Kelvin Watson
OSU ID 932540242

CS 162 ASSIGNMENT 1 REPORT

Requirements

In this assignment, I am being asked to design and implement a version of Conway's Game of Life using concepts I have learned in CS 161 and thus far in CS 162. In order to write this program, I will need to perform the following requirements:

- practice and become accustomed to designing before performing any coding
- make use of 2-dimensional arrays, use 2D arrays as arguments by passing them into functions
- dynamically allocate parallel 1D arrays for storing user-inputted coordinates
- incorporate input validation for user-inputted coordinates
- use the atoi() function to convert char's to int's
- write functions to perform the following actions:
    - parse C-style strings for input validation
    - save coordinates into dynamically-allocated arrays because the number of coordinates provided by the user is not known until runtime
    - determine which cells are "live" or "dead" after each generation depending on the states of the cells in the previous generation, update the cells, and display the results to the user
- write a makefile
- test my program, record and discuss the results of the testing

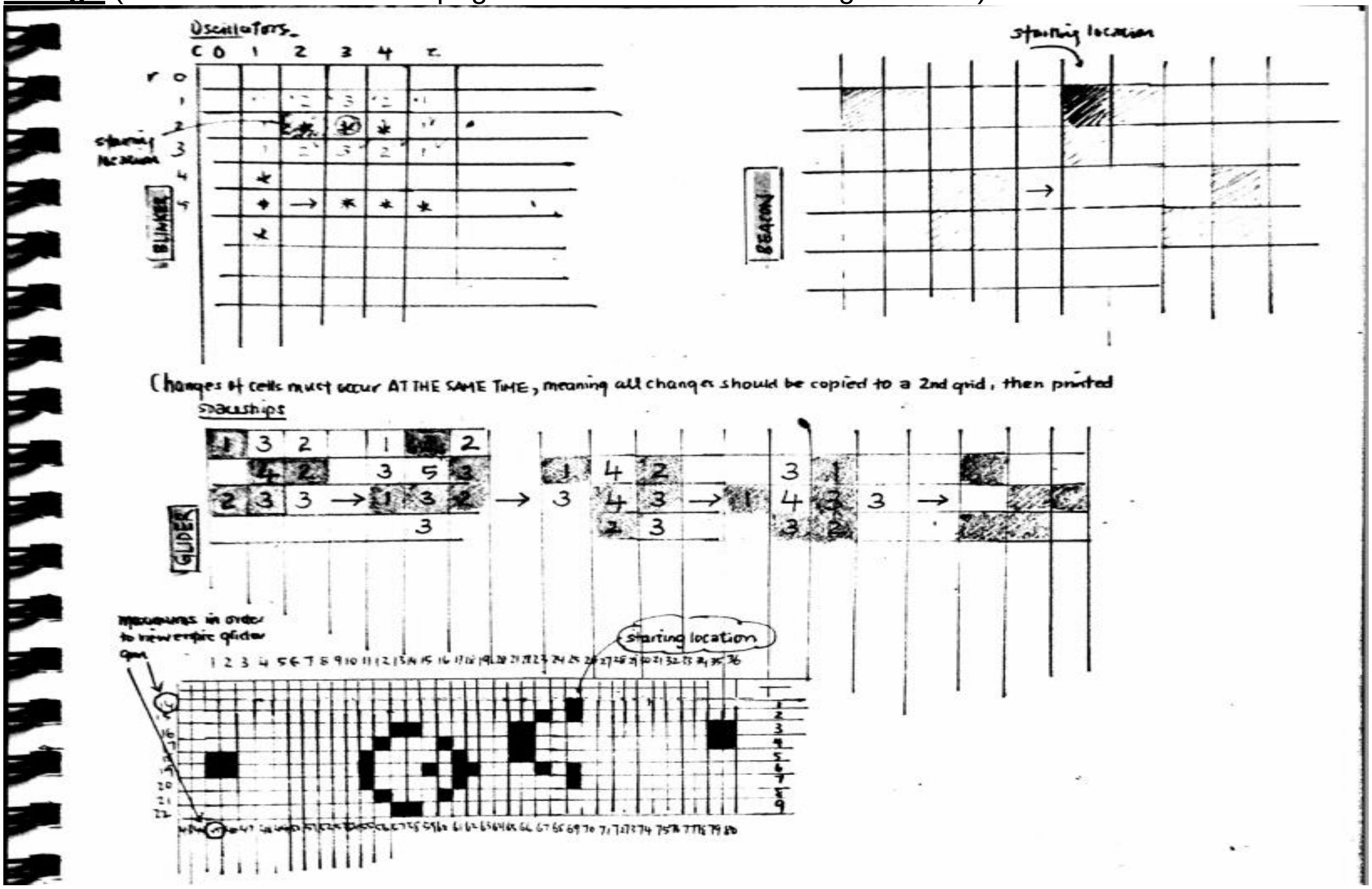
Assumptions:

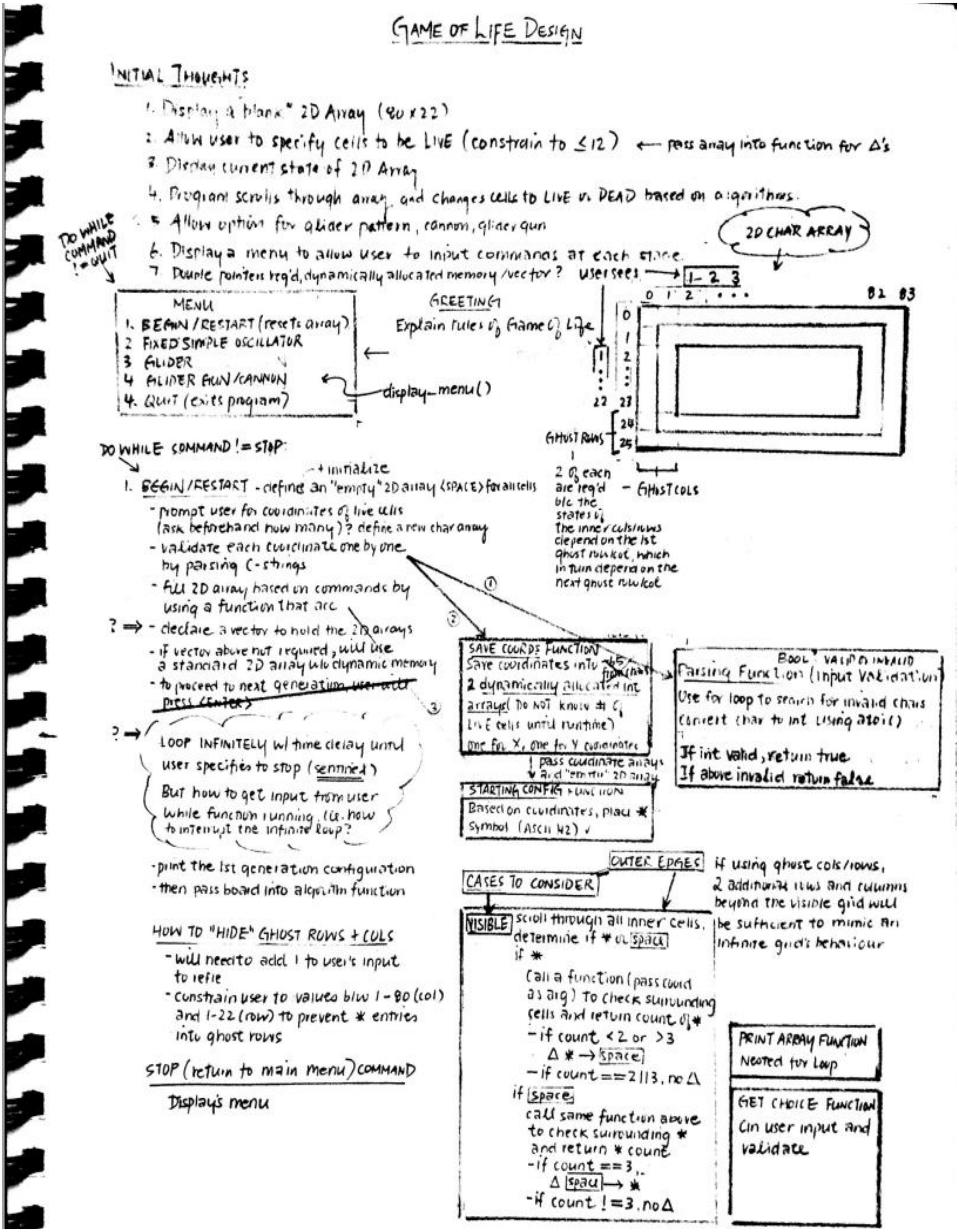
- I am assuming that two ghost rows and columns are sufficient for the program to mimic the behavior of an infinite grid.
- As discussed under "Uncertainties" below, I assumed that I do not need to save previous generation configurations, and that I can overwrite the 2D array's configuration with each new generation.

Uncertainties:

- The assignment instructions say to "allow the user to see the change(s) in the pattern." It is therefore unclear whether or not I am supposed to allow the user to scroll through previous generations.

Design (Please also see next 2 pages for continuation of Design section)


Oscillators
c 0 1 2 3 4 5
r 0 1 2 3 4 5
starting location
BLINKER
starting location
BEACON
Changes of cells must occur AT THE SAME TIME, meaning all changes should be copied to a 2nd grid, then printed
Spaceships
GLIDER
Movements in order to view entire glider gun
starting location



GAME OF LIFE DESIGN

INITIAL THOUGHTS

1. Display a "blank" 2D Array (80 x 22)
2. Allow user to specify cells to be LIVE (constrain to ≤12) ← pass array into function for Δ's
3. Display current state of 2D Array
4. Program scrolls through array, and changes cells to LIVE or DEAD based on algorithms.
5. Allow option for glider pattern, cannon, glider gun
6. Display a menu to allow user to input commands at each stage.
7. Double pointers req'd, dynamically allocated memory / vector?

DO WHILE COMMAND != QUIT

MENU
1. BEGIN / RESTART (resets array)
2 FIXED SIMPLE OSCILLATOR
3 GLIDER
4 GLIDER GUN / CANNON
4. QUIT (exits program)

display_menu()

GREETING
Explain rules of Game of Life

2D CHAR ARRAY

user sees → 1- 2 3

0 1 2 . . . 82 83

0 1 2 : : 22 23 24 25

1 : : 22

GHOST ROWS

GHOST COLS

2 of each are req'd b/c the states of the inner cols/rows depend on the 1st ghost row/col, which in turn depend on the next ghost row/col

DO WHILE COMMAND != STOP:

1. BEGIN / RESTART - define an "empty" 2D array (SPACE) for all cells + initialize
   - prompt user for coordinates of live cells (ask beforehand how many)? define a new char array
   - validate each coordinate one by one by parsing C-strings
   - fill 2D array based on commands by using a function that acc
   - ? ⇒ - declare a vector to hold the 2D arrays
   - if vector above not required, will use a standard 2D array w/o dynamic memory
   - to proceed to next ~~generation, user enter press (ENTER)~~

? → LOOP INFINITELY w/ time delay until user specifies to stop (sentinel)
But how to get input from user while function running (i.e. how to interrupt the infinite loop?

- print the 1st generation configuration
- then pass board into algorithm function

(1) (2) (3)

SAVE COORDS FUNCTION
Save coordinates into ~~2D~~ from char 2 dynamically allocated int arrays (DO NOT know # of LIVE cells until runtime)
one for X, one for Y coordinates

pass coordinate arrays and "empty" 2D array

STARTING CONFIG FUNCTION
Based on coordinates, place * symbol (ASCII 42) ✓

BOOL VALID or INVALID
Parsing Function (input Validation)
Use for loop to search for invalid chars
Convert char to int using atoi()

If int valid, return true
If above invalid return false

HOW TO "HIDE" GHOST ROWS + COLS
- will need to add 1 to user's input to refle
- constrain user to values b/w 1-80 (col) and 1-22 (row) to prevent * entries into ghost rows

STOP (return to main menu) COMMAND
Display's menu

CASES TO CONSIDER

OUTER EDGES

If using ghost cols/rows, 2 additional rows and columns beyond the visible grid will be sufficient to mimic an infinite grid's behaviour

```
VISIBLE  scroll through all inner cells,
         determine if * or space
         if *
            call a function (pass coord
            as arg) To check surrounding
            cells and return count of *
            - if count < 2 or > 3
                 Δ * → space
            - if count == 2||3, no Δ
         if space
            call same function above
            to check surrounding *
            and return * count
            - if count == 3,
                 Δ space → *
            - if count != 3, no Δ
```

PRINT ARRAY FUNCTION
Nested for loop

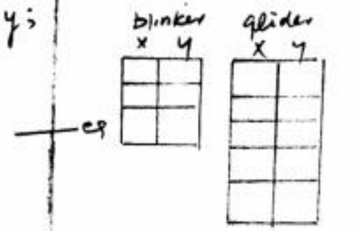
GET CHOICE FUNCTION
Cin user input and validate

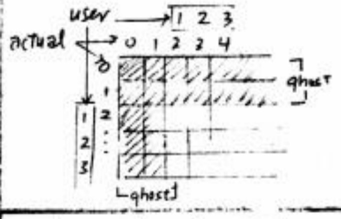
Attributes

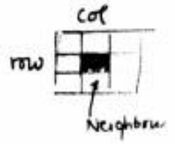
CONFIGURATION

```
static const int ACTUAL_MAX_COL = 81
static const int ACTUAL_MAX_ROW=23
static const int ACTUAL_MIN_COL =2
static const int ACTUAL_MIN_ROW=2
```

```
starting_x : int        actual_starting_x;
starting_y : int        actual_starting_y;
seconds : int   (for delay)
iterations : int
grid [ROWS][COLS];
```







Behaviors

CONSTRUCTORS

```
                      default
Configuration ( ) {   }

Configuiation (starting_x , starting_y,
               num_coordinates )
      this→ starting_x = starting_y;
      this→ starting_y = starting_y;
      actual_starting_X = starting_x + 1;
      actual_starting_y = starting_y + 1;
      set_live_cells (num_coordinates);
```

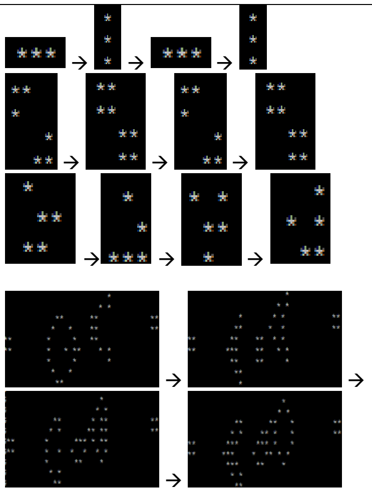
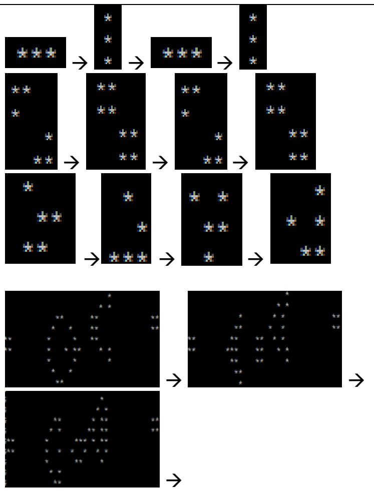
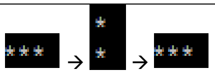
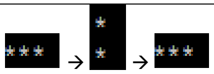
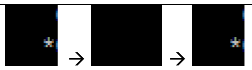
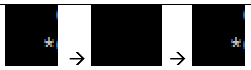
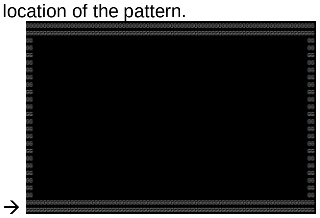
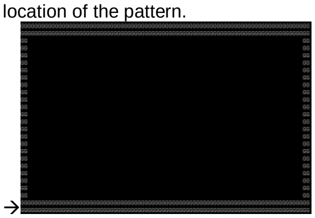
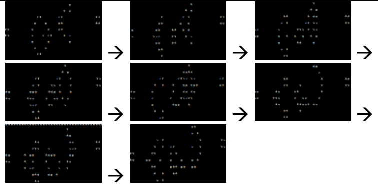
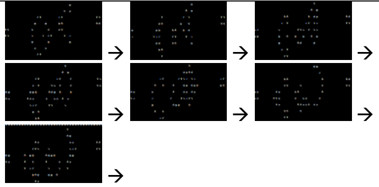
```
get_neighbours (grid, live_cell_x, live_cell_y)
  //check neighbours and increment count
  int count = 0;        — 8 neighbours
  if grid[live_cell_x-1][live_cell_y]=='*', count++;
  if grid[live_cell_x+1][live_cell_y]=='*', count ++;
  if grid [live_cell_x][live_cell_y-1]=='*', count ++;
  if grid [live_cell_x][live_cell_y+1]=='*', count ++;
  if grid[live_cell_x-1][live_cell_y-1]=='*', count ++;
  if grid[live_cell_x+1][live_cell_y-1]=='*', count ++;
  if grid[live_cell_x-1][live_cell_y+1]=='*', count++;
  if grid[live_cell_x+1][live_cell_y+1]=='*', count++,
  return count;
```

BEHAVIORS

```
set_live_cells(int num_coordinates, grid, actual x, actual y)
    if (num_coordinates == 3)   ←—— BLINKER!
       //set live cells for blinker                    .tic int!
0 ▨     grid [actual_starting_x][actual_starting_y] = '*';    ⌉ use for loop instead
1 ▨     grid [actual_starting_x + 1 ][actual_starting_y]='*';
2 ▨     grid [actual_starting_x + 2] [actual_starting_y]='*';
       algorithm (grid);

algorithm(grid, seconds iterations,                          continue_iterating(
for(int i = 0, i < iterations; i++)                            algorithm(
   for (int row = ACTUAL_MIN_ROW; row ≤ ACTUAL_MAX_ROW; row++)
     for(int col = ACTUAL_MIN_COL; col ≤ ACTUAL_MAX_COL; col++)
       if   grid[row][col]=='*', live_neighbours = get_neighbors();
         if live_neighbors < 2 or >3 . Δ from '*' → [space]
         else if live_neighbors == 2 or 3 , no Δ
       else if grid[row][col] == [space], live_neighbours = get_neighbors;
         if live_neighbors == 3 Δ from [space] → '*'
         else if live_neighbors != 3 , no Δ
   print_grid();
   time_delay();
```

Testing (Continues onto next page)

Test Case	Input	Expected Output	Actual Output	Discussion of Output
VALID COMMANDS AND INPUT: All pattern within the bounds of the visible grid, and all input (seconds, generations, starting location) valid	Command? blinker, then beacon, then glider, then glider gun Number of generations? 2 Delay (in seconds) between generation output? 1 Starting row? 3 Starting column? 10 (25 for glider gun) Continue? Y Number of generations? 1 Continue? stop			Program accepts valid strings as commands, and integers within the specified ranges as valid input, and continues to iterate when prompted. These outcomes occurred as expected.
MINIMUM ROW AND COLUMN: To look for truncation of pattern (edge case)	Command? blinker Number of generations? 2 Delay (in seconds) between generation output? 1 Starting row? 1 Starting column? 1 Continue? Y			In the vertical configuration of the blinker, one of the live cells (asterisk) is in the ghost cell, so the configuration appears truncated. This outcome is expected.
MAXIMUM ROW AND COLUMN: To look for proper truncation of blinker (edge case)	Command? blinker Number of generations? 2 Delay (in seconds) between generation output? 1 Starting row? 1 Starting column? 1 Continue? Y			Only the left-most live cell (asterisk) shows up because the rest of the configuration is in the ghost cells.This outcome is expected.
MAXIMUM ROW AND COLUMN: To look for proper disappearing behavior of glider (edge case)	Command? glider Number of generations? 5 Delay (in seconds) between generation output? 1 Starting row? **22** Starting column? **80**	 Although difficult to see, there is an asterisk (indicating a live cell) at the bottom left hand corner), which is the upper-left-most corner of the glider pattern, which is the starting location of the pattern.  The glider disappeared off the edge in the next generation.	 Although difficult to see, there is an asterisk (indicating a live cell) at the bottom left hand corner), which is the upper-left-most corner of the glider pattern, which is the starting location of the pattern.  The glider disappeared off the edge in the next generation.	The starting location of the each pattern in this program is the upper-left-most occupied cell of the pattern, so a starting location of row 22 and column 80 would result in a glider with only the top-left occupied cell showing. The glider disappears off the screen as expected.
CONTINUING AND STOPPING ITERATIONS: Testing the continue and stop commands	Command? gun Number of generations? 5 Delay (in seconds) between generation output? 2 Starting row? 1 Starting column? 25 Continue? stop Continue? Y Number of generations? 2			Program continues to iterate, and stops iterating as expected. Total number of generations displayed is correct.

<table>
<tr><th>Test Case</th><th>Input</th><th>Expected Output</th><th>Actual Output</th><th>Discussion of Output</th></tr>
<tr><td>INVALID COMMANDS: Testing invalid commands, then valid commands</td><td>Command? hello
Command? world28
Command? gun</td><td>Invalid selection. Please re-enter menu selection:
Invalid selection. Please re-enter menu selection:
You have selected the GLIDER GUN:
Please specify the number of generations:</td><td>Invalid selection. Please re-enter menu selection:
Invalid selection. Please re-enter menu selection:
You have selected the GLIDER GUN:
Please specify the number of generations:</td><td>Input rejected twice, then accepted as expected.</td></tr>
<tr><td>GENERATIONS INVALID AND OUT OF BOUNDS: Number of generations out of bounds and invalid generations, then valid generations</td><td>Number of generations? 9999999, then -11, then A1, then 20</td><td>Invalid number of generations (must be between 1 and 9999). Please specify the number of generations: Invalid number of generations (must be between 1 and 9999).
Please specify the number of generations:
Invalid number of generations. Please specify the number of generations:
Please specify the delay in seconds (range 1-10) between generation output:</td><td>Invalid number of generations (must be between 1 and 9999). Please specify the number of generations: Invalid number of generations (must be between 1 and 9999).
Please specify the number of generations:
Invalid number of generations. Please specify the number of generations:
Please specify the delay in seconds (range 1-10) between generation output:</td><td>Input validated as expected.</td></tr>
<tr><td>SECONDS OUT OF BOUNDS, AND INVALID: Number of seconds out of bounds, then valid seconds.</td><td>Command? beacon
Number of generations? 0, then A1, then 2</td><td>Invalid number of seconds (range 1-10).
Please specify the delay in seconds (range 1-10) between generation output.
Invalid number of seconds. Please specify the delay in seconds (range 1-10) between generation output:</td><td>Invalid number of seconds (range 1-10).
Please specify the delay in seconds (range 1-10) between generation output.
Invalid number of seconds. Please specify the delay in seconds (range 1-10) between generation output:</td><td>Input validated as expected.</td></tr>
<tr><td>TESTING RULES AND QUIT COMMANDS: Testing a command (quit) after the rules command</td><td>Command? rules
Command? quit</td><td>RULES:
1. If an occupied cell has <2 neighbors, it dies of loneliness.
2. If an occupied cell has >3 neighbors, it dies of overcrowding.
3. If an occupied cell has 2 or 3 neighbours, it survives to the next generation.
4. If an unoccupied cell has 3 occupied neighbours, it becomes occupied.
<table><tr><td>COMMAND</td><td>DESCRIPTION</td></tr><tr><td>blinker</td><td>Blinker pattern (oscillator)</td></tr><tr><td>beacon</td><td>Beacon pattern (oscillator)</td></tr><tr><td>glider</td><td>Glider pattern (spaceship)</td></tr><tr><td>gun</td><td>Glider gun pattern</td></tr><tr><td>rules</td><td>Show game rules</td></tr><tr><td>quit</td><td>Exit program</td></tr></table>
Please enter a command from the menu above: quit
You are exiting the Game of Life program. Goodbye.</td><td>RULES:
1. If an occupied cell has <2 neighbors, it dies of loneliness.
2. If an occupied cell has >3 neighbors, it dies of overcrowding.
3. If an occupied cell has 2 or 3 neighbours, it survives to the next generation.
4. If an unoccupied cell has 3 occupied neighbours, it becomes occupied.
<table><tr><td>COMMAND</td><td>DESCRIPTION</td></tr><tr><td>blinker</td><td>Blinker pattern (oscillator)</td></tr><tr><td>beacon</td><td>Beacon pattern (oscillator)</td></tr><tr><td>glider</td><td>Glider pattern (spaceship)</td></tr><tr><td>gun</td><td>Glider gun pattern</td></tr><tr><td>rules</td><td>Show game rules</td></tr><tr><td>quit</td><td>Exit program</td></tr></table>
Please enter a command from the menu above: quit
You are exiting the Game of Life program. Goodbye.</td><td>Output as expected.</td></tr>
</table>

Implementation

Please see source files (functions.cpp, configuration.cpp, main.cpp)

Reflection

My original design had to be revised twice. The first time I redesigned my program was due to my misunderstanding of the requirements of the assignment. I discovered that the user did not have to be able to specify 12 coordinates for the pattern. Later, I revised my design a second time because I realized I had not considered the object-based programming approach and its potential advantages to this assignment[1]. Prior to coding, I decided to change my design to incorporate the configuration class. Instead of having a separate function for each pattern, each instance of the configuration class (pattern) can have its own attributes (starting location, initial configuration, number of generations, time delay) and behaviors, which made the program more organized.

Assumptions

For this assignment, I assumed that a time delay between output of generations was needed in order to view the changes between generations. Doing so resulted in the “feel” of an animation.

Another assumption I made was for the `clock()` function and `clock_t` typedef (from the `<ctime>` library). Because clock ticks are of system-specific length, whether the delay will be accurate was (and still is) of concern.

I also felt the need to constrain the user to certain coordinates for the glider gun in order for the user to have a full view of the gun portion of the configuration. Otherwise, part of the view of the gun would be “off the screen.” I did not constrain the user to certain coordinates for the other configurations because at least one asterisk of the blinker and beacon would always be seen, and the glider must slide off the screen as described in the assignment’s requirements.

Difficulties encountered

During design, I had some trouble figuring out how many “ghost” cells to incorporate in order to allow the inner cells to behave according to the Game of Life rules. After some trial and error on pencil and paper, I concluded that two additional “ghost” rows were required for the horizontal edges, and two additional columns were required for the vertical edges. One additional row on the horizontal edges and one additional column for the vertical edges was insufficient and resulted in erroneous patterns. Although it took a few hours of sketching on paper, it was not as difficult as I had anticipated.

When I started testing the program, I started getting segmentation faults, and unpredictable errors, such the values of member variables changing on subsequent iterations through my `for` loops. Member variables were printing out as different values on cycling through subsequent iterations in for loops, which led me to think that variables were being overwritten. I eventually discovered that the `ROWS` and `COLS` constants were defined as 25 and 83 respectively, causing memory to be accessed outside of the bounds of the 2D arrays, and subsequently memory to be overwritten in unpredictable places and bugs that were difficult to locate. I corrected the `ROWS` to 26 and the `COLS` to 84 to account for the two additional ghost rows for each vertical edge, and two additional ghost columns for each horizontal edge.

Making sense of the output from the implementation

I found the Conway’s Game of Life articles on Wikipedia[4,6] quite helpful as they provided animations of some fixed oscillators, the glider, and the glider gun. They were a great resource for checking that the output patterns were correct.

References

1. http://www.cplusplus.com/reference/ctime/clock/
2. http://www.cplusplus.com/forum/unices/10491/
3. http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
4. http://www.tech.org/~stuart/life/rules.html
5. http://www.conwaylife.com/wiki/Gun
6. http://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/makefile.html