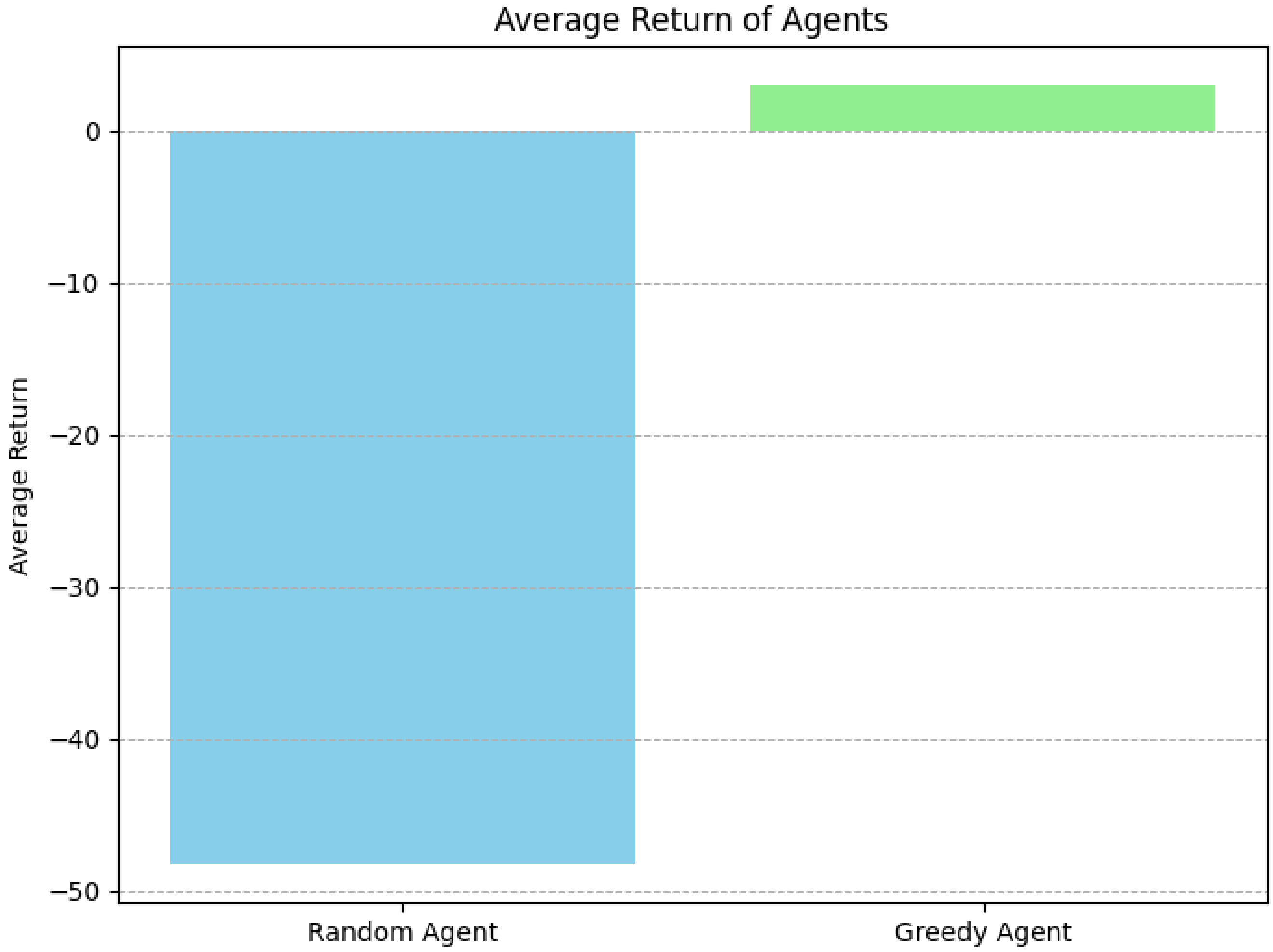
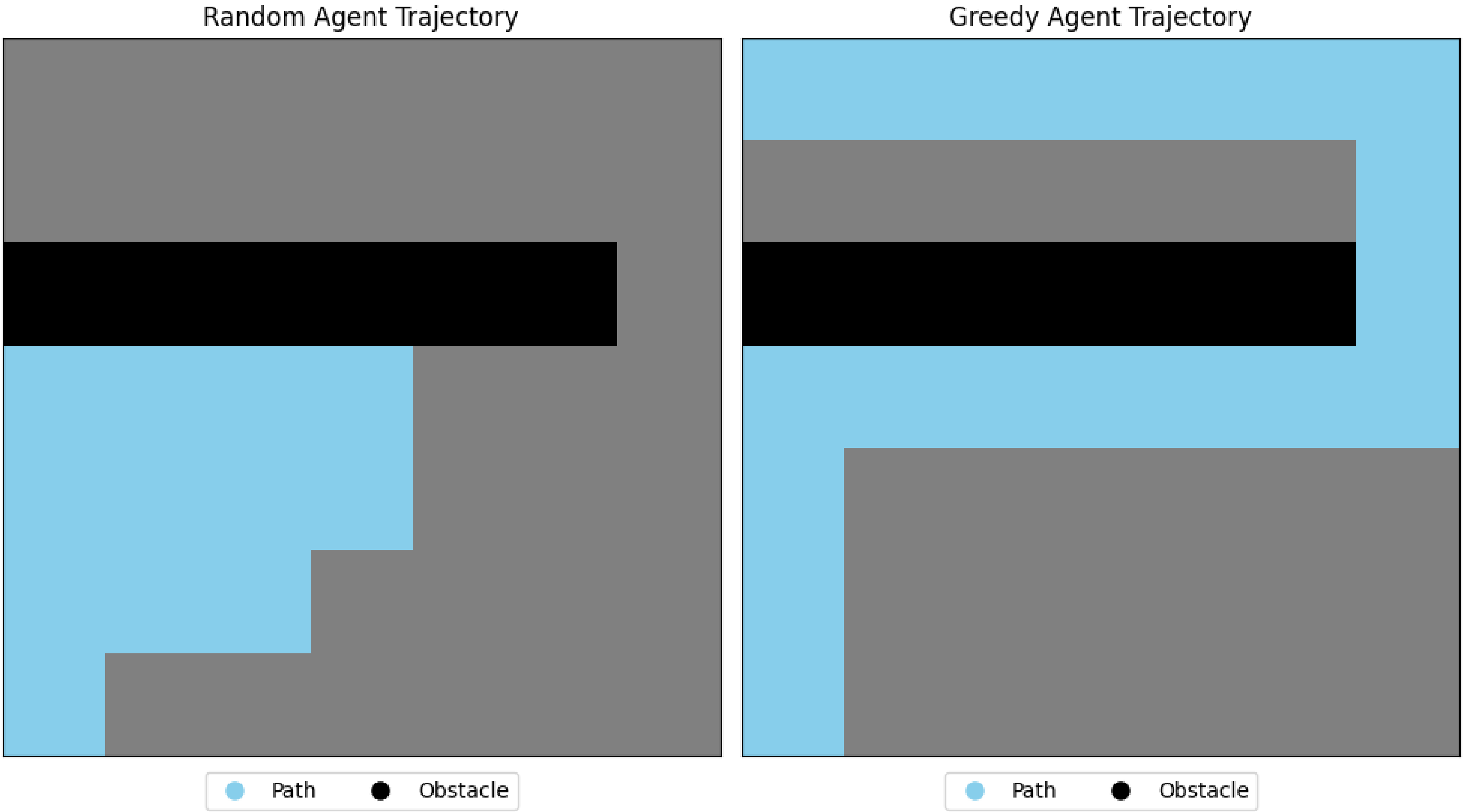
```python
#Nihal Ranchod - 2427378
#Lisa Godwin - 2437980

import numpy as np
import random
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

class GridworldMDP:
    def __init__(self):
        self.grid_size = 7
        self.initial_state = (6, 0)
        self.goal_state = (0, 0)
        self.obstacles = [(2, i) for i in range(6)]
        self.actions = ['up', 'down', 'left', 'right']
        self.state = self.initial_state

    # Reset the environment to the initial state
    def reset(self):
        self.state = self.initial_state
        return self.state

    def step(self, action):
        if action not in self.actions:
            raise ValueError("Invalid action")

        x, y = self.state
        if action == 'up':
            x = max(x - 1, 0)
        elif action == 'down':
            x = min(x + 1, self.grid_size - 1)
        elif action == 'left':
            y = max(y - 1, 0)
        elif action == 'right':
            y = min(y + 1, self.grid_size - 1)

        new_state = (x, y)
        if new_state in self.obstacles:
            new_state = self.state

        reward = -1
        if new_state == self.goal_state:
            reward = 20

        self.state = new_state
        return new_state, reward

# Random Agent
def random_agent(env, steps=50):
    state = env.reset()
    total_reward = 0
    trajectory = [state]
    for _ in range(steps):
        action = random.choice(env.actions)
        state, reward = env.step(action)
        total_reward += reward
        trajectory.append(state)
        if state == env.goal_state:
            break
    return total_reward, trajectory

# Optimal Value Grid
optimal_value_function = np.array([
    [20, 19, 18, 17, 16, 15, 14],
    [19, 18, 17, 16, 15, 14, 13],
    [-1, -1, -1, -1, -1, -1, 12],
    [5, 6, 7, 8, 9, 10, 11],
    [4, 5, 6, 7, 8, 9, 10],
    [3, 4, 5, 6, 7, 8, 9],
    [2, 3, 4, 5, 6, 7, 8],
])

# Greedy Agent
def greedy_agent(env, optimal_value_grid, steps=50):
    state = env.reset()
    total_reward = 0
    trajectory = [state]
    for _ in range(steps):
        x, y = state
        best_action = None
        best_value = -float('inf')

        for action in env.actions:
            if action == 'up':
                new_x, new_y = max(x - 1, 0), y
            elif action == 'down':
                new_x, new_y = min(x + 1, env.grid_size - 1), y
            elif action == 'left':
                new_x, new_y = x, max(y - 1, 0)
            elif action == 'right':
                new_x, new_y = x, min(y + 1, env.grid_size - 1)

            if (new_x, new_y) not in env.obstacles and optimal_value_grid[new_x][new_y] > best_value:
                best_value = optimal_value_grid[new_x][new_y]
                best_action = action

        state, reward = env.step(best_action)
        total_reward += reward
        trajectory.append(state)
        if state == env.goal_state:
            break
    return total_reward, trajectory

# Plot sample trajectories
def plot_trajectories(random_agent_trajectories, greedy_agent_trajectories):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

```

```python
    def plot_trajectory(ax, trajectory, title):
        grid = np.zeros((GRID_SIZE, GRID_SIZE))
        for x, y in OBSTACLES:
            grid[x, y] = -1
        for x, y in trajectory:
            grid[x, y] = 1

        cmap = mcolors.ListedColormap(['black', 'gray', 'skyblue'])
        bounds = [-1, 0, 1, 2]
        norm = mcolors.BoundaryNorm(bounds, cmap.N)
        ax.imshow(grid, cmap=cmap, norm=norm, origin='upper')

        ax.set_title(title)
        ax.set_xticks([])
        ax.set_yticks([])
        legend_labels = ['Path', 'Obstacle']
        legend_colors = ['skyblue', 'black']
        handles = [plt.Line2D([0], [0], marker='o', color='w', label=label, markersize=10, markerfacecolor=color) for label, color in zip(legend_labels, legend_colors)]
        ax.legend(handles=handles, loc='center', bbox_to_anchor=(0.5, -0.05), ncol=2)

    plot_trajectory(ax1, random_agent_trajectories[0], 'Random Agent Trajectory')
    plot_trajectory(ax2, greedy_agent_trajectories[0], 'Greedy Agent Trajectory')
    plt.tight_layout()
    plt.show()

def main():
    # Define global variables for plotting
    global GRID_SIZE, OBSTACLES
    GRID_SIZE = 7
    OBSTACLES = [(2, i) for i in range(6)]

    # Run experiments
    env = GridworldMDP()
    random_trajectories = [random_agent(env)[1] for _ in range(20)]
    greedy_trajectories = [greedy_agent(env, optimal_value_function)[1] for _ in range(20)]

    # Compute average returns
    random_returns = [random_agent(env)[0] for _ in range(20)]
    greedy_returns = [greedy_agent(env, optimal_value_function)[0] for _ in range(20)]

    #print(f'Random Agent Returns: {random_returns}')
    #print(f'Greedy Agent Returns: {greedy_returns}')

    # Plot average returns
    plt.figure(figsize=(8, 6))
    plt.bar(['Random Agent', 'Greedy Agent'], [np.mean(random_returns), np.mean(greedy_returns)], color=['skyblue', 'lightgreen'])
    plt.ylabel('Average Return')
    plt.title('Average Return of Agents')
    plt.grid(axis='y', linestyle='--', linewidth=0.7)
    plt.show()

    # Plot sample trajectories
    plot_trajectories(random_trajectories[:1], greedy_trajectories[:1])

if __name__ == '__main__':
    main()
```


Average Return of Agents
Average Return
0
−10
−20
−30
−40
−50
Random Agent
Greedy Agent


Random Agent Trajectory
Path
Obstacle
Greedy Agent Trajectory
Path
Obstacle