In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import random
from IPython import get_ipython
from IPython.display import display, HTML

# --- 1. 初期設定 ---
N_CITY = 64 #16  # 都市数
MAX_COUNT = 10000  # 最大試行回数
TEMP = 0.02   # 温度（アニーリングの温度）

print("ライブラリのインポートと初期設定が完了しました。")
```

```
ライブラリのインポートと初期設定が完了しました。
```

In [2]:
```python
# --- 2. 都市の座標と距離行列の生成 ---
# 0から1の範囲で都市の座標をランダムに生成
positions = np.random.rand(N_CITY, 2)

# 都市間の距離を行列として計算
distance_matrix = np.zeros((N_CITY, N_CITY))
for i in range(N_CITY):
    for j in range(i + 1, N_CITY):
        dist = np.linalg.norm(positions[i] - positions[j])
        distance_matrix[i, j] = dist
        distance_matrix[j, i] = dist

print(f"{N_CITY}個の都市の座標を生成し、距離行列を計算しました。")
# print("都市の座標:\n", positions)
```

```
64個の都市の座標を生成し、距離行列を計算しました。
```

In [3]:
```python
# --- 3. 経路関連の関数 ---
def calculate_total_distance(path, dist_matrix):
    """経路全体の距離を計算する"""
    total_dist = 0
    # pathは [0, 1, ..., N-1, 0] のようになっている
    for i in range(len(path) - 1):
        total_dist += dist_matrix[path[i], path[i+1]]
    return total_dist

def create_new_path(current_path, selector, n_city):
    """新しい経路候補を生成する"""
    # 経路の一部をランダムに2点選択（始点と終点は除く）
    i, j = random.sample(range(1, n_city), 2)

    new_path = list(current_path)  # コピーを作成

    if selector == 'two_city_swap':
        # 2点交換: 2都市の位置を入れ替える
        new_path[i], new_path[j] = new_path[j], new_path[i]

    elif selector == 'route_reverse':
        # 2-opt: 区間 [i, j] を反転
        if i > j:
            i, j = j, i
        new_path[i:j+1] = reversed(new_path[i:j+1])
```

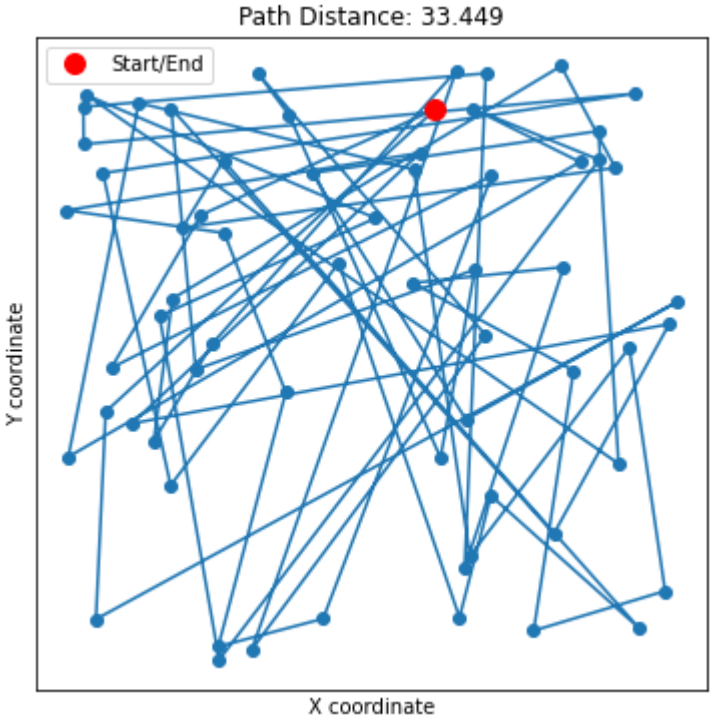
```
    return new_path

print("経路距離計算関数を定義しました。")
```

経路距離計算関数を定義しました。

In [4]:
```
# --- 4. メインループ（シミュレーテッドアニーリング） ---
# 初期経路 (0, 1, 2, ..., N_CITY-1, 0)
current_path = list(range(N_CITY)) + [0]
current_dist = calculate_total_distance(current_path, distance_matrix)

# 記録用
dist_history = [current_dist]
path_history_for_anim = []

# 新しい経路候補の作成方法を選択
# 'two_city_swap' / 'route_reverse' / 'route_reverse_2'
selector = 'route_reverse'

print(f"初期経路の距離: {current_dist:.5f}")
```

初期経路の距離: 33.44894

In [5]:
```
# --- 4b. 初期経路のプロット ---

# `%matplotlib inline` を指定して、静的なプロットをセル内に表示
get_ipython().run_line_magic('matplotlib', 'inline')

# 初期経路の座標リストを作成
initial_path_coords = np.array([positions[city] for city in current_path])

# プロットを作成
plt.figure(figsize=(6, 6))
plt.plot(initial_path_coords[:, 0], initial_path_coords[:, 1], 'o-')
# 開始/終了点を赤丸で強調
plt.plot(initial_path_coords[0, 0], initial_path_coords[0, 1], 'ro', mark

# タイトルとラベル
plt.title(f"Path Distance: {current_dist:.3f}")
plt.xlabel("X coordinate")
plt.ylabel("Y coordinate")
plt.xticks([])
plt.yticks([])
plt.legend()
plt.grid(True)
plt.show()
```



In [6]:
```
selector = 'route_reverse'

for count in range(MAX_COUNT):
    # 新しい経路候補を作成
    new_path = create_new_path(current_path, selector, N_CITY)

    new_dist = calculate_total_distance(new_path, distance_matrix)

    # 経路を更新するかどうかを確率的に決定（メトロポリス法）
    delta_e = new_dist - current_dist
    # Mapleの `exp(-dE/Temp)>evalf(rand()/10^12)` と同じ条件
    if np.exp(-delta_e / TEMP) > random.random():
        current_path = new_path
        current_dist = new_dist

        # 記録
        dist_history.append(current_dist)

        # アニメーション用に経路が更新されるたびに記録する
        path_coords = np.array([positions[city] for city in current_path])
        path_history_for_anim.append((path_coords, current_dist, count))

print(f"計算完了。最終的な最短距離: {current_dist:.5f}")
```

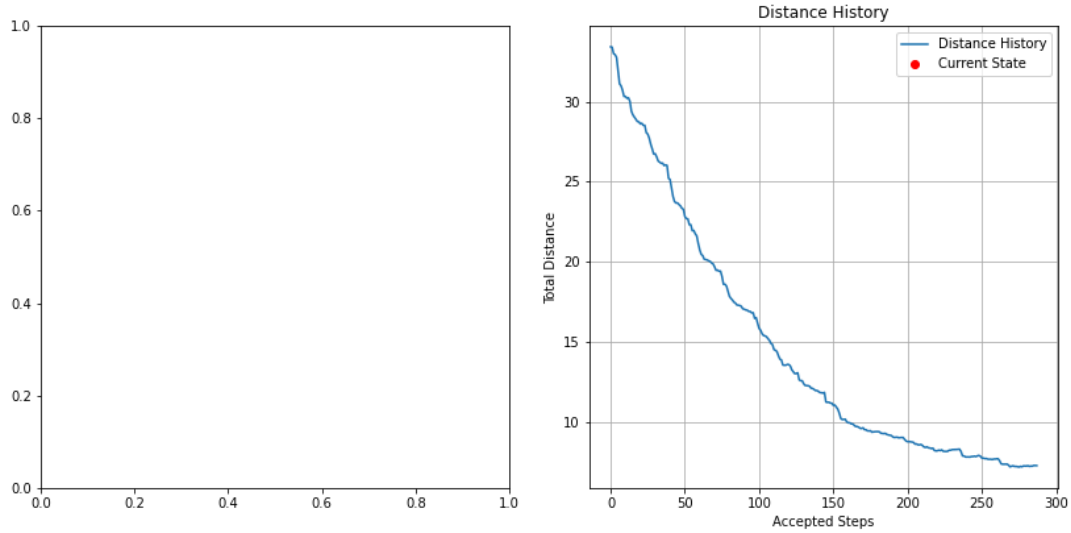
計算完了。最終的な最短距離: 7.25009

In [7]:
```
# %pip install ipympl
```

In [8]:
```
# --- 6. インタラクティブウィジェットとしてアニメーションを表示 ---

# ipymplバックエンドを有効にする
get_ipython().run_line_magic('matplotlib', 'widget')

# 描画領域(Figure)と2つのプロット領域(Axes)を1行2列で作成
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
plt.tight_layout(pad=3.0) # プロット間の間隔を調整
```

```python
# 距離履歴のデータを準備
# アニメーションフレームに対応するイテレーションと距離を抽出
anim_counts = [data[2] for data in path_history_for_anim]
anim_dists = [data[1] for data in path_history_for_anim]

# 2つ目のプロット(距離の履歴)を初期設定
ax2.plot(dist_history, label='Distance History')
# 現在の距離を示す赤丸を初期化(最初は非表示)
current_dist_marker, = ax2.plot([], [], 'ro', label='Current State')
ax2.set_title("Distance History")
ax2.set_xlabel("Accepted Steps")
ax2.set_ylabel("Total Distance")
ax2.legend()
ax2.grid(True)

# アニメーションの各フレームを更新する関数
def update_widget_animation(frame_idx):
    path_coords, dist, count = path_history_for_anim[frame_idx]

    # 1つ目のプロット(経路)を更新
    ax1.clear()
    ax1.plot(path_coords[:, 0], path_coords[:, 1], 'o-')
    ax1.plot(path_coords[0, 0], path_coords[0, 1], 'ro', markersize=10)
    ax1.set_title(f"Iteration: {count}, Distance: {dist:.3f}")
    ax1.set_xticks([])
    ax1.set_yticks([])
    ax1.grid(True)

    # 2つ目のプロットの赤丸を更新
    # frame_idxはdist_historyのインデックスと1対1で対応する(初期値を除く)
    step_index = frame_idx + 1
    current_dist_marker.set_data([step_index], [dist])

    # y軸の範囲を動的に調整
    ax2.relim()
    ax2.autoscale_view()


# FuncAnimationを使ってアニメーションオブジェクトを作成
ani_widget = animation.FuncAnimation(fig,
                                     update_widget_animation,
                                     frames=len(path_history_for_anim),
                                     interval=200,
                                     blit=False,
                                     repeat=False)
```

Figure


Distance History
Distance History
Current State
Total Distance
Accepted Steps
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
30
25
20
15
10
0
50
100
150
200
250
300

In [ ]: