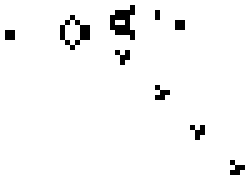
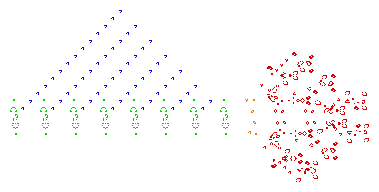
Conway's Game of Life



A single [Gosper](https://www.wikiwand.com/en/Bill_Gosper)'s [Glider Gun](https://www.wikiwand.com/en/Gun_(cellular_automaton)) creating "[gliders](https://www.wikiwand.com/en/Glider_(Conway%27s_Life))"



A screenshot of a [puffer-type breeder](https://www.wikiwand.com/en/Puffer_train_(cellular_automaton)) (red) that leaves [glider guns](https://www.wikiwand.com/en/Gun_(cellular_automaton)) (green) in its wake, which in turn create gliders (blue). ([animation](https://www.wikiwand.com/en/File:Conways_game_of_life_breeder_animation.gif))

The **Game of Life**, also known simply as **Life**, is a [cellular automaton](https://www.wikiwand.com/en/Cellular_automaton)devised by the British [mathematician](https://www.wikiwand.com/en/Mathematician) [John Horton Conway](https://www.wikiwand.com/en/John_Horton_Conway) in 1970.[[1]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote1)

The "game" is a [zero-player game](https://www.wikiwand.com/en/Zero-player_game), meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves, or, for advanced "players", by creating patterns with particular properties.

Rules

The universe of the Game of Life is an infinite two-dimensional[orthogonal](https://www.wikiwand.com/en/Orthogonal) grid of square *cells*, each of which is in one of two possible states, *alive* or *dead*, or "populated" or "unpopulated" (the difference may seem minor, except when viewing it as an early model of human/urban behavior simulation or how one views a blank space on a grid). Every cell interacts with its eight *[neighbours](https://www.wikiwand.com/en/Moore_neighborhood" \o "Moore neighborhood)*, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

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 over-population.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The initial pattern constitutes the *seed* of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed—births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a *tick* (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.

Origins

Conway was interested in a problem presented in the 1940s by mathematician [John von Neumann](https://www.wikiwand.com/en/John_von_Neumann), who attempted to find a hypothetical machine that could build copies of itself and succeeded when he found a mathematical model for such a machine with very complicated rules on a rectangular grid. The Game of Life emerged as Conway's successful attempt to drastically simplify von Neumann's ideas. The game made its first public appearance in the October 1970 issue of [*Scientific American*](https://www.wikiwand.com/en/Scientific_American), in [Martin Gardner](https://www.wikiwand.com/en/Martin_Gardner)'s "[Mathematical Games](https://www.wikiwand.com/en/Mathematical_Games_(column))" column. From a theoretical point of view, it is interesting because it has the power of a [universal Turing machine](https://www.wikiwand.com/en/Universal_Turing_machine): that is, anything that can be computed [algorithmically](https://www.wikiwand.com/en/Algorithm) can be computed within Conway's Game of Life.[[2]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenotechapman2)[[3]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenotebcg3) Gardner wrote:

*The game made Conway instantly famous, but it also opened up a whole new field of mathematical research, the field of*[*cellular automata*](https://www.wikiwand.com/en/Cellular_automata)*... Because of Life's analogies with the rise, fall and alterations of a society of living organisms, it belongs to a growing class of what are called "simulation games" (games that resemble real life processes).*

Ever since its publication, Conway's Game of Life has attracted much interest, because of the surprising ways in which the patterns can evolve. Life provides an example of [emergence](https://www.wikiwand.com/en/Emergence) and [self-organization](https://www.wikiwand.com/en/Self-organization). Scholars in various fields, such as [computer science](https://www.wikiwand.com/en/Computer_science), [physics](https://www.wikiwand.com/en/Physics), [biology](https://www.wikiwand.com/en/Biology), [biochemistry](https://www.wikiwand.com/en/Biochemistry), [economics](https://www.wikiwand.com/en/Economics), [mathematics](https://www.wikiwand.com/en/Mathematics), [philosophy](https://www.wikiwand.com/en/Philosophy), and[generative sciences](https://www.wikiwand.com/en/Generative_sciences) have made use of the way that complex patterns can emerge from the implementation of the game's simple rules[[*citation needed*](https://www.wikiwand.com/en/Wikipedia:Citation_needed)]. The game can also serve as a didactic [analogy](https://www.wikiwand.com/en/Analogy), used to convey the somewhat counter-intuitive notion that "design" and "organization" can spontaneously emerge in the absence of a designer. For example, philosopher and cognitive scientist [Daniel Dennett](https://www.wikiwand.com/en/Daniel_Dennett) has used the analogue of Conway's Life "universe" extensively to illustrate the possible evolution of complex philosophical constructs, such as[consciousness](https://www.wikiwand.com/en/Consciousness) and [free will](https://www.wikiwand.com/en/Free_will), from the relatively simple set of deterministic physical laws, which might govern our universe.[[4]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote4)[[5]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote5)[[6]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote6)

The popularity of Conway's Game of Life was helped by its coming into being just in time for a new generation of inexpensive computer access which were being released into the market. The game could be run for hours on these machines, which would otherwise have remained unused at night. In this respect, it foreshadowed the later popularity of computer-generated [fractals](https://www.wikiwand.com/en/Fractals). For many, Life was simply a programming challenge: a fun way to use otherwise wasted [CPU](https://www.wikiwand.com/en/Central_processing_unit) cycles. For some, however, Life had more philosophical connotations. It developed a cult following through the 1970s and beyond; current developments have gone so far as to create theoretic emulations of computer systems within the confines of a Life board.[[7]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote7)[[8]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote8)

Conway chose his rules carefully, after considerable experimentation, to meet these criteria:

1. There should be no explosive growth.
2. There should exist small initial patterns with chaotic, unpredictable outcomes.
3. There should be potential for [von Neumann universal constructors](https://www.wikiwand.com/en/Von_Neumann_universal_constructor).
4. The rules should be as simple as possible, whilst adhering to the above constraints.[[9]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote9)

Examples of patterns

The earliest interesting patterns in the Game of Life were discovered without the use of computers. The simplest static patterns ("[still lifes](https://www.wikiwand.com/en/Still_life_(CA))") and repeating patterns ("[oscillators](https://www.wikiwand.com/en/Oscillator_(CA))"—a superset of still lifes) were discovered while tracking the fates of various small starting configurations using graph paper, blackboards, physical game boards (such as [Go](https://www.wikiwand.com/en/Go_(board_game))) and the like. During this early research, Conway discovered that the R-[pentomino](https://www.wikiwand.com/en/Pentomino" \o "Pentomino) failed to stabilize in a small number of generations. In fact, it takes 1103 generations to stabilize, by which time it has a population of 116 and has fired six escaping gliders[[10]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote10) (these were the first gliders ever discovered).[[11]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote11)

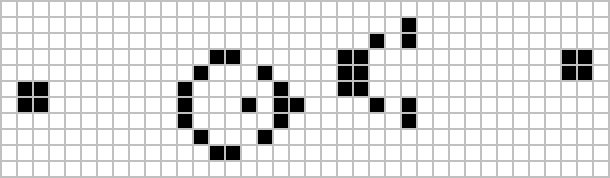
Many different types of patterns occur in the Game of Life, including still lifes, oscillators, and patterns that translate themselves across the board ("[spaceships](https://www.wikiwand.com/en/Spaceship_(CA))"). Some frequently occurring[[12]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote12) [[13]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote13) examples of these three classes are shown below, with live cells shown in black, and dead cells shown in white.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Still lifes*** | | | Block | https://upload.wikimedia.org/wikipedia/commons/thumb/9/96/Game_of_life_block_with_border.svg/132px-Game_of_life_block_with_border.svg.png | | Beehive | https://upload.wikimedia.org/wikipedia/commons/thumb/6/67/Game_of_life_beehive.svg/196px-Game_of_life_beehive.svg.png | | Loaf | https://upload.wikimedia.org/wikipedia/commons/thumb/f/f4/Game_of_life_loaf.svg/196px-Game_of_life_loaf.svg.png | | Boat | https://upload.wikimedia.org/wikipedia/commons/thumb/7/7f/Game_of_life_boat.svg/164px-Game_of_life_boat.svg.png | | |  |  | | --- | --- | | ***Oscillators*** | | | Blinker (period 2) | https://upload.wikimedia.org/wikipedia/commons/9/95/Game_of_life_blinker.gif | | Toad (period 2) | https://upload.wikimedia.org/wikipedia/commons/1/12/Game_of_life_toad.gif | | Beacon (period 2) | https://upload.wikimedia.org/wikipedia/commons/1/1c/Game_of_life_beacon.gif | | Pulsar (period 3) | https://upload.wikimedia.org/wikipedia/commons/0/07/Game_of_life_pulsar.gif | | Pentadecathlon (period 15) | https://upload.wikimedia.org/wikipedia/commons/f/fb/I-Column.gif | | |  |  | | --- | --- | | ***Spaceships*** | | | Glider | https://upload.wikimedia.org/wikipedia/commons/f/f2/Game_of_life_animated_glider.gif | | Lightweight spaceship (LWSS) | https://upload.wikimedia.org/wikipedia/commons/3/37/Game_of_life_animated_LWSS.gif | |

The "pulsar"[[14]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote14) is the most common period 3 oscillator. The great majority of naturally occurring oscillators are period 2, like the blinker and the toad, but oscillators of many periods are known to exist,[[15]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote15) and oscillators of periods 4, 8, 14, 15, 30 and a few others have been seen to arise from random initial conditions.[[16]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote16) Patterns called "[Methuselahs](https://www.wikiwand.com/en/Methuselah_(cellular_automata))" can evolve for long periods before stabilizing, the first-discovered of which was the R-pentomino. "Diehard" is a pattern that eventually disappears (rather than merely stabilizing) after 130 generations, which is conjectured to be maximal for patterns with seven or fewer cells.[[17]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote17) "Acorn" takes 5206 generations to generate 633 cells including 13 escaped gliders.[[18]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote18)

|  |  |  |
| --- | --- | --- |
| The R-pentomino  The R-pentomino | Diehard  Diehard | Acorn  Acorn |

Conway originally conjectured that no pattern can grow indefinitely—i.e., that for any initial configuration with a finite number of living cells, the population cannot grow beyond some finite upper limit. In the game's original appearance in "[Mathematical Games](https://www.wikiwand.com/en/Mathematical_Games_(column))", Conway offered a $50 prize to the first person who could prove or disprove the conjecture before the end of 1970. The prize was won in November of the same year by a team from the[Massachusetts Institute of Technology](https://www.wikiwand.com/en/Massachusetts_Institute_of_Technology), led by [Bill Gosper](https://www.wikiwand.com/en/Bill_Gosper); the "Gosper glider gun" produces its first glider on the 15th generation, and another glider every 30th generation from then on. For many years this glider gun was the smallest one known.[[19]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote19) In 2015 a period-120 gun was discovered that has fewer live cells but a larger bounding box.[[20]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote20)



Gosper glider gun

Smaller patterns were later found that also exhibit infinite growth. All three of the following patterns grow indefinitely: the first two create one "block-laying switch engine"[[21]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote21) each, while the third creates two. The first has only 10 live cells (which has been proven to be minimal).[[22]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote22) The second fits in a 5 × 5 square. The third is only one cell high:

|  |
| --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/7/72/Game_of_life_infinite1.svg/324px-Game_of_life_infinite1.svg.png    https://upload.wikimedia.org/wikipedia/commons/thumb/a/ae/Game_of_life_infinite2.svg/228px-Game_of_life_infinite2.svg.png |
| https://upload.wikimedia.org/wikipedia/commons/thumb/9/95/Game_of_life_infinite3.svg/1316px-Game_of_life_infinite3.svg.png |

Later discoveries included other "[guns](https://www.wikiwand.com/en/Gun_(CA))", which are stationary, and which shoot out gliders or other spaceships; "[puffers](https://www.wikiwand.com/en/Puffer_train_(CA))", which move along leaving behind a trail of debris; and "[rakes](https://www.wikiwand.com/en/Rake_(cellular_automaton))", which move and emit spaceships.[[23]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote23) Gosper also constructed the first pattern with an [asymptotically optimal](https://www.wikiwand.com/en/Asymptotically_optimal) [quadratic growth rate](https://www.wikiwand.com/en/Quadratic_growth), called a "[breeder](https://www.wikiwand.com/en/Breeder_(CA))", or "lobster", which worked by leaving behind a trail of guns.

It is possible for gliders to interact with other objects in interesting ways. For example, if two gliders are shot at a block in just the right way, the block will move closer to the source of the gliders. If three gliders are shot in just the right way, the block will move farther away. This "sliding block memory" can be used to simulate a [counter](https://www.wikiwand.com/en/Counter_(digital)). It is possible to construct [logic gates](https://www.wikiwand.com/en/Logic_gate) such as [*AND*](https://www.wikiwand.com/en/Logical_conjunction), [*OR*](https://www.wikiwand.com/en/Logical_disjunction) and [*NOT*](https://www.wikiwand.com/en/Negation) using gliders. It is possible to build a pattern that acts like a [finite state machine](https://www.wikiwand.com/en/Finite_state_machine) connected to two counters. This has the same computational power as a [universal Turing machine](https://www.wikiwand.com/en/Universal_Turing_machine), so the Game of Life is theoretically as powerful as any computer with unlimited memory and no time constraints: it is [Turing complete](https://www.wikiwand.com/en/Turing_complete).[[2]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenotechapman2)[[3]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenotebcg3)

Furthermore, a pattern can contain a collection of guns that fire gliders in such a way as to construct new objects, including copies of the original pattern. A "universal constructor" can be built which contains a Turing complete computer, and which can build many types of complex objects, including more copies of itself.[[3]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenotebcg3)

Undecidability

Many patterns in the game of life eventually become a combination of still lives, oscillators and spaceships; other patterns may be called chaotic. A pattern may stay chaotic for a very long time until it eventually settles to such a combination.

It can be asked whether the game of life is [decidable](https://www.wikiwand.com/en/Undecidable_problem): whether an algorithm exists, so that given an "initial" pattern and a "later" pattern, the algorithm can tell whether, starting with the initial pattern, the later pattern is ever going to appear. This turns out to be impossible: no such algorithm exists. This is in fact a corollary of the [halting problem](https://www.wikiwand.com/en/Halting_problem).[[24]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote24)

Indeed, since the game of life includes a pattern that is equivalent to a UTM (universal Turing machine), this "deciding" algorithm, if it existed, could have been used to solve the halting problem, by taking the initial pattern as the one corresponding to a UTM+input and the later pattern as the one corresponding to a halting state of the machine with an empty tape (as one can modify the Turing machine to always erase the tape before halting). However the halting problem is provably [undecidable](https://www.wikiwand.com/en/Undecidable_problem) and so such an algorithm does not exist.

It also follows that some patterns exist that remain chaotic forever: otherwise one could just progress the game of life sequentially until a non-chaotic pattern emerges, and then easily compute whether the later pattern is going to appear.

Self-replication

On May 18, 2010, Andrew J. Wade announced a self-constructing pattern dubbed Gemini which creates a copy of itself while destroying its parent.[[25]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote25)[[26]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote26) This pattern replicates in 34 million generations, and uses an instruction tape made of gliders which oscillate between two stable configurations made of Chapman-Greene construction arms. These, in turn, create new copies of the pattern, and destroy the previous copy. Gemini is also a spaceship, and is in fact the first spaceship constructed in the Game of Life which is neither orthogonal nor purely diagonal (these are called knightships).[[27]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote27)[[28]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote28)

On November 23, 2013, Dave Greene built the first replicator in Conway's Game of Life that creates a complete copy of itself, including the instruction tape.[[29]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote29)

In December 2015, diagonal versions of the Gemini were built.[[30]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote30)

Iteration

From any random initial pattern of living cells on the grid, observers will find the population constantly changing as the generations tick by. The patterns that emerge from the simple rules may be considered a form of [beauty](https://www.wikiwand.com/en/Mathematical_beauty). Small isolated subpatterns with no initial symmetry tend to become symmetrical. Once this happens, the symmetry may increase in richness, but it cannot be lost unless a nearby subpattern comes close enough to disturb it. In a very few cases the society eventually dies out, with all living cells vanishing, though this may not happen for a great many generations. Most initial patterns eventually "burn out", producing either stable figures or patterns that oscillate forever between two or more states;[[31]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote31)[[32]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote32) many also produce one or more gliders or spaceships that travel indefinitely away from the initial location. Because of the nearest-neighbor based rules, no "information" can travel through the grid at a greater rate than one cell per unit time, so this velocity is said to be the [cellular automaton speed of light](https://www.wikiwand.com/en/Speed_of_light_(cellular_automaton)) and denoted .

Algorithms

Early patterns with unknown futures, such as the R-pentomino, led computer programmers across the world to write programs to track the evolution of Life patterns. Most of the early [algorithms](https://www.wikiwand.com/en/Algorithm) were similar; they represented Life patterns as two-dimensional arrays in computer memory. Typically two arrays are used, one to hold the current generation, and one in which to calculate its successor. Often 0 and 1 represent dead and live cells respectively. A nested for-loop considers each element of the current array in turn, counting the live neighbours of each cell to decide whether the corresponding element of the successor array should be 0 or 1. The successor array is displayed. For the next iteration the arrays swap roles so that the successor array in the last iteration becomes the current array in the next iteration.

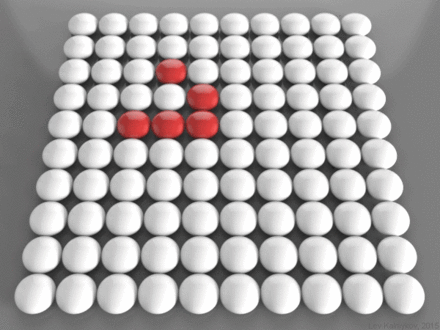
A variety of minor enhancements to this basic scheme are possible, and there are many ways to save unnecessary computation. A cell that did not change at the last time step, and none of whose neighbours changed, is guaranteed not to change at the current time step as well. So, a program that keeps track of which areas are active can save time by not updating the inactive zones.[[33]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote33)

To avoid decisions and branches in the counting loop, the rules can be rearranged from an egocentric approach of the inner field regarding its neighbors to a scientific observer's viewpoint: if the sum of all nine fields is 3, the inner field state for the next generation will be life (no matter of its previous contents); if the all-field sum is 4, the inner field retains its current state and every other sum sets the inner field to death.

If it is desired to save memory, the storage can be reduced to one array plus 3 line buffers. One line buffer is used to calculate the successor state for a line, then the second line buffer is used to calculate the successor state for the next line. The first buffer is then written to its line and freed to hold the successor state for the third line. If a[toroidal](https://www.wikiwand.com/en/Torus) array is used, a third buffer is needed so that the original state of the first line in the array can be saved until the last line is computed.

Glider gun within a toroidal array. The stream of gliders eventually wraps round and destroys the gun.

Glider gun within a toroidal array. The stream of gliders eventually wraps round and destroys the gun.



Red glider on the square lattice with periodic boundary conditions.

In principle, the Life field is infinite, but computers have finite memory. This leads to problems when the active area encroaches on the border of the array. Programmers have used several strategies to address these problems. The simplest strategy is simply to assume that every cell outside the array is dead. This is easy to program, but leads to inaccurate results when the active area crosses the boundary. A more sophisticated trick is to consider the left and right edges of the field to be stitched together, and the top and bottom edges also, yielding a [toroidal](https://www.wikiwand.com/en/Torus) array. The result is that active areas that move across a field edge reappear at the opposite edge. Inaccuracy can still result if the pattern grows too large, but at least there are no pathological edge effects. Techniques of dynamic storage allocation may also be used, creating ever-larger arrays to hold growing patterns.

Alternatively, the programmer may abandon the notion of representing the Life field with a 2-dimensional array, and use a different data structure, like a vector of coordinate pairs representing live cells. This approach allows the pattern to move about the field unhindered, as long as the population does not exceed the size of the live-coordinate array. The drawback is that counting live neighbours becomes a hash-table lookup or search operation, slowing down simulation speed. With more sophisticated data structures this problem can also be largely solved.

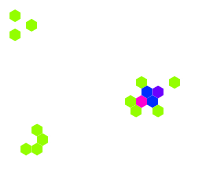
For exploring large patterns at great time-depths, sophisticated algorithms such as [Hashlife](https://www.wikiwand.com/en/Hashlife" \o "Hashlife) may be useful. There is also a method, applicable to other cellular automata too, for implementation of the Game of Life using arbitrary asynchronous updates whilst still exactly emulating the behaviour of the synchronous game.[[34]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote34)

Source code examples that implements the basic Game of Life scenario in various programming languages, including C, C++, Java and Python can be found here[[35]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote35)

Variations on Life

|  |  |
| --- | --- |
|  | Main article: [Life-like cellular automaton](https://www.wikiwand.com/en/Life-like_cellular_automaton) |

Since Life's inception, new similar cellular automata have been developed. The standard Game of Life is symbolised as "B3/S23": A cell is "**B**orn" if it has exactly 3 neighbours, "**S**urvives" if it has 2 or 3 living neighbours; it dies otherwise. The first number, or list of numbers, is what is required for a dead cell to be born. The second set is the requirement for a live cell to survive to the next generation. Hence "B6/S16" means "a cell is born if there are 6 neighbours, and lives on if there are either 1 or 6 neighbours". Cellular automata on a two-dimensional grid that can be described in this way are known as [Life-like cellular automata](https://www.wikiwand.com/en/Life-like_cellular_automata). Another common Life-like automaton,[*Highlife*](https://www.wikiwand.com/en/Highlife_(cellular_automaton)), is described by the rule B36/S23, because having 6 neighbours, in addition to the original game's B3/S23 rule, causes a birth. HighLife is best known for its frequently occurring replicators.[[36]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote36)[[37]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote37) Additional Life-like cellular automata exist, although the vast majority of them produce universes that are either too chaotic or too desolate to be of interest.



A sample of a 48-step oscillator along with a 2-step oscillator and a 4-step oscillator from a 2-D hexagonal Game of Life (rule H:B2/S34)

Some variations on Life modify the geometry of the universe as well as the rule. The above variations can be thought of as 2-D square, because the world is two-dimensional and laid out in a square grid. 1-D square variations (known as [elementary cellular automata](https://www.wikiwand.com/en/Elementary_cellular_automaton))[[38]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote38) and 3-D square variations have been developed, as have 2-D hexagonal and 2-D triangular variations. A variant using non-periodic tile grids has also been made.[[39]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote39)

Conway's rules may also be generalized such that instead of two states (*live* and *dead*) there are three or more. State transitions are then determined either by a weighting system or by a table specifying separate transition rules for each state; for example, [Mirek's Cellebration](https://www.wikiwand.com/en/Mirek%27s_Cellebration" \o "Mirek's Cellebration)'s multi-coloured "Rules Table" and "Weighted Life" rule families each include sample rules equivalent to Conway's Life.

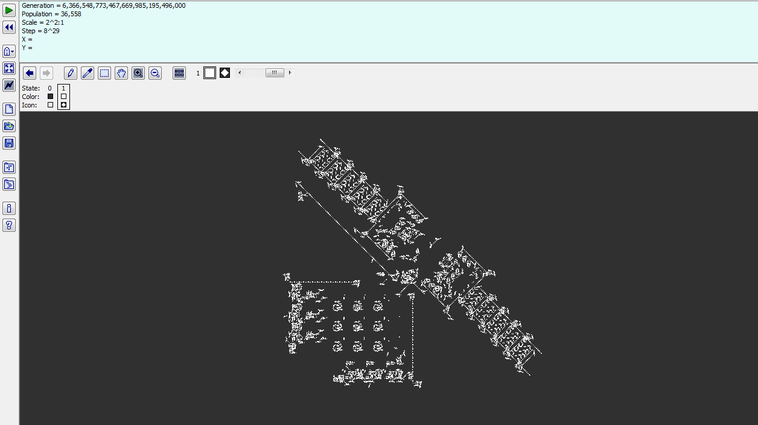
Patterns relating to fractals and fractal systems may also be observed in certain Life-like variations. For example, the automaton B1/S12 generates four very close approximations to the[Sierpiński triangle](https://www.wikiwand.com/en/Sierpi%C5%84ski_triangle) when applied to a single live cell. The Sierpiński triangle can also be observed in Conway's Game of Life by examining the long-term growth of a long single-cell-thick line of live cells,[[40]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote40) as well as in[Highlife](https://www.wikiwand.com/en/Highlife_(cellular_automaton)), [Seeds (B2/S)](https://www.wikiwand.com/en/Seeds_(cellular_automaton)), and Wolfram's [Rule 90](https://www.wikiwand.com/en/Rule_90).[[41]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote41)

*Immigration* is a variation that is very similar to Conway's Game of Life, except that there are two ON states (often expressed as two different colours). Whenever a new cell is born, it takes on the ON state that is the majority in the three cells that gave it birth. This feature can be used to examine interactions between [spaceships](https://www.wikiwand.com/en/Spaceship_(CA)) and other "objects" within the game.[[42]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote42) Another similar variation, called *QuadLife*, involves four different ON states. When a new cell is born from three different ON neighbours, it takes on the fourth value, and otherwise, like Immigration, it takes the majority value.[[43]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote43) Except for the variation among ON cells, both of these variations act identically to Life.

Music

Various musical composition techniques use Conway's Life, especially in [MIDI](https://www.wikiwand.com/en/MIDI) sequencing.[[44]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote44) A variety of programs exist for creating sound from patterns generated in Life (see footnotes for links to examples).[[45]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote45)[[46]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote46)[[47]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote47)

Notable Life programs



The 6,366,548,773,467,669,985,195,496,000th (6 [octillionth](https://www.wikiwand.com/en/Octillion)) generation of a [Turing machine](https://www.wikiwand.com/en/Turing_machine), made in the game of Life, computed in less than 30 seconds on an [Intel](https://www.wikiwand.com/en/Intel" \o "Intel)Core Duo 2 GHz CPU using Golly in [Hashlife](https://www.wikiwand.com/en/Hashlife" \o "Hashlife)mode

Computers have been used to follow Life configurations from the earliest days. When John Conway was first investigating how various starting configurations developed, he tracked them by hand using a [Go](https://www.wikiwand.com/en/Go_(game)) board with its black and white stones. This was tedious and prone to errors. The first interactive Life program was written in [ALGOL 68](https://www.wikiwand.com/en/ALGOL_68) for the [PDP-7](https://www.wikiwand.com/en/PDP-7)by [M. J. T. Guy](https://www.wikiwand.com/en/Michael_Guy_(computer_scientist)) and [S. R. Bourne](https://www.wikiwand.com/en/Stephen_R._Bourne). The results were published in the October 1970 issue of [*Scientific American*](https://www.wikiwand.com/en/Scientific_American)[[48]](https://www.wikiwand.com/en/Conway%27s_Game_of_Life#citenote48) and—regarding the use of the program—reports "Without its help, some discoveries about the game would have been difficult to make."

There are now thousands of Life programs online, so a full list will not be provided here. The following is a small selection of programs with some special claim to notability, such as popularity or unusual features. Most of these programs incorporate a graphical user interface for pattern editing and simulation, the capability for simulating multiple rules including Life, and a large library of interesting patterns in Life and other CA rules.

* [Golly](https://www.wikiwand.com/en/Golly_(program)). A cross-platform (Windows, Macintosh, Linux and also iOS and Android) open-source simulation system for Life and other cellular automata, by Andrew Trevorrow and Tomas Rokicki. It includes the [hashlife](https://www.wikiwand.com/en/Hashlife" \o "Hashlife)algorithm for extremely fast generation, and [Perl](https://www.wikiwand.com/en/Perl) or [Python](https://www.wikiwand.com/en/Python_(programming_language)) scriptability for both editing and simulation.
* [Mirek's Cellebration](https://www.wikiwand.com/en/Mirek%27s_Cellebration). Free 1-D and 2-D cellular automata viewer, explorer and editor for Windows. Includes powerful facilities for simulating and viewing a wide variety of CA rules including Life, and a scriptable editor.
* [Xlife](http://www.conwaylife.com/wiki/Xlife). A cellular-automaton laboratory by Jon Bennett. The standard UNIX X11 Life simulation application for a long time, it has also been ported to Windows. Can handle cellular automaton rules with the same neighbourhood as Life, and up to eight possible states per cell.

Google implemented an [easter egg](https://www.wikiwand.com/en/Easter_egg_(media)" \o "Easter egg (media)) of Conway's Game of Life in 2012. Users who search for the term are shown an implementation of the game in the search results page.

Iteration

Much the same as iteration in any programming language, thereâ€™s a small leap to be made to understand the basic principle of interation in a spreadsheet. An iterative spreadsheet is one which no longer simply produces the answer given a set of inputs – it is an evolving process. Each time you calculate the spreadsheet, it will produce a new answer based upon the current data it contains. Iteration has been in spreadsheets for a long time â€“ it was in Lotus 1-2-3 and was a feature in Excel’s predecessor, Multiplan. To use iteration and retain all of your hair you need to have a much finer control over when Excel calculates, so letâ€™s go ahead and switch into manual calculation mode by clicking the Office Button, then “Excel Options” and in the “Formulas” section under “Workbook Calculation”, check the â€œManualâ€ flag. And while weâ€™re here, letâ€™s switch on iteration by checking â€œEnable ierative calculationâ€, and setting â€œMaximum Iterationsâ€ to 1.

Now, into cell A1 enter the formula =A1+1.

This, of course, is a circular reference, but thereâ€™s no warning message. Excel turns the message off because circular references are the bread and butter of an iterative spreadsheet – each time we calculate it we want to base the results on the previous values, so circular references are a necessity. The result of that formula should show up as 1, because A1 was empty before the formula was entered, and empty cells evaluate to zero. Press the F9 key to recalculate the spreadsheet again â€“ A1 will become 2, and so on. If you went back into the options and turned the number of iterations up, the workook would be iterating several times per recalc and the value of A1 would jump in larger steps.

The next problem is how to restart the counter â€“ this is most easily done with a â€œresetâ€ flag somewhere on the sheet. So into A2 enter TRUE, hit Ctrl-F3 to bring up the Name Manager and give that cell the name â€œresetâ€. Change A1 to read =IF(reset,0,A1+1). Recalculate the spreadsheet once, and A1 will become zero. Type FALSE into A2 and recalculate â€“ now the count has started again.

The Game of Life

Conwayâ€™s Game of Life is a simulation involving very simple â€œlife formsâ€ on an infinite grid. There is an excellent Wikipedia article about it with some history, a detailed explanation and some examples. Essentially, each cell in the grid is determined to be either be alive or dead based on a particular set of criteria related to its immediate neighbours â€“ these being the three cells above it, the ones to the left and right and the three below.

The criteria for life or death are:

A live cell with zero or one neighbours will die from loneliness

A live cell with more than three neighbours will die from overcrowding

A live cell with two or three neighbours will remain alive

A dead cell with three live neighbours will spring into life

Itâ€™s quite possible to simulate this game using a pen and some paper, but itâ€™s a little time-consuming.

Implementation

Weâ€™re going to create an iterative spreadsheet with each cell representing one, erm, cell in Conwayâ€™s Game of Life. Inside each will be a formula which determines whether it is alive or dead at the end of the iteration, and each time we calculate the spreadsheet weâ€™ll perform one more iteration in the game.

To keep our â€œalive or deadâ€ formula simple, letâ€™s assume that the named range â€œnborsâ€ represents the number of neighbours a cell has. And letâ€™s assume that TRUE and FALSE are used to represent the living state of each cell. Based on the above table, we now have the logic:

Cell State Number of Neighbours Resulting State

TRUE 1 or fewer FALSE

TRUE 2 TRUE

TRUE 3 TRUE

TRUE 4 or more FALSE

FALSE 3 TRUE

FALSE anything except 3 FALSE

Letâ€™s deal first with what to do in the case where the current cell is alive. Excel has the CHOOSE function, which will pick the nth item from an array â€“ so =CHOOSE(1,â€pigâ€,â€dogâ€) returns â€œpigâ€, and =CHOOSE(2,â€pigâ€,â€dogâ€) returns â€œdogâ€. If the cell is alive, we can use CHOOSE to pick its resulting state using the number of neighbours as an index. Each cell has a minimum of zero and a maximum eight possible neighbours, so our new cell value could look something like:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| =CHOOSE(nbors+1, | FALSE, | FALSE, | TRUE, | TRUE, | FALSE, | FALSE, | FALSE, | FALSE, | FALSE, | FALSE) |
| the CHOOSE function starts its indexing at 1, but the lowest number of nbors is 0 | Value if 0 | Value if 1 | Value if 2 | Value if 3 | Value if 4 | Value if 5 | Value if 6 | Value if 7 | Value if 8 | Value if 9 |

Thereâ€™s a bit of wastage in there because the CHOOSE function doesnâ€™t have any ability to understand â€œ4 or moreâ€. To get around this, we can use the MIN function to cap the number if it was 4 or more. So we can shorten the above formula to the following (remember weâ€™re using nbors+1):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **=CHOOSE(** | **MIN(5,nbors+1),** | **FALSE,** | **FALSE,** | **TRUE,** | **TRUE,** | **FALSE))** |
|  | The number of neighbours, capped at 4 and increased by 1 for the CHOOSE function | Value if 0 | Value if 1 | Value if 2 | Value if 3 | Value if 4 or more |

So thatâ€™s the case dealt with where the cell is alive. If itâ€™s dead, we need to bring it alive if it has exactly three neighbours. All the formulas here will relate to the spreadsheet cell B2 â€“ the reason being that the very top-left cell becomes a bit more complex when trying to count its neighbours, which is something Iâ€™ll deal with later. We can use a circular reference to check the current state â€“ if the cell is alive weâ€™ll use the CHOOSE() formula above to set our state; if itâ€™s dead itâ€™ll become alive if nbors is exactly 3. We could enter:

=IF(B2,CHOOSE(MIN(5,nbors+1),FALSE,FALSE,TRUE,TRUE,FALSE),IF(nbors=3,TRUE,FALSE))

In actual fact the formula â€œnbors=3â€ is a statement which itself evaluates to TRUE or FALSE. Whenever you want the results of an IF() to be TRUE or FALSE, you can just write the condition in instead of the whole IF(x,TRUE,FALSE) thing, so:

=IF(B2,CHOOSE(MIN(5,nbors+1),FALSE,FALSE,TRUE,TRUE,FALSE),nbors=3)

Itâ€™s mostly for this reason that TRUE and FALSE are better things to use for spreadsheets like this than, say, 1 and 0, or â€œxâ€ and â€œâ€.

So we now have a formula for one iteration of the game, but weâ€™ve no real way to tell the simulator which values to actually use to start off with. So let’s create a new worksheet called â€œtemplateâ€, and type some values to start the process. The sample object here is whatâ€™s called a â€œgliderâ€ â€“ itâ€™s a cyclical formation which will move forever.

Letâ€™s add a reset cell to our spreadsheet using Ctrl-F3 as before. We can now extend our cell formula in B2 on the main grid sheet to read the starter values from the template if the reset flag is set:

=IF(Reset,template!B2,IF(B2,CHOOSE(MIN(5,nbors+1),FALSE,FALSE,TRUE,TRUE,FALSE),nbors=3))

We now have the complete formula for one cell in our game board. Right now itâ€™s #NAME? though, because we donâ€™t have â€œnborsâ€ defined. As I mentioned earlier, a cellâ€™s neighbours are all the cells which border it. For B2 this would be A1, B1, C1 above, then A2 and C2 to the sides, and then A3, B3 and C3 below. If you use TRUE and FALSE as if they were numbers they will evaluate to 1 and 0, so adding TRUE/FALSE cells together using â€œ+â€ will return the number of TRUEs in the given cells. So a tempting way to define â€œnborsâ€ would be to select cell B2, then create new name which evaluated to:

A1+B1+C1+A2+C2+A3+B3+C3

Note the lack of $ signsâ€“ this name contains relative references, and will point to a different range when a different cell is selected. If you choose cell C3 and then bring up the name manager, youâ€™ll see:

B2+C2+D2+B3+D3+B4+C4+D4

Anyway â€“ at any particular point in the grid, this will return how many neighbours the cell has. Thereâ€™s a problem, though. The Game of Life is played on a board which flips state in one single move, and unfortunately what weâ€™ve made is a board which will change as we iterate through the cells establishing their new values. The way the game is supposed to work, no changes are made as you run through the cells â€“ you just work out what the new values will be and then set them all at once at the end of the pass. Fiddlesticks.

We need to find some way of saying to Excel that the number of neighbours the cell has should come from the board as it stood at the beginning of the iteration, not from the board as it stands now with us in the middle of modifying it. To do this we can take advantage of one of the powerful foibles of iterative calculation mode â€“ in iterative mode, Excel will calculate worksheets one by one in alphabetical order.

So – to get around this problem, we can create a new sheet containing the number of neighbours for each cell and we know that this sheet wonâ€™t be calculated until the whole of the main grid has finished. Letâ€™s go ahead and rename our main sheet to â€œ1.runâ€ and create a new â€œ2.nborsâ€ worksheet. Iâ€™ve used the numeric prefixes to remind myself that these sheets need to be calculated in that order. In actual fact in this instance it doesnâ€™t matter in which order theyâ€™re calculated, but for other iterative sheets it might well, and I find it makes debugging easier.

Into B2 on the new â€œ2.nborsâ€ sheet we can now enter a formula to count the number of neighbours that â€˜1.runâ€™!B2 has. B2 is going to be the top left of our board and as such some of the surrounding cells are going to be empty â€“ letâ€™s ignore this for the moment and fix it later. The formula I was intending using above now becomes:

=â€™1.runâ€™!A1+â€™1.runâ€™!B1+â€™1.runâ€™!C1+â€™1.runâ€™!A2+â€™1.runâ€™!C2+â€™1.runâ€™!A3+â€™1.runâ€™!B3+â€™1.runâ€™!C3

This isnâ€™t the prettiest formula. Because iterative calc allows circular references, we can simplify it by just including B2 itself, and taking account of that in our original single-cell formula. So we could change that formula to:

=SUM(â€˜1.runâ€™!A1:C3)

Well, no. Unfortunately SUM doesnâ€™t treat boolean TRUE/FALSE values quite the same as â€œ+â€ does â€“ SUM only sums proper numeric values, and so we get zero from summing TRUEs. A handy trick here is to use Excelâ€™s SIGN() function. This returns 1 if a number is positive and 0 if itâ€™s zero â€“ it is quite happy taking booleans, so =SIGN(TRUE) is 1 and =SIGN(FALSE) is 0. We can array-enter this to make it act upon a range of booleans, and then SUM the results. If youâ€™re new to array formulas I donâ€™t have room to go into too much detail here â€“ esentially they are formulas which can act upon an array of cells sequentially, and also return more than one result. If you search for â€œarray formulasâ€ in Excelâ€™s help there are some good examples.

Still working on cell â€˜2.nborsâ€™!B2 we can array-enter (Ctrl-Shift-Enter) the following formula to sum all of the TRUE values in A1:C3 on the 1.run sheet:

{=SUM(SIGN(â€˜1.runâ€™!A1:C3))}

A handy tip (especially handy for array formulas) is to use F9 in the formula bar to evaluate parts of your formula â€“ if you highlight just the SIGN() portion of that formula:

And hit F9:

It shows the results of the SIGN() function. If you then highlight the SUM segment:

And F9 again:

You can see it evaluated the SUM portion too. Hit escape now to cancel editing the formula.

Now that â€œnborsâ€ isnâ€™t exactly the number of neighbours any more (remember weâ€™re including the cell itself now) weâ€™ll have to go back to our original formula and fix it up. Thereâ€™s no change needed if the cell is dead, but now if the cell is alive the number of neighbours is going to be greater by 1. We can change our original cell formula (on â€˜1.runâ€™!B2) to stop adding 1 to nbors in the â€œif aliveâ€ case:

=IF(Reset,template!B2,IF(B2,CHOOSE(MIN(5,nbors),FALSE,FALSE,TRUE,TRUE, FALSE),nbors=3))

We should now be able to populate some more cells and run the sheet. Extend the formulas on 1.run and 2.nbors right and downwards to make them into a bigger board, change the reset flag to TRUE and hit F9 to reset the sheet to the starting values. Let’s make the live cells a bit clearer to spot – bring up the conditional formatting dialog (Home..Styles..Conditional Formatting..New Rule..Format only cells that contain) and format B2 to have a dark background if its value is TRUE. You can now copy/paste this format onto the rest of the board.

Now change the reset flag to FALSE and hit F9 a few times â€“ successive recalculates will show the iteration sequence in the game.

Thereâ€™s only one problem left â€“ the Game of Life is supposed to be played on an infinite board, but what weâ€™ve created is one with edges â€“ any shape which slips off the edge of the board will vanish forever, whereas what we really ought to do is have it come on the other side. We can effect this by modifying the 2.nbors sheet â€“ the way I found to do this isnâ€™t overly pretty so Iâ€™d appreciate other suggestions. On the 2.nbors sheet, B2 is at the top left so the cells above and to the left of it arenâ€™t actually on the board at all. We really want the cells above it to come from the bottom row and the cells to the left of it to come from the rightmost column. We can do this by simply switching back to using â€œ+â€ for the corner cells â€“ in my somewhat arbitrary 40×35 board, B2 becomes:

=’1.run’!AO36+’1.run’!B36+’1.run’!C36+’1.run’!AO2+’1.run’!B2+’1.run’!C2+’1.run’!AO3+’1.run’!B3+’1.run’!C3

I know, I know. Itâ€™s not pretty. Perhaps something nicer could be made with INDEX() and ROW()/COLUMN(). We need to do a similar wraparound at the other corners and a less complicated one just using two SUM(SIGN())s for the top, left, right and bottom rows. Now when 1.run!B2 needs to know how many neighbours it has, weâ€™ll accurately be taking into account the ones from the opposite edge of our â€œinfiniteâ€ board.

Now that there are more rows and columns in Excel 2007, we can run a bigger simulation â€“ the one below is 500×500. The row and column headers look a little odd as Iâ€™ve had to make the rows heights and column widths rather small.

And there we have it. There are a few websites with interesting Game of Life models on them â€“ The Internet Encyclopedia of Science has some good simple ones, and there are more on the Wikipedia page. Here is the workbook I created during this post â€“ itâ€™s in XLS format and will work fine in Excel versions 97 and above. If you extend this at all, I’d be interested to see the results.