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/

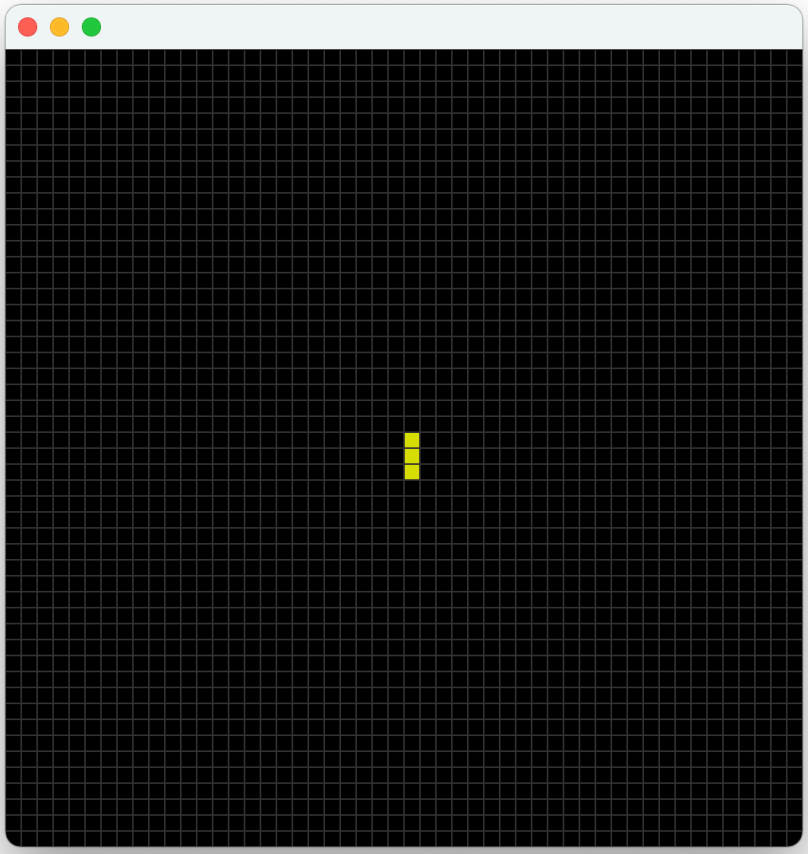
ASSIGNMENT 6
DUE FRIDAY 11/19
EMAIL ME TODAY TO WORK IN TEAM

questions?

WHERE WE ARE

CODING A 2D CA

- Created the class
- Added instance variables
- Added a constructor
- Adding a display method to draw cells in window



ADD A DISPLAY METHOD

```
void displayCurrentStates(Graphics g) {
    for (int i=0;i<size;i++) {
        for (int j = 0; j < size; j++) {

       }
    }
}
```

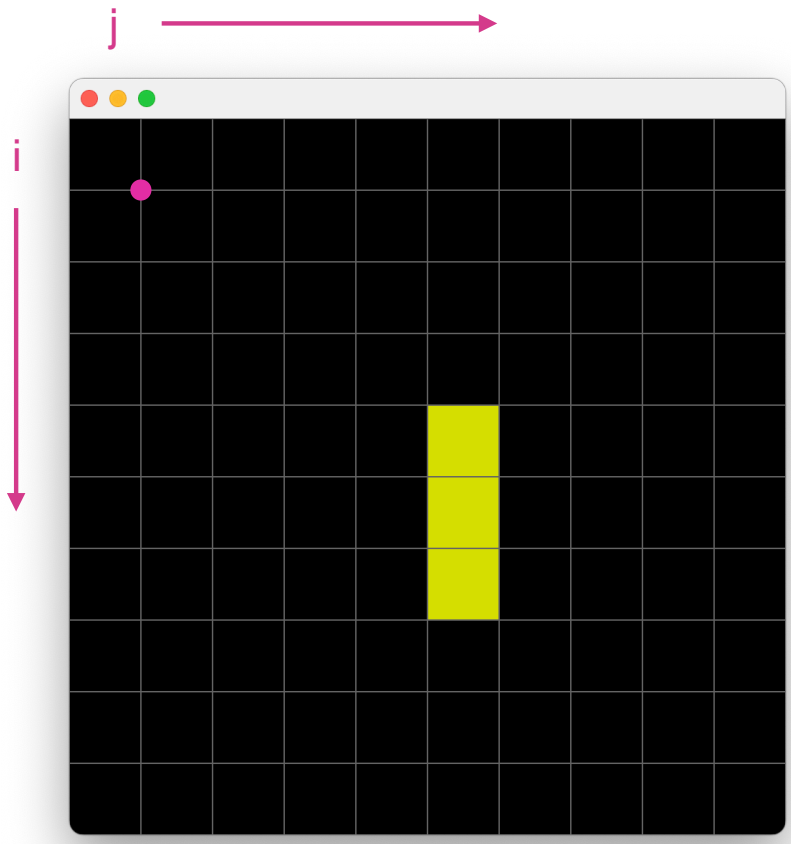
loop through 2D array

ADD A DISPLAY METHOD

```
void displayCurrentStates(Graphics g) {
    for (int i=0;i<size;i++) {
        for (int j = 0; j < size; j++) {
            if (currentStates[i][j] == ALIVE) {
                g.setColor(ALIVE_COLOR);
            }
            else {
                g.setColor(DEAD_COLOR);
            }
       }
    }
}
```

set color depending on cell state

DRAW RECTANGLE FOR EACH CELL



- Grid is size x size cells
- Square for each cell of dimension CELLSIZE

- What is x coordinate for each cell?
- Using variables and constants?
- j*CELLSIZE

- What is y coordinate for each cell?
- Using variables and constants?
- i*CELLSIZE

DRAW RECTANGLES

```
void displayCurrentStates(Graphics g) {
    for (int i=0;i<size;i++) {
        for (int j = 0; j < size; j++) {
            if (currentStates[i][j] == ALIVE) {
                g.setColor(ALIVE_COLOR);
            }
            else {
                g.setColor(DEAD_COLOR);
            }
            g.fillRect(j * CELLSIZE, i * CELLSIZE, CELLSIZE, CELLSIZE);
        }
    }
}
```

DRAW GRID ON TOP FOR LINES

```
void displayCurrentStates(Graphics g) {
    for (int i=0;i<size;i++) {
        for (int j = 0; j < size; j++) {
            if (currentStates[i][j] == ALIVE) {
                g.setColor(ALIVE_COLOR);
            }
            else {
                g.setColor(DEAD_COLOR);
            }
            g.fillRect(j * CELLSIZE, i * CELLSIZE, CELLSIZE, CELLSIZE);
            g.setColor(GRID_COLOR);
            g.drawRect(j * CELLSIZE, i * CELLSIZE, CELLSIZE, CELLSIZE);
        }
    }
}
```

use drawRect instead of fillRect

questions?

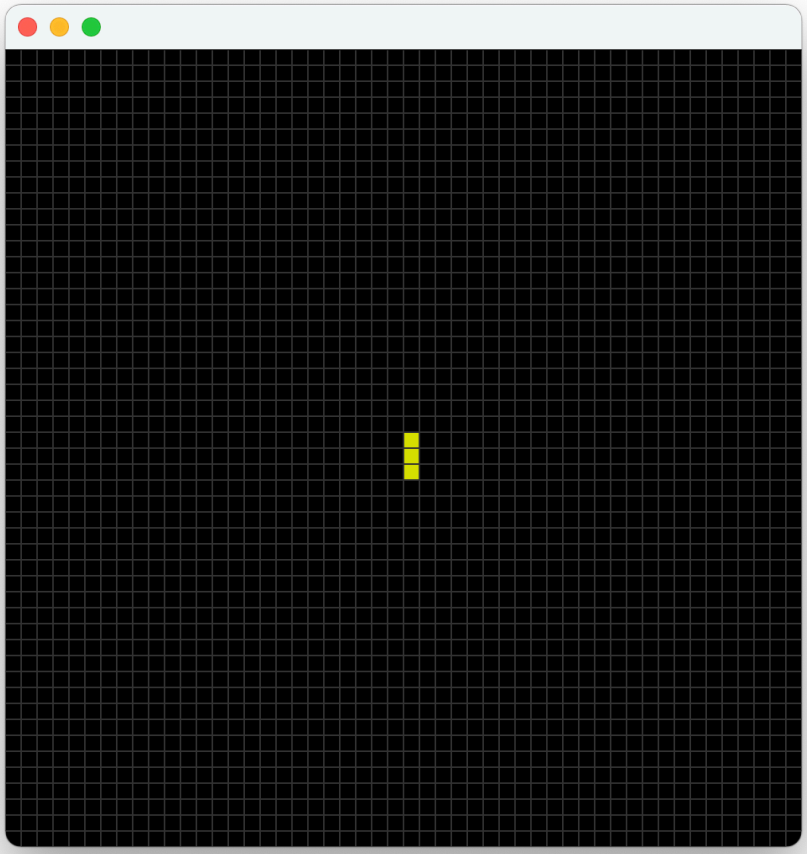
ADD A paintComponent METHOD

```
@Override
protected void paintComponent(Graphics g) {
    displayCurrentStates(g);
}
```

ADD A main METHOD

```
public static void main(String[] args) {
    CellularAutomata2D CA2D = new CellularAutomata2D();
    MyFrame frame = new MyFrame(CA2D);
}
```

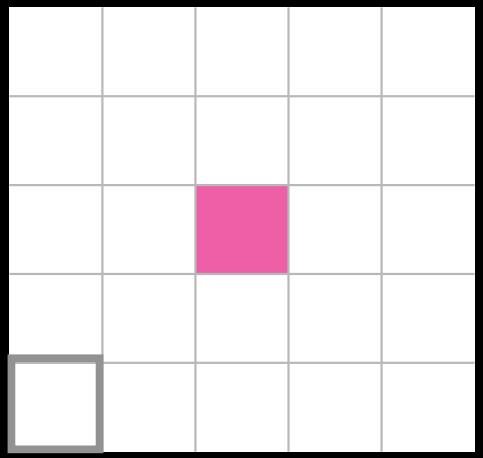
compile and run



questions?

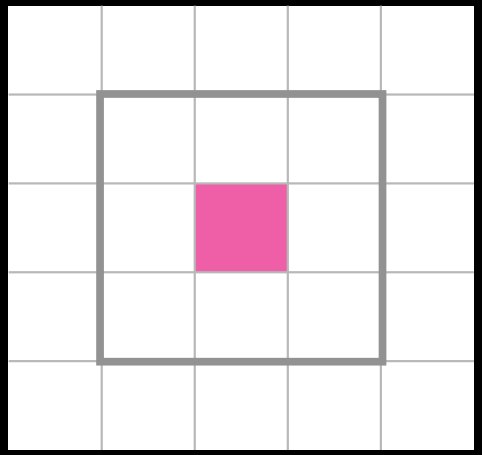
COMPUTING THE RULE

A 2D CELLULAR AUTOMATON

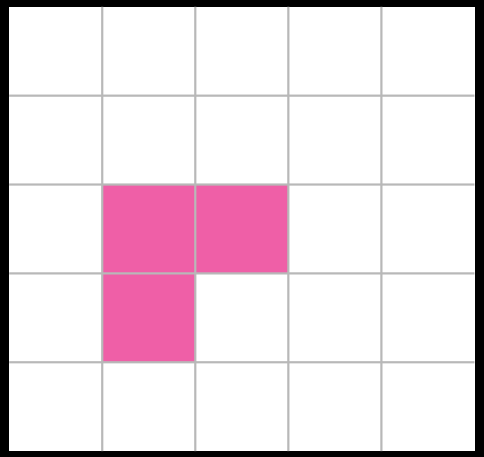


Cell

State: pink = alive
white = dead



Neighborhood

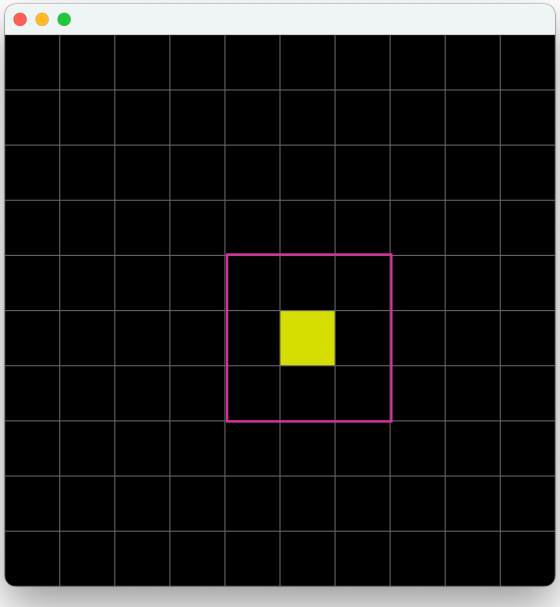


Game of Life Rule:
if alive and 2 or 3 neighbors are alive, stay alive

if dead and 3 neighbors are alive, come alive

FOR EACH CELL

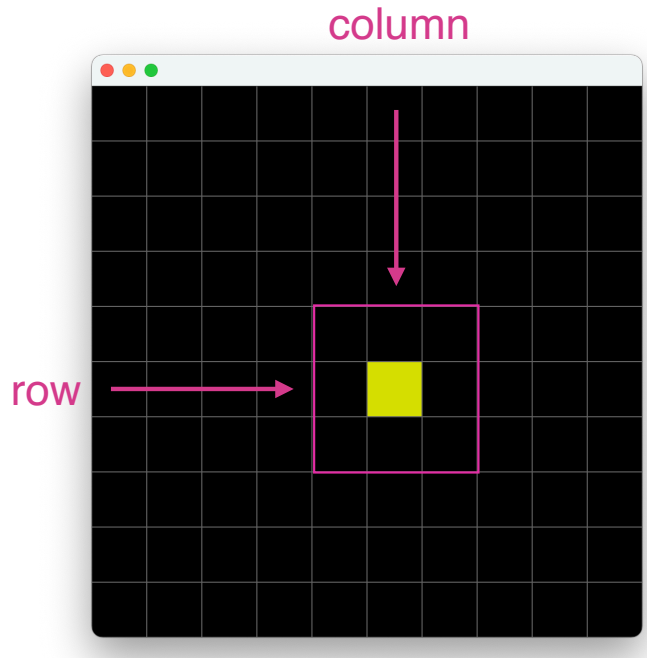
- Determine how many neighbors are alive
- Check rule condition
- Return resulting state



ADD A rule METHOD

```
int rule(int row, int column) {

}
```

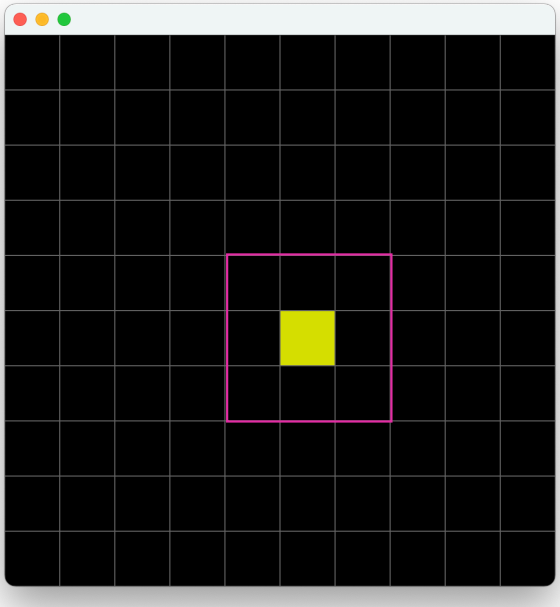
row and column
define a cell



rule METHOD

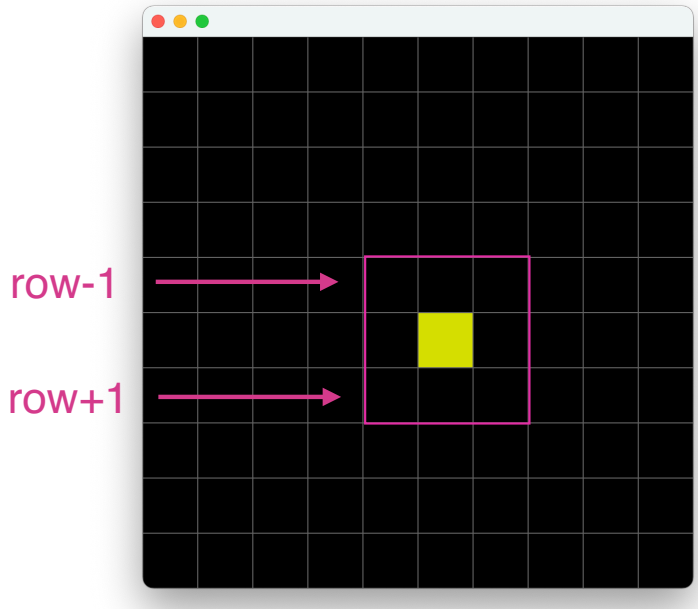
```
int rule(int row, int column) {
    int liveNeighbors=0;
}
```

count of live neighbors



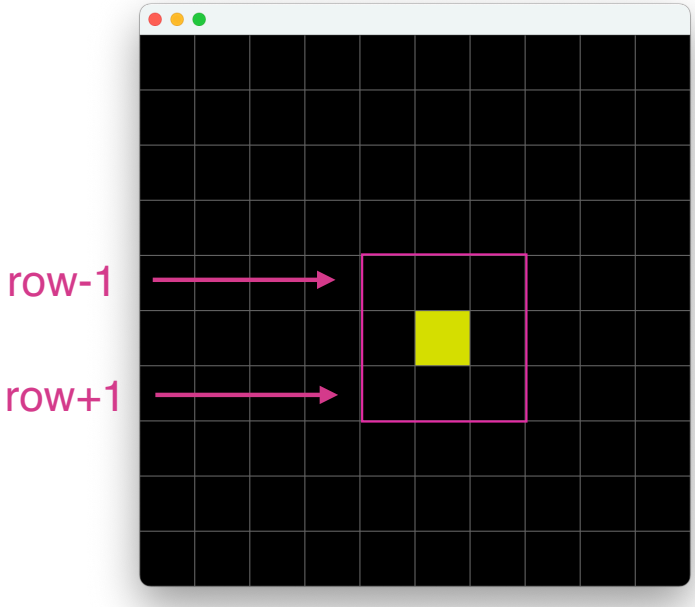
rule METHOD

```
int rule(int row, int column) {
    int liveNeighbors=0;
    for (int i=row-1;i<=row+1;i++) {

   }

}
```



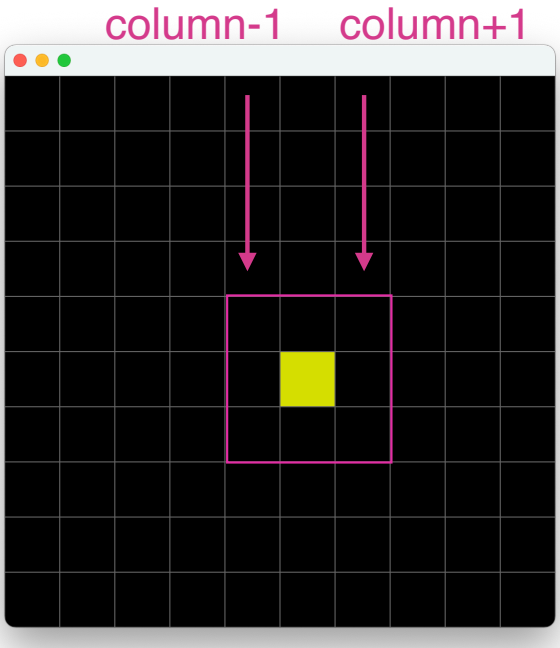
rule METHOD

```
int rule(int row, int column) {
    int liveNeighbors=0;
    for (int i=row-1;i<=row+1;i++) {

   }

}
```



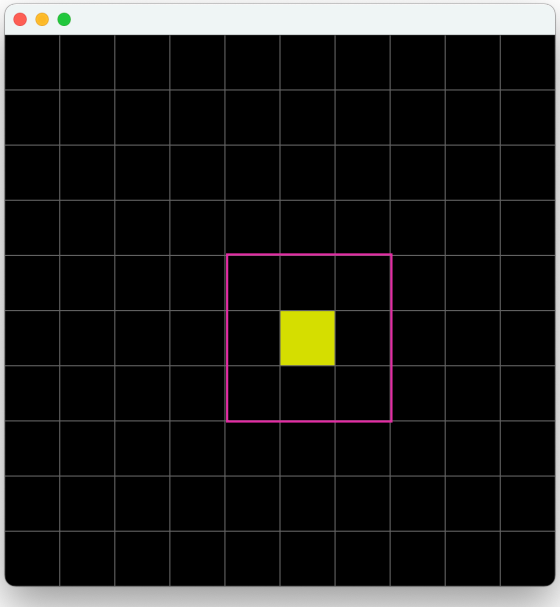
rule METHOD

```
int rule(int row, int column) {
    int liveNeighbors=0;
    for (int i=row-1;i<=row+1;i++) {
        for (int j=column-1;j<=column+1;j++) {


        }
    }
}
```



rule METHOD

```
int rule(int row, int column) {
    int liveNeighbors=0;
    for (int i=row-1;i<=row+1;i++) {
        for (int j=column-1;j<=column+1;j++) {
                liveNeighbors = liveNeighbors+currentStates[i][j];
        }
    }
}
```



add up live neighbors

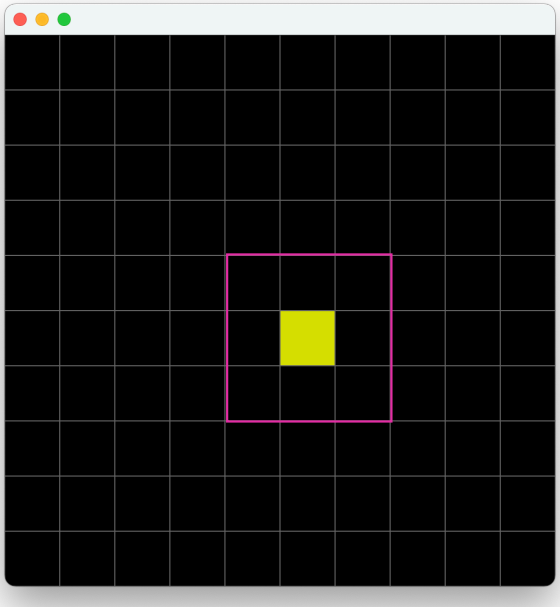
see any problems?

current cell is not a neighbor to itself!

can’t add it to live neighbor count

rule METHOD

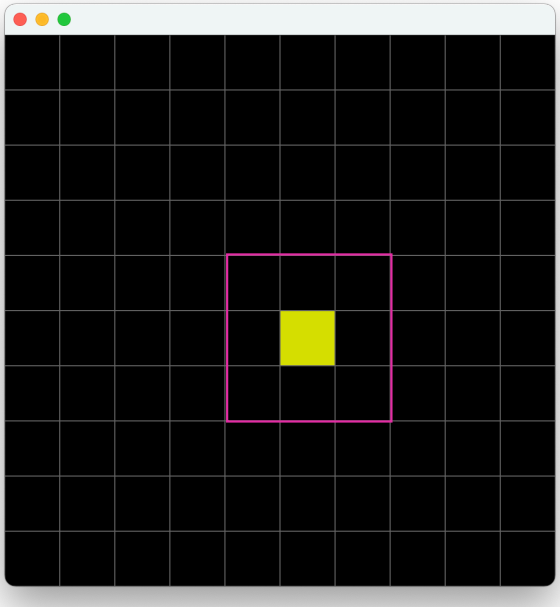
```
int rule(int row, int column) {
    int liveNeighbors=0;
    for (int i=row-1;i<=row+1;i++) {
        for (int j=column-1;j<=column+1;j++) {
            if (!(i==row && j==column))
                liveNeighbors = liveNeighbors+currentStates[i][j];
        }
    }
}
```



if condition is not true

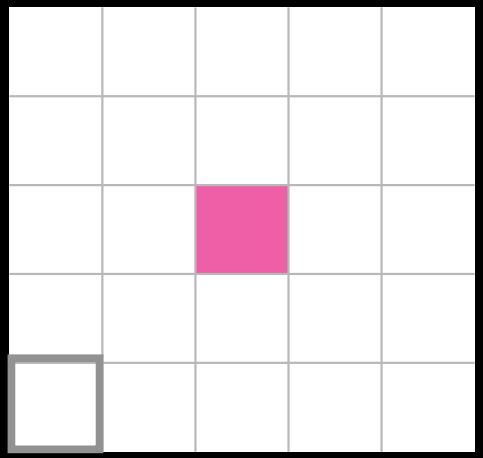
FOR EACH CELL

- ~~Determine how many neighbors are alive~~
- Check rule condition
- Return resulting state



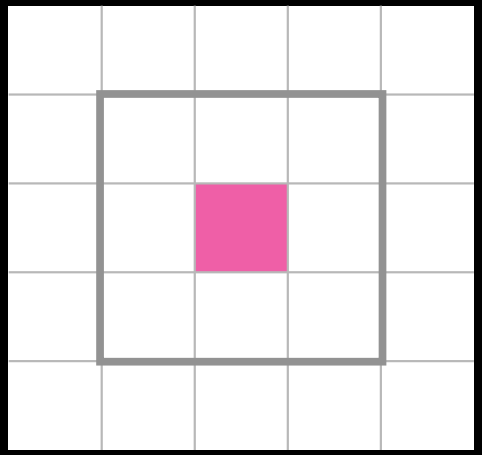
questions?

A 2D CELLULAR AUTOMATON

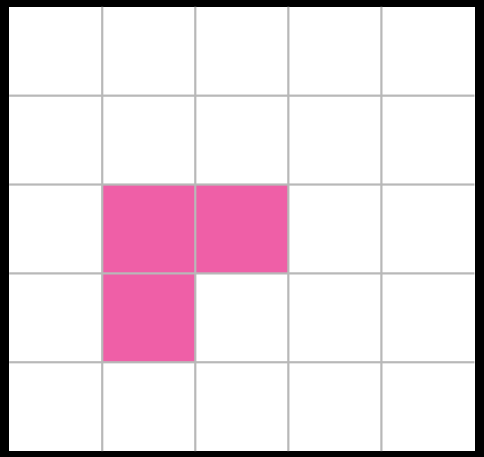


Cell

State: pink = alive
white = dead



Neighborhood



Game of Life Rule:
if alive and 2 or 3 neighbors are alive, stay alive

if dead and 3 neighbors are alive, come alive

rule METHOD: CHECK CONDITION

```
int rule(int row, int column) {
    int liveNeighbors=0;
    for (int i=row-1;i<=row+1;i++) {
        for (int j=column-1;j<=column+1;j++) {
            if (!(i==row && j==column))
                liveNeighbors = liveNeighbors+currentStates[i][j];
        }
    }
    if (currentStates[row][column]==ALIVE && (liveNeighbors==2 || liveNeighbors==3))
        return ALIVE;
}
```

if alive and 2 or 3 neighbors alive, stay alive

rule METHOD: CHECK CONDITION

```
int rule(int row, int column) {
    int liveNeighbors=0;
    for (int i=row-1;i<=row+1;i++) {
        for (int j=column-1;j<=column+1;j++) {
            if (!(i==row && j==column))
                liveNeighbors = liveNeighbors+currentStates[i][j];
        }
    }
    if (currentStates[row][column]==ALIVE && (liveNeighbors==2 || liveNeighbors==3))
        return ALIVE;
    if (currentStates[row][column]==DEAD && liveNeighbors==3)
        return ALIVE;
}
```

if dead and 3 neighbors alive, come alive

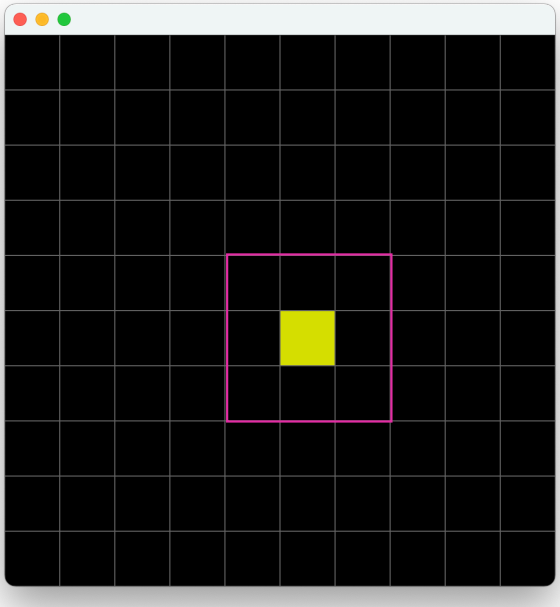
rule METHOD: CHECK CONDITION

```
int rule(int row, int column) {
    int liveNeighbors=0;
    for (int i=row-1;i<=row+1;i++) {
        for (int j=column-1;j<=column+1;j++) {
            if (!(i==row && j==column))
                liveNeighbors = liveNeighbors+currentStates[i][j];
        }
    }
    if (currentStates[row][column]==ALIVE && (liveNeighbors==2 || liveNeighbors==3))
        return ALIVE;
    if (currentStates[row][column]==DEAD && liveNeighbors==3)
        return ALIVE;
    return DEAD;
}
```

otherwise, die or stay dead

FOR EACH CELL

- ~~Determine how many neighbors are alive~~
- ~~Check rule condition~~
- ~~Return resulting state~~



questions?

COMPUTING ONE ITERATION
COMPUTING RULE FOR EACH CELL

ADD AN iterate METHOD

```
void iterate() {

}
```

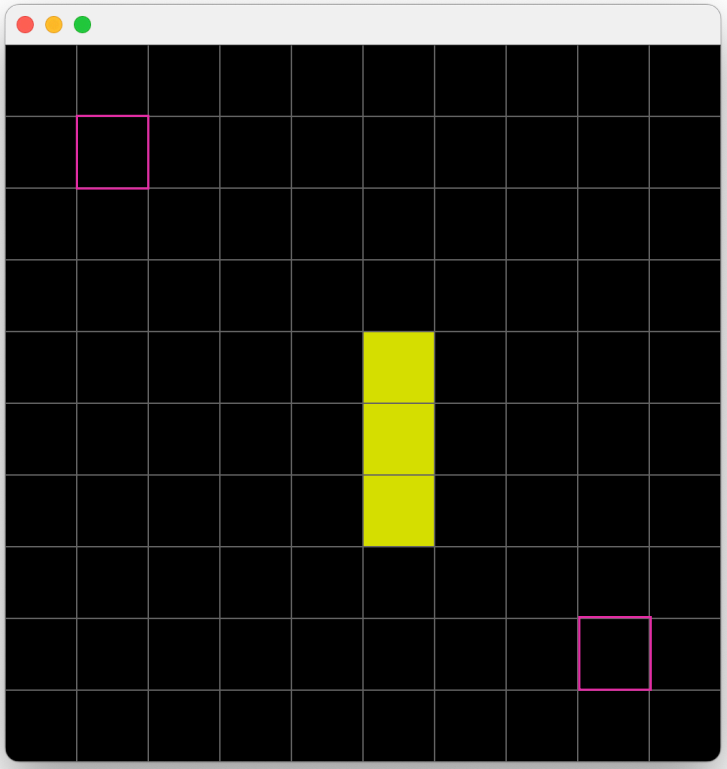
iterate METHOD
COMPUTE RULE FOR EACH CELL

```
void iterate() {
    for (int i=1;i<size-1;i++) {
        for (int j=1;j<size-1;j++) {
            nextStates[i][j] = rule(i,j);
        }
    }
}
```

note loop conditions

start at 1

stop at size-1



iterate METHOD
SET currentStates to nextStates

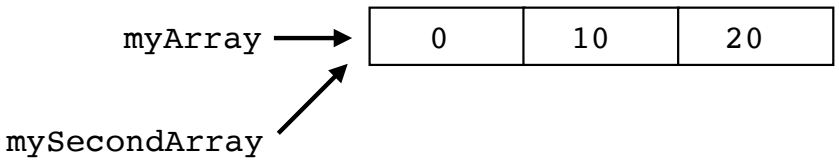
```
void iterate() {
    for (int i=1;i<size-1;i++) {
        for (int j=1;j<size-1;j++) {
            nextStates[i][j] = rule(i,j);
        }
    }
    currentStates = nextStates.clone();
}
```

see any problems?

ARRAYS and REFERENCES

```
int[] myArray = {0,10,20};
int[] mySecondArray;
mySecondArray = myArray;
```

in the computer's
memory somewhere

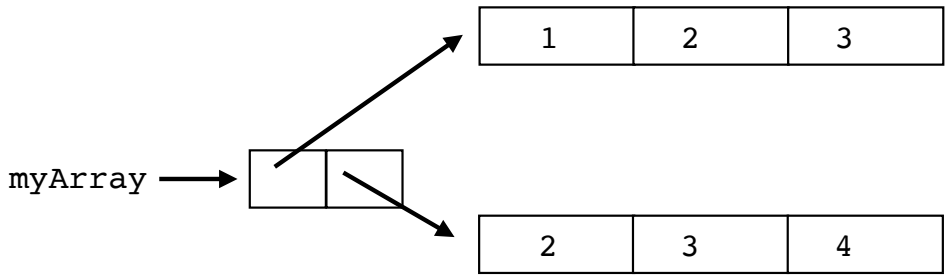


2D ARRAYS ARE ARRAYS OF ARRAYS

2D ARRAYS: ARRAYS OF ARRAYS

```
int[][] myArray = {{1,2,3},
                   {2,3,4}};
```

in the computer's
memory somewhere

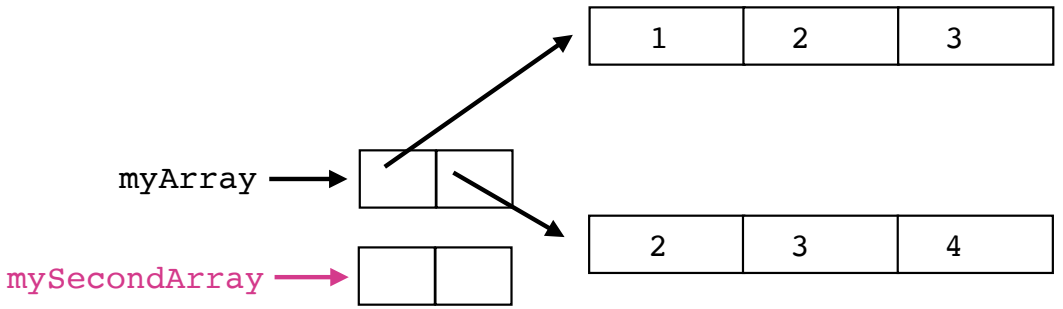


2D ARRAYS: ARRAYS OF ARRAYS

```
int[][] myArray = {{1,2,3},
                   {2,3,4}};

int[][] mySecondArray;
mySecondArray = myArray.clone();
```

clone() creates a
“deep” copy
of the array

in the computer’s
memory somewhere



2D ARRAYS: ARRAYS OF ARRAYS

```
int[][] myArray = {{1,2,3},
                   {2,3,4}};

int[][] mySecondArray;
mySecondArray = myArray.clone();
```

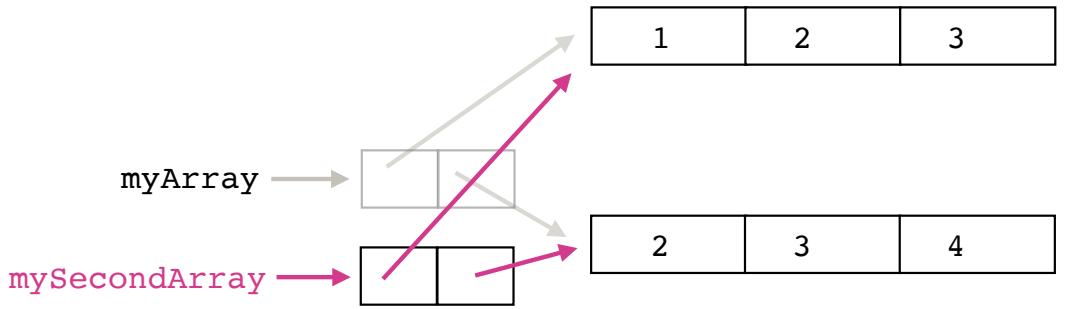
but it is an

array of arrays

an array of references!

the actual data is not copied

in the computer's
memory somewhere



THIS CODE WON'T WORK!

```
void iterate() {
    for (int i=1;i<size-1;i++) {
        for (int j=1;j<size-1;j++) {
            nextStates[i][j] = rule(i,j);
        }
    }
    currentStates = nextStates.clone();
}
```

questions?

A SIMPLE FIX: ANOTHER LOOP

```
void iterate() {
    for (int i=1;i<size-1;i++) {
        for (int j=1;j<size-1;j++) {
            nextStates[i][j] = rule(i,j);
        }
    }
    for (int i=0;i<size;i++) {
        for (int j=0;j<size;j++) {
            currentStates[i][j]  = nextStates[i][j];
        }
    }
}
```

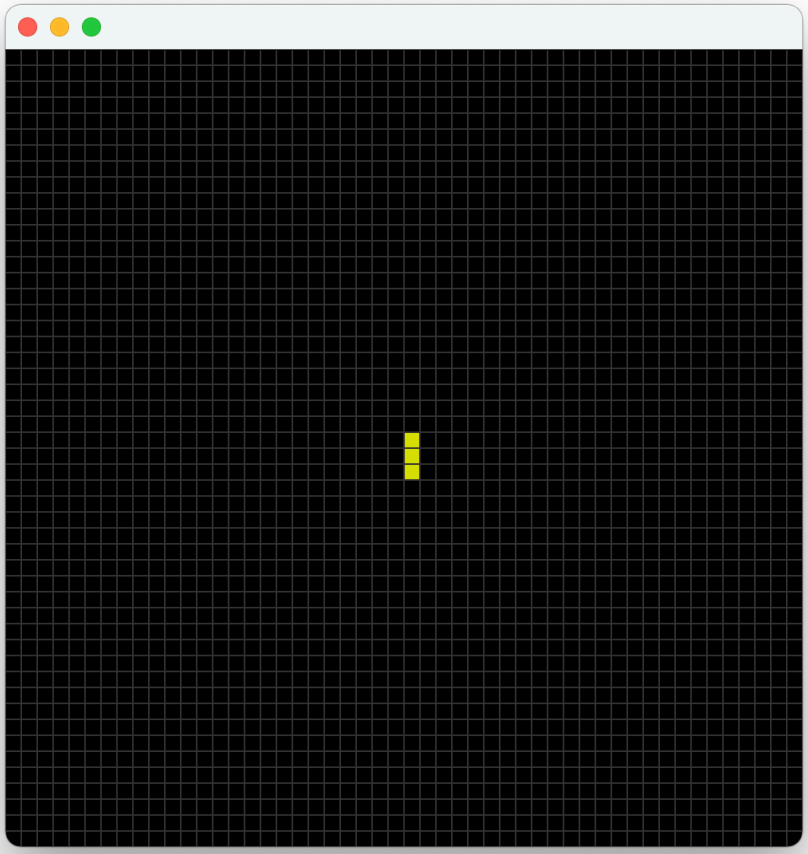
questions?

PUTTING IT ALL TOGETHER

WHAT WE DID TO CODE THE CA

- Created the class
- Added instance variables
- Added a constructor
- Added a display method (draws cells in window)
- Added a rule method (computes rule for one cell)
- Added iterate method (computes rule for all cells)

- Last step: display and animate iterations

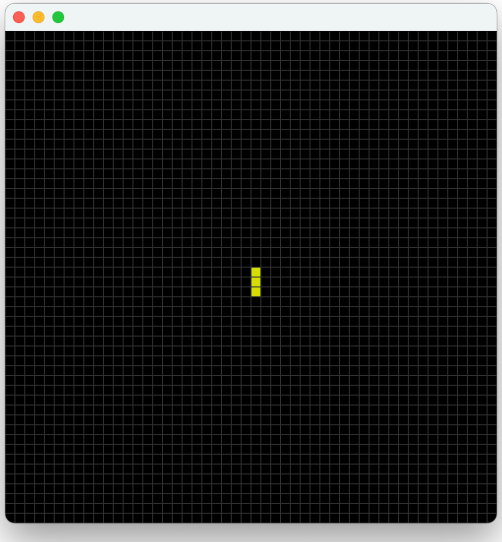


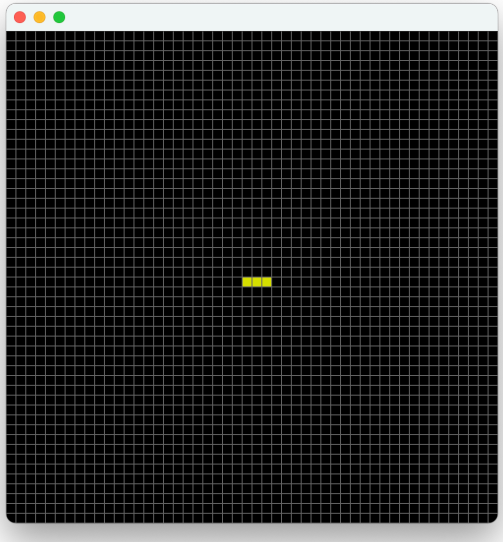
ITERATE IN paintComponent

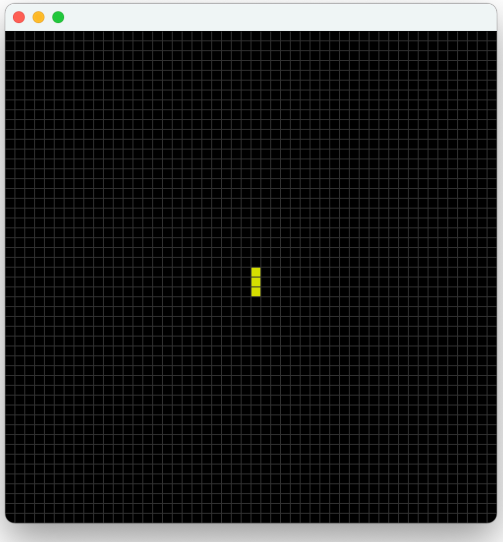
```
@Override
protected void paintComponent(Graphics g) {
    displayCurrentStates(g);
    iterate();
}
```

ANIMATE IN main

```
public static void main(String[] args) {
    CellularAutomata2DTest CA2D = new CellularAutomata2DTest();
    MyFrame frame = new MyFrame(CA2D);
    CA2D.animate(1);
}
```







questions?

Thank you!

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/