Ricky Ma 82943424
Bruce Cui 13412151
Late days claimed: 0

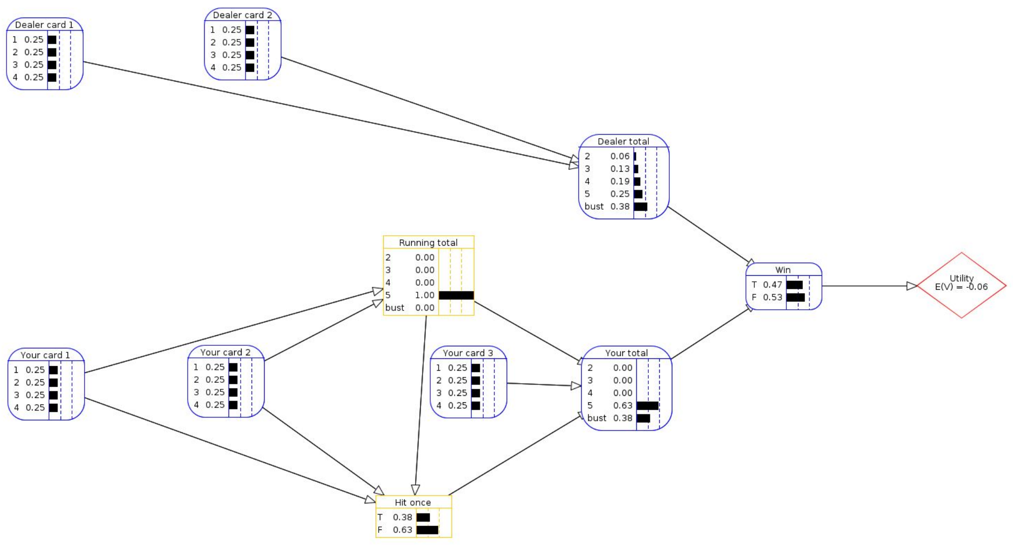
Question 1: Value of Information and Value of Control

a. Value of information for *DC1* = (max expected utility, knowing DC1) - (expected utility, **not** knowing DC1)
   - Expected utility of not knowing DC1 = -0.25

   Setting observed values of DC1 to 1, 2, 3, 4, respectively, and calculating expected utility:
   - Maximum expected utility of knowing DC1 = -0.23437

   Value of information for DC1 = -0.23437 - (-0.25) = 0.01563

b. Value of control for *running total* = (expected utility when RN is a decision variable) - (expected utility when RN is a random variable)
   - Expected utility when RN is a random variable = -0.25
   - Network changed:
     - Running total set as a decision variable
     - No-forgetting arc added: running total -> hit once
     - Decisions are optimized



   - Expected utility when RN is a decision variable = -0.0625

   Value of control for *running total* = -0.0625 - (-0.25) = 0.1875

c. Value of control for *DC2*
   - Expected utility when DC2 is a random variable = -0.25
   - Expected utility when DC2 is a decision variable = -0.0625
   - Value of control for *DC2* = -0.0625 - (-0.25) = 0.1875

Example for the first row: given that you know your first two cards are (1, 1), the optimal policy is for the dealer's card 2 to be 4.

Your card 1	Your card 2	DC2 (optimal policy)
1	1	4
1	2	4
1	3	4
1	4	1
2	1	4
2	2	4
2	3	1
2	4	1
3	1	4
3	2	1
3	3	1
3	4	1
4	1	1
4	2	1
4	3	1
4	4	1

Question 2: Value Iteration

a. U^1((10,8)) = -0.1 + 0.9*max(

0.7*0 + 0.1*0 + 0.1*0 + 0.1*0,	UP
0.7*0 + 0.1*0 + 0.1*0 + 0.1*0,	LEFT
0.7*0 + 0.1*0 + 0.1*0 + 0.1*0,	DOWN
0.8*0 + 0.1*0 + 0.1*0)	RIGHT = -0.1 + 0.9*0 = -0.1

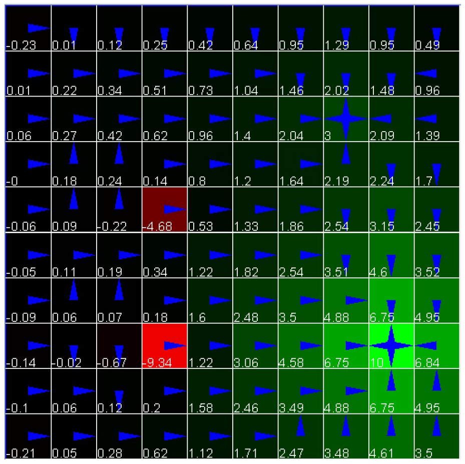
b. U^2((10,8)) = -0.1 + 0.9*max(

0.7*-0.1 + 0.1*10 + 0.1*-0.1 + 0.1*-0.1,	UP
0.7*10 + 0.1*-0.1 + 0.1*-0.1 + 0.1*-0.1,	LEFT
0.7*-0.1 + 0.1*10 + 0.1*-0.1 + 0.1*-0.1,	DOWN
0.8*-0.1 + 0.1*-0.1 + 0.1*-0.1)	RIGHT = -0.1 + 0.9*max(0.91, 6.97, 0.91, -0.1) = 6.173

- U^2((9,9)) is larger than U^((10,8)) because (10,8) is closer to a wall (the right one, specifically) and bumping into a wall has a penalty of -1. Due to the wall, the values of (10,7) and (10,9) are both -0.1 after the first iteration. Since the agent may move up, down, and right with a probability of 0.1 when moving left, the values of (10,7) and (10,9) end up affecting the value of (10,8) in the second iteration.
- Likewise, because (9,9) is near two walls while (9,7) is only near one, U^2((9,7)) > U^2((9,9)). When calculating U^2((9,9)), the values in the states around (9,9) are used. These have two values of -0.1, since (9,9) is near a corner, which ends up affecting the value of (9,9) in the second iteration.

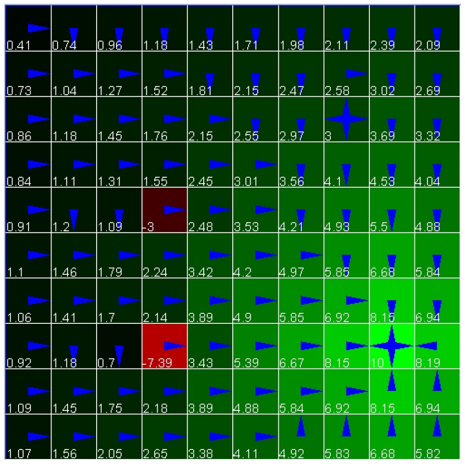
c. Importance of the discount factor

i. 0.8:



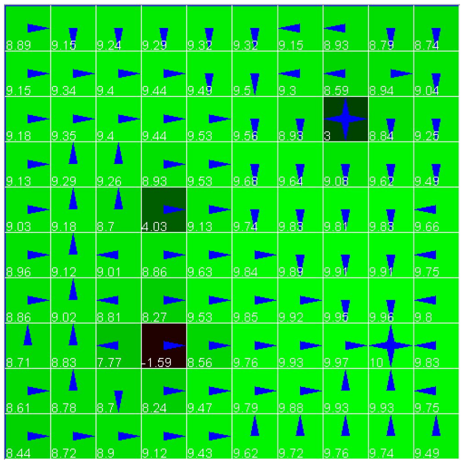
→ -0.23	↓ 0.01	↓ 0.12	↓ 0.25	↓ 0.42	↓ 0.64	↓ 0.95	↓ 1.29	↓ 0.95	↓ 0.49
→ 0.01	→ 0.22	→ 0.34	→ 0.51	→ 0.73	→ 1.04	↓ 1.46	↓ 2.02	↓ 1.48	← 0.96
→ 0.06	→ 0.27	→ 0.42	→ 0.62	→ 0.96	→ 1.4	→ 2.04	✛ 3	← 2.09	← 1.39
→ -0	↑ 0.18	↑ 0.24	→ 0.14	→ 0.8	→ 1.2	→ 1.64	↑ 2.19	↓ 2.24	↓ 1.7
→ -0.06	↑ 0.09	↑ -0.22	→ -4.68	→ 0.53	→ 1.33	→ 1.86	↓ 2.54	↓ 3.15	↓ 2.45
→ -0.05	→ 0.11	→ 0.19	→ 0.34	→ 1.22	→ 1.82	→ 2.54	↓ 3.51	↓ 4.6	↓ 3.52
→ -0.09	↑ 0.06	↑ 0.07	→ 0.18	→ 1.6	→ 2.48	→ 3.5	→ 4.88	↓ 6.75	↓ 4.95
→ -0.14	↓ -0.02	↓ -0.67	→ -9.34	→ 1.22	→ 3.06	→ 4.58	→ 6.75	✛ 10	← 6.84
→ -0.1	→ 0.06	↓ 0.12	→ 0.2	→ 1.58	→ 2.46	→ 3.49	→ 4.88	↑ 6.75	↑ 4.95
→ -0.21	→ 0.05	→ 0.28	→ 0.62	→ 1.12	→ 1.71	↑ 2.47	↑ 3.48	↑ 4.61	↑ 3.5

0.9:



→ 0.41	↓ 0.74	↓ 0.96	↓ 1.18	↓ 1.43	↓ 1.71	↓ 1.98	↓ 2.11	↓ 2.39	↓ 2.09
→ 0.73	→ 1.04	→ 1.27	→ 1.52	↓ 1.81	↓ 2.15	↓ 2.47	→ 2.58	↓ 3.02	↓ 2.69
→ 0.86	→ 1.18	→ 1.45	→ 1.76	→ 2.15	↓ 2.55	↓ 2.97	✛ 3	↓ 3.69	↓ 3.32
→ 0.84	→ 1.11	→ 1.31	→ 1.55	→ 2.45	→ 3.01	↓ 3.56	↓ 4.1	↓ 4.53	↓ 4.04
→ 0.91	↓ 1.2	↓ 1.09	→ -3	→ 2.48	→ 3.53	↓ 4.21	↓ 4.93	↓ 5.5	↓ 4.88
→ 1.1	→ 1.46	→ 1.79	→ 2.24	→ 3.42	→ 4.2	→ 4.97	↓ 5.85	↓ 6.68	↓ 5.84
→ 1.06	→ 1.41	→ 1.7	→ 2.14	→ 3.89	→ 4.9	→ 5.85	→ 6.92	↓ 8.15	↓ 6.94
→ 0.92	↓ 1.18	↓ 0.7	→ -7.39	→ 3.43	→ 5.39	→ 6.67	→ 8.15	✛ 10	← 8.19
→ 1.09	→ 1.45	→ 1.75	→ 2.18	→ 3.89	→ 4.88	→ 5.84	→ 6.92	↑ 8.15	↑ 6.94
→ 1.07	→ 1.56	→ 2.05	→ 2.65	→ 3.38	→ 4.11	↑ 4.92	↑ 5.83	↑ 6.68	↑ 5.82

0.999:



→ 8.89	↓ 9.15	↓ 9.24	↓ 9.29	↓ 9.32	↓ 9.32	← 9.15	← 8.93	↓ 8.79	↓ 8.74
→ 9.15	→ 9.34	→ 9.4	→ 9.44	↓ 9.49	↓ 9.5	← 9.3	← 8.59	→ 8.94	↓ 9.04
→ 9.18	→ 9.35	→ 9.4	→ 9.44	→ 9.53	↓ 9.56	↓ 8.93	✛ 3	↓ 8.84	↓ 9.25
→ 9.13	↑ 9.29	↑ 9.26	→ 8.93	→ 9.53	↓ 9.68	↓ 9.64	↓ 9.08	↓ 9.62	↓ 9.49
→ 9.03	↑ 9.18	↑ 8.7	→ 4.03	→ 9.13	↓ 9.74	↓ 9.83	↓ 9.81	↓ 9.83	← 9.66
→ 8.96	↑ 9.12	← 9.01	→ 8.86	→ 9.63	→ 9.84	↓ 9.89	↓ 9.91	↓ 9.91	← 9.75
→ 8.86	↑ 9.02	← 8.81	→ 8.27	→ 9.53	→ 9.85	→ 9.92	↓ 9.96	↓ 9.96	← 9.8
↑ 8.71	↑ 8.83	← 7.77	→ -1.59	→ 8.56	→ 9.76	→ 9.93	→ 9.97	✛ 10	← 9.83
→ 8.61	↑ 8.78	↓ 8.7	→ 8.24	→ 9.47	→ 9.79	→ 9.88	↑ 9.93	↑ 9.93	← 9.75
→ 8.44	→ 8.72	→ 8.9	→ 9.12	→ 9.43	↑ 9.62	↑ 9.72	↑ 9.76	↑ 9.74	↑ 9.49

ii. When discount=0.8, agents near (8,3) are incentivized to settle on (8,3) since the cost of traveling to state (9,8) is greater than staying at (8,3). When discount=0.9, agents are more likely to avoid (8,3) since the expected reward of traveling to (9,8) is greater than the expected reward of staying at (8,3). When discount=0.999, there is nearly no punishment for remaining in the environment, so agents are highly incentivized to avoid (8,3) and pursue (9,8).

iii. When discount=0.8, the value of (4,8) is so low that agents must avoid the state at all costs because the reward of reaching the terminal state would likely not offset this penalty, since the reward is heavily discounted. There is a 10% probability that an agent travelling RIGHT at (4,7) may move DOWN, resulting in the optimal policies of both (2,7) and (3,7) to be UP. When discount=0.9, the agents must still avoid (4,8), but the risk of travelling next to (4,8) on row 7 is "worth it" because the reward agent would receive in (9,8) would offset this risk. When discount=0.999, the cost of remaining in the environment is so low that agents are incentivized to travel *all the way around* to reach state (9,8).

Question 3: Belief State Update in POMDPs

For all questions below, the last output matrix represents the belief state of the agent after performing a1:n and observing e1:n. The unit of each value in a matrix is unitless because the values represent probabilities.

o (up, up , up) (2,2,2)

```
[[0.11111111 0.11111111 0.11111111 0.        ]
 [0.11111111 0.         0.11111111 0.        ]
 [0.11111111 0.11111111 0.11111111 0.11111111]]


[[0.3266129  0.18145161 0.03427419 0.        ]
 [0.18145161 0.         0.01814516 0.        ]
 [0.03629032 0.18145161 0.00403226 0.03629032]]


[[0.51464878 0.20399879 0.00751349 0.        ]
 [0.07352125 0.         0.00063033 0.        ]
 [0.02450708 0.16791891 0.00272301 0.00453835]]


[[5.68370409e-01 2.25729132e-01 3.13373244e-03 0.00000000e+00]
 [3.59526444e-02 0.00000000e+00 2.60973127e-04 0.00000000e+00]
 [2.01639170e-02 1.43620352e-01 2.00793722e-03 7.60902527e-04]]
```

Output makes sense because intuitively after performing 3 up actions and still seeing 2 walls usually means the agent is stuck in the top left corner or in cell (2,1) or (2,3). Thus, we see these probabilities of being in these three cells are high in the final belief state.

o (up, up, up) (1,1,1)

```
[[0.11111111 0.11111111 0.11111111 0.        ]
 [0.11111111 0.         0.11111111 0.        ]
 [0.11111111 0.11111111 0.11111111 0.11111111]]


[[0.05921053 0.03289474 0.50328947 0.        ]
 [0.03289474 0.         0.26644737 0.        ]
 [0.00657895 0.03289474 0.05921053 0.00657895]]


[[1.27646642e-02 1.27140107e-02 8.57967784e-01 0.00000000e+00]
 [1.82352345e-03 0.00000000e+00 1.02573194e-01 0.00000000e+00]
 [6.07841151e-04 5.06534292e-03 5.47057036e-03 1.01306858e-03]]


[[1.97953610e-03 1.35387011e-02 9.64447334e-01 0.00000000e+00]
 [1.18475976e-04 0.00000000e+00 1.83362729e-02 0.00000000e+00]
 [7.89839839e-05 6.48797010e-04 7.61631273e-04 9.02674102e-05]]
```

Output makes sense because we observed 1 wall 3 times after moving up which means we are probably in the third column, because the probability of observing 1 wall in the 3rd column non terminal states is 0.9, compared to 0.1 in other non terminal states. In addition, the cell (3,3) has a really high probability (close to 1) because the agent probably believes it was first starting out from someplace in column 3 (because the first observation is “1 wall”), and the probability of moving “up” correctly and staying in column 3 is high (0.8).

o (right, right, up) (1,1,end) with S0 = (2,3)

```
[[0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]


[[0.         0.01369863 0.98630137 0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]]


[[0.         0.00073046 0.52593134 0.        ]
 [0.         0.         0.4733382  0.        ]
 [0.         0.         0.         0.        ]]


[[0.         0.         0.         0.52631579]
 [0.         0.         0.         0.47368421]
 [0.         0.         0.         0.        ]]
```

Intuitively output makes sense because "end" is the last observation, so the agent must be in one of the two terminal states. In addition, the probability of being in (4,3) is higher than (4,2) because based on the second last belief state, we can see that the agent was likely in (3,3) or (3,2) prior to performing the last "up" action, which caused the agent to move "right." In the second last belief state, we see the agent has higher probability being in (3,3) than (3,2), so in the last belief state (4,3) has higher probability than (4,2)

o (up, right, right, right) (2,2,1,1) with S0 = (1,1)

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [1. 0. 0. 0.]]


[[0.  0.  0.  0. ]
 [0.8 0.  0.  0. ]
 [0.1 0.1 0.  0. ]]


[[0.0861244  0.         0.         0.        ]
 [0.69976077 0.         0.         0.        ]
 [0.09688995 0.1076555  0.00956938 0.        ]]


[[0.05090782 0.04022346 0.         0.        ]
 [0.3375     0.         0.00502793 0.        ]
 [0.04650838 0.05782123 0.4575419  0.00446927]]


[[0.01858202 0.01892761 0.12441184 0.        ]
 [0.11832416 0.         0.17417657 0.        ]
 [0.01624266 0.02062897 0.35218119 0.15652498]]
```

Output seems correct because observing two "1 wall" in a row as the last two observations probably means the agent is in column 3 when performing the final two actions (observing 1 wall in column 3 is 0.9), which seems to be why in the final belief state, the probabilities are higher in column 3 compared to other columns.

Appendix - Code for Q3

```
# -*- coding: utf-8 -*-
"""422 assignment1 q3

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1UsJdmZyoH1MRSCQbfGl3gxZldZFommtx
"""

import numpy as np
from enum import Enum

#constants
num_rows = 3
num_cols = 4
prob_going_in_correct_dir = 0.8
prob_moving_at_right_angles_of_intended_direction = 0.1
terminal_states = [(4,2),(4,3)]
empty_states = [(2,2)]
class Walls(Enum):
   ONE = 1
   TWO = 2
   END = 3

class Actions(Enum):
   UP = 1
   DOWN = 2
   LEFT = 3
   RIGHT = 4

#returns the element in grid that is a np array when given col and row coordinates given in
lecture
def getGridElement(grid,col,row):
  return grid[(len(grid) - row,col-1)]

def updateGridElement(grid,col,row,value):
  grid[(len(grid) - row,col-1)] = value

def isValid(state):
  col,row = state
  if col<=0 or col >num_cols or row<=0 or row>num_rows:
    return False
  if state in empty_states:
    return False
  return True

def get_left_right_up_down_neighbors(state):
  col,row = state
  return [(col-1,row),(col+1,row),(col,row+1),(col,row-1)]

def get_num_walls_around_state(state):
  num_walls = 0
```

```
    for neighbor in get_left_right_up_down_neighbors(state):
      if not isValid(neighbor):
        num_walls+=1
    return num_walls

  def get_prob_s_prime_given_a_s(s_prime, a, s):
    if (s in get_left_right_up_down_neighbors(s_prime) or s == s_prime) and isValid(s) and (s
  not in terminal_states):
      left_neighbor,right_neighbor,up_neighbor,down_neighbor =
  get_left_right_up_down_neighbors(s_prime)

      if a==Actions.UP:
        if s==left_neighbor or  s==right_neighbor:
          return prob_moving_at_right_angles_of_intended_direction
        elif s==down_neighbor:
          return prob_going_in_correct_dir
        elif s == s_prime:
          prob = 0
          if not isValid((s[0],s[1]+1)):
            prob+=prob_going_in_correct_dir
          if not isValid((s[0]-1,s[1])):
            prob+=prob_moving_at_right_angles_of_intended_direction
          if not isValid((s[0]+1,s[1])):
            prob+=prob_moving_at_right_angles_of_intended_direction
          return prob
        else:
          return 0

      if a==Actions.DOWN:
        if s==left_neighbor or  s==right_neighbor:
          return prob_moving_at_right_angles_of_intended_direction
        elif s==up_neighbor:
          return prob_going_in_correct_dir
        elif s == s_prime:
          prob = 0
          if not isValid((s[0],s[1]-1)):
            prob+=prob_going_in_correct_dir
          if not isValid((s[0]-1,s[1])):
            prob+=prob_moving_at_right_angles_of_intended_direction
          if not isValid((s[0]+1,s[1])):
            prob+=prob_moving_at_right_angles_of_intended_direction
          return prob
        else:
          return 0

      if a==Actions.LEFT:
        if s==up_neighbor or  s==down_neighbor:
          return prob_moving_at_right_angles_of_intended_direction
        elif s==right_neighbor:
          return prob_going_in_correct_dir
        elif s == s_prime:
          prob = 0
          if not isValid((s[0]-1,s[1])):
```

```
          prob+=prob_going_in_correct_dir
        if not isValid((s[0],s[1]+1)):
          prob+=prob_moving_at_right_angles_of_intended_direction
        if not isValid((s[0]+1,s[1]-1)):
          prob+=prob_moving_at_right_angles_of_intended_direction
        return prob
      else:
        return 0

    if a==Actions.RIGHT:
      if s==up_neighbor or  s==down_neighbor:
        return prob_moving_at_right_angles_of_intended_direction
      elif s==left_neighbor:
        return prob_going_in_correct_dir
      elif s == s_prime:
        prob = 0
        if not isValid((s[0]+1,s[1])):
          prob+=prob_going_in_correct_dir
        if not isValid((s[0],s[1]+1)):
          prob+=prob_moving_at_right_angles_of_intended_direction
        if not isValid((s[0]+1,s[1]-1)):
          prob+=prob_moving_at_right_angles_of_intended_direction
        return prob
      else:
        return 0
  else:
    return 0

def get_prob_e_given_s_prime(e,s_prime):
  col,row = s_prime
  #terminal
  if s_prime == (4,3) or s_prime == (4,2):
    if e == Walls.END:
      return 1
    else:
      return 0

  #non terminal in third column
  if col ==3:
    if e==Walls.ONE:
      return 0.9
    elif e==Walls.TWO:
      return 0.1
    elif e==Walls.END:
      return 0

  # all other columns
  if e==Walls.ONE:
    return 0.1
  elif e==Walls.TWO:
    return 0.9
  elif e==Walls.END:
    return 0
```

```

def updateSingleBeliefState(b_s,a,e):
  new_b_s = b_s.copy()

  for row in range(1, len(new_b_s)+1):
    for col in range(1, len(new_b_s[0])+1):
      if isValid((col,row)):
        sum_term = 0
        for neighbor in get_left_right_up_down_neighbors((col,row)) + [(col,row)]:
          if isValid(neighbor):
            sum_term += get_prob_s_prime_given_a_s((col,row), a, neighbor) *
getGridElement(b_s,neighbor[0],neighbor[1])

        updateGridElement(new_b_s,col,row, get_prob_e_given_s_prime(e,(col,row))*sum_term)

  #return new_b_s #temp for testing to delete
  return new_b_s/(sum(sum(new_b_s))).copy()

def updateBeliefState(b_s,a1_n,e1_n):
  for a,e in zip(a1_n,e1_n):
    print(b_s)
    print("\n")
    b_s = updateSingleBeliefState(b_s,a,e)
  return (b_s)

"""### Tests"""

def get_uniform_belief_state_on_non_terminal_states():
  b_s = np.zeros((3,4))
  for row in range(1, len(b_s)+1):
    for col in range(1, len(b_s[0])+1):
      if isValid((col,row)) and (col,row) not in terminal_states:
        updateGridElement(b_s,col,row, 1)

  b_s = b_s/(sum(sum(b_s)))
  return b_s

# (up, up , up) (2,2,2)
b_s = get_uniform_belief_state_on_non_terminal_states()
a1_n = (Actions.UP, Actions.UP, Actions.UP)
e1_n = (Walls.TWO,Walls.TWO,Walls.TWO)
res = updateBeliefState(b_s,a1_n,e1_n)
print(res)

#(up, up, up) (1,1,1)
b_s = get_uniform_belief_state_on_non_terminal_states()
a1_n = (Actions.UP, Actions.UP, Actions.UP)
e1_n = (Walls.ONE,Walls.ONE,Walls.ONE)
res = updateBeliefState(b_s,a1_n,e1_n)
print(res)

#(right, right, up) (1,1,end) with S0 = (2,3)
b_s = np.zeros((3,4))
```

```python
b_s[(0,1)] = 1
a1_n = (Actions.RIGHT, Actions.RIGHT, Actions.UP)
e1_n = (Walls.ONE,Walls.ONE,Walls.END)
res = updateBeliefState(b_s,a1_n,e1_n)
print(res)

#(up, right, right, right) (2,2,1,1) with S0 = (1,1)
b_s = np.zeros((3,4))
b_s[(2,0)] = 1
a1_n = (Actions.UP, Actions.RIGHT, Actions.RIGHT,Actions.RIGHT)
e1_n = (Walls.TWO,Walls.TWO,Walls.ONE,Walls.ONE)
res = updateBeliefState(b_s,a1_n,e1_n)
print(res)
```