In [2]:

```python
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 intialize_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()


def plot_metrics_vs_time(data_list, time_steps, title, ylabel):
    """
    Plots a line graph of a given metric over time steps.
    Parameters:
```

```
    data_list (list): List of metric values to plot.
    time_steps (list): List of time steps corresponding to the metric values.
    title (str): Title of the graph.
    ylabel (str): Label for the Y-axis.
    Returns:
    None
    """

    plt.figure(figsize=(10, 6))


    plt.plot(time_steps, data_list, marker='o', linestyle='-', markersize=4)


    plt.title(title)
    plt.xlabel("Paso de Tiempo (t)")
    plt.ylabel(ylabel)
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)


    if time_steps:
        plt.xlim(0, time_steps[-1])

    plt.show()
```

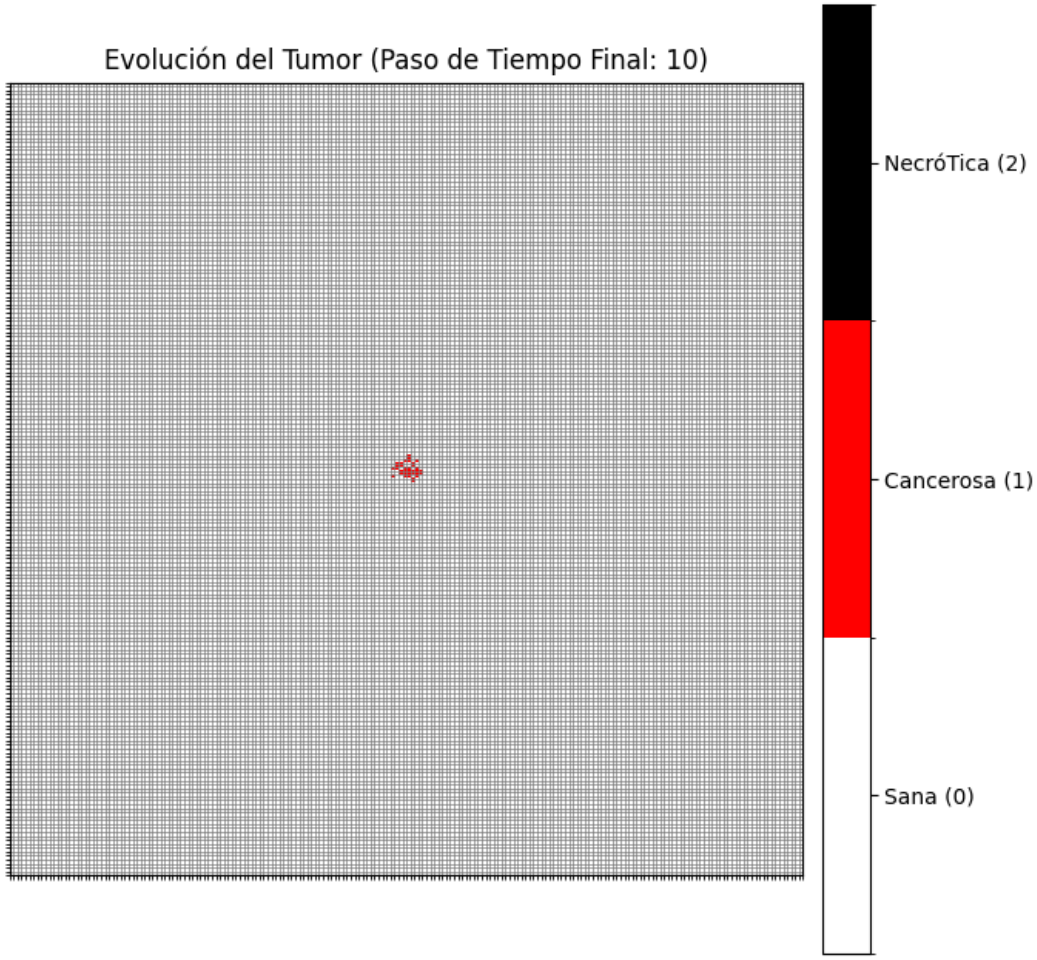
In [3]:

```
ROWS = 200
COLS = 200
Time_Units = 300

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

alpha = 0.5

RESISTANCE_SCORE = 0.2

P_clean = 0.5
P_damage = 0.05
P_necro_dead = 0.1

grid = intialize_grid(ROWS, COLS)

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

densidad_global = []
celulas_activas = []
time_step_list = []
plot_interval = 10

for t in range(Time_Units):

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

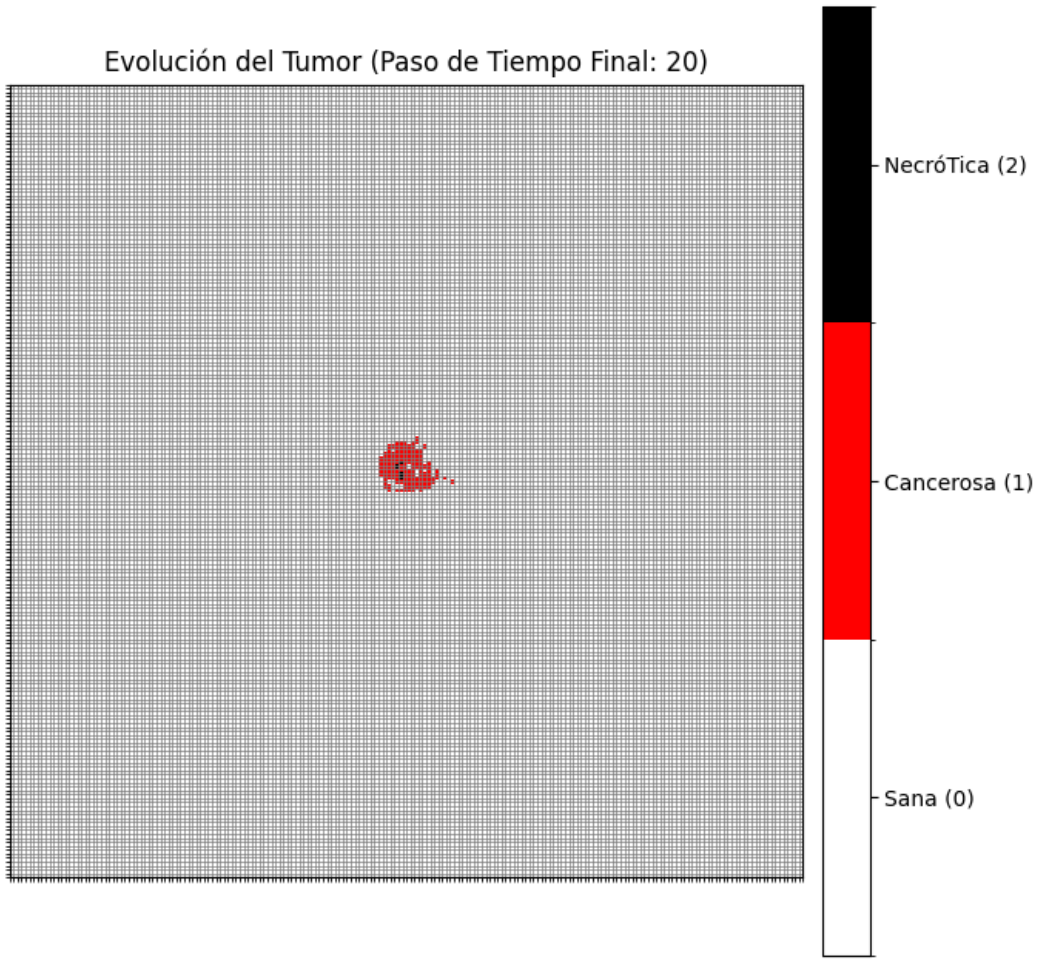
```
        break

    global_density = N_active / total_cells_in_grid

    alpha_t = (1 - RESISTANCE_SCORE) * global_density

    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_t)

            ## 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) % plot_interval == 0 or (t + 1) == Time_Units:
        print(f"Generando imagen en el paso de tiempo: {t + 1}")
        print(f"Células activas: {len(active_cell_set)}")
        print(f"Densidad global: {len(active_cell_set) / total_cells_in_grid:.4f}")
        plot_grid(grid, title="Evolución del Tumor", t_step=t + 1)
        time_step_list.append(t + 1)
        densidad_global.append(len(active_cell_set) / total_cells_in_grid)
        celulas_activas.append(len(active_cell_set))



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

```
plot_metrics_vs_time(densidad_global,
                     time_step_list,
                     "Evolución de la Densidad Global vs. Tiempo",
                     "Densidad Global (N_activas / N_total)")

# Llama a la función para las Células Activas
plot_metrics_vs_time(celulas_activas,
                     time_step_list,
                     "Evolución de Células Activas vs. Tiempo",
                     "Conteo de Células Activas (Cancerosa + Necrótica)")
```

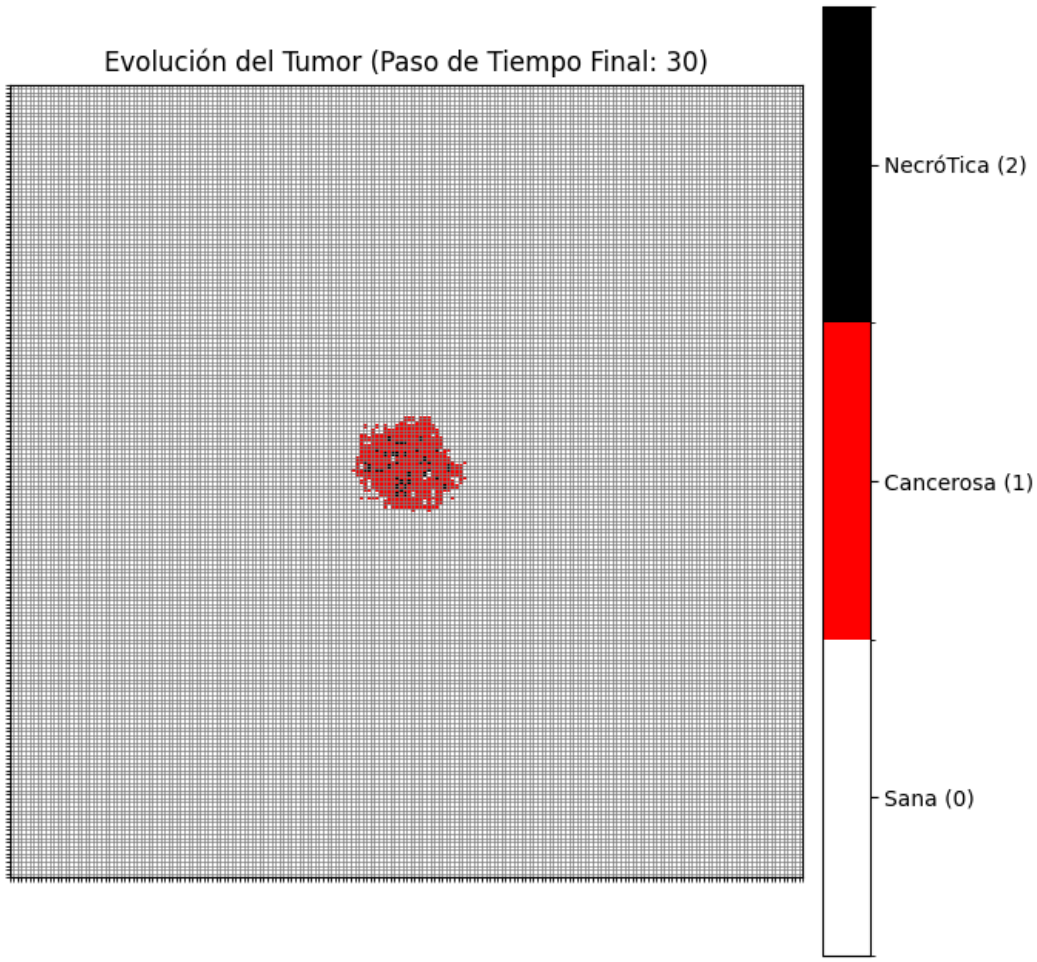
```
Generando imagen en el paso de tiempo: 10
Células activas: 23
Densidad global: 0.0006
```



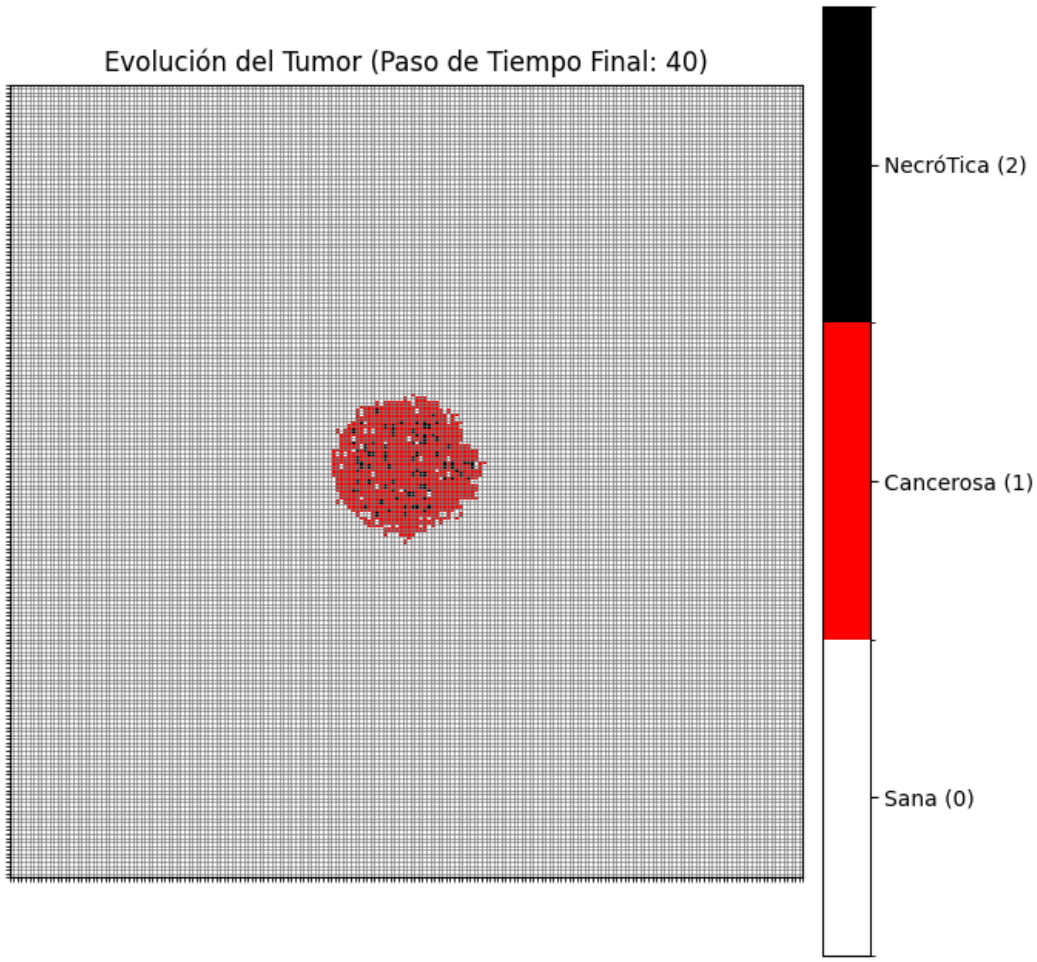

```
Generando imagen en el paso de tiempo: 20
Células activas: 135
Densidad global: 0.0034
```



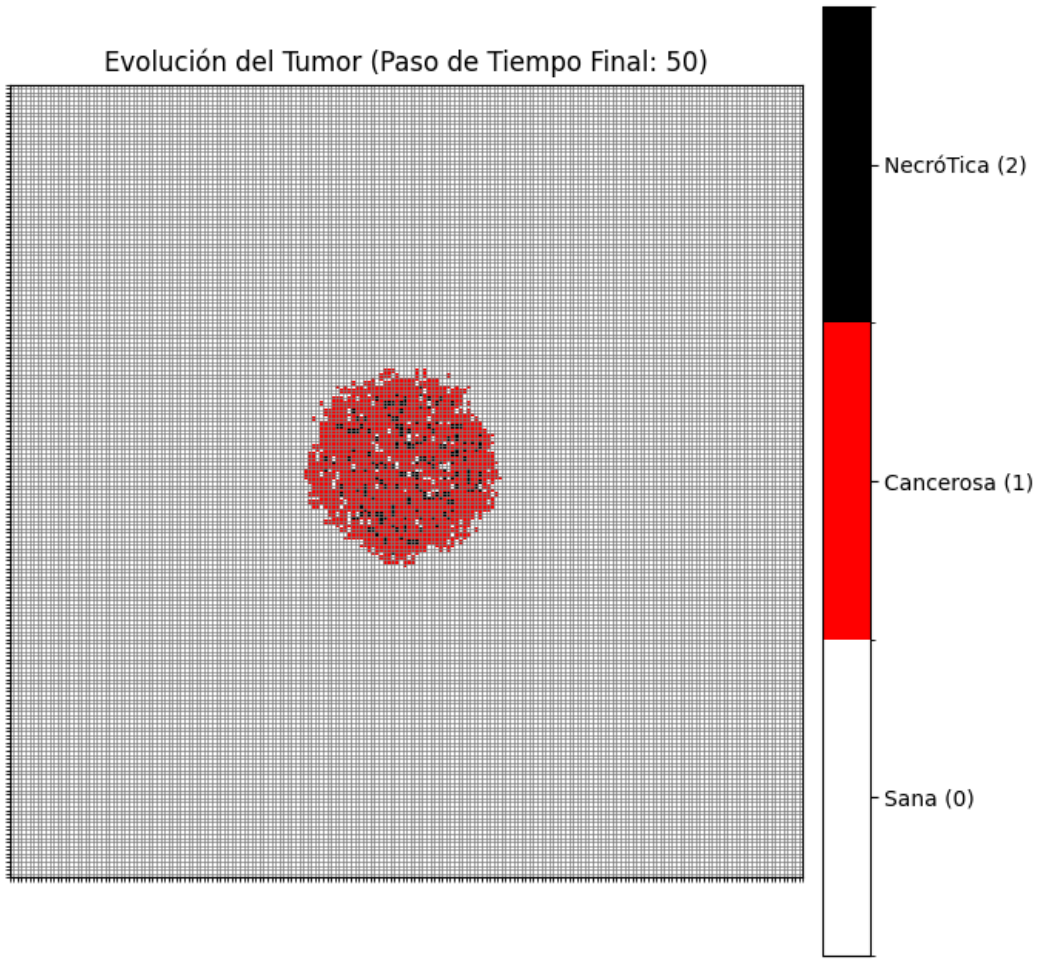

```
Generando imagen en el paso de tiempo: 30
Células activas: 443
Densidad global: 0.0111
```



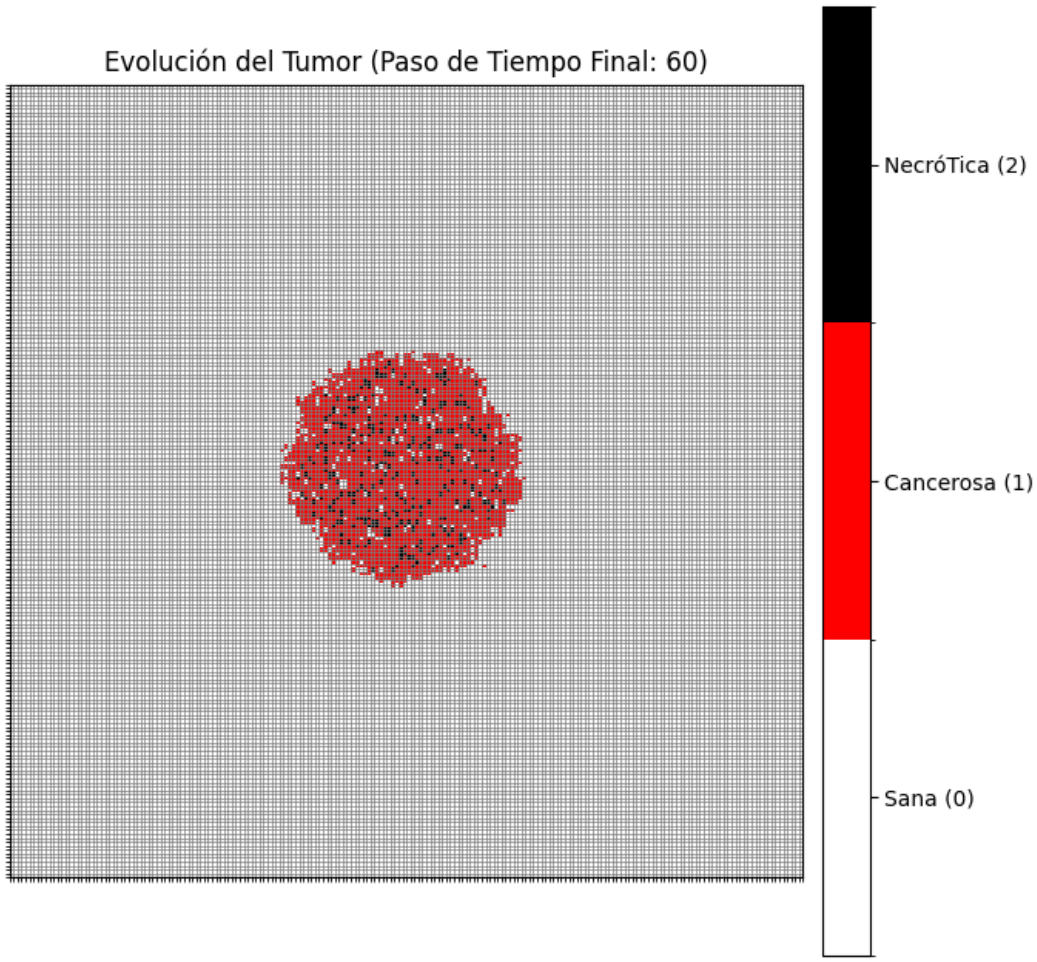

```
Generando imagen en el paso de tiempo: 40
Células activas: 944
Densidad global: 0.0236
```



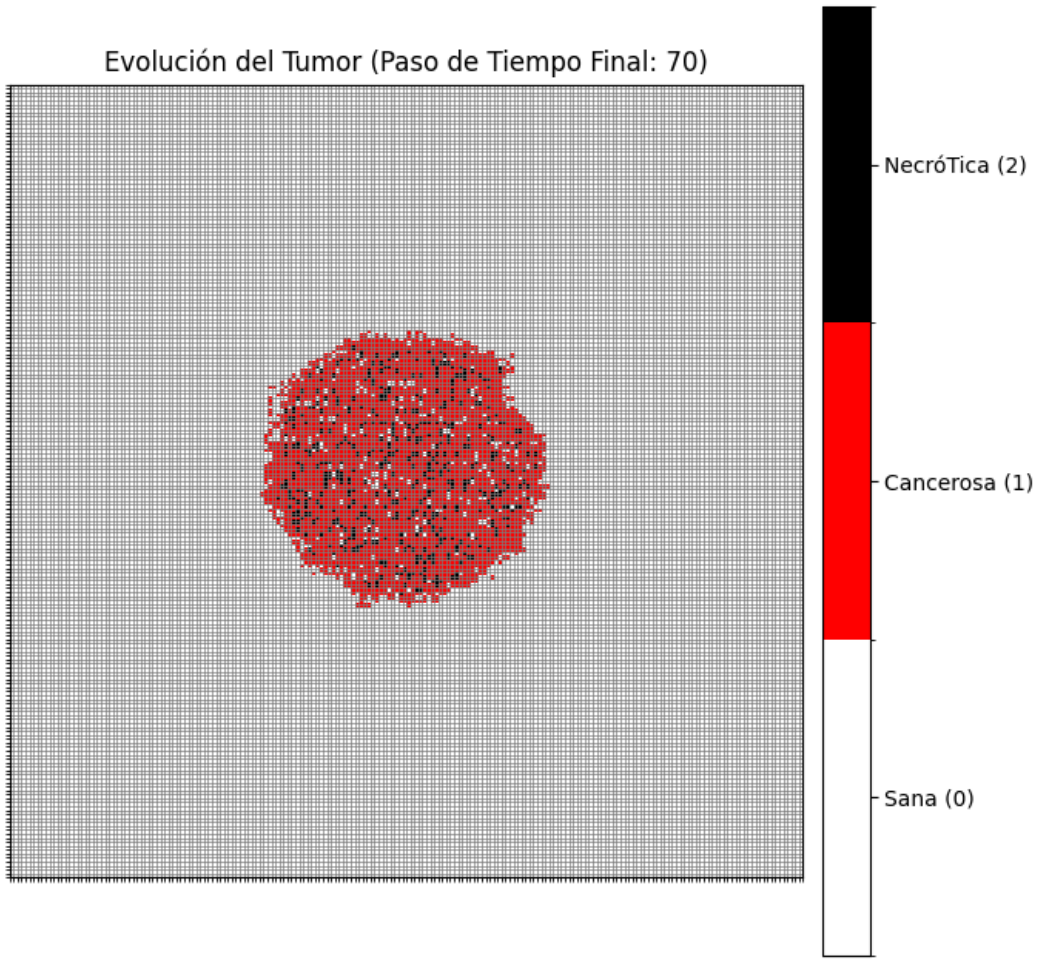

```
Generando imagen en el paso de tiempo: 50
Células activas: 1620
Densidad global: 0.0405
```



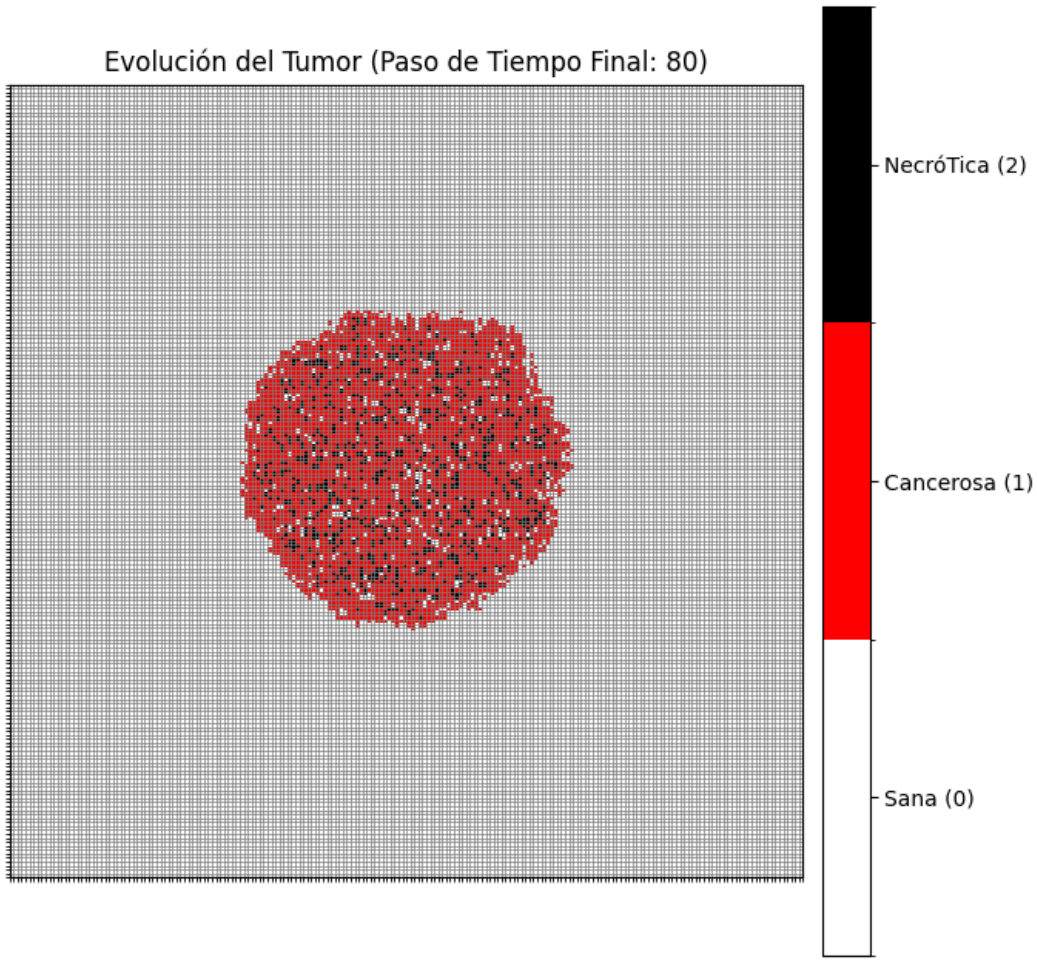

```
Generando imagen en el paso de tiempo: 60
Células activas: 2478
Densidad global: 0.0619
```



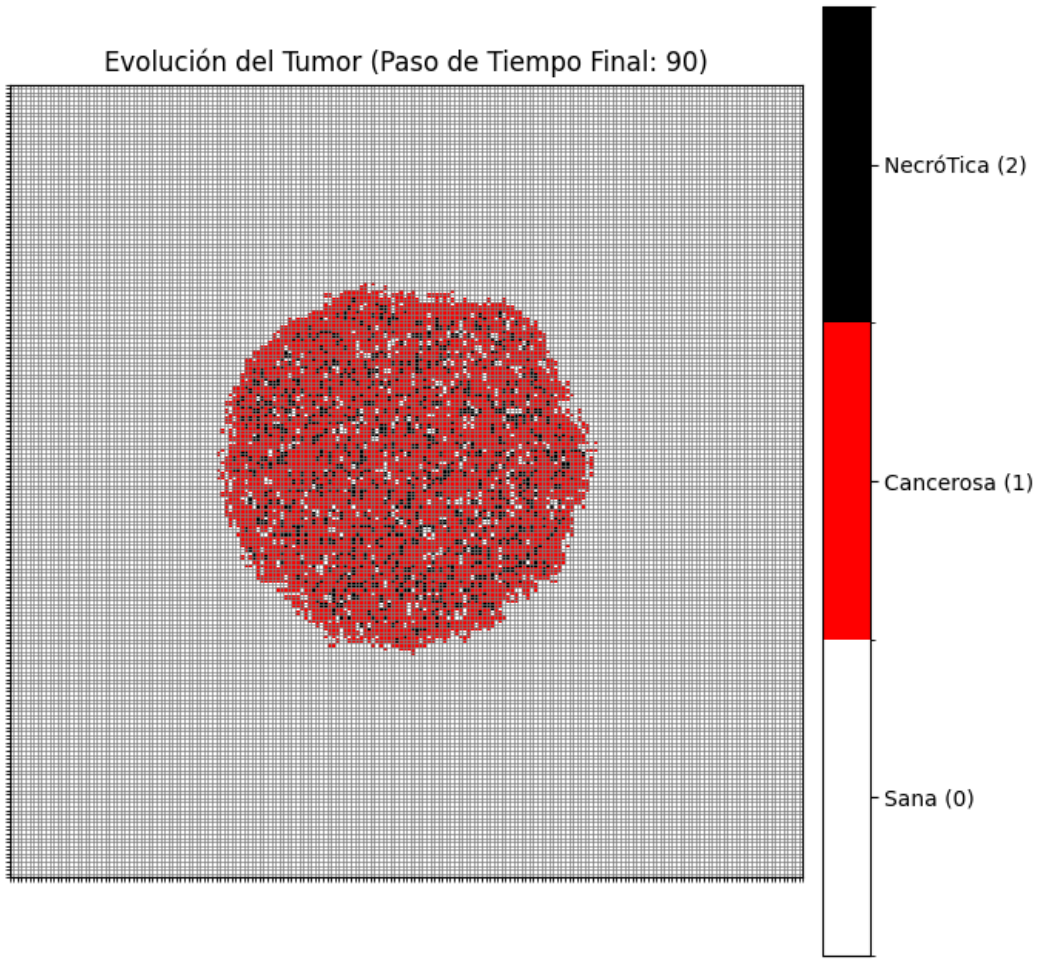

```
Generando imagen en el paso de tiempo: 70
Células activas: 3509
Densidad global: 0.0877
```



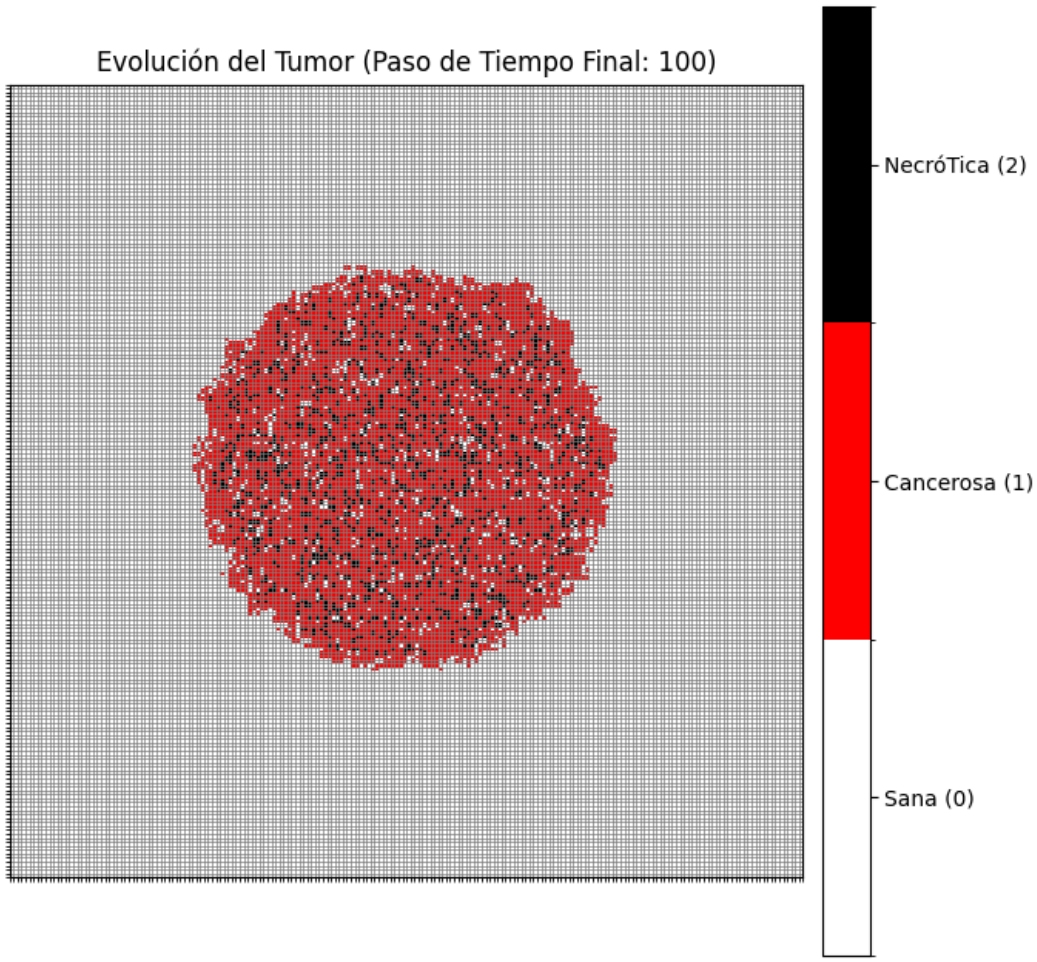

```
Generando imagen en el paso de tiempo: 80
Células activas: 4743
Densidad global: 0.1186
```



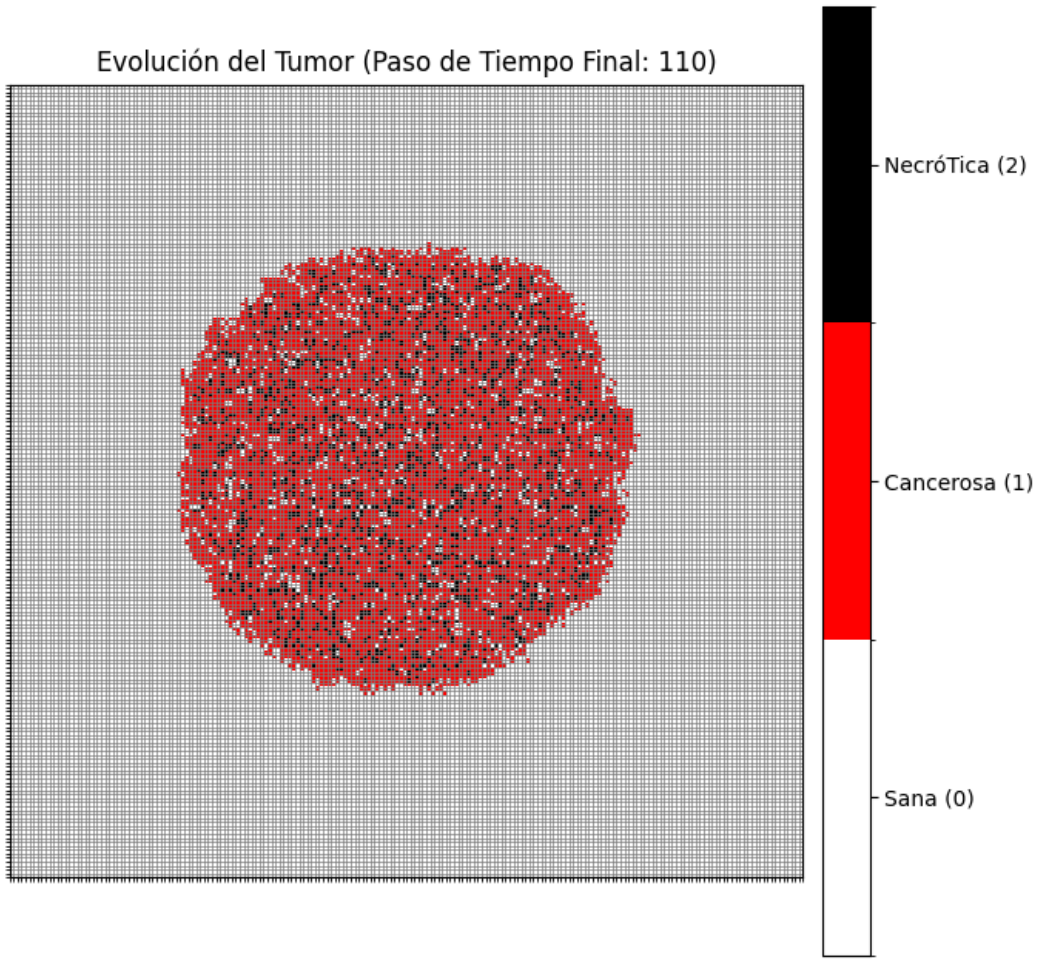

```
Generando imagen en el paso de tiempo: 90
Células activas: 6076
Densidad global: 0.1519
```



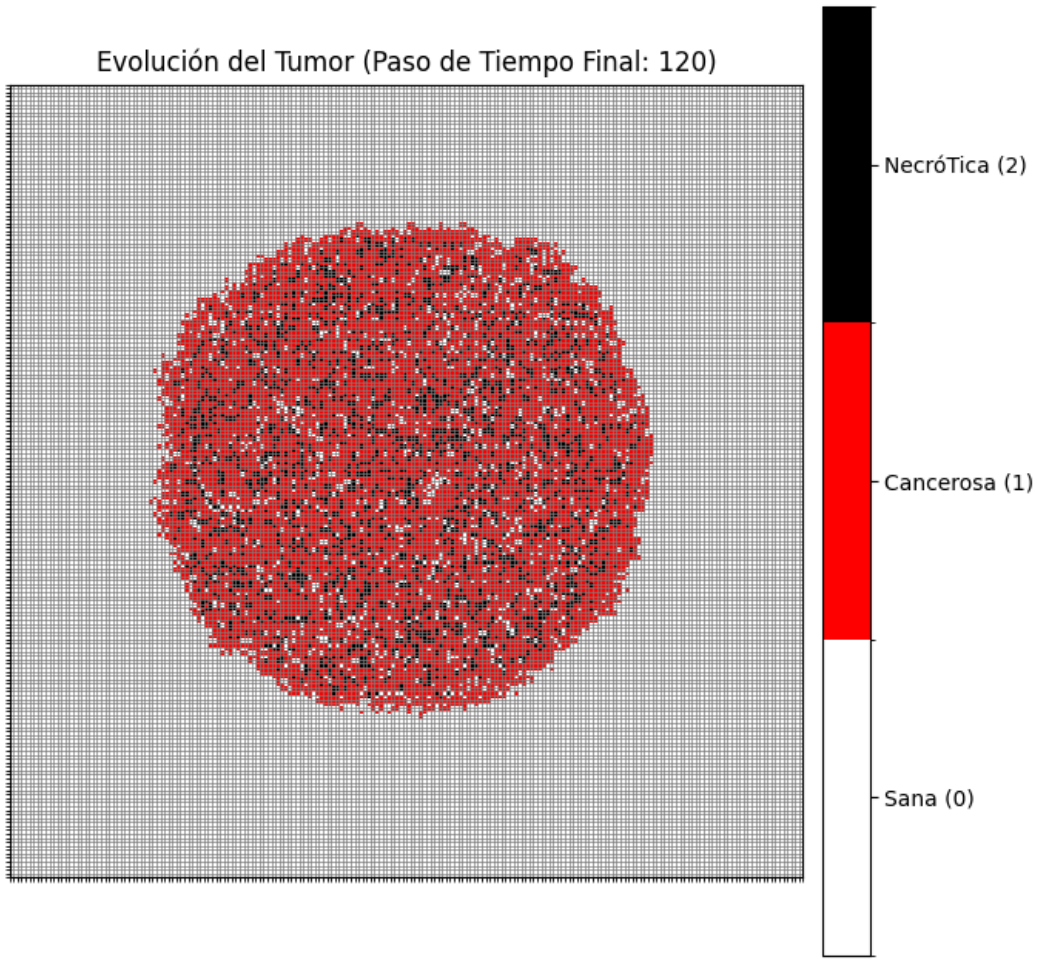

```
Generando imagen en el paso de tiempo: 100
Células activas: 7626
Densidad global: 0.1906
```



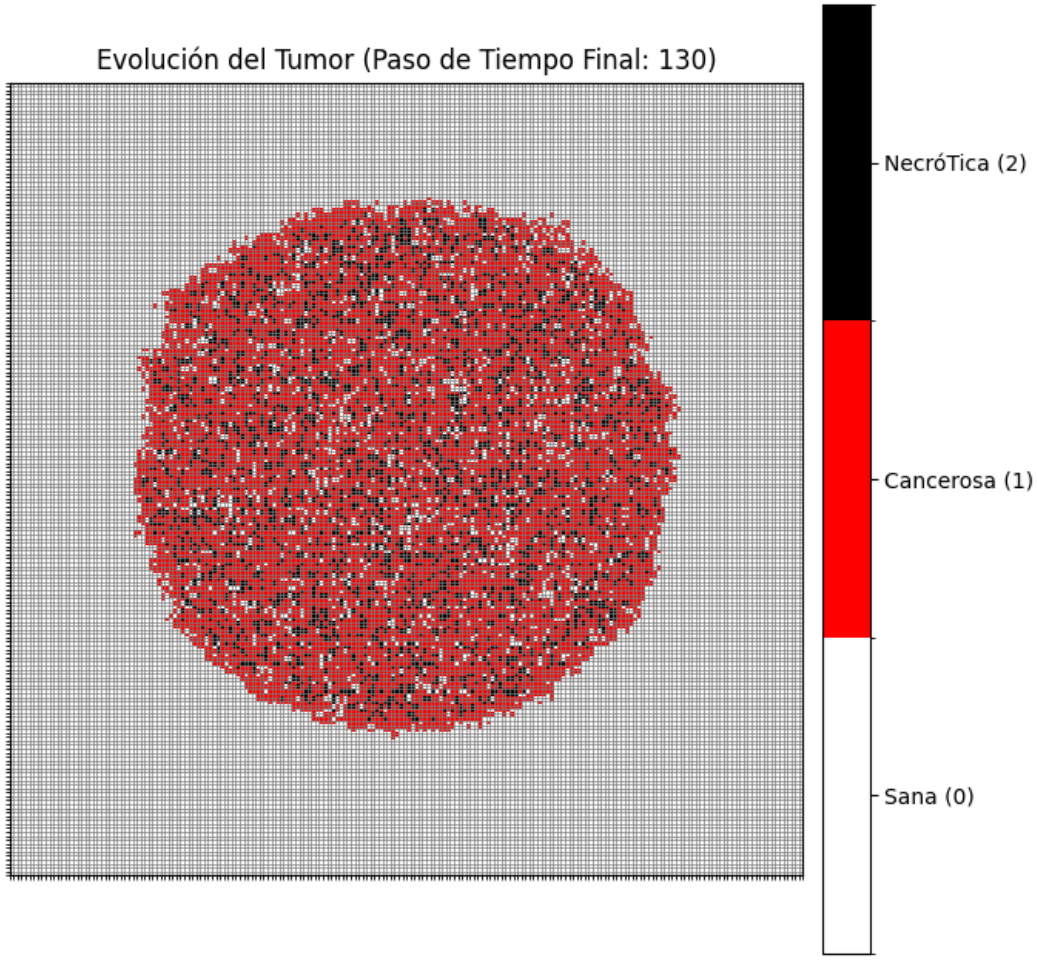

```
Generando imagen en el paso de tiempo: 110
Células activas: 9373
Densidad global: 0.2343
```



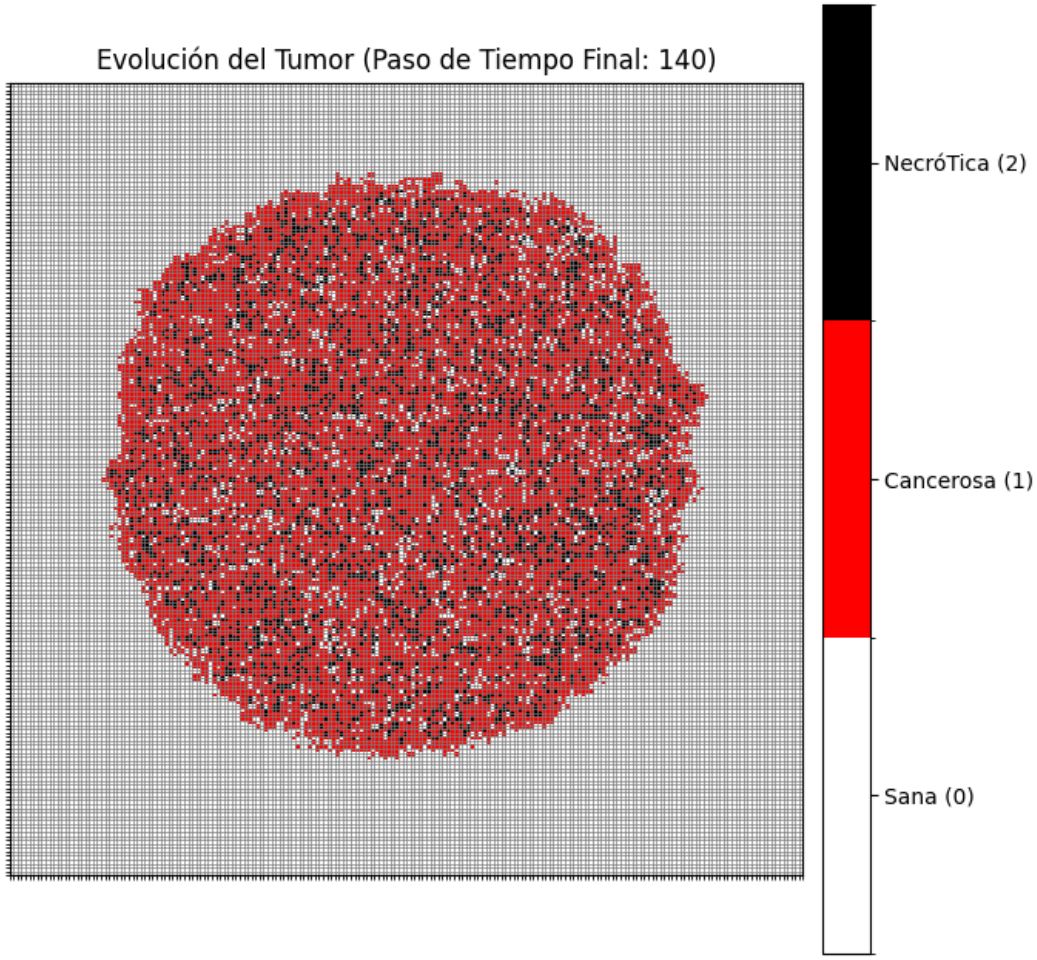

```
Generando imagen en el paso de tiempo: 120
Células activas: 11228
Densidad global: 0.2807
```



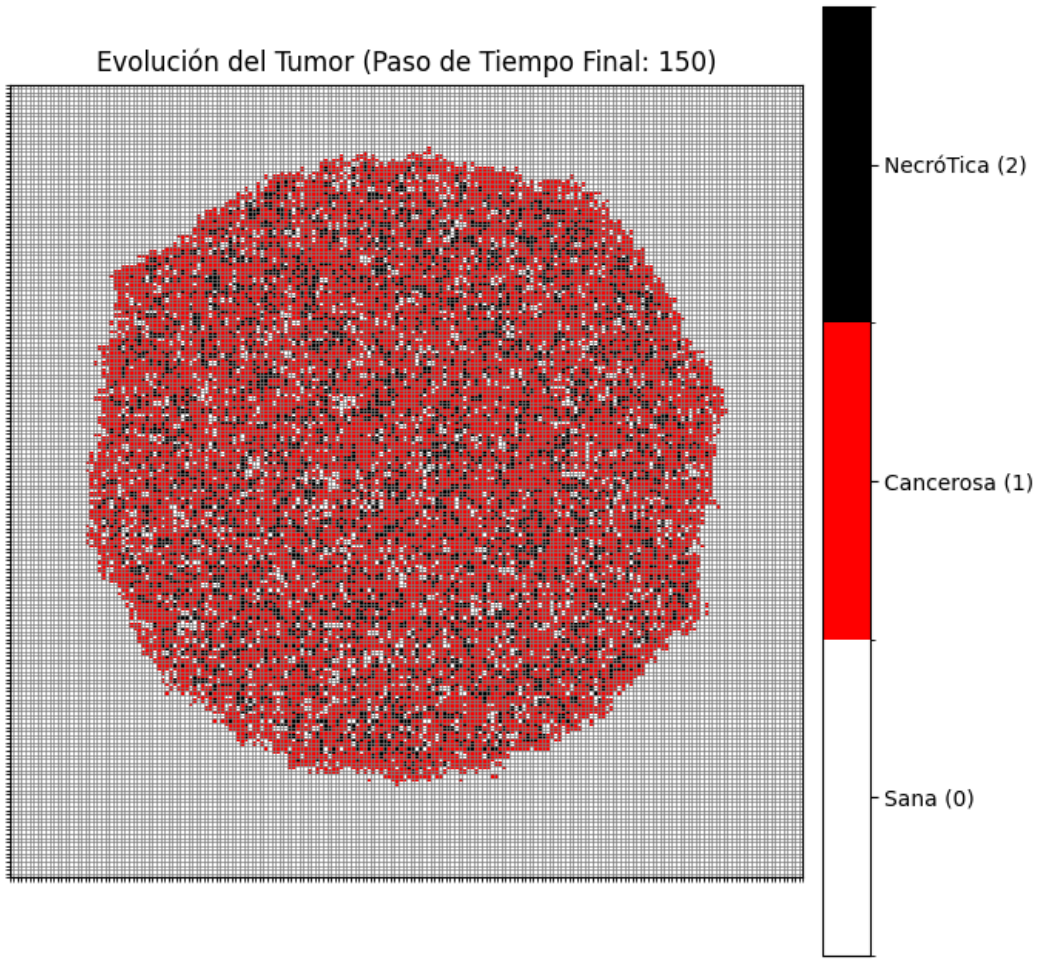

```
Generando imagen en el paso de tiempo: 130
Células activas: 13207
Densidad global: 0.3302
```



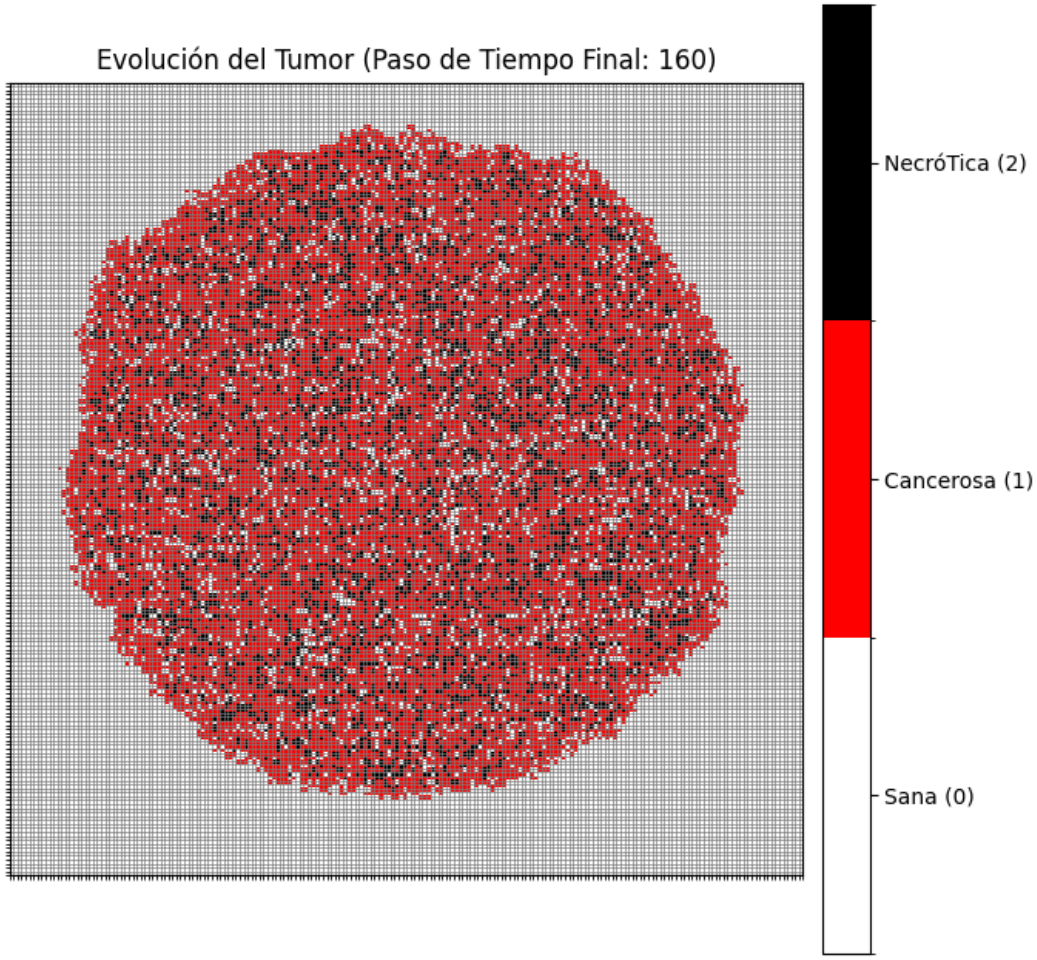

```
Generando imagen en el paso de tiempo: 140
Células activas: 15389
Densidad global: 0.3847
```



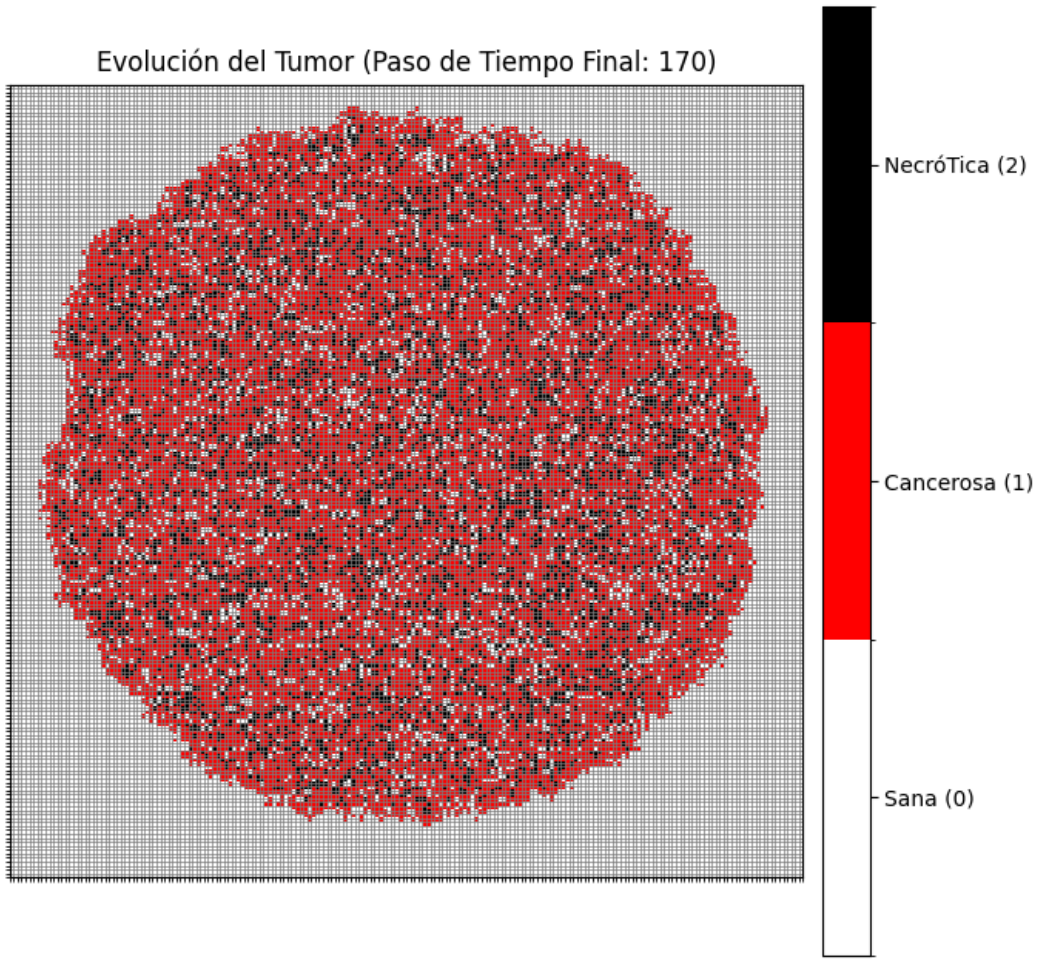

```
Generando imagen en el paso de tiempo: 150
Células activas: 17653
Densidad global: 0.4413
```



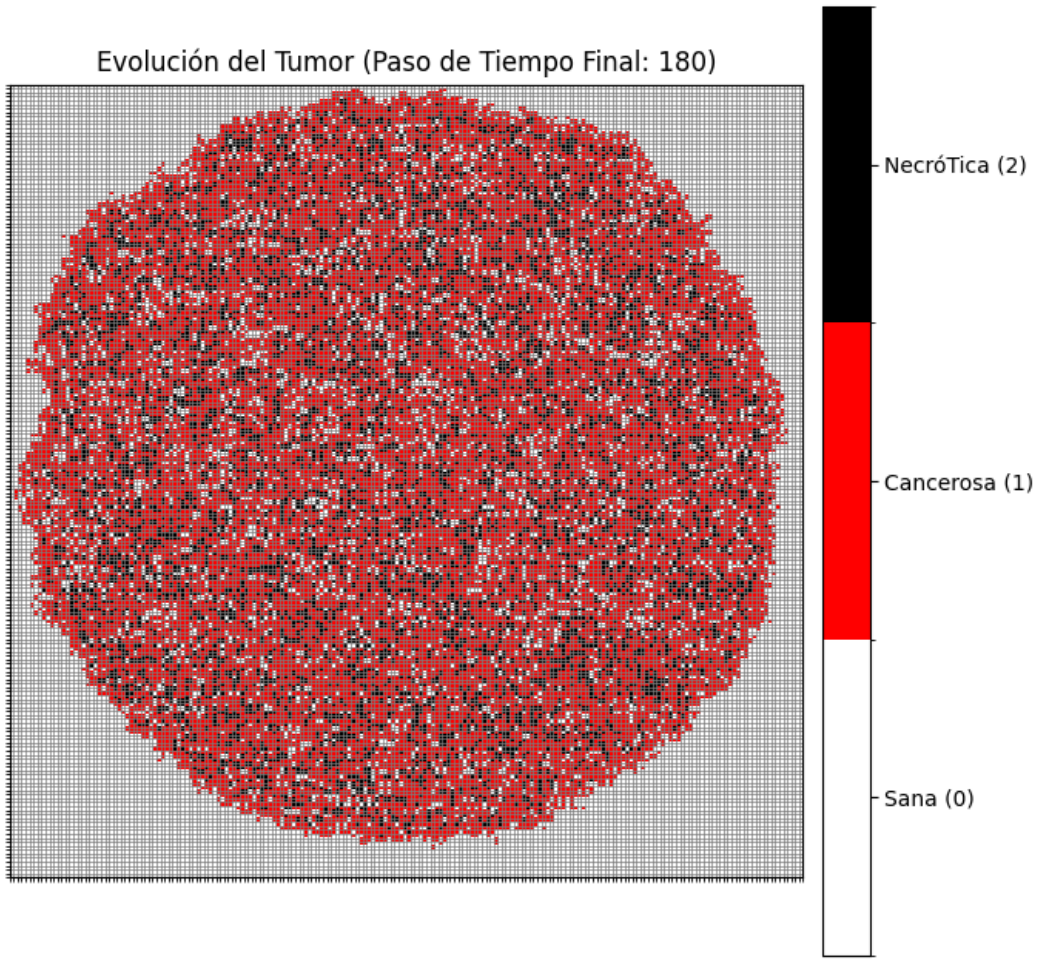

```
Generando imagen en el paso de tiempo: 160
Células activas: 20033
Densidad global: 0.5008
```



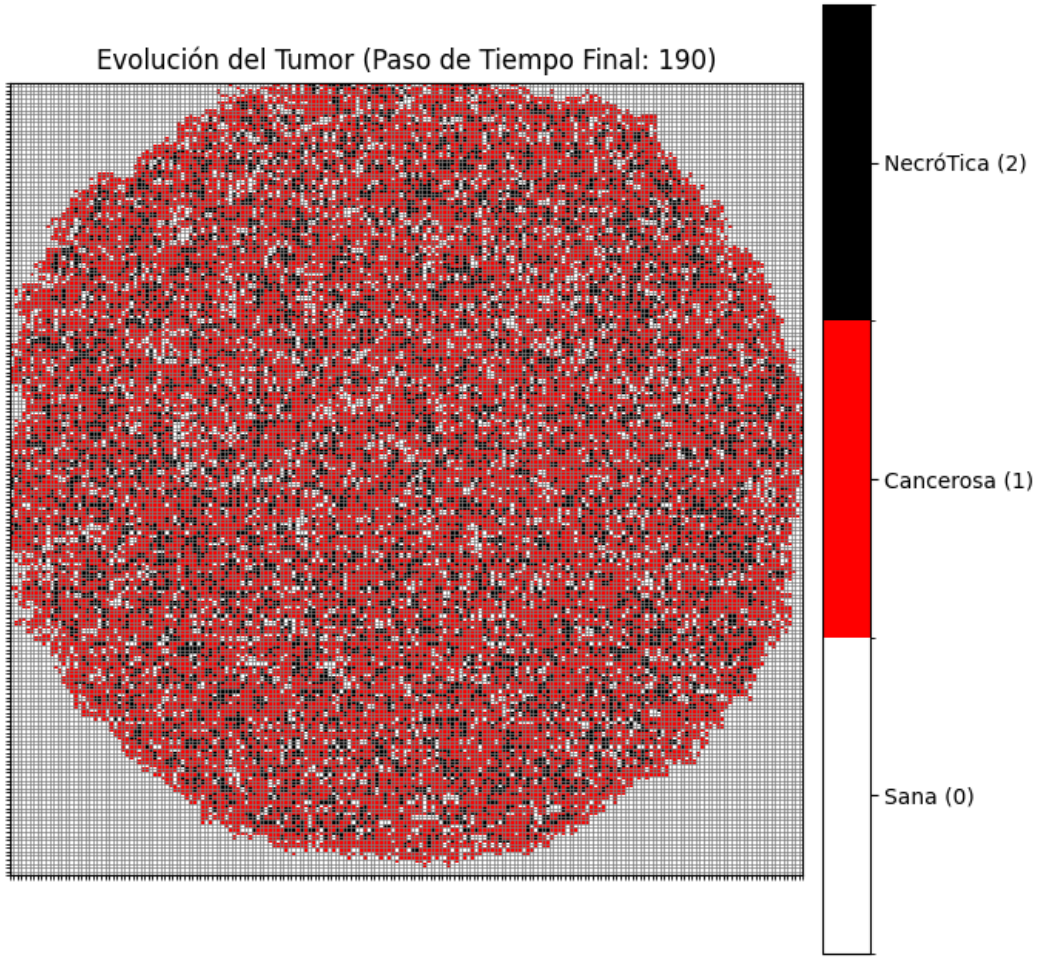

```
Generando imagen en el paso de tiempo: 170
Células activas: 22433
Densidad global: 0.5608
```



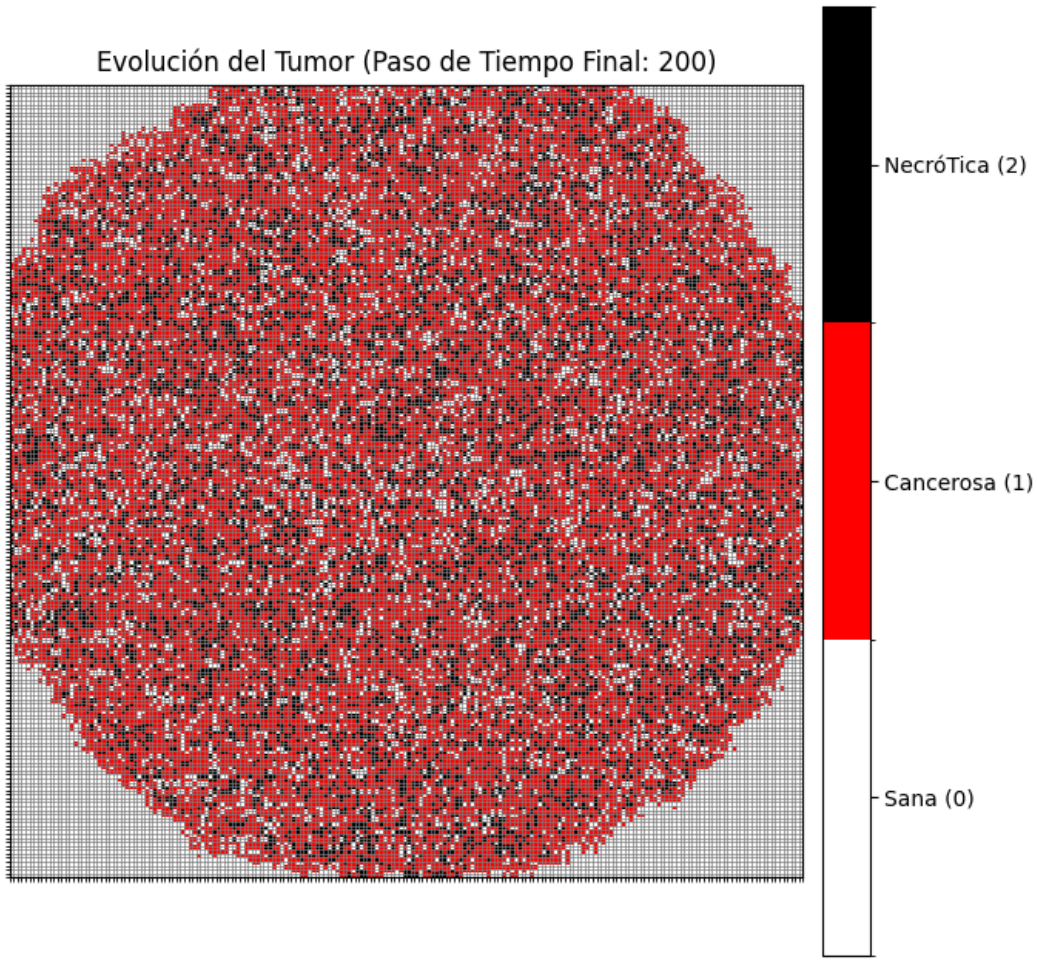

```
Generando imagen en el paso de tiempo: 180
Células activas: 25198
Densidad global: 0.6300
```



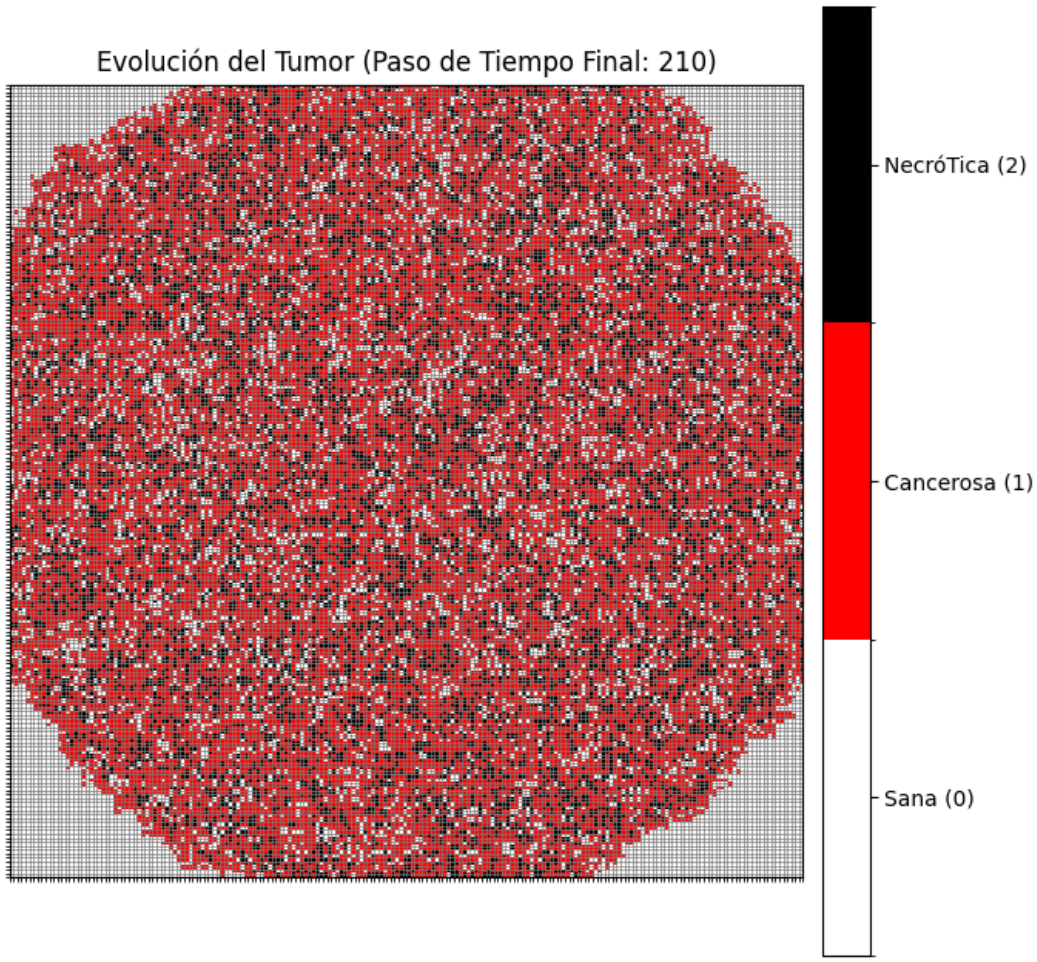

```
Generando imagen en el paso de tiempo: 190
Células activas: 27568
Densidad global: 0.6892
```



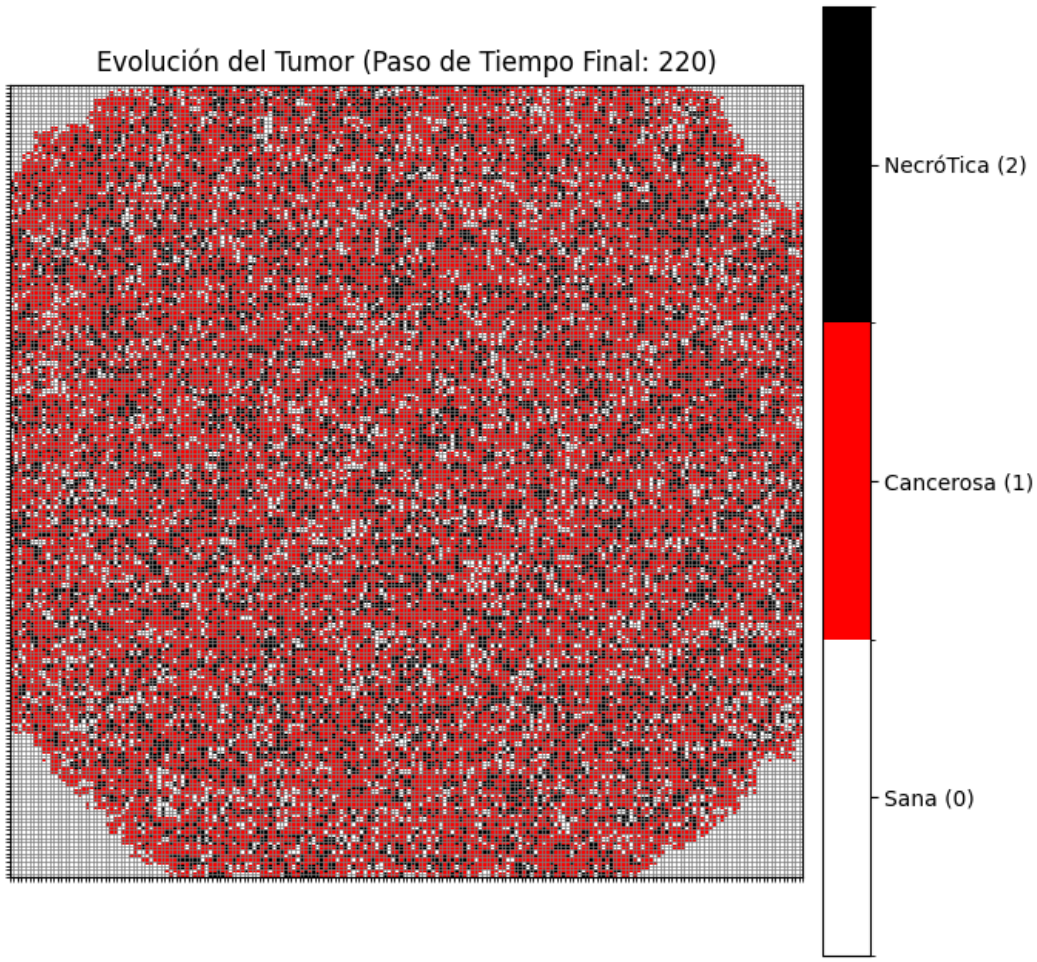

```
Generando imagen en el paso de tiempo: 200
Células activas: 29412
Densidad global: 0.7353
```



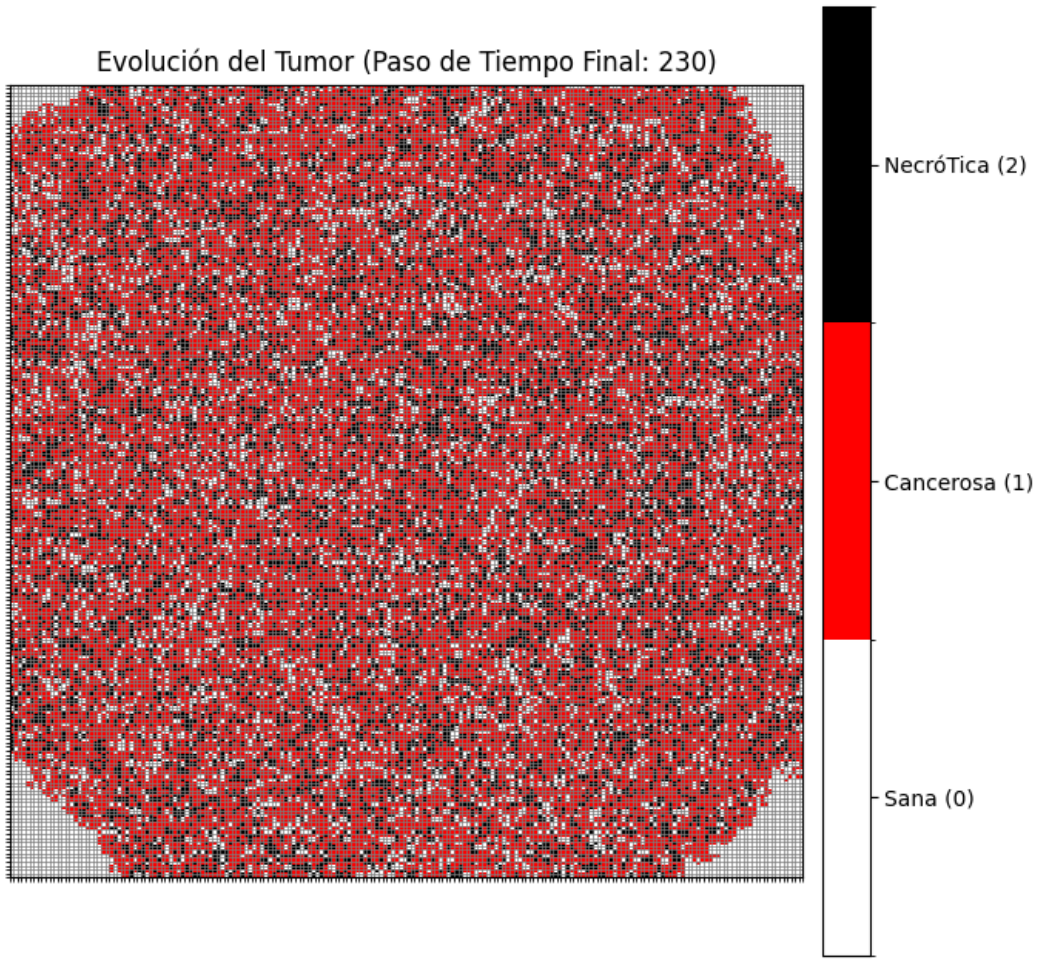

```
Generando imagen en el paso de tiempo: 210
Células activas: 30835
Densidad global: 0.7709
```



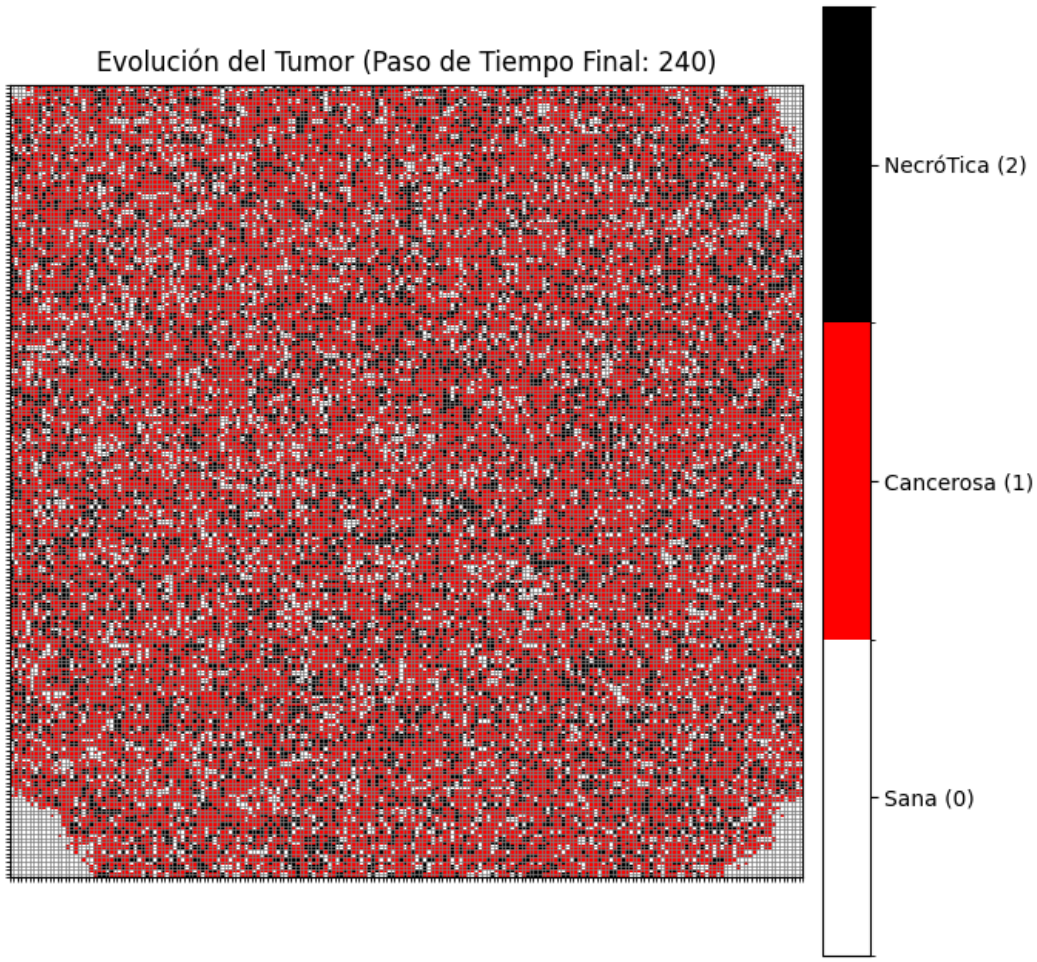

```
Generando imagen en el paso de tiempo: 220
Células activas: 32146
Densidad global: 0.8036
```



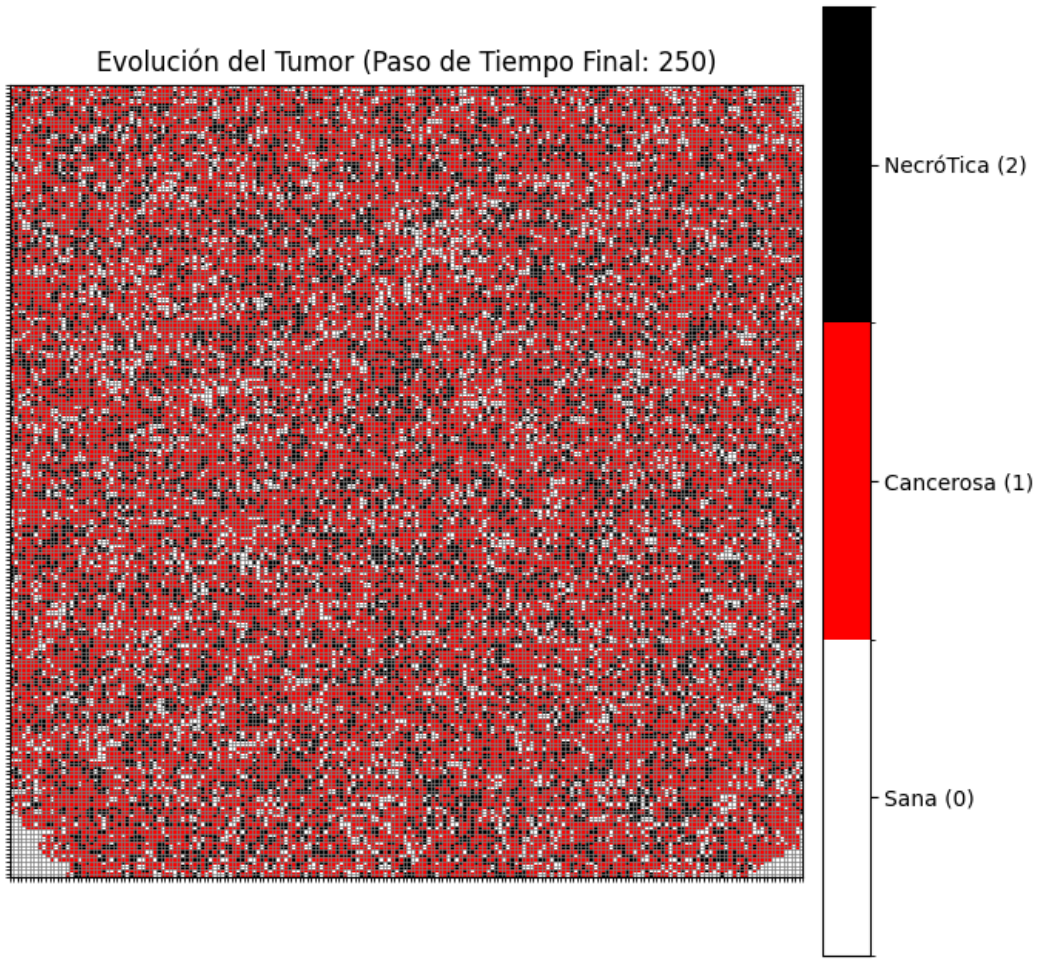

```
Generando imagen en el paso de tiempo: 230
Células activas: 32636
Densidad global: 0.8159
```



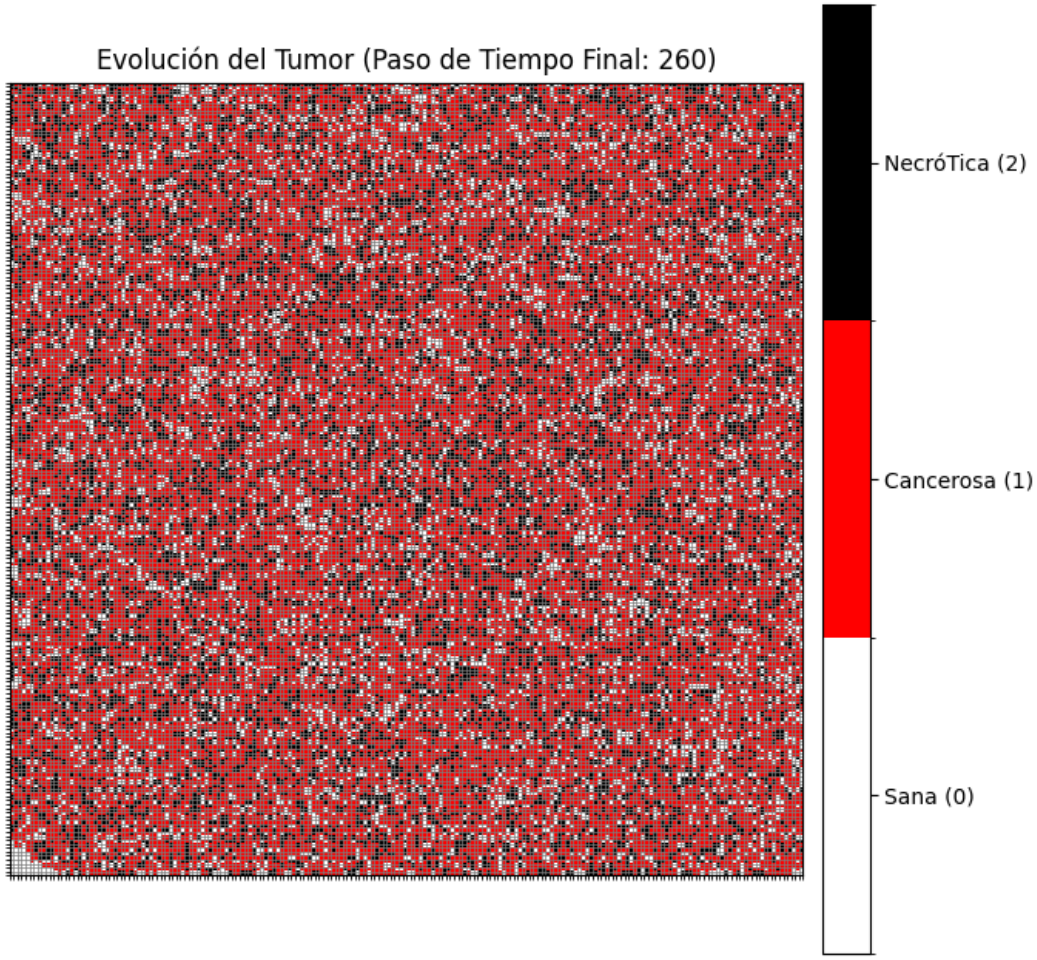

```
Generando imagen en el paso de tiempo: 240
Células activas: 33308
Densidad global: 0.8327
```



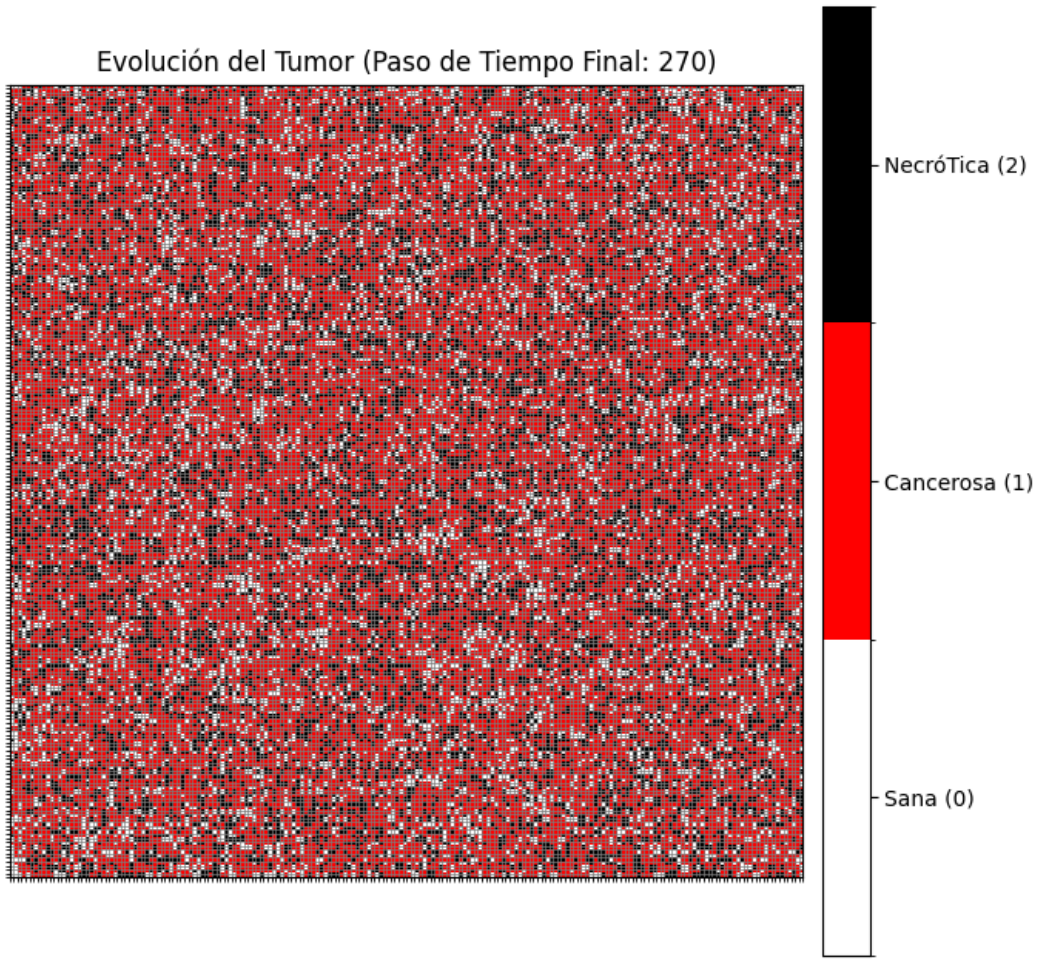

```
Generando imagen en el paso de tiempo: 250
Células activas: 33611
Densidad global: 0.8403
```



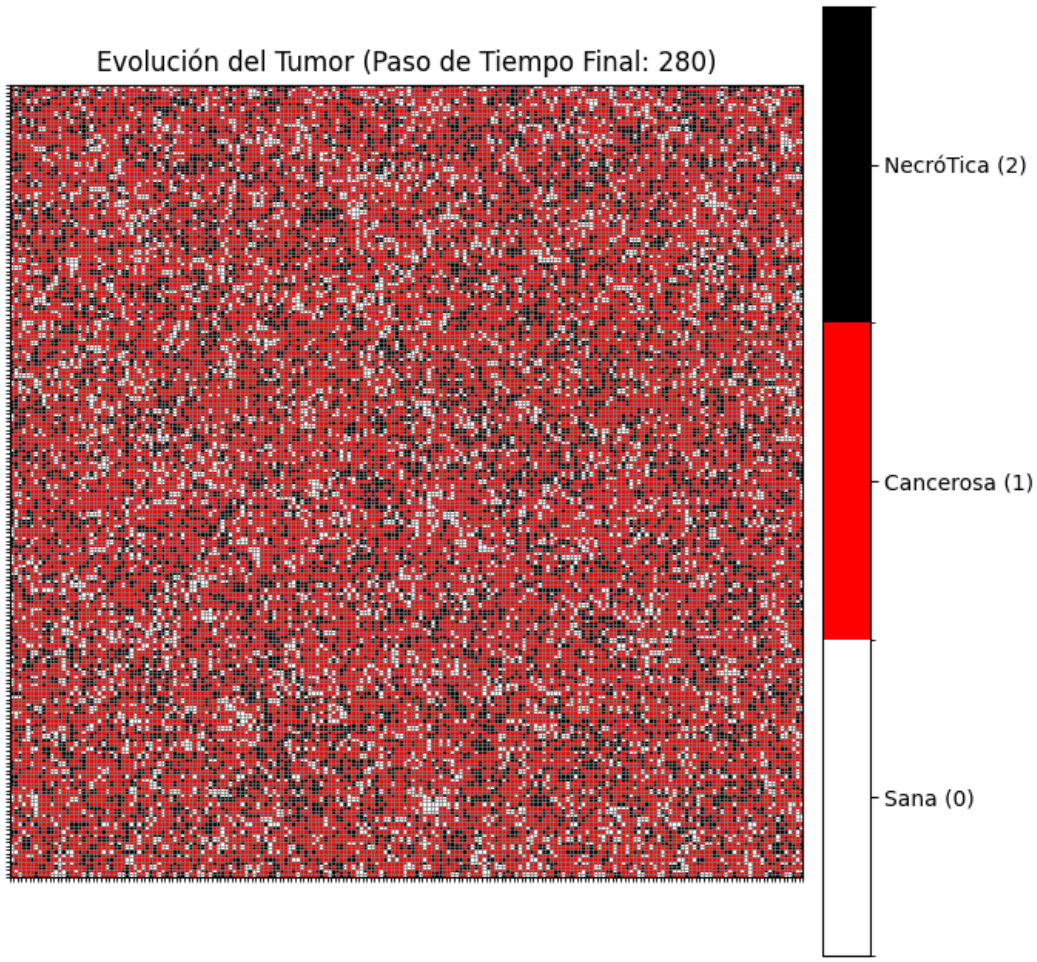

```
Generando imagen en el paso de tiempo: 260
Células activas: 33673
Densidad global: 0.8418
```



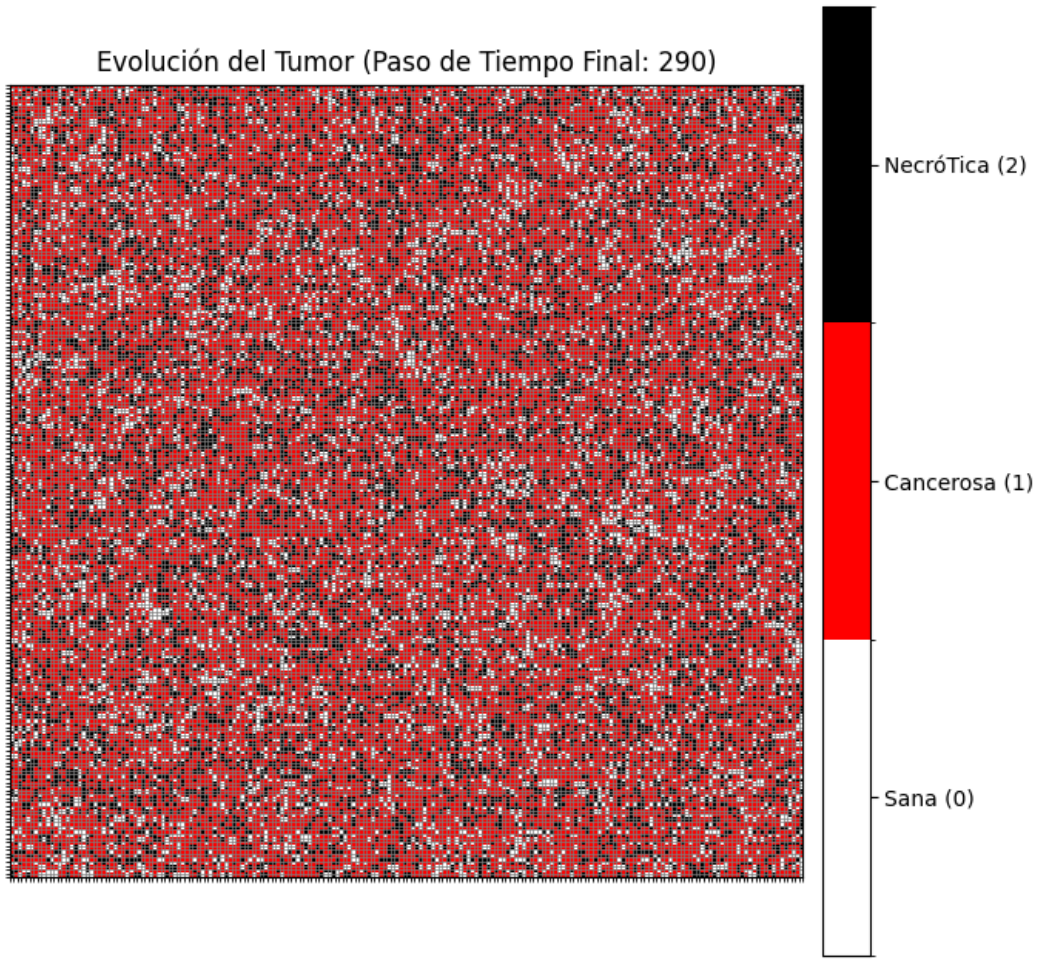

```
Generando imagen en el paso de tiempo: 270
Células activas: 33612
Densidad global: 0.8403
```



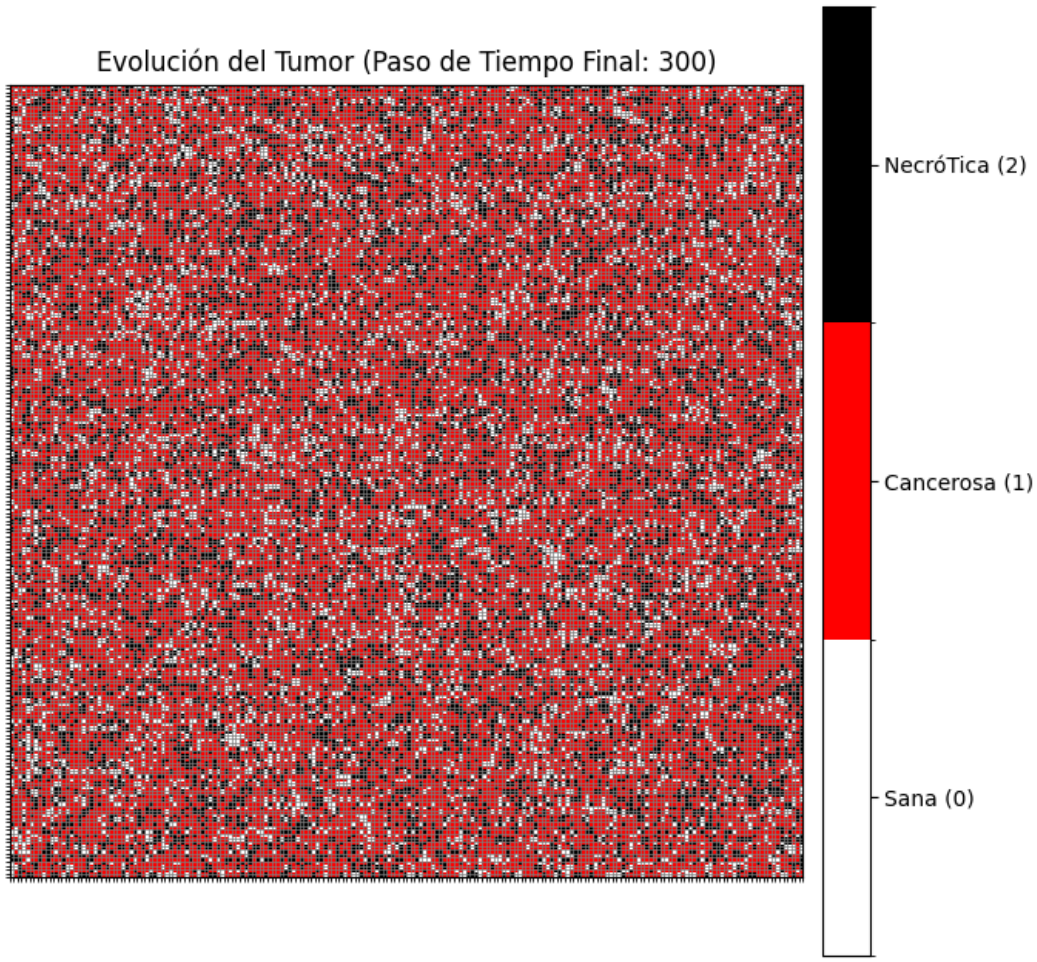

```
Generando imagen en el paso de tiempo: 280
Células activas: 33515
Densidad global: 0.8379
```



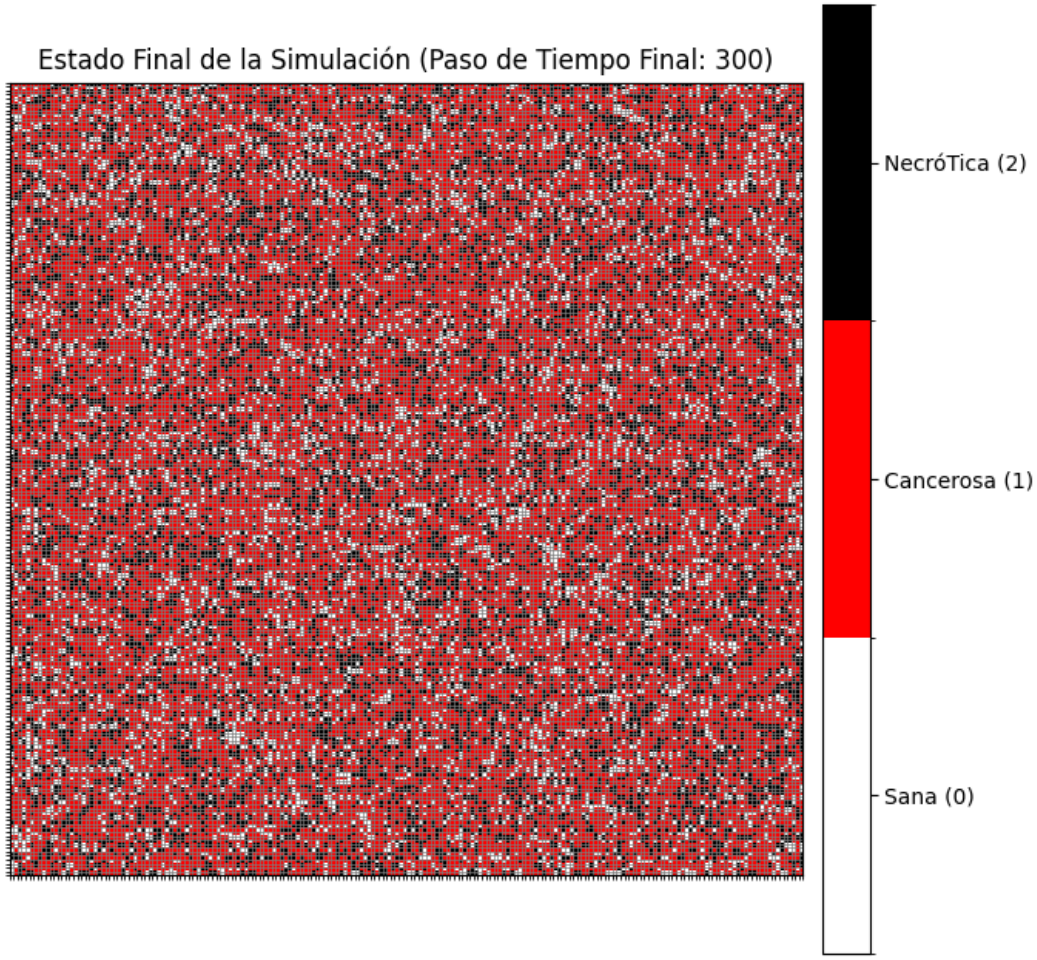

```
Generando imagen en el paso de tiempo: 290
Células activas: 33734
Densidad global: 0.8434
```

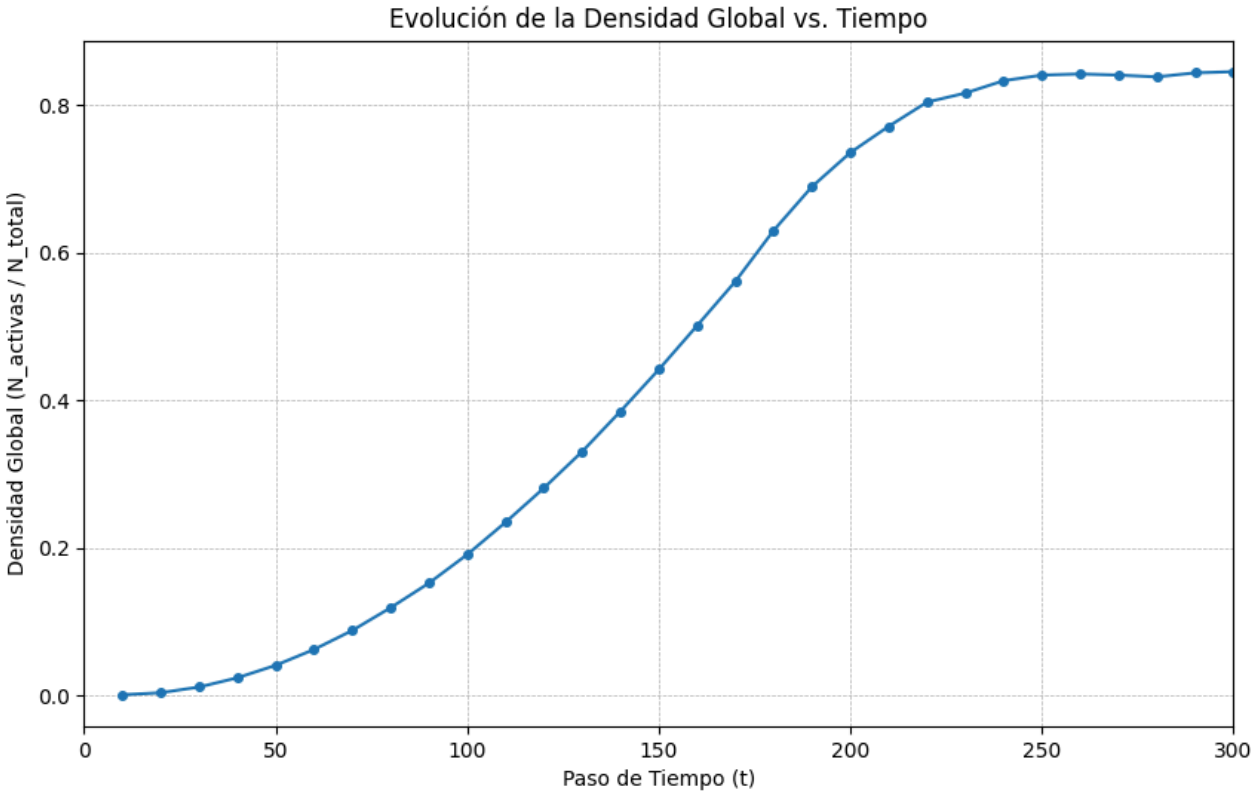


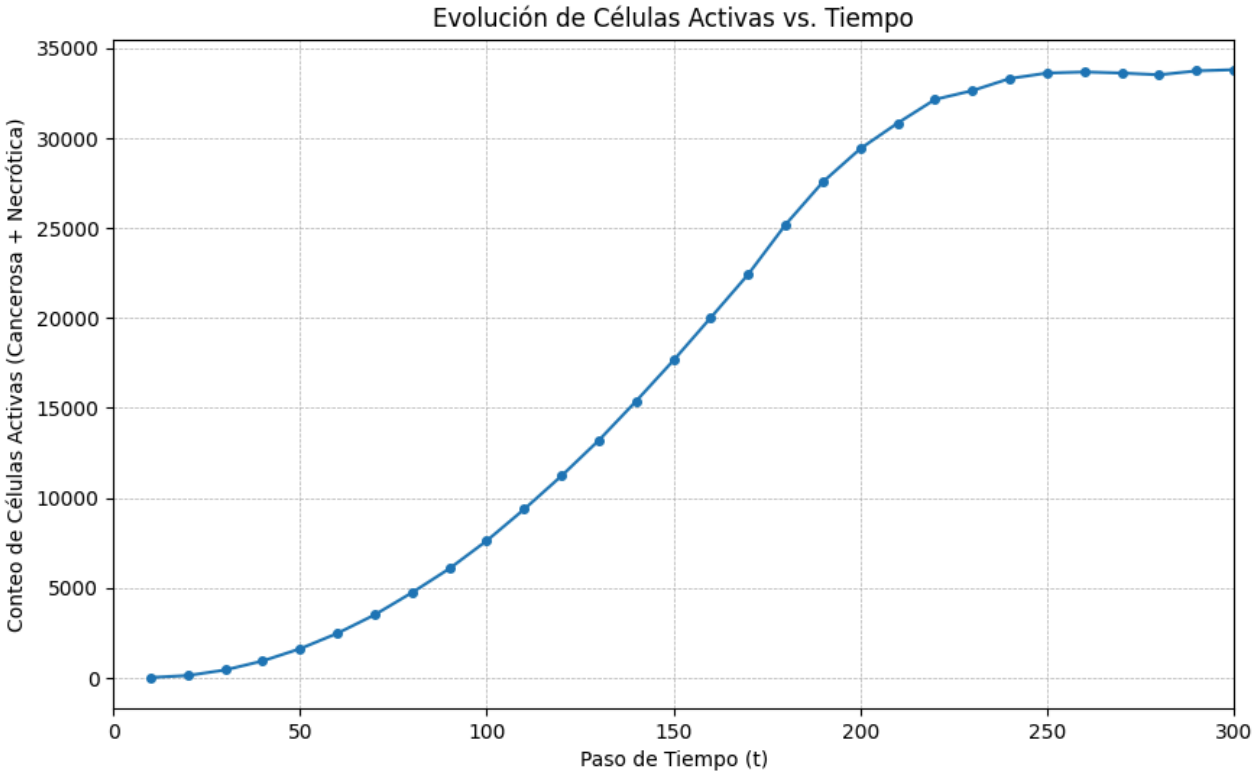

```
Generando imagen en el paso de tiempo: 300
Células activas: 33796
Densidad global: 0.8449
```




```
Simulación terminada. Generando imagen final...
```


Estado Final de la Simulación (Paso de Tiempo Final: 300)
NecróTica (2)
Cancerosa (1)
Sana (0)


Evolución de la Densidad Global vs. Tiempo
Densidad Global (N_activas / N_total)
0.0
0.2
0.4
0.6
0.8
0
50
100
150
200
250
300
Paso de Tiempo (t)


Evolución de Células Activas vs. Tiempo
Conteo de Células Activas (Cancerosa + Necrótica)
Paso de Tiempo (t)
0
5000
10000
15000
20000
25000
30000
35000
0
50
100
150
200
250
300