Final Project

0. Libary & setting

In [4]:
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import gym
import matplotlib.pyplot as plt
import pandas as pd
from itertools import combinations


# multiple output in notebook without print()
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
```

Define CartePoleDynamics including environment, parameter, state space setting

In [5]:
```
class CartPoleDynamics:
    def __init__(self):
        # System parameters
        self.M = 1.0  # Mass of the cart
        self.m = 0.1  # Mass of the pole
        self.g = -9.8  # Gravity
        self.l = 0.5  # Length of the pole
        self.mu_c = 0.0005  # Friction for the cart
        self.mu_p = 0.000002  # Friction for the pole
        self.delta_t = 0.02  # Time step
        self.actions = [-10, 10]  # Available actions (forces in Newtons)

        # State space discretization
        self.theta_boxes = np.array([-12, -6, -1, 0, 1, 6, 12]) * np.pi / 180  # radians
        self.x_boxes = np.array([-2.4, -0.8, 0.8, 2.4])  # meters
        self.theta_dot_boxes = np.array([-50, 0, 50]) * np.pi / 180  # radians/s
        self.x_dot_boxes = np.array([-0.5, 0, 0.5])  # m/s
```

```
        # Define state space size
        self.state_space_size = (
            len(self.theta_boxes) + 1,
            len(self.theta_dot_boxes) + 1,
            len(self.x_boxes) + 1,
            len(self.x_dot_boxes) + 1
        )

    def compute_accelerations(self, theta, theta_dot, x_dot, F):
        """Compute angular and linear accelerations based on the model."""
        sin_theta = np.sin(theta)
        cos_theta = np.cos(theta)

        # Calculate angular acceleration (theta_ddot)
        numerator = (self.g * sin_theta +
                     cos_theta * ((-F - self.m * self.l * theta_dot**2 * sin_theta +
                                  self.mu_c * np.sign(x_dot)) / (self.M + self.m)) -
                     self.mu_p * theta_dot / (self.m * self.l))
        denominator = self.l * (4.0/3.0 - (self.m * cos_theta**2) / (self.M + self.m))
        theta_ddot = numerator / denominator

        # Calculate linear acceleration (x_ddot)
        x_ddot = (F + self.m * self.l * (theta_dot**2 * sin_theta - theta_ddot * cos_theta) -
                  self.mu_c * np.sign(x_dot)) / (self.M + self.m)

        return theta_ddot, x_ddot

    def update_state(self, state, action):
        """Update the state using Euler integration"""
        theta, theta_dot, x, x_dot = state
        F = action

        theta_ddot, x_ddot = self.compute_accelerations(theta, theta_dot, x_dot, F)

        # Update using Euler integration
        x_dot += self.delta_t * x_ddot
        x += self.delta_t * x_dot
        theta_dot += self.delta_t * theta_ddot
        theta += self.delta_t * theta_dot

        return np.array([theta, theta_dot, x, x_dot])
```

```python
    def discretize_state(self, state):
        """Discretize a continuous state based on provided thresholds"""
        theta, theta_dot, x, x_dot = state

        theta_idx = np.digitize(theta, self.theta_boxes, right=True)
        theta_dot_idx = np.digitize(theta_dot, self.theta_dot_boxes, right=True)
        x_idx = np.digitize(x, self.x_boxes, right=True)
        x_dot_idx = np.digitize(x_dot, self.x_dot_boxes, right=True)

        return (theta_idx, theta_dot_idx, x_idx, x_dot_idx)

    def is_state_valid(self, state):
        """Check if the state is within valid bounds"""
        theta, _, x, _ = state
        return (abs(theta) <= 12 * np.pi / 180 and abs(x) <= 2.4)
```

Define Dynamic Programming class for Implementing Algorithm

In [6]:
```python
class DynamicProgramming:
    def __init__(self):
        self.dynamics = CartPoleDynamics()

    """ Policy Iteration Algorithm"""
    def policy_iteration(self, gamma=0.99, threshold=1e-5):
        """Policy Iteration Algorithm with Policy Evaluation and Policy Improvement"""
        # Initialize policy and value function
        policy = np.random.choice([0, 1], size=self.dynamics.state_space_size)
        value_function = np.zeros(self.dynamics.state_space_size)

        while True:
            # Policy Evaluation
            while True:
                delta = 0
                for theta_idx in range(self.dynamics.state_space_size[0]):
                    for theta_dot_idx in range(self.dynamics.state_space_size[1]):
                        for x_idx in range(self.dynamics.state_space_size[2]):
                            for x_dot_idx in range(self.dynamics.state_space_size[3]):
                                state = (theta_idx, theta_dot_idx, x_idx, x_dot_idx)
                                action = self.dynamics.actions[policy[state]]
                                value = value_function[state]
                                new_value = self._compute_state_value(state, action, value_function, gamma)
```

```
                        value_function[state] = new_value
                        delta = max(delta, abs(value - new_value))

            if delta < threshold:
                break

        # Policy Improvement
        policy_stable = True
        for theta_idx in range(self.dynamics.state_space_size[0]):
            for theta_dot_idx in range(self.dynamics.state_space_size[1]):
                for x_idx in range(self.dynamics.state_space_size[2]):
                    for x_dot_idx in range(self.dynamics.state_space_size[3]):
                        state = (theta_idx, theta_dot_idx, x_idx, x_dot_idx)
                        old_action = policy[state]

                        # Find best action
                        action_values = []
                        for action_idx, action in enumerate(self.dynamics.actions):
                            value = self._compute_state_value(state, action, value_function, gamma)
                            action_values.append(value)

                        best_action = np.argmax(action_values)
                        policy[state] = best_action

                        if old_action != best_action:
                            policy_stable = False

        if policy_stable:
            break

    return policy, value_function

### Value Iteration Algorithm
def value_iteration(self, gamma=0.99, threshold=1e-5):
    """Value Iteration Algorithm"""
    value_function = np.zeros(self.dynamics.state_space_size)
    policy = np.zeros(self.dynamics.state_space_size, dtype=int)

    while True:
        delta = 0
        for theta_idx in range(self.dynamics.state_space_size[0]):
            for theta_dot_idx in range(self.dynamics.state_space_size[1]):
```

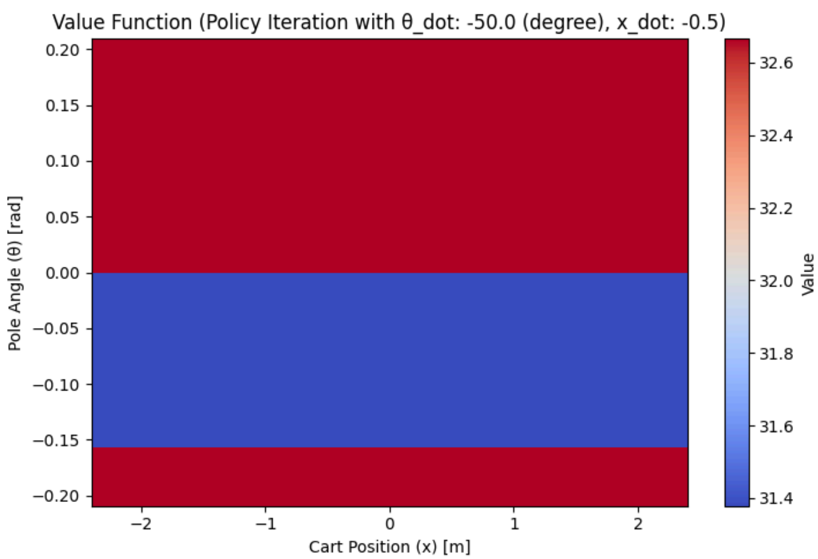
```
                for x_idx in range(self.dynamics.state_space_size[2]):
                    for x_dot_idx in range(self.dynamics.state_space_size[3]):
                        state = (theta_idx, theta_dot_idx, x_idx, x_dot_idx)
                        value = value_function[state]

                        # Find maximum value over all actions
                        action_values = []
                        for action in self.dynamics.actions:
                            new_value = self._compute_state_value(state, action, value_function, gamma)
                            action_values.append(new_value)

                        value_function[state] = max(action_values)
                        policy[state] = np.argmax(action_values)
                        delta = max(delta, abs(value - value_function[state]))

            if delta < threshold:
                break

        return policy, value_function

    def _compute_state_value(self, state, action, value_function, gamma):
        """Helper method to compute value for a state-action pair"""
        theta_idx, theta_dot_idx, x_idx, x_dot_idx = state

        # Convert discrete state to continuous
        continuous_state = [
            self.dynamics.theta_boxes[theta_idx - 1] if theta_idx > 0 else -np.inf,
            self.dynamics.theta_dot_boxes[theta_dot_idx - 1] if theta_dot_idx > 0 else -np.inf,
            self.dynamics.x_boxes[x_idx - 1] if x_idx > 0 else -np.inf,
            self.dynamics.x_dot_boxes[x_dot_idx - 1] if x_dot_idx > 0 else -np.inf
        ]

        # Get next state
        next_state_continuous = self.dynamics.update_state(continuous_state, action)
        next_state = self.dynamics.discretize_state(next_state_continuous)

        # Compute reward
        reward = 1 if self.dynamics.is_state_valid(next_state_continuous) else 0

        # Compute value
        return reward + gamma * value_function[next_state]
```

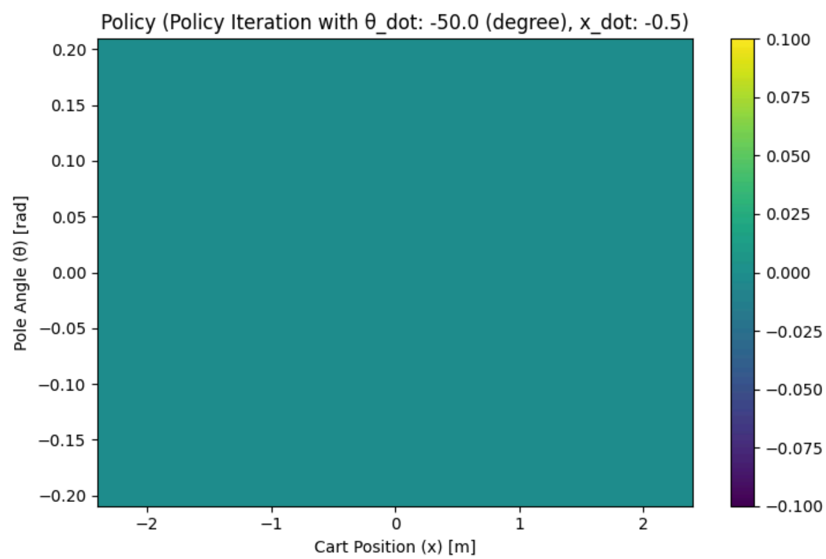
```
    def plot_results(self, policy, value_function, method_name, fixed_theta_dot_idx, fixed_x_dot_idx):
        """Plot value function and policy for a given slice of state space"""
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

        # Plot value function
        value_slice = value_function[:, fixed_theta_dot_idx, :, fixed_x_dot_idx]
        im1 = ax1.imshow(value_slice, extent=[self.dynamics.x_boxes[0],
                                              self.dynamics.x_boxes[-1],
                                              self.dynamics.theta_boxes[0],
                                              self.dynamics.theta_boxes[-1]],
                        aspect='auto', origin='lower', cmap='coolwarm')
        plt.colorbar(im1, ax=ax1, label='Value')
        ax1.set_title(f'Value Function ({method_name})')
        ax1.set_xlabel('Cart Position (x) [m]')
        ax1.set_ylabel('Pole Angle (θ) [rad]')

        # Plot policy
        policy_slice = policy[:, fixed_theta_dot_idx, :, fixed_x_dot_idx]
        im2 = ax2.imshow(policy_slice, extent=[self.dynamics.x_boxes[0],
                                               self.dynamics.x_boxes[-1],
                                               self.dynamics.theta_boxes[0],
                                               self.dynamics.theta_boxes[-1]],
                        aspect='auto', origin='lower', cmap='viridis')
        plt.colorbar(im2, ax=ax2, label='Action Index')
        ax2.set_title(f'Policy ({method_name})')
        ax2.set_xlabel('Cart Position (x) [m]')
        ax2.set_ylabel('Pole Angle (θ) [rad]')

        plt.tight_layout()
        plt.show()

def main():
    # Create instance of DP solver
    dp = DynamicProgramming()

    # Run policy iteration
    pi_policy, pi_value = dp.policy_iteration()

    # Run value iteration
    vi_policy, vi_value = dp.value_iteration()

    # Plot results for both methods
```

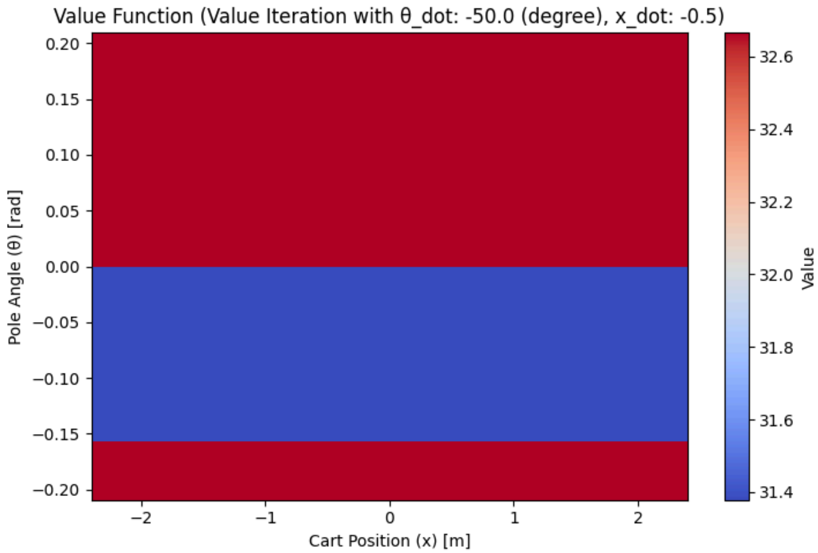
```
    fixed_states = [(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]  # Different combinations
    for theta_dot_idx, x_dot_idx in fixed_states:
        # Plot policy iteration results
        dp.plot_results(pi_policy, pi_value, f"Policy Iteration with θ_dot: {dp.dynamics.theta_dot_boxes[theta_dot_id

        dp.dynamics.theta_dot_boxes

        # Plot value iteration results
        dp.plot_results(vi_policy, vi_value, f"Value Iteration with θ_dot: {dp.dynamics.theta_dot_boxes[theta_dot_idx

    # Printing value functions
    print(f'Value functions from Policy Iteration: \n \
          {pi_value} \n' )

    print(f'Value functions from Value Iteration: \n \
          {vi_value}')
```

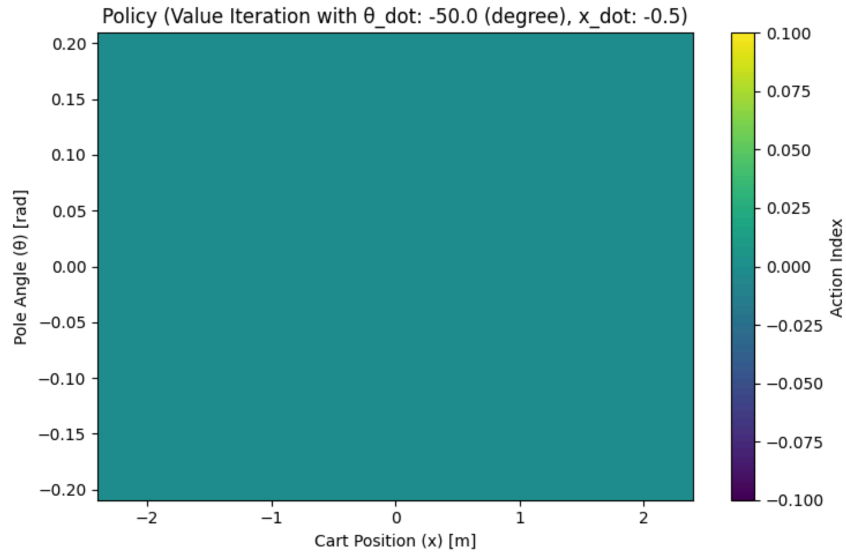
Implementing Algorithm and plotting

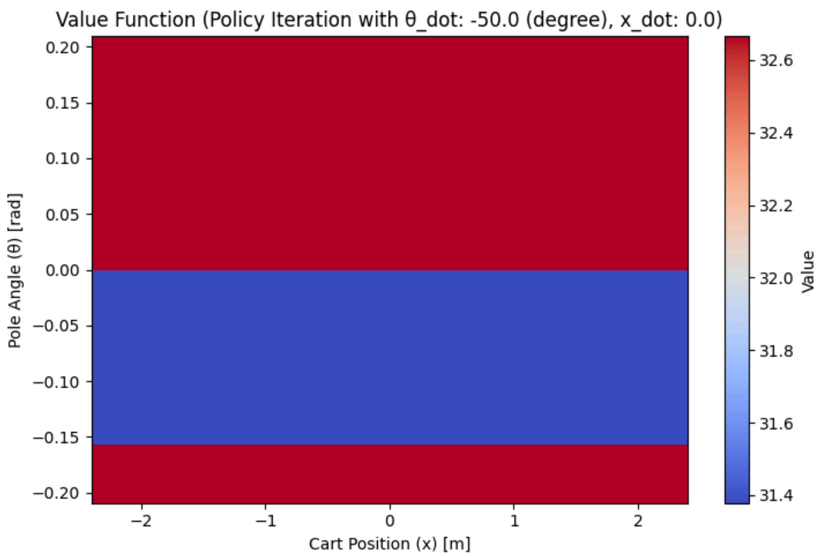
In [7]:
```
### Algorithm Implementation and Plotting
if __name__ == "__main__":
    main()
```

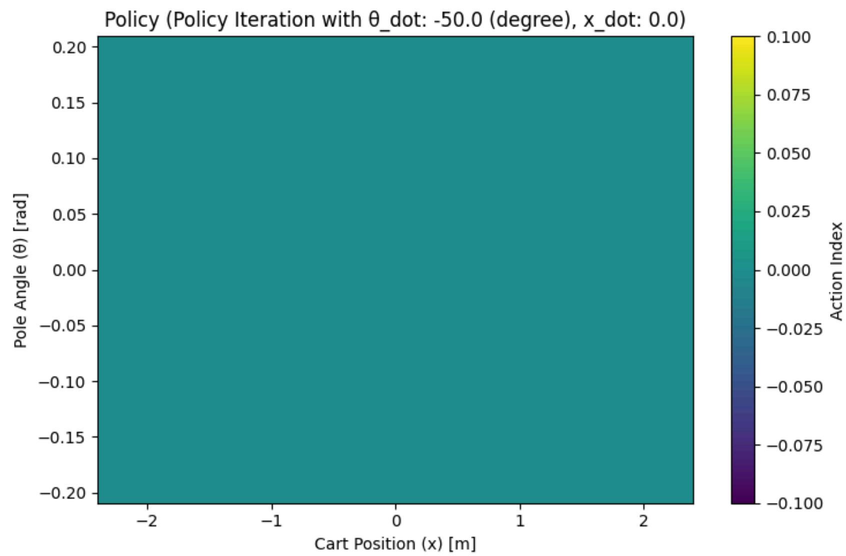


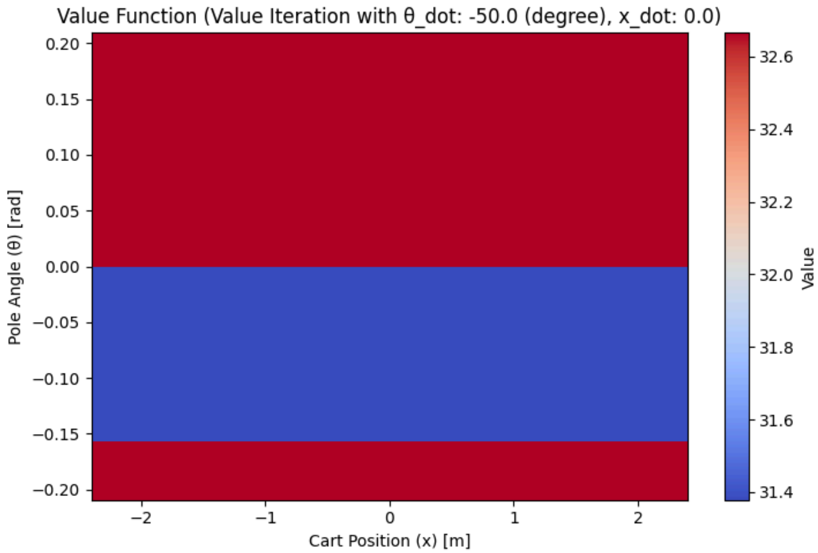


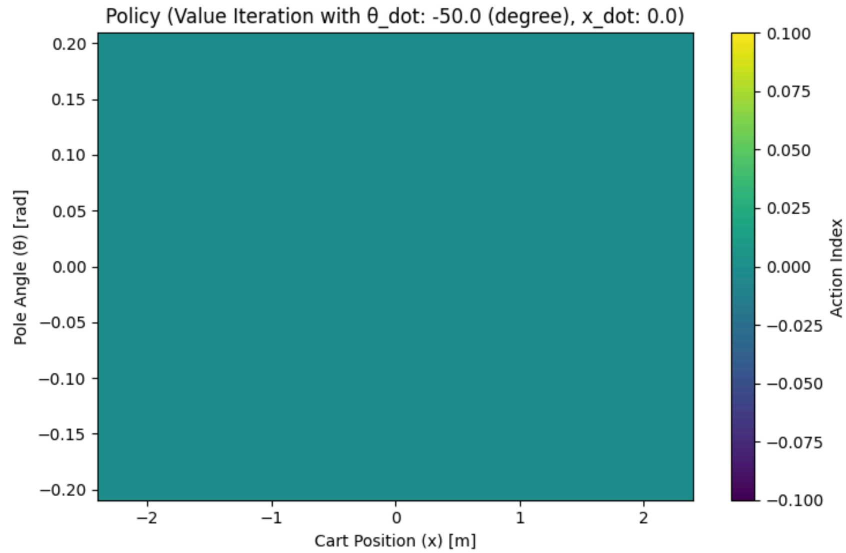

Value Function (Value Iteration with θ_dot: -50.0 (degree), x_dot: -0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

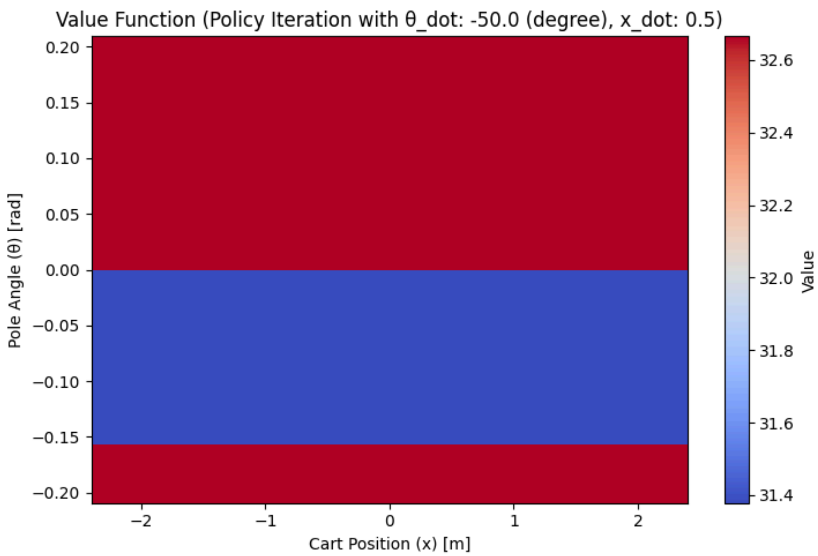

Policy (Value Iteration with θ_dot: -50.0 (degree), x_dot: -0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

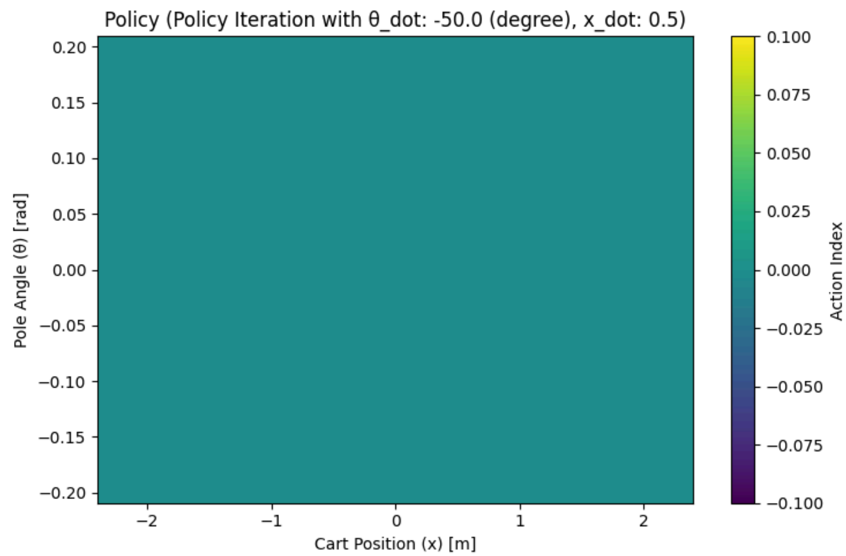

Value Function (Policy Iteration with θ_dot: -50.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

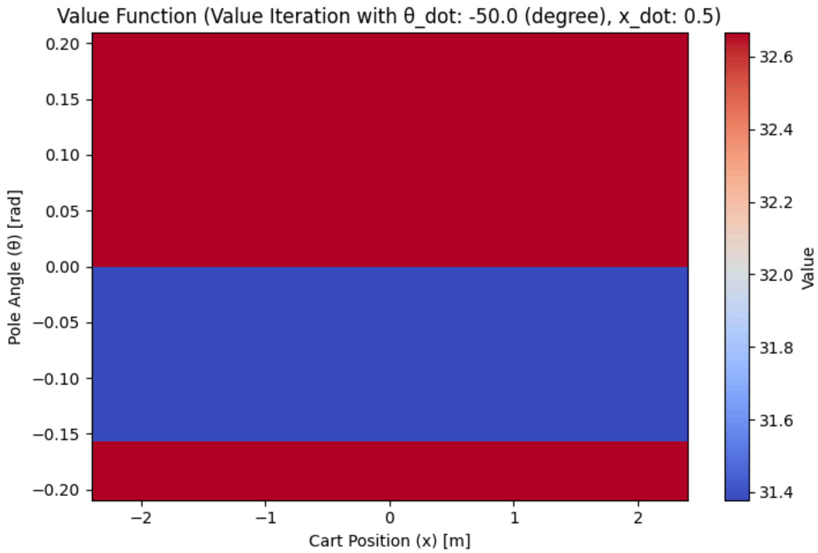

Policy (Policy Iteration with θ_dot: -50.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

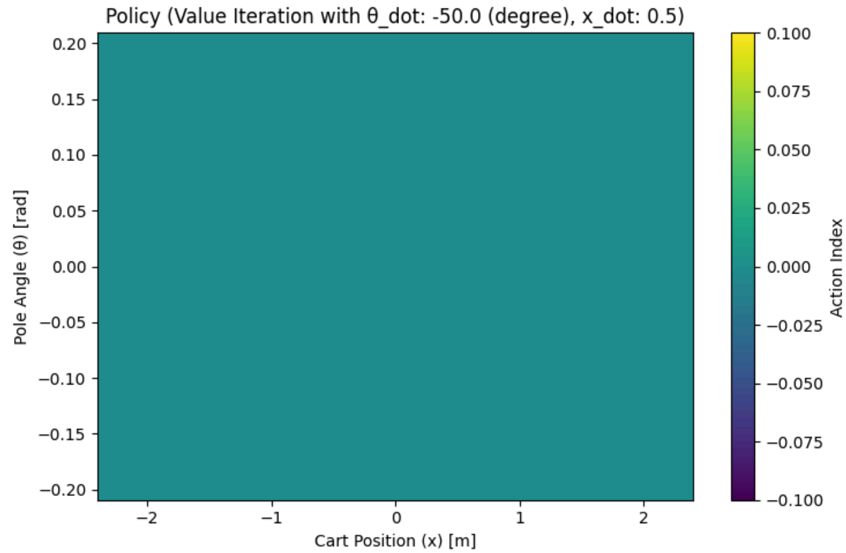

Value Function (Value Iteration with θ_dot: -50.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

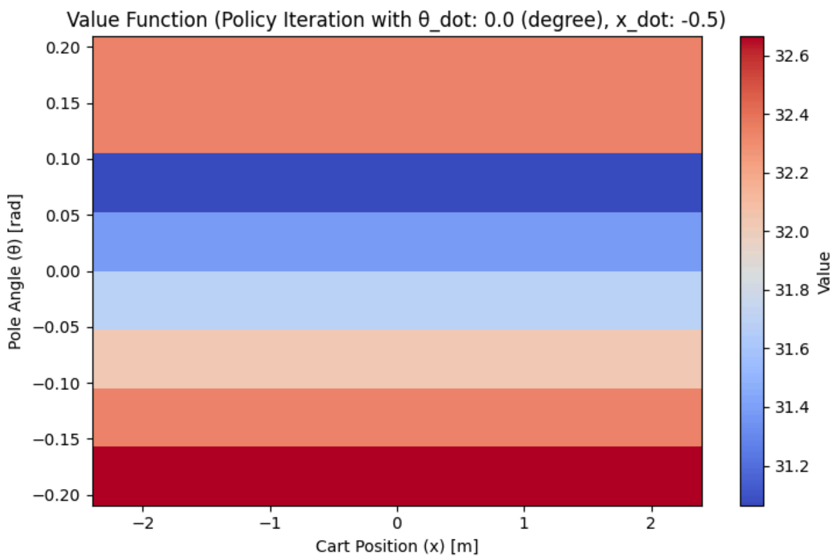

Policy (Value Iteration with θ_dot: -50.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

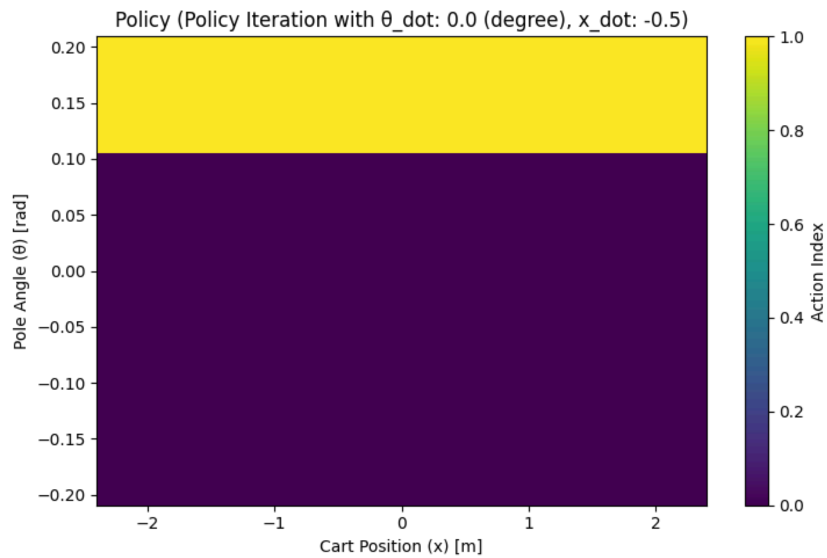

Value Function (Policy Iteration with θ_dot: -50.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

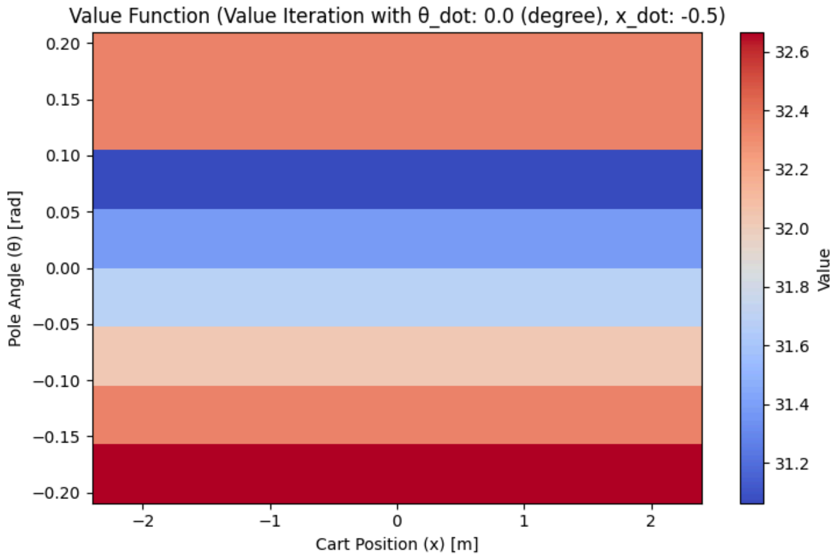

Policy (Policy Iteration with θ_dot: -50.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

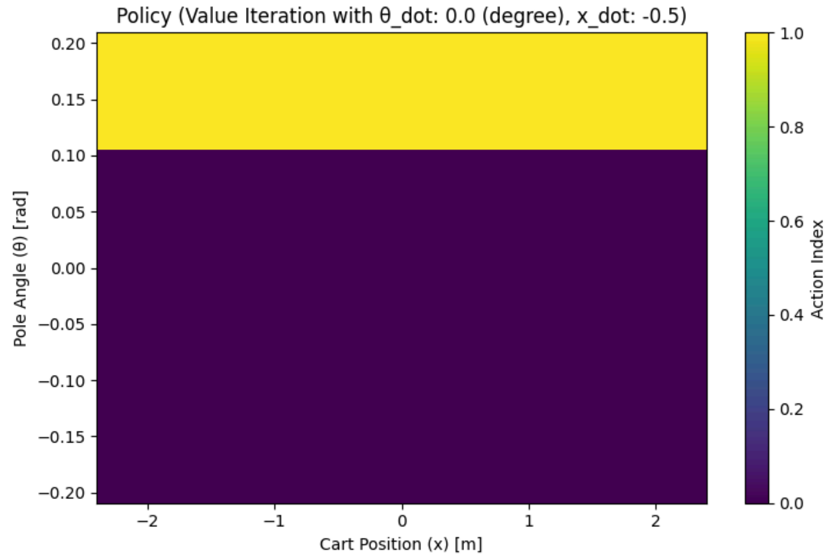

Value Function (Value Iteration with θ_dot: -50.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

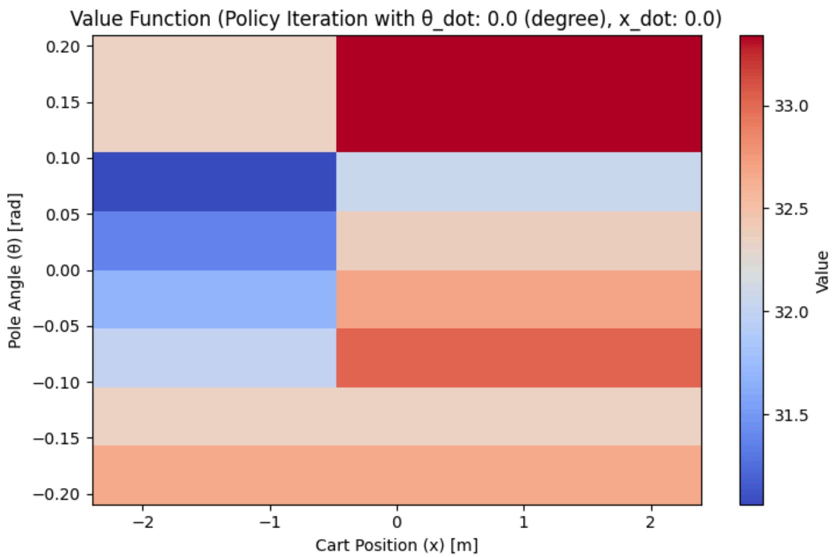

Policy (Value Iteration with θ_dot: -50.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

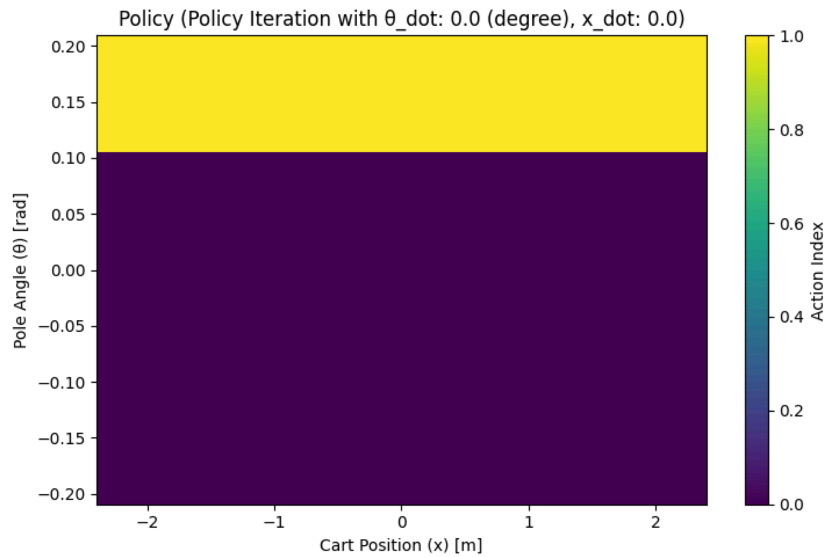

Value Function (Policy Iteration with θ_dot: 0.0 (degree), x_dot: -0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

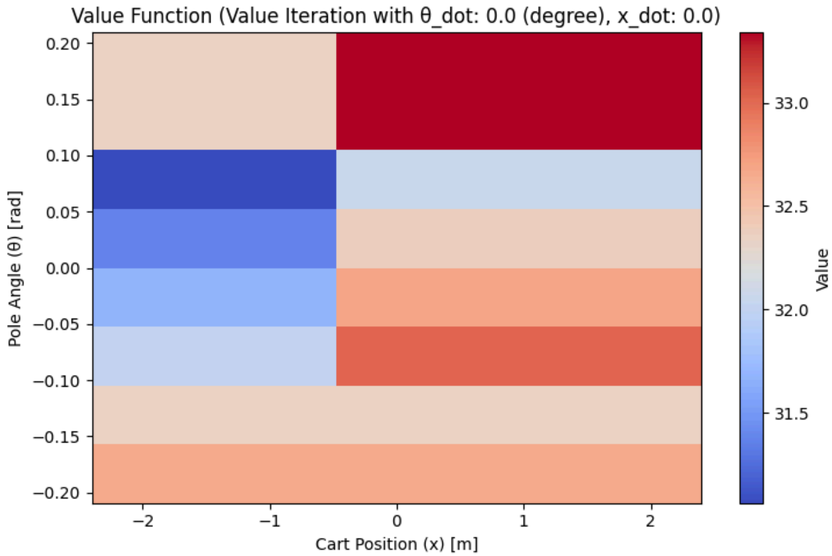

Policy (Policy Iteration with θ_dot: 0.0 (degree), x_dot: -0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

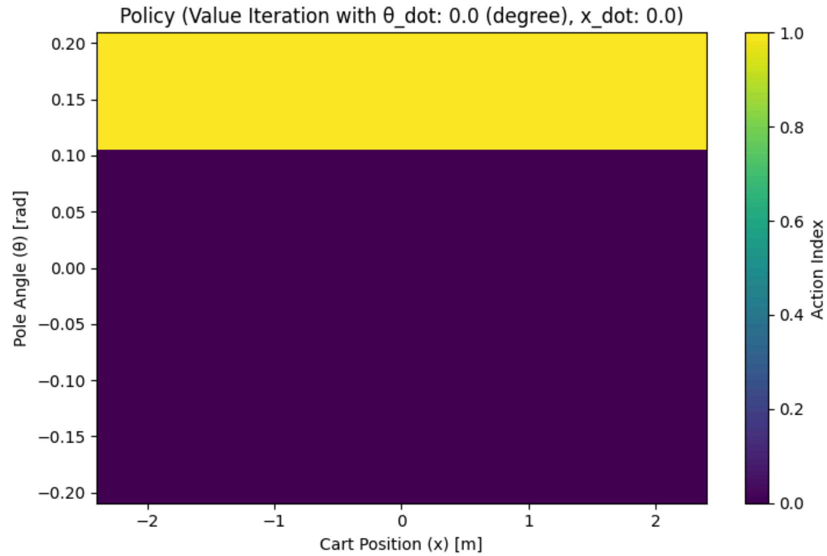

Value Function (Value Iteration with θ_dot: 0.0 (degree), x_dot: -0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

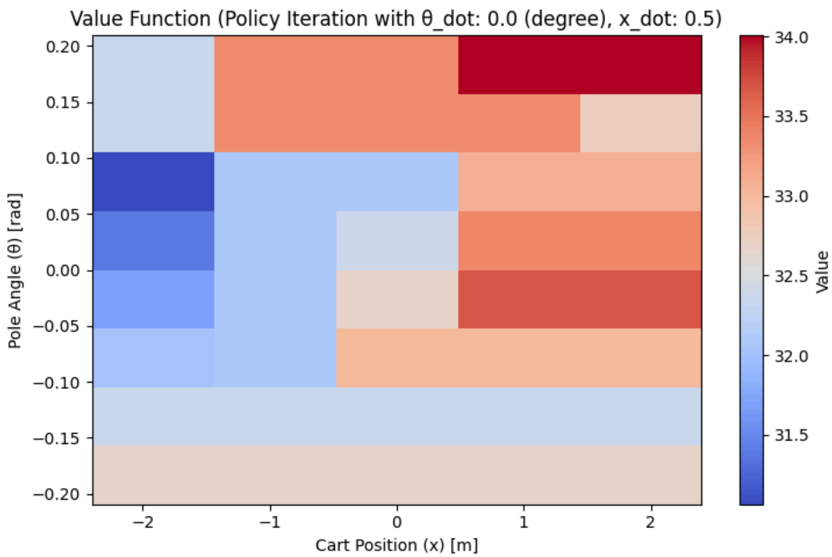

Policy (Value Iteration with θ_dot: 0.0 (degree), x_dot: -0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

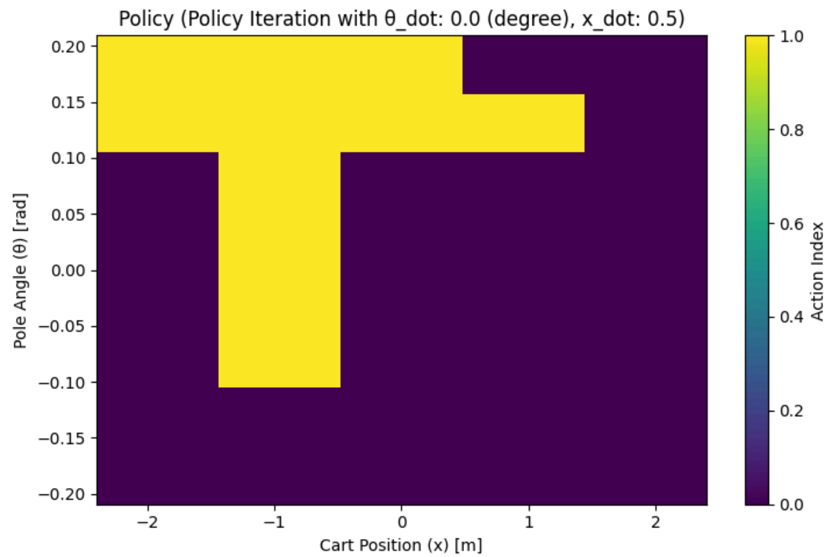

Value Function (Policy Iteration with θ_dot: 0.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

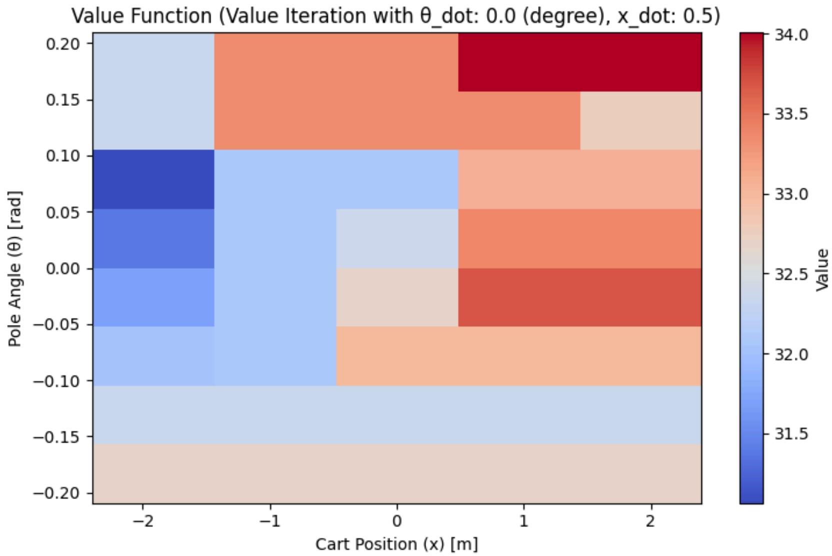

Policy (Policy Iteration with θ_dot: 0.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

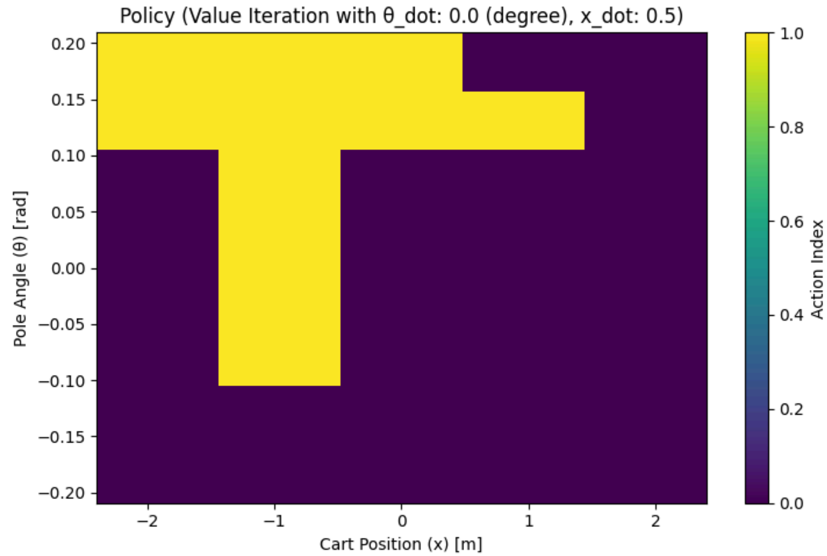

Value Function (Value Iteration with θ_dot: 0.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

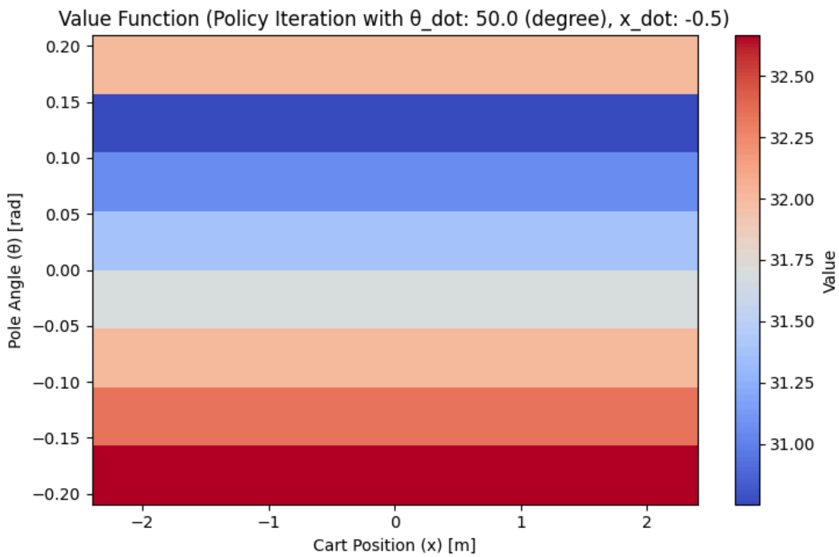

Policy (Value Iteration with θ_dot: 0.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

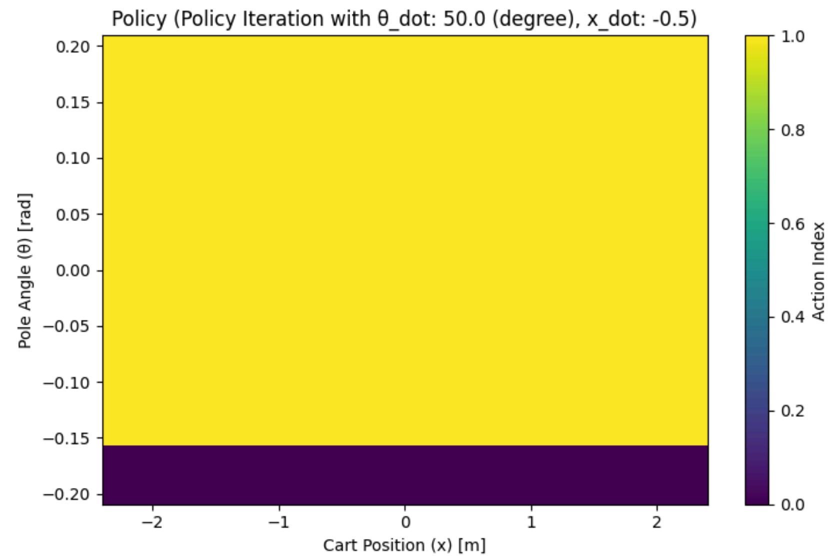

Value Function (Policy Iteration with θ_dot: 0.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

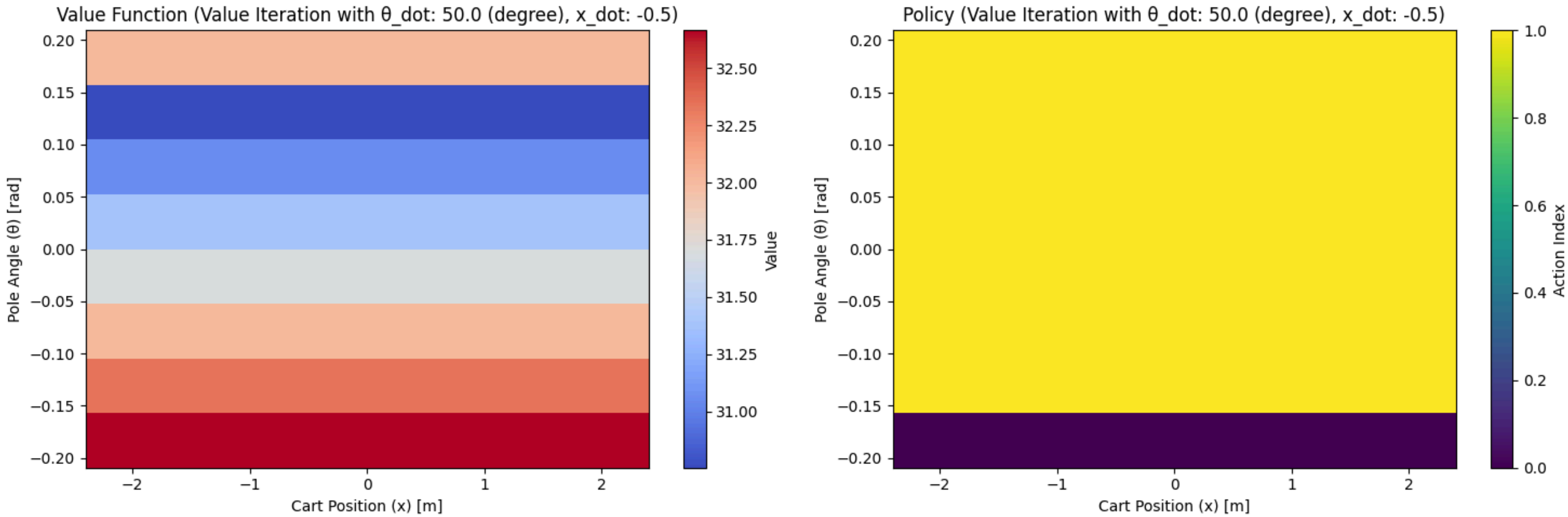

Policy (Policy Iteration with θ_dot: 0.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

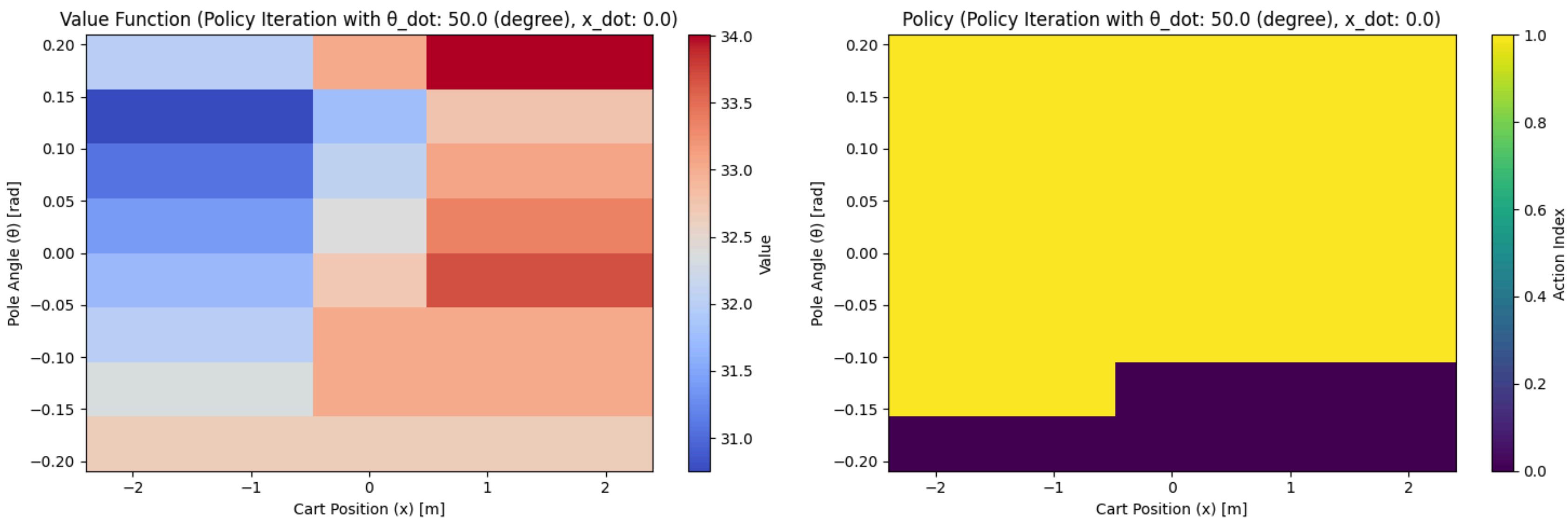

Value Function (Value Iteration with θ_dot: 0.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

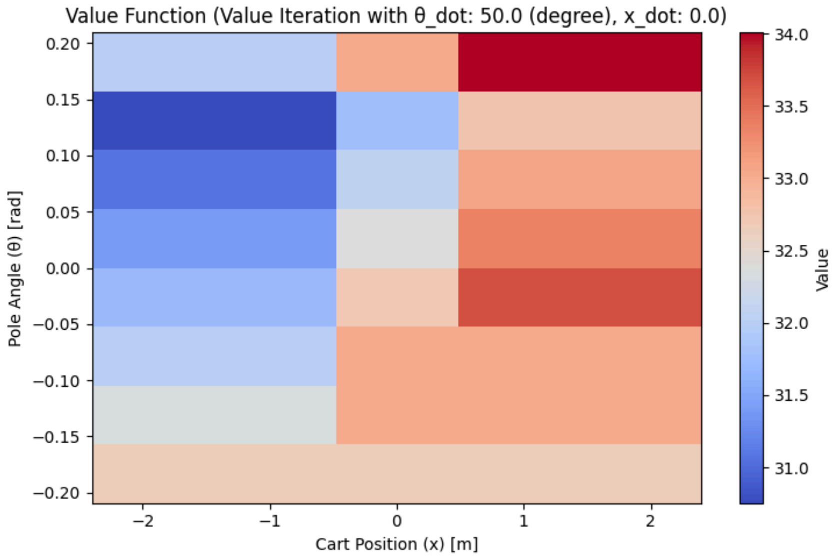

Policy (Value Iteration with θ_dot: 0.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

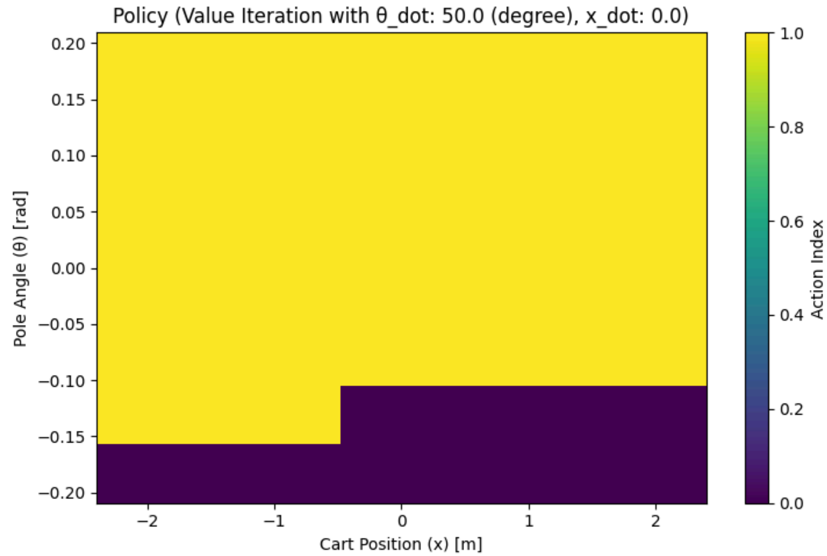

Value Function (Policy Iteration with θ_dot: 50.0 (degree), x_dot: -0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value

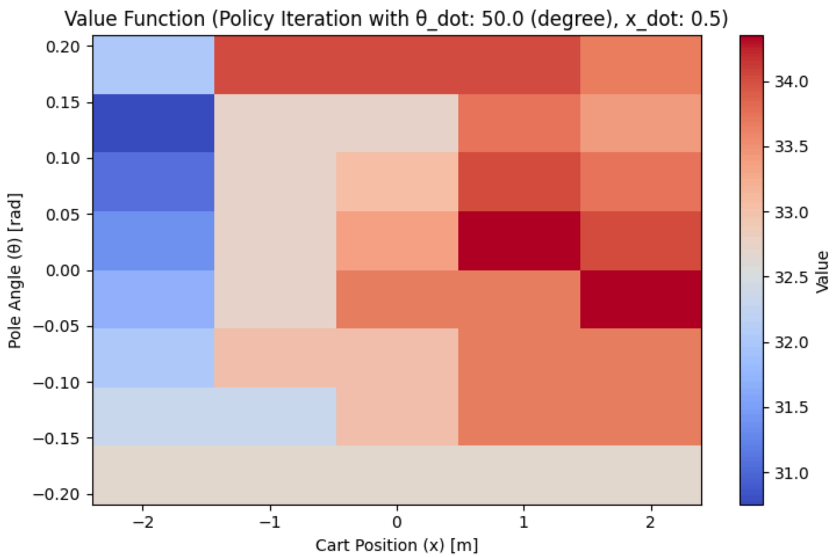

Policy (Policy Iteration with θ_dot: 50.0 (degree), x_dot: -0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index

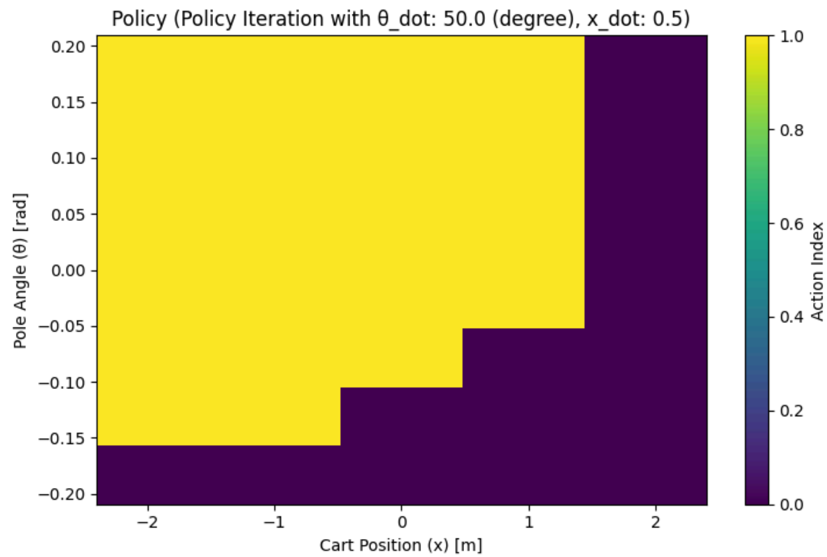

Value Function (Value Iteration with θ_dot: 50.0 (degree), x_dot: -0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value
Policy (Value Iteration with θ_dot: 50.0 (degree), x_dot: -0.5)
Action Index

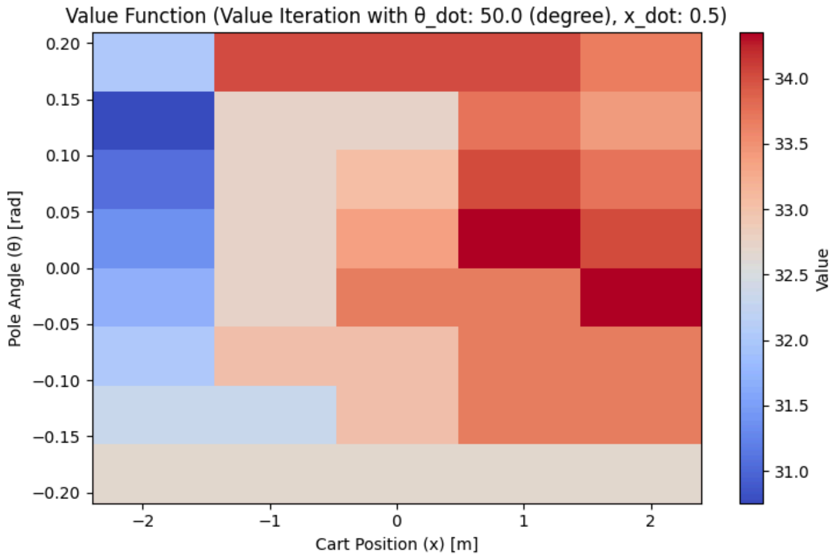

Value Function (Policy Iteration with θ_dot: 50.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value
Policy (Policy Iteration with θ_dot: 50.0 (degree), x_dot: 0.0)
Action Index

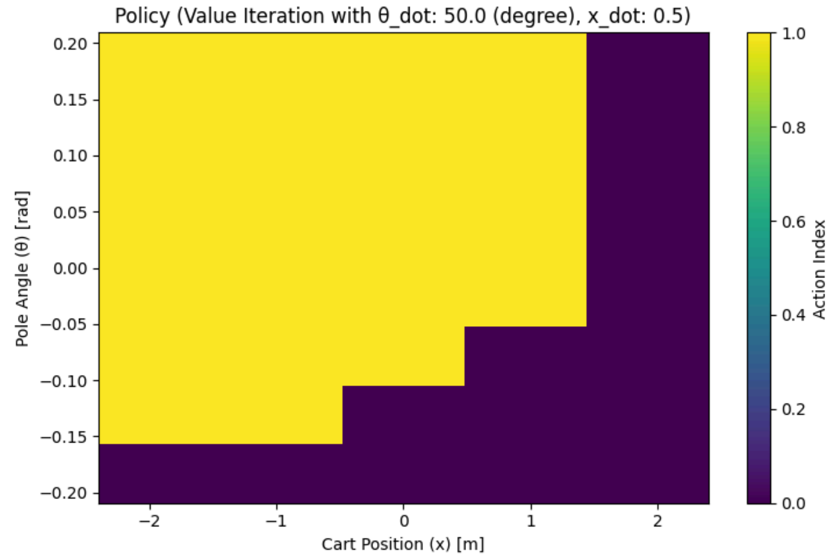

Value Function (Value Iteration with θ_dot: 50.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value


Policy (Value Iteration with θ_dot: 50.0 (degree), x_dot: 0.0)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index


Value Function (Policy Iteration with θ_dot: 50.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value


Policy (Policy Iteration with θ_dot: 50.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index


Value Function (Value Iteration with θ_dot: 50.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Value
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−2
−1
0
1
2
34.0
33.5
33.0
32.5
32.0
31.5
31.0


Policy (Value Iteration with θ_dot: 50.0 (degree), x_dot: 0.5)
Pole Angle (θ) [rad]
Cart Position (x) [m]
Action Index
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−2
−1
0
1
2
1.0
0.8
0.6
0.4
0.2
0.0

```
Value functions from Policy Iteration:
          [[[[32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]]

  [[32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]]

  [[32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]]

  [[32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]]]


 [[[31.37830049 31.37830049 31.37830049 31.37830049]
   [31.37830049 31.37830049 31.37830049 31.37830049]
   [31.37830049 31.37830049 31.37830049 31.37830049]
   [31.37830049 31.37830049 31.37830049 31.37830049]
   [31.37830049 31.37830049 31.37830049 31.37830049]]

  [[32.33880379 32.33880379 32.33880379 32.33880379]
   [32.33880379 32.33880379 32.33880379 32.33880379]
   [32.33880379 32.33880379 32.33880379 32.33880379]
   [32.33880379 32.33880379 32.33880379 32.33880379]
   [32.33880379 32.33880379 32.33880379 32.33880379]]

  [[32.33880379 32.33880379 32.33880379 32.33880379]
   [32.33880379 32.33880379 32.33880379 33.01541575]
   [32.33880379 33.01541575 33.01541575 33.68526159]
   [32.33880379 33.01541575 33.68526159 34.34840898]
```

```
  [32.33880379 33.01541575 33.68526159 33.34840898]]

 [[32.01541575 32.01541575 32.01541575 32.01541575]
  [32.01541575 32.01541575 33.01541575 33.68526159]
  [32.01541575 33.01541575 33.68526159 34.34840898]
  [32.01541575 33.68526159 34.34840898 35.00492489]
  [32.01541575 33.68526159 34.34840898 34.00492489]]]


[[[31.37830049 31.37830049 31.37830049 31.37830049]
  [31.37830049 31.37830049 31.37830049 31.37830049]
  [31.37830049 31.37830049 31.37830049 31.37830049]
  [31.37830049 31.37830049 31.37830049 31.37830049]
  [31.37830049 31.37830049 31.37830049 31.37830049]]

 [[32.01541575 32.01541575 32.01541575 32.01541575]
  [32.01541575 32.01541575 32.06451749 33.01541575]
  [32.01541575 33.01541575 33.01541575 33.01541575]
  [32.01541575 33.01541575 33.01541575 33.01541575]
  [32.01541575 33.01541575 33.01541575 32.01541575]]

 [[32.01541575 32.01541575 32.01541575 32.01541575]
  [32.01541575 32.01541575 33.01541575 33.68526159]
  [32.01541575 33.01541575 33.01541575 33.68526159]
  [32.01541575 33.01541575 33.68526159 34.34840898]
  [32.01541575 33.01541575 33.68526159 33.34840898]]

 [[31.69526159 31.69526159 31.69526159 31.69526159]
  [31.69526159 31.69526159 33.68526159 34.34840898]
  [31.69526159 32.69526159 33.68526159 34.34840898]
  [31.69526159 33.68526159 34.34840898 35.00492489]
  [31.69526159 33.68526159 34.34840898 34.00492489]]]


[[[31.37830049 31.37830049 31.37830049 31.37830049]
  [31.37830049 31.37830049 31.37830049 31.37830049]
  [31.37830049 31.37830049 31.37830049 31.37830049]
  [31.37830049 31.37830049 31.37830049 31.37830049]
  [31.37830049 31.37830049 31.37830049 31.37830049]]

 [[31.69526159 31.69526159 31.69526159 31.69526159]
  [31.69526159 31.69526159 32.06451749 32.74387231]
```

```
  [31.69526159 32.69526159 32.69526159 33.68526159]
  [31.69526159 32.69526159 33.68526159 33.68526159]
  [31.69526159 32.69526159 33.68526159 32.68526159]]

 [[31.69526159 31.69526159 31.69526159 31.69526159]
  [31.69526159 31.69526159 32.74387231 33.68526159]
  [31.69526159 32.69526159 33.68526159 34.34840898]
  [31.69526159 33.68526159 33.68526159 34.34840898]
  [31.69526159 33.68526159 34.34840898 34.00492489]]

 [[31.37830898 31.37830898 31.37830898 31.37830898]
  [31.37830898 31.37830898 33.41643359 34.34840898]
  [31.37830898 32.37830898 34.34840898 35.00492489]
  [31.37830898 33.36830898 34.34840898 35.65486732]
  [31.37830898 34.34840898 34.00492489 33.66487564]]]


[[[32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]]

 [[31.37830898 31.37830898 31.37830898 31.37830898]
  [31.37830898 31.37830898 32.06451749 32.74387231]
  [31.37830898 32.37830898 32.37830898 33.36830898]
  [31.37830898 32.37830898 33.36830898 34.34840898]
  [31.37830898 32.37830898 33.36830898 33.34840898]]

 [[31.37830898 31.37830898 31.37830898 31.37830898]
  [31.37830898 31.37830898 32.74387231 33.41643359]
  [31.37830898 32.37830898 33.36830898 34.34840898]
  [31.37830898 33.36830898 34.34840898 35.00492489]
  [31.37830898 33.36830898 34.03462589 33.69427963]]

 [[31.06452589 31.06452589 31.06452589 31.06452589]
  [31.06452589 31.06452589 33.41643359 34.0822606 ]
  [31.06452589 32.06452589 34.03462589 35.00492489]
  [31.06452589 33.05452589 35.00492489 35.65487564]
  [31.06452589 34.03462589 33.69427963 33.35733683]]]
```

```
[[[32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]]

 [[31.06452589 31.06452589 31.06452589 31.06452589]
  [31.06452589 31.06452589 32.06451749 32.74387231]
  [31.06452589 32.06452589 32.06452589 33.05452589]
  [31.06452589 32.06452589 33.05452589 34.03462589]
  [31.06452589 32.06452589 33.05452589 33.03462589]]

 [[31.06452589 31.06452589 31.06452589 31.06452589]
  [31.06452589 31.06452589 32.74387231 33.41643359]
  [31.06452589 32.06452589 33.05452589 34.03462589]
  [31.06452589 33.05452589 34.03462589 35.00492489]
  [31.06452589 33.05452589 33.72398063 33.38674082]]

 [[30.75388063 30.75388063 30.75388063 30.75388063]
  [30.75388063 30.75388063 33.41643359 34.08226925]
  [30.75388063 31.75388063 33.72398063 34.69427963]
  [30.75388063 32.74388063 34.69427963 35.65487564]
  [30.75388063 33.72398063 33.41644182 33.0822774 ]]]


[[[32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]
  [32.66545837 32.66545837 32.66545837 32.66545837]]

 [[32.33880379 32.33880379 32.33880379 32.33880379]
  [32.33880379 32.33880379 33.33880379 33.33880379]
  [32.33880379 33.33880379 33.33880379 33.33880379]
  [32.33880379 33.33880379 33.33880379 33.72398063]
  [32.33880379 33.33880379 32.74388063 32.72398063]]

 [[30.75388063 30.75388063 30.75388063 30.75388063]
  [30.75388063 30.75388063 32.74387231 33.41643359]
  [30.75388063 31.75388063 32.74388063 33.72398063]
  [30.75388063 32.74388063 33.72398063 34.69427963]
  [30.75388063 32.74388063 33.41644182 33.0822774 ]]
```

```
  [[30.44634182 30.44634182 30.44634182 30.44634182]
   [30.44634182 30.44634182 33.41643359 34.08226925]
   [30.44634182 31.44634182 33.41644182 34.38674082]
   [30.44634182 32.43634182 34.38674082 35.34733683]
   [30.44634182 33.41644182 33.1119784  32.78085862]]]


 [[[32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]
   [32.66545837 32.66545837 32.66545837 32.66545837]]

  [[32.33880379 32.33880379 32.33880379 32.33880379]
   [32.33880379 32.33880379 33.33880379 34.00541575]
   [32.33880379 33.33880379 33.33880379 34.00541575]
   [32.33880379 33.33880379 34.00541575 34.00541575]
   [32.33880379 33.33880379 34.00541575 32.41644182]]

  [[32.01541575 32.01541575 32.01541575 32.01541575]
   [32.01541575 32.01541575 34.00541575 34.00541575]
   [32.01541575 33.01541575 34.00541575 34.00541575]
   [32.01541575 34.00541575 34.00541575 34.38674082]
   [32.01541575 34.00541575 33.66536159 33.32870798]]

  [[31.69526159 31.69526159 31.69526159 31.69526159]
   [31.69526159 31.69526159 33.66536159 33.66536159]
   [31.69526159 31.69526159 33.66536159 33.66536159]
   [31.69526159 32.68526159 33.66536159 34.04287341]
   [31.69526159 33.66536159 33.32870798 32.9954209 ]]]]

Value functions from Value Iteration:
           [[[[32.66544281 32.66544281 32.66544281 32.66544281]
   [32.66544281 32.66544281 32.66544281 32.66544281]
   [32.66544281 32.66544281 32.66544281 32.66544281]
   [32.66544281 32.66544281 32.66544281 32.66544281]
   [32.66544281 32.66544281 32.66544281 32.66544281]]

  [[32.66544281 32.66544281 32.66544281 32.66544281]
   [32.66544281 32.66544281 32.66544281 32.66544281]
   [32.66544281 32.66544281 32.66544281 32.66544281]
```

```
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]]

 [[32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]]

 [[32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]]]


[[[31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]]

 [[32.33878838 32.33878838 32.33878838 32.33878838]
  [32.33878838 32.33878838 32.33878838 32.33878838]
  [32.33878838 32.33878838 32.33878838 32.33878838]
  [32.33878838 32.33878838 32.33878838 32.33878838]
  [32.33878838 32.33878838 32.33878838 32.33878838]]

 [[32.33878838 32.33878838 32.33878838 32.33878838]
  [32.33878838 32.33878838 32.33878838 33.0154005 ]
  [32.33878838 33.0154005  33.0154005  33.68524649]
  [32.33878838 33.0154005  33.68524649 34.34839403]
  [32.33878838 33.0154005  33.68524649 33.34839403]]

 [[32.0154005  32.0154005  32.0154005  32.0154005 ]
  [32.0154005  32.0154005  33.0154005  33.68524649]
  [32.0154005  33.0154005  33.68524649 34.34839403]
  [32.0154005  33.68524649 34.34839403 35.00491009]
  [32.0154005  33.68524649 34.34839403 34.00491009]]]


[[[31.37828461 31.37828461 31.37828461 31.37828461]
```

```
  [31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]]

 [[32.0154005  32.0154005  32.0154005  32.0154005 ]
  [32.0154005  32.0154005  32.06450177 33.0154005 ]
  [32.0154005  33.0154005  33.0154005  33.0154005 ]
  [32.0154005  33.0154005  33.0154005  33.0154005 ]
  [32.0154005  33.0154005  33.0154005  32.0154005 ]]

 [[32.0154005  32.0154005  32.0154005  32.0154005 ]
  [32.0154005  32.0154005  33.0154005  33.68524649]
  [32.0154005  33.0154005  33.0154005  33.68524649]
  [32.0154005  33.0154005  33.68524649 34.34839403]
  [32.0154005  33.0154005  33.68524649 33.34839403]]

 [[31.69524649 31.69524649 31.69524649 31.69524649]
  [31.69524649 31.69524649 33.68524649 34.34839403]
  [31.69524649 32.69524649 33.68524649 34.34839403]
  [31.69524649 33.68524649 34.34839403 35.00491009]
  [31.69524649 33.68524649 34.34839403 34.00491009]]]


[[[31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]
  [31.37828461 31.37828461 31.37828461 31.37828461]]

 [[31.69524649 31.69524649 31.69524649 31.69524649]
  [31.69524649 31.69524649 32.06450177 32.74385675]
  [31.69524649 32.69524649 32.69524649 33.68524649]
  [31.69524649 32.69524649 33.68524649 33.68524649]
  [31.69524649 32.69524649 33.68524649 32.68524649]]

 [[31.69524649 31.69524649 31.69524649 31.69524649]
  [31.69524649 31.69524649 32.74385675 33.68524649]
  [31.69524649 32.69524649 33.68524649 34.34839403]
  [31.69524649 33.68524649 33.68524649 34.34839403]
  [31.69524649 33.68524649 34.34839403 34.00491009]]
```

```
 [[31.37829403 31.37829403 31.37829403 31.37829403]
  [31.37829403 31.37829403 33.41641818 34.34839403]
  [31.37829403 32.37829403 34.34839403 35.00491009]
  [31.37829403 33.36829403 34.34839403 35.65485176]
  [31.37829403 34.34839403 34.00491009 33.66486099]]]


[[[32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]]

 [[31.37829403 31.37829403 31.37829403 31.37829403]
  [31.37829403 31.37829403 32.06450177 32.74385675]
  [31.37829403 32.37829403 32.37829403 33.36829403]
  [31.37829403 32.37829403 33.36829403 34.34839403]
  [31.37829403 32.37829403 33.36829403 33.34839403]]

 [[31.37829403 31.37829403 31.37829403 31.37829403]
  [31.37829403 31.37829403 32.74385675 33.41641818]
  [31.37829403 32.37829403 33.36829403 34.34839403]
  [31.37829403 33.36829403 34.34839403 35.00491009]
  [31.37829403 33.36829403 34.03461109 33.69426498]]

 [[31.06451109 31.06451109 31.06451109 31.06451109]
  [31.06451109 31.06451109 33.41641818 34.082254  ]
  [31.06451109 32.06451109 34.03461109 35.00491009]
  [31.06451109 33.05451109 35.00491009 35.65486099]
  [31.06451109 34.03461109 33.69426498 33.35732233]]]


[[[32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]]

 [[31.06451109 31.06451109 31.06451109 31.06451109]
  [31.06451109 31.06451109 32.06450177 32.74385675]
  [31.06451109 32.06451109 32.06451109 33.05451109]
  [31.06451109 32.06451109 33.05451109 34.03461109]
```

```
  [31.06451109 32.06451109 33.05451109 33.03461109]]

 [[31.06451109 31.06451109 31.06451109 31.06451109]
  [31.06451109 31.06451109 32.74385675 33.41641818]
  [31.06451109 32.06451109 33.05451109 34.03461109]
  [31.06451109 33.05451109 34.03461109 35.00491009]
  [31.06451109 33.05451109 33.72396598 33.38672632]]

 [[30.75386598 30.75386598 30.75386598 30.75386598]
  [30.75386598 30.75386598 33.41641818 34.082254  ]
  [30.75386598 31.75386598 33.72396598 34.69426498]
  [30.75386598 32.74386598 34.69426498 35.65486099]
  [30.75386598 33.72396598 33.41642732 33.08226304]]]


[[[32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]]

 [[32.33878838 32.33878838 32.33878838 32.33878838]
  [32.33878838 32.33878838 33.33878838 33.33878838]
  [32.33878838 33.33878838 33.33878838 33.33878838]
  [32.33878838 33.33878838 33.33878838 33.72396598]
  [32.33878838 33.33878838 32.74386598 32.72396598]]

 [[30.75386598 30.75386598 30.75386598 30.75386598]
  [30.75386598 30.75386598 32.74385675 33.41641818]
  [30.75386598 31.75386598 32.74386598 33.72396598]
  [30.75386598 32.74386598 33.72396598 34.69426498]
  [30.75386598 32.74386598 33.41642732 33.08226304]]

 [[30.44632732 30.44632732 30.44632732 30.44632732]
  [30.44632732 30.44632732 33.41641818 34.082254  ]
  [30.44632732 31.44632732 33.41642732 34.38672632]
  [30.44632732 32.43632732 34.38672632 35.34732233]
  [30.44632732 33.41642732 33.11196404 32.7808444 ]]]


[[[32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
```

```
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]
  [32.66544281 32.66544281 32.66544281 32.66544281]]

 [[32.33878838 32.33878838 32.33878838 32.33878838]
  [32.33878838 32.33878838 33.33878838 34.0054005 ]
  [32.33878838 33.33878838 33.33878838 34.0054005 ]
  [32.33878838 33.33878838 34.0054005  34.0054005 ]
  [32.33878838 33.33878838 34.0054005  32.41642732]]

 [[32.0154005  32.0154005  32.0154005  32.0154005 ]
  [32.0154005  32.0154005  34.0054005  34.0054005 ]
  [32.0154005  33.0154005  34.0054005  34.0054005 ]
  [32.0154005  34.0054005  34.0054005  34.38672632]
  [32.0154005  34.0054005  33.66534649 33.32869303]]

 [[31.69524649 31.69524649 31.69524649 31.69524649]
  [31.69524649 31.69524649 33.66534649 33.66534649]
  [31.69524649 31.69524649 33.66534649 33.66534649]
  [31.69524649 32.68524649 33.66534649 34.04285905]
  [31.69524649 33.66534649 33.32869303 32.9954061 ]]]]
```