Value Iteration

```
num_iterations = 100
import matplotlib.pyplot as plt
import numpy as np

num_rows = 4
num_cols = 4
num_states = num_rows * num_cols
num_actions = 4
num_episodes = 500
alpha = 0.6
gamma = 0.9
```

```
# Define rewards, goal state, and obstacles
rewards = np.random.randint(1, 10, num_states)
goal_state = num_states - 1
obstacles = [5, 7, 10]
rewards[goal_state] = 10
for obstacle in obstacles:
    rewards[obstacle] = -10
```

```
# Function to convert state index to coordinates
def state_to_coords(state):
    row = state // num_cols
    col = state % num_cols
    return row, col

# Function to convert coordinates to state index
def coords_to_state(row, col):
    return row * num_cols + col

# Initialize policy randomly
policy = np.random.randint(0, num_actions, num_states)
```

```
# Initialize value function V randomly
V = np.random.rand(num_states)

# Value Iteration
for _ in range(num_iterations):
    delta = 0
    for s in range(num_states):
        v = V[s]
        row, col = state_to_coords(s)
        next_states = []
        if row > 0:  # Up
            next_states.append(coords_to_state(row - 1, col))
        if row < num_rows - 1:  # Down
            next_states.append(coords_to_state(row + 1, col))
        if col > 0:  # Left
            next_states.append(coords_to_state(row, col - 1))
        if col < num_cols - 1:  # Right
            next_states.append(coords_to_state(row, col + 1))
        q_values = [rewards[next_state] + gamma * V[next_state] for next_state in next_states]
        V[s] = max(q_values)
        delta = max(delta, abs(v - V[s]))
    if delta < 1e-6:
        break
```

```
# Extract optimal policy from value function
optimal_policy = np.zeros(num_states, dtype=int)
for s in range(num_states):
    row, col = state_to_coords(s)
    next_states = []
    if row > 0:  # Up
        next_states.append(coords_to_state(row - 1, col))
    if row < num_rows - 1:  # Down
        next_states.append(coords_to_state(row + 1, col))
    if col > 0:  # Left
        next_states.append(coords_to_state(row, col - 1))
    if col < num_cols - 1:  # Right
        next_states.append(coords_to_state(row, col + 1))
    q_values = [rewards[next_state] + gamma * V[next_state] for next_state in next_states]
    optimal_policy[s] = np.argmax(q_values)

# Display the final optimal policy and values
optimal_policy_matrix = np.array(['U', 'D', 'L', 'R'])[optimal_policy]
optimal_policy_matrix = optimal_policy_matrix.reshape((num_rows, num_cols))
```

```
print("Final Optimal Policy:")
print(optimal_policy_matrix)
print("\nOptimal Values:")
print(V.reshape((num_rows, num_cols)))
```

```
Final Optimal Policy:
[['D' 'L' 'D' 'D']
 ['U' 'U' 'U' 'U']
 ['U' 'L' 'U' 'D']
 ['U' 'D' 'L' 'D']]

Optimal Values:
[[85.26315412 84.73683871 85.26315484 84.73683936]
 [81.73683871 85.26315484 84.73683936 80.26315542]
 [76.56315484 74.90683936 77.26315542 76.31578704]
 [74.90683936 74.41615542 76.31578704 73.68420833]]
```