```cpp
//Jung Kim
//CS-102 Programming II
//Professor Charles Hamaker
//Project: Game of Life

#include <iostream>
#include <stdlib.h>
#include <windows.h>

using namespace std;

const int world_column = 70, //These constants set the dimensions of the 2 dimensional grid
          world_row = 25;

const char life = '*', //Life is represented by *'s
           death = ' ', //Death is an empty space
           limbo = 'x'; //Limbo is the temporary state of the neighbors during evaluation from life or death

class Generation //Decided to do object-oriented programming for the Game of Life for the sake of convenience
{
public:
    void createWorld(), //create is the same as set, createWorld() sets the 2D space
         createLife(), //sets up the initial configuration of the World
         showLife(), //display function
         startGame(), //begins the game of life
         rules_of_the_game(); //self-explanatory, concerns evaluating neighbors of each cell and shifts from temp world and real world to make the changes

private:
    char world[world_column][world_row], //the world and the neighbors are private
         world_temp[world_column][world_row],
         neighbors[8];
};

void Generation::createWorld() //creates the 2D world, everything starts dead until life is created
{
    for (int row = 0; row < world_row; row++)
        for (int column = 0; column < world_column; column++)
        {
            world[column][row] = death;
            world_temp[column][row] = death;
        }

    for (int row = 0; row < world_row; row ++) //these are borders that never change, could have made them const but opted to not
        {                                      //these borders are both for visual aesthetics and ensuring living cells do not go past the grid
            world[0][row] = '|';
            world[world_column - 1][row] = '|';
        }

    for (int column = 0; column < world_column; column++) //the borders are placed in the 0th and the last index, inputting coordinates during createLife into these points will do nothing.
        {
            world[column][0] = '_';
            world[column][world_row - 1] = '-';
        }
}

void Generation::createLife() //creates life as in the user is given the option to input the starting cells
{
    int birth_row, //coordinates for the cells the user inputs
        birth_column,
        user_input; //the user's answer to prompted questions throughout the setup is stored here
```

```
61
62      showLife();
63      cout << "*********************************************************" << endl
64           << "************** Welcome to the Game of Life **************" << endl
65           << "******* To begin, initialize the first generation *******" << endl
66           << "*** Initialization will require two coordinate points ***" << endl
67           << "*********************************************************" << endl
68           << endl; //introduction
69
70      do //this do-while loop allows user to either start the game or continue adding cells
71      {
72          do //this do-while loop allows user to confirm or change last inputted cell coordinate
73          {
74              cout << "The current grid is " << world_column << " x " << world_row << " cells.\nPlease keep
the coordinates within the boundaries." << endl
75                   << "Enter the coordinates of the cell to give it Life: " << endl
76                   << "Column: ";
77              cin >> birth_column;
78              if (birth_column > world_column)
79              {
80                  birth_column = world_column - 2;
81                  cout << "A coordinate point that exceeded the boundaries of this World was entered." <<
endl
82                       << "The point has been defaulted into " << world_column - 2 << endl;
83              }
84
85              cout << "Row: ";
86              cin >> birth_row;
87              if (birth_row > world_row)
88              {
89                  birth_row = world_row - 2;
90                  cout << "A coordinate point that exceeded the boundaries of this World was entered." <<
endl
91                       << "The point has been defaulted into " << world_row - 2 << endl;
92                  Sleep(3000);
93              }
94
95              world[birth_column][birth_row] = life;
96              system("CLS"); //clears and refreshes the grid to show the cell the user just resurrected
97              showLife();
98
99              cout << "The coordinate you entered was (" << birth_column << ", " << birth_row << ")" << "
correct?" << endl
100                  << "Enter \'1\' to confirm \'Yes\'. Enter any other digit to re-enter the coordinates." <<
endl
101                  << "Do NOT enter anything other than an integer." << endl; //entering a non-integer will
result in the program crashing.
102             cin >> user_input; //prompts user to confirm or change coordinates
103
104             if (user_input != 1) //if the user changes coordinates the cell is killed and grid is refreshed
105             {
106                 world[birth_column][birth_row] = death;
107                 system("CLS");
108                 showLife();
109             }
110         } while (user_input != 1);
111
112         world[birth_column][birth_row] = life; //might be unnecessary
113         system("CLS");
114         showLife();
115
116         cout << "Life has been given to the cell located in point (" << birth_column << ", " << birth_row
<< ") of World." << endl;
117         cout << "Would you like to give Life to more cells?" << endl
118              << "Enter \'1\' to confirm \'Yes\'. Enter any other digit to start the Game of Life." << endl
119              << "Do NOT enter anything other than an integer." << endl;
```

```
120         cin >> user_input;
121     } while (user_input == 1);
122 }
123
124 void Generation::showLife() //simple display function
125 {
126     for (int row = 0; row < world_row; row++)
127         for (int column = 0; column < world_column; column++)
128             if (column == (world_column - 1)) //if condition to make sure that the console prints on a new
line if loop goes past last column of the current row
129                 cout << world[column][row] << endl;
130             else
131                 cout << world[column][row];
132 }
133
134 void Generation::rules_of_the_game() //4 rules of the game, each concerning neighboring cells
135 {
136     int neighbor_counter = 0; //by counting how many neighbors, rule conditions are fulfilled
137     for (int row = 1; row < world_row; row++)
138         for (int column = 1; column < world_column; column++) //since the 0th index are borders and are not
cells that can live/die the for loops start at 1
139         {
140             neighbors[0] = world[column][row - 1];
141
142             if (row < (world_row - 1)) //these if statements just ensure that index is not out of bounds
143                 neighbors[1] = world[column][row + 1];
144             else
145                 neighbors[1] = limbo;
146
147             neighbors[2] = world[column - 1][row]; //the neighbors are cells directly above, below, to the
left, right, top left, top right, bottom left and bottom right of the cell
148
149             if (column < (world_column - 1))
150                 neighbors[3] = world[column + 1][row];
151             else
152                 neighbors[3] = limbo;
153
154             neighbors[4] = world[column - 1][row - 1];
155
156             if (column < (world_column - 1))
157                 neighbors[5] = world[column + 1][row - 1];
158             else
159                 neighbors[5] = limbo;
160
161             if (row < (world_row - 1))
162                 neighbors[6] = world[column - 1][row + 1];
163             else
164                 neighbors[6] = limbo;
165
166             if ((row < world_row - 1) && (column < world_column - 1))
167                 neighbors[7] = world[column + 1][row + 1];
168             else
169                 neighbors[7] = limbo;
170
171             for (int i = 0; i < 8; i ++) //counts all the neighbor cells of the cell under evaluation that
are alive
172                 if (neighbors[i] == life)
173                     neighbor_counter++;
174
175             if (world[column][row] == life) //the first 3 rules require the cell under evaluation to be
alive
176             { //everything is set to a temporary world so that all the changes can be made
'instantaneously' to prevent cells from new generation affecting cells of older generations
177                 if (neighbor_counter < 2)           //first rule - under population
178                     world_temp[column][row] = death;
179                 else if (neighbor_counter > 3)      //third rule - over-population
```

```
180                     world_temp[column][row] = death;
181                 else
182                     world_temp[column][row] = world[column][row]; //second rule - survival
183             }
184 
185             else if (world[column][row] == death) //the last rule require the cell under evaluation to be
dead
186                 if (neighbor_counter == 3)          //fourth rule - birth
187                     world_temp[column][row] = life;
188 
189             neighbor_counter = 0; //resets the count of alive neighbors back to 0 to prevent the next cell
from having incorrect number of living neighbors
190             for (int i = 0; i < 8; i ++)
191                 neighbors[i] = limbo; //also sets all the neighbors to a limbo state for a new evaluation
192 
193             for (int row = 0; row < world_row; row ++) //ensures that borders remain as borders
194             {
195                 world_temp[0][row] = '|';
196                 world_temp[world_column - 1][row] = '|';
197             }
198 
199             for (int column = 0; column < world_column; column++)
200             {
201                 world_temp[column][0] = '_';
202                 world_temp[column][world_row - 1] = '-';
203             }
204         }
205     for (int row = 0; row < world_row; row++) //the changes are now made onto the 'real' world so that all
the cells are of the same generation
206         for (int column = 0; column < world_column; column++)
207             world[column][row] = world_temp[column][row];
208 }
209 
210 void Generation::startGame() //how the game is run
211 {
212     createWorld();
213     createLife();
214     for (int i = 0; i < 5000; i++) //set the loop to 5000 times for now
215     {
216         rules_of_the_game();
217         system("CLS");
218         Sleep(10);
219         showLife();
220     }
221 }
222 
223 int main()
224 {
225     Generation gameOfLife; //creates the class
226     gameOfLife.startGame(); //starts the game
227     return 0;
228 }
```