

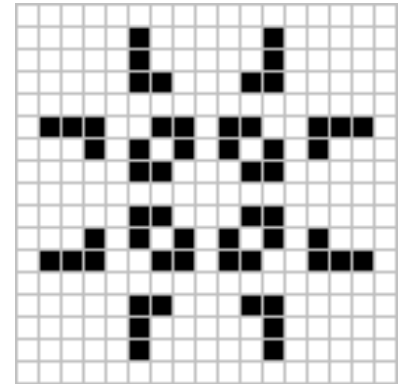


CAL (Cellular Automaton Language)

Calvin Hu + Nathan Keane + Eugene Kim

What's Cellular Automaton?

- Discrete, abstract computational system that provides useful models of non-linear dynamics
- First discovered in 1940
- Conway's **Game of Life** in 1970s



Motivation - It's everywhere!

- **Biology**

- Seashells, plants
- Cephalopod
- Neurons



- **Chemistry**

- Belousov–Zhabotinsky reaction

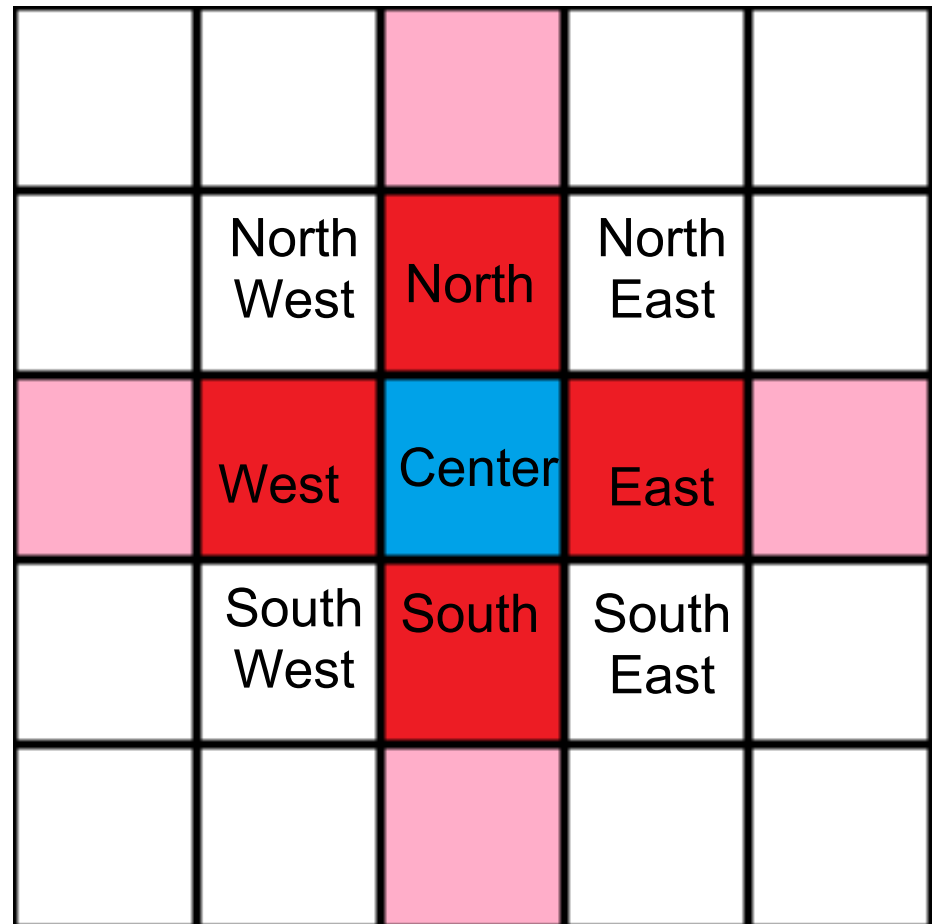
- **Computer Science**

- Cryptography
- Random number generation
- Parallel computing



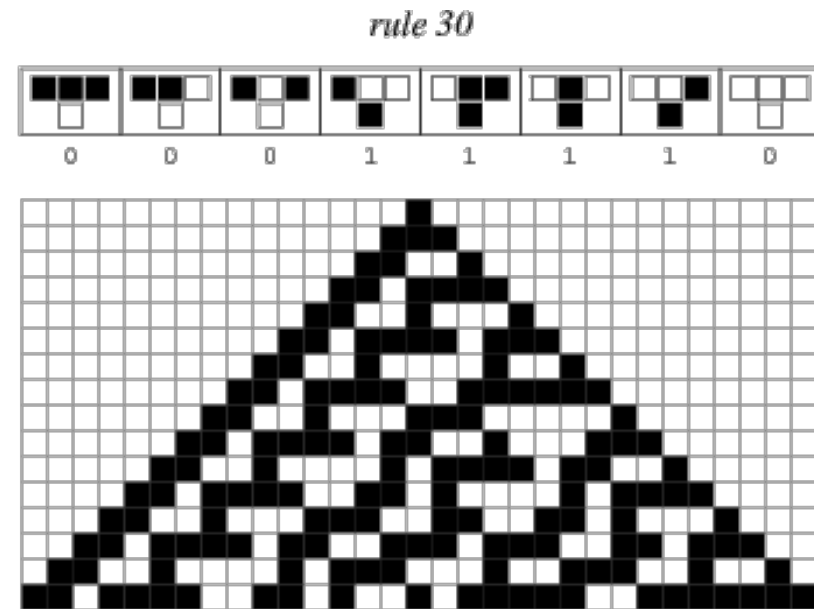
Development Process

- A collection of **colored** cells
- On a **grid** of specified size
- That **evolves** through a number of **discrete steps**
- According to a set of **rules** based on
- The states of **neighboring** cells



Development Process

- The **rules** are then applied iteratively for **as many time steps** as desired



Introduction to CAL - Grid

- State of an entire cellular automaton encapsulated in a **primitive** called Grid
- `set_grid_size(100, 100);`
- `grid g = [A, B, 2,B; B, A, 1, C; A, A, A ,A;];`

A	B	2	B
B	A	1	C
A	A	A	A

Introduction to CAL - actor_type

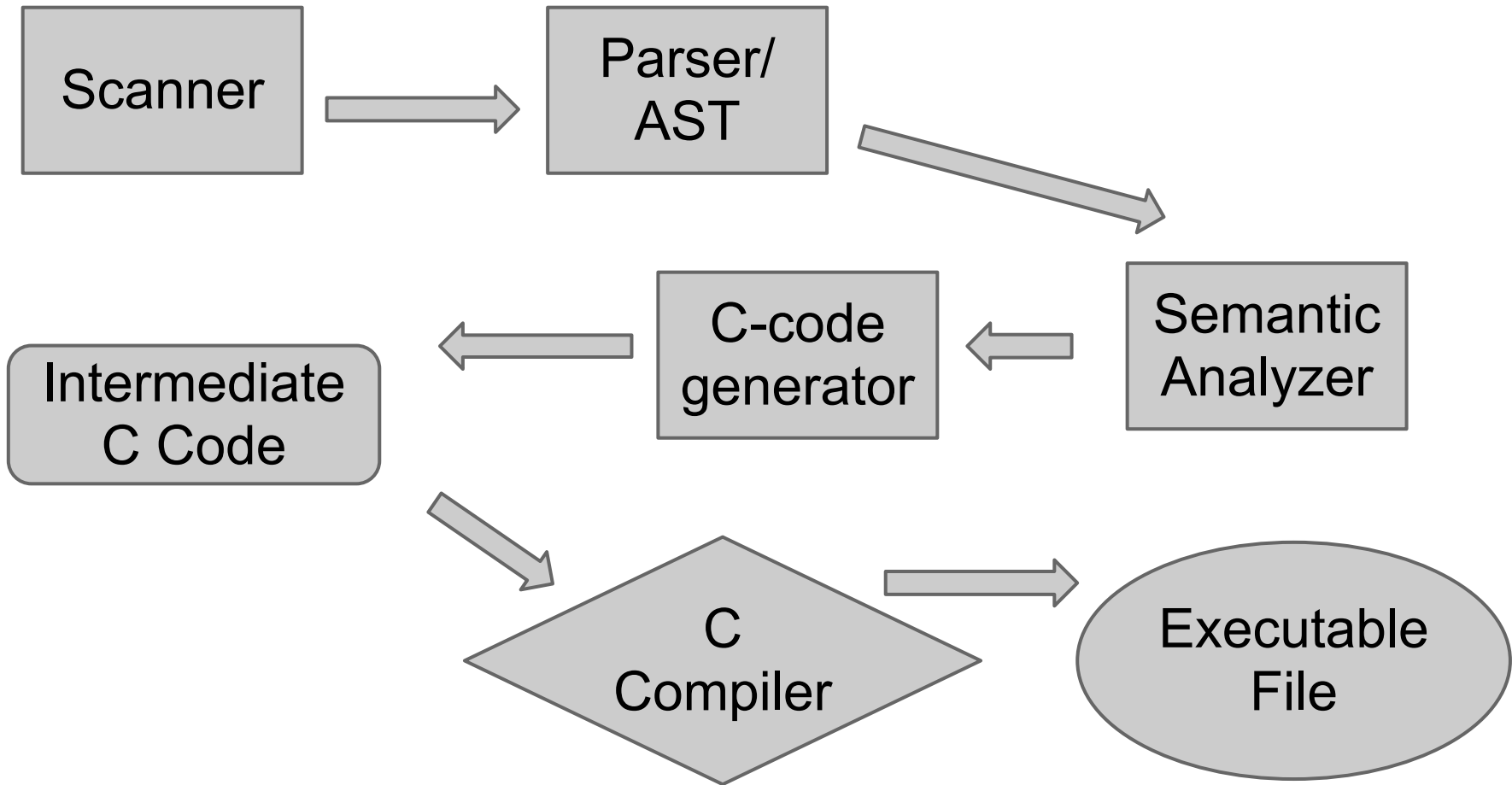
```
actor_type Fish = |  
  init:  
    int counter = 0;  
  rules:
```

```
    counter <= 10 && neighborhood(Free) > 0 => {  
      move(randomof(Free), Free);}  
    counter > 10 && neighborhood(Free) > 0 => {  
      assign_type(randomof(Free), Fish);  
      counter = 0; }  
    default => { counter = counter + 1; } |
```



- Creates and configures **actors** in the grid
- “init” block allocates and initializes **variables**
- “rules” block contains **transition logic**

Project Architecture



CAL Demo - Seeds

```
1 actor_type Off = |
2     init:
3
4 rules:
5     neighborhood(On) == 2 => {
6         assign_type(center, On);
7     }
8     default => {}
9
10 |
11
12 actor_type On = |
13     init:
14
15 rules:
16     default => { assign_type(center, Off); }
17
18 |
```

CAL Demo - Brian's Brain

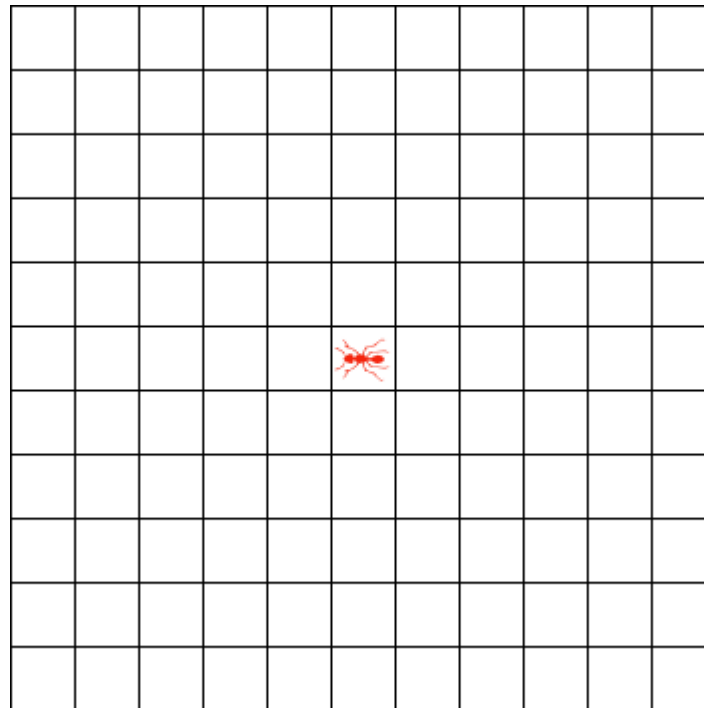
```
1 actor_type Off = |
2     init:
3
4 rules:
5     neighborhood(On) == 2 => {
6         assign_type(center, On);
7     }
8     default => {}
9
10|
11
12 actor_type On = |
13     init:
14
15 rules:
16     default => { assign_type(center, Dying); }
17
```

CAL Demo - Brian's Brain

```
18 |
19 |
20 actor_type Dying = |
21     init:
22 |
23     rules:
24         default => { assign_type(center, Off); }
25 |
26 |
27 |
28 def void setup(){
29     grid_size(250, 250);
30     set_chronon(20);
31     set_actor(50, 50, On);
32     //set_grid_pattern(1, On, Off);
33     set_grid_random();
34 }
```

CAL Demo - Langton's Ant

```
1 actor_type White = |
2   init:
3
4 rules:
5   default => {}
6
7 |
8
9 actor_type Black = |
10  init:
11
12 rules:
13  default => {}
14 |
15
```



CAL Demo - Langton's Ant

```
18 actor_type Ant = |
19     init:
20         actor_type atype = White;
21         direction ant_dir = north;
22
23     rules:
24         (cellat(ant_dir) == White) => {
25             if(ant_dir == north){
26                 ant_dir = east;
27                 move(north, atype);
28             }else if(ant_dir == east){
29                 ant_dir = south;
30                 move(east, atype);
31             }else if(ant_dir == south){
32                 ant_dir = west;
33                 move(south, atype);
34             }else{
35                 ant_dir = north;
36                 move(west, atype);
37             }
38             atype = Black;
39             printf("Direction: %d, Atype: %c\n", ant_dir, atype);
40         }
```

CAL Demo - Langton's Ant

```
41         default => {
42             if(ant_dir == north){
43                 ant_dir = west;
44                 move(north, atype);
45             }else if(ant_dir == west){
46                 ant_dir = south;
47                 move(west, atype);
48             }else if(ant_dir == south){
49                 ant_dir = east;
50                 move(south, atype);
51             }else{
52                 ant_dir = north;
53                 move(east, atype);
54             }
55             atype = White;
56             printf("Direction: %d, Atype: %d\n", ant_dir, atype);
57         }
58
59 |
```

CAL Demo - Langton's Ant

```
62 def void setup(){
63     grid_size(200, 200);
64     set_chronon(1);
65     set_cell_size(4);
66     set_grid_pattern(3, Black, White, 200, 200, 0, 0);
67     //set_grid_pattern(1, Black, White, 40, 40, 75, 75);
68     set_actor(75, 75, Ant);
69     //set_grid_random();
70 }
```

CAL Demo - Rule 90

- Rule 90 is a one-dimensional cellular automaton based on the exclusive or function
- Each cell can hold either a 0 or a 1 value and at each time step all values are simultaneously replaced by the exclusive or of the two neighboring values

CAL Demo - Rule 90

```
1 actor_type On = |
2     init:
3
4     rules:
5         cellat(west) == On && cellat(east) == On => {
6             assign_type(center, SetOn);
7             assign_type(south, Off);
8         }
9         cellat(west) == On && cellat(east) == Off => {
10            assign_type(center, SetOn);
11            assign_type(south, On);
12        }
13        cellat(west) == Off && cellat(east) == On => {
14            assign_type(center, SetOn);
15            assign_type(south, On);
16        }
17        cellat(west) == Off && cellat(east) == Off => {
18            assign_type(center, SetOn);
19            assign_type(south, Off);
20        }
21    default => { }
```

CAL Demo - Rule 90

```
24 actor_type Off = |
25     init:
26
27     rules:
28         cellat(west) == On && cellat(east) == On => {
29             assign_type(center, SetOff);
30             assign_type(south, Off);
31         }
32         cellat(west) == On && cellat(east) == Off => {
33             assign_type(center, SetOff);
34             assign_type(south, On);
35         }
36         cellat(west) == Off && cellat(east) == On => {
37             assign_type(center, SetOff);
38             assign_type(south, On);
39         }
40         cellat(west) == Off && cellat(east) == Off => {
41             assign_type(center, SetOff);
42             assign_type(south, Off);
43         }
44     default => { }
45 |
```

CAL Demo - Rule 90

```
47 actor_type SetOn = |
48     init:
49
50     rules:
51         default => { }
52 |
53
54 actor_type SetOff = |
55     init:
56
57     rules:
58         default => { }
59 |
60
61 def void setup(){
62     grid_size(300, 300);
63     set_cell_size(2);
64     set_grid_pattern(3, SetOff, On, 300, 300, 0, 0);
65     set_grid_pattern(3, Off, On, 300, 1, 0, 0);
66     set_actor(150, 0, On);
67     set_chronon(20);
68 }
```

Advantages of CAL

WE ♥ CAL

- Easily designate the set of **initial states**
- Succinctly declare **rules**
- See the outcome in both textual and **graphical** formats

Summary and Lessons Learned

- Well-thought-out LRM pays off
- Synchronize with teammates
- Prioritize and persist

**KEEP
CALM
AND
IT'S
DONE**