
         [ 0, 1 ]] # go right
```

  - Delta Name: Icons representing ‘moves’

```
delta_name = ['^', '<', 'v', '>']
```

What you must Do

- Complete the function `def compute_value(grid, goal, cost)`
  - Returns a 2D array ‘value’ that contains the heuristic steps to goal

```
gridB = [[0, 1, 0, 1, 0, 1, 0],
         [0, 1, 0, 1, 0, 1, 0],
         [0, 0, 0, 1, 0, 1, 0],
         [0, 1, 0, 1, 0, 1, 0],
         [0, 1, 0, 0, 0, 0, 0],
         [0, 1, 0, 1, 0, 1, 0]]

goalB = [0, len(gridB[0])-1]
```

Input: gridB

```
[14, 99, 12, 99, 10, 99, 0]
[13, 99, 11, 99, 9, 99, 1]
[12, 11, 10, 99, 8, 99, 2]
[13, 99, 9, 99, 7, 99, 3]
[14, 99, 8, 7, 6, 5, 4]
[15, 99, 9, 99, 7, 99, 5]
```

Output: value

Algorithm: Compute Heuristic Value(grid, goal, cost)

Compute Heuristic Value(grid, goal, cost) (Non Recursive)

Initialize value as blank Array (not numpy)
Copy grid into value
Replace value 1 with 99 in value

Initialize visited as empty Array
Initialize count = 0
Initialize x = x position of goal
Initialize y = y position of goal
Intialize point = [count, y, x]
Append visited with point

Algorithm: Compute Heuristic Value(grid, goal, cost) (Continued)

```
Initialize allChecked = false
Initialize index = 0
count = 1

While (not allChecked):
        currentPoint = visited[index]
        index = index + 1
        // Populate Neighbors
        Initialize neighborList to empty Array
        For each move in delta:
                newPoint = currentPoint + move // Move in direction from delta
                if newPoint in grid and not barrier and not already visited:
                        Place newPoint in neighborList
```

Algorithm: Compute Heuristic Value(grid, goal, cost) (Continued)

```
// Check neighborList
for each n in neighborList:
        newX = x of n
        newY = y of n
        c = value[currentPoint] + 1
        newPoint = [c, newY, newX]
        Append visited with newPoint
        value[newY][newX] = c // Assign heuristic

// Check if all visited – This is the exit condition
if index > length of visited – 1:
        allChecked = true
// End while loop
```

Algorithm: Compute Heuristic Value(grid, goal, cost) (Continued)

Replace all 0 values with 99 in value

Set Coordinate of Goal in value to 0

Return value

// End Algorithm Compute Heuristic Value

Algorithm: Compute Heuristic (Recursive)

value = Copy of grid
Replace ‘1’ in value with ‘99’
call fillHeuristic(value, start, point, cost)
Adjust value by -1 where value != 99
return value

#---

```
def fillHeuristic(value, start, point, cost):

    value[point] = value[start] + cost
        Initialize neighbors as empty list
        for d in delta
            Initialize newPoint = point + d  #adjust y and x positions by delta
            if newPoint in Grid
                if value[newPoint] == 0 or ( value[newPoint] != 99 and value[newPoint] > value[point]+1)
                    Append neighobors with newPoint

    if length of neighbors < 1
        return
    else:
        for n in neighbors
            fillHeuristic(value, point, n, cost)   # Recursive Call
```

Module for ps10.py

- Download `ps10.py` at: Problem Set 10
- Complete Code for `def compute_value(grid,goal,cost):`
  - You may define a recursive function to complete the above function
- Test on at least 4 Different Grids and Goals
- Email `ps10.py` module to Mr. Michaud
- Good Luck!

Next Step: A* Algorithm . . .

```
[13, 99, 7, 6, 5, 4]
[12, 99, 99, 7, 99, 3]
[11, 10, 9, 8, 99, 2]
[12, 99, 99, 99, 99, 1]
[13, 99, 99, 99, 99, 0]
```

Heuristic

```
['v', ' ', '>', '>', '>', 'v']
['v', ' ', ' ', '^', ' ', 'v']
['>', '>', '>', '^', ' ', 'v']
['^', ' ', ' ', ' ', ' ', 'v']
['^', ' ', ' ', ' ', ' ', '*']
```

A* Policies