A data structure is just a stupid programming language. 
— Bill Gosper

A programming language is a very dangerous data structure. 
— Guy Steele
The Game of Life

- Designed by John H. Conway in 1970
- Infinite grid of cells, each alive or dead.
- Ruleset 23/3: In each generation...
  - Live cells with 2 or 3 live neighbors stay alive
  - Dead cells with 3 live neighbors comes to life
  - All other cells become/stay dead
A survey of Life-forms

J. H. Conway. 20/7/70.
“I used to go around saying, ‘I hate Life,’...
But then I was giving a lecture somewhere,
and I was introduced as ‘John Conway, Creator of Life.’
And I thought, ‘Oh, that’s quite a nice way to be known.’
So I stopped saying ‘I hate Life’ after that.”
If arithmetic overflow is a fatal error, some fascist pig with a read-only mind is trying to enforce machine independence.

Abstract: Contrary to everybody, this self contained paper will show that continued fractions are not only perfectly amenable to arithmetic, they are amenable to perfect arithmetic.
Rule 23/3

Glider [Guy 1969]
Gallery: Spaceships

Glider [Guy 1969]

Lightweight [Conway 1970]

Walrus [Gaucher 2023]

Weekender [Eppstein 2000]

Heavyweight [Conway 1970]

Spider [Bell 1997]

Copperhead [zfind 2016]

Sir Robin [Goucher 2018]
Gallery: Guns and Rakes

Glider Gun [Gosper 1970]

Glider Gun [Simkin 2015]

Gliderless LWSS gun [Merzinich 2023]

Spacerake [Gosper(?) 1971]
Gallery: Breeders

Breeder 1
[Gosper ≤1971]
Gallery: Life in Life

OTCA metapixel [Due 2006]
Video: [Pincombe 2012]
Gallery: Programmable computer

[Loizeau 2016]

write f 224  
write g 2   
write a 5   
write b 3   
write c 0   
not a e    
increment e
add b e d  
increment c
add d b d  
and d f e  
jumpif e   
goto 14    
goto 8    
jumpif d   
goto 24    
not c e    
add b e e  
sign e e  
jumpif e   
goto 23    
add b g b  
goto 4    
print a   
add a g a  
goto 3    

print primes 
starting at 5
Build a working game of Tetris in Conway's Game of Life

Here is a theoretical question - one that doesn't afford an easy answer in any case, not even the trivial one.

In Conway's Game of Life, there exist constructs such as the `metapixel` which allow the Game of Life to simulate any other Game-of-Life rule system as well. In addition, it is known that the Game of Life is Turing-complete.

Your task is to build a cellular automaton using the rules of Conway's game of life that will allow for the playing of a game of Tetris.

Your program will receive input by manually changing the state of the automaton at a specific generation to represent an interrupt (e.g. moving a piece left or right, dropping it, rotating it, or randomly generating a new piece to place onto the grid), counting a specific number of generations as waiting time, and displaying the result somewhere on the automaton. The displayed result must visibly resemble an actual Tetris grid.
Tetris in Life

- Use metapixel to simulate “wired” variant of Life
- Build logic gates in VarLife
- Build a RISC processor from logic gates
- Define assembly language QFTASM
- Define higher-level language Cogol
- Write Cogol-to-QFTASM compiler
- Implement Tetris in Cogol

➔ 2,940,928 × 10,295,296 bounding box, >29 billion live cells!
Data Structures

- 2D array large enough to contain every live cell
  - $O(HW)$ space, $O(HW)$ time per generation

- List of live cells
  - $O(n)$ space, $O(n)$ time per generation (via hashing, ugly)

- Sorted list of live rows, sorted list of live cells in each live row
  - $O(n)$ space, $O(n)$ time per generation (ugly)

- Quadtree!!
Quadtree

Meatball [Charity Engine 2022]
Memoization

[Samuel 1959, Mitchie 1967]
[Samuel 1959, Mitchie 1967]
[Gosper 1984]

Meatball [Charity Engine 2022]
Macro-cells

- Order-\(k\) macro-cell represents a \(2^k \times 2^k\) square of cells
- If \(k \geq 3\), store “pointers” to four order-(\(k-1\)) macro-cells = children
- *Use hashing to avoid duplication*

\[
c.\text{hash} = \text{Hash}(c.\text{nw}.\text{hash}, c.\text{ne}.\text{hash}, c.\text{sw}.\text{hash}, c.\text{se}.\text{hash})
\]
One generation

- 4×4 block at time $T$ determines central 2×2 block at time $T+1$
- Store lookup table of all $2^{16} = 65536$ possible 4×4 blocks
One generation
Macro-cell evolution

- Compute central order-$(k-1)$ macrocell, $2^{k-2}$ steps in the future

[ Gosper 1984 ]
13 recursive calls

[Gosper 1984]
13 recursive calls

[Gosper 1984]
Macro-cell evolution

- Compute central order-$(k-1)$ macrocell, $2^{k-2}$ steps in the future

- 13 recursive calls on order-$(k-1)$ macrocells, either children or assembled from grandchildren
  - Naïvely: $T(k) = 13 \cdot T(k-1) + O(1) \Rightarrow T(k) = O(13^k) = O(n^{3.7005})$

- Use hashing (memoization) to avoid duplication
  - $\Rightarrow T(k) \leq O(4^k) = O(n^2)$
  - In practice, significantly faster
Thus, proliferations of cells is limited by indistinguishability when they are small, and by infrequency of creation when they are large. [Gosper 1984]
Gosper’s “stupid programming language”

- Store “pointers” to five order-\((k-1)\) macro-cells:
  - nw child
  - ne child
  - sw child
  - sw child
  - evolution result

- Use hashing to avoid duplication
Intuitively, the running time of level-\(k\) Hashlife is proportional to
the “amount of novelty” in the next \(2^{k-2}\) generations.

Can this intuition be formalized?

**Bottleneck:** Translations of the same pattern by a non-power-of-2
are distinct.

Can HashLife be improved to avoid this bottleneck?

Is HashLife “optimal” in any formal sense?
3. Conclusion

Even in simulating such an unpredictable and irreversible automaton as Life, considerable economies are possible. By attributing similar thriftiness to whatever implements our own reality, our (simulated) imaginations may be stimulated.