ChatGPT

Top 20 Artificial Life and Evolutionary Simulations

1. Conway's Game of Life

The **Game of Life** (1970) is a classic two-dimensional cellular automaton devised by John H. Conway [1]. It consists of an infinite grid of cells that are either "alive" or "dead," evolving in discrete time steps by simple rules of birth, survival, and death [2] [3]. From trivial initial conditions, Life produces surprisingly complex patterns, providing a striking example of emergence and self-organization [4]. It is Turing-complete and has yielded many known patterns (oscillators, gliders, etc.), captivating programmers and researchers for decades [5] [6]. The Game of Life is easy to implement (just loop over a grid applying the rules); numerous open-source versions exist – for example, Rosetta Code hosts implementations in C, C++, Java, Python and more [7].

2. Langton's Ant

Langton's Ant (invented by Chris Langton in 1986) is a simple 2D cellular automaton/Turing machine that exhibits chaotic behavior and order [8]. The "ant" moves on an infinite checkerboard of black and white cells, turning left on black or right on white, and flipping the color of each cell it visits [9]. These extremely simple rules produce "fantastically rich and complex" patterns: initially the ant's path is chaotic, but after about 10,000 steps it suddenly begins building a diagonal highway that extends indefinitely [10]. Remarkably, no matter the starting configuration, the ant apparently always builds this everlasting highway (though proving this is an open problem) [11]. Langton's Ant is often cited as a demonstration of how simple rules can lead to undecidability and complex emergent behavior [12] [13].

3. Boids (Flocking Simulation)

Boids is an artificial life program developed by Craig Reynolds (1986) to simulate flocking behavior of birds (and schoolings of fish) [14]. Each boid (agent) follows a few simple rules that govern its movement: **separation** (avoid crowding neighbors), **alignment** (steer towards the average heading of neighbors), and **cohesion** (steer towards the average position of neighbors) [15]. These local rules produce realistic collective motion – flocks that swirl, split, and reform – illustrating emergent behavior arising from agent interactions [15] [16]. The Boids model has been widely used in computer graphics and games to animate lifelike swarms of creatures. It inspired many variations (adding obstacle avoidance, leaders, etc.) and is a cornerstone example of **swarm intelligence** and decentralized animation techniques [17]. Boids can be implemented in any language by updating each agent's velocity based on neighbors; numerous open-source demos exist.

4. Ant Colony Optimization (Foraging Ant Simulation)

Ant Colony Optimization (ACO), introduced by Marco Dorigo in 1992, is a family of algorithms inspired by the foraging behavior of real ants [18]. In nature, ants find shortest paths to food by laying pheromone trails – shorter paths get reinforced faster, leading to a collective convergence on efficient routes [19] [20]. ACO simulates this by having a colony of artificial "ants" (agents) traverse a graph representing the problem (e.g.

cities in a TSP), depositing virtual pheromone on paths. Paths that lead to better solutions get stronger pheromone and attract more ants in subsequent iterations, a positive feedback loop that emergently yields near-optimal solutions [20] [21]. ACO was originally applied to routing problems and has since been extended widely. Visualizing ACO often shows many agents exploring paths and gradually funneling onto the best route as pheromone intensifies. It's a key example of **swarm intelligence** optimization guided by simple local behaviors.

5. Dawkins' Biomorphs (Interactive Evolution)

Biomorphs are an early demonstration of evolutionary algorithms applied to imagery, described by Richard Dawkins in *The Blind Watchmaker* (1986). Dawkins' program evolved simple stick-figure "creatures" defined by 9 genetic parameters (genes) for features like branching and symmetry [22]. The evolution was driven by **artificial selection**: the user would pick a creature they found interesting or aesthetically pleasing, and the program would generate a new generation of variants (via random mutations of the genes) based on that choice [23] [24]. In this way, complex tree-like forms "evolved" over successive iterations, even though the computer had no explicit fitness function – only the user's preference ("un-natural selection") guided it [22]. Dawkins was astonished by the variety of delicate, animal-like forms that emerged from such a simple evolutionary algorithm [25]. The **lesson** of the biomorphs is that large complexity and diversity can evolve from a few simple rules and mutations, illustrating the power of cumulative selection [26] [27]. Modern interactive evolutionary art programs (e.g. evolving abstract images or shapes) are direct descendants of this idea.

6. Karl Sims' Evolving Virtual Creatures

In 1994, Karl Sims demonstrated a groundbreaking simulation of evolving virtual creatures in a physics environment [28]. In his system, each creature is an assemblage of connected 3D blocks with virtual muscles; their "genome" encodes the morphology and control network. A population of hundreds of creatures is seeded and then subjected to simulated Darwinian evolution to achieve certain tasks (like swimming, walking, jumping, or fighting for a resource) [29] [30]. Creatures compete in the task, and the most successful ones are selected to reproduce – their genomes are combined and mutated to produce offspring for the next generation [31]. Over many generations, complex locomotion strategies emerge: Sims' famous video shows blocky creatures that learned to **swim** elegantly, **walk** on land, **jump** over obstacles, and even **fight** for possession of a cube [32]. This was an early demonstration of open-ended evolution of morphology and behavior. Implementing such a system involves a physics engine and an evolutionary loop (selection, crossover, mutation). Sims' work inspired many later projects in evolutionary robotics and artificial life, showing that simulated evolution can invent creative solutions and body plans.

7. Tierra (Digital Organisms in Memory)

Tierra is a landmark artificial life simulation created by Thomas S. Ray in the early 1990s, in which self-replicating computer programs compete for CPU time and memory space [33]. Ray designed a special virtual computer and instruction set in which small programs (written in this "Tierra assembly language") can copy themselves, mutate, and even engage in parasitic or symbiotic behaviors. The programs in Tierra are **digital organisms** – they can undergo mutation and recombination, and they experience natural selection simply by virtue of survival and replication (there is no explicit fitness function, only the competition for resources) [34] [35]. Over thousands of cycles, complex ecological dynamics emerged: for

example, shorter parasitic programs evolved that exploit longer hosts, hosts evolved immunity, hyper-parasites evolved, etc., mirroring co-evolutionary arms races [36]. Tierra demonstrated the possibility of **open-ended evolution** in a computer: evolution produced varieties of "creatures" and unexpected strategies, though like many such systems it eventually reached an equilibrium. Tierra's source code has been published, and it remains a foundational experiment in digital evolution and artificial life research.

8. Avida (Digital Evolution Platform)

Avida (begun in 1993) is a widely used open-source artificial life research platform for evolving self-replicating programs [37]. Inspired by Tierra, Avida creates a population of digital organisms, each a small program with a protected memory space, that compete for CPU time and occasionally mutate [37] [38]. Unlike Tierra, each organism runs in its own space (preventing direct rewriting of others' code), and the researcher can assign explicit tasks (e.g. mathematical computations) that grant organisms extra "energy" if performed, shaping an evolutionary reward landscape [37] [39]. Avida has been used to study fundamental questions in evolutionary biology – for example, the 2003 Avida experiments demonstrated the spontaneous evolution of complex logic functions from simpler components, supporting theories of how biological complexity arises [40] [41]. It has also been used in undergraduate education via a simplified Avida-ED version. In Avida, one typically defines an environment and fitness criteria, then observes generations of digital organisms adapt and evolve in real-time. It remains one of the most robust platforms for **in silico** evolution studies, producing insights into evolvability, genetic architecture, and evolutionary dynamics in a controlled setting [38] [42].

9. Reaction-Diffusion (Turing Pattern Simulation)

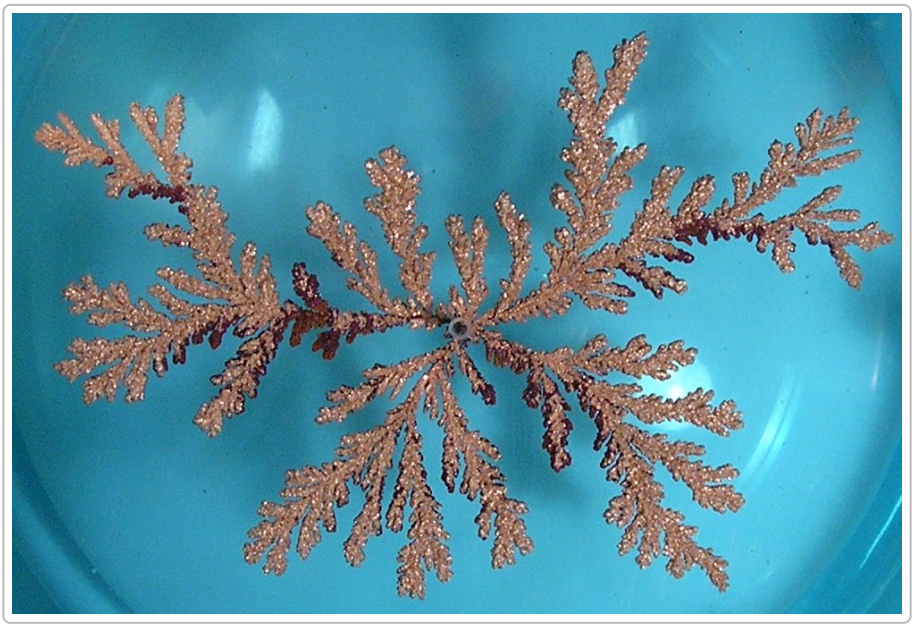
Reaction-diffusion systems are mathematical simulations of interacting chemicals that diffuse through space and react with each other. Alan Turing famously proposed in 1952 that such systems could explain how animals develop patterned skins (stripes, spots) from uniform initial conditions [43]. A classic example is the **Gray-Scott model**, which involves two chemicals (an "activator" and an "inhibitor") diffusing at different rates and reacting locally. When simulated on a grid, reaction-diffusion models often spontaneously form stable patterns – spots, stripes, spirals, labyrinths, and other complex motifs – out of random noise [44] [45]. These are known as **Turing patterns**. In computing, reaction-diffusion is implemented as a set of coupled partial differential equations or iterative rules on each cell. With the right parameters, beautiful organic patterns emerge, resembling animal print patterns, coral growth, or frost on a window. Such simulations are popular in generative art for producing natural-looking textures. For instance, the Gray-Scott CA is essentially a cellular automaton that computes chemical concentrations per cell and yields leopard-like spots or zebra-like stripes, demonstrating how simple chemical rules can generate biological-looking complexity [43] [45].

10. Lindenmayer Systems (Fractal Plants)

L-systems (Lindenmayer systems) are a generative modeling technique introduced by biologist Aristid Lindenmayer in 1968 to describe plant growth mathematically [46]. An L-system is essentially a parallel rewriting grammar: you start with an initial string (axiom) and repeatedly apply production rules to replace symbols with larger strings [47]. By interpreting the resulting strings as drawing instructions (often using turtle graphics), L-systems can generate complex self-similar structures like fractal trees, ferns, and other botanical forms [46] [47]. For example, a simple L-system rule like "X → F-[[X]+X]+F[+FX]-X" can produce a

remarkably realistic branching plant after a few iterations. L-systems capture the essence of developmental growth – each part of the string (plant) expands into sub-parts in parallel. They have been widely used in computer graphics and generative art to create organic patterns and recursive motifs. Because a few recursive rules can yield a whole complex organism, L-systems demonstrate how **fractals** and biological forms can emerge from algorithmic rule-sets. Many libraries (in Python, JavaScript, etc.) exist to experiment with L-systems, making it easy to grow virtual plants and fractal curves.

11. Diffusion-Limited Aggregation (Fractal Growth)



Diffusion-Limited Aggregation (DLA) is a process that produces striking tree-like fractal clusters. First described by Witten and Sander in 1981, DLA involves particles undergoing random Brownian motion that stick together upon contact [48]. A classic simulation places a "seed" particle and then releases new particles from far away: each particle performs a random walk until it hits the cluster, where it stays, and then the next particle is released. Over time, a highly branched, treelike cluster grows (often called a *Brownian tree*) [48] [49]. The structure is fractal – in 2D its fractal dimension is about 1.7, meaning it's very tenuous [49]. DLA patterns closely resemble natural forms like lightning bolts, coral, branching minerals, frost crystals, and dielectric breakdown patterns. The process is "diffusion-limited" because the particles only move by diffusion. DLA can be implemented with a simple loop of random walk and stick, and visualized as growing clusters of dots. It has applications in modeling aggregation phenomena in physics and chemistry. The image above shows a copper sulfate electrodeposition forming a DLA fractal cluster – the branching pattern is characteristic of this algorithm [50] [51].

12. Predator-Prey Ecosystem (Wolf-Sheep Simulation)

Agent-based predator-prey simulations illustrate ecological dynamics like oscillating populations. A well-known example (from the NetLogo models) is the **Wolf-Sheep Predation** model [52]. In this simulation, wolves (predators) and sheep (prey) move around a grid; sheep eat grass to gain energy, wolves eat sheep to gain energy [52]. Both wolves and sheep expend energy moving, and if an agent's energy drops to zero it dies. With sufficient energy, they reproduce. Grass on each cell regrows after being eaten, providing a

renewable food source for sheep [53]. This simple setup leads to emergent cycles: sheep multiply until overgrazing causes a crash, wolves starve after sheep collapse, grass regrows, and the cycle repeats – echoing classic Lotka–Volterra predator-prey oscillations. By adjusting parameters (growth rates, initial populations), one can observe different outcomes (stable coexistence, extinction, etc.). The model demonstrates how complex population dynamics can arise from simple rules at the individual level. Implementations exist in many frameworks; for instance, Mesa (Python) provides a Wolf-Sheep example with visualization [52] [54].

13. Schelling's Segregation Model

Economist Thomas Schelling's segregation model (1971) is a classic agent-based simulation showing how individual preferences can lead to unexpected collective outcomes. The setup: agents of two types (say, orange and blue) occupy cells on a grid, some cells being empty [55]. Each agent wants a certain minimum fraction of its neighbors to be of the same type (even a very modest preference). If that condition isn't met (i.e. an agent is "unhappy"), the agent will move to a random empty location [55]. Schelling found that even if agents *don't* mind having a large proportion of different-type neighbors – for example, they are content if at least 30% are like themselves (meaning 70% could be different) – the simulation inexorably produces highly segregated neighborhoods [56]. Small biases become amplified: eventually most agents end up surrounded by neighbors of their own color, far more segregation than anyone intended [56]. This model vividly demonstrates how local interactions can yield global patterns (a form of emergent self-organization in social systems). Modern implementations (e.g. in NetLogo, Mesa) allow tweaking the tolerance threshold and show via color-coded maps how clusters form. It's an important example in social science of using simulation to explain how segregation can occur "without racist agents." [55] [57]

14. Slime Mold Path Formation (Physarum Simulation)

The unicellular slime mold *Physarum polycephalum* is famous for forming efficient networks connecting food sources. Computer simulations have been developed to mimic this **slime mold behavior** using simple agents and pheromone-like chemicals. In a typical particle-based Physarum model, hundreds or thousands of particles (representing slime mold plasmodium) wander randomly on a grid, leaving a diffusing trail behind [58] [59]. Particles also bias their movement toward higher concentrations of the trail (positive feedback). The result is that initially random motion self-organizes into a network of persistent trails connecting resource points – very much like a slime mold finding the shortest paths between food locations. In essence, it simulates **chemotaxis**: agents follow a chemoattractant that they themselves deposit. The algorithm is surprisingly simple, yet produces complex vein-like patterns [60] [61]. Slime mold simulations have been used to solve mazes and even to compute approximations of road networks. The emergent structure is a minimalist transportation network that often balances efficiency with redundancy (just as the real slime mold does). This has applications in biomimetic design and demonstrates how decentralized coordination can yield near-optimal paths. Many generative artists have also used Physarum models to create visually intriguing patterns.

15. Lenia (Continuous Cellular Automata)

Lenia (from Latin *lenis*, "smooth") is a modern cellular automaton family created by Bert W. Chan (2018) that generalizes Conway's Life into a continuous domain [62]. In Lenia, space, time, and cell states are continuous (or high-resolution) rather than binary and discrete. Cells have a floating-point "mass" and

update according to convolution-like growth rules. This richer framework supports a menagerie of complex self-organizing "lifeforms" – persistent structured patterns that move, grow, and interact in lifelike ways [63]. Lenia forms are often geometric or symmetric and exhibit behaviors reminiscent of organisms (hunting, replicating, etc.), yet they arise purely from local rules in the CA. Chan's work won awards in the artificial life community [64] for demonstrating **evolutionary and biological phenomena** in a CA. Hundreds of Lenia species have been discovered by scanning rule parameter spaces – examples have names like Orbium, fungia, and amoebas, which look and behave in distinct ways. Lenia's significance is in showing that by smoothing out the harsh binary rules of Life into continuous variables, one unlocks a diversity of higher-order emergent phenomena. There are open-source implementations of Lenia, and enthusiasts have used machine learning to search for new "creatures" within it. It stands at the intersection of artificial life and mathematical art.

16. Abelian Sandpile Model (Self-Organized Criticality)

The Abelian Sandpile (Bak–Tang–Wiesenfeld model, 1987) is a simple cellular automaton that was the first discovered example of **self-organized criticality** [65]. Imagine a grid where each cell contains some number of "sand grains." Grains are added randomly one by one. If any cell accumulates too many grains (typically 4 or more), it topples: it loses 4 grains, and each of its four orthogonal neighbors gains one (grains falling outward) [66]. Those neighbors might in turn exceed the threshold and topple, causing a chain reaction (avalanche) that eventually settles into a new stable state [67] [68]. The remarkable outcome: the sandpile system naturally self-organizes to a critical state where avalanches of all sizes occur, following a power-law distribution. In other words, the model exhibits complexity without any tuning of parameters – it reaches a poised state at the edge of chaos. Visually, if you start with an empty grid and drop grains, you'll see bursts of activity (topplings) of varying scope. If you color cells by their number of grains, beautiful fractal patterns (like the Sierpinski triangle) can appear in the aftermath of avalanches [69] [70]. The Abelian sandpile is "abelian" because the order of topplings doesn't affect the final result, making it mathematically tractable. This model has helped scientists understand natural phenomena like earthquakes, forest fires, and other systems that exhibit punctuated equilibria. It's easy to implement and is a staple example in complexity science.

17. Firefly Synchronization

Certain species of fireflies are known to flash in unison – a breathtaking natural phenomenon of self synchronization. An elegant simulation of this is **Nicky Case's Fireflies** interactive (2016), which illustrates how individual oscillators can spontaneously sync up. In the model, each firefly has an internal periodic clock that triggers a flash every so often [71]. The only interaction rule is: when a firefly sees a neighbor flash, it nudges its own clock ahead slightly [71]. That's it. If you initialize a bunch of fireflies with random phases, at first their flashes are scattered. But gradually, clusters of synchrony form, then merge, until eventually (in most cases) the entire swarm is flashing in near-perfect unison [72]. There is no leader or global signal – the synchronization arises from local coupling alone. This firefly model is essentially a discrete version of the Kuramoto synchronization model. It demonstrates how **small-scale interactions** (adjusting phase to neighbors) can produce large-scale coordination [73] [74]. The concept has parallels in many systems: pacemaker cells in the heart, crickets chirping together, even metronomes on a shared board will sync via slight vibrations. The firefly simulation is visual and fun: one can play with it by disturbing some fireflies (out-of-sync flashes) and watching the system resynchronize robustly [75] [76].

18. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an algorithm (Kennedy and Eberhart, 1995) inspired by collective behavior such as bird flocks or fish schools searching for food [77]. It's an optimization method where a "swarm" of particles explores the solution space, each particle representing a candidate solution. The particles move through the space with an initial random velocity, and at each step they are pulled towards two attractors: their own best-known position, and the swarm's globally best-known position [78] [79]. This simple social sharing of information leads the whole swarm to converge on high-quality solutions over time. In effect, PSO simulates a kind of **collaborative learning**: particles remember good positions and communicate them, so the swarm "flocks" to promising regions of the search space. PSO has the flavor of a stochastic simulation – one can visualize particles flying around a landscape, clustering near optima. It was initially intended as a social behavior simulation and only later recognized as an optimizer [77]. The algorithm is popular in machine learning and engineering for continuous optimization problems and is prized for its simplicity (few parameters, easy to implement). It demonstrates how analogies from animal group behavior can yield powerful computational techniques.

19. Genetic Algorithms (Evolutionary Optimization)

Genetic Algorithms (GAs), pioneered by John Holland in the 1970s, are optimization and search procedures modeled on biological evolution. In a GA, a population of candidate solutions (individuals encoded typically as strings, analogous to chromosomes) evolves over generations to improve their fitness with respect to some objective function [80]. The process starts with a random population, then repeatedly applies genetic operators: **selection** (preferentially picking the fitter solutions), **crossover** (recombining parts of two solutions to create offspring), and **mutation** (randomly tweaking parts of a solution) [81] [82]. Over successive generations, good traits mix and spread, and the population "adapts" to produce better and better solutions on average. GAs are versatile because they make few assumptions about the search space – they have been applied to engineering design, scheduling, evolving neural network weights, game playing, and more. For example, a GA can evolve an 8-bit string to maximize the number of 1s: starting with random strings, it will quickly discover the all-1s solution through selection and crossover. More interestingly, GAs can evolve complex structures – for instance, evolving antenna designs for NASA or creating art. The key hallmark of a GA is that it **maintains diversity** in a population and uses randomized exchanges to explore the space, as opposed to greedy improvement of a single solution. This can avoid local optima and is a reason GAs often find creative, non-intuitive solutions to problems.

20. Artificial Bee Colony (Swarm Optimization)

The **Artificial Bee Colony (ABC)** algorithm (Derviş Karaboğa, 2005) is a swarm intelligence method inspired by the foraging behavior of honey bees [83]. It imagines a colony of artificial bees cooperating to find the best "food sources," which correspond to good solutions of an optimization problem. The colony is typically divided into three roles: **employed bees** (each exploits a specific food source and shares information about it via a "dance"), **onlooker bees** (they watch the dances and decide which food source to explore next, probabilistically favoring richer sources), and **scout bees** (if a food source is exhausted, a scout searches for new random sources) [84] [85]. In algorithm terms, each candidate solution is like a food source. Employed bees repeatedly refine their assigned solution (perform a local search) and report its quality (nectar amount). Onlookers pick promising solutions to further explore based on those quality reports, which introduces a selection pressure. Scouts introduce new random solutions to replace ones that have

stagnated (avoiding convergence at a local optimum) [84] [86]. Through many cycles of this process, the bee colony collaboratively hones in on high-quality solutions. ABC has been successfully applied to various optimization problems and is valued for its balance of exploration (scouts) and exploitation (employed/onlooker bees). It's another example of how metaphors from nature – in this case, honey bee foraging – can lead to effective search algorithms grounded in **collective intelligence** [87] [88].

Each of these simulations – from cellular automata to swarm algorithms – offers a unique lens on how simple rules at the micro-level can generate complex, often lifelike phenomena at the macro-level. Whether for pure visualization fun (e.g. making art and patterns) or serious research (optimizing engineering designs, exploring evolution), these well-known examples form a toolkit of ideas that can be implemented and experimented with in code. They span the gamut from purely digital "organisms" to abstract optimization agents, but all show the power of emergence, adaptation, and self-organization in computational systems. Each can be visualized in creative ways, and indeed many have thriving communities sharing demos on GitHub or other platforms – providing a rich playground for anyone interested in generative art or artificial life. Enjoy exploring these simulations and creating your own variations!

Sources: The descriptions above are synthesized from classic literature and online resources on artificial life and generative algorithms [1] [4] [20] [22] [28] [33] [37] [43] [46] [48] [52] [55] [59] [62] [66] [71] [77] [80] [83], with specific attributions for direct quotations and factual details as indicated. Each simulation is an active field of study and creativity, and further details (including sample implementations) can be found in the linked references.

---

[1] [2] [3] [4] [5] [6] [7] Conway's Game of Life - Wikipedia
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

[8] Langton's ant - Esolang
https://esolangs.org/wiki/Langton%27s_ant

[9] [10] [11] [12] [13] Langton's Ant | Home
https://bnray53.github.io/langtonsAnt/

[14] [15] [16] [17] Boids - Wikipedia
https://en.wikipedia.org/wiki/Boids

[18] [19] [20] [21] Ant colony optimization algorithms - Wikipedia
https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

[22] [23] [24] [25] [26] [27] Richard Dawkins: Biomorphs 1986 – media+art+innovation
https://mediartinnovation.com/2014/06/01/richard-dawkins-biomorphs-1986/

[28] [29] [30] [31] [32] Evolved Virtual Creatures
https://www.karlsims.com/evolved-virtual-creatures.html

[33] [34] [35] [36] Tierra (computer simulation) - Wikipedia
https://en.wikipedia.org/wiki/Tierra_(computer_simulation)

[37] [39] Avida (software) - Wikipedia
https://en.wikipedia.org/wiki/Avida_(software)

38 40 41 42 Avida Digital Evolution Platform
https://alife.org/encyclopedia/digital-evolution/avida/

43 44 45 Turing pattern - Wikipedia
https://en.wikipedia.org/wiki/Turing_pattern

46 47 GitHub - danbgray/lsystem: Lindenmayer system editor
https://github.com/danbgray/lsystem

48 49 50 51 Diffusion-limited aggregation - Wikipedia
https://en.wikipedia.org/wiki/Diffusion-limited_aggregation

52 53 54 Wolf-Sheep Predation Model — Mesa .1 documentation
https://mesa.readthedocs.io/stable/examples/advanced/wolf_sheep.html

55 56 57 Schelling Segregation Model — Mesa .1 documentation
https://mesa.readthedocs.io/stable/examples/basic/schelling.html

58 Slime Mold Simulation. Experiments. - YouTube
https://www.youtube.com/watch?v=qryINYcgO1s

59 60 61 Michael Fogleman: Physarum Simulation
https://www.michaelfogleman.com/projects/physarum/

62 63 64 Lenia - Wikipedia
https://en.wikipedia.org/wiki/Lenia

65 66 67 68 69 70 Abelian sandpile model - Wikipedia
https://en.wikipedia.org/wiki/Abelian_sandpile_model

71 72 73 74 75 76 Fireflies
https://ncase.me/fireflies/

77 78 79 Particle swarm optimization - Wikipedia
https://en.wikipedia.org/wiki/Particle_swarm_optimization

80 81 82 Genetic algorithm - Wikipedia
https://en.wikipedia.org/wiki/Genetic_algorithm

83 84 85 86 87 88 Artificial bee colony algorithm - Wikipedia
https://en.wikipedia.org/wiki/Artificial_bee_colony_algorithm