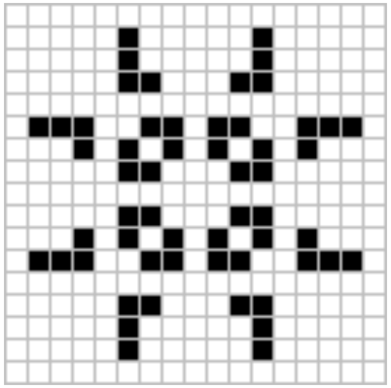
CAL

(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!

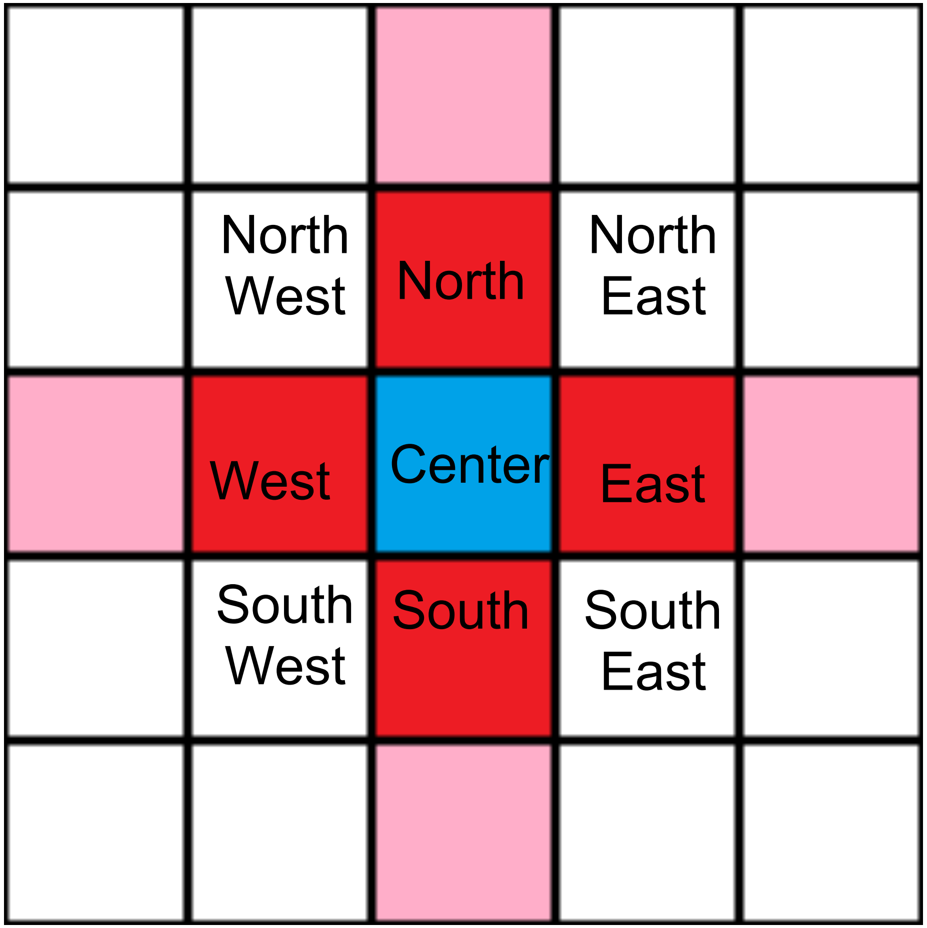
- Biology
  - Seashells, plants
  - Cephalopod
  - Neurons

- Chemistry
  - Belousov–Zhabotinsky reaction

- Computer Science
  - Cryptography
  - Random number generation
  - Parallel computing

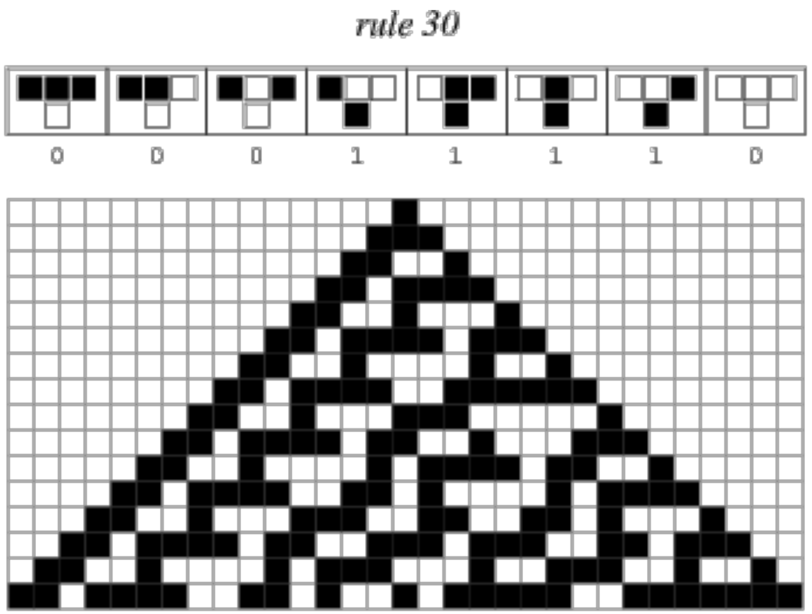
Development Process

- 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;];

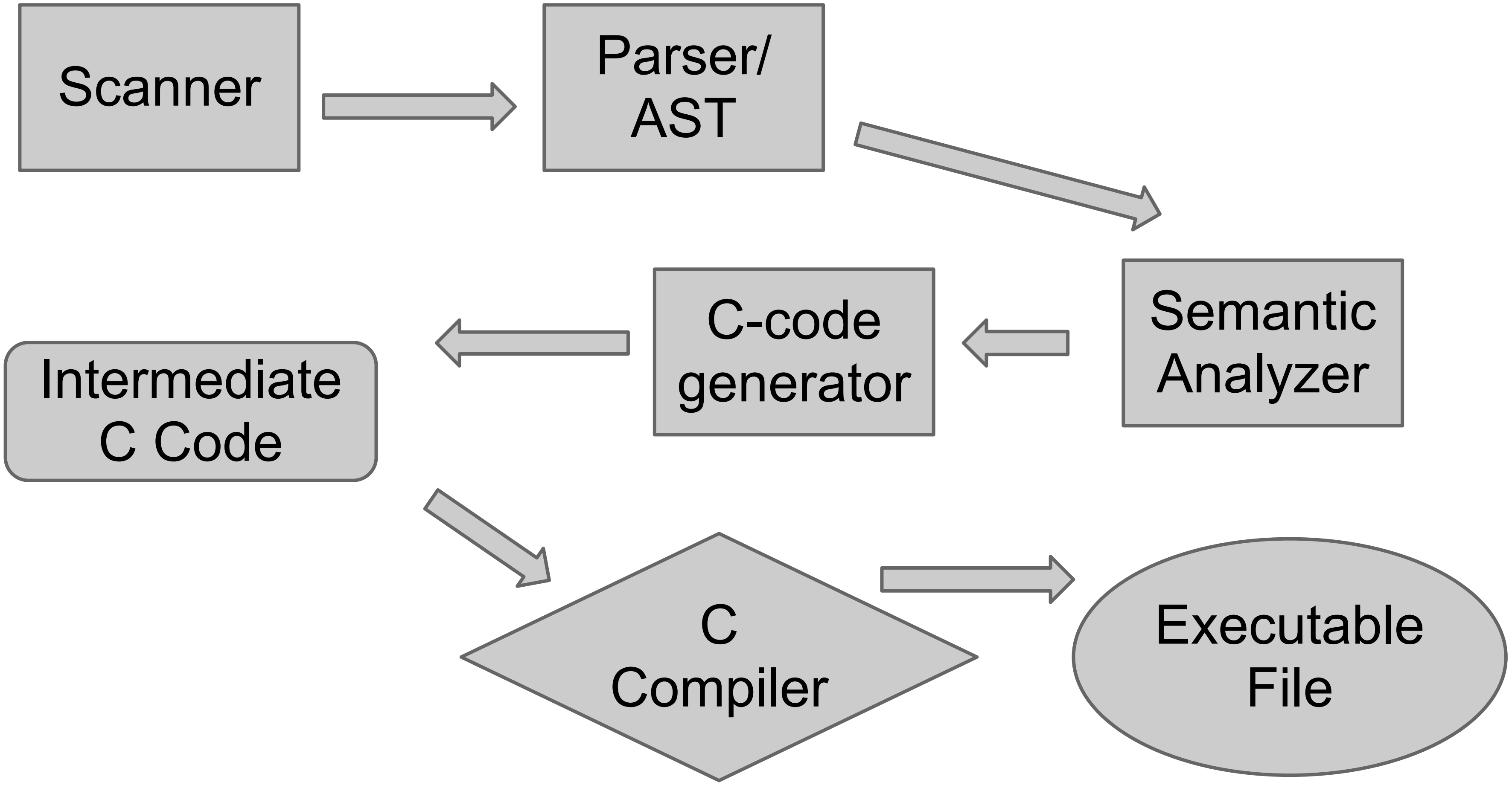
A	B	2	B
B	A	1	C
A	A	A	A

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

```
actor_type Off =  |
        init:

rules:
                neighborhood(On) == 2 => {
                        assign_type(center, On);
                }
                default => { }

|

actor_type On = |
        init:

rules:
                default => { assign_type(center, Off); }

|
```

CAL Demo - Brian's Brain

```
actor_type Off = |
        init:

rules:
                neighborhood(On) == 2 => {
                        assign_type(center, On);
                }
                default => { }

|

actor_type On = |
        init:

rules:
                default => { assign_type(center, Dying); }

```

CAL Demo - Brian’s Brain

```
|

actor_type Dying = |
        init:

        rules:
                default => { assign_type(center, Off); }

|

def void setup(){
        grid_size(250, 250);
        set_chronon(20);
        set_actor(50, 50, On);
        //set_grid_pattern(1, On, Off);
        set_grid_random();
}
```

CAL Demo - Langton's Ant

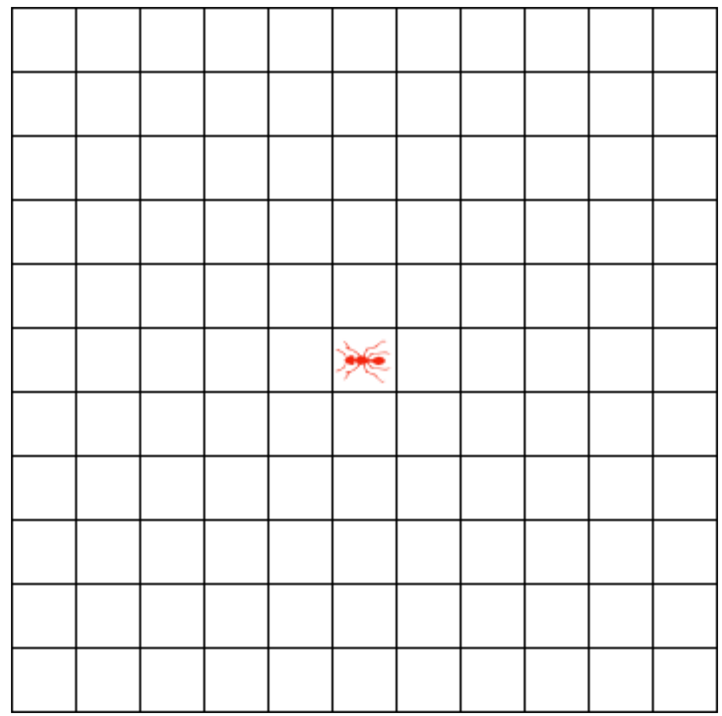
```
actor_type White = |
        init:

rules:
                default => { }

|

actor_type Black = |
        init:

rules:
                default => { }
|
```



CAL Demo - Langton's Ant

```
actor_type Ant =  |
        init:
                actor_type atype = White;
                direction ant_dir = north;

            rules:
                (cellat(ant_dir) == White) => {
                        if(ant_dir == north){
                                ant_dir = east;
                                move(north, atype);
                        }else if(ant_dir == east){
                                ant_dir = south;
                                move(east, atype);
                        }else if(ant_dir == south){
                                ant_dir = west;
                                move(south, atype);
                        }else{
                                ant_dir = north;
                                move(west, atype);
                        }
                        atype = Black;
                        printf("Direction: %d, Atype: %c\n", ant_dir, atype);
                }
```

CAL Demo - Langton's Ant

```
default => {
    if(ant_dir == north){
        ant_dir = west;
        move(north, atype);
    }else if(ant_dir == west){
        ant_dir = south;
        move(west, atype);
    }else if(ant_dir == south){
        ant_dir = east;
        move(south, atype);
    }else{
        ant_dir = north;
        move(east, atype);
    }
    atype = White;
    printf("Direction: %d, Atype: %d\n", ant_dir, atype);
}
```

CAL Demo - Langton’s Ant

```
def void setup(){
        grid_size(200, 200);
        set_chronon(1);
        set_cell_size(4);
        set_grid_pattern(3, Black, White, 200, 200, 0, 0);
        //set_grid_pattern(1, Black, White, 40, 40, 75, 75);
        set_actor(75, 75, Ant);
        //set_grid_random();
}
```

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

```
actor_type On = |
        init:

        rules:
           cellat(west) == On  && cellat(east) == On => {
                          assign_type(center, SetOn);
                 assign_type(south, Off);
                 }
           cellat(west) == On  && cellat(east) == Off => {
                          assign_type(center, SetOn);
                 assign_type(south, On);
                 }
           cellat(west) == Off  && cellat(east) == On => {
                          assign_type(center, SetOn);
                 assign_type(south, On);
                 }
           cellat(west) == Off  && cellat(east) == Off => {
                          assign_type(center, SetOn);
                 assign_type(south, Off);
           }
        default => { }
```

CAL Demo - Rule 90

```
actor_type Off = |
        init:

            rules:
                cellat(west) == On && cellat(east) == On => {
                                    assign_type(center, SetOff);
                        assign_type(south, Off);
                        }
                cellat(west) == On && cellat(east) == Off => {
                                    assign_type(center, SetOff);
                        assign_type(south, On);
                        }
                cellat(west) == Off && cellat(east) == On => {
                                    assign_type(center, SetOff);
                        assign_type(south, On);
                        }
                cellat(west) == Off && cellat(east) == Off => {
                                    assign_type(center, SetOff);
                        assign_type(south, Off);
                        }
                default => { }
|
```

CAL Demo - Rule 90

```
actor_type SetOn = |
        init:

        rules:
                default => { }
|

actor_type SetOff = |
        init:

        rules:
                default => { }
|

def void setup(){
     grid_size(300, 300);
     set_cell_size(2);
     set_grid_pattern(3, SetOff, On, 300, 300, 0, 0);
     set_grid_pattern(3, Off, On, 300, 1, 0, 0);
     set_actor(150, 0, On);
     set_chronon(20);
}
```

Advantages of 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