```c
#include "decl.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/**
 * Returns TRUE if the specified row and col is within the grid of cells
 * @param row The current row
 * @param col The current col
 * @param maxRow The maximum row value
 * @param maxCol The maximum col value
 * @return TRUE if row and col is within the grid of cells
 */
static bool isInRange(int row, int col, int maxRow, int maxCol)
{
    return (row >= 0 && row < maxRow) && (col >= 0 && col < maxCol);
}

/**
 * Returns TRUE if the cell located at the specified row and column is alive
 * @param state The game grid
 * @param rowOffset The specified row
 * @param colOffset The specified column
 * @return TRUE if the cell located at the specified row and column is alive
 */
static bool isAlive(char *state, int rowOffset, int colOffset)
{
    return *(state + rowOffset + colOffset) == ALIVE_CELL;
}

/**
 * Counts the number alive cells adjacent to the specified cell.
 * @param row The row of the cell.
 * @param col The column of the cell.
 * @param maxRow The maximum row index.
 * @param maxCol The maximum col index.
 */
static int countAdjacentAliveCells(char *state, int row, int col, int maxRow, int maxCol)
{
    int rowOffset = 0, colOffset = 0, aliveCount = 0;
    char charAtOffset;

    /*This iterates over every adjacent cell in a 3x3 by offsetting the current cell*/
    for (int i = -1; i <= 1; i++) {
        for (int j = -1; j <= 1; j++) {
            /*If i and j are both 0, there is no offset so skip it*/
            if(i == 0 && j == 0) {continue;}

            if (isInRange(row + i, col + j, maxRow, maxCol)) {
                rowOffset = (row + i) * maxRow;
                colOffset = col+j;
                charAtOffset = *(state + rowOffset + colOffset);

                if (isAlive(state, rowOffset, colOffset)){
                    aliveCount++;
                }
            }
        }
    }
    return aliveCount;
}

/**
 * This method advances the state of the board and returns true if there is another
 generation
 * to occur. That is, not every cell is dead.
 * @param state The current state.
 * @param nextState the next state of the grid.
 * @param rows The number of rows in the grid.
 * @param cols The number of columns in the grid.
 * @return TRUE if there are alive cells in the next generation.
 */
bool generations(char *state, char *nextState, int rows, int cols)
{
    int aliveCount = 0;
    bool hasAliveCells = FALSE;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            aliveCount = countAdjacentAliveCells(state, i, j, rows, cols);

            if (aliveCount > 0) {hasAliveCells = TRUE;}
            *(nextState + rows * i + j) = *(state + rows * i + j);

            if (isAlive(state, rows * i, j)) {
                if (aliveCount < 2 || aliveCount > 3) {
                    *(nextState + rows * i + j) = DEAD_CELL;
                }
            }
            else {
                if (aliveCount == 3) {
                    *(nextState + rows * i + j) = ALIVE_CELL;
                }
            }
        }
    }
    return hasAliveCells;
}
```