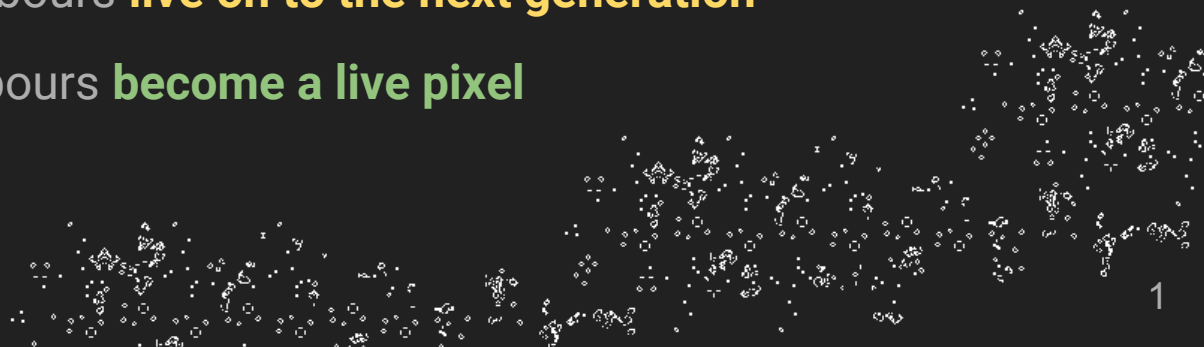
Mosaic Cellular Automata

Max Proske
Ian Springer

Conway’s “Game of Life”

A simulation in which a grid of pixels can be **dead**, **alive**, or **born** depending on the state of their neighbours.

- Live pixels with < 2 neighbours **die of under-population**
- Live pixels with > 3 neighbours **die of over-population**
- Live pixels with 2-3 neighbours **live on to the next generation**
- Dead pixels with 3 neighbours **become a live pixel**



Project Overview

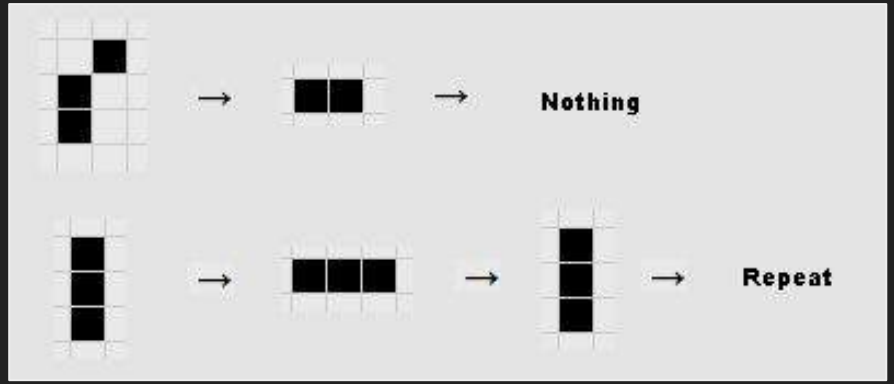
Our method recreates an image as a **living, breathing, infinitely sustainable** simulation of Conway's "Game of Life"

- Given an image as input, create an initial grid of alive and dead pixels
- Run the result as a "Game of Life" simulation, requiring **no further input**

Method.java

Method Core

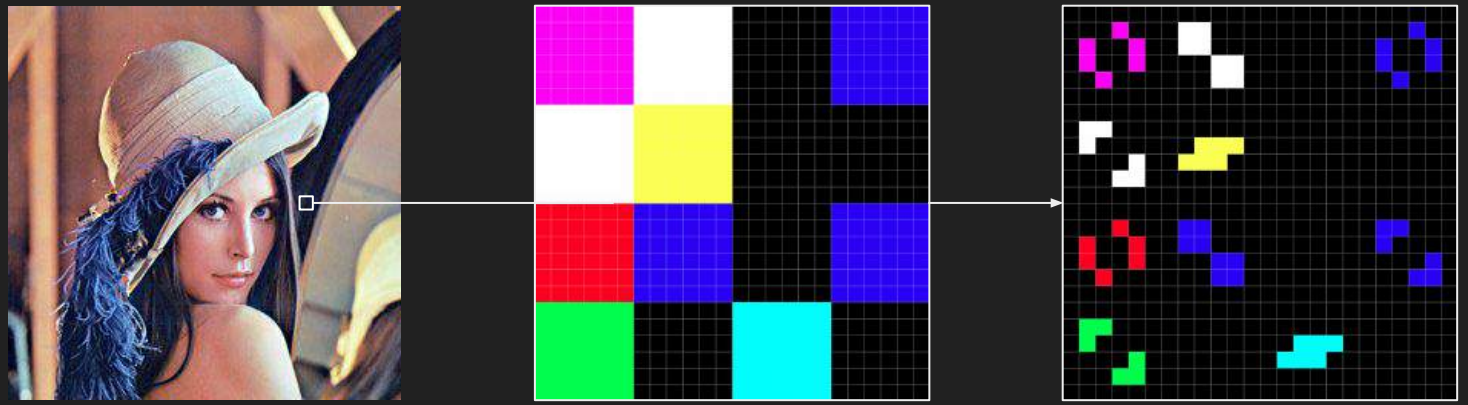
- An **oscillator** is a pattern of pixels that are killed and born in perfect equilibrium, such that it sustains itself **indefinitely**



Method Foundation

The smallest complex oscillators are **6x6 pixels**, our **atomic unit**

- Enlarge an image by 6x
- Replace each pixel with a **6x6 oscillator**



How can we...

- Combine three 6x6 oscillators into a **17x5 oscillator**?
- Create **complex life forms** that move around the image?!
- Render **2^{22} cells** (4.19 million objects) in **real-time**?!!!

… All while maintaining the **integrity** of the image.

Filters.java

Ordered Dither Threshold Filter

Dithering fits an image to a **limited colour palette**

- We weren't interested in recoloring our image

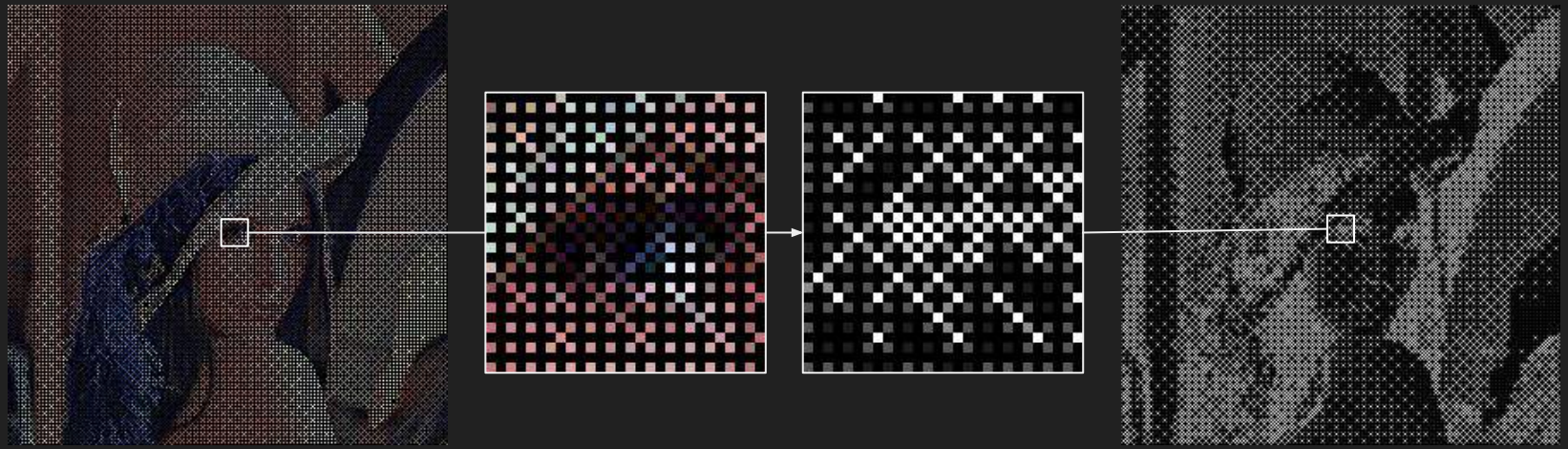
We're interested in the distinct **crosshatch** byproduct

- **Suppress pixel values** below the dither threshold value
- This allows us to calculate the **population density**

Population Density Matte

Assigns a pixel value based on the number of **adjacent pixels**

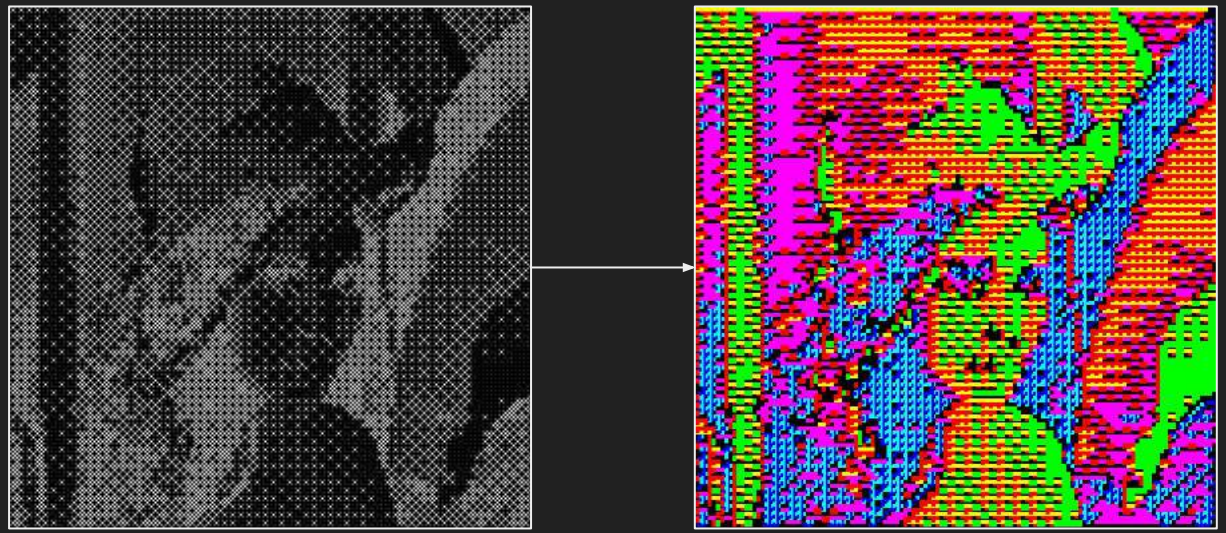
- This allows us to group pixels into **similar neighbours**



Similar Neighbour Matte

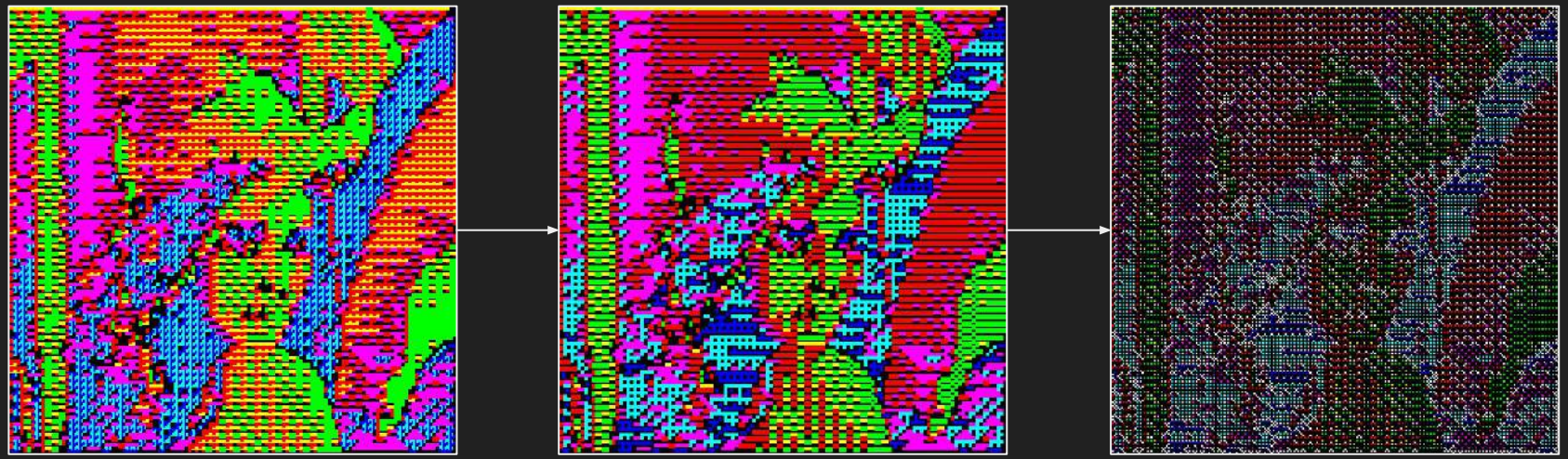
Merges pixels with similar neighbour density into **cells**

- This allows us to form the **bounding boxes** of larger oscillators



Bounding Box Matte

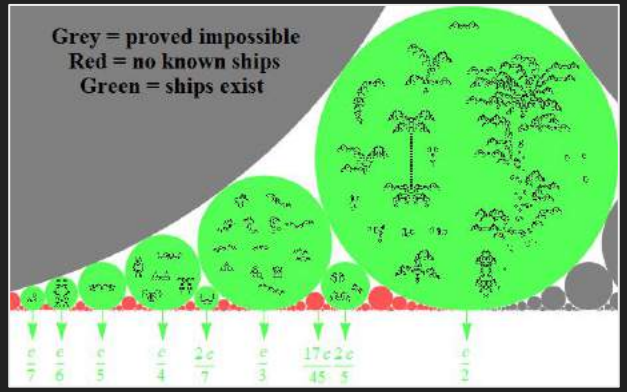
- Order of importance between **overlapping cells**
- Unmerged cells become atomic 6x6 oscillators
- **Each pixel** becomes an **instruction** of where to place a **specific size oscillator**



Patterns.java

Research

- What are the rules? What **complex life** is possible?
- “Computational Methods For Conway's Game of Life Cellular Automaton.” 2013. Oxman, G., Weiss, S., Be’ery, Y.
- “Variations on Conway’s Game of Life and Other Cellular Automata.” 2012. Hua, D., Pelikan, M.



Research

- **LifeWiki** – the most comprehensive archive of patterns on the internet.
- Compiled a list of **250+** oscillators by **size**, **period**, **heat**, **volatility**, etc.
- Implemented in code those with **long, interesting lifespans**

Max's List of "Game of Life" Oscillators

	6x6 Cells Required	Oscillator	Image	Bounding Box	Bounding Box of 6x6 Cells	Periods	Volatility
yes, atomic oscillator	1	Beacon		6x6	1x1	2	2
yes, atomic oscillator	1	Clock		6x6	1x1	2	2
yes, atomic oscillator	1	Toad		6x6	1x1	2	2
no, too small	1	Blinker		5x5	1x1	2	2
no, just combined patterns	2	Killer_toads		6x11	1x2	2	1
no, just combined patterns	2	Cis-beacon_and_cap		6x10	1x2	2	1
no, just combined patterns	2	Cis-beacon_and_table		6x9	1x2	2	1
no, lifespan too boring	4	29P9		12x12	2x2	9	2.25
yes, interesting lifespan, volatile	4	Figure_eight		12x12	2x2	8	2
yes, interesting lifespan, volatile	4	A_for_all		12x12	2x2	6	1.5
no, lifespan too boring	4	Block_frob		12x12	2x2	4	1
	4	Cross		12x12	2x2	3	0.75

...

no, too small	6	Mathematician		11x13	2x3	5	0.83
no, too small	6	Boat_on_spark_coil		10x13	2x3	2	0.33
no, too small	6	20P2		10x13	2x3	2	0.33
no, too small	6	Mold_and_long_hook_eating_tub		8x16	2x3	12	2
maybe, too boring	8	Blocked_p4-2		24x10	4x2	4	0.5
yes, interesting lifespan, volatile	8	Queen_bee_shuttle		24x9	4x2	30	3.75
no, too boring	8	Blocked_p4-3		23x11	4x2	4	0.5
no, too boring	8	Killer_candlefrobras		23x7	4x2	3	0.38
yes, interesting lifespan, volatile	8	Caterer_on_figure_eight		22x12	4x2	24	3
no, too boring	8	Ellison_p4_HW_emulator_hybrid		22x11	4x2	4	0.5
no, too boring	8	Blocked_p4-4		21x12	4x2	4	0.5
maybe, too boring, volatile	9	Mold_on_pentadecathlon		18x18	3x3	60	6.67
no, too boring	9	44P7.2		18x18	3x3	7	0.78

Application.java

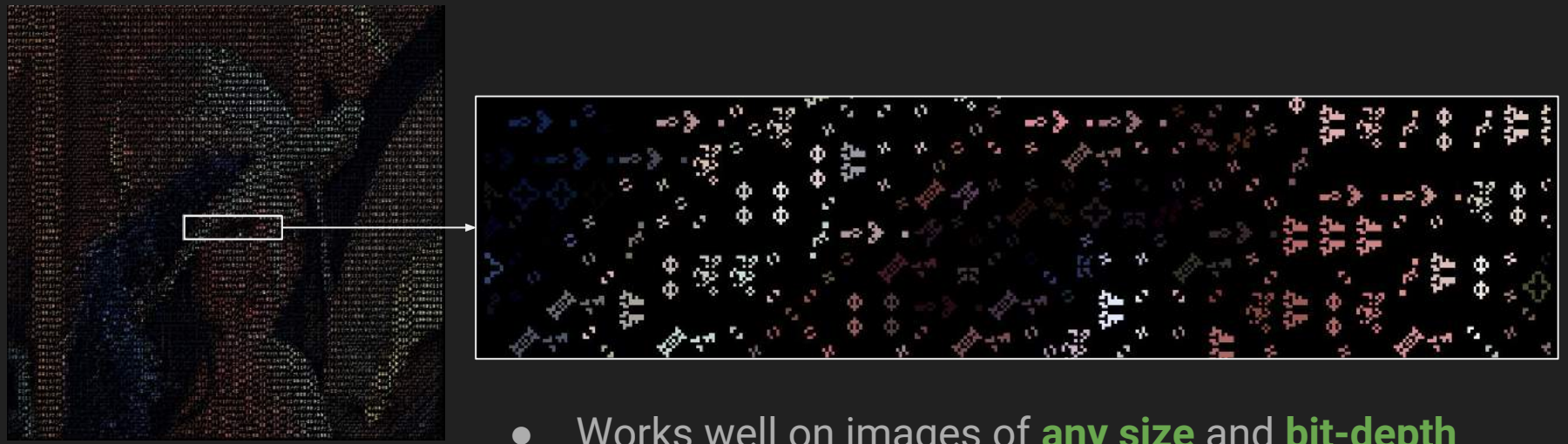
Feeding Life

- Passed in a prepared array of cells
- Each cell has a **color** and **alpha value**
- Array is iterated over and fed into the game’s array
  - **0** opacity cells are marked as dead
  - **255** opacity cells are marked as alive

Running Life

- Game tick cycle
- Neighbor checking cycle
- Drawing cycle

Final Result



- Works well on images of **any size** and **bit-depth**
- For images with **little color**, or **lots of color**
- Image integrity successfully maintained

Reflection.java

Application Challenges

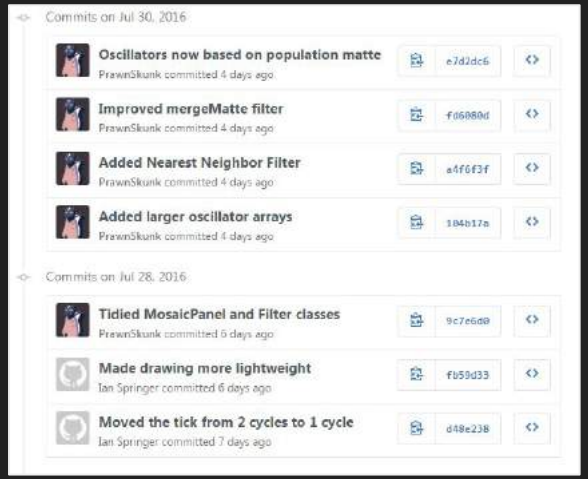
- EFFICIENCY
- EFFICIENCY
- **EFFICIENCY**

Object Name	Average Live Instance Count	Memory Size
java.awt.geom.AffineTransform	6,314,328	454 MB
java.awt.Color	4,718,637	150 MB
Cell	2,359,296	75 MB
java.lang.Object[ ]	2,399	29 MB
int[ ]	7,730	14 MB
...	...	...
Total:	**13,455,070**	**727 MB**

Also other issues...

Method Challenges

- ITERATION
- ITERATION
- **ITERATION**



- Massive planning overhead
- Frustrating to debug when ***one misplaced pixel*** will rip the image apart
- Constant prototyping, testing, refining

Conclusion

- Setting large goals isn’t always a bad thing
- Learned to break down a large application into manageable chunks
- Created our own **original mattes** and algorithms
- Satisfied all of our personal project objectives

End Presentation