

# CS 351

## Design of Large Programs

### *Cellular Automaton: Conway's Life*

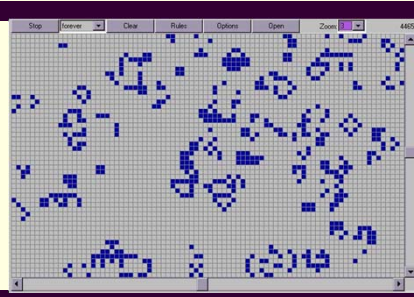
Instructor:

Joel Castellanos

e-mail: [joel@unm.edu](mailto:joel@unm.edu)

Web: <http://cs.unm.edu/~joel/>

Office: Farris Engineering  
Center (FEC) room 319



2/2/2017

## Quiz: Collections Video

In the assigned Lynda video on collections, describe **one** of the following:

- 1) Something the narrator said that was incorrect in substance or by omission.
- 2) Somethings that was poorly stated: either because it confused you or because it was about something you already knew and you can see how someone who did not know what you do would likely be confused by what was said.

## Abstraction

---

- In computer science, what is abstraction?
- What are some examples of abstraction?
- What are some programming languages that use more abstraction and less abstraction than Java?
- What are advantages and disadvantages of abstraction?
- In Java, what is an example of when it is best practice to use more abstraction? What is an example of when it is best to use less abstraction?

3

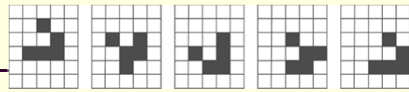
## A Cellular Automaton

---

- A discrete model consisting of a regular grid of cells, each in one of a finite number of states at any discrete time.
- The grid can be in any finite number of dimensions.
- The state of a cell changes discretely at each time step and is a function of the states within a finite neighborhood of cells.
- All cells have the same definition of a neighborhood and use the same set of rules for transitioning from one state to another.
- A "generation" is the state of all cells in the grid after the transition rules have been applied to all cells in the grid.
- Cellular Automaton are studied in
  - Computability Theory
  - Mathematics
  - Theoretical Biology
  - Microstructure Modeling

4

## Conway's Game of Life



- The Game of Life is a cellular automaton devised by the British mathematician John Horton Conway in 1970. It is the best-known example of a cellular automaton.
- The Game of Life is a zero-player game, meaning that its evolution is determined by its initial state, needing no input from human players. One interacts with the Game of Life by creating an initial configuration and observing how it evolves.

5

## Conway's Game of Life - Rules

- Finite two-dimensional grid of square cells.
- Each cell is in one of two possible states, **Alive** or **Dead**.
- Every cell interacts with its 8 adjacent neighbors.
- At each step in time, the following transitions occur:
  - 1) Any **alive** cell with fewer than two live neighbors **dies**, as if by loneliness.
  - 2) Any **alive** cell with more than three live neighbors **dies**, as if by overcrowding.
  - 3) Any **alive** cell with two or three live neighbors stays **alive**.
  - 4) Any **dead** cell with exactly three live neighbors **comes alive**, as if a baby is born to one of the neighbors into the empty cell.
- The initial pattern constitutes the 'seed' of the system.
- The first generation is created by applying the above rules simultaneously to every cell.


1	2	3
4		5
6	7	8

6

## Conway's Game of Life - Example

1. Any **alive** cell with fewer than two live neighbors **dies**.
2. Any **alive** cell with more than three live neighbors **dies**.
3. Any **alive** cell with two or three live neighbors stays **alive**.
4. Any **dead** cell with exactly three live neighbors **comes to life**.

	0	1	2	3	4
0	0	0	0	0	0
1	1	2	3	2	1
2	1	1	2	1	1
3	1	2	3	2	1
4	0	0	0	0	0



	0	1	2	3	4
0					
1					
2					
3					
4					

7


## Synchronous Update: Two Copies of Grid

**Synchronous update** means that the update rules must be implemented such that the result is the same as if all cells were updated simultaneously.

Implementing synchronous update usually requires building the next generation in a temporary grid.

For example, a cell that is dying on this update may also serve as the neighbor count for a different cell that either dies or that comes to life on the same update.

0	0	0	0	0
1	2	3	2	1
1	1	2	1	1
1	2	3	2	1




8

## Data Structure for Game of Life

Use a **boarder** to avoid treating the edges as special cases.

The boarder is used when counting neighbors, but is never updated.

This figure shows a 5×5 grid of cells in a 7×7 array.

	0	1	2	3	4	5	6
0		-1,-1	0,-1	+1,-1			
1		-1,0	0,0	+1,0			
2		-1,+1	0,+1	+1,+1			
3							
4							
5							
6							

9

## Multi-Threaded Game of Life Requirements (1 of 5)

- Implement the game of life on a 10,000 × 10,000 grid.
- Fill inside window with cells zoomable (with mouse wheel) from 1 × 1 pixels to 50x50 pixels.
- Window must be resizable.
- User must be able to scroll window through full grid using either click-&-drag or scroll bars (or both).
- Seed with random values (each cell with 50% chance of being alive at start).
- When paused, user can click in a cell to toggle life.
- Must run making good use of 1 to 8 cores with 1 to 8 threads for workers plus one thread for GUI.

10

## Multi-Threaded Game of Life Requirements (2 of 5)

- Operation must be smooth and real-time.
  - Depending on your hardware capabilities and your viewing window size, you may or may not get 30 to 60 frames per second.
  - GUI must remain responsive on all platforms.
- GUI Controls:
  - Pause/play, next, reset, and 6 presets.
  - Number of worker threads (1 through 8). This could be a popup that can only be set at startup or could be in main window, but it is expected that changing this would restart the board.

11

## Multi-Threaded Game of Life Requirements (3 of 5)

- JavaFX: For windows, containers and widgets.
- Gridlines:
  - When cells are viewed at a size of 5×5 pixels, use one pixel row and column for gridlines (Draw the cells 4x4).
  - Cells viewed larger than 5×5, \*may\* have wider gridlines.
  - When cells are less than 5×5, do not show gridlines.

12

## Multi-Threaded Game of Life Requirements (4 of 5)

### Age:

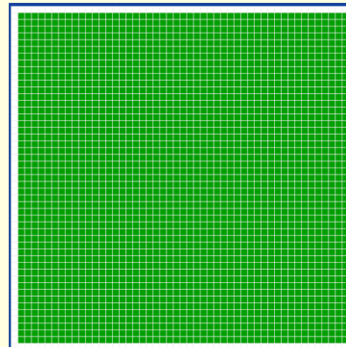
- Cells in your implementation have 10 states: 0=dead, 1=born in the current generation, 2=born last generation, ... 10=born nine or more generations ago.
- Each generation must have a different shade of the same color with younger cells being more different from the background.

13

## Multi-Threaded Game of Life Requirements (5 of 5)

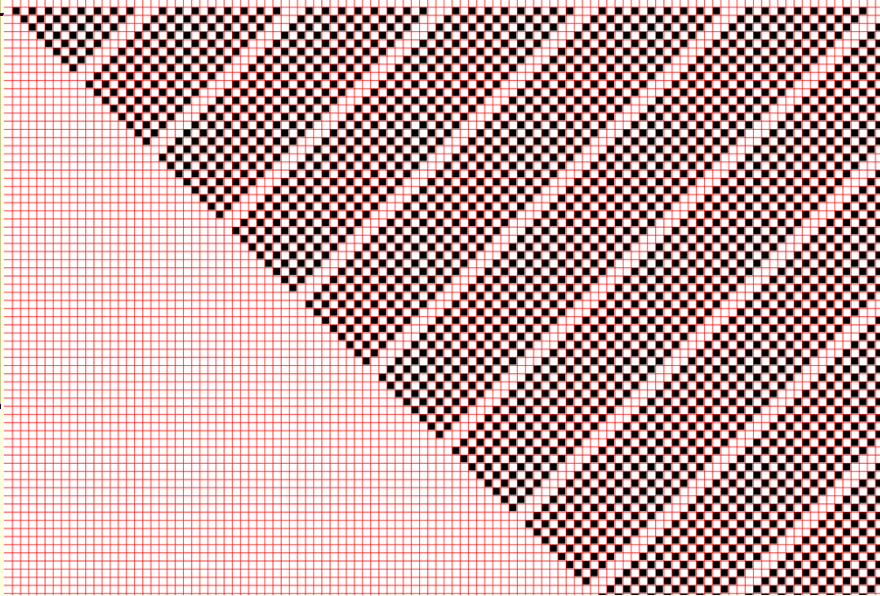
### ■ Presets (Must be selectable from drop-down menu):

- 1) All dead (for easy click building)
- 2) Random (50% alive)
- 3) Glider gun (of your choice). Gun in upper left, shooting toward lower right.
- 4) All Alive except edges.
- 5) Upper-Right checkerboard (see next slide)
- 6) Something you think is cool.  
If you want, you are welcome to add additional presets.



14

## Upper-Left Checkerboard Preset



15

## Quiz: Derived Requirements

When software specifications do not provide all details pertaining to a requirement, then the software engineer is free to:

- a) Ignore the requirement since, being underspecified, is invalid.
- b) Leave the unspecified details undone as they are not requirements.
- c) Do whatever she or he wants.
- d) Choose between reasonable alternatives or ask
- e) Assume the requirement author is an incompetent idiot and respecify the project in his or her image.

16



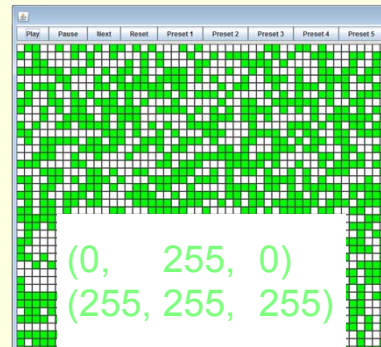
## Derived Requirements: Game of Life (1 of 5)

Must be good brightness contrast between Live and dead cells.  
For example,

- Poor Brightness Contrast: **bright green** on white.
- Good Brightness Contrast: **medium** or **dark green** on white.



Poor  
choices



17

## Derived Requirements: Game of Life (2 of 5)

**Zoom** should zoom in and out while keeping either the **curser** or **display center** fixed.

- Zooming should NOT move upper-left world to upper-left window.
- Zooming should NOT be centered about the upper-left window.

**Selecting a preset** should **reset** and **refresh** display with visible window showing interesting part of the preset. The preset must be started in a **location** that allows it to work.

**Pause** should **change something** visible in the GUI to show the GUI is paused and pending updates should be **halted**.

**Console output** should exist and be politely informative:  
"loading preset X", ... MUST NOT run like diarrhea.

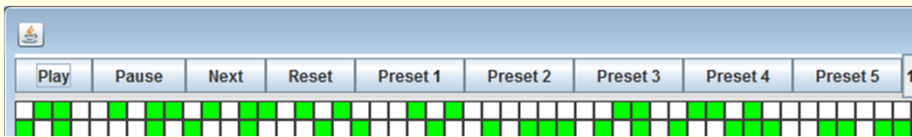
18

## Derived Requirements: Game of Life (3 of 5)

**Do not crash or stream Errors:** when the user drags the mouse outside the window or other *reasonable* but unspecified actions.

**Window Resizing** implies auto *adjusting the GUI layout* to fit the new size. It is okay if your GUI looks stupid at stupid sizes (i.e. 10 x 2000 pixels.)

**Window Title:** A window without a title looks unfinished.



19

## Derived Requirements: Game of Life (4 of 5)

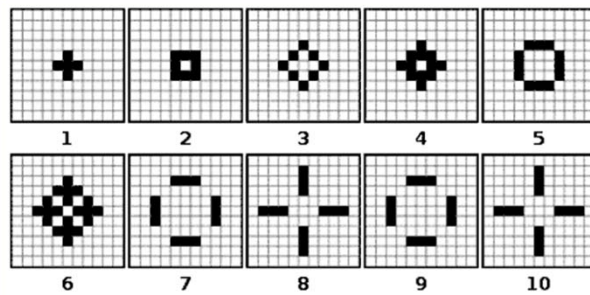
**Good Use of up to 8 Cores** implies:

- When run on 1 core, your program should be efficient:
  - At least 1 frame/sec in lab on 10k × 10k grid.
  - Only one core should be maxed out and only one other core should be working at ~25% doing GUI work (with spikes when the user is scrolling).
- Your program should run 6 or 7 times faster on 8 cores with 8 worker threads than on 1 core.
- When your program is paused, it should be using almost zero CPU power.

20

## Most Important Derived Requirement (5 of 5)

The patterns shown all over the Internet for Conway's Game of Life must work in your implementation.



These sequences are *implied* by the required game rules.

21

## Implementation Suggestions (1 of 2)

- Store game state on “board” that is 2D array of primitive types (i.e. byte). A array of objects (on a 64-bit machine) will use  $10,000 \times 10,000 \times 8 = 800\text{MB}$  just to store the pointers to all the grid objects!
- Use two game state boards total for the whole program: one for reading the current generation, one for writing the next.
- Reading can be done by all threads sharing the same board. It is ok if read areas overlap.
- Writing can be done by all threads sharing the same board, as long as each thread's write area is 100% disjoint.
- Access memory to leverage CPU cache and page size.
- Minimize computation of most common task: is this cell alive?
- Let background be cell borders (rather than drawing a grid).

22

## Implementation Suggestions (2 of 2)

Use synchronized `generationDone()` method to control synchronization between generations:

- Called when a thread finishes its generation.
- Decrement `threadCount`.
- If `threadCount > 0`, return `threadCount` and worker sleeps until notified.
- if `threadCount == 0`,
  - Swap current grid with next generation grid (just swap global pointers).
  - Set `threadCount` to number of worker threads.
  - Notify all threads to wake up and start working.
  - Return 0 so thread that called this method knows not to sleep.

23

## Implementation Suggestions (3 of 3)

Overview Package **Class** Use Tree Deprecated Index Help

Java™ Platform  
Standard Ed. 7

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util.concurrent

### Class `CyclicBarrier`

This java built-in class does much of what is outlined on the previous slide

java.lang.Object  
java.util.concurrent.CyclicBarrier

```
public class CyclicBarrier  
extends Object
```

A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point. `CyclicBarriers` are useful in programs involving a fixed sized party of threads that must occasionally wait for each other. The barrier is called *cyclic* because it can be re-used after the waiting threads are released.

A `CyclicBarrier` supports an optional `Runnable` command that is run once per barrier point, after the last thread in the party arrives, but before any threads are released. This *barrier action* is useful for updating shared-state before any of the parties continue.

24

**Sample usage:** Here is an example of using a barrier in a parallel decomposition design:

## Grading Rubric (50 points total)

0	[Turn-in: -5 points]
0	[Code Style: -10 points] Repeated code, poor class structure...
5	Implement the game of life on a 10,000 × 10,000 grid.
2	Window resizeable, where larger window shows more cells.
3	Zoomable from 1×1 pixels to 50×50 pixels per cell.
2	Age (with color as specified).
3	Scroll window through full grid.
1	Start up: Random 50% alive and paused.
4	When paused, user can click in a cell to toggle life.
14	Must run making good use of 1 to 8 cores with 1 to 8 threads for workers plus GUI and main threads.
9	Operation must be smooth and real-time.
1	GUI Controls: Pause/play, next, reset.
2	GUI Controls: Number of worker threads (1 through 8).
2	GUI Controls: 6 presets
2	Gridlines

25