Computer Programming Fundamentals

CS 152
Professor: Leah Buechley
TAs: Melody Horn, Noah Garcia, Andrew Geyko, Juan Ormaza
Time: MWF 10:00-10:50am
https://handandmachine.cs.unm.edu/classes/CS152_Fall2021/

ASSIGNMENT 5

Due today, 11/8 at midnight

NO EXTENSIONS
LATE DAYS

MAKE VISUALIZING FASTER

A LITTLE LSYSTEM WRAP UP

GENERATE A FULL TREE WITH EVERY SPACE BAR PRESS

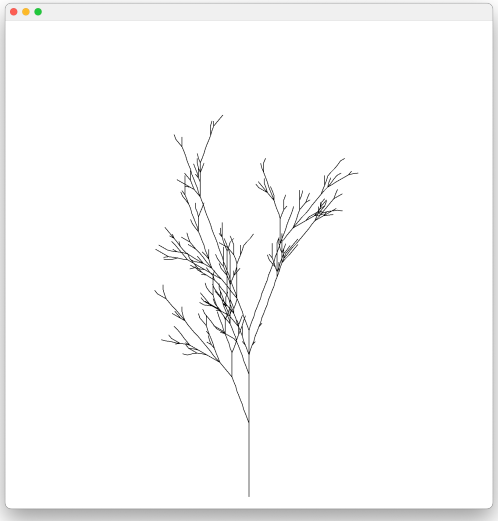
```
@Override
public void keyTyped(KeyEvent e) {
    char c = e.getKeyChar();
    if (c==' ') {
        size = 8;
        t.clearTurtleHistory();
        l = new LSystemTest("X",randomTree);
        for (int i=0;i<6;i++) {
            l.iterationRandom();
        }
        l.draw(t,size,angle);
        repaint();
    }
}
```

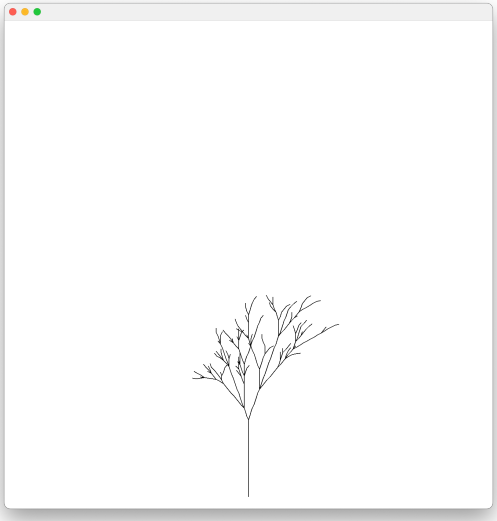
add a new method
to LSystemVis class

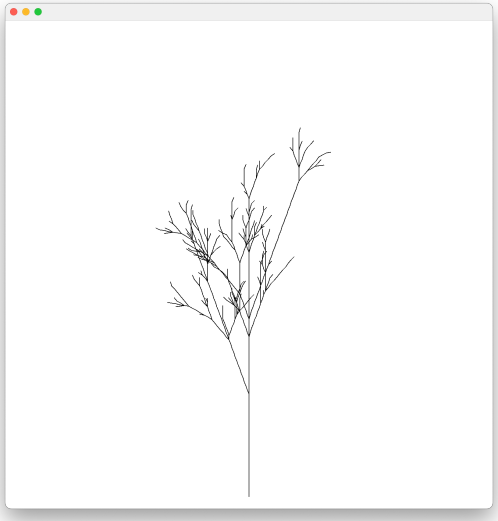
questions?

WHERE ARE Xs ON TREE?

```
iteration 4: FFFF[-F[+F[-X][+X]][-F[+X][-X]FX]FFF[-X][+X]][+FF-[[F[+X][-X]FX]+F-[[X]+X]+F[+FX]-X]
+FF[+FFF-[[X]+X]+F[+FX]-X]-F[-X][+X]]
```







LET'S SEE!
DRAWING Xs

VISUALIZING Xs

```
void draw (Turtle t, double size, double angle) {
    for (int i=0;i<computedWord.length();i++) {
        if (computedWord.charAt(i)=='F')
            t.forward(size);
        else if (computedWord.charAt(i)=='-')
            t.left(angle);
        else if (computedWord.charAt(i)=='+')
            t.right(angle);
        else if (computedWord.charAt(i)=='L')
            t.forward(size);
        else if (computedWord.charAt(i)=='R')
            t.forward(size);
        else if (computedWord.charAt(i)=='[')
            t.push();
        else if (computedWord.charAt(i)==']')
            t.pop();
    }
}
```

no drawing for Xs
let's draw a red circle

A redCircle METHOD

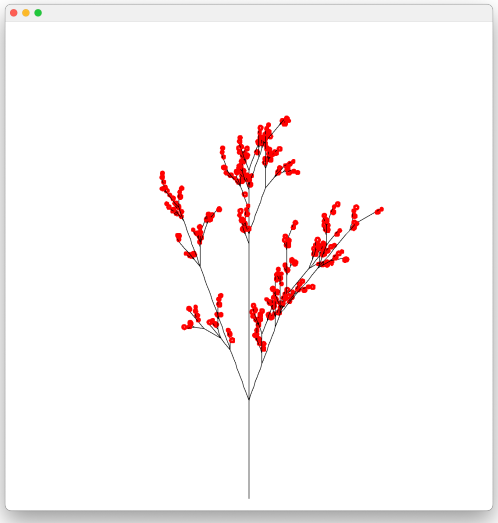
```
static void redCircle(Turtle t, double size) {
    Color c = t.getColor();
    double s = t.getStroke();
    t.setColor(Color.RED);
    t.setStroke(3);
    for (int i=0;i<20;i++) {
        t.forward(size);
        t.right(360/20);
    }
    t.setColor(c);
    t.setStroke(s);
}
```

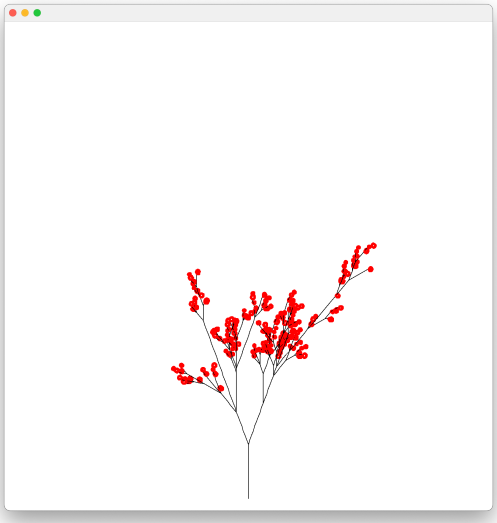
IN draw METHOD

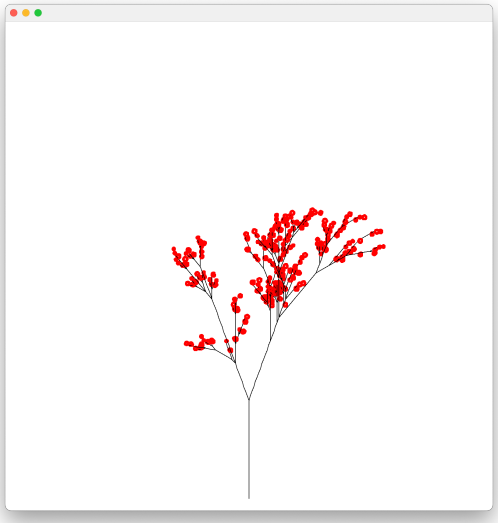
```
void draw (Turtle t, double size, double angle) {
    for (int i=0;i<computedWord.length();i++) {
        if (computedWord.charAt(i)=='F')
            t.forward(size);
        else if (computedWord.charAt(i)=='-')
            t.left(angle);
        else if (computedWord.charAt(i)=='+')
            t.right(angle);
        else if (computedWord.charAt(i)=='L')
            t.forward(size);
        else if (computedWord.charAt(i)=='R')
            t.forward(size);
        else if (computedWord.charAt(i)=='[')
            t.push();
        else if (computedWord.charAt(i)==']')
            t.pop();
        else if (computedWord.charAt(i)=='X')
            redCircle(t,.5);
    }
}
```

WHERE ARE Xs ON TREE?

```
iteration 4: FFFF[-F[+F[-X][+X]][-F[+X][-X]FX]FFF[-X][+X]][+FF-[[F[+X][-X]FX]+F-[[X]+X]+F[+FX]-X]
+FF[+FFF-[[X]+X]+F[+FX]-X]-F[-X][+X]]
```

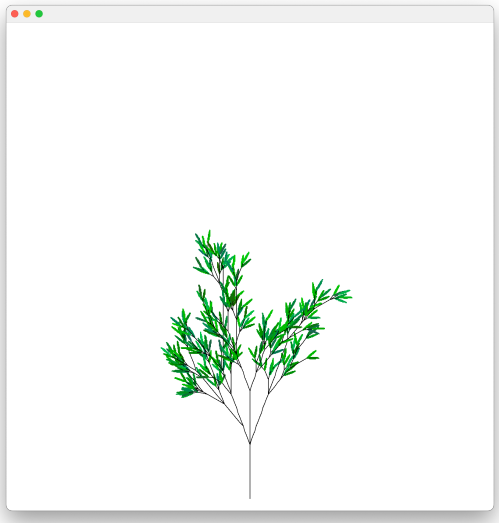


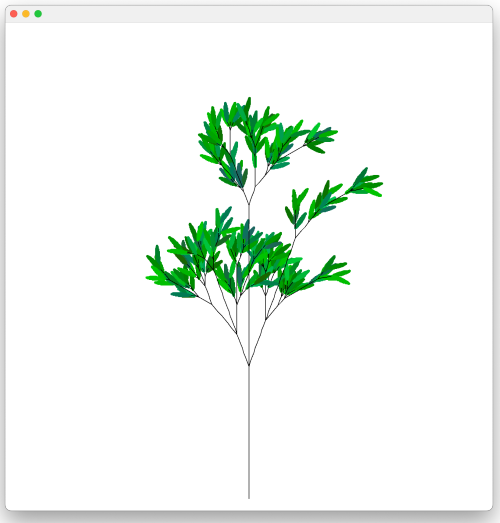




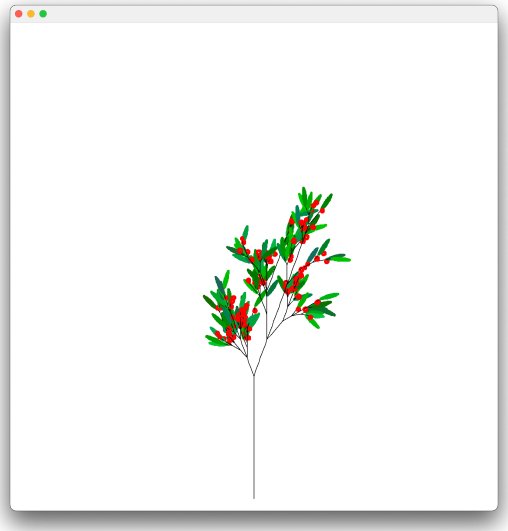
AT THE TIPS OF THE BRANCHES THINK ABOUT WHY…

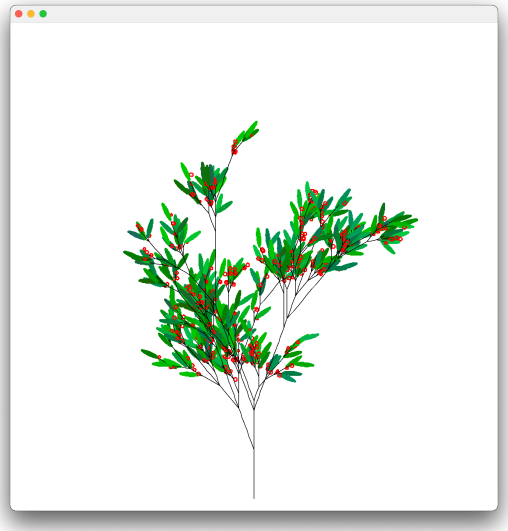
LEAVES INSTEAD OF BERRIES

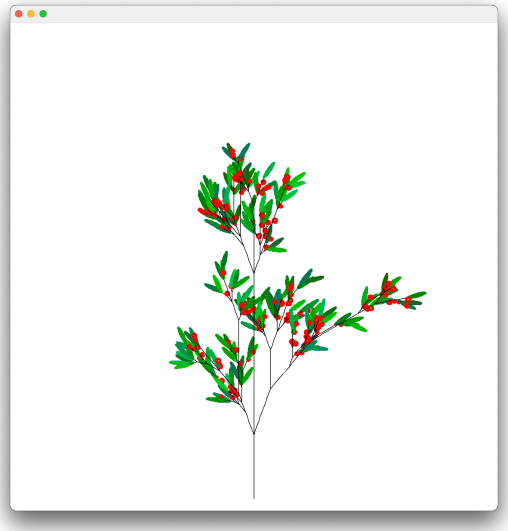




LEAVES & BERRIES







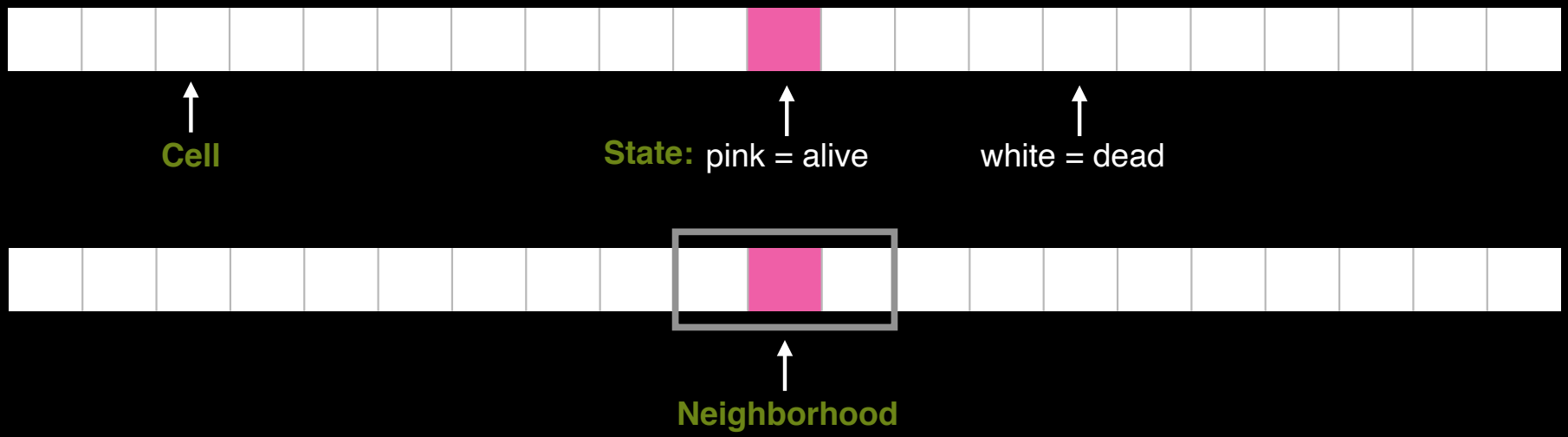
questions?

A NEW TOPIC:
CELLULAR AUTOMATA

CELLULAR AUTOMATA

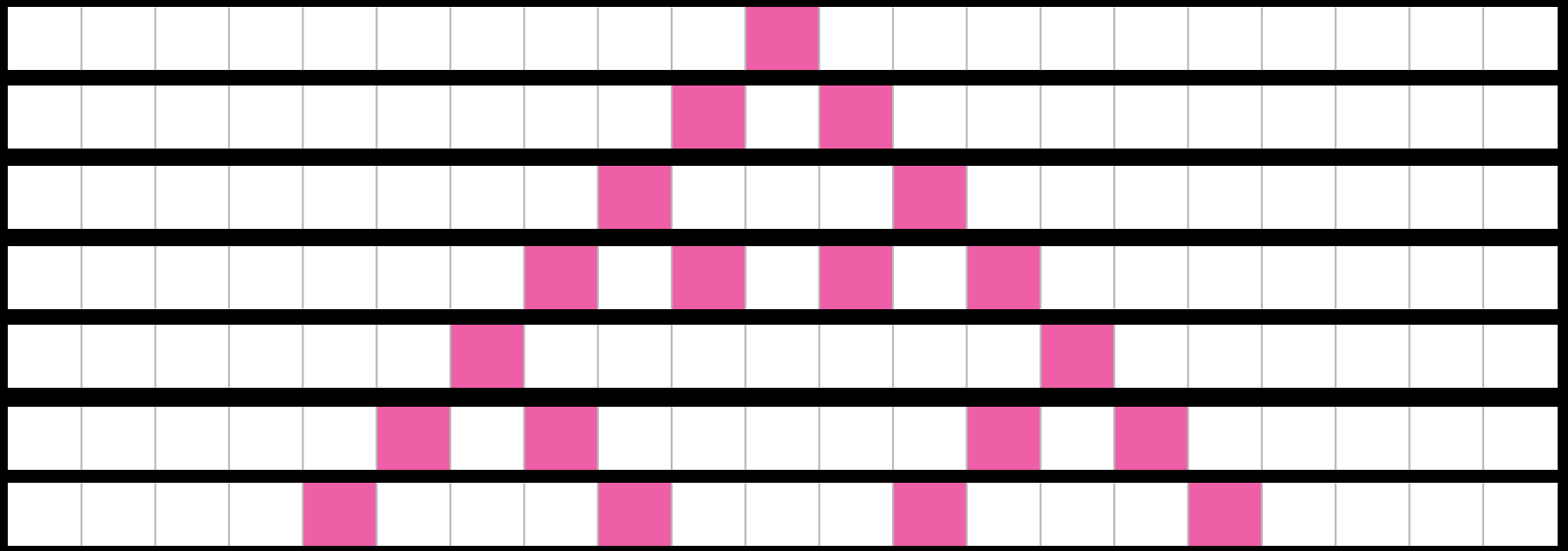
- Cellular Automata = plural
- Cellular Automaton = singular
- A model based on cells that live on a grid or line.
- Each cell has a neighborhood that consists of nearby cells
- Each cell has a state. ie: “alive” or “dead”
- The state of each cell changes over time according to a rule, based on its state and the state of its neighbors

A 1D CELLULAR AUTOMATON

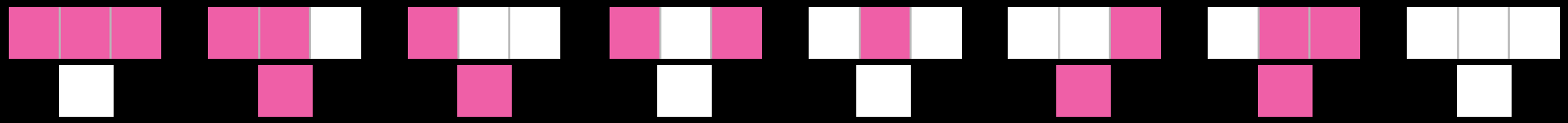


Rule:

A 1D CELLULAR AUTOMATON



Rule:



questions?

1D CELLULAR AUTOMATA IN CODE

CREATE A NEW PROJECT:
Week12

CREATE A CellularAutomata1D.java CLASS

1D CELLULAR AUTOMATA IN CODE

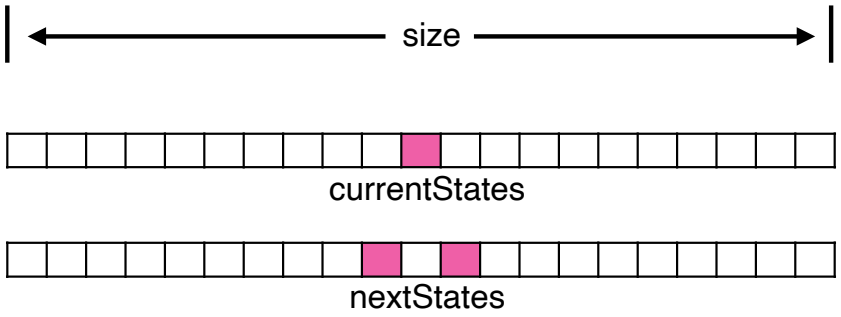
```
public class CellularAutomata1D {

}
```

1D CELLULAR AUTOMATA IN CODE

```
public class CellularAutomata1D {
    int size;
}
```

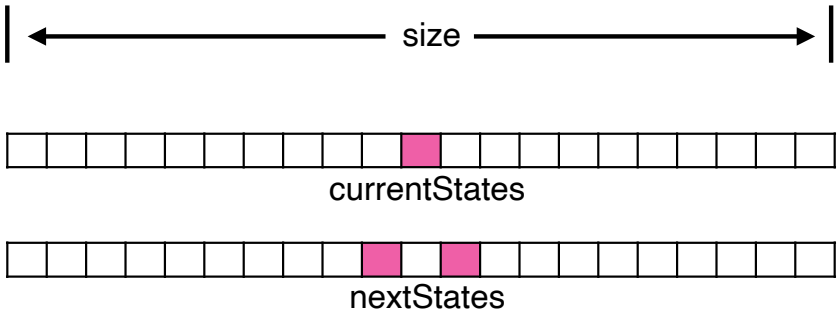
1D CELLULAR AUTOMATA IN CODE

```
public class CellularAutomata1D {
    int size;
    int[] currentStates;
    int[] nextStates;
}
```



1D CELLULAR AUTOMATA IN CODE

```
public class CellularAutomata1D {
    int size;
    int[] currentStates;
    int[] nextStates;
    final int ALIVE = 1;   //pink
    final int DEAD = 0;    //white
}
```



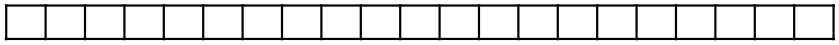
questions?

ADD A CONSTRUCTOOR

```
public class CellularAutomata1D {
    int size;
    int[] currentStates;
    int[] nextStates;
    final int ALIVE = 1;  //pink
    final int DEAD = 0;   //white

    CellularAutomata1D() {
        size = 100;
        currentStates = new int[size];
        nextStates = new int[size];
    }
}
```

ADD A CONSTRUCTOOR

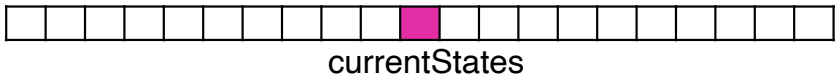
```
public class CellularAutomata1D {
    int size;
    int[] currentStates;
    int[] nextStates;
    final int ALIVE = 1;  //pink
    final int DEAD = 0;   //white

    CellularAutomata1D() {
        size = 100;
        currentStates = new int[size];
        nextStates = new int[size];
        //initialize currentStates array
        for (int i=0;i<size;i++) {
            currentStates[i] = DEAD;
        }
   }
}
```



currentStates

ADD A CONSTRUCTOOR

```
public class CellularAutomata1D {
    int size;
    int[] currentStates;
    int[] nextStates;
    final int ALIVE = 1;  //pink
    final int DEAD = 0;   //white

    CellularAutomata1D() {
        size = 100;
        currentStates = new int[size];
        nextStates = new int[size];
        //initialize currentStates array
        for (int i=0;i<size;i++) {
            currentStates[i] = DEAD;
        }
        currentStates[size/2] = ALIVE;
    }
}
```



questions?

DISPLAY WITH PRINTING

```
void displayCurrentStates() {
    for (int i=0;i<currentStates.length; i++) {
        if (currentStates[i] == ALIVE)
            System.out.print("*");
        else
            System.out.print(" ");
    }
    System.out.println();
}
```

DISPLAY WITH PRINTING

```
public static void main(String[] args) {
    CellularAutomata1D CA = new CellularAutomata1D();
    CA.displayCurrentStates();
}
```

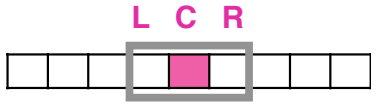
*

questions?

COMPUTE RULE FOR ONE CELL

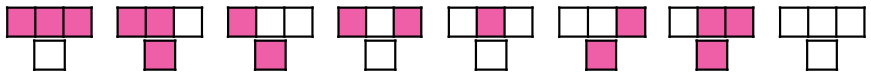
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {

}
```



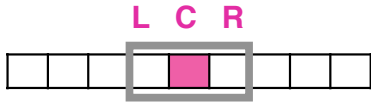
Neighborhood

Rule:



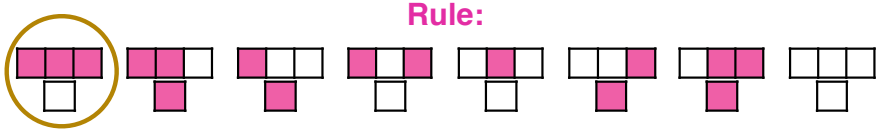
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
}
```



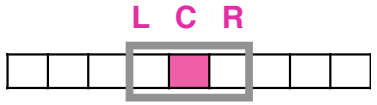
Neighborhood

Rule:



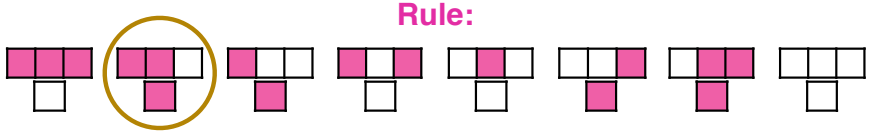
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
}
```



Neighborhood

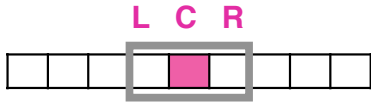
Rule:



why don’t we need an else?
the return statement will get us
out of the method

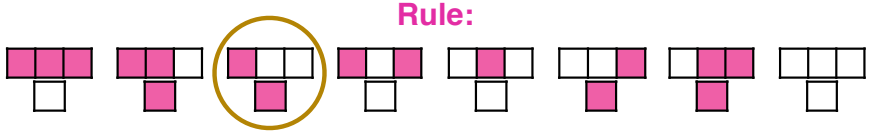
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
}
```



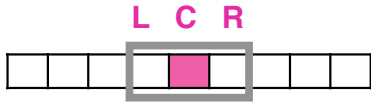
Neighborhood

Rule:

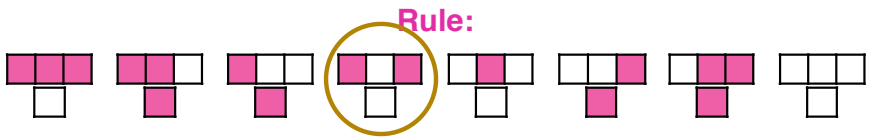


OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
}
```

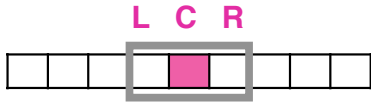


Neighborhood



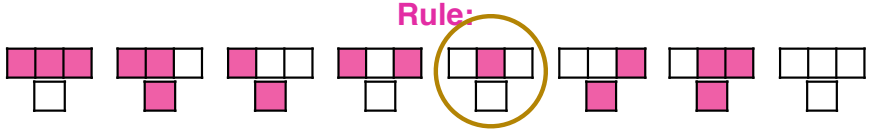
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
}
```



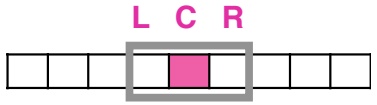
Neighborhood

Rule:



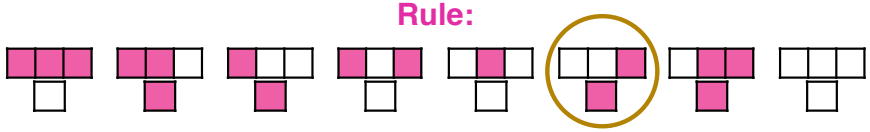
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
    if (left == DEAD && center == DEAD && right == ALIVE)
        return ALIVE;
}
```



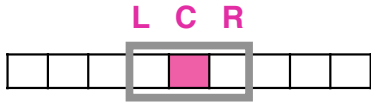
Neighborhood

Rule:



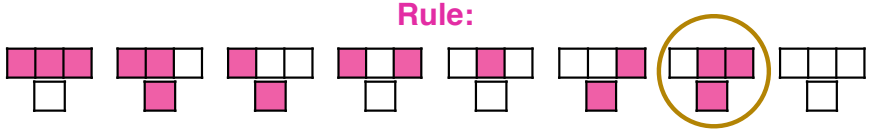
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
    if (left == DEAD && center == DEAD && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == ALIVE && right == ALIVE)
        return ALIVE;
}
```



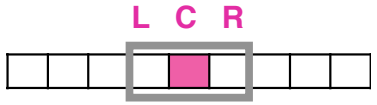
Neighborhood

Rule:



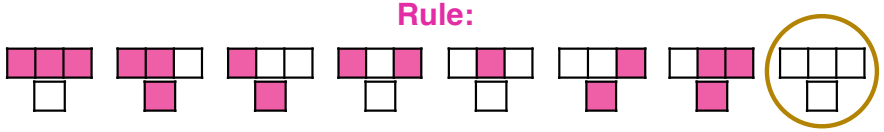
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
    if (left == DEAD && center == DEAD && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == ALIVE && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == DEAD && right == DEAD)
        return DEAD;
}
```



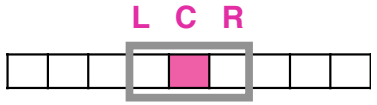
Neighborhood

Rule:



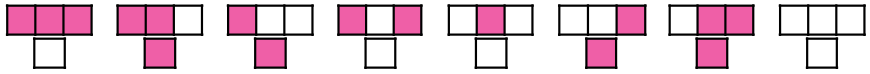
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
    if (left == DEAD && center == DEAD && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == ALIVE && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == DEAD && right == DEAD)
        return DEAD;
    return -1;
}
```



Neighborhood

Rule:



questions?