AI388U-assignment4

Graded

Student

Jashan Shah

Total Points

44 / 41 pts

Autograder Score
44.0 / 41.0

Passed Tests

TestBayesianInference.test_sample_observations_with_update[initial_state0-observation_list0-uniform] (1/1)
TestBayesianInference.test_sample_observations_with_update[initial_state4-observation_list4-dirac] (1/1)
TestBayesianInference.test_sample_observations_with_update[initial_state2-observation_list2-uniform] (1/1)
TestBayesianInference.test_sample_observations_with_update[initial_state1-observation_list1-uniform] (1/1)
TestBayesianInference.test_sample_observations_with_update[initial_state5-observation_list5-dirac] (1/1)
TestBayesianInference.test_sample_observations_with_update[initial_state6-observation_list6-dirac] (1/1)
TestBayesianInference.test_sample_observations_with_update[initial_state3-observation_list3-uniform] (1/1)
TestBayesianInference.test_sample_predictions[initial_state0-action_list0-uniform] (1/1)
TestBayesianInference.test_sample_observations_with_update[initial_state7-observation_list7-dirac] (1/1)
TestBayesianInference.test_sample_observation[state0] (1/1)
TestBayesianInference.test_sample_observation[state1] (1/1)
TestBayesianInference.test_sample_observation[state2] (1/1)
TestBayesianInference.test_sample_predictions[initial_state1-action_list1-uniform] (1/1)
TestBayesianInference.test_sample_observation[state3] (1/1)
TestBayesianInference.test_sample_observation[state4] (1/1)
TestBayesianInference.test_sample_observation[state5] (1/1)
TestBayesianInference.test_sample_observation[state7] (1/1)
TestBayesianInference.test_sample_predictions[initial_state6-action_list6-dirac] (1/1)
TestBayesianInference.test_sample_predictions[initial_state3-action_list3-uniform] (1/1)
TestBayesianInference.test_sample_observation[state8] (1/1)
TestBayesianInference.test_sample_observation[state9] (1/1)
TestBayesianInference.test_sample_observation[state11] (1/1)
TestBayesianInference.test_sample_observation[state12] (1/1)
TestBayesianInference.test_sample_predictions[initial_state2-action_list2-uniform] (1/1)
TestBayesianInference.test_sample_observation[state13] (1/1)
TestBayesianInference.test_sample_predictions[initial_state4-action_list4-dirac] (1/1)
TestBayesianInference.test_sample_observation[state6] (1/1)
TestBayesianInference.test_sample_observation[state14] (1/1)
TestBayesianInference.test_sample_observation[state15] (1/1)
TestBayesianInference.test_sample_observation[state16] (1/1)
TestBayesianInference.test_bayesian_update[initial_state1-observation1-dirac] (1/1)
TestBayesianInference.test_bayesian_update[initial_state2-observation2-uniform] (1/1)
TestBayesianInference.test_sample_predictions[initial_state7-action_list7-dirac] (1/1)
TestBayesianInference.test_sample_observation[state10] (1/1)
TestBayesianInference.test_sample_predictions[initial_state5-action_list5-dirac] (1/1)
TestBayesianInference.test_predict_update[initial_state0-action_list0-uniform] (1/1)
TestBayesianInference.test_bayesian_update[initial_state0-observation0-uniform] (1/1)
TestBayesianInference.test_predict_update[initial_state4-action_list4-dirac] (1/1)

TestBayesianInference.test_predict_update[initial_state1-action_list1-uniform] (1/1)
TestBayesianInference.test_predict_update[initial_state2-action_list2-uniform] (1/1)
TestBayesianInference.test_predict_update[initial_state6-action_list6-dirac] (1/1)
TestBayesianInference.test_predict_update[initial_state5-action_list5-dirac] (1/1)
TestBayesianInference.test_predict_update[initial_state3-action_list3-uniform] (1/1)
TestBayesianInference.test_predict_update[initial_state7-action_list7-dirac] (1/1)

Autograder Results

Autograder Output

```
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.3.3, pluggy-1.5.0
rootdir: /autograder/submission
plugins: timeout-2.3.1, xdist-3.6.1, utils-0.0.0
created: 4/4 workers
4 workers [44 items]

.............................................                            [100%]
============================== 44 passed in 0.64s ==============================
{
  "tests": [
    {
      "score": 1,
      "max_score": 1,
      "name": "TestBayesianInference.test_sample_observations_with_update[initial_state0-observation_list0-ur
      "output": "",
      "visibility": "visible",
      "status": "passed"
    },
    {
      "score": 1,
      "max_score": 1,
      "name": "TestBayesianInference.test_sample_observations_with_update[initial_state4-observation_list4-di
      "output": "",
      "visibility": "visible",
      "status": "passed"
    },
    {
      "score": 1,
      "max_score": 1,
      "name": "TestBayesianInference.test_sample_observations_with_update[initial_state2-observation_list2-ur
      "output": "",
      "visibility": "visible",
      "status": "passed"
    },
    {
      "score": 1,
      "max_score": 1,
      "name": "TestBayesianInference.test_sample_observations_with_update[initial_state1-observation_list1-ur
      "output": "",
      "visibility": "visible",
      "status": "passed"
    },
    {
      "score": 1,
      "max_score": 1,
      "name": "TestBayesianInference.test_sample_observations_with_update[initial_state5-observation_list5-di
      "output": "",
      "visibility": "visible",
      "status": "passed"
    },
    {
```

```
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observations_with_update[initial_state6-observation_list6-di
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observations_with_update[initial_state3-observation_list3-ur
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_predictions[initial_state0-action_list0-uniform]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observations_with_update[initial_state7-observation_list7-di
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state0]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state1]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state2]",
  "output": "",
```

```
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_predictions[initial_state1-action_list1-uniform]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state3]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state4]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state5]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state7]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_predictions[initial_state6-action_list6-dirac]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
```

```
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_predictions[initial_state3-action_list3-uniform]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state8]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state9]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state11]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state12]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_predictions[initial_state2-action_list2-uniform]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state13]",
  "output": "",
```

```
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_predictions[initial_state4-action_list4-dirac]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state6]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state14]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state15]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state16]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_bayesian_update[initial_state1-observation1-dirac]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
```

```
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_bayesian_update[initial_state2-observation2-uniform]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_predictions[initial_state7-action_list7-dirac]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_observation[state10]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_sample_predictions[initial_state5-action_list5-dirac]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_predict_update[initial_state0-action_list0-uniform]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_bayesian_update[initial_state0-observation0-uniform]",
  "output": "",
  "visibility": "visible",
  "status": "passed"
 },
 {
  "score": 1,
  "max_score": 1,
  "name": "TestBayesianInference.test_predict_update[initial_state4-action_list4-dirac]",
  "output": "",
```

```
    "visibility": "visible",
    "status": "passed"
  },
  {
    "score": 1,
    "max_score": 1,
    "name": "TestBayesianInference.test_predict_update[initial_state1-action_list1-uniform]",
    "output": "",
    "visibility": "visible",
    "status": "passed"
  },
  {
    "score": 1,
    "max_score": 1,
    "name": "TestBayesianInference.test_predict_update[initial_state2-action_list2-uniform]",
    "output": "",
    "visibility": "visible",
    "status": "passed"
  },
  {
    "score": 1,
    "max_score": 1,
    "name": "TestBayesianInference.test_predict_update[initial_state6-action_list6-dirac]",
    "output": "",
    "visibility": "visible",
    "status": "passed"
  },
  {
    "score": 1,
    "max_score": 1,
    "name": "TestBayesianInference.test_predict_update[initial_state5-action_list5-dirac]",
    "output": "",
    "visibility": "visible",
    "status": "passed"
  },
  {
    "score": 1,
    "max_score": 1,
    "name": "TestBayesianInference.test_predict_update[initial_state3-action_list3-uniform]",
    "output": "",
    "visibility": "visible",
    "status": "passed"
  },
  {
    "score": 1,
    "max_score": 1,
    "name": "TestBayesianInference.test_predict_update[initial_state7-action_list7-dirac]",
    "output": "",
    "visibility": "visible",
    "status": "passed"
  }
],
```

```
  "score": 44,
  "visibility": "visible",
  "stdout_visibility": "after_due_date"
}
```


TestBayesianInference.test_sample_observations_with_update[initial_state0-observation_list0-uniform] (1/1)

TestBayesianInference.test_sample_observations_with_update[initial_state4-observation_list4-dirac] (1/1)

TestBayesianInference.test_sample_observations_with_update[initial_state2-observation_list2-uniform] (1/1)

TestBayesianInference.test_sample_observations_with_update[initial_state1-observation_list1-uniform] (1/1)

TestBayesianInference.test_sample_observations_with_update[initial_state5-observation_list5-dirac] (1/1)

TestBayesianInference.test_sample_observations_with_update[initial_state6-observation_list6-dirac] (1/1)

TestBayesianInference.test_sample_observations_with_update[initial_state3-observation_list3-uniform] (1/1)

TestBayesianInference.test_sample_predictions[initial_state0-action_list0-uniform] (1/1)

TestBayesianInference.test_sample_observations_with_update[initial_state7-observation_list7-dirac] (1/1)

TestBayesianInference.test_sample_observation[state0] (1/1)

TestBayesianInference.test_sample_observation[state1] (1/1)

TestBayesianInference.test_sample_observation[state2] (1/1)

TestBayesianInference.test_sample_predictions[initial_state1-action_list1-uniform] (1/1)

TestBayesianInference.test_sample_observation[state3] (1/1)

TestBayesianInference.test_sample_observation[state4] (1/1)

TestBayesianInference.test_sample_observation[state5] (1/1)

TestBayesianInference.test_sample_observation[state7] (1/1)

TestBayesianInference.test_sample_predictions[initial_state6-action_list6-dirac] (1/1)

TestBayesianInference.test_sample_predictions[initial_state3-action_list3-uniform] (1/1)

TestBayesianInference.test_sample_observation[state8] (1/1)

TestBayesianInference.test_sample_observation[state9] (1/1)

TestBayesianInference.test_sample_observation[state11] (1/1)

TestBayesianInference.test_sample_observation[state12] (1/1)

TestBayesianInference.test_sample_predictions[initial_state2-action_list2-uniform] (1/1)

TestBayesianInference.test_sample_observation[state13] (1/1)

TestBayesianInference.test_sample_predictions[initial_state4-action_list4-dirac] (1/1)

TestBayesianInference.test_sample_observation[state6] (1/1)

TestBayesianInference.test_sample_observation[state14] (1/1)

TestBayesianInference.test_sample_observation[state15] (1/1)

TestBayesianInference.test_sample_observation[state16] (1/1)

TestBayesianInference.test_bayesian_update[initial_state1-observation1-dirac] (1/1)

TestBayesianInference.test_bayesian_update[initial_state2-observation2-uniform] (1/1)

TestBayesianInference.test_sample_predictions[initial_state7-action_list7-dirac] (1/1)

TestBayesianInference.test_sample_observation[state10] (1/1)

TestBayesianInference.test_sample_predictions[initial_state5-action_list5-dirac] (1/1)

TestBayesianInference.test_predict_update[initial_state0-action_list0-uniform] (1/1)

TestBayesianInference.test_bayesian_update[initial_state0-observation0-uniform] (1/1)

TestBayesianInference.test_predict_update[initial_state4-action_list4-dirac] (1/1)

TestBayesianInference.test_predict_update[initial_state1-action_list1-uniform] (1/1)

TestBayesianInference.test_predict_update[initial_state2-action_list2-uniform] (1/1)

TestBayesianInference.test_predict_update[initial_state6-action_list6-dirac] (1/1)

TestBayesianInference.test_predict_update[initial_state5-action_list5-dirac] (1/1)

TestBayesianInference.test_predict_update[initial_state3-action_list3-uniform] (1/1)

TestBayesianInference.test_predict_update[initial_state7-action_list7-dirac] (1/1)

Submitted Files

bayesian.py Download

```python
import numpy as np

class StateGenerator:

    def __init__(self, nrows=8, ncols=7, npieces=10):
        """
        Initialize a generator for sampling valid states from
        an npieces dimensional state space.
        """
        self.nrows = nrows
        self.ncols = ncols
        self.npieces = npieces
        self.rng = np.random.default_rng()

    def sample_state(self):
        """
        Samples a self.npieces length tuple.

        Output:
            Returns a state. A state is as 2-tuple (positions, dimensions), where
            - Positions is represented as a list of position (c,r) tuples
            - Dimensions is a 2-tuple (self.nrows, self.ncols)

            For example, if the dimensions of the board are 2 rows, 3 columns, and the number of pieces
            is 4, then a valid return state would be ([(0, 0) , (1, 0), (2, 0), (1, 1)], (2,3))
        """
        ## Returns positions in decoded format. i.e. list of (c,r) i.e. (x,y)
        ## Without loss of generalization, we assume that positions[1:] are fixes; only
        ## positions[0] will be moved
        positions = self.rng.choice(self.nrows * self.ncols, size=self.npieces, replace=False)
        pos = list(self.decode(p) for p in positions)
        return pos, (self.nrows, self.ncols)

    def decode(self, position):
        r = position // self.ncols
        c = position - self.ncols * r
        return (c, r)


def sample_observation(state):
    """
        Given a state, sample an observation from it. Specifically, the positions[1:] locations are
        all known, while positions[0] should have a noisy observation applied.

        Input:
            State: a 2-tuple of (positions, dimensions), the same as defined in StateGenerator.sample_state

        Returns:
            A tuple (position, distribution) where:
```

```
50          - Position is a sampled position which is a 2-tuple (c, r), which represents the sampled observation
51          - Distribution is a 2D numpy array representing the observation distribution
52
53      NOTE: the array representing the distribution should have a shape of (nrows, ncols)
54      """
55      # # We need to return the first element in pos which is the first part of the state
56      # pos, dimensions = state
57      #
58      # # First element in pos
59      # movable_piece = pos[0]
60      #
61      # # We now need to get the distribution, this will be a nrows x ncols grid.
62      # # Need to also check if there were any pieces above, below, and to the right and left of that piece
63      # probability = .6
64      # count = 0
65      # above = right = below = left = -1
66      #
67      # # Create a numpy array to return
68      # distribution = np.zeros(dimensions)
69      #
70      # if movable_piece[1] < dimensions[0]:
71      #     # Checking for above
72      #     for i in range(1, len(pos)):
73      #         # Row value needs to be greater by 1 and column needs to be the same
74      #         if movable_piece[1] + 1 == pos[i][1] and movable_piece[0] == pos[i][0]:
75      #             count = count + 1
76      #             # Get the pos[i] column and row value then update that column and row value in numpy array to 10
77      #             above = .1
78      #             distribution[pos[i][1]][pos[i][0]] = above
79      #             # In case more than one piece is at the same spot (should not happen)
80      #             break
81      #
82      # if movable_piece[0] < dimensions[1]:
83      #     # Checking for right
84      #     for i in range(1, len(pos)):
85      #         # Row value needs to be the same, column needs to be larger for the pos by 1
86      #         if movable_piece[1] == pos[i][1] and movable_piece[0] + 1 == pos[i][0]:
87      #             count = count + 1
88      #             right = .1
89      #             distribution[pos[i][1]][pos[i][0]] = right
90      #             # In case more than one piece is at the same spot (should not happen)
91      #             break
92      #
93      # if movable_piece[1] > 0:
94      #     # Checking for the bottom
95      #     for i in range(1, len(pos)):
96      #         if movable_piece[1] - 1 == pos[i][1] and movable_piece[0] == pos[i][0]:
97      #             count = count + 1
98      #             bottom = .1
99      #             distribution[pos[i][1]][pos[i][0]] = bottom
```

```
#             # In case more than one piece is at the same spot (should not happen)
#             break
#
# if movable_piece[0] > 0:
#     # Checking for the left
#     for i in range(1, len(pos)):
#         if movable_piece[0] - 1 == pos[i][0] and movable_piece[1] == pos[i][0]:
#             count = count + 1
#             left = .1
#             distribution[pos[i][1]][pos[i][0]] = left
#             # In case more than one piece is at the same spot (should not happen)
#             break
#
# # Calculate total probability piece is where we believe it is
# probability += count * 0.1
#
# # Adding it to our distribution
# distribution[movable_piece[1]][movable_piece[0]] = probability
#
# return movable_piece, distribution


positions, dimensions = state
movable_piece = positions[0]  # The piece to track
nrows, ncols = dimensions
distribution = np.zeros(dimensions)
adjacent_prob = 0.1
prob = 0.6
# Define movement directions (delta for column, delta for row)
directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
c, r = movable_piece
distribution[r, c] += prob

# Iterate over all directions to check neighbors
for delta_col, delta_row in directions:
    neighbor_col = c + delta_col
    neighbor_row = r + delta_row
    # Check if the neighbor is within bounds and not occupied
    if (neighbor_col >= 0 and neighbor_col < ncols and neighbor_row >= 0 and neighbor_row < nrows
            and (neighbor_col, neighbor_row) not in positions[1:]):
        distribution[neighbor_row, neighbor_col] = adjacent_prob
    else:
        distribution[r, c] += adjacent_prob

# Normalization step
distribution /= np.sum(distribution)
flattened_distribution = distribution.flatten()
sampled_index = np.random.choice(ncols * nrows, p=flattened_distribution)
sampled_column = sampled_index % ncols
sampled_row = sampled_index // ncols

return (sampled_row, sampled_column), distribution
```

```
152
153 def sample_transition(state, action):
154     """
155     Given a state and an action,
156     returns:
157         a resulting state, and a probability distribution represented by a 2D numpy array
158     If a transition is invalid, returns None for the state, and a zero probability distribution
159     NOTE: the array representing the distribution should have a shape of (nrows, ncols)
160
161     Inputs:
162         State: a 2-tuple of (positions, dimensions), the same as defined in StateGenerator.sample_state
163         Action: a 2-tuple (dc, dr) representing the difference in positions of position[0] as a result of
164                 executing this transition.
165
166     Outputs:
167         A 2-tuple (new_position, transition_probabilities), where
168             - new_position is:
169                 A 2-tuple (new_column, new_row) if the action is valid.
170                 None if the action is invalid.
171             - transition_probabilities is a 2D numpy array with shape (nrows, ncols) that accurately reflects
172                 the probability of ending up at a certain position on the board given the action.
173     """
174     # Need to check if the addition of the action takes it outside of the board
175     pos, dimensions = state
176     movable_piece = pos[0]
177     new_location = (movable_piece[0] + action[0], movable_piece[1] + action[1])
178
179     # Initialize transition_probabilities
180     transition_probabilities = np.zeros(dimensions)
181     # Need a converted new_location value for row-major order
182     Row_major_new_location = (dimensions[0] - new_location[1], new_location[0])
183     # Now to check if a valid location
184     if (Row_major_new_location[0] > dimensions[1] or Row_major_new_location[0] < 0
185             or Row_major_new_location[1] > dimensions[0] or Row_major_new_location[1] < 0)\
186             or new_location in pos[1:]:
187         return (None, transition_probabilities) # All probabilities should be 0 since an invalid action
188     else:
189         transition_probabilities[Row_major_new_location[0]][Row_major_new_location[1]] = 100
190         return (new_location, transition_probabilities)
191
192
193 def initialize_belief(initial_state, style="uniform"):
194     """
195     Create an initial belief, based on the type of belief we want to start with
196
197     Inputs:
198         Initial_state: a 2-tuple of (positions, dimensions), the same as defined in
    StateGenerator.sample_state
199         style: an element of the set {"uniform", "dirac"}
200
201     Returns:
202         an initial belief, represented by a 2D numpy array with shape (nrows, ncols)
```

```
203
204     NOTE:
205         The array representing the distribution should have a shape of (nrows, ncols).
206         The occupied spaces (if any) should be zeroed out in the belief.
207         We define two types of priors: a uniform prior (equal probability over all
208         unoccupied spaces), and a dirac prior (which concentrates all the probability
209         onto the actual position on the piece).
210
211     """
212     pos, dimensions = initial_state
213     initial_belief = np.zeros(dimensions)
214     nrows, ncols = dimensions
215     if style == "uniform":
216
217         occupied_spaces_row_major = []
218         for i in range(1, len(pos)):
219             occupied_spaces_row_major.append((pos[i][1], pos[i][0]))
220
221         total_locations = dimensions[0] * dimensions[1]
222
223         # We are getting the total amount of spots subtracted by amount of occupied spots
224         unoccupied_spaces = total_locations - (len(pos) -1) # We can consider pos[0] to be unoccupied
225         probability_uniform = 1 / unoccupied_spaces
226
227         # Nested for loop
228         for row in range(0, nrows):
229             for col in range(0, ncols):
230                 if (row, col) not in occupied_spaces_row_major:
231                     initial_belief[row, col] = probability_uniform
232
233     if style == "dirac":
234         actual_pos = pos[0]
235         row_major_actual_pos = (actual_pos[1], actual_pos[0])
236         initial_belief[row_major_actual_pos[0]][row_major_actual_pos[1]] = 1
237
238     return initial_belief
239
240
241 def belief_update(prior, observation, reference_state):
242     """
243     Given a prior an observation, compute the posterior belief
244
245     Inputs:
246         prior: a 2D numpy array with shape (nrows, ncols)
247         observation: a 2-tuple (col, row) representing the observation of a piece at a position
248         reference_state: a 2-tuple of (positions, dimensions), the same as defined in
    StateGenerator.sample_state
249
250     Returns:
251         posterior: a 2D numpy array with shape (nrows, ncols)
252     """
253     # pos, dimensions = reference_state
```

```
    # nrow, ncol = dimensions
    # c_observation, r_observation = observation
    # _, observation_dist = sample_observation((pos, dimensions))
    #
    # pos[0] = (c_observation, r_observation)
    # posterior = np.zeros(dimensions)
    #
    # posterior = observation_dist * prior
    # # Normalization of the posterior distribution
    # posterior /= np.sum(posterior)
    #
    # return posterior
    pos, dimensions = reference_state
    nrow, ncol = dimensions
    c_observation, r_observation = observation
    temp_state = ([(c_observation, r_observation)] + pos[1:], dimensions)
    _, observation_dist = sample_observation(temp_state)

    # pos[0] = (c_observation, r_observation)

    posterior = observation_dist * prior

    for p in pos[1:]:
        row, col = p[1], p[0]
        posterior[row, col] = 0

    # Normalization.
    total = np.sum(posterior)
    if total > 0:
        posterior /= total

    return posterior

def belief_predict(prior, action, reference_state):
    """
    Given a prior, and an action, compute the posterior belief.

    Actions will be given in terms of dc, dr

   Inputs:
        prior: a 2D numpy array with shape (nrows, ncols)
        action: a 2-tuple (dc, dr) as defined for action in sample_transition
        reference_state: a 2-tuple of (positions, dimensions), the same as defined in
StateGenerator.sample_state

    Returns:
        posterior: a 2D numpy array with shape (nrows, ncols)
    """
    pos, dimensions = reference_state
    nrow, ncol = dimensions
    dc, dr = action
    posterior = np.zeros(dimensions)
```

```
305
306     for row in range(nrow):
307         for col in range(ncol):
308             new_col, new_row = col + dc, row + dr
309             if (new_col >= 0 and new_col < ncol and new_row >= 0 and new_row < nrow):
310                 posterior[new_row, new_col] += prior[row, col]
311                 if (new_col, new_row) in pos[1:]:
312                     posterior[new_row, new_col] = 0
313     posterior /= np.sum(posterior)
314
315     return posterior
316
317 if __name__ == "__main__":
318     gen = StateGenerator()
319     initial_state = gen.sample_state()
320     obs, dist = sample_observation(initial_state)
321     print(initial_state)
322     print(obs)
323     print(dist)
324     b = initialize_belief(initial_state, style="uniform")
325     print(b)
326     b = belief_update(b, obs, initial_state)
327     print(b)
328     b = belief_predict(b, (1, 0), initial_state)
329     print(b)
330
```