MY PSEUDOCODE – Lab 8 *METAVERSE*

– *Arnav A Bhalgat*

1. to calculate the number of neighbors of a given location safely (i.e., without accessing locations of the Metaverse that are either
    - too large or
    - too small);

Function count_neighbors(..) Return Type: int.

Initialize count to 0, this will store the number of neighbours.

Initialize a direction vector which will have maximum number of directions the function should check for the presence of neighbours.

For each direction in neighbouring directions. Calculate new_row and new_column

If new_row and new_column are within valid bounds of the board, check if the cell at (new_row, new_column) is occupied:

Increment count

Return count

2. to determine the occupancy of a location during a transition between generations; and

Function *occupied_in_next_tick(...)* ->Return type bool:

If currently_occupied and neighbor_count is 2 or 3:

Return true

Else if not currently_occupied and neighbor_count is 3:

Return true

Else:

Return false

3. to read in the initial occupancy of your Metaverse from a configuration file (see below).

    Function read_metaverse_configuration_line_from_file(...) -> Return Type bool:

    Initialise the appropriate variables (int size, generations; char comma) Using the ifstream method (inputfile>>var) Read size, comma, and generations from metaverse_file.

    If successful:

        Return true

    Else:

        Return false

4. Helper Functions to ensure smooth operation of the function.

    Function citizenship_row_to_metaverse_row(..) -> Return Type bool:

    For each character (loop) in input_row:

        If character is '1':

            Set corresponding cell in board to 1

        Else if character is '0':

            Set corresponding cell in board to 0

        Else:

            Return false

    Return true

    Function *resize_metaverse*(rows: int, board: metaverse_t) -> bool:

    Resize the board to have 'rows' number of rows and columns

    Return true

Function *tick(..)* -> Return Type metaverse_t:

Create a new metaverse for next tick with the same size as the current board - *resize_metaverse(..)*

For each cell in the board:

Count occupied neighbors for the cell - *count_neighbors(..)*

Determine if cell will be occupied in the next tick - *occupied_in_next_tick(...)*

Update corresponding cell in new metaverse

Return new metaverse

Function *model_metaverse*(...):

Set current_metaverse to starting_metaverse

For each generation from 0 to generations-1:

Display current_metaverse

Update current_metaverse by applying tick