# CS 251
# Project 4
# A Tour of Cellular Automata

**Due 12-6-20**

**NO Free Late Days Can Be Used**

*Joseph Haugh*

## Project Description

A cellular automata (CA) in precise terms is a discrete model of computation. It performs computation by utilizing a grid of cells. Each cell can theoretically have any finite number of states. For example in game of life a cell can either be alive or dead (1 or 0). However, cellular automata exist with more than 2 states (one of which you all will program in this project). The grid can have any number of dimensions. The grid is theoretically infinite.
This project focuses on 2 different types of CA, namely 1 dimensional and 2 dimensional. You all will be tasked with programming a 1 dimensional CA with 2 states, a 2 dimensions CA with 2 states (game of life), and a 2 dimensional CA with 8 states (Langton's loop).
This project focuses on the following concepts:

- Functions
- Lists
- Keeping track of state
- Enums
- I/O such as scanner and println
- Java FX
- Parsing

## Project Requirements

In order to receive full credit for each of the CAs you must follow the requirements given on the rubric. Make sure to look closely at the rubric as it should be able to guide you on how to complete the project as well as detailing extra credit opportunities.

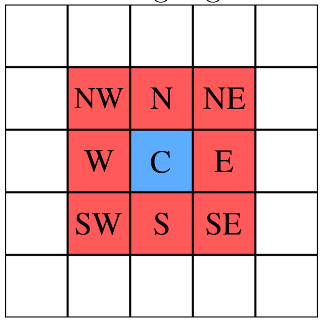
- Since all grids should be theoretically infinite we will simulate this by having our finite grid loop around. For example if you go off the edge on the right side of the grid you should come back around on the left side of the grid
- **One Dimensional CA AKA Elementary CA**
  - In a 1D CA your neighborhood only consists of the cell immediately to your left and right
  - Thus each neighborhood consists of 3 cells, the cell you are examining, the cell to its left and the cell to its right
  - Thus there are 8 possible permutations of possible states of a given neighborhood, $2^3 = 8$
  - Namely these states are:
    - 111
    - 110
    - 101
    - 100
    - 011
    - 010
    - 001
    - 000
  - These states can be read as follows:
    - The first digit is the left neighbor
    - The middle digit is the cell you are looking at
    - The right digit is the right neighbor
    - For example 101 can be read as, the cell you are currently examining is dead (0) and it has an alive neighbor on its left and right
  - A 1D CA with 2 states is specified by giving outputs to the above states.
  - Thus a 1D CA with 2 states can be described by an 8 bit binary number, where each bit in this number specifies the output of one of the above states
  - For example rule 30 is given by the binary number 00011110, which is 30 in binary padded to be 8 bits
  - This binary number determines how the 1D CA behaves for each given state in the following way:

| neighborhood state | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|---|
| new state for center cell | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

  - Since an 8 bit number can represent 0-255 then there are 256 possible different 1D CAs with 2 states
  - You must construct these 1D CAs with 2 states in two different ways:
    - Read an 8 bit number from a file along with an initial state

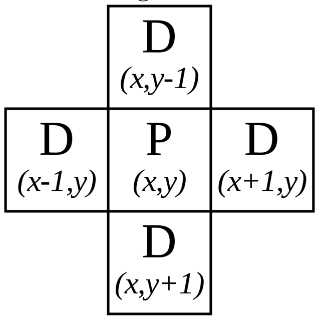
        - The file will consist of exactly 2 lines of text
        - The first line is the 8 bit string specifying the behavior of the CA
        - The second line is the initial state
        - The text will not have any delimiters
    - Read an int from the user along with an initial state, then create the 8 bit string from the given int
        - If the user enters a number outside of the range [0, 255] then you must tell the user to try again with a number in that range
        - After the user enters the initial state you must verify that it only consists of 1s and 0s otherwise you must ask them to enter a valid initial state
- Files I am giving you:
    - Cell: an enum that represents the 2 possible states of a cell
    - Grid1D: a class that handles showing a generations of cells to the screen
    - Runner: a class that has a static function that runs the animation timer which continuously updates the screen using the Grid class
    - Main: a simplified main that sets up the JavaFX window and creates a new grid with a hard coded 8 bit string, then starts the animation
- You do **NOT** have to use these files if you do not want to
- If you do use them then you can build on them in order to fulfill the requirements
- In order to get my skeleton files to work initially you need to provide the logic evolving from one generation to the next
- More detailed requirements can be found on the rubric

- **Conway's Game of Life**
  - I will **NOT** be giving you any source files for this part
  - However, you can certainly draw inspiration from the files given for the 1D CA
  - In 1970 John Conway devised a 0 player game called the Game of Life
  - Since it has 0 players its evolution is completely determined by the initial state
  - The Game of Life is a 2 dimensional CA
  - It uses the so called Moore's neighborhood, which means you look at the 8 cells surrounding a given cell when determining the next state



  - The rules are as follows:
    - Any live cell with fewer than two live neighbors dies
    - Any live cell with two or three neighbors lives
    - Any live cell with more than three live neighbors dies
    - Any dead cell with exactly three live neighbors becomes a live cell
  - You must be able to initialize the game in one of two ways:
    - Read in a file with the number of rows/cols and initial grid
      - The first line of the file consists of the number of rows and cols separated by a space
      - The next $n$ of lines, where $n$ is the number of rows, is the initial grid to be used by the game
    - Read in the number of rows/cols from the user then read in there initial grid line by line. Then you must verify there input grid matches the dimensions they gave, if it doesn't then they must re enter the grid

- **Langton's Loop**
  - I will **NOT** be giving you any source files for this part
  - In 1984 Christopher Langton came up with CA that is now called Langton's loop
  - It is a CA with 8 states that creates loops that self replicate
  - This CA uses a Von Neumann neighborhood with rotational symmetry, which means a neighborhood consists of the cells immediately north, east, south and west of a given cell



  - Its rule table is given in the file called "rule_table.txt" under the "resources/langtonsLoop" directory
  - An initial configuration is given in the file called "init_config.txt" under the "resources/langtonsLoop" directory
  - Each rule is given by a 6 digit string which can be interpreted as follows:
    - The first digit is the state of the current cell being examined
    - The next four digits represent the four neighbors of the cell being examined
    - Since the neighborhood has rotational symmetry this means those four digits have to appear in the order given but can be rotated
    - The last digit represents the state the cell being examined should transform into
    - For example given the string 123456:
      - The 1 is the state of the cell being examined
      - 2345 are its neighbors. For example 2 could be to the north, 3 to the east 4 to the south and 5 to the west. Or it could be 2 to the east, 3 to the south, 4 to the west and 5 to the north and so on.
      - 6 is the state the cell being examined should transition to
    - You must represent each state with a different color
    - You must be able to parse the rule table file into executable code, **DO NOT** hard code all the rules

## Project Submission

- You must submit a zipped file on learn
- The zipped file **ONLY** contain the following folders and files:
  - src: folder where all your code is
  - resources: configuration/specification files that I provide you as well as any your create yourself
  - README.md: file that briefly describes your project. This should include any features you did not finish as well as any bugs that exist
  - your_project_name.jar: executable jar file that the graders can use to run your project