Cellular Automata for Molecular Life Games

Ian Holmes
Department of Bioengineering
University of California
California, Berkeley
ihh@berkeley.edu

ABSTRACT

An experimental turn-based multiplayer game framework for mobile devices is described. Using cellular automata, the framework implements models of condensed-matter physics, mathematical biology, and the computational study of artificial life.

1. INTRODUCTION

Cellular automata (*CA*) are a versatile platform for a wide range of models from physics, biology and chemistry[1]. They are also widely employed in the study of artificial computational life (*A-Life*). CAs have supported diverse A-Life designs, whether inspired by electronic circuits[2, 3], cells, insects, or other macro-biological systems[4, 5], or kinematic systems[6].

Another type of A-Life involves code that is placed under direct pressure to evolve. The ancestor of all these is the competitive programming game CoreWar [7, 8]. Successors Tierra[9] and Avida[10] more directly investigate the behavior of genetic algorithms.

With the goal of investigating systems such as this and developing associated game mechanics, the PixelZoo game engine has been designed to support a persistent, multiplayer, computationally-dense world using CA. This paper reviews the design and progress of PixelZoo, including implementation of several biophysically-inspired models simulating polymer and RNA folding kinetics.

CAs have been used as game devices before, as in Sim City, Dwarf Fortress, and Minecraft. They have also been used in software toys and art such as PowderToy and CAPOW. There are many freely available engines to simulate various kinds of CA, such as Golly. In relation to such prior efforts, PixelZoo (i) is more focused on molecular biophysics and evolution, (ii) is intended to be a game rather than a simulator.

1.1 Molecular Evolution Models

One of the best-studied models of the origin of life is the "RNA World" hypothesis, which posits that RNA molecules were the first self-replicating entities[11]. The chemistry of RNA naturally favors self-replication under favorable conditions, such as spatial concentration of the necessary ingredients. Early life has been conjectured to involve concentration of RNA precursors within *micelles*: essentially, spontaneously-forming bubbles of amphiphilic (detergent) molecules.

Testing hypotheses about the distant past is difficult, but synthetic life experiments have lent plausibility to the RNA World[12]. RNA has been discussed in the context of A-Life models[13]. Biophysicists have devoted considerable attention to CA-like lattice models of RNA folding [14, 15, 16, 17], following earlier studies of protein folding on the lattice [18, 19].

2. SIMULATION

The PixelZoo board is an $S \times S \times D$ slab of the cubic lattice, requiring $16S^2D$ bytes on a 64-bit architecture (128 bits/cell), plus any Scheme programs that are attached. Gameplay is generally restricted to a thin 2D slab ($D \leq 2$).

In a diffusion-based simulation, speed is paramount. The PixelZoo language exposes a fast low-level state machine for majority usage during play, generalizing the concept of the CA state lookup table to allow partial matching and rewriting of bitfields in the local lattice neighborhood (but not much else). There is also a high-level functional language (Scheme) available both for generation of low-level instructions, and for occasional in-game scripting usage.

2.1 Low-Level Virtual State Machine

The low-level state machine supports two alternatives for periodic updates: synchronous (fixed time interval between calls) and asynchronous (exponentially-distributed time between calls, scaled by particle's update rate). The random number generator is a Mersenne twister, and supports replay.

Cells can address their local neighborhood. Each cell has a program that can be copied from neighboring cells, or re-initialized from a global program list, but cannot be otherwise modified. There are 64 bits of state per cell (of which 16 are reserved). There is also a pointer to dynamic storage that can only be accessed from Scheme.

The 64 bits of state are divided into bitfields. 16 bits are reserved for a type field, indexing a global program table that determines how the cell will be updated. The remaining 48 bits are used as dynamic storage; the division into bitfields depends on the type, with some universal restrictions (e.g. bitfields cannot straddle a 32-bit boundary).

The 16-bit type field selects the global program to update the cell and its neighborhood, and specifies the naming and sizing scheme for dynamic bitfields.

The virtual machine resembles an ultra-minimal subset of a typical RISC register-machine architecture. The instruction set is low-level, recognizably close to ARM[20] and not too far from CoreWar's RedCode[7]. The instructions can be quickly interpreted from or implemented in C, and will be optimized reasonably well by C compilers. Such details of the implementation are left opaque by design. Code written for the low-level state machine cannot reflect on or modify its own program.

INSTRUCTION SET. The following minimalistic operations were selected with the goal of fast simulation of particles and polymers. Excluding random numbers and Scheme, the core instructions are readily compiled to C (`switch`, `if`, `goto`, integer math) or directly to a RISC instruction set like ARM[20].
`Load`, `Store`: read and write neighborhood bitfields.
`Add, Subtract`: register arithmetic.
`Compare, Switch`: arithmetic comparison, or C-like `switch`, dispatching control based on register value.
`Copy-Program`: copy a program unmodified to a target cell; either from a source cell, or from a global program list. Can optionally copy (or destroy) the associated S-expression storage.
`Branch`: forward branch to labeled routine.
`Branch-Random`: branches conditional on random bit.
`Branch-Neighbor`: yields to neighbor cell's program.
`Call-Script`: yields to arbitrary Scheme function.

It bears re-emphasis that this choice of instruction set is intended to *limit* the expressive power of programs to something comparable to state lookup tables. Excluding the `Call-Script` and `Branch-Neighbor` instructions, control flow is monotonic: there is no stack, and no gosub or return keywords. Branches always go forward, so loops are explicitly prohibited. Each program is a Directed Acyclic Graph (DAG) of finite determinable length. It is thus easy for a server to reason about programs, i.e. to analyze player-uploaded code and maintain strong performance guarantees while running thousands of player programs effectively in parallel. The constraints do not prevent players from implementing more sophisticated computational behaviors within the CA, such as Turing machines, Langton ants, and Wireworld-type circuits. However, the constraints do help ensure that such user-uploaded programs multi-task cooperatively and do not hog resources, or hang.

2.1.1 Addressing Modes

The machine supports several addressing modes for memory locations and operands. *Immediate operands* allows specification of constants within the code, while *register operands* allows these constants to be loaded into registers first (enabling some code re-use).

Memory addressing is local to the neighborhood cell. *Direct addressing* requires bitfield addresses and relative cell coordinates to be hardwired into the code, while in *indirect addressing* these are specified by register values at run time. A relative address *(x-offset,y-offset,bitfield-shift,bitfield-width)* is encoded in 32 bits.

In principle, indirect or register addressing are not necessary to implement polymers: we can explicitly enumerate all cases via immediate and direct addressing. In practice, it's useful to allow some indirect/register addressing, to reduce program sizes via re-use of subroutine-like code.

2.1.2 Assembly Language

The virtual machine may be programmed directly in XML, but this XML is more usually generated using a Scheme-based assembly language.

The assembler has a built-in library for programming CAs on the square lattice: replicating patterns over Moore or von Neumann neighborhoods, implementing reaction-diffusion models, implementing turtle-type agents, and so on.

Selected functions from this library are exposed to the player, to facilitate game design (e.g. diffusions, turtles) and reduce program sizes.

2.2 Higher-Level Functions

As well as being used at assembly time to specify macros, Scheme functions can be invoked dynamically by `Call-Script` for higher-level supervisory functions of an occasional nature, such as path-finding or goal-directed planning. These features are restricted to game designers.

An optional S-expression can be associated with any cell for dynamic storage associated with the agent scripts. This storage is opaque to the state machine: there is one instruction (`Copy-Program`) that, when rewriting the type field of a cell, is allowed to relocate or destroy the S-expression associated with it. Other than this, the low-level machine may not copy, create, read or modify cell S-expressions. In contrast, the Scheme supervisor has access to the state machine environment, board API, and multiplayer client API.

The polymers, RNA molecules, and other models described here (Sections 2.3 through 2.5) do not require Scheme scripting, only Scheme assembly language generation (which is a one-time cost).

2.3 Polymer State Machines

The fundamental biological molecules (DNA, RNA and proteins) are all polymers. In 2D, closed-loop polymers can also represent thermodynamically fluctuating enclosures like micelles.

Drawing on the stochastic physics of polymers [21] and their simulation on lattices [22, 23], PixelZoo uses a bidirectionally-linked list model. The possible configurations of a cell, its neighbors, and adjacent empty spaces must be explicitly enumerated, as shown in Figure 1.

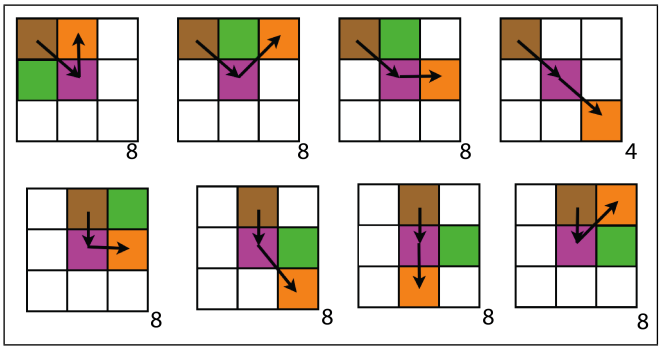

Figure 1: Configurations of upstream (brown) and downstream (orange) neighbors adjacent to a central cell (purple) and a mutually-adjacent neighbor (green) on a 2D lattice. Numbers indicate the degeneracy of each configuration (via rotation and reflection). There are 60 configurations in total.

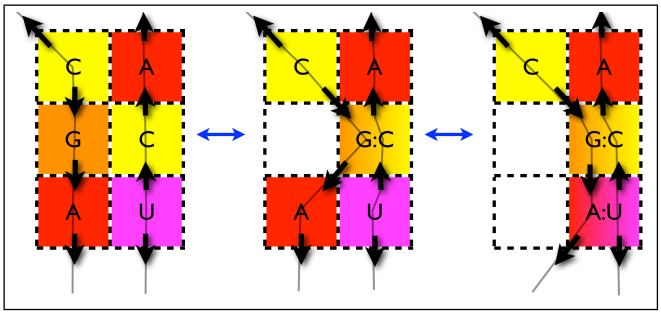

Figure 2: Base-pairing in the RNA model involves two nucleotides occupying the same cell, which then has up to four neighbor bonds.

The implementation of each case requires several steps (including verifying the bidirectional bond integrities, and checking the target cell is empty). To illustrate the need for some of the "frills" in the core CA (specifically indirect addressing and a Scheme-based assembler), we observe that when using `Load`, `Switch`, and `Store` with no indirect addressing (essentially explicit lookup tables), the polymer program XML was 2.3Mb (52k gzipped). With indirect addressing, this reduces to 127k (gzipped: 3k); using the Scheme assembler, it is just 5k (gzipped: 1k). Thus the Scheme assembler is, first and foremost, a compression strategy.

2.4 RNA State Machines

We model RNA as a two-tolerant random walk on the lattice[14]; that is, up to two nucleotides can occupy a single cell, as long as they are base-paired (Figure 2). In biochemistry, base-pairing is the mechanism whereby a nucleotide polymer pairs up to its complementary partner, generating folded structures such as DNA's double-stranded helix. It is fundamental to the self-replication of nucleotide sequences.

RNA basepair models require up to 4 bond directions and two base values to be stored in each cell. The full range of bond configurations can be encoded in 16 bits/cell on the 2D lattice (23 bits/cell on the 3D lattice). In practice it is convenient, albeit slightly more wasteful of bits, to use distinct bitfields to encode the bond directions, presence/absence flags, and bases. This requires 21 bits/cell on the 2D lattice, and 25 bits/cell on the 3D lattice.

2.5 Agent State Machines

PixelZoo also implements various agent models inspired by natural systems at longer-than-molecular length scales. These include the bacterial tumble-run model of directed motion alternating with random walks[24], various forms of reaction-diffusion, directed diffusion, and diffusion-limited aggregation [25], the two-dimensional Ising model of magnetic spins [26], the Lotka-Volterra predator-prey systems [27, 28], ecological rock-paper-scissors games [29], forest fires [30], and lattice versions of the Wright-Fisher model of genetic drift [31].

3. GAME DESIGN

Rendering of the board uses an isometric view switching from monochromatic tiles (at distant zooms) to sprites (at close zooms).

3.1 Basic Play

The background story involves terraforming planets from an orbital craft. The game plays like an isometric painting program, but with live pixels.

The primary game goal is to maintain dynamic equilibrium inside a micelle among a population of three chemical species in a rock-paper-scissors ecology.

3.2 Network Play

The multiplayer game is implemented using a RESTful server [32]. The client posts a temporary lock on a board at the start of the player's turn, and can then "check out" the board, saving it periodically during the turn. Locks (and hence turns) are time-limited, and there is a minimum time between turns by the same player. There are also daily limits on the particles that players can place on a board.

The competitive multiplayer goal is to keep your population alive under attack. Keeping the population alive earns in-game money which can be spent designing new particles.

The mobile game client presents an interface for the player to log in and out, browse worlds, pick a world, select terraforming tools, and play.

3.3 Creating New Tools

A player can post XML to server describing new particles and tools (with restrictions; e.g. no Scheme code is allowed at present, due partly to App Store prohibitions on downloaded code).

Creating and using new reaction-diffusion particles and spray-tools costs in-game money, which can be earned by playing, or when others buy a player's tools.

3.4 Implementation

PixelZoo is implemented in Gnu C, libXML, GDataXML, ChibiScheme, XCode, and Perl Catalyst. The prototype RNA CA was implemented in Java.

4. ACKNOWLEDGEMENTS

Many thanks are due Alex Shinn, Richard Evans, Michael Mateas, Sean Eddy, Gerald Joyce, Chris Quince, and Rudy Rucker for help and inspiration.

5. REFERENCES

[1] J. L. Schiff. *Cellular Automata: A Discrete View of the World*. Wiley Series in Discrete Mathematics & Optimization. Wiley, 2007.

[2] J. Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[3] A. K. Dewdney. The cellular automata programs that create wireworld, rugworld and other diversions. *Scientific American*, pages 146–149, 1990.

[4] M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, October 1970.

[5] C. G. Langton. Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1-3):120 – 149, 1986.

[6] W. M. Stevens. A self-replicating programmable constructor in a kinematic simulation environment. *Robotica*, 29(1):153–176, January 2011.

[7] D.G. Jones and A.K. Dewdney. *Core War Guidelines*. Department of Computer Science, the University of Western Ontario, 1984.

[8] B. Vowk, A. Wait, and C. Schmidt. An evolutionary approach generates human competitive Corewar programs. In M. Bedau, P. Husbands, T. Hutton, S. Kumar, and H. Sizuki, editors, *Workshop and Tutorial Proceedings Ninth International Conference on the Simulation and Synthesis of Living Systems(Alife XI)*, pages 33–36, Boston, Massachusetts, 12 September 2004. Artificial Chemistry and its applications workshop.

[9] T. S. Ray. Synthetic life: Evolution and optimization of digital organisms. In K. R. Billingsley, Brown H. U. Iii, and E. Derohanes, editors, *Scientific Excellence in Supercomputing: The 1990 IBM Contest Prize Papers*, pages 489–531, 1992.

[10] C. Adami and C. Titus Brown. Evolutionary Learning in the 2D Artificial Life System "Avida". In *eprint arXiv:adap-org/9405003*, page 5003, May 1994.

[11] C.R. Woese. *The genetic code: the molecular basis for genetic expression*. Modern perspectives in biology. Harper & Row, 1967.

[12] N. Paul and G. F. Joyce. A self-replicating ligase ribozyme. *Proceedings of the National Academy of Sciences*, 99(20):12733–12740, 2002.

[13] P. Schuster. Extended molecular evolutionary biology: Artificial life bridging the gap between chemistry and biology. *Artificial Life*, 1(1-2):39–60, 1994.

[14] P. Leoni and C. Vanderzande. Statistical mechanics of RNA folding: A lattice approach. *Phys. Rev. E*, 68:051904, Nov 2003.

[15] D. Jost and R. Everaers. Prediction of RNA multiloop and pseudoknot conformations from a lattice-based, coarse-grain tertiary structure model. *The Journal of Chemical Physics*, 132(9):–, 2010.

[16] R. A. Zara and M. Pretti. Exact solution of a RNA-like polymer model on the Husimi lattice. *The Journal of Chemical Physics*, 127(18):–, 2007.

[17] J. Gillespie, M. Mayne, and M. Jiang. RNA folding on the 3D triangular lattice. *BMC Bioinformatics*, 10:369, 2009.

[18] K. A. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509, 1985.

[19] V. S. Pande and D. S. Rokhsar. Folding pathway of a lattice model for proteins. *Proceedings of the National Academy of Sciences*, 96(4):1273–1278, 1999.

[20] D. Seal. *ARM Architecture Reference Manual*. Addison-Wesley, London, UK, 2000.

[21] M. Doi and S.F. Edwards. *The Theory of Polymer Dynamics*. International series of monographs on physics. Clarendon Press, 1988.

[22] J. M. Vianney A. Koelman. Cellular-automaton-based simulation of 2D polymer dynamics. *Phys. Rev. Lett.*, 64:1915–1918, Apr 1990.

[23] B. Ostrovsky, G. Crooks, M. A. Smith, and Yaneer Bar-Yam. Cellular automata for polymer simulation with application to polymer melts and polymer collapse including implications for protein folding. *Parallel Computing*, 27(5):613–641, 2001.

[24] G. Rosser, A. G. Fletcher, D. A. Wilkinson, J. A. de Beyer, C. A. Yates, J. P. Armitage, P. K. Maini, and R. E. Baker. Novel methods for analysing bacterial tracks reveal persistence in Rhodobacter sphaeroides. *PLoS Comput Biol*, 9(10), 10 2013.

[25] T. A. Witten and L. M. Sander. Diffusion-limited aggregation, a kinetic critical phenomenon. *Phys. Rev. Lett.*, 47:1400–1403, Nov 1981.

[26] L. Onsager. Crystal statistics. I. A two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, Feb 1944.

[27] A. J. Lotka. Contribution to the theory of periodic reactions. *The Journal of Physical Chemistry*, 14(3):271–274, 1909.

[28] R. Hirota, M. Iwao, A. Ramani, D. Takahashi, B. Grammaticos, and Y. Ohta. From integrability to chaos in a Lotka-Volterra cellular automaton. *Physics Letters A*, 236(1-2):39 – 44, 1997.

[29] K. Tainaka. Physics and ecology of rock-paper-scissors game. In *Revised Papers from the Second International Conference on Computers and Games*, CG '00, pages 384–395, London, UK, UK, 2002. Springer-Verlag.

[30] I. Karafyllidis and A. Thanailakis. A model for predicting forest fire spreading using cellular automata. *Ecological Modelling*, 99(1):87 – 97, 1997.

[31] I. Mathieson and G. McVean. Estimating selection coefficients in spatially structured populations from time series data of allele frequencies. *Genetics*, 193(3):973–984, 2013.

[32] R. T. Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.