Poster: A Visual Programming Language for Cellular Automata

Deacon McIntyre
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
mcintydeac@myvuw.ac.nz

Michael Homer
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
mwh@ecs.vuw.ac.nz

***Abstract*—Cellular automata are simulations of cells interacting with each other based on simple rules. Despite the simplicity, they can exhibit complex behaviour, and have a number of applications in fields such as medicine, biology, mathematics, and more. As a result, people from a variety of different backgrounds and skill-sets may find cellular automata useful to their work or research. There exist tools to explore well-known automata, but many require some form of textual programming ability, or do not offer easy and approachable ways to customise automata. Our software aims to be more accessible for those without backgrounds in programming or cellular automata, to allow users to more easily explore and modify automata. To achieve this, we have developed a visual programming language, where users can connect components to create their own automata from scratch, without any textual programming.**

***Index Terms*—visual languages, drag and drop, cellular automata**

I. Introduction

Cellular automata (CA) have applications in a number of different fields, so people with a reason to explore CA may not have a technical background. Existing exploratory tools for CA have significant barriers that make it hard for non-technical users to explore CA as they might need. We have built a visual programming language (VPL) and tool to allow CA rulesets to be defined without textual programming and with a visual approach supporting exploration.

Figure 1 depicts the Game of Life implemented in the language. While our prototype is not intended for large-scale or rigorous simulation, the language and interaction design should be transferrable to larger systems. This prototype runs in a commodity web browser, and is already sufficient for initial exploration of a CA or experimenting with variations.

A. Goals

The design goals for our VPL were for the tool itself to be readily accessible, and provide prompt feedback to support experimentation; for the language notation to be understandable to non-technical users; and for the system not to require prior knowledge of CA, but nonetheless to support rules of sufficient complexity to allow defining a wide variety of well-established CA including Conway's Game of Life [1], Wireworld [2], and Langton's Ant [3]. Explicit non-goals were simulation performance, world scale, compatibility with existing CA rule formats, and non-2D automata.

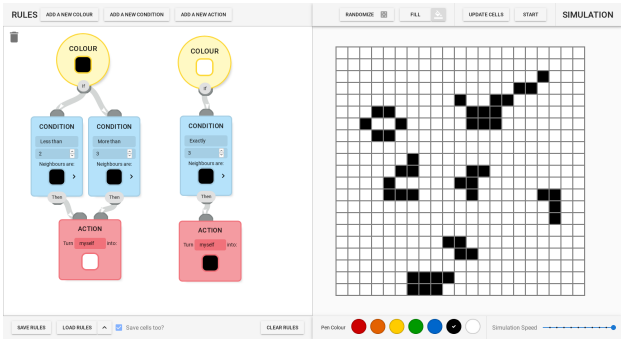

Fig. 1. Conway's Game of Life implemented using our software.

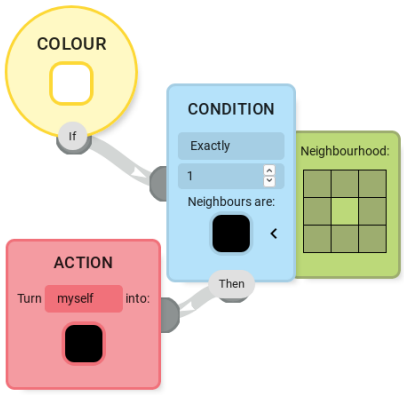

Fig. 2. A very simple ruleset in our system: any white cell where exactly one of the eight surrounding cells is black will become a black cell in the next generation.

II. Language Design

Programs (CA rulesets) in this language consist of a graph of connected components describing the transformation that should occur for particular groups of cells between one generation and the next, with different kinds of vertex contributing different behaviour. Each component follows a "flow-chart"–style model. A very simple ruleset is presented in Figure 2. There are three kinds of node that appear in the program:

This is the author's version of this paper, published in the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 10-14 August 2020, Dunedin, New Zealand. The final publication is available in IEEE Xplore via https://doi.org/10.1109/VL/HCC50065.2020.9127283.

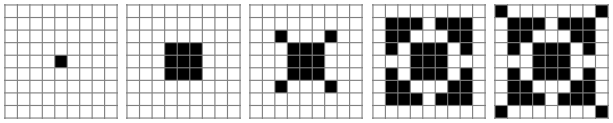

Fig. 3. The first five generations of the ruleset from Figure 2.

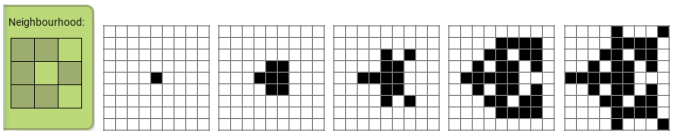

Fig. 4. The first five generations of the ruleset from Figure 2, with a modified asymmetric neighborhood.

a) Colour: (round, yellow): any cell of the specified colour (here, white). At run time, execution starts for each cell from a matching Colour node and follows the graph edges.

b) Condition: (rectangular, blue): a filter (here, exactly one neighbouring cell is black). Execution continues in this branch only if the condition is met. The green box to the right allows specifying exactly which surrounding cells are "neighbors" (shown/hidden with the arrow).

c) Action: (rectangular, red): a change of state (here, the cell turns black). Alternatively, the action could affect selected neighboring cells instead.

Colours and Conditions can have arbitrary out-degree, while Conditions and Actions can have arbitrary in-degree. Conditions can be connected in series for conjunction and in parallel for disjunction by virtue of the "flow" style. Connections are formed by dragging from the buttons on the edge of each node.

Execution of Figure 2 proceeds from each white cell in turn, checking for exactly one black neighbour, and marking that cell to turn black in the next generation. Figure 3 depicts the first five generations, starting from a single cell, while Figure 4 shows experimenting with an asymmetric neighborhood in the same ruleset, which is trivially accomplished in this system. Figure 1 shows a more complex ruleset implementing Game of Life, with two Colours and three Conditions.

Nodes can be added, removed, changed, or reconnected at will, including "live" as the automaton is executing, to see the effects of modifications immediately. The tool also supports single-stepping the CA for more detailed inspection.

III. Future Work and Alternative Designs

Our initial implementation (Figure 5) had two major differences, which we removed following a preliminary user experiment (approved VUW HEC). A Colour had multiple "Property" outputs, representing itself, neighbours, and potentially extra per-cell data, which could be connected separately. A "Transform" node could manipulate data (for example, refining or combining neighbour sets, adding numbers, or performing logical operations). This model was more of a data-flow than control-flow graph, but Transform nodes in particular were highly confusing to users. Future work could incorporate these functional elements better, supporting a wider range

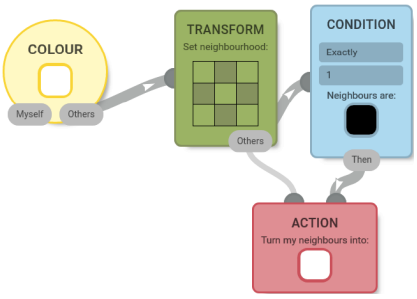

Fig. 5. A Transform node in use to specify a restricted neighborhood to be reused. The Colour node has two properties.

of automata. Ideally, all possible connections would have meaning, which our initial design did not quite reach.

IV. Implementation

The tool is implemented to run in a web browser and available online at https://homepages.ecs.vuw.ac.nz/~mwh/demos/dm-vpl-ca/, using the Vue and jsPlumb libraries.

V. Related Work

Existing CA tools with overlapping goals to ours include the One-dimensional Cellular Automata Playground [4] and Cellular Automata Laboratory [5] web-based tools, but neither allows two-dimensional automata without standard programming. Golly [6] is a popular CA simulation and experimental tool, but its textual rule format is complex and internal or live editing is not supported.

As a visual language our system most resembles LabVIEW, and has particular design similarities in using domain-specific visual notations [7]–[9], though we avoid using domain-specific *terms* where possible. We also drew from Scratch [10] in particular in its *liveness* and *tinkerability*, finding that editing the program during execution is invaluable for experimentation when uncertain about what will or should happen.

References

[1] M. Gardner, "Mathematical games — the fantastic combinations of john conway's new solitaire game "life"," *Scientific American*, vol. 223, no. 4, pp. 120–123, 1970.

[2] A. K. Dewdney, "Computer recreations," *Scientific American*, vol. 262, no. 1, 1990.

[3] C. G. Langton, "Studying artificial life with cellular automata," *Physica D: Nonlinear Phenomena*, vol. 22, no. 1–3, pp. 120–149, 1986.

[4] A. Bell, "One-dimensional cellular automata playground," https://albell.github.io/cellular-automata-playground, 2018.

[5] R. Rucker and J. Walker, "Cellular automata laboratory," https://www.fourmilab.ch/cellab/webca/?show=demo, 2017.

[6] A. Trevorrow and T. Rokicki, "Golly," https://golly.sourceforge.net/, 2018.

[7] E. Howard, "Visual programming: Concepts and implementations," Master's thesis, Miami University, Ohio, 1994.

[8] M. Erwig and B. Meyer, "Heterogeneous visual languages—integrating visual and textual programming," in *Proceedings of Symposium on Visual Languages*, 1995, pp. 318–325.

[9] M. Noone and A. Mooney, "Visual and textual programming languages: A systematic review of the literature," *Journal of Computers in Education*, vol. 5, no. 2, pp. 149–174, 2018.

[10] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. M. a nd Eric Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009.