The HexCore programming environment and simulator for CA experiments

Divan Burger, Megan Duncan, Apurva Kumar, Leon van Dyk and Karl Zöller

Team HexCore successfully completed a project in 2011 for the Department of Computer Science's final-year module in Software Engineering on the evolution and refinement of a cellular automata simulator and editor, designed by Team Core in 2009 (see article in *Innovate* 5, 2010).

Cellular automata (CA) have always been useful and valuable tools for the study of many phenomena, both practical and theoretical. Many researchers find cellular automata helpful in biology, artificial intelligence, geology and many other fields. The aim with this project was to implement a new cellular automaton simulator and editor that would provide enough generality to be adapted to many situations, as well as one with a large enough arsenal of features to describe a particular model in great detail. If used on a network, the system would make efficient use of the computational power at hand.

The goal was to develop a system that provided all the functionality of the 2009 system, as well as added support for heterogeneous worlds, rules that accept mathematical functions, CA cells that have multiple values per cell, and allow computation to be distributed across a network of computers.

Internal states of computational cells

The Core system of 2009 only supported one type of cell for the entire grid of the CA. The new system supports a nearly unlimited number of types coexisting on the same grid. This is known as a 'heterogeneous world'. Different cell types follow different transition rules. During the run of a CA, each cell value at any point in time anywhere in the CA world is a double precision floating point number that can change from time to time as the simulation goes on. In Conway's well-known *Game of life*, for example, there is one cell type and every cell on the grid belongs to this cell type. There is a single property representing whether the cell is dead or alive, which can be represented using a 1 and 0 value so that a simple type of cell only has two possible internal states.

A new programming language for cellular automata

Transitions between subsequent generations of a cellular automaton are normally specified by simple rules in which predecessor states, neighbour states and successor states are explicitly listed by value. This method is characterised by its inability to specify replacements with mathematical equations, as well as the large number of replacement rules needed even for simple simulations, such as Conway's *Game of life* (see Figure 1).

Thus, the need arose for a more explicit way of expressing the rules of a CA. Team HexCore chose to implement a new domain-specific language for this purpose. Furthermore, it was designed simply, so that a new user did not need to

```
[1,1,1,0,*,0,0,0,0] => [1,1,1,0,1,0,0,0,0]
[1,1,0,1,*,0,0,0,0] => [1,1,0,1,1,0,0,0,0]
[1,1,0,0,*,1,0,0,0] => [1,1,0,0,1,1,0,0,0]
[1,1,0,0,*,0,1,0,0] => [1,1,0,0,1,0,1,0,0]
[1,1,0,0,*,0,0,1,0] => [1,1,0,0,1,0,0,1,0]
[1,1,0,0,*,0,0,0,1] => [1,1,0,0,1,0,0,0,1]
[0,1,1,1,*,0,0,0,0] => [0,1,1,1,1,0,0,0,0]
[0,1,1,0,*,1,0,0,0] => [0,1,1,0,1,1,0,0,0]
[0,1,1,0,*,0,1,0,0] => [0,1,1,0,1,0,1,0,0]
...etc
```

Figure 1: The rule that states a cell may become alive if exactly three neighbours are alive would have to be specified, in explicit fashion.

spend much time to become familiar with this language. The language allows each CA world to be associated with a rule set. In order to support heterogeneous worlds, the number of cell types must be declared and rules specified for each type in a Java-like syntax. Each cell type may also have multiple floating-point properties, which need to be specified. That means that all the details of the world's behaviour are specified in one place. Lastly, the language makes standard variables available, such as 'self' and 'neighbours', giving the user explicit access to the values of the CA cells.

The new language supports various standard constructs such as 'for loops' and 'if statements' for the computation of cell values. The extensible standard function library includes functions such as sum, max, min, sin, cos, ln, log, random, round and count. The random function makes it possible to write non-deterministic CA simulations.

Multiprocessing

The 2009 version of the system only allowed a choice to divide one CA world into one, two, four or eight segments, which were then distributed to that number of processor cores through threading. For the reimplementation, Team HexCore went a few steps further. Now all simulations are multi-threaded so as to make full use of all processor cores on a single processor system. Moreover, the user also has a choice to distribute the workload over many computers across a network (see Figure 2).

Each of the initial segments sent to a computer also contains a read-only section of neighbour values. This allows a remote computer to know the values of neighbouring cells without requesting them across the network. The algorithm intrinsically balances the load across all the computers, since it sends out work as it gets results. Computers that work faster will automatically do more work than slower computers. Due to the timeout feature in the algorithm, the system has a built-in fail-safe for client machines that collapse during a simulation – their work will simply be sent to other clients resulting in only a momentary loss of speed in the simulation (see Figure 3).

Graphical user interface

The simulation screen is designed with usability and ease of use in mind. Many control possibilities were added for the user to allow for a more immersive simulation experience. Multiple viewports can be dynamically added and removed while a simulation is running. Furthermore, the user can choose to view the simulation in 2D, 3D, with or without wireframe and in full-screen mode. Users also have the possibility to control playbacks of a simulation. The user can pause, play, reset and step forward or back one frame at a time. An interesting feature in the new system is the ability to represent cells using colour gradients instead of discrete colours. Two colours can be chosen for a bound on a property's range and the system will interpolate a gradient between those two colours.

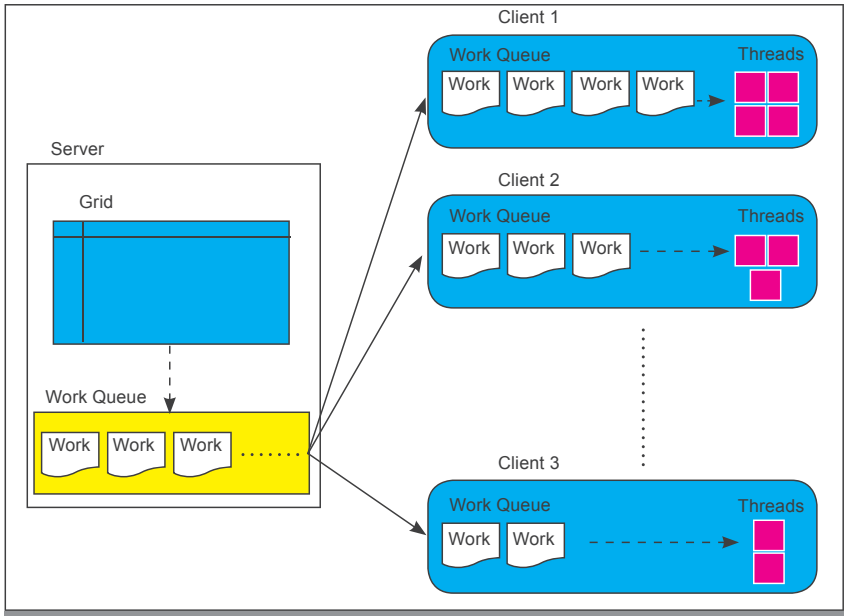

Figure 3: The graphical user interface.

```
Split world into segments related to the number of processing cores available on the network.
For each generation
   Send each computer an initial number of segments based on how many cores it has.
   Each client works on segments in parallel using its available cores.

   On results returned:
      Send more segments to that computer.
   On timeout:
      Resend segments to a different computer.
```

Figure 2: Pseudo-code to illustrate the method of running a distributed CA simulation.

Rendering the images

One of the team's goals was to improve the performance of the rendering system of the HexCore CA software. For the 3D representation, the rendering component uses vertex and index buffers, which resulted in a major performance boost compared to the 2009 system, so that now everything is much more aesthetically pleasing.

Due to the new system supporting multiple properties per cell, the team had to come up with a way to represent multiple properties in an intuitive and effective manner. The renderer can draw each property separately as a slice. A slice is a rectangular prism placed on top of each grid cell, the height and colour of which is determined by the property that it will represent. Many slices can be placed on top of each other so that multiple properties can be visualised.

In a flood simulation, for example, the first slice could be the land height property, which would result in a 3D height map terrain. The second slice could be the water depth property, which would be placed on top. This results in a realistic representation of the water level in each cell, as the land height and water depth would both contribute to the height of the cell (see Figure 4).

The resultant system is one that allows a scientist to design a CA and simulate it in an intuitive and efficient manner. Small additional features, such as the saving of world configurations, as well as the ability to export and import cellular automaton language (CAL) program code, make it easy for a scientist to collaborate with others regarding CA research. This is now made even easier, since the system is cross-platform compatible and fully tested on both Windows and Linux (with a Mac version in preparation).

The aim is to continually improve the HexCore system. It has been made available as open source software and is freely available at https://github.com/Hexcore/HxCAS.

Acknowledgements

At the School of Information Technology's Project Day at the end of 2011, the team was awarded the Grintek Ewation prize for overall software engineering excellence. The team also received the prize for the best COS 301 project at the School of Information Technology's prizegiving ceremony in April 2012.

Divan Burger, Megan Duncan, Apurva Kumar, Leon van Dyk *and* ***Karl Zöller*** *are all Computer Science students at the University of Pretoria, who worked under the supervision of Prof Stefan Gruner.*

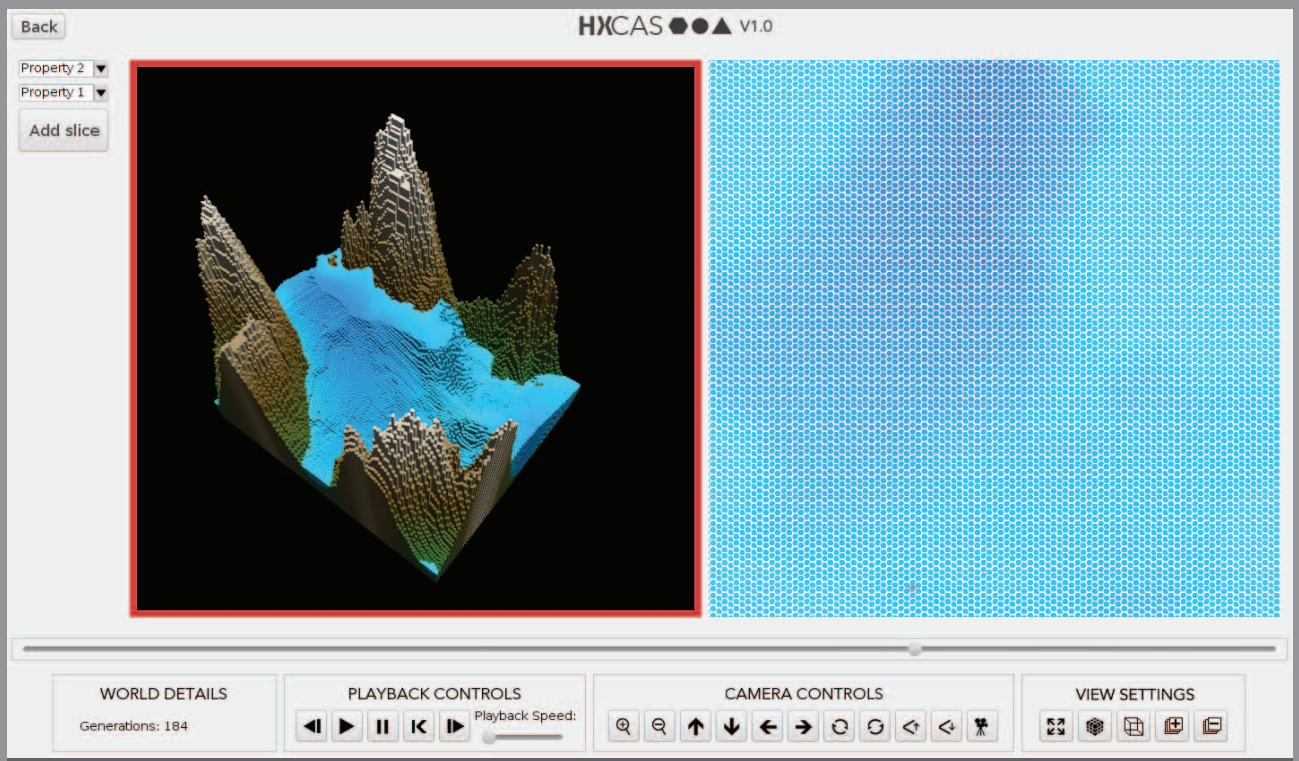

Figure 4: An example of the 3D representation.