

```

In [6]: from numpy import random
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

def action(P_dead, P_migra, P_repro):
    """
    Determine the action of a cancerigenous cell based on given probabilities.

    Parameters:
    P_dead (float): Probability of the cell dying.
    P_migra (float): Probability of the cell migrating.
    P_repro (float): Probability of the cell reproducing.

    Returns:
    str: The action taken by the cell ("dead", "migrate", "reproduce", or "stay").
    """

    r = random.random()
    #print(f"Random number: {r}")
    if r < P_dead:
        return "dead"
    elif r < P_dead + P_migra:
        return "migrate"
    elif r < P_dead + P_migra + P_repro:
        return "reproduce"
    else:
        return "stay"

def destiny(free_n):
    """
    Select a random free neighboring position for the cell to move or reproduce into.
    Parameters:
    free_n (list): List of free neighboring positions.
    Returns: The selected free neighboring position or None if there are no free positions.
    """

    N_L = len(free_n)
    if N_L == 0:
        return None

```

```

else:
    idx = random.randint(0, N_L)
    return free_n[idx]

def select_active_cell(active_cells):
    """
    Select a random cancer or necrotic cell from the list of cancer cells.
    Parameters:
    cancer_cells (list): List of cancer cell objects.
    Returns: The selected cancer cell.
    """
    N_C = len(active_cells)
    if N_C == 0:
        return None
    idx = random.randint(0, N_C)
    return active_cells[idx]

def get_free_neighbors(grid, x, y):
    """
    Get a list of free neighboring positions around a given cell in the grid.
    Parameters:
    grid (list of list): The grid representing the environment.
    x (int): The x-coordinate of the cell.
    y (int): The y-coordinate of the cell.
    Returns:
    list: A list of tuples representing the coordinates of free neighboring positions.
    """

    rows = len(grid)
    cols = len(grid[0])
    free_neighbors = []
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            if dx == 0 and dy == 0:
                continue

            nx, ny = x + dx, y + dy
            if 0 <= nx < rows and 0 <= ny < cols:
                if grid[nx][ny] == 0:

```

```

        free_neighbors.append((nx, ny))

    return free_neighbors

def adjusted_probability(P_base, N_L, alpha):
    """
    Adjust the base probabilities of cell actions based on the availability of free neighboring positions.
    Parameters:
    P_base (dict): Dictionary containing base probabilities for 'dead', 'migrate', 'reproduce', and 'stay'.
    N_L (int): Number of free neighboring positions.
    alpha (float): Fraction of blocked action probability to be added to 'dead' action.
    Returns:
    dict: Adjusted probabilities for cell actions.
    """
    P_adjust = P_base.copy()

    if N_L > 0:
        return P_adjust

    P_bloq = P_base['migrate'] + P_base['reproduce']

    P_adjust['migrate'] = 0
    P_adjust['reproduce'] = 0
    P_adjust['dead'] = P_adjust['dead'] + alpha * P_bloq
    P_adjust['stay'] = P_adjust['stay'] + (1 - alpha) * P_bloq

    return P_adjust

def evolve_necrotic_cell(grid, x, y, P_clean, P_damage, P_necro_dead, active_cell_set):
    """
    Evolve the state of a necrotic cell based on given probabilities.
    Parameters:
    grid (list of list): The grid representing the environment.
    x (int): The x-coordinate of the necrotic cell.
    y (int): The y-coordinate of the necrotic cell.
    P_clean (float): Probability of the necrotic cell being cleaned.
    P_damage (float): Probability of the necrotic cell damaging neighboring cells.
    P_necro_dead (float): Probability of a neighboring cell becoming necrotic or dead.
    Returns:
    """

```

list of list: The updated grid after evolving the necrotic cell.

"""

```
rows = len(grid)
cols = len(grid[0])
```

```
if random.random() < P_clean:
    grid[x][y] = 0
    active_cell_set.remove((x, y))
    return grid
```

```
if random.random() < P_damage:
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            if dx == 0 and dy == 0:
                continue

            nx, ny = x + dx, y + dy

            if 0 <= nx < rows and 0 <= ny < cols:
                neighbour_state = grid[nx][ny]

                if neighbour_state == 1:
                    if random.random() < P_necro_dead:
                        grid[nx][ny] = 2
```

```
return grid
```

```
def initialize_grid(rows, cols):
```

"""

Initialize the grid with given dimensions and place a single cancer cell in the center.

Parameters:

rows (int): Number of rows in the grid.

cols (int): Number of columns in the grid.

Returns:

list of list: The initialized grid.

"""

```

grid = [[0 for _ in range(cols)] for _ in range(rows)]
center_x, center_y = rows // 2, cols // 2
grid[center_x][center_y] = 1
return grid

```

```

def plot_grid(grid, title="Simulación de Células", t_step=None):
    """
    Genera una visualización de la malla con colores específicos:
    Blanco: Sana (0)
    Rojo: Cancerosa (1)
    Negro: Necrótica (2)
    """

    # 1. Convertir la Lista de Listas a un array de NumPy (necesario para matplotlib)
    import numpy as np
    grid_np = np.array(grid)

    # 2. Definir los colores y su mapeo
    cmap = mcolors.ListedColormap(['white', 'red', 'black'])
    bounds = [0, 1, 2, 3] # Límites para (0, 1, 2)
    norm = mcolors.BoundaryNorm(bounds, cmap.N)

    # 3. Crear la figura
    fig, ax = plt.subplots(figsize=(8, 8))

    # 4. Mostrar la imagen (CORREGIDO: sin vmin/vmax)
    im = ax.imshow(grid_np, cmap=cmap, norm=norm, origin='upper',
                    interpolation='nearest') # <-- ¡AQUÍ ESTÁ EL CAMBIO!

    # 5. Configurar la cuadrícula (opcional)
    ax.set_xticks(np.arange(-0.5, grid_np.shape[1], 1), minor=True)
    ax.set_yticks(np.arange(-0.5, grid_np.shape[0], 1), minor=True)
    ax.grid(which='minor', color='gray', linestyle='-', linewidth=0.5)
    ax.set_xticks([])
    ax.set_yticks([])

    # 6. Añadir Leyenda de color
    cbar = plt.colorbar(im, ax=ax, cmap=cmap, norm=norm, boundaries=bounds, ticks=[0.5, 1.5, 2.5], pad=0.02)
    cbar.set_ticklabels(['Sana (0)', 'Cancerosa (1)', 'Necrótica (2)'])

```

```
# 7. Título
if t_step is not None:
    ax.set_title(f"{title} (Paso de Tiempo Final: {t_step})")
else:
    ax.set_title(title)

# 8. Mostrar La gráfica
plt.show()
```

```
In [11]: ROWS = 10
COLS = 10
Time_Units = 50

P_base = {
    'dead': 0.1,
    'migrate': 0.4,
    'reproduce': 0.4,
    'stay': 0.1
}

alpha = 0.5

P_clean = 0.5
P_damage = 0.05
P_necro_dead = 0.1
```

```

grid = initialize_grid(ROWS, COLS)

active_cell_set = {(ROWS // 2, COLS // 2)} ## Initial active cell is the center cancer cell

for t in range(Time_Units):

    N_active = len(active_cell_set)
    if N_active == 0:
        print("All cells have died.")
        break

    active_cell_list = list(active_cell_set)

    for _ in range(N_active):

        select_coords = select_active_cell(active_cell_list)

        if select_coords is None:
            continue

        if select_coords not in active_cell_set: ## If the cell has already changed state, skip it
            continue

        x, y = select_coords

        cell_state = grid[x][y]

        if cell_state == 1: # Cancer cell

            ## Get free neighbors
            free_neighbors = get_free_neighbors(grid, x, y)
            N_L = len(free_neighbors)

            ## Adjust probabilities
            P_adjust = adjusted_probability(P_base, N_L, alpha)

            ## Determine action
            act = action(P_adjust['dead'], P_adjust['migrate'], P_adjust['reproduce'])

```

```

if act == "dead":
    if N_L: ## If there are free neighbors
        grid[x][y] = 0 # Dies and becomes free
        active_cell_set.remove((x, y))
    else: ## No free neighbors
        grid[x][y] = 2 # Becomes necrotic

elif act == "migrate":
    dest = destiny(free_neighbors)
    if dest:
        grid[dest[0]][dest[1]] = 1 # Move to new position
        active_cell_set.add(dest)
        grid[x][y] = 0 # Original position becomes free
        active_cell_set.remove((x, y))

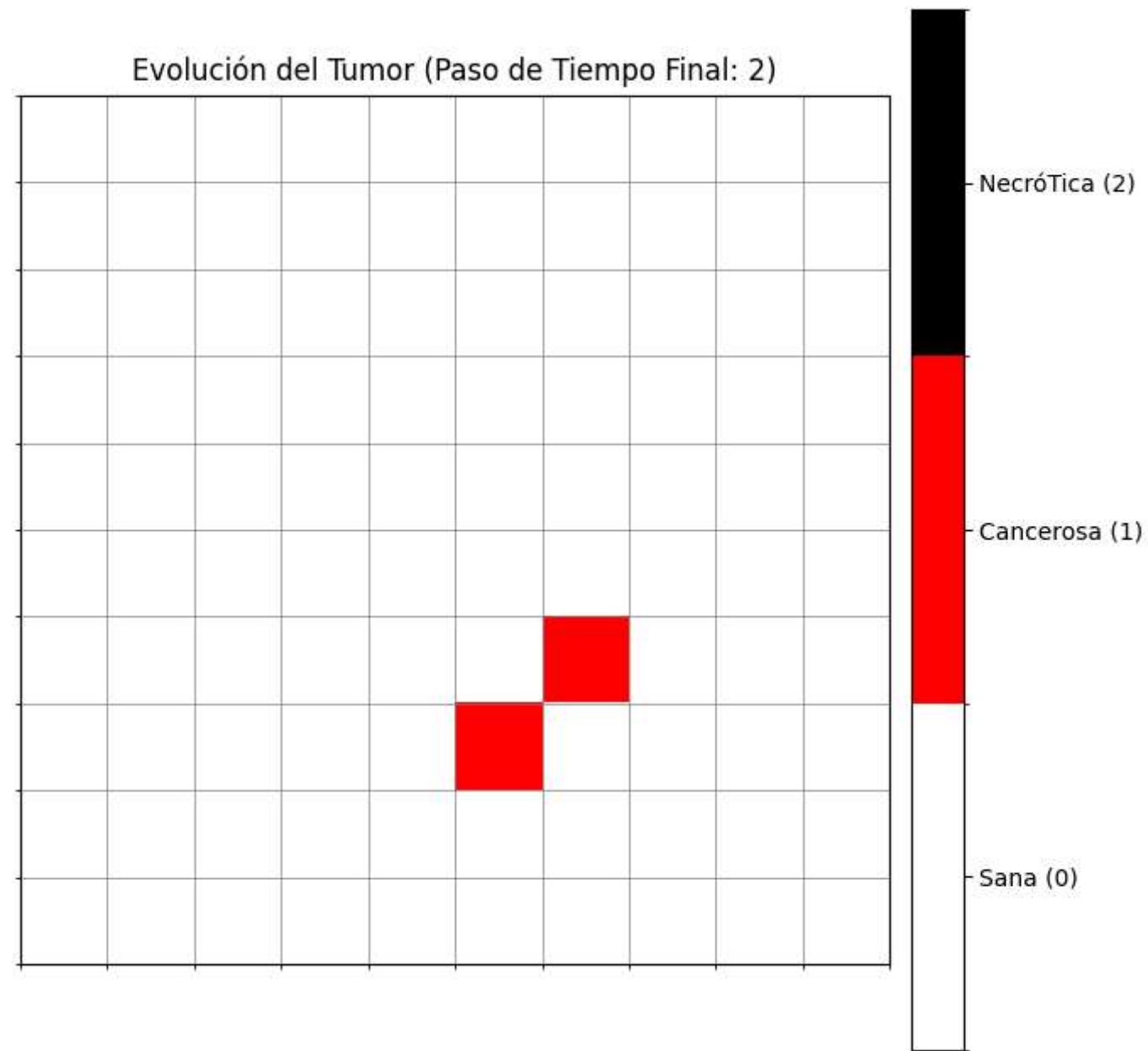
elif act == "reproduce":
    dest = destiny(free_neighbors)
    if dest:
        grid[dest[0]][dest[1]] = 1 # New cancer cell at the destination
        active_cell_set.add(dest)
    # If action is "stay", do nothing

elif cell_state == 2: # Necrotic cell
    grid = evolve_necrotic_cell(grid, x, y, P_clean, P_damage, P_necro_dead, active_cell_set)
if (t + 1) % 2 == 0:
    print(f"Generando imagen en el paso de tiempo: {t + 1}")
    plot_grid(grid, title="Evolución del Tumor", t_step=t + 1)

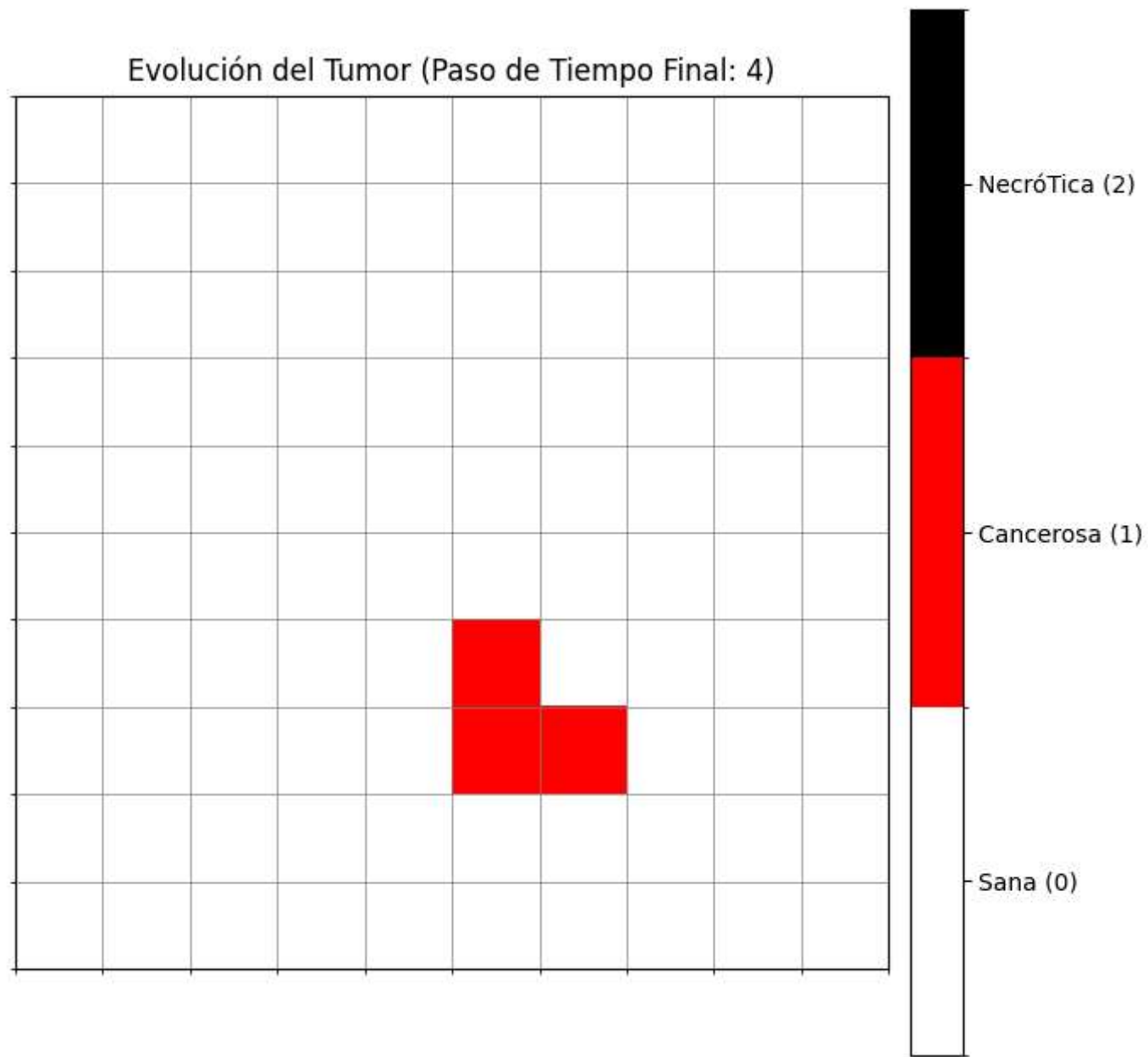
print("Simulación terminada. Generando imagen final...")
plot_grid(grid, title="Estado Final de la Simulación", t_step=Time_Units)

```

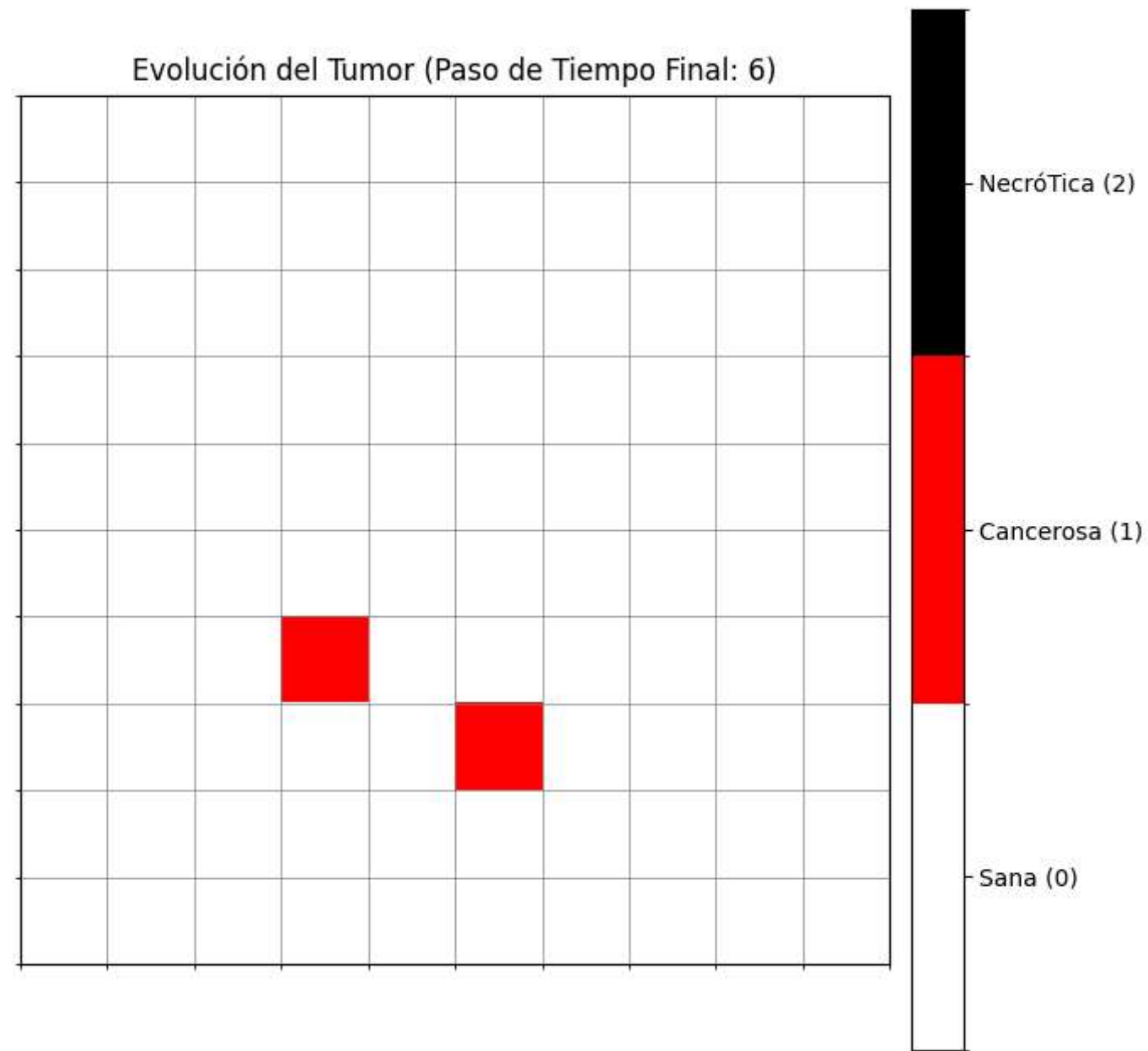
Generando imagen en el paso de tiempo: 2



Generando imagen en el paso de tiempo: 4

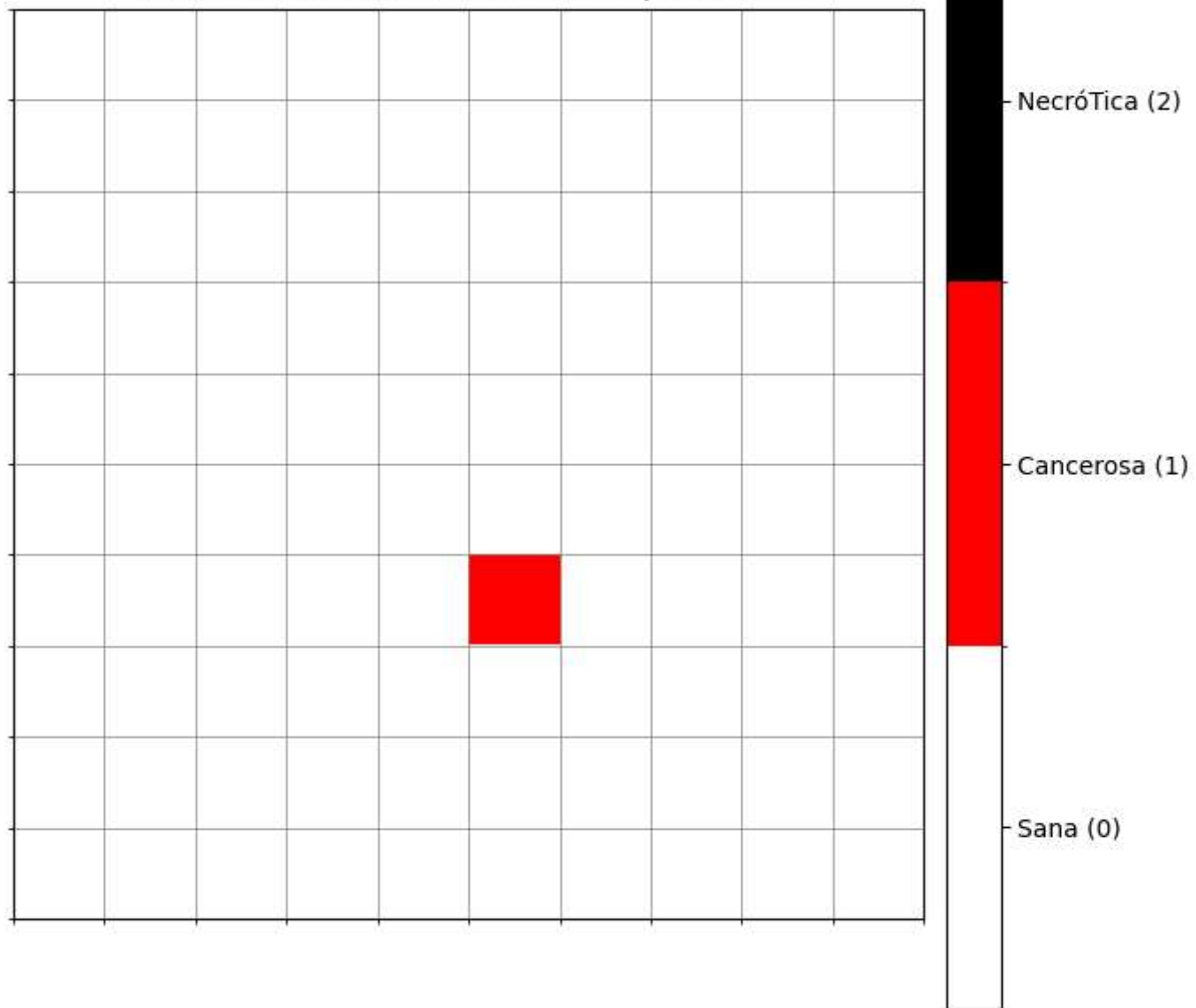


Generando imagen en el paso de tiempo: 6

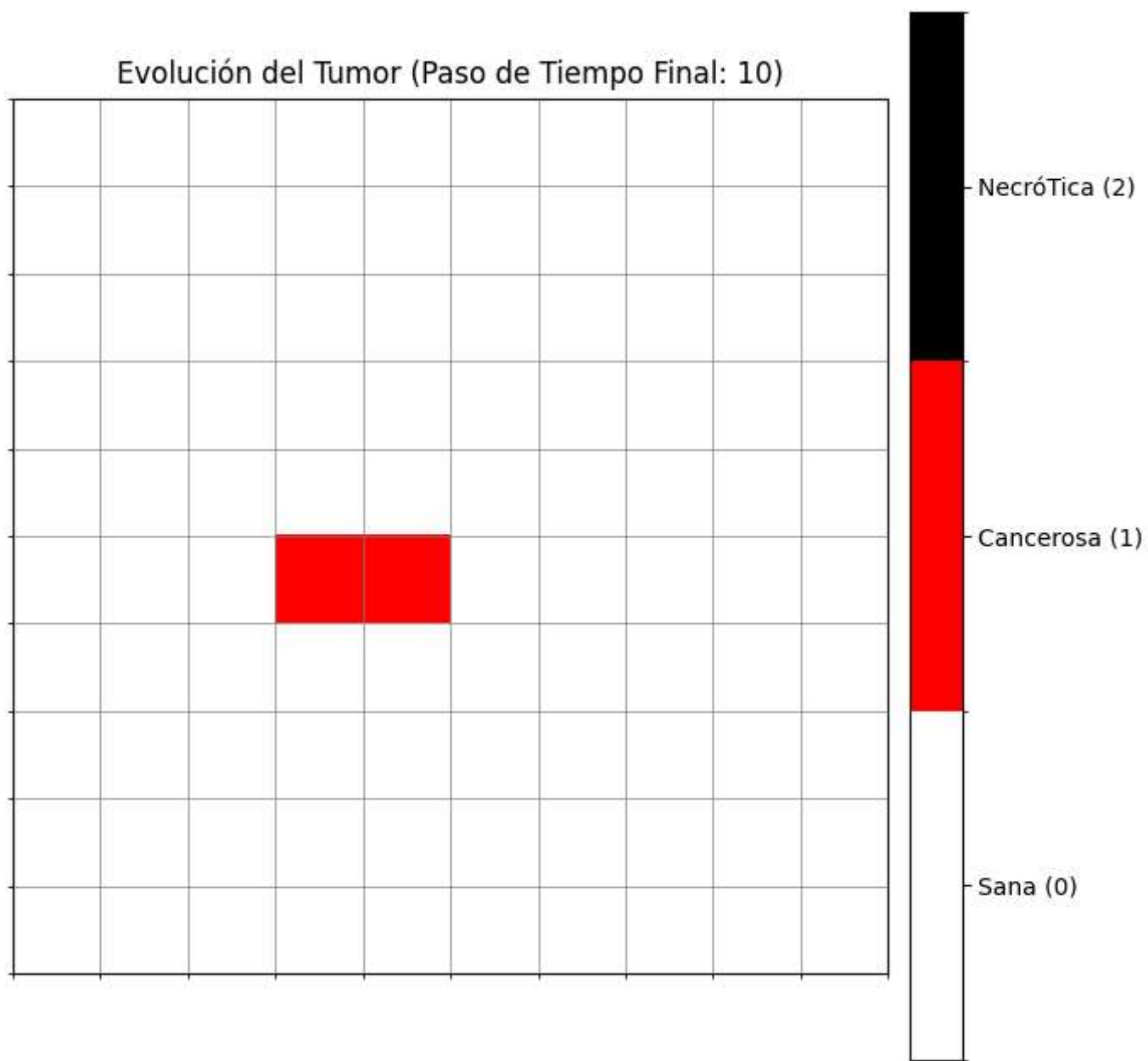


Generando imagen en el paso de tiempo: 8

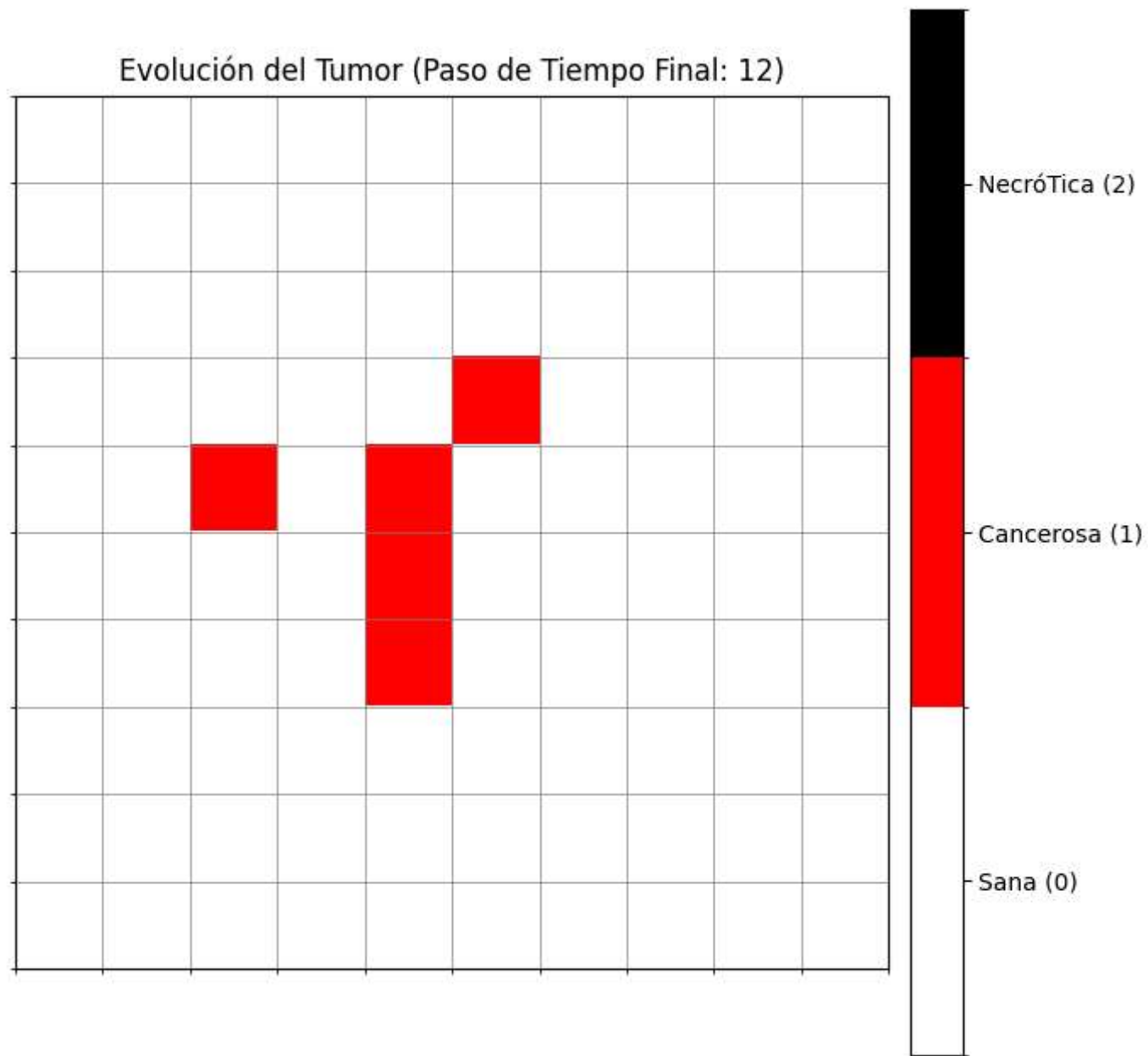
Evolución del Tumor (Paso de Tiempo Final: 8)



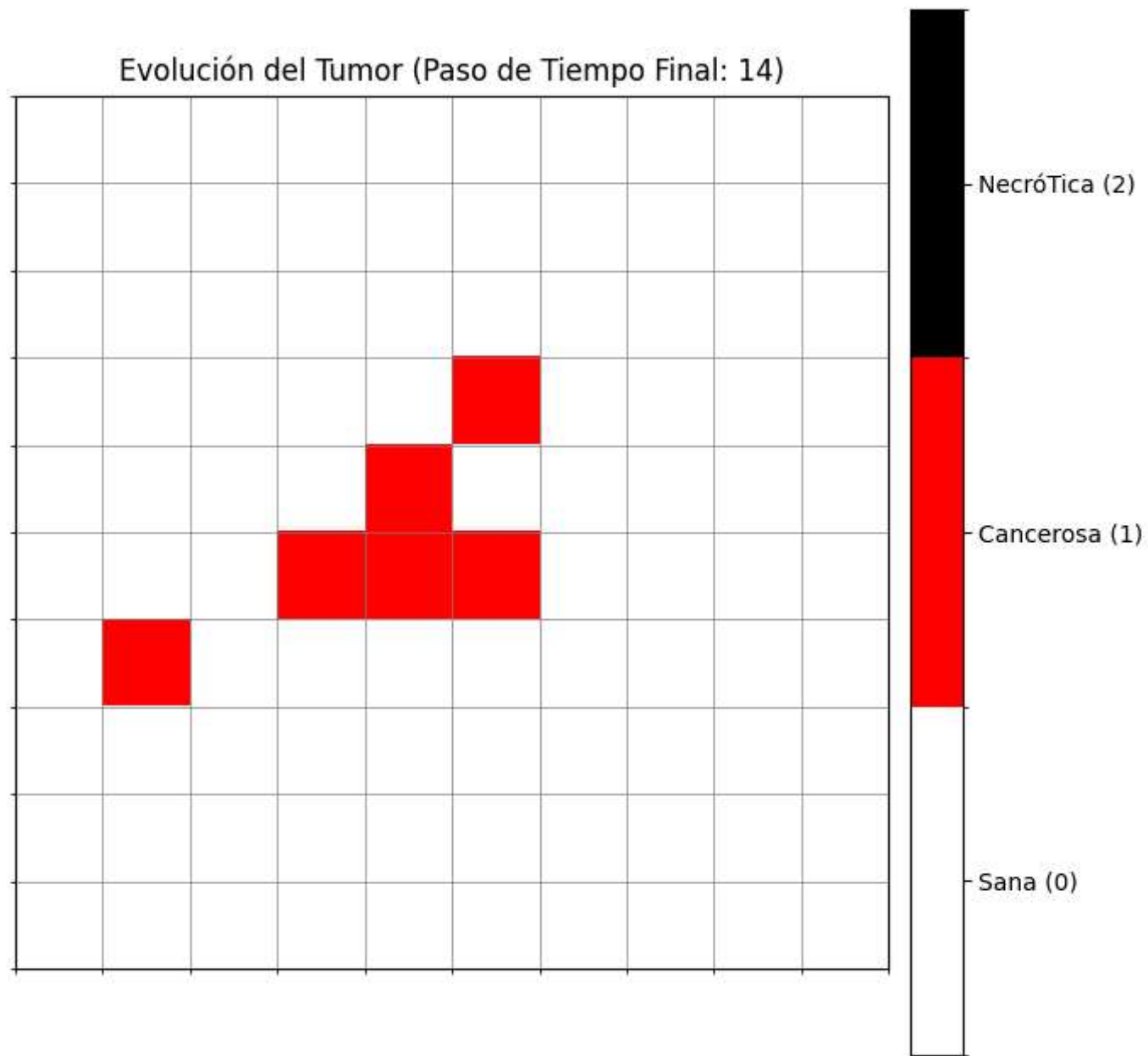
Generando imagen en el paso de tiempo: 10



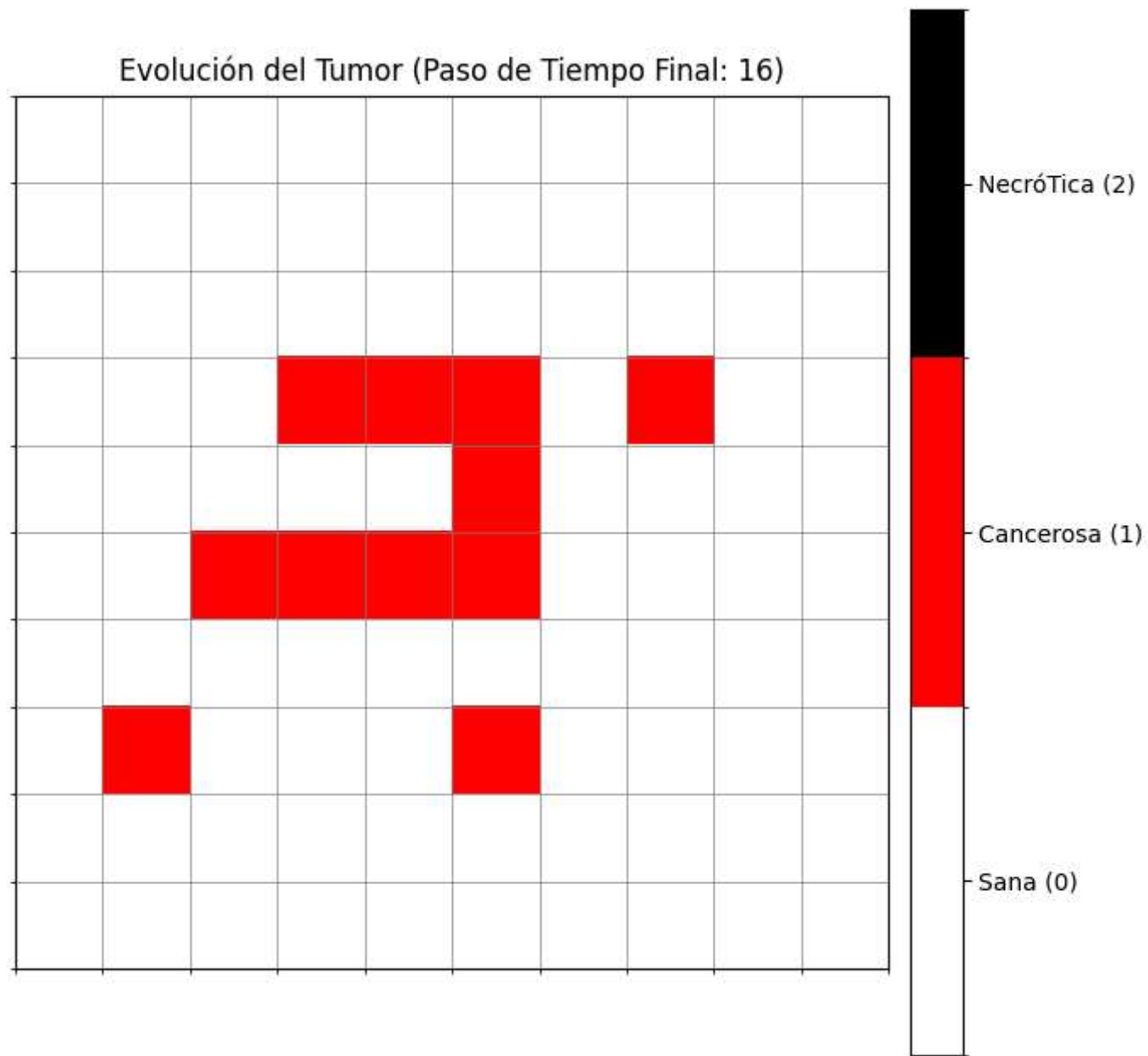
Generando imagen en el paso de tiempo: 12



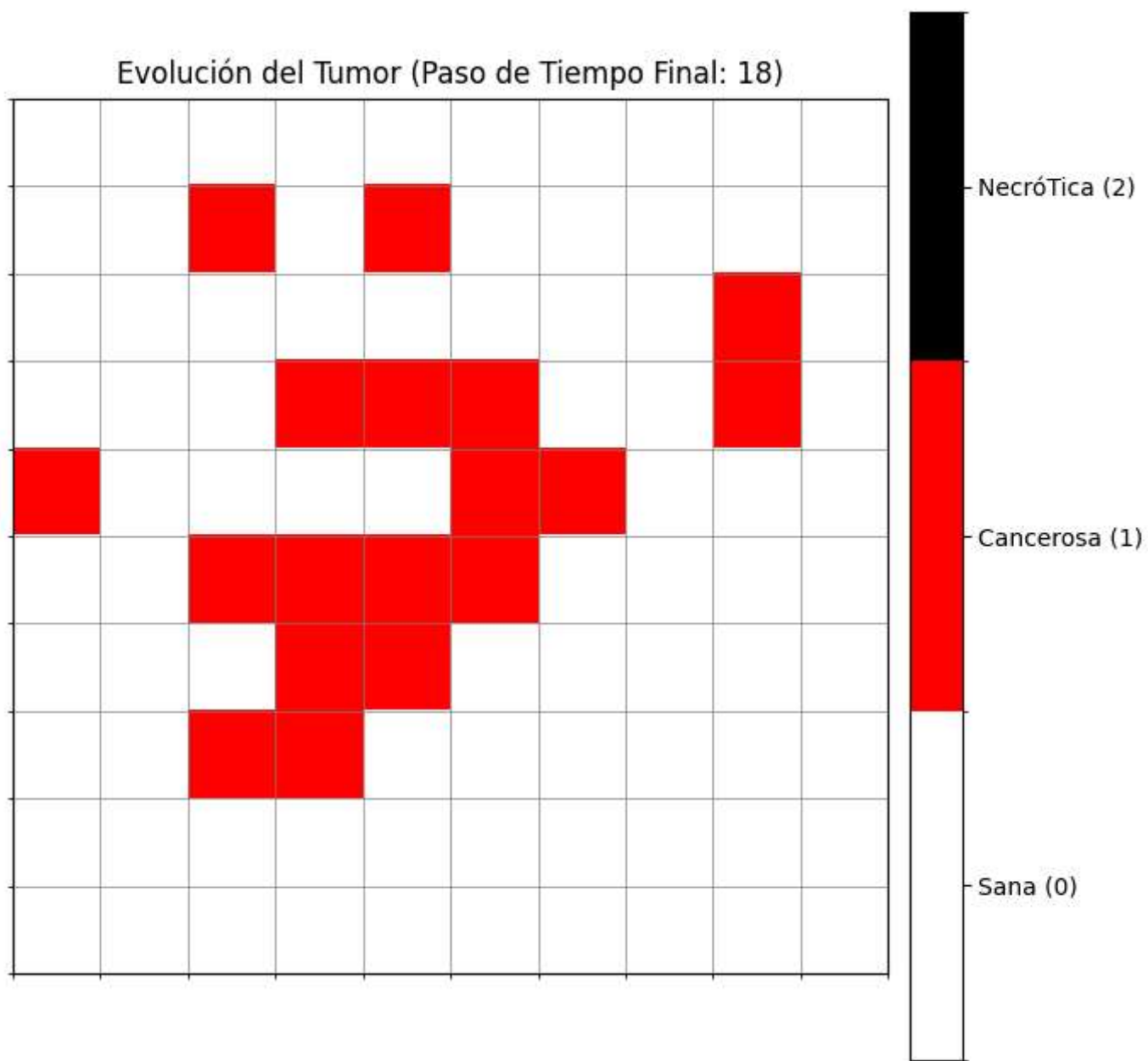
Generando imagen en el paso de tiempo: 14



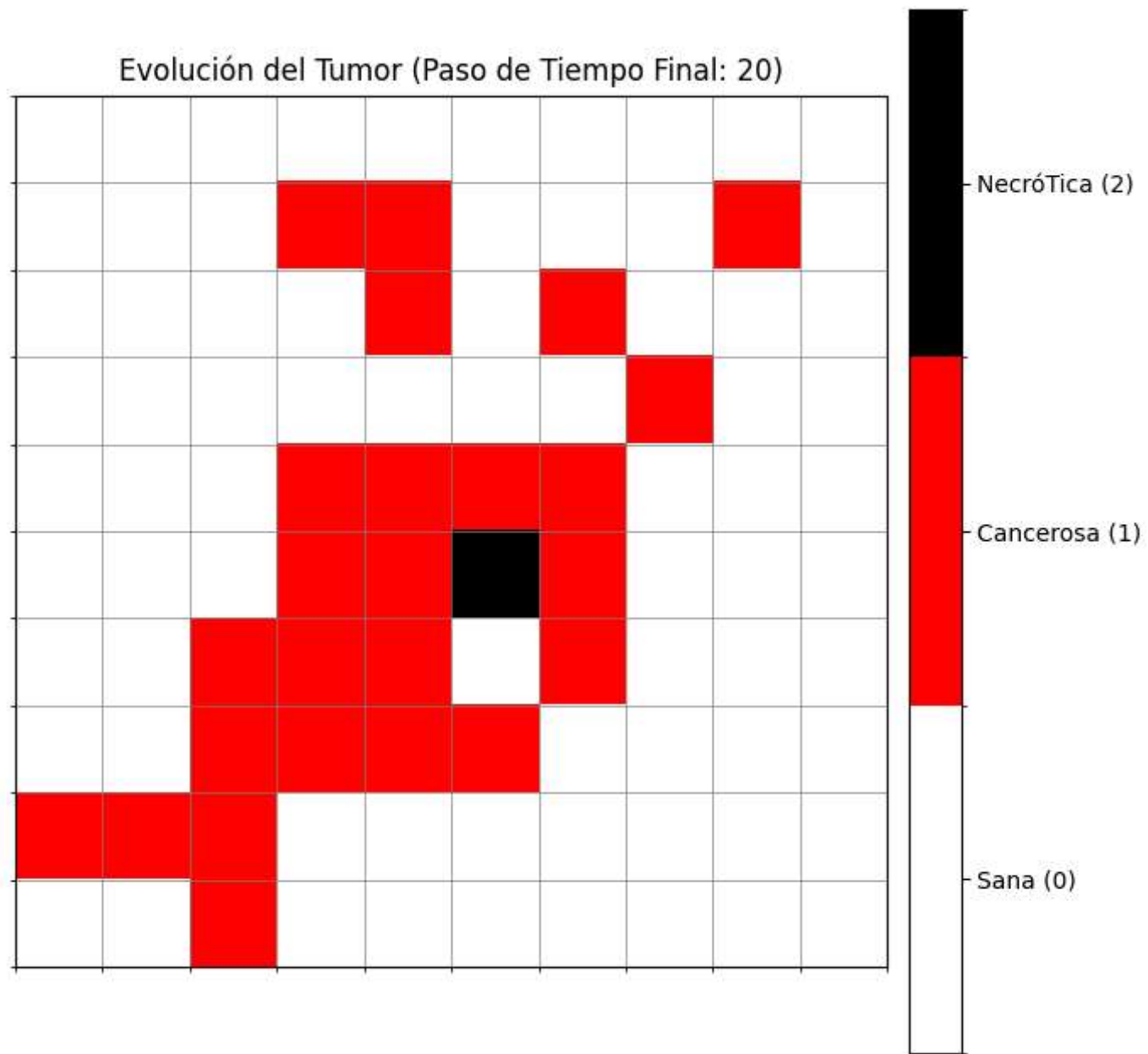
Generando imagen en el paso de tiempo: 16



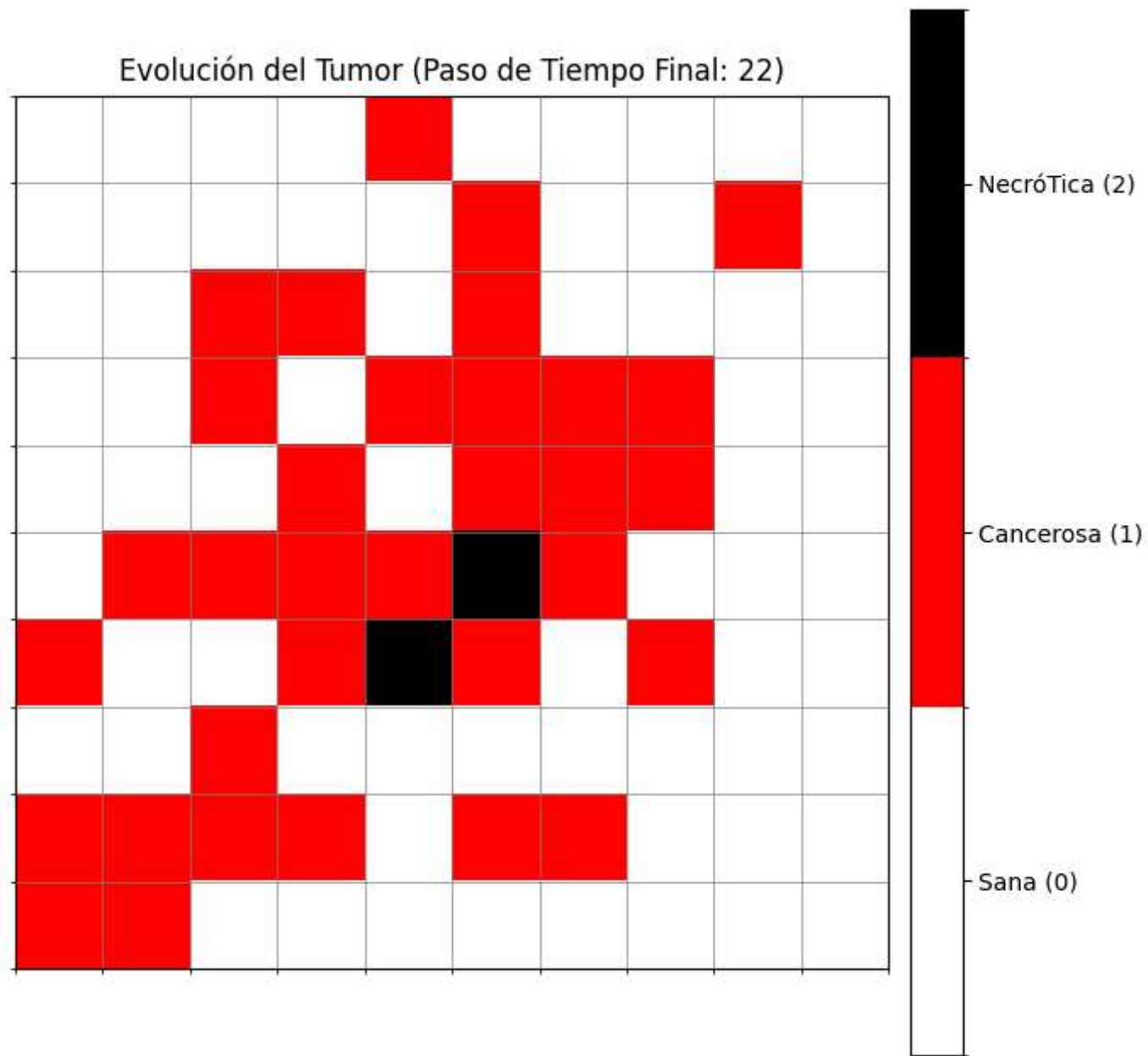
Generando imagen en el paso de tiempo: 18



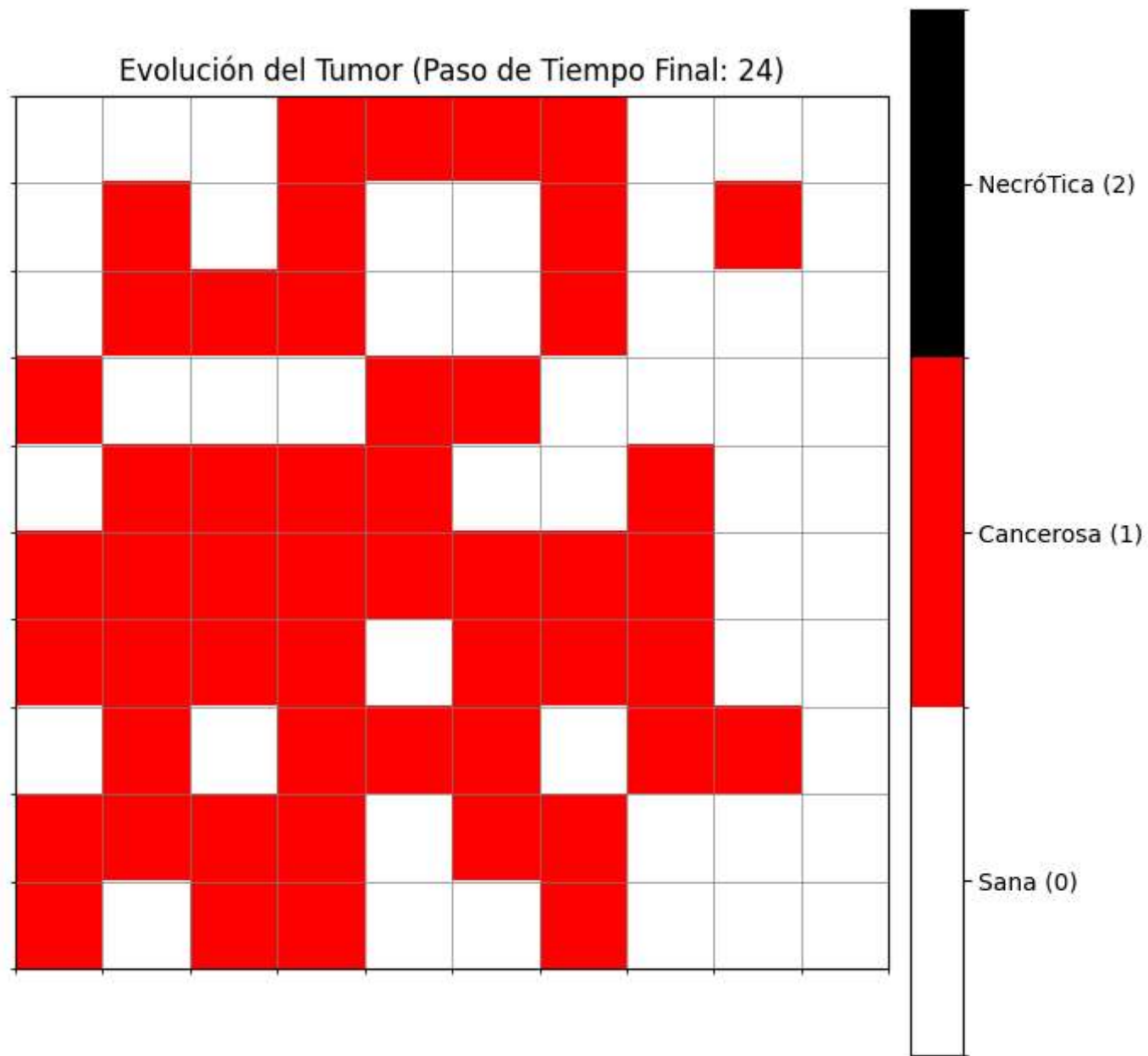
Generando imagen en el paso de tiempo: 20



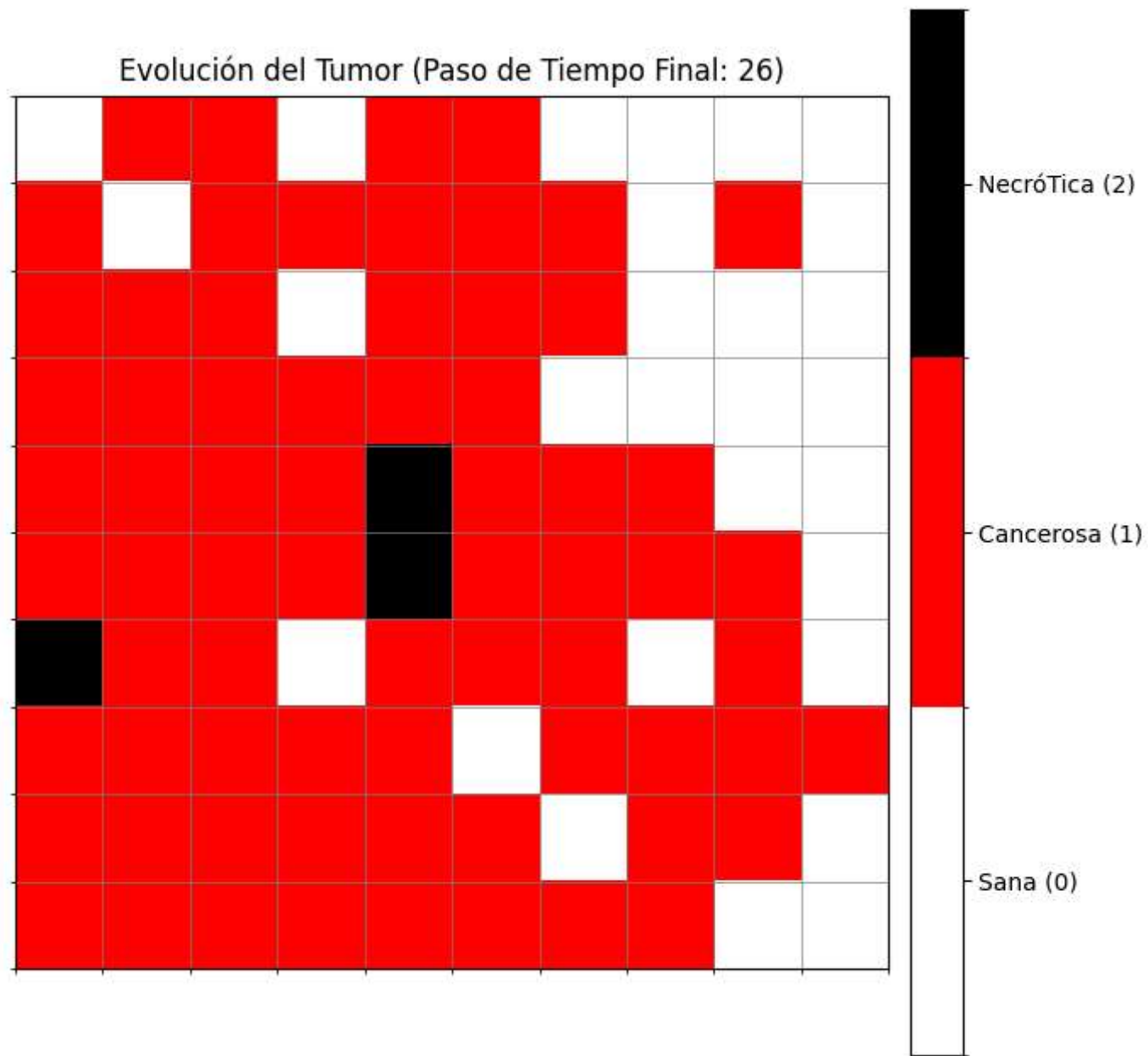
Generando imagen en el paso de tiempo: 22



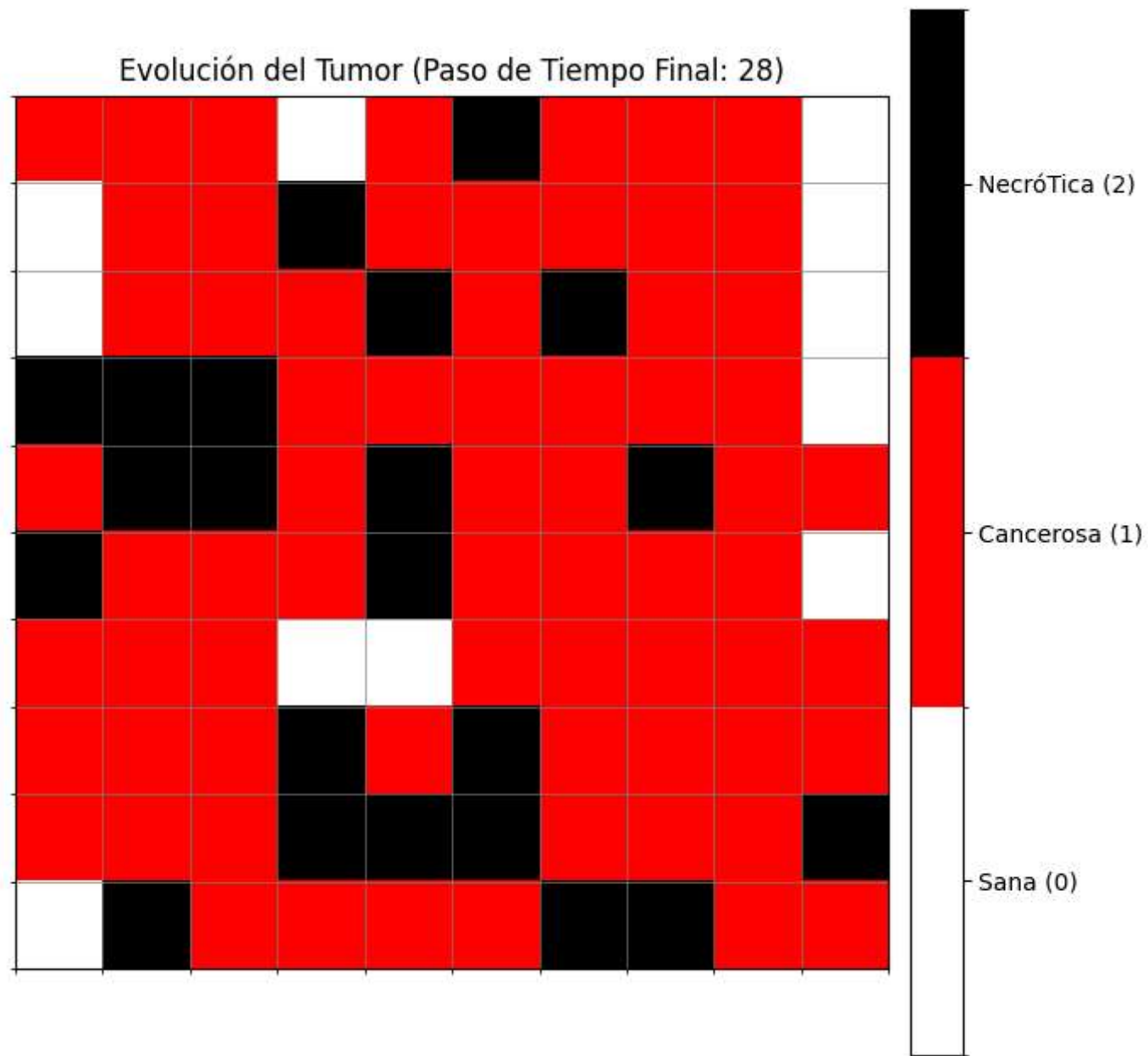
Generando imagen en el paso de tiempo: 24

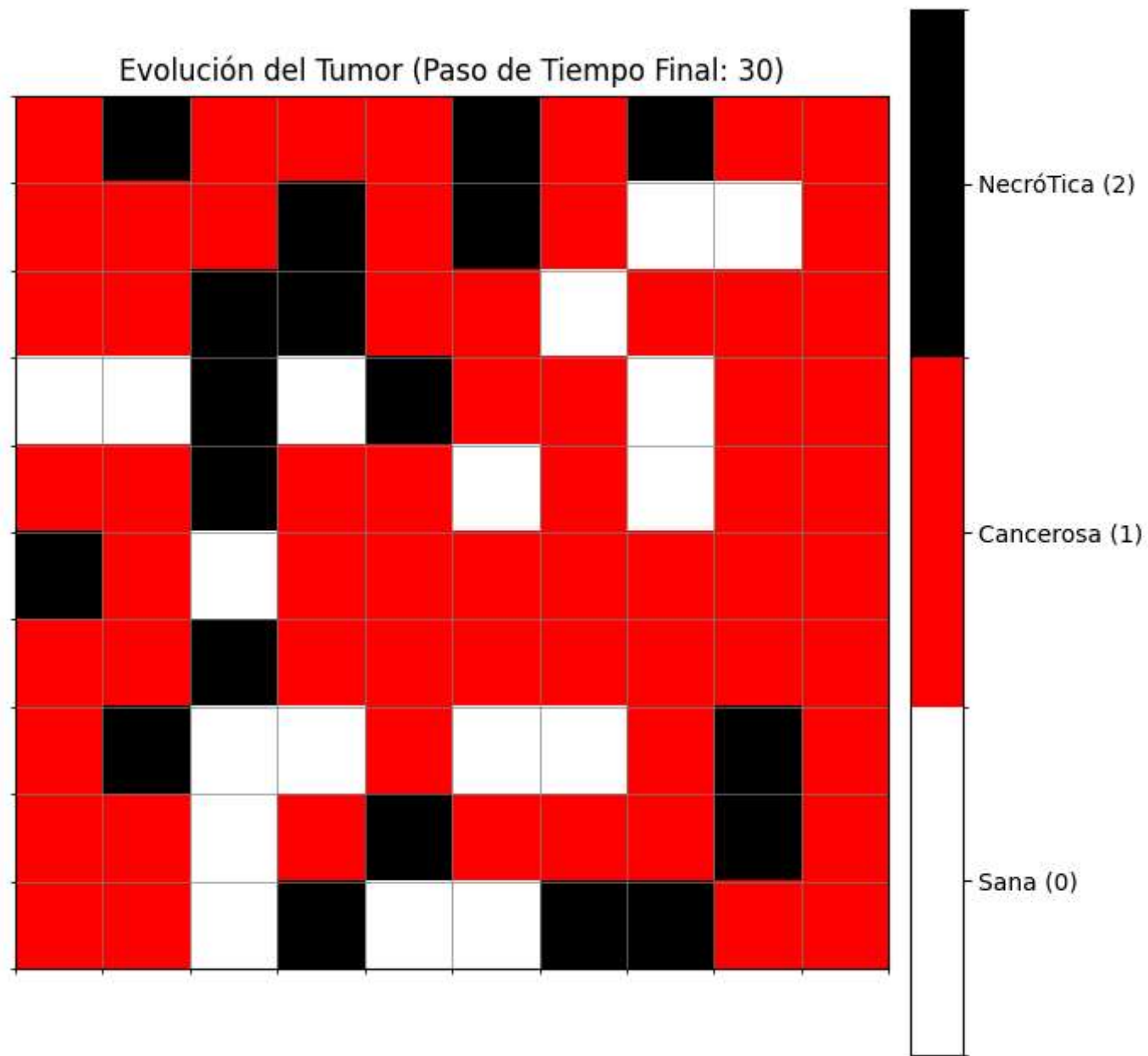


Generando imagen en el paso de tiempo: 26

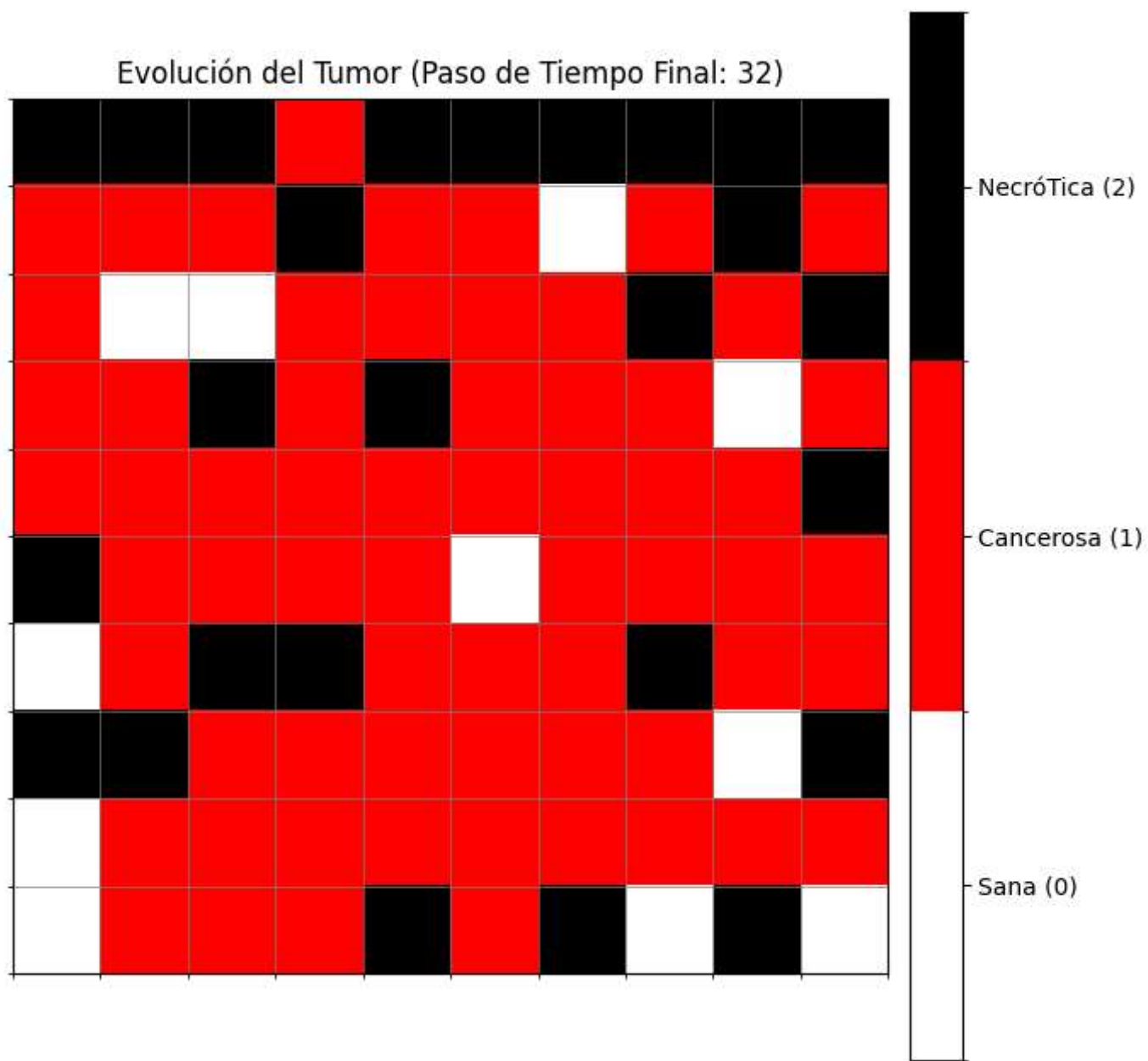


Generando imagen en el paso de tiempo: 28

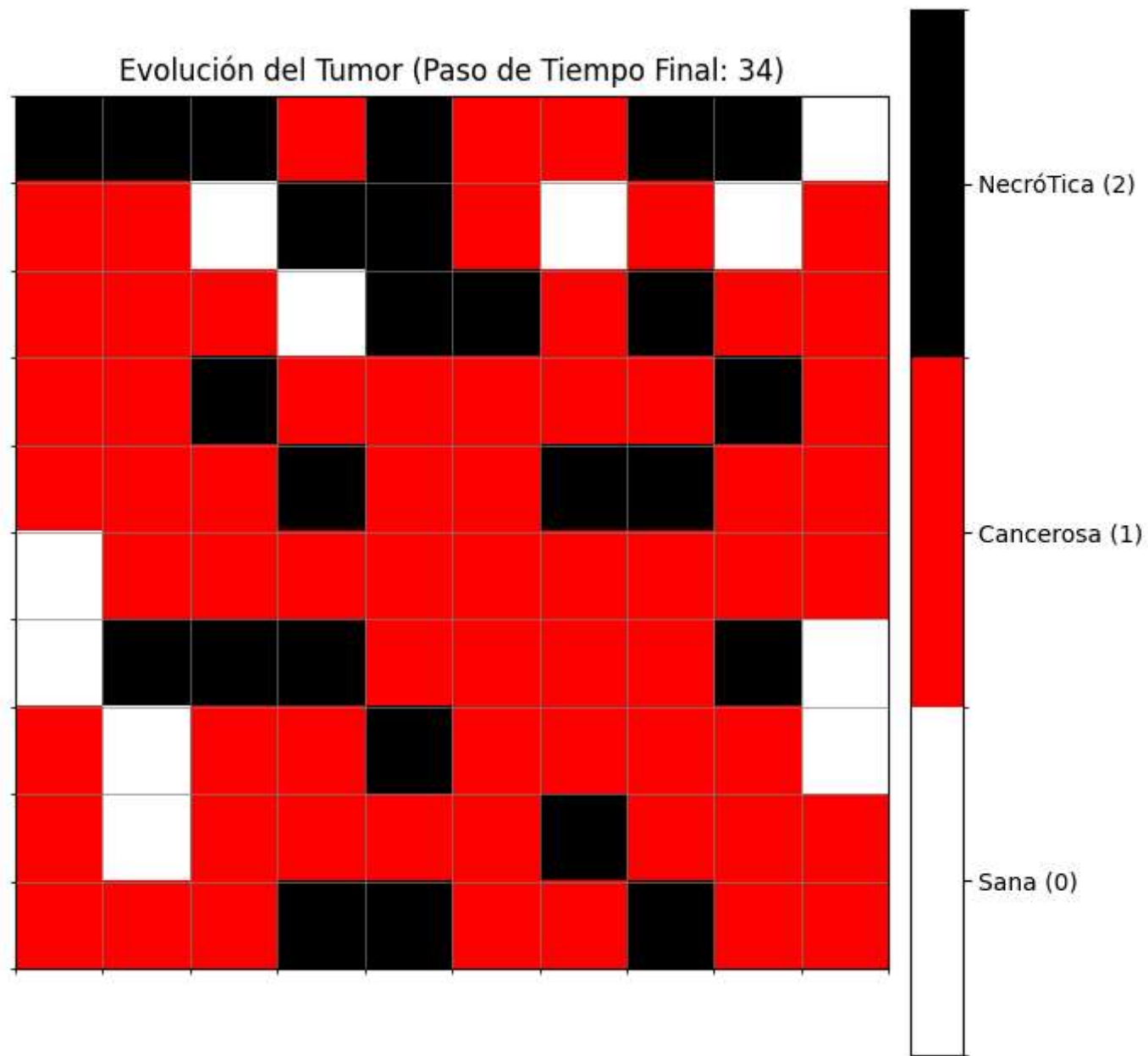




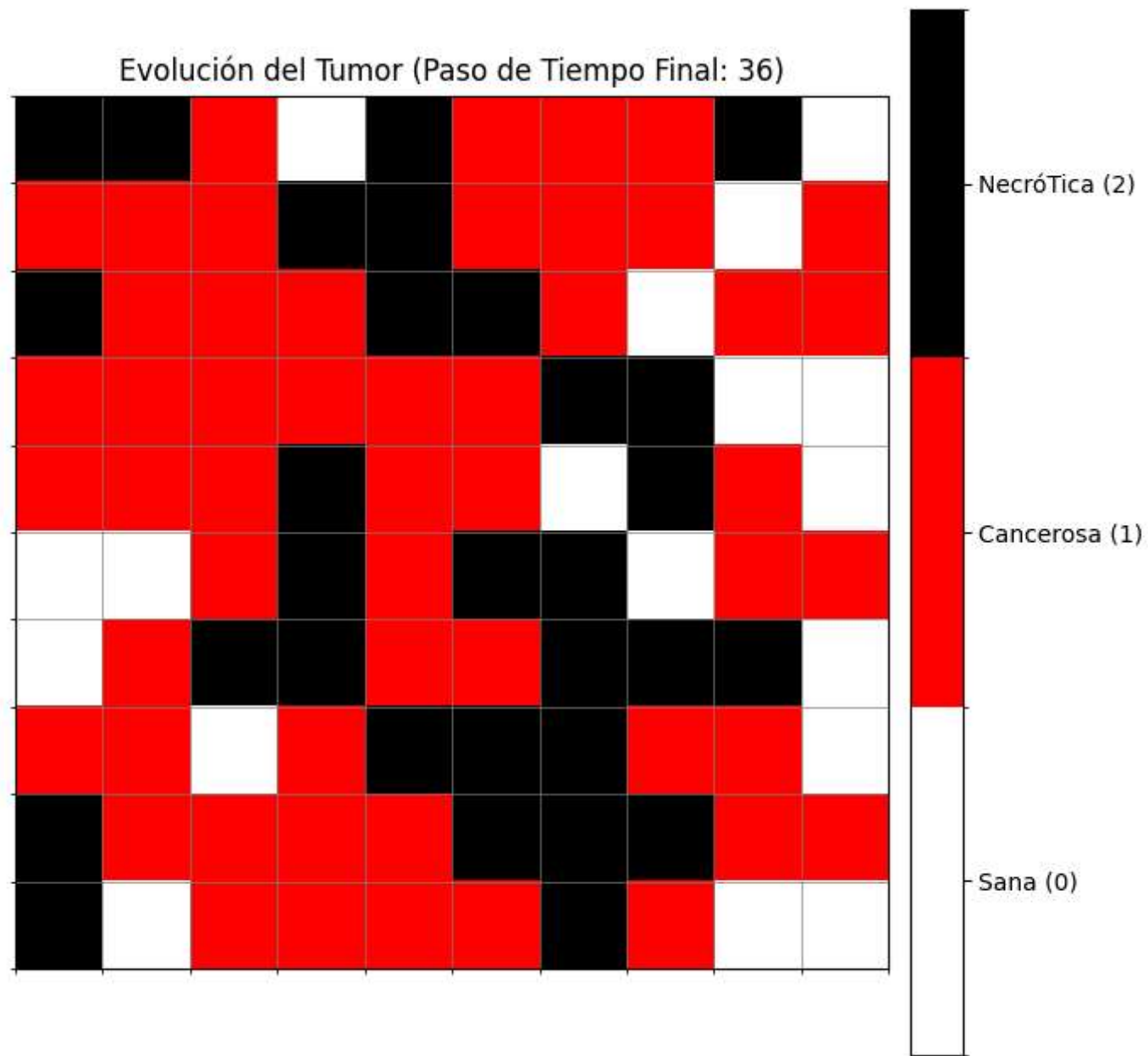
Generando imagen en el paso de tiempo: 32



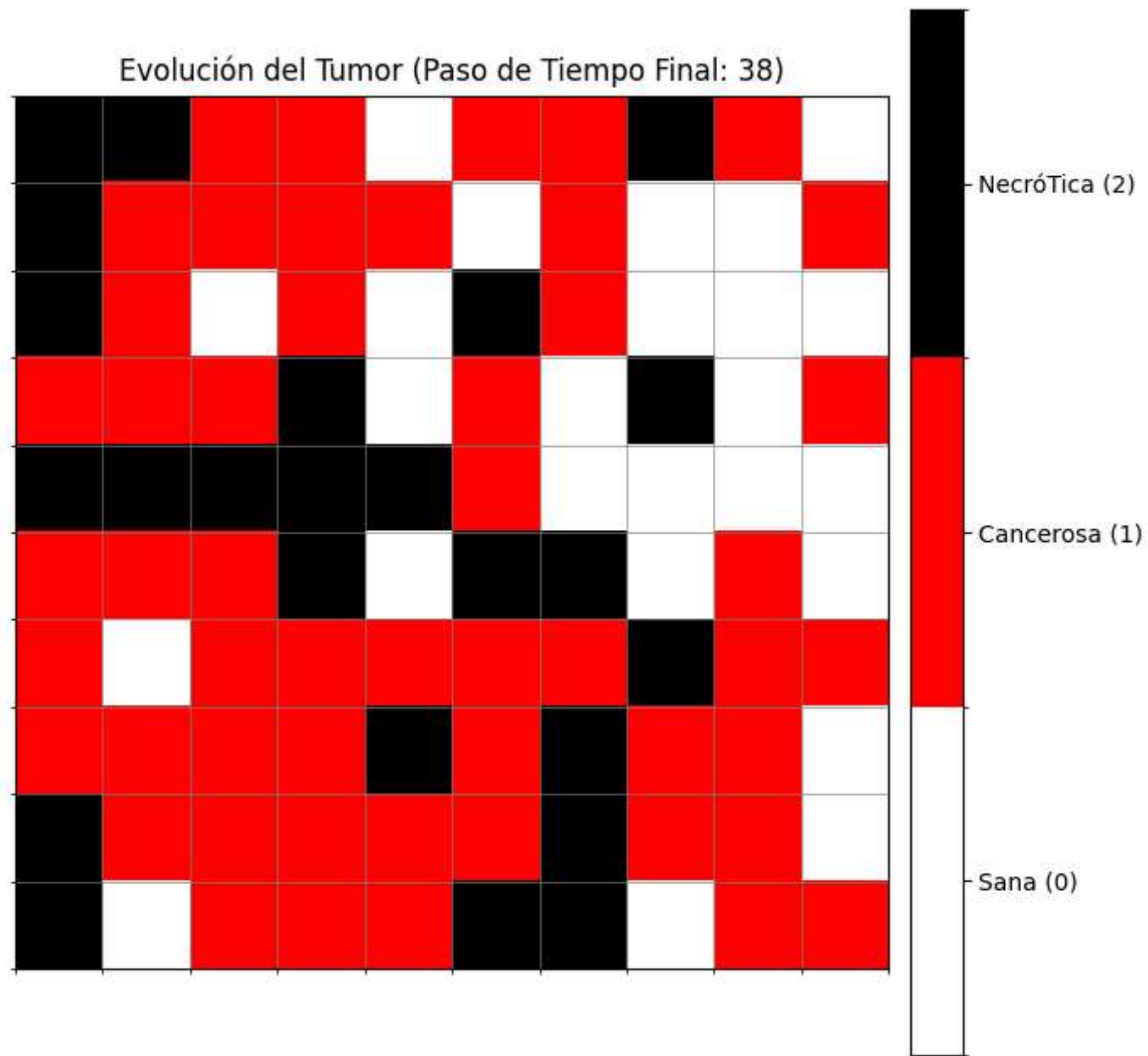
Generando imagen en el paso de tiempo: 34



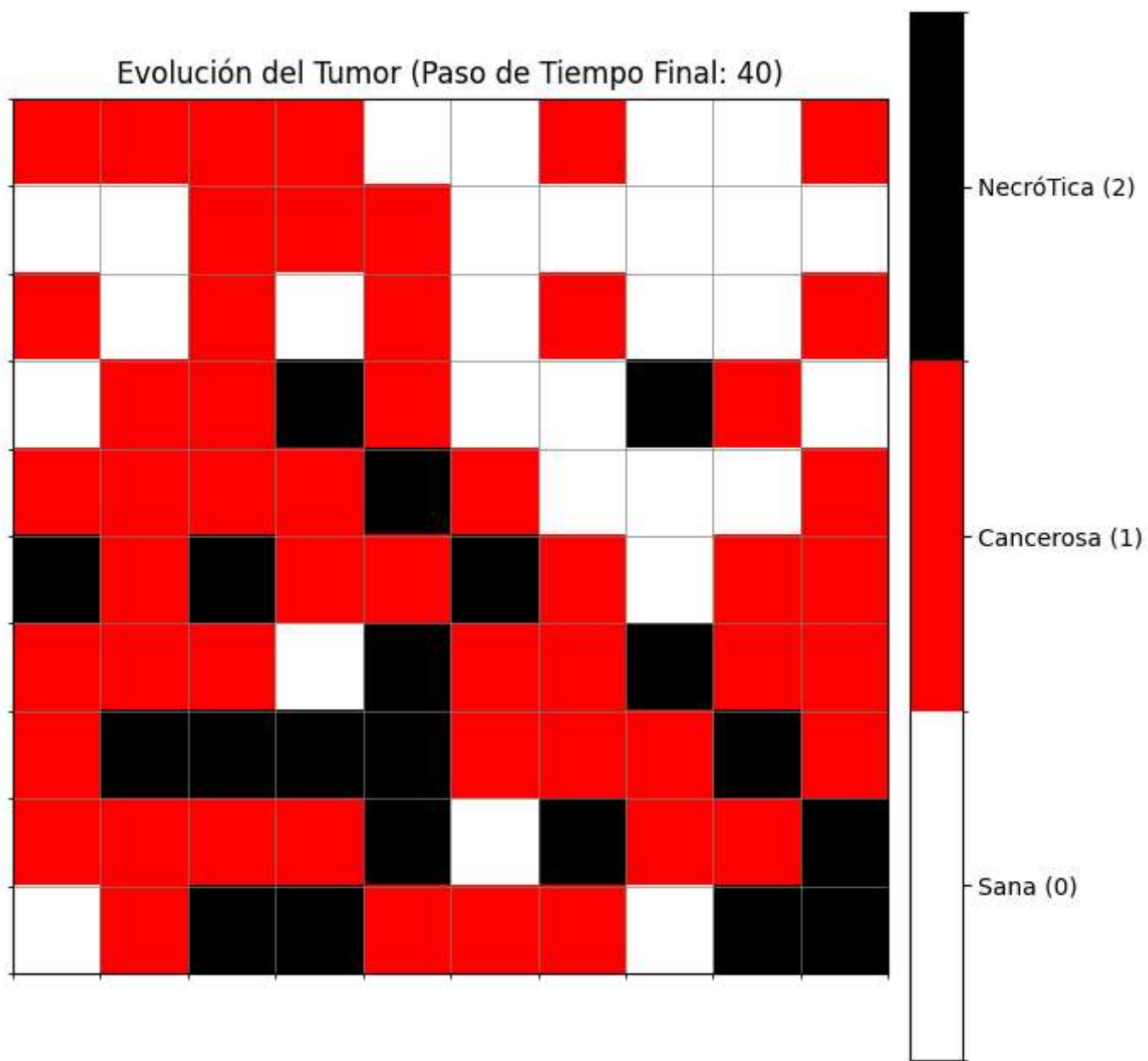
Generando imagen en el paso de tiempo: 36



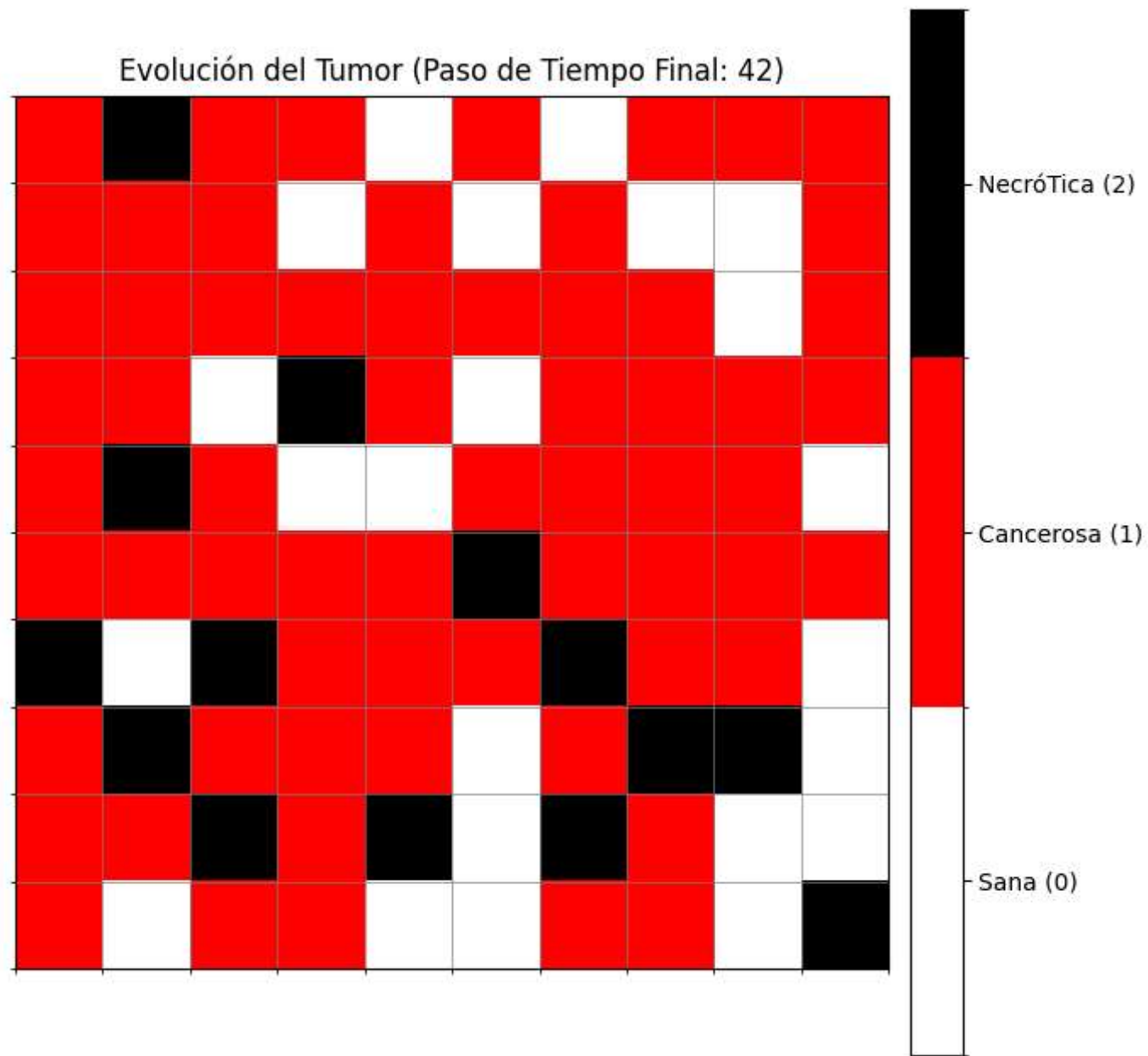
Generando imagen en el paso de tiempo: 38



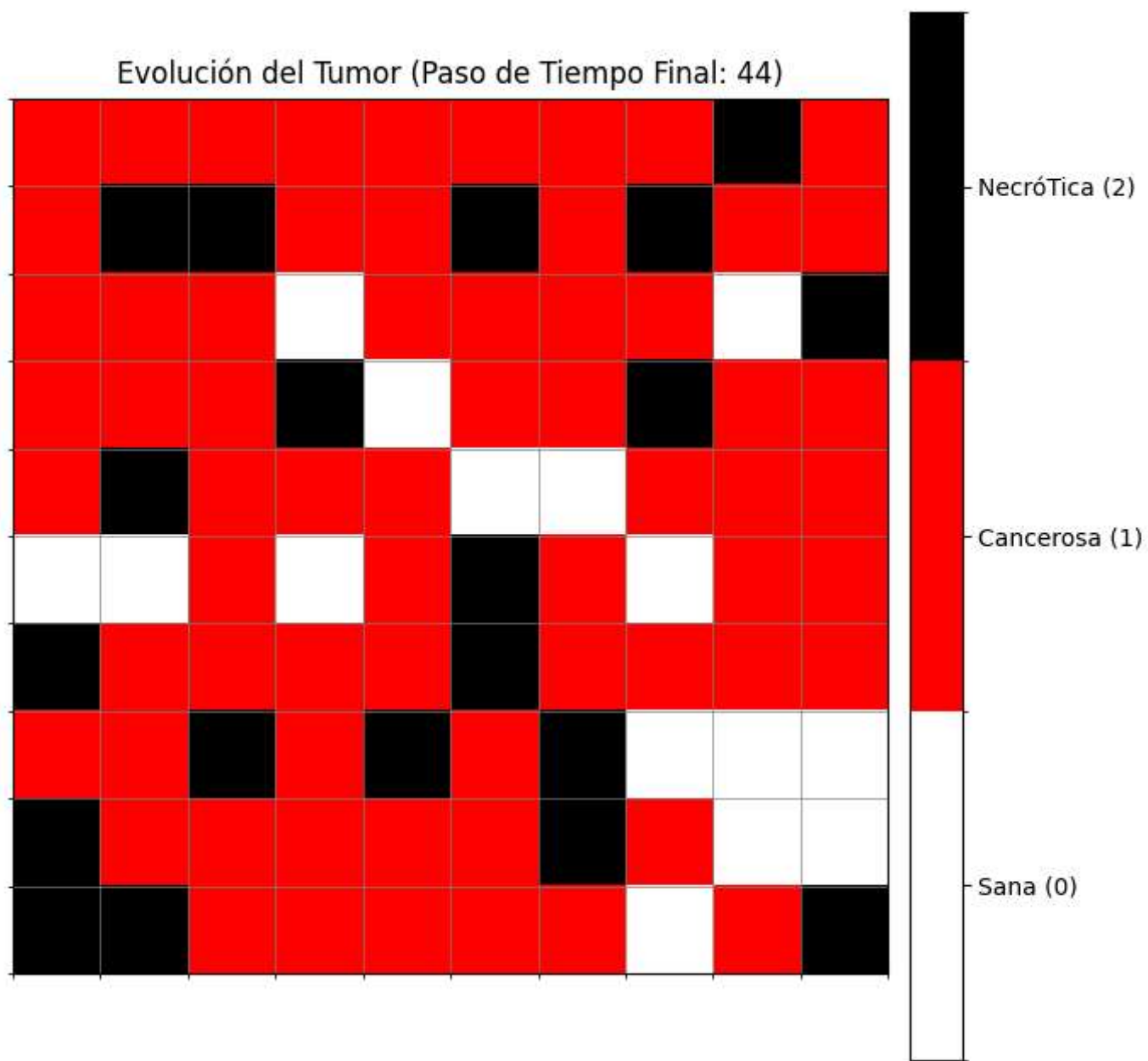
Generando imagen en el paso de tiempo: 40



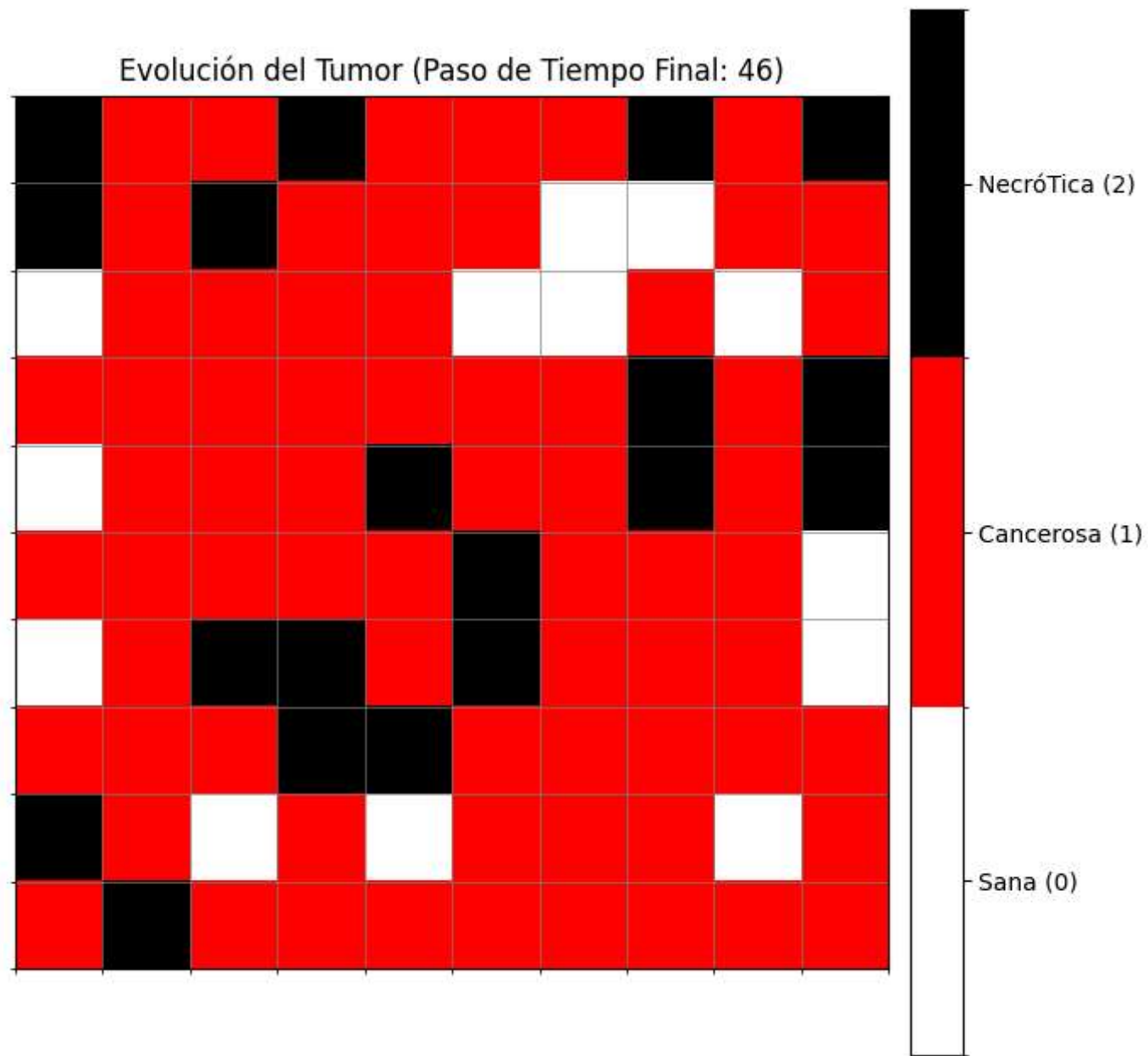
Generando imagen en el paso de tiempo: 42



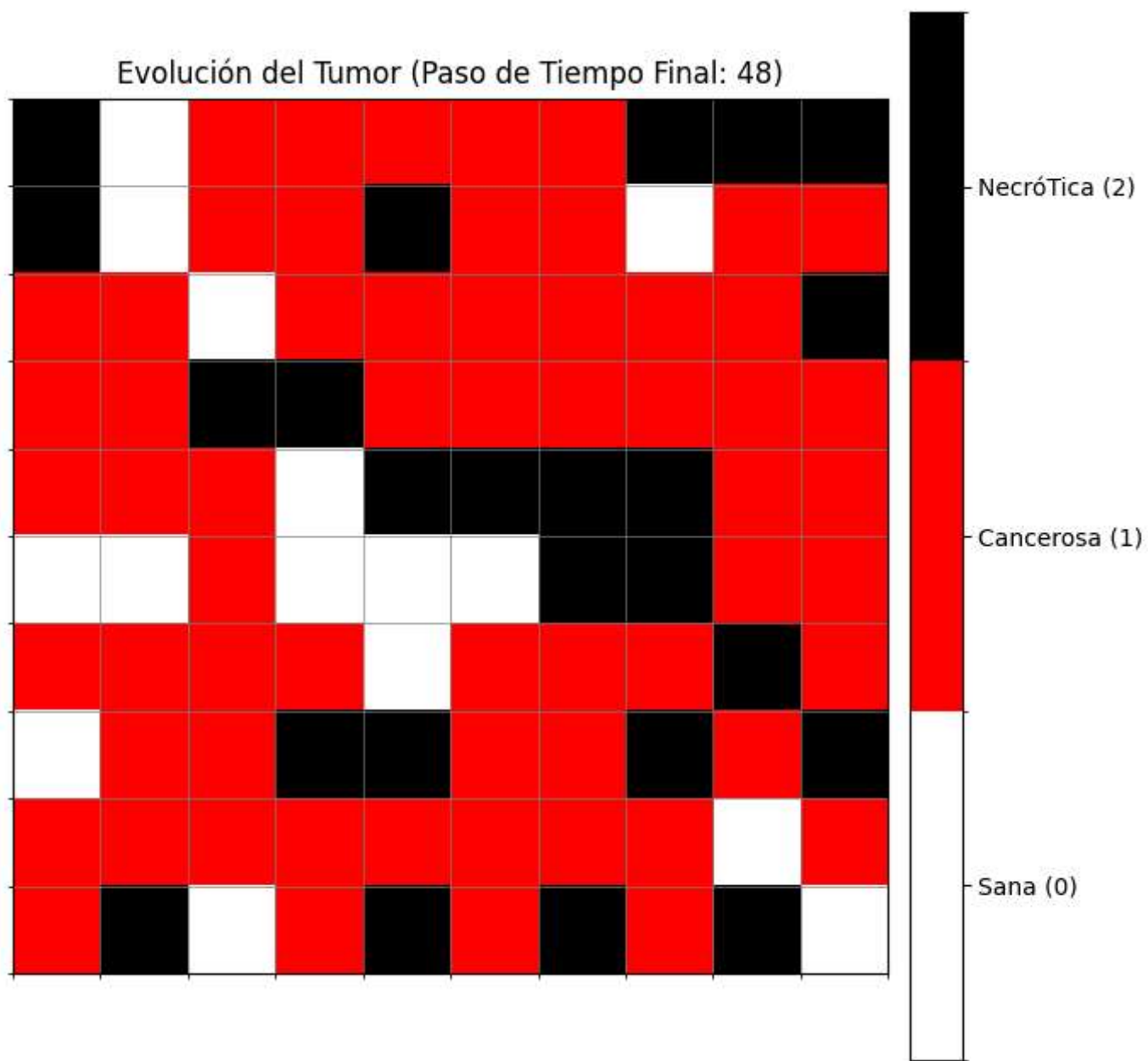
Generando imagen en el paso de tiempo: 44



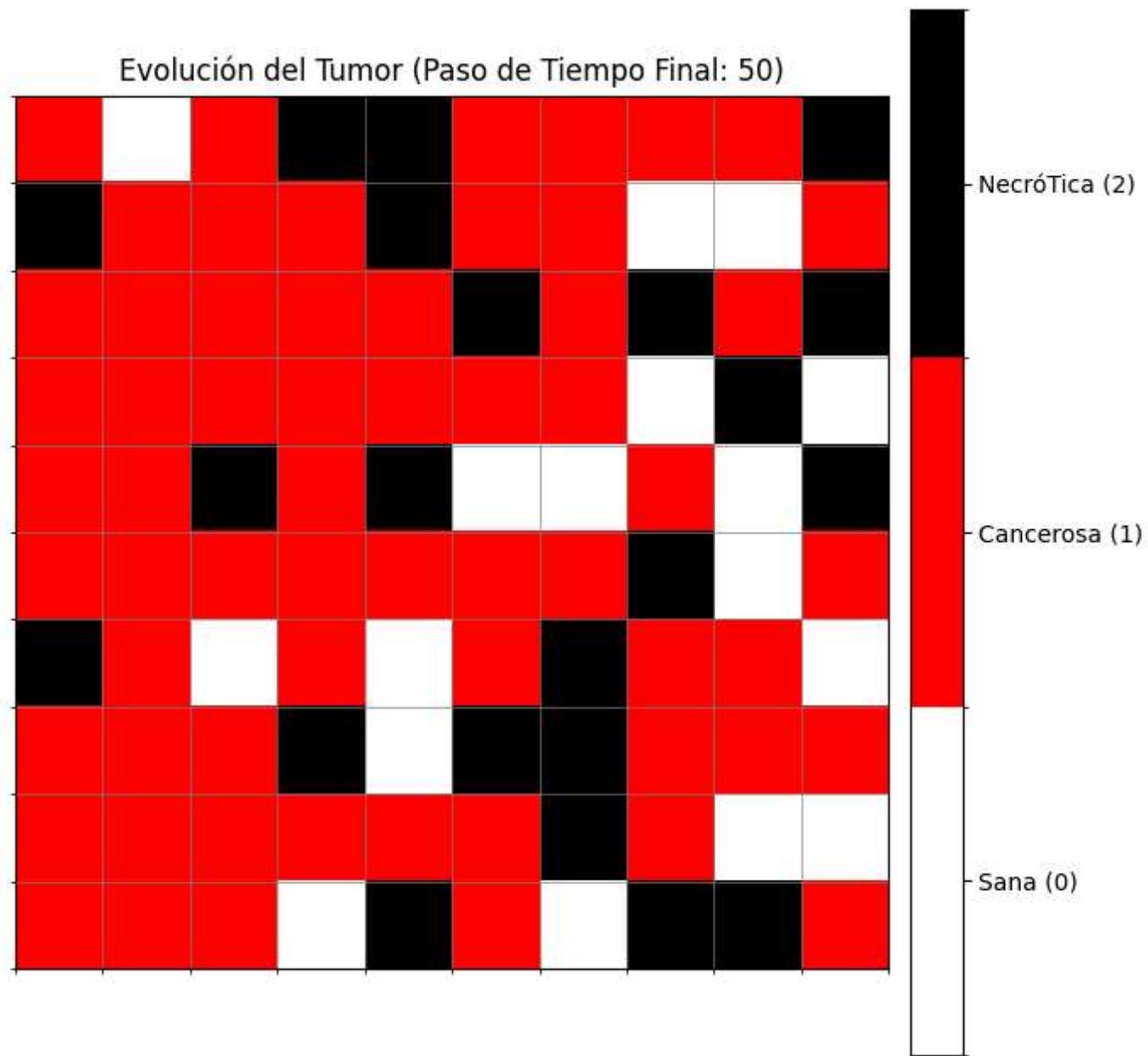
Generando imagen en el paso de tiempo: 46



Generando imagen en el paso de tiempo: 48



Generando imagen en el paso de tiempo: 50



Estado Final de la Simulación (Paso de Tiempo Final: 50)

