Not Able to get dataset to generated random

In [12]:

```
import pandas as pd
import random

# Create a list of users
users = [f'user_{i}' for i in range(1, 51)]  # 50 users

# Generate random interactions between users
interactions = []
for _ in range(200):  # Generating 200 random interactions
    user1 = random.choice(users)
    user2 = random.choice(users)
    while user1 == user2:  # Avoid self-interactions
        user2 = random.choice(users)
    interactions.append((user1, user2))

# Create a DataFrame
df_interactions = pd.DataFrame(interactions, columns=['user1', 'user2'])

# Save to CSV (for your reference)
df_interactions.to_csv('synthetic_social_network.csv', index=False)

# Show the first few rows of the dataset
df_interactions.head()
```

Out[12]:

	user1	user2
0	user_9	user_35
1	user_34	user_15
2	user_28	user_49
3	user_24	user_43
4	user_37	user_38

Load and Preprocess the Data

In [3]:

```
import pandas as pd
import networkx as nx

# Load the dataset (e.g., Twitter dataset with user interactions)
df = pd.read_csv('synthetic_social_network.csv')

# Inspect the dataset structure
df.head()

# Preprocessing: create edges for user interactions, remove duplicates, e
# Example: Let's say the dataset has 'user1' and 'user2' columns represen
edges = df[['user1', 'user2']].drop_duplicates()
```

```python
# Construct a graph using NetworkX
G = nx.from_pandas_edgelist(edges, 'user1', 'user2')
```

Graph Construction and Representation

In [7]:

```python
# Initialize each node to 'Susceptible' at the start
nx.set_node_attributes(G, 'Susceptible', 'status')

# Optionally, print some statistics about the graph
print(f"Number of nodes: {G.number_of_nodes()}")
print(f"Number of edges: {G.number_of_edges()}")
```

```
Number of nodes: 50
Number of edges: 180
```

Apply Information Diffusion Model

In [9]:

```python
import random

# Set transition probabilities
beta = 0.3  # S -> E
gamma = 0.1  # E -> I
delta = 0.05  # I -> R

# Randomly infect a small set of nodes initially
for node in random.sample(list(G.nodes()), 5):  # Convert G.nodes() to a
    G.nodes[node]['status'] = 'Infected'

def update_status(G):
    new_status = {}
    for node in G.nodes():
        status = G.nodes[node]['status']

        if status == 'Susceptible':
            # Check if any neighbor is infected
            infected_neighbors = [n for n in G.neighbors(node) if G.nodes
            if infected_neighbors and random.random() < beta:
                new_status[node] = 'Exposed'
        elif status == 'Exposed':
            # Transition from Exposed to Infected
            if random.random() < gamma:
                new_status[node] = 'Infected'
        elif status == 'Infected':
            # Transition from Infected to Recovered
            if random.random() < delta:
                new_status[node] = 'Recovered'

    # Update statuses for the next round
    nx.set_node_attributes(G, new_status, 'status')

# Run the simulation for a certain number of time steps
time_steps = 20
for t in range(time_steps):
    update_status(G)
```

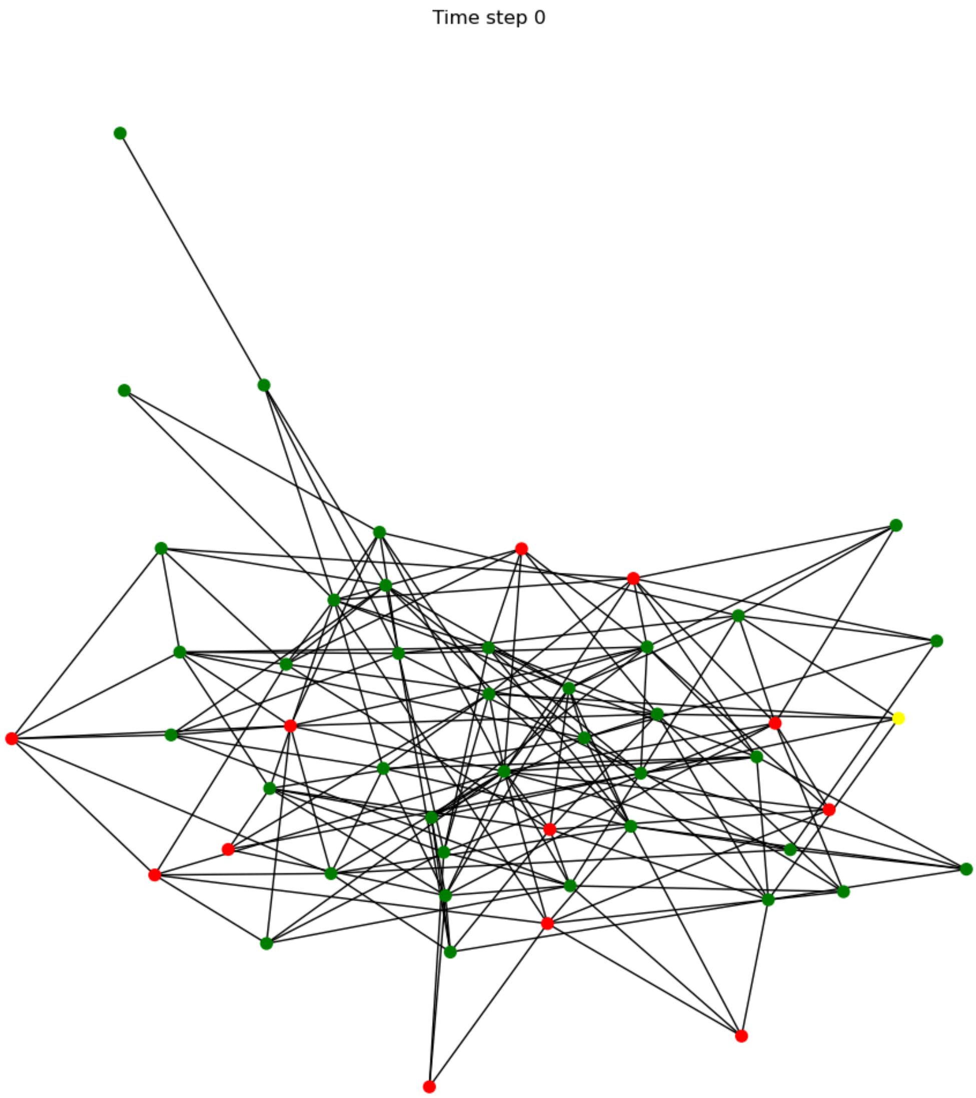
Evaluate the Results

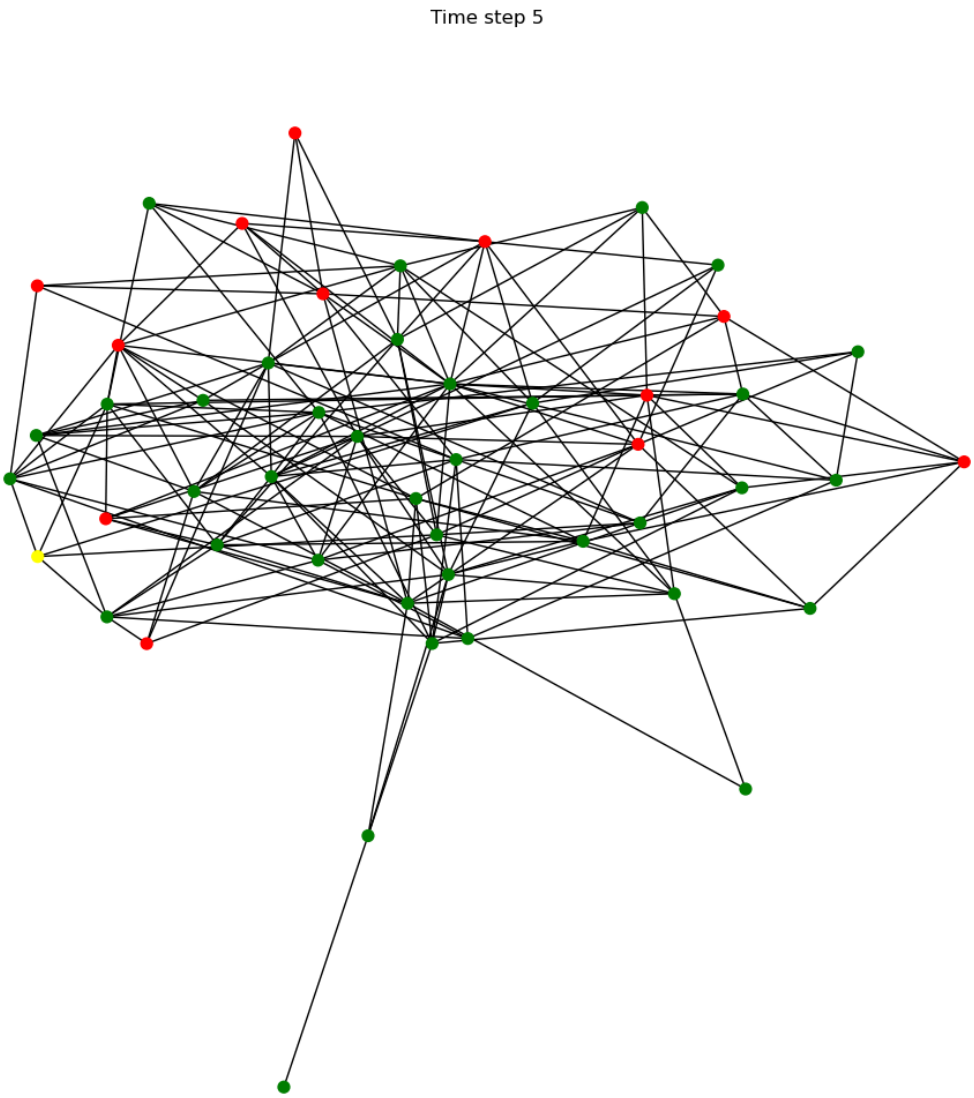
In [10]:
```
def count_states(G):
    counts = {'Susceptible': 0, 'Exposed': 0, 'Infected': 0, 'Recovered':
    for node in G.nodes():
        counts[G.nodes[node]['status']] += 1
    return counts

# Track the spread over time
history = []
for t in range(time_steps):
    update_status(G)
    history.append(count_states(G))

# Print final counts
print("Final counts:", history[-1])
```

```
Final counts: {'Susceptible': 0, 'Exposed': 1, 'Infected': 12, 'Recovere
d': 37}
```

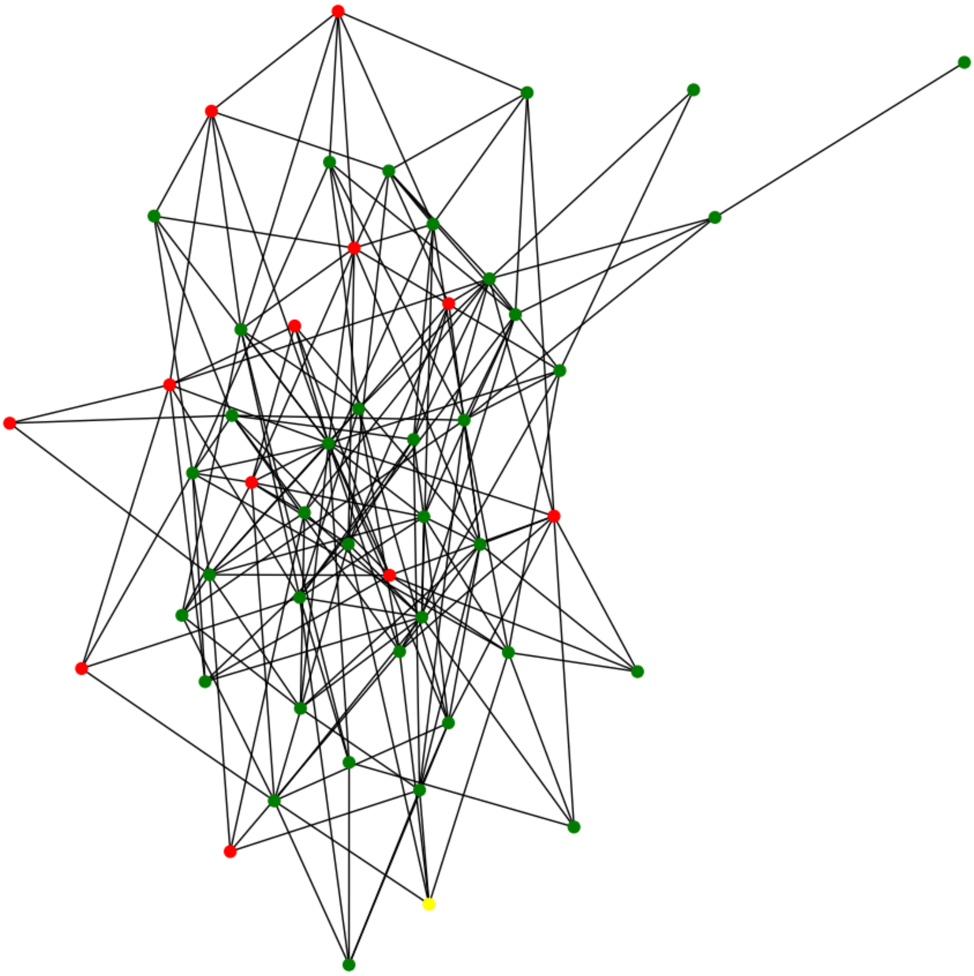
Visualize the Graph

In [11]:
```
import matplotlib.pyplot as plt

def draw_graph(G, t):
    color_map = {
        'Susceptible': 'blue',
        'Exposed': 'yellow',
        'Infected': 'red',
        'Recovered': 'green'
    }

    colors = [color_map[G.nodes[node]['status']] for node in G.nodes()]
    plt.figure(figsize=(10, 10))
    nx.draw(G, node_color=colors, with_labels=False, node_size=50)
    plt.title(f'Time step {t}')
    plt.show()

# Visualize at different time steps
for t in [0, 5, 10, 15, 20]:
    draw_graph(G, t)
```

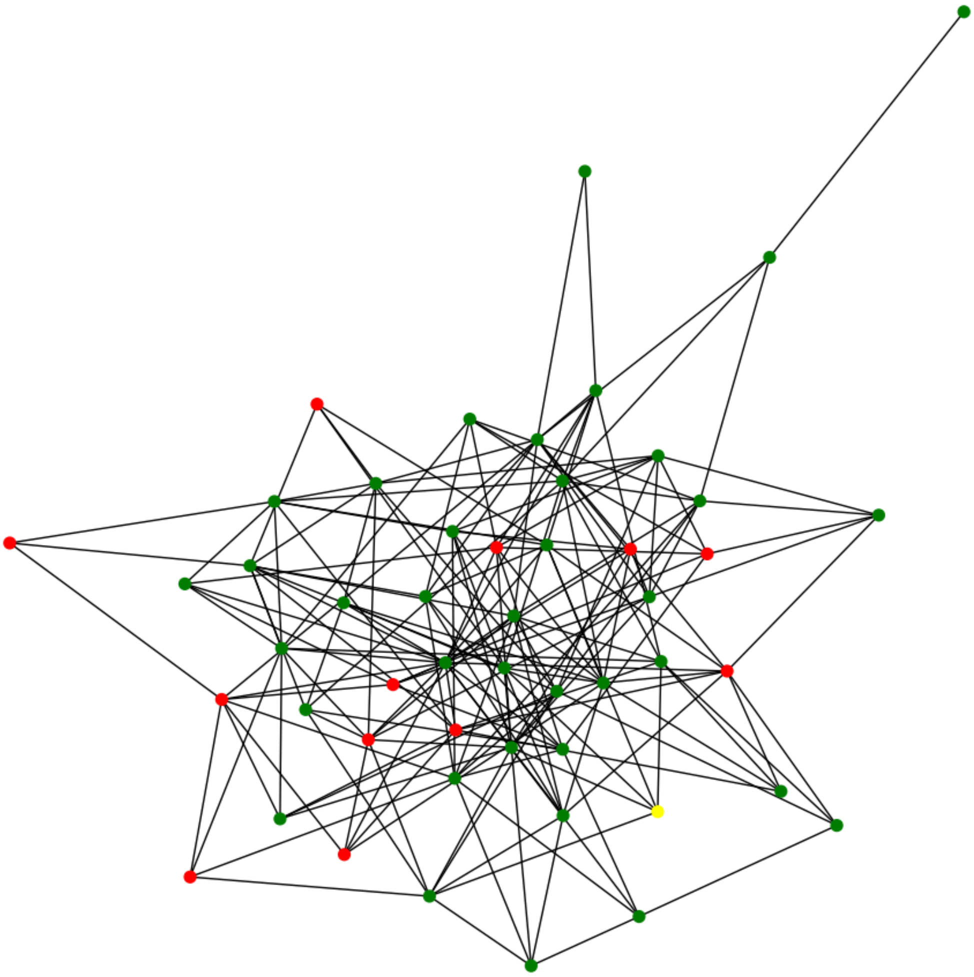

Time step 0


Time step 5

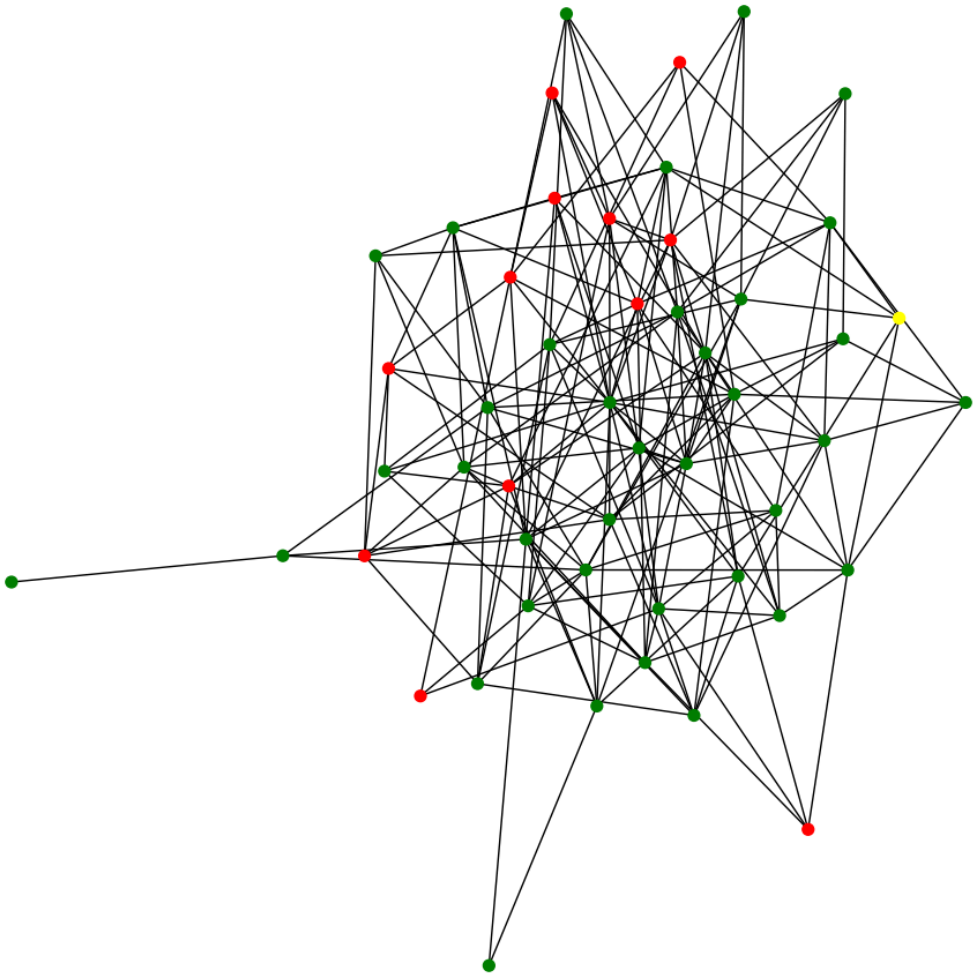
Time step 10



Time step 15



Time step 20



In [ ]: