Recall from Last Lecture

- Localization
- Markov Localization
- Probabilities
- Bayes rule for measurement update

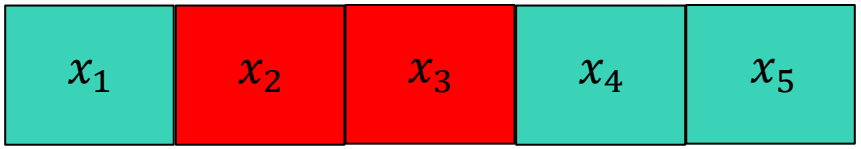
$$\bar{P}(x_i|Z) = P(Z|x_i)P(x_i)$$

$$\alpha = \sum \bar{P}(x_i|Z)$$

$$P(x_i|Z) = \frac{\bar{P}(x_i|Z)}{\alpha}$$

- Total probability for motion update

$$P(x_i^t) = \sum_j P(x_j^{t-1})P(x_i|x_j)$$



Sense: Z = red
P(Z|R) = 0.8 and
P(Z|G) = 0.2

Move: U = 2

$$P(x_{i+2}|x_i) = 0.8$$
$$P(x_{i+1}|x_i) = 0.1$$
$$P(x_{i+3}|x_i) = 0.1$$

Localization with using Python: Measurement Update

- Python code

Sense: Z = red
P(Z|R) = 0.8 and
P(Z|G) = 0.2

$$\bar{P}(x_i|Z) = P(Z|x_i)P(x_i)$$

$$\alpha = \sum \bar{P}(x_i|Z)$$

$$P(x_i|Z) = \frac{\bar{P}(x_i|Z)}{\alpha}$$

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
Z = 'red'
pHit = 0.8
pMiss = 0.2

def sense(p, Z):
    q=[]
    sum = 0.0
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
        sum += q[i]

    for i in range(len(q)):
        q[i] = q[i]/sum

    return q

print (sense(p,Z))
```

- Result

```
[0.0909090909090909, 0.3636363636363636, 0.3636363636363636, 0.0909090909090909,
 0.0909090909090909]
>>>
```

Localization with using Python: Move Update

- Python code

Move: U=1

$$P(x_{i+1}|x_i) = 0.8$$
$$P(x_i|x_i) = 0.1$$
$$P(x_{i+2}|x_i) = 0.1$$

$$P(x_i^t) = \sum_j P(x_j^{t-1}) P(x_i|x_j)$$

```
p=[0, 1, 0, 0, 0]
world=['green', 'red', 'red', 'green', 'green']
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def move(p, U):
    if (U == 0):
        return p
    q = []
    for i in range(len(p)):
        q.append(p[(i-U-1)%len(p)]*pOvershoot+
                 p[(i-U)%len(p)]*pExact+
                 p[(i-U+1)%len(p)]*pUndershoot)
    return q

print (move(p, 1))
```

- Result

```
[0.0, 0.1, 0.8, 0.1, 0.0]
>>>
```

Localization with using Python: Move & Sense Update

- Python code

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'red', 'green']
motions = [0,1,1]
pHit = 0.8
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
```

Localization with using Python: Move & Sense Update

- Python code

```
def move(p, U):
    if (U == 0):
        return p
    q = []
    for i in range(len(p)):
        s = pExact * p[(i-U) % len(p)]
        s = s + pOvershoot * p[(i-U-1) % len(p)]
        s = s + pUndershoot * p[(i-U+1) % len(p)]
        q.append(s)
    return q

for k in range(len(measurements)):
    p = move(p, motions[k])
    p = sense(p, measurements[k])

print (p)
```

- Result

```
[0.07327429333980347, 0.01759068300376077, 0.06963484168385298, 0.6177362610699988,
0.221763920902584041
>>>
```

Programming Assignment 1: Multiple Measurements

PA1A-MultipleMeasurements.py

```
#Modify the code so that it updates the probability twice
#and gives the posterior distribution after both
#measurements are incorporated. Make sure that your code
#allows for any sequence of measurement of any length.

p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.8
pMiss = 0.2

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
#
#ADD YOUR CODE HERE


#
print (p)
```

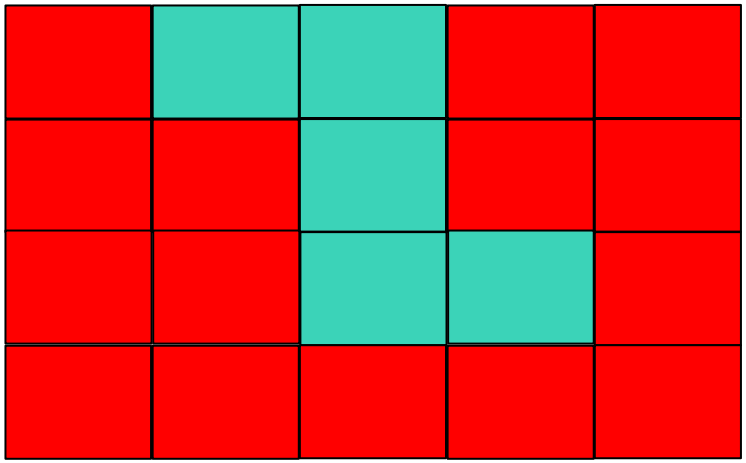
Programming Assignment 2: Move 1000 Times

PA1B-Move1000Times.py

```
#write code that moves 1000 times and then prints the
#resulting probability distribution.

p=[0, 1, 0, 0, 0]
world=['green', 'red', 'red', 'green', 'green']
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def move(p, U):
    if(U==0):
        return p
    q = []
    for i in range(len(p)):
        q.append(p[(i-U-1)%len(p)]*pOvershoot+
                 p[(i-U)%len(p)]*pExact+
                 p[(i-U+1)%len(p)]*pUndershoot)
    return q

# ADD CODE HERE


#
print (move(p, 1))
```

Programming Project #1: Localization in 2D

- Two dimensional grid

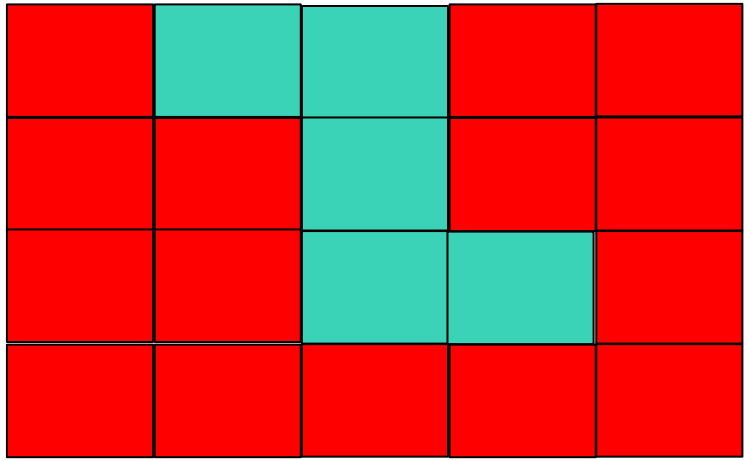


- Begin with uniform distribution
- Measurement
  - pHit = sensor_right
  - pMiss = 1 - pHit
- Motion encoding
  - [0,0] – no move
  - [0,1] – move right
  - [0,-1] – move left
  - [1,0] – move down
  - [-1,0] – move up
- Move
  - No overshot
    - pExact = p_move
    - pOff = 1 - pExact

Programming Project #1: Localization in 2D

PP1-Localization2D.py

```
measurements = ['green', 'green', 'green' ,'green', 'green']

p = [[1./20, 1./20, 1./20, 1./20, 1./20],
     [1./20, 1./20, 1./20, 1./20, 1./20],
     [1./20, 1./20, 1./20, 1./20, 1./20],
     [1./20, 1./20, 1./20, 1./20, 1./20]]

# Motion encoding
# [0,0] - no move
# [0,1] - move right
# [0,-1] - move left
# [1,0] - move down
# [-1,0] - move up

motions = [[0,0],[0,1],[1,0],[1,0],[0,1]]

sensor_right = 0.7

p_move = 0.8

def show(p):
    for i in range(len(p)):
        print (p[i])
```



```
pHit = sensor_right
pMiss = 1 - pHit
```

```
pExact = p_move
pOff = 1 - pExact
```

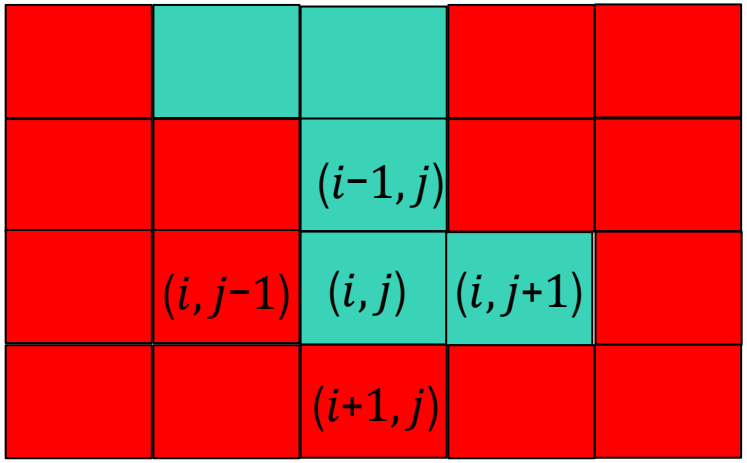
Assignment Summary

- Two Programming Assignments
  - PA1A: Multiple Measurements:
    - Download: PA1A-MultipleMeasurements.py
  - PA1B: Move 1000 Times
    - Download: PA1B-Move1000Times.py
- One Programming Project #1
  - Download PP1-Localization2D.py
  - Define sense function and move function
  - Write a loop and call the functions move and sense inside the loop body.
- Submission
  - Submit the three Python source code in Canvas

Programming Project #1: Localization in 2D

Hint for move function move(p, U)

```
p = [[1./20, 1./20, 1./20, 1./20, 1./20],
     [1./20, 1./20, 1./20, 1./20, 1./20],
     [1./20, 1./20, 1./20, 1./20, 1./20],
     [1./20, 1./20, 1./20, 1./20, 1./20]]

# Motion encoding
# [0,0] - no move
# [0,1] - move right
# [0,-1] - move left
# [1,0] - move down
# [-1,0] - move up

motions = [[0,0],[0,1],[1,0],[1,0],[0,1]]
```




```
pExact = p_move
pOff = 1 - pExact
```

How to update p[i, j] after executing a motion command U

pExact * $p[(i - U[0])\%len(p)][(j - U[1])\%len(p[i])]$ + pOff * $p[i][j]$