In [37]:
```
import numpy as np
import random
import time
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [38]:
```
 # ---- Step 2: Define Maze Environment ---
# Maze Layout:
 # 0 - free cell
 # 1 - wall
 # 2 - goal
maze = np.array([
 [1, 0, 0, 0, 0],
 [0, 1, 0, 1, 0],
 [0, 0, 0, 1, 2],
 [0, 1, 0, 0, 0],
 [0, 0, 0, 1, 0]
 ])
```

In [39]:
```
 # Maze shape
rows, cols = maze.shape
 # Define possible actions (Up, Down, Left, Right)
actions = ['up', 'down', 'left', 'right']
```

In [40]:
```
# ---- Step 3: Define Parameters ---
alpha = 0.7 # Learning rate
gamma = 0.9 # Discount factor
epsilon = 0.1 # Exploration rate
episodes = 500 # Number of training episodes
max_steps = 100 # Max steps per episode
```

In [41]:
```
# Initialize Q-table (rows x cols x actions)
Q = np.zeros((rows, cols, len(actions)))
```

In [42]:
```
 # ---- Helper Function: Get Next State and Reward ---
def get_next_state_reward(state, action):
    i, j = state
    if action == 'up':
        i = max(i-1, 0)
    elif action == 'down':
        i = min(i + 1, rows-1)
    elif action == 'left':
        j = max(j-1, 0)
    elif action == 'right':
        j = min(j + 1, cols-1)
 # If it's a wall, no movement
    if maze[i, j] == 1:
        return state,-5 # penalty for hitting a wall
    elif maze[i, j] == 2:
        return (i, j), 10 # reward for reaching goal
```

```
    else:
        return (i, j),-1 # small penalty for each move
```

In [43]:

```
# ---- Step 4: Train the Agent ---
rewards_per_episode = []
for episode in range(episodes):
 state = (0, 0) # start at top-left corner
 total_reward = 0
 for step in range(max_steps):
    # Choose action (ε-greedy policy)
    if random.uniform(0, 1) < epsilon:
        action_index = random.choice(range(len(actions)))
    else:
        action_index = np.argmax(Q[state[0], state[1]])
    action = actions[action_index]
    next_state, reward = get_next_state_reward(state, action)
    # Update Q-value using Q-learning formula
    best_next_action = np.max(Q[next_state[0], next_state[1]])
    Q[state[0], state[1], action_index] += alpha * (
        reward + gamma * best_next_action-Q[state[0], state[1], action_index]
    )
    state = next_state
    total_reward += reward
 # Stop if goal is reached
    if maze[state] == 2:
        break
rewards_per_episode.append(total_reward)
```

In [44]:

```
 # ---- Step 5: Results ---
print("\nTraining Completed!")
print("Q-Table Learned (Partial View):")
print(np.round(Q, 2))
```

```
Training Completed!
Q-Table Learned (Partial View):
[[[-3.37 -0.43 -3.37  1.81]
  [ 1.77 -2.19 -2.19  3.12]
  [ 3.12  1.81  1.81  4.58]
  [ 4.58  0.58  3.12  6.2 ]
  [ 6.2   8.    4.58  6.2 ]]

 [[-6.76  0.63 -3.98 -6.76]
  [ 0.    0.    0.    0.  ]
  [ 3.12 -0.8  -3.5  -2.59]
  [ 0.    0.    0.    0.  ]
  [ 6.2  10.    4.    8.  ]]

 [[-3.77 -4.02 -3.44  1.81]
  [-3.5  -3.03 -3.34  3.12]
  [ 0.65  4.58 -2.36 -3.5 ]
  [ 0.    0.    0.    0.  ]
  [ 0.    0.    0.    0.  ]]

 [[-0.6  -3.58 -3.53 -3.5 ]
  [ 0.    0.    0.    0.  ]
  [-1.48  1.9  -3.5   6.2 ]
  [-3.5  -3.5  -1.14  8.  ]
  [10.    0.    0.    5.6 ]]

 [[-3.51 -3.53 -3.53 -2.86]
  [-3.5  -2.98 -3.01  1.5 ]
  [ 4.52 -1.96 -2.42 -3.5 ]
  [ 0.    0.    0.    0.  ]
  [ 0.    0.    0.    0.  ]]]
```

In [45]:

```
# ---- Step 6: Test the Agent ---
state = (0, 0)
path = [state]
for _ in range(50):
 action_index = np.argmax(Q[state[0], state[1]])
 action = actions[action_index]
 next_state, reward = get_next_state_reward(state, action)
 path.append(next_state)
 state = next_state
 if maze[state] == 2:
    print("\nGoal reached!")
    break
print("\nOptimal Path Learned by Agent:")
print(path)
```

```
Goal reached!

Optimal Path Learned by Agent:
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4)]
```

In [46]:

```
# ---- Step 7: Visualize Path on Maze ---
maze_display = maze.copy()
for step in path:
```

```
    if maze_display[step] == 0:
        maze_display[step] = 3 # mark visited path
```

In [47]:
```
plt.figure(figsize=(5,5))
sns.heatmap(maze_display, cmap="coolwarm", cbar=False, linewidths=0.5,
 linecolor='black', annot=True, fmt=".0f",
 annot_kws={'size': 12})
plt.title("Maze Navigation Path (3 = Path, 2 = Goal, 1 = Wall, 0 = Free)")
plt.show()
```