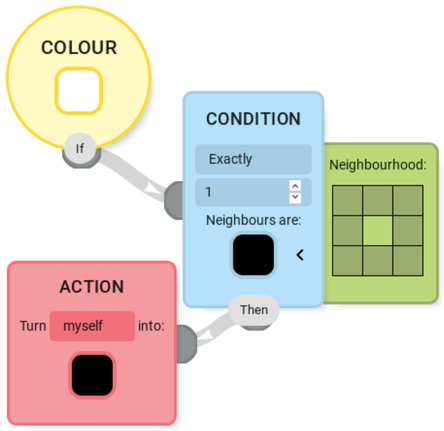
A Visual Programming Language for Cellular Automata

"Nodes and lines" control-flow editing of cellular automata rules

Deacon McIntyre and Michael Homer

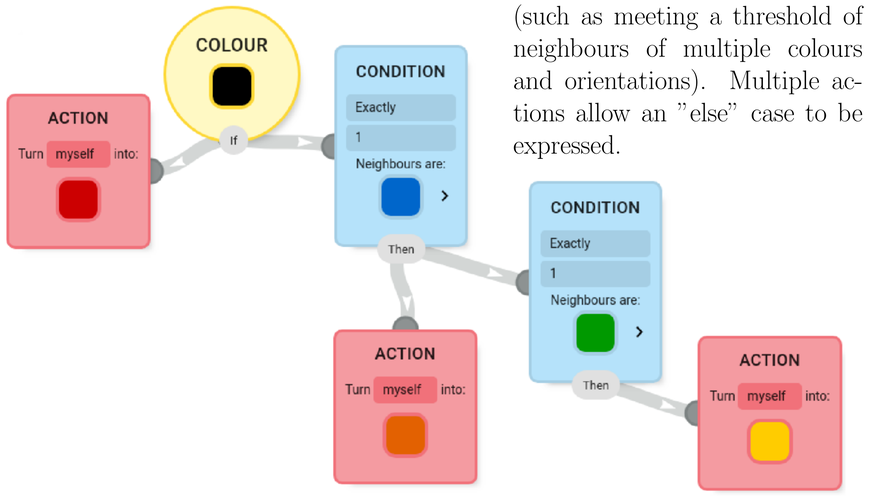
A simple automaton



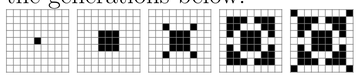
Three main nodes:

- **Colour**, in yellow, representing a cell with a specified colour at the start of the round, and the starting point of each parallel computation
- **Condition**, in blue, which can examine the world around the cell and decide whether to continue or not (with the optional custom neighbourhood setting in green)
- **Action**, in red, which lets the cell decide to change itself, or its neighbours, to a different colour for the next round

Connecting these nodes together by drag and drop can create a range of complex automata behaviours including nested, conjunctive, and disjunctive branches.

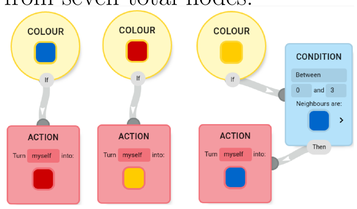


The program to the left produces the generations below:



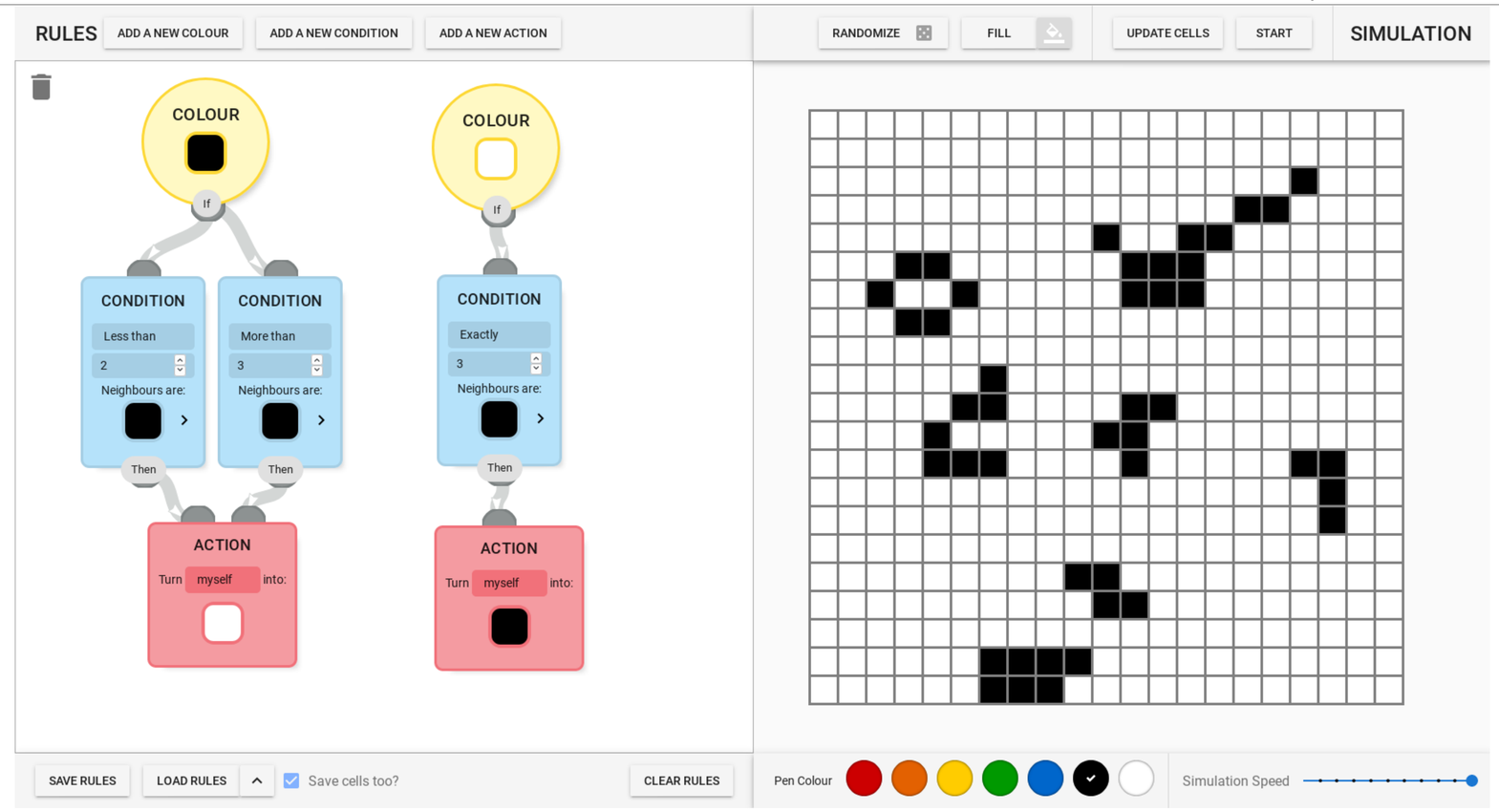
Implementing an existing automaton

Below: Wireworld, a path-following automaton, assembled from seven total nodes.



Combining conditions in parallel (right) creates a disjunction, where either condition is sufficient to reach the action.

Combining conditions in series (below) creates a conjunction, where all conditions must hold (such as meeting a threshold of neighbours of multiple colours and orientations). Multiple actions allow an "else" case to be expressed.

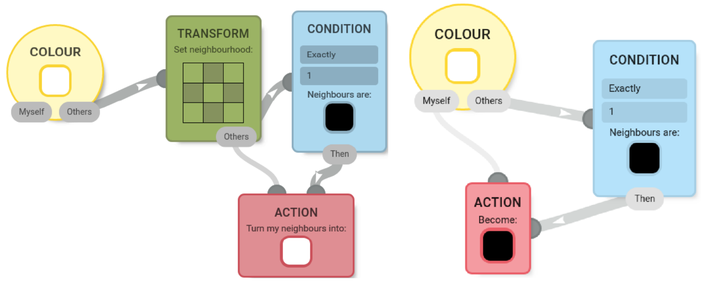


Above: Conway's Game of Life implemented in our system.

Future Work

Some automata, like Langton's Ant and colony simulations, rely on some "extra" data known by a cell's rules (for example, which direction it came from, or how long since it encountered a green cell). We'd like to support this too, but additional node and connection types seem to tax the user significantly. Improving discoverability and decreasing friction is an ongoing goal.

An earlier version



The first version of the language supported a "transform" node, and multiple output points on each node, with the idea that neighbours, extra data, and different targets would all make use of these, with every available connection meaningful. Preliminary user testing found this too confusing, and we settled on the simpler three-node version.

Quickly experimenting with custom automata

Tweaks to rules, or whole new automata, can be prototyped just through connecting nodes together and visual selection of colours and neighbourhoods, even while the program is running.

Dragging a connection from one node to another creates a runnable program immediately, and it can be single-stepped or the world manually edited while it runs to explore different behaviours. With multiple colours and custom neighbourhoods very advanced behaviours can be realised, including clustering and long-distance cyclic motion.

The low entry barrier makes experiments cheap and live editing enables real-time testing of candidate changes. We aim to allow non-programmer users to create their own automata for their own domain or for fun, as well as a range of standard machines.

Try it!

Point any browser at **mwh.nz/demos/vlhcc2020** to use the system live, including all of these examples and implementations of some other well-known automata. (n.b. some easy-to-create automata will flash rapidly; simulation speed is adjustable)

mwh.nz/pubs/vlhcc2020