Cellular Automata Project

GHHS Intro CS

Background

Cellular Automata, invented in the 1970s, create simulated worlds where "cells" interact with the cells around them using simple rules. From these rules, complex behaviors can arise. Even today, new discoveries are made. Maybe you will discover something new!

In this project, you will implement a cellular automata runtime in SNAP. We will give you some of the code and some blocks that you need to implement yourself. There are also three extra components; you need to choose one and experiment.

Basics

For our project, we'll think of the "world" as a 2-D grid of "cells." Each cell can be either on (1) or off (0). Think of the values of each cell in the world as a single entity called an "iteration." When "time" moves forward, we call it a new *iteration*. The value of a cell in the next time depends only on the value of that cell and its "neighbors" (the 4 adjacent and 4 diagonal cells) in the previous time.

The rules

Dr. Conway's original rules are stated below. Think about how you would translate these rules into "if-else" statements.

1. Any live cell with fewer than two live neighbors dies, as if caused by underpopulation.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Example

We will show the "live" cells as black, and "dead" cells as white. When we use format (3, 2) we mean (row, column). In the following grid, row 3 column 2 is "alive" but row 2 column 3 is dead.

	1	2	3	4	5
1					
2					
3		■	■	■	
4					
5					

What will the value of (3, 2) be in the next iteration? First, let's count it's neighbors. It only has one live neighbor at (3, 3). This means that **rule 1** applies, so (3, 2) will be dead in the next iteration. Same for (3, 4) as it only has one live neighbor.

	1	2	3	4	5
1	?	?	?	?	?
2	?	?	?	?	?
3	?		■		?
4	?	?	?	?	?
5	?	?	?	?	?

However, the middle cell (3, 3) has two live neighbors. According to rule 2, it will survive.

Are there any other cells that change in the next generation? Yes! (2, 3) and (4, 3) both have three neighbors (two diagonal and one adjacent). According to rule (4) these dead cells will come to life.

	1	2	3	4	5
1	?	?	?	?	?
2	?	?	■	?	?
3	?		■		?
4	?	?	■	?	?
5	?	?	?	?	?

Since none of the other cells are near any live cells, we have the new world for our second iteration. Notice that the pattern is the same except it rotated 90 degrees. This particular pattern is called a "blinker" because it switches back and forth every iteration. Would four in a row do the same thing, though? How about a 2x2 block of live cells with no live neighbors?

Your tasks

You will take some existing code and finish the simulator. You will also conduct some experiments and write up the results.

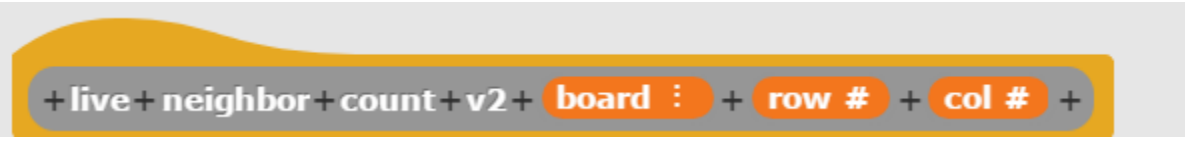
How the world is represented

In the Hangman project, you used lists to track words and letters. In this project, you'll track the world using one list. The index in the list is a function of the row and column. We choose to store the rows appended to each other. The table below shows an example for a 5 rows × 4 columns grid. Your first task is to figure out a formula to express the index into the list as a function of row and column. Then, find the reporter block called *board index* and return the correct value.

	1	**2**	**3**	**4**
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16
5	17	18	19	20

Count the live neighbors

Look back at the rules. It's clear that in order to figure out what happens in the future, we need to count the number of live neighbors around a cell. Find the block called "live neighbor count board row col" and modify it to report the number of live neighbors.



This seems easy! You could use loops, or even a bunch of  reporters chained together. But wait! What happens if we are on the borders (first or last row or column)? This behavior is your decision. You can either assume that any off-grid cells are "dead" or you can make it "wrap around" – that is, the left neighbors of column 1 are considered to be the last column and vice versa. The first method is easier, but the second method is more interesting.

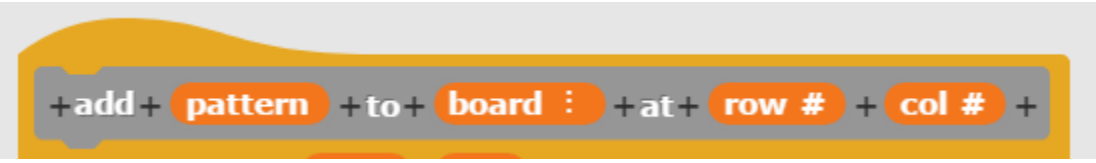
To make counting the live neighbors easier, I suggest making a "helper function" called "board board value at r c" that looks at the given *board* at row *r*, column *c* and reports the value. This helper function uses the *heuristic* you chose above: if r = 0 or r > rows or c = 0 or c > columns then report 0 or wrap around.

Program the rules

We discussed the rules on page 1. Find the block called "run old board." This block takes the existing world and reports a new world using the rules. Edit this block to populate each item in the variable new board with the values according to the rules.

Testing your code

You can test your code by using the block:



This will add a predefined pattern to the board at the given row, column. Then, press **s** to step through or space to run continuously. Look at the Wikipedia page to find out how the patterns should behave and make sure your code runs correctly.

If it doesn't, be sure make sure that your neighbor count works. You can repeat the process above: add a sample pattern and check your neighbor counter for each cell.

The list of predefined patterns is: Blinker, Glider, LWSS, Pentadecathlon. The names are case-sensitive (i.e. blinker with a lowercase b will not work).

Experiments

Cellular Automata are still actively researched including by some amateur scientists.

Pick two of the following three projects. Write the code to run the experiments and then write a 100 word analysis addressing each experiment you choose.

Experiment 1. Synthetic music

Modify the run board block to generate sounds when a cell is born, survives, or dies. Play with varying the pitch and/or duration for each event. Try your music generator with several random board and some boards using predefined patterns. In your analysis, describe how the music changed and how the pitch and duration affected the feel of the music.

Experiment 2. Convergence

Create a block to determine if two boards are the same. Modify the main routine (when space is pressed) to check if the new board is the same as the previous board. Ten times, start with a random board and have your program run until the new and previous boards are the same (i.e. nothing changed: everything survived or is dead, but no births or deaths). Write down the iteration number when this happened. In your analysis, present the table of how long it took to converge (or "never" if it exceeds 100 steps). If your boards oscillated, state this as well. Describe any emergent behaviors you noticed in your world

Experiment 3. Break the rules

Change the rules in some way: perhaps cells are born if there are 3 *or 4* live neighbors instead of just 3. Run some random boards using the original rules and your new rules. In your analysis, describe how the rules change the behavior of the world. Note: create a new block to implement the new rules.

Grading

Implement "board index" reporter	2

Implement “live neighbor count” reporter	3
Implement the original B3S23 rules correctly	3
Experiment #1 implemented	3
Experiment #1 write up complete	2
Experiment choice #2 implemented	3
Experiment choice #2 write up complete	2
Written code is well-documented and uses good style	2