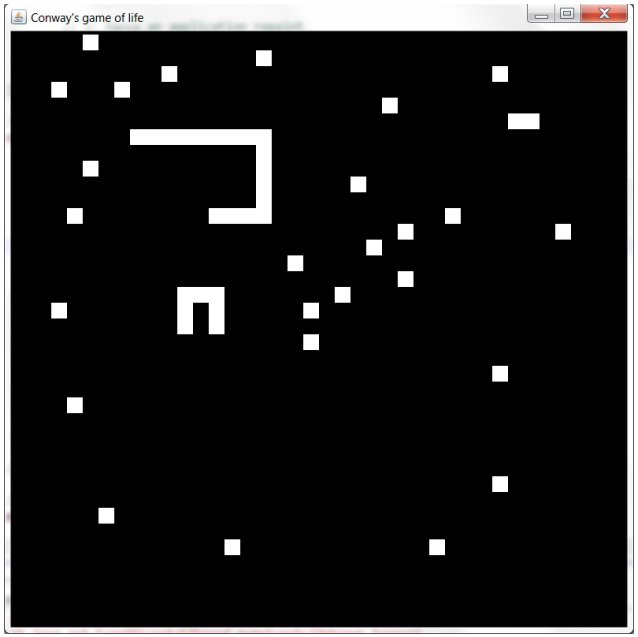
CT255 / NGT2

Week 7
[2D Games in Java]

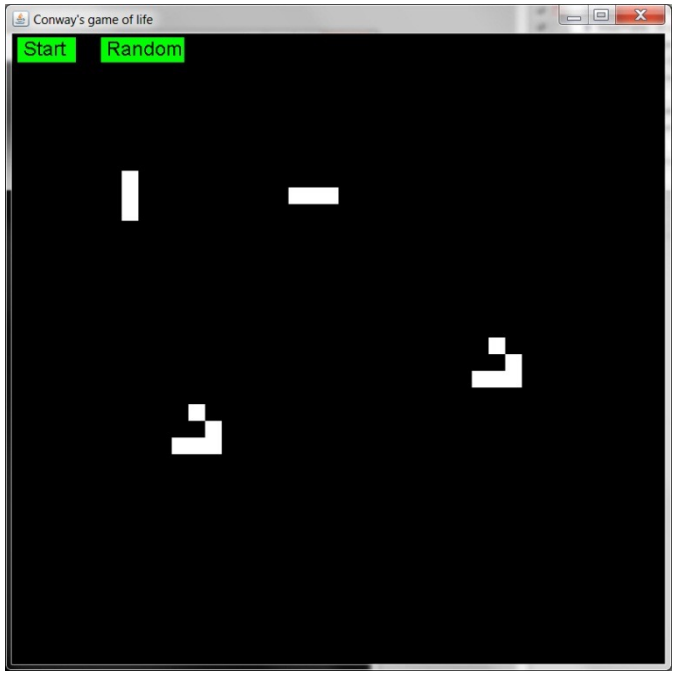
Dr. Sam Redfern
sam.redfern@universityofgalway.ie

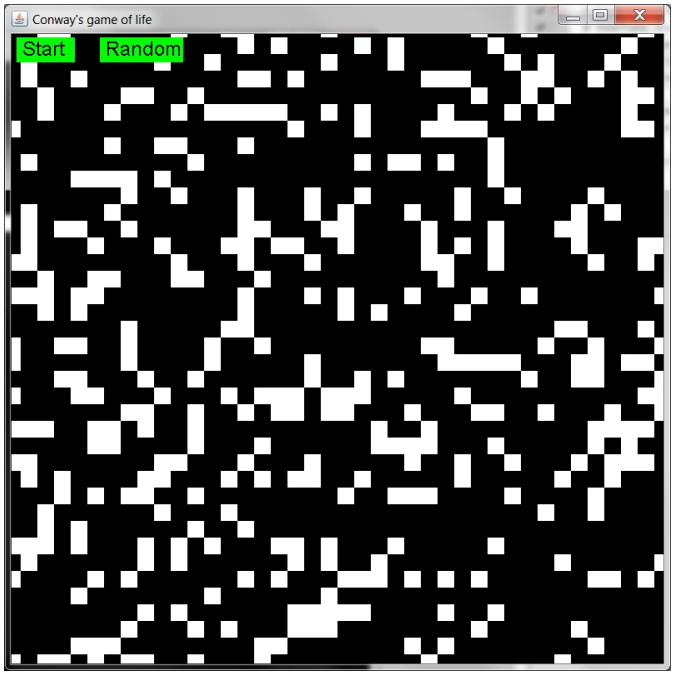
Last week's assignment
Starting the Game of Life

- Create a new Java project, with a main application class that extends JFrame and implements Runnable and MouseListener
- The window should be 800x800 pixels in size
- Use double buffering to avoid flicker when we animate [next week]
- Implement periodic repainting (and later, animation) via a Thread
- Create a 2 dimensional array to store the game state: e.g. a 40x40 array of boolean
- When the mouse clicks on the window, toggle the state of the game state cell at that position (i.e. true becomes false, and false becomes true)
- The paint method should paint, as a rectangle, each game state cell that is currently 'true'



This week's assignment





- Add game states (playing and not playing)
- When not playing, render two rectangles as 'buttons'
- Modify the mousePressed method so that it checks for clicks on the button's regions
  - Start – switches the game state to 'playing'
  - Random – randomises the game state
- When in playing state, apply the rules of Conway's Game of Life at each repaint (see next slide)

Conway's Life: Rules

1. Any live cell with fewer than two live neighbours dies, as if caused by under-population.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

- We will need to iterate through all game cells, counting the amount of live neighbours that each has, before applying the above rules

Conway's Life: Rules

- Each generation (iteration) is created by applying the above rules simultaneously to every cell in its preceding generation: births and deaths occur simultaneously
- To implement this properly, we will need to have two separate game states in memory:
  - one is the 'front buffer' that we're currently displaying, and which we are checking the above rules on
  - the other is the 'back buffer' that we're applying the results of the rules to.
  - the 'back' is switched to 'front' after applying the rules to every cell

```
private boolean gameState[][][] = new boolean[40][40][2];
```

Checking the 8 neighbours of each cell

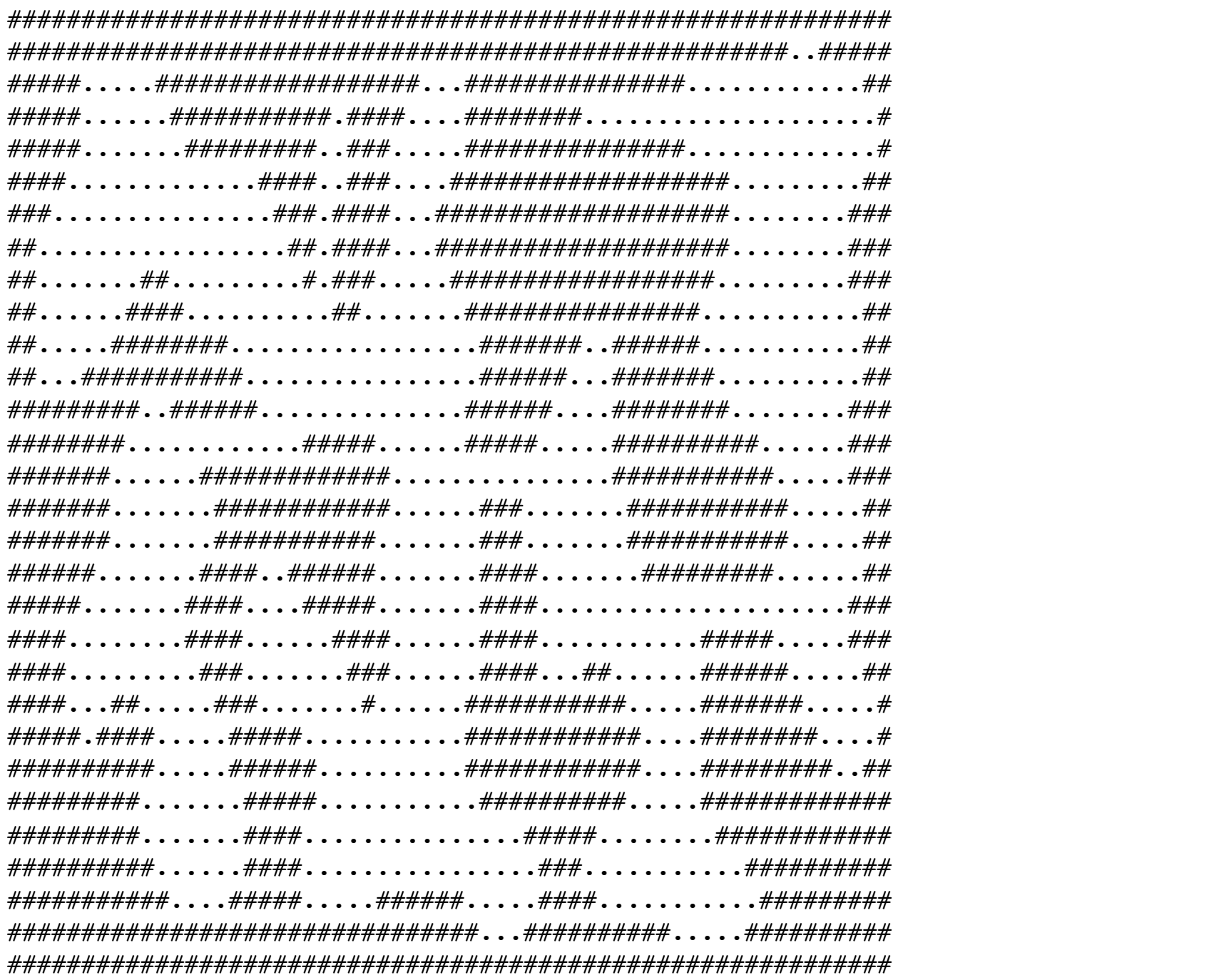
```
for (int x=0;x<40;x++) {
    for (int y=0;y<40;y++) {
            // count the live neighbours of cell [x][y][0]
            for (int xx=-1;xx<=1;xx++) {
                for (int yy=-1;yy<=1;yy++) {
                    if (xx!=0 || yy!=0) {
                        // check cell [x+xx][y+yy][0]
                        // but.. what if x+xx==-1, etc. ?

                    }
                }
            }
     }
}
```

NB we need to define the neighbours for cells at the edges of the map. The usual procedure is to ‘wrap around’ to the opposite side.

Another example of a Cellular Automata algorithm in use

- The image on the next slide is of an algorithmically-generated cave-like structure, for use in a 2D computer game. Each of the cells, laid out in a 60x30 grid, either has a wall (denoted by '#') or a floor (denoted by '.').

- The cellular automata algorithm which generated this output uses the following steps:
  - For each cell, randomly define it as: *wall* (60% chance) or *floor* (40% chance)
  - Perform the following procedure 4 times:
    - Calculate the number of wall neighbours of each cell, and define each cell which has at least 5 neighbouring wall cells, as a wall cell itself. Otherwise (i.e. if it has less than 5 wall neighbours) define it as a floor cell.



Exam Question (2017)

- Your task is to write a Java class to implement this cellular automata algorithm:

- The class should store the cave-like structure in suitable member data
- The data should be randomly initialized according to the 1st step of the algorithm indicated above.
  - Hint: use Math.random() to generate a random float between 0 and 1
- The 2nd step of the algorithm (which repeats 4 times) should be implemented. You should pay particular attention to array bounds when examining a cell's neighbours.
- The resulting data should be printed to the console, (using System.out.println) as the '#' and '.' symbols, as shown below.

“Genetix”

An artificial life program I wrote a while ago (1998) – in order to learn Java!

This Artifical Life program simulates the evolution of a population of abstract creatures (*'agents'*). The 'genetic make-up' of each agent is defined by its speed and vision abilities, which determine how fast it can move and how far it can see. The colour of an agent reflects its genetics- the greener the agent, the better its vision is; the redder an agent is, the faster it can move. When the program starts, all agents have speed and vision scores of 1, and appear as khaki-green blobs.

In order to survive, agents must eat food, and on each move (*'epoch'*) an agent will move towards the greatest source of food that is within its vision range. Food is depicted by grey blobs: light grey indiciates a strong food source, while dark grey indicates a weak food source.

Healthy agents may reproduce (asexually). In most cases, an agent's offspring will be identical to it; occasionally, a newly born agent may have either its speed or its vision abilities increased or decreased (*'mutated’*).

Before running the program, you can decide the number of food deposits, the size of each deposit, the speed at which food replenishes after being eaten, and the number of agents.

http://www2.it.nuigalway.ie/~sredfern/genetix.html

Some examples of “Genetix” running

- “Conquest”
  - in this example, the effect of population pool size is evident as several separate populations develop on the 'islands' of food, and the agents from the larger islands eventually discover and conquer the less advanced agents from the other islands.
  - https://www.youtube.com/watch?v=30ztf6bMZSY

- “Extinction”
  - in this example, slow-growing food leads to cycles of population explosion, famine, and mass migration. Eventually, the instability of this model becomes evident as total extinction occurs.
  - https://www.youtube.com/watch?v=RJv0Z-sO17o