```
# Function to count the number of occupied neighbors for a given cell in the Metaverse
function count_neighbors(board: metaverse_t, row: integer, column: integer) -> integer
   # Initialize a variable to keep track of the count of occupied neighbors
   count = 0

   # Get the total number of rows and columns in the Metaverse grid
   rows = size(board)
   cols = size(board[0])

   # Define arrays for relative positions of neighboring cells
   dr = [-1, -1, -1, 0, 0, 1, 1, 1]  # Offsets for rows
   dc = [-1, 0, 1, -1, 1, -1, 0, 1]   # Offsets for columns

   # Iterate through the eight neighboring positions surrounding the specified cell
   for i from 0 to 7
      # Calculate the row and column indices of the neighboring cell
      newRow = row + dr[i]
      newCol = column + dc[i]

      # Check if the neighboring cell is within the valid grid boundaries
      if newRow >= 0 and newRow < rows and newCol >= 0 and newCol < cols
         # Check if the neighboring cell is occupied (contains a 'true' value)
         if board[newRow][newCol] is true
            # Increment the count of occupied neighbors
            count = count + 1

   # Return the final count, indicating the number of occupied neighbors
   return count

# Function to determine if a cell is occupied in the next generation
function occupied_in_next_tick(currently_occupied: boolean, neighbor_count: integer) ->
boolean
   if currently_occupied is true
      # If the cell is currently occupied, it will remain occupied in the next generation
      return neighbor_count is 2 or neighbor_count is 3
   else
      # If the cell is currently unoccupied, it will become occupied in the next generation
      return neighbor_count is 3

# Function to create the next generation of the Metaverse
function tick(currentGeneration: metaverse_t) -> metaverse_t
   # Get the total number of rows and columns in the current generation
   rows = size(currentGeneration)
   cols = size(currentGeneration[0])
   # Create an empty Metaverse for the next generation with the same dimensions
   nextGeneration = create a new metaverse_t of size (rows, cols) with all elements initialized to
false

   # Iterate through each cell in the current generation
   for row from 0 to rows - 1
      for col from 0 to cols - 1
         # Count the number of occupied neighbors for the current cell
         aliveNeighbors = count_neighbors(currentGeneration, row, col)

         # Determine if the current cell should be occupied in the next generation
         if currentGeneration[row][col] is true
            # If the cell is currently occupied, it remains occupied with 2 or 3 occupied neighbors
            if aliveNeighbors is 2 or aliveNeighbors is 3
               set nextGeneration[row][col] to true
         else
            # If the cell is currently unoccupied, it becomes occupied with exactly 3 occupied
neighbors
            if aliveNeighbors is 3
               set nextGeneration[row][col] to true

   # Return the next generation of the Metaverse
   return nextGeneration

# Function to resize a Metaverse according to a specified size
function resize_metaverse(rows: integer, board: reference to metaverse_t) -> boolean
   # Resize the Metaverse to the specified number of rows and columns
   board.resize(rows, create a new vector of booleans of size rows, with all elements initialized
to false)

   # Return true to indicate successful resizing
   return true

# Function to update Metaverse row based on a string of characters
function citizenship_row_to_metaverse_row(input_row: const reference to string, row: integer,
board: reference to metaverse_t) -> boolean
   # Check if the provided row number is within the valid range of the Metaverse
   if row is less than size(board) and length of input_row is equal to size(board[row])
      # Iterate through each character in the input_row
      for i from 0 to length of input_row - 1
         # Check if the character represents an occupied cell ('1')
         if input_row[i] is equal to '1'
            # Set the corresponding cell in the Metaverse to occupied (true)
            set board[row][i] to true
          else
            # Set the corresponding cell in the Metaverse to unoccupied (false)
            set board[row][i] to false

        # Return true to indicate successful update
        return true

    # Return false if the row or input is invalid, indicating a failed update
    return false

# Function to read and parse the configuration line from a Universe File
function read_metaverse_configuration_line_from_file(metaverse_file: reference to ifstream,
size: reference to integer, generations: reference to integer) -> boolean
    # Initialize variables to store size and generation values
    a = 0
    b = ' '
    c = 0

    # Read integers from the Universe File to extract size and generation values
    metaverse_file >> a
    metaverse_file >> b
    metaverse_file >> c

    # Update the size and generation variables with the extracted values
    size = a
    generations = c

    # Check if the extracted values are valid (size > 0 and generations >= 0)
    if size > 0 and generations >= 0
        # Return true to indicate a successful configuration read
        return true

    # Return false if the configuration read was not successful or if values are invalid
    return false

# Function to initialize the Metaverse from a Universe File
function initialize_metaverse_from_file(metaverse_file: reference to ifstream, metaverse:
reference to metaverse_t, generations: reference to integer) -> boolean
    # Initialize variables for size and the current row
    size = 0
    actual_row = 0

    # Reset the generations count
    generations = 0

    # Read the configuration line from the Universe File and update size and generations
    if not read_metaverse_configuration_line_from_file(metaverse_file, size, generations)
        return false  # Return false if the configuration read fails

    # Resize the Metaverse to the specified size
    if not resize_metaverse(size, metaverse)
        return false  # Return false if resizing fails

    # Skip any leading whitespace in the Universe File
    metaverse_file >> std::ws

    # Initialize a loop to read each line of the Metaverse from the Universe File
    while not metaverse_file.eof()
        # Initialize a string to store a line from the Universe File
        line = ""

        # Read a line from the Universe File
        if not getline(metaverse_file, line)
            return false  # Return false if reading a line fails

        # Update the Metaverse row based on the line and the current row number
        if not citizenship_row_to_metaverse_row(line, actual_row, metaverse)
            return false  # Return false if updating the Metaverse row fails

        # Increment the current row number
        actual_row++

        # Skip any additional leading whitespace in the Universe File
        metaverse_file >> std::ws

    # Check if the actual row count matches the specified size
    if actual_row is equal to size
        return true  # Return true to indicate a successful initialization

    return false  # Return false if the row count doesn't match the size

# Function to model the evolution of the Metaverse for a specified number of generations
procedure model_metaverse(starting_metaverse: const reference to metaverse_t, generations:
integer)
    # Initialize the currentMetaverse with the starting_metaverse
    currentMetaverse = starting_metaverse
```

```
# Loop through each generation
for generation from 0 to generations - 1
    # Display the current state of the Metaverse
    display_metaverse(cout, currentMetaverse)

    # Calculate the next generation Metaverse and update currentMetaverse
    currentMetaverse = tick(currentMetaverse)
end for
```