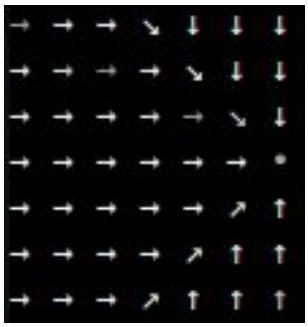
**Case 1:**

Value Function
[-6, -5, -4, -3, -3, -3, -3]
[-6, -5, -4, -3, -2, -2, -2]
[-6, -5, -4, -3, -2, -1, -1]
[-6, -5, -4, -3, -2, -1,  0]
[-6, -5, -4, -3, -2, -1, -1]
[-6, -5, -4, -3, -2, -2, -2]
[-6, -5, -4, -3, -3, -3, -3]

Example Policy Map (others exist, see second π*)



I store the action that maximizes utility at each iteration in a second matrix. Its actually pretty fun to watch these converge on an optimal path if you print at each iteration.

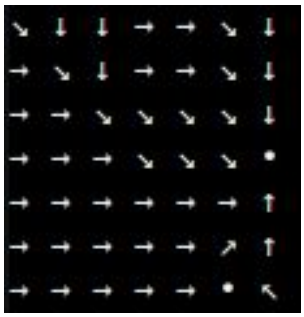
π*(3,0) = [
  (3,0) →
  (3,1) →
  (3,2) →
  (3,3) →
  (3,4) →
  (3,5) →
  (3,6) •
]

another π*(3,0) = [
  (3,0) →
  (3,1) →
  (3,2) ↘
  (4,3) →
  (4,4) →
  (4,5) ↗
  (3,6) •

**Case 2:**

Value Function
[-6, -6, -6, -5, -4, -3, -3]
[-6, -5, -5, -5, -4, -3, -2]
[-6, -5, -4, -4, -3, -2, -1]
[-6, -5, -4, -3, -2, -1,  0]
[-6, -5, -4, -3, -2, -1, -1]
[-6, -5, -4, -3, -2, -1, -2]
[-6, -5, -4, -3, -2, -2, -2]

Policy Map



This one is my favorite. The (6,5) stay happens b/c of the wind!

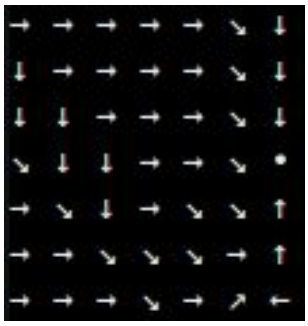
π*(3,0) = [
  (3,0) →
  (3,1) →
  (3,2) →
  (3,3) ↘
  (3,4) ↘
  (3,5  ↘
  (3,6) •
]

another π*(3,0) = [
  (3,0) ↘
  (4,1) →
  (4,2) →
  (4,3) →
  (3,4) ↘
  (3,5) ↘
  (3,6) •
]

**Case 3:**

Value Function
[-8, -7, -6, -5, -4, -3, -3]
[-8, -7, -6, -5, -4, -3, -2]
[-7, -7, -6, -5, -4, -3, -1]
[-6, -6, -6, -5, -4, -2,  0]
[-6, -5, -5, -5, -3, -1, -1]
[-6, -5, -4, -4, -2, -1, -2]
[-6, -5, -4, -3, -2, -1, -2]

Policy Map



Notice the ← at s = (6,6)!! The algorithm purposefully goes into the wind to exploit the -2 row effect that will take place on the NEXT move. Makes it look like the algorithm is 'thinking ahead'! So cool.

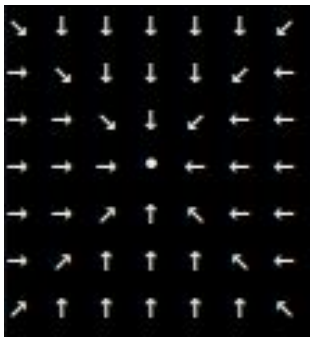
π*(3,0) = [
  (3,0) ↘
  (4,1) ↘
  (5,2) ↘
  (6,3) ↘
  (5,4) ↘
  (4,5) ↘
  (3,6) •
]
I dont see another π* starting from (0,3).
from (0,3) there is only 1 best option, -5 at (4,1).
from (4,1) there is only 1 best option, -4 at (5,2).
from (5,2) there is only 1 best option, -3 at (6,3).
from (6,3) there are two -2's, (5,4) and (6,4) but with win (6,4) cant be reached so we have to pick (5,4).
from (5,4) there are three -1's nearby but two of them cant be reached factoring in wind, we pick (3,5).
from (3,5) the best we can get is 0 and we pick that.

**Extra cases:**

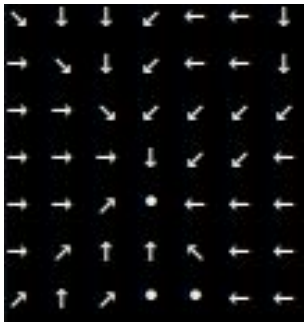
Moving the 0 reward state + no wind
Wind = 0
Reward function =

```
if s[0] == 3 and s[1] == 3:
    return 0
else:
    return -1
```

Policy Map:



Moving the 0 reward state + case2_wind
Wind = 1
Reward function = same as just above
Value Function:
[-3, -3, -3, -3, -4, -5, -5]
[-3, -2, -2, -3, -4, -5, -4]
[-3, -2, -1, -2, -3, -4, -3]
[-3, -2, -1,  0, -1, -2, -3]
[-3, -2, -1, -1, -1, -2, -3]
[-3, -2, -2, -1, -1, -2, -3]
[-3, -3, -2, -2, -2, -2, -3]
Policy Map:



Two 0 reward states+ no wind
Wind = 0
Reward function =

```
if (s[0] == 1 and s[1] == 1) or (s[0] == 5 and s[1] == 5):
    return 0
else:
    return -1
```

Policy Map: