A CELL-DEVS VISUALIZATION AND ANALYSIS PLATFORM

Bruno St-Aubin
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, Canada
bruno.staubin@carleton.ca

Omar Hesham
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, Canada
omarhesham@carleton.ca

Gabriel Wainer
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, Canada
gwainer@sce.carleton.ca

ABSTRACT

Although Cell-DEVS engines have been optimized for efficient modelling and simulation over a cell space, visualization and analysis of the results they generate remains a complex task. They are limited by the high volume of data that must be processed to identify patterns and tendencies in the model's behavior, particularly over time. In this paper, we present a lightweight, web-based visualization and analysis platform as an alternative to costly proprietary software. The Cell-DEVS Simulation WebViewer is written in HTML5 and JavaScript, requires no installation and offers a user-friendly way to post-process Cell-DEVS simulation results. It allows users to visualize and animate their simulation results, to navigate to different time steps, record videos of their simulation, inspect the state of individual cells, and export the raw data in JSON for further processing in other external programs. It leverages the data-driven document (D3) JavaScript API to provide statistical analysis capabilities in the form of animated charts that display data derived from the simulation as it is being executed.

Keywords: DEVS, Cell-DEVS, Visualization and Analysis, D3.

1 INTRODUCTION

Visualization and analysis of simulation results is normally conducted as a post-processing step, once a simulation has been successfully executed. The first step to analyze simulation results is often to simply visualize the output of the simulation. A proper visualization of the results provides a better understanding of the model's behavior and helps identify issues that were undetected in the modelling phase. Further analysis steps commonly include the use of visual statistical analysis tools such as charts and graphs to identify patterns and tendencies in the results.

DEVS software development kits (SDK) typically focus on performance or the modeling process rather than visualization and analysis tools. As can be observed in recent surveys of the field, DEVS frameworks tend to provide only summary visualization and analysis capabilities (Van Tendeloo, Vangheluwe 2017; Franceschini et al. 2014). More comprehensive exploration of results is usually relegated to third party software such as MatLab, R or Tableau. Narrowing this gap would provide greater usability and possibly contribute to a higher adoption rate of DEVS among the community of simulation and modeling users.

In this paper, a web based simulation visualization and analysis platform for cell based simulations will be presented. The Cell-DEVS WebViewer was originally developed by the Advanced Real-time Simulation Laboratory at Carleton University for academic use in courses and publications (Van Schyndel et al. 2016). It has since been subject to many improvements. It is developed entirely in HTML5 and JavaScript, requires no separate installation, can be run locally on a user's computer and minimizes external dependencies.

Those features contribute towards the project's aim of improving two key aspects of M&S:

i. Improving modeler's feedback and iteration cycle through a user-friendly portable application that uses hardware-accelerated rendering (HTML5 Canvas/WebGL) and stream-processing of large simulation data sets.

ii. Improving communication and collaboration with domain experts and stakeholders through a more interactive means of data visualization than traditional static media like images or video.

As research on modeling and simulation in the cloud continues to evolve, we wanted to build a client-side platform mindful of the potential to support cloud services in the future. Thus the choice of web technologies (HTML5/JavaScript/CSS) as the foundation for this project. This paper will discuss the features of the WebViewer as well as its data visualization and analysis capabilities.

2 BACKGROUND

2.1 Discrete Event System Specification Overview

Although reviewing the DEVS method in detail is beyond the scope of this paper, a brief overview will provide insight on how it made the development of a web based visualization and analysis platform possible. Discrete Event Systems Specifications (DEVS) is a well-established technique for efficiently modeling real-life systems that was first described by Bernard Zeigler in 1976 (Zeigler, Praehofer, Kim 1976). It provides a discrete event based method to abstract systems into models that can be used for experimentation in cases where it is impractical or impossible to experiment on the real system. DEVS supports hierarchical and modular model development, favoring the reusability of models. It defines a rigorous formalism to manage inputs and outputs of atomic models, the basic blocks of DEVS. This facilitates their integration into larger, more complex coupled models. It has been demonstrated that the DEVS formalism can be used as a common denominator for any formal method of modeling, whether discrete or continuous (Vangheluwe 2000).

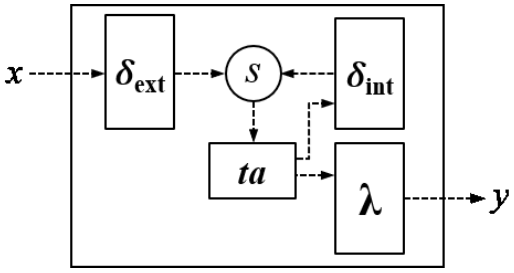

Figure 1: DEVS Atomic model definition.

Briefly put, DEVS atomic models are event-driven models with state transitions occurring for two types of events: scheduled (internal) events, and received messages (external) events. Every atomic model has a persistent state s. When it receives an input x, the external state transition function δ_{ext} is triggered and the state is updated. The time advance ta function, which specifies the time to the upcoming internal transition δ_{int}, is dependent on the new state. At this point, the atomic model will wait until either it receives another external input, or the time advance function expires, whichever occurs first. In the latter case, the output function λ, also dependent on the current state, will issue an output value y before triggering an internal state transition function δ_{int}. In turn, the internal transition function will update the state s and a new ta is

calculated. The model will execute this loop until it passivates (i.e. *ta* is set to infinity). Without any further external inputs, the model will remain passive. It is possible to integrate multiple atomic models into a larger composite model, known as a coupled model. The integration is achieved by simply coupling the inputs and output ports of one model to the appropriate input or output ports of other models.

Since the state in a DEVS model is updated in an event-driven fashion, processing cycles are not wasted on uneventful time periods of the simulation. This is in contrast to discrete-time simulation where the simulation is advanced in discrete timesteps regardless of the underlying state changes. Thus DEVS generates the minimum amount of output data required for visualization and analysis of all state updates.

2.2 Cell-DEVS Overview

Cell-DEVS is an extension of DEVS that can be used for modeling and simulation of systems in a cell space. It is a combination of DEVS and cellular automata with explicit timing delays (Wainer 2009). Each cell acts as an individual atomic model and the cell space becomes a coupled model where all the cells are linked to their neighbors through input and output ports, as in regular DEVS models. A cell-DEVS model can be described by the conceptual diagram in Figure 2.

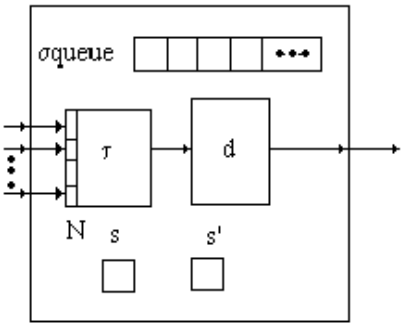

Figure 2: Cell-DEVS atomic model definition (with transport delay).

Each cell receives N inputs, usually from neighboring cells but they can also be provided by a regular DEVS model. When a cell receives these inputs, it triggers τ, the local computing function that determines the next state, s′. At this moment, if the cell's future state s′ differs from its current state s, it will schedule an output of its new state following a transport delay specified by d. Whenever a cell changes state, its new state s′ and scheduled time for transition are added to a local queue. Cells with transport delays will always output their new state, provided a state change has occurred. If a cell's state does not change following the local computing function τ, it becomes passive and waits for further external events.

Cells can also be implemented using inertial delays which allow preemption of the cell's state transition. A cell with inertial delay that does not manage to keep its new state until the next scheduled time elapses is preempted and foregoes its output phase for the preempted state transition. The definition of a cell with inertial delay is slightly different than the one described by Figure 2, interested readers should refer to (Wainer 2009) for a more thorough explanation.

3 THE PROPOSED WEB BASED CELL-DEVS WEBVIEWER

The Cell-DEVS WebViewer is a minimalistic client-side web application built entirely in HTML5 and JavaScript. It requires only two external dependencies, Whammy.js to record videos of canvas animations in .webm format and the Data-Driven Documents (D3) library for dynamic charts. In this section, the user interface, the data structures and the general flow of the application will be discussed.

The WebViewer was originally built as an alternative to the desktop-only simulation viewer that was included in the CD++ development environment (Chidisiuc, Wainer 2007). It is meant to offer a better loading process for the simulation results, more control over the visualization and more user-friendly

visualization of results. A screen capture of the full WebViewer is shown in the appendix to this paper (See Appendix A). This paper does not contain any code samples of the WebViewer. Instead, the complete code for the WebViewer is made available at https://github.com/SimulationEverywhere/CD-WebViewer/ .

3.1 Loading Simulation Result Data

The first step to visualize a CD++ Cell-DEVS simulation is to load its output log files through a *FileInput* widget (See 1 in Appendix A). When the user clicks on any of the 4 boxes, a standard file dialog will appear from which the user can bulk upload the 4 files that are required to view a CD++ simulation: the model file (*.ma*), the initial values file (*.val*), the log file (*.log*) and the color palette file (*.pal*). The model file (*.ma*) is a text file that contains the simulation configuration parameters required to visualize the results. Notably, input and output ports, grid size and, in some cases initial values are read from that file. They are parsed from the *.ma* file using regular expressions. The initial values file (*.val*) contains the cell-space state for time step 0. The *.val* file replaces the initial values parameter in the *.ma* file, only one of them is required. The color palette (*.pal*) file defines the colors associated to each cell state for cell-space rendering.

The *.log* file contains all the transitions that were made by each cell in the cell-space for the duration of the simulation. A single ASCII line is logged and timestamped for each output message, state transition, and simulator substep in CD++ (e.g. "0 / L / Y / 00:00:29:600:0 / Cell(24,23)(625) / out / 30.0 / Cell (01)" indicates an output of value 30 from cell (24, 23) at 29s:600ms). It is the largest of the 4 files and as such, requires special consideration when parsing and storing in memory. For example, in the use case presented in section 4, the output log reached 162 MB, for a cell space of 100 X 100 X 2 and 600 time steps. Compared to other CD++ simulations, this one generates a small output due to its relatively small grid and short duration. Depending on the simulation parameters, CD++ log files can reach a size of 50 GB.

Simply reading the file was the first challenge. It is impossible to directly load such large files in a string, even on 64-bit systems, as we are constrained by browser-imposed memory limits (handful of GBs at best). Hence, the log has to be streamed in chunks. Prior to HTML5, this would have been impractical if not impossible to achieve using a web based platform. Indeed, for security reasons implemented by browsers, files had to first be sent to a server then returned to the user either as is or after being parsed and transformed by a service. The bandwidth and transfer time required for such a large file would have made the WebViewer nearly impossible to use. Fortunately, significant and recent advances in web technologies brought solutions to these issues. Particularly, the HTML5 specification was adopted in 2008 by the World Wide Web Consortium (W3C) and started to see wider adoption (W3C 2017). Among HTML5's features, the *FileReader* object allows browsers to read local files with byte-wise precision. We took advantage of this by reading the CD++ log files in smaller chunks of ~2MB each, processing each chunk into a more optimized internal data structure, then proceeding to the next chunk. This process repeated until the file has been read entirely.

As chunks of the *.log* file are being read, they must also be parsed and stored into memory. Considering the simulation case study in section 4, if the entire cell-space state was stored in memory, it would require 20,000 number variables for each time step. JavaScript has only one format for numbers and it requires 64 bits of memory. For 600 time steps, approximately 600 X 20 000 X 64 bits of memory would be required to store each cell-space state at each time step. This adds up to 384,000,000 bits or approximately 0.09 gigabyte of memory for a 162 MB file. Considering that *.log* files can reach over 300 times that size, storing the entire cell-space state at each time step becomes unrealistic, the memory requirements would exceed the capacity of standard computers. Furthermore, traditional compression techniques (analogous to video encoding) would not be suitable as they would slow down read/write access to individual cell data. To work around this issue, only the event-driven state changes are stored. When the user launches the visualization playback, they are applied iteratively and the cell-space state is rendered in the canvas after each iteration.

3.2 Simulation Playback

Visualizing a Cell-Devs simulation is a straightforward process. Starting from the values of the cell-space at time 0, the transitions for the next time step in the simulation log file are applied simultaneously. Then, the cell-space is rendered to the screen. This process is repeated until the end of the log file is reached. However, since the visualization is reconstructed from the transitions, navigating freely through the simulation requires special consideration. When the user skips ahead or navigates back using the slider, many time-steps may need to be applied to the current cell-space state to obtain the time-step requested by the user. Depending on the duration of simulation, the size of the cell-space and the number of transitions to apply, this can introduce significant delays. To work around this issue, as it reads the *.log* file, the WebViewer caches the entire cell-space once every 10 time-steps. Then, when the user skips ahead or back, the nearest cached frame is found and only the transitions from the cached frame to the requested time frame need to be applied. This measure was implemented to improve the user experience.

Once the transitions have been read from the *.log* file and the cache built, the initial cell-space is drawn using the initial values. Earlier versions of the viewer used HTML divs and Scalable Vector Graphics (SVG) elements to render the cell grid. This worked well initially, but did not efficiently scale for more complex models involving multiple layers with many ports each. It became prohibitive to use Document Object Model (DOM) elements such as divs and SVG objects for rendering due to their computational and memory costs. Thus, later developments employed the HTML5 Canvas instead (See 2 in Appendix A). Using this approach meant that the cell grid remained a static HTML element, avoiding the overhead of DOM tree manipulation. With finer control over the rendering logic, the Canvas element is well-suited for scenes with a very large number of elements that need to be drawn, removed or redrawn frequently (MSDN 2017). Furthermore, the Canvas element exposes the WebGL API for hardware-accelerated 2D and 3D rendering, and it is compatible with third-party libraries such as *Whammy* (Kwok 2016) used here to record videos directly from the canvas.

Once the initial cell-space is rendered, the WebViewer is ready to playback the reconstructed simulation using the playback widget (See 3 in Appendix A). The user can control the speed of playback (Framerate), navigate to specific time-steps.

The Cell-DEVS WebViewer also provides statistical analysis capabilities to the user in the form of charts. The first one is a bar chart that shows the number of cells in each state for a given time frame of the simulation. The second one is a heatmap that shows the number of transitions that were executed for each cell at a given time step of the simulation. Both charts were implemented using the Data-Driven-Documents (D3) API built by Mike Bostock around 2010. It is currently one of the most widespread JavaScript data visualization library on the web and serves as a base for other higher-level data visualization libraries that are derived from it. D3 provides a series of tools which allows developers to easily inspect, manipulate and transform a native representation of the web, the document object model (DOM). It provides an easy way to bind data to document elements (Bostock, Ogievetsky, Heer 2011).

4 A USE CASE FOR THE CELL-DEVS WEBVIEWER: THE LOGISTIC URBAN GROWTH MODEL

Logistic urban growth (LUG) is a model used to simulate the urbanization of space based on the logistic equation advanced by Pierre-François Verhulst in the early 19th century (Verhulst 1845; Brauer, Castillo-Chàvez 2001). It stipulates that population growth, unimpeded by external factors, will follow an exponential curve that will diminish as population grows and completely stop at population maturity. The equation has been implemented successfully in many fields including chemistry, biology, ecology, demography, economics, etc. For a given cell i,j, the probability of transition from a non-urbanized to an urbanized state can be computed as follows:

$$P_{ij}^{t} = \frac{1}{1 + e^{-(a_0 + \sum_{n=1}^{m} a_n x_{n,ij})}} \times \frac{\sum_{x,y=1}^{x,y=n} con(S_{xy} = urban)}{n^2 - 1} \times Bin(cell_{ij} \neq restricted) \times (1 + (-\ln r)^a) \quad (1)$$

Where

a_0	A constant
$a_1, a_2 \ldots a_m$	Regression coefficient factors for distance to geographic features
$x_{n,ij}$	Normalized distance to geographic feature
con	A condition that returns 1 if the x,y cell is urbanized.
Bin	A binary function that returns 1 if the cell is not restricted
r	Random real number between 0 and 1
a	A parameter (0 to 10) that adjusts the effect of the stochastic factor

This section presents results of a LUG simulation using the Cell-Devs WebViewer. The cell-space used for the simulation is a map with a simple spatial configuration. Using real data such as Google Maps images or any other map providers would have required complex and extensive image analysis before being able to use the data in a Cell-DEVS simulation. Although that is possible, the work required would have been well beyond the scope of this paper. The simplified map provided enough context to adequately study the capability of the WebViewer to view and analyze simulation results on a cell-space.

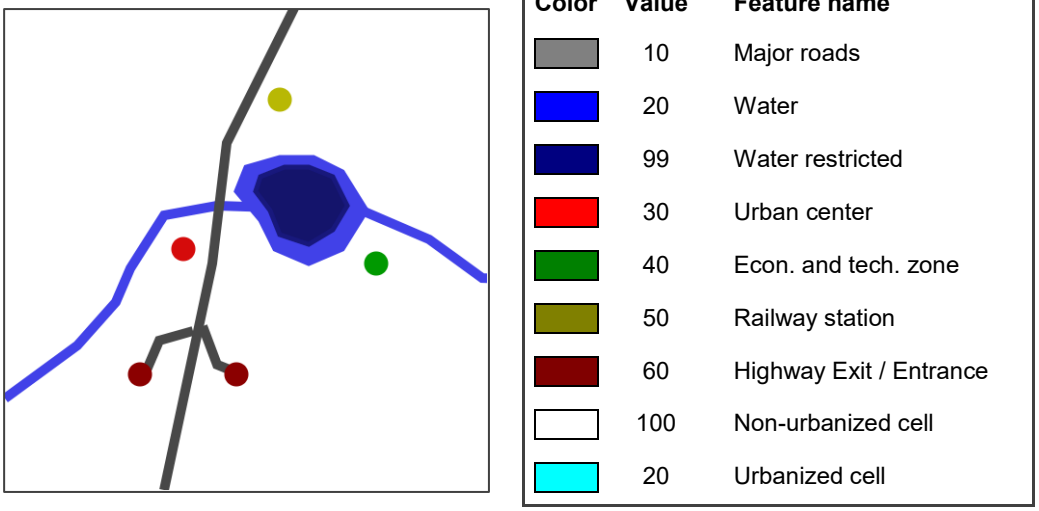

Figure 3: Sample of geographic maps used with a legend detailing the values associated to each color.

4.1 Rendering the Simulation

To render the simulation in the canvas element, the first step is to draw the cell-space for the initial values that are stored in the transitions for time step 0. Looping through the X, Y and Z dimensions, an equal width and height square is drawn for each cell and filled with the color that corresponds to its state as specified in the palette file. For subsequent time-steps, only the cells that with a state transition are redrawn according to the color palette. The canvas API provides all the necessary functions to achieve this. Though the concept is easy to understand, it is a very meticulous process to implement since the canvas renders using a raster bitmap. This requires that the developer continuously calculate the X, Y positions for the cells drawn. SVG graphics would have likely been more convenient to work with since each cell becomes its own DOM entity. With a handle on the DOM node, updating the color of the element requires only changing the fill attribute value. Alas, using DOM nodes instead of canvas elements would limit the viewer's ability to use the canvas-only *Whammy* video capture library, and limit future 3D (WebGL) visualizaiton opportunities.

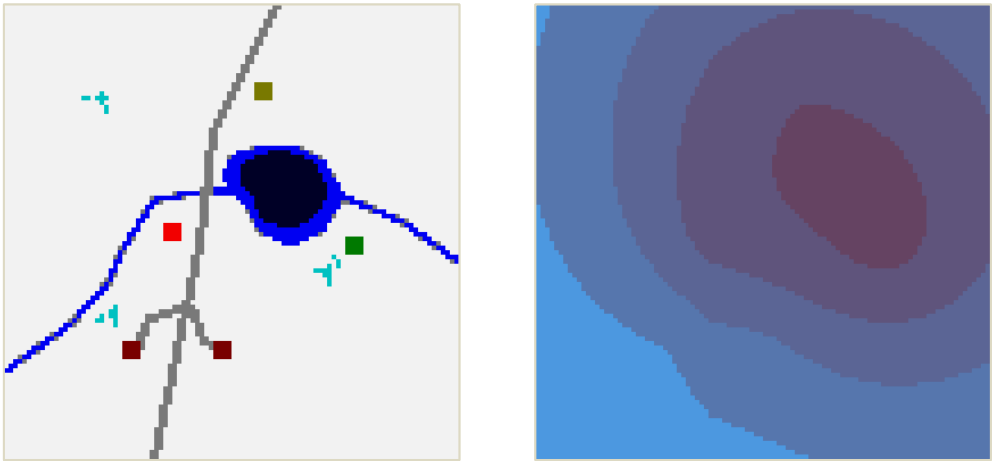

Figure 4: The initial values as rendered in the canvas. The left image is the layer that contains the cell-space state. The right image is the layer that contains the P_{di} factor (time-invariant).

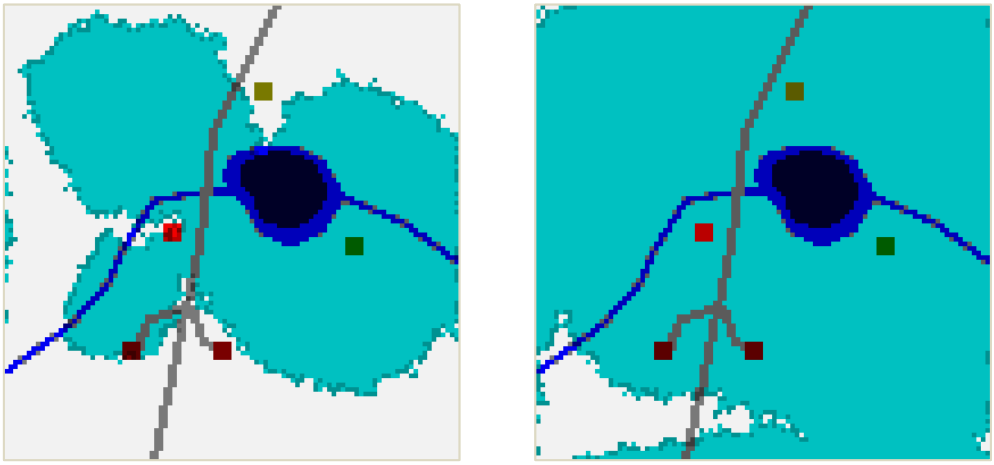

Figure 5 : The same simulation as Figure 4 at different time-steps. Left image is time-step 200, right image is time-step 400. As time progresses, more and more cells are drawn as urbanized.

As can be seen from figures 4 and 5, as time progresses in the simulation, more and more cells become urbanized, as expected from the LUG model presented in section 4. Furthermore, the urbanization pattern follows the P_{di} layer, also as the model intended. Other factors can be confirmed visually from the animated version. For example, it is easy to notice the stochastic disturbance (eq 1) when visualizing the results from two simulations run with the same parameters (i.e, same regression coefficients, same spatial configuration, same restricted area, etc.). The urbanization process will vary and the final state of the cell-space will be different from one set of results to the other. Visual representation of the simulation is the first step in confirming that the model behaves as intended. In depth data analysis using statistical tools is another method that can help a user confirm that a model is appropriately implemented.

4.2 Data Analysis using D3

Currently, three data analysis tools have been integrated to the WebViewer, two that rely on Data-Driven Documents (D3) to provide graphic charts and one that shows general statistics for the simulation. Each of these tools are navigation enabled, meaning that they are updated as the user navigates across time-frames using the slider or the fast forward and rewind buttons. The data required by these widgets is derived from

the current cell-space state and the transitions data used to reconstruct it. The derived data must be computed at each time-step which requires many operations. Further work on this feature should include optimization.

The functions required to convert the data into a format that the statistics widgets can consume were coded in a separate static class to decouple them from the data and make them easier to reuse. This pattern is very loosely inspired by the Model-View-Controller pattern. The resulting data is also stored as a class variable so the results are persistent and only need to be incremented when the user proceeds to the next time frame. This way, less processing is required to go from one time-step to another.

The graphic chart developed (Figure 7) is a bar chart that shows the total number of cells for each state in the cell-space at any given time in the simulation. The data required to fill the chart is obtained from the current view buffer of the WebViewer. The current view buffer is the object that contains the entire cell-space for the current time-step. It is basically a grid of dimensions X, Y, Z where each of the cell has a number value equal to its state. Note that the statistical analysis tools currently operate only on one layer and the user can select the layer from the *LayerControl* widget. To obtain the total number of cells for each possible state, a nested loop through the X, Y dimensions for the selected Z layer suffices. For each iteration, a variable corresponding to the total number of cells for that cell's state is incremented. Also note that the user can control which states he wishes to track for this chart and he can specify a range of states to be tracked at once. The latter feature was implemented to be able to follow states in simulations where the cell's state is an interval value rather than nominal.

The bar chart complements the grid visualization and provides the exact total of cells for each state at a given time-step. This can help confirm characteristics of the model and compare results for different simulations. For example, considering two simulations that are identically configured except for having different P_{di} factors in layer 2:

(a) (b)

Figure 6 : Two possible P_{di} layer configurations (right) that share the same initial urban map (left).

We can see that the bar charts at different time-steps, presented in Figure 7, show that the simulation of configuration (b) transitions at a much faster rate than (a). Although this information can already be gleaned visually through the canvas element, the bar chart provides exact numbers. These observations are in line with the model since higher P_{di} factors will make it easier for each cell to meet the probability threshold of the LUG equation (eq 1). We can easily see from the P_{di} layers above that the alternative (b) has a higher average value than alternative (a). The bar chart could also help to study the impact of the other factors in the LUG model such as the value of the threshold for the LUG equation or the spatial configuration of the map and the initial clusters of urban cells.

The second analysis chart developed for the Cell-DEVS Simulation WebViewer is a heatmap chart that shows, for every cell in the cell-space, the total number of transitions accomplished. The data required to fill the chart is obtained from the transitions data by adding up the transitions made by each cell for the whole cell-space from time-step 0 to the current time-step. To avoid unnecessarily computing these values from time-step 0 at each iteration, the total transitions for the current time-step are stored as a class variable

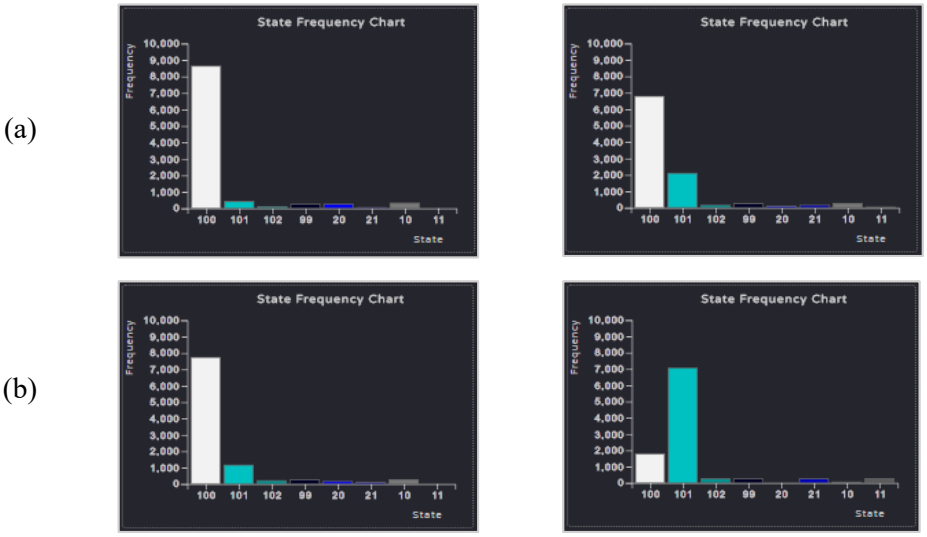

Figure 7: Bar chart comparing simulation statistics of different P_{di} layer configurations.

along with the current time-step. When the WebViewer proceeds to the next time-step, only one set of transitions must be added to the total transitions. When the user skips or goes back multiple frames, all the sets of transitions between the previous time-step and the new time-step must be applied. This process could benefit from using cached data, much like the cell-space rendering does for the view buffers that contain the state of the cell-space.

The transition heatmap (Figure 8) can help a modeler determine whether cells are transitioning as expected or to identify areas where more interesting transition patterns occur. In the case of the LUG simulation presented in section 4, it confirms that the P_{di} layer has the intended effect on the simulation and that the restricted area cells (value 99) prevent any transitions. Considering the same initial values presented in Figure 6, the transitions heatmaps for various time-steps are presented in Figure 8.

Many observations about the LUG simulation can be made by analyzing the heatmaps in Figure 8. A first and obvious one is that the *Bin* factor in the LUG model works as intended. Indeed, the area of cells with value 99 (i.e deep lake) have seen no transitions across the 600 time-steps of both simulations. This is expected since the *Bin* function returns 0 when the current cell is restricted, thus resulting in a 0 probability of transition which will never surpass the threshold. The impact of the P_{di} factor is also clearly observable. In both cases, unurbanized cells far from attracting features have much more difficulty converting to an urbanized state. In scenario (a), the top-left cluster is dark red indicating that most cells in the cluster have alternated between the urban and growing state but that their neighborhood did not urbanize efficiently. Some cells have even reached 600 transitions, the maximum number possible, meaning that they have alternated for the duration of the simulation without settling on the urban state since their neighborhood was never fully urbanized. In simulation scenario (b), the cell-space never reaches full urbanization but there are no cases where a cell has alternated for the entire duration of the simulation. However, it is obvious that as the urbanized area expands outwards, away from the attracting factors, urbanization becomes more difficult. The further the urban cluster is from the attracting factors, the darker the heatmap becomes.

The last tool added to the Cell-DEVS WebViewer is a general statistics widget (See 4 in Appendix A). It shows the number of cells with a given state at each time-step as well as the average, mean, median and standard deviation for the number of transitions made in the entire simulation. The transitions statistics are calculated using built-in functions in D3. The states statistics are updated as the simulation runs but the other values are not because it was too processor intensive and it slowed down the playback rate significantly. For both cases, the data shown is derived from the same data that is used by both charts.

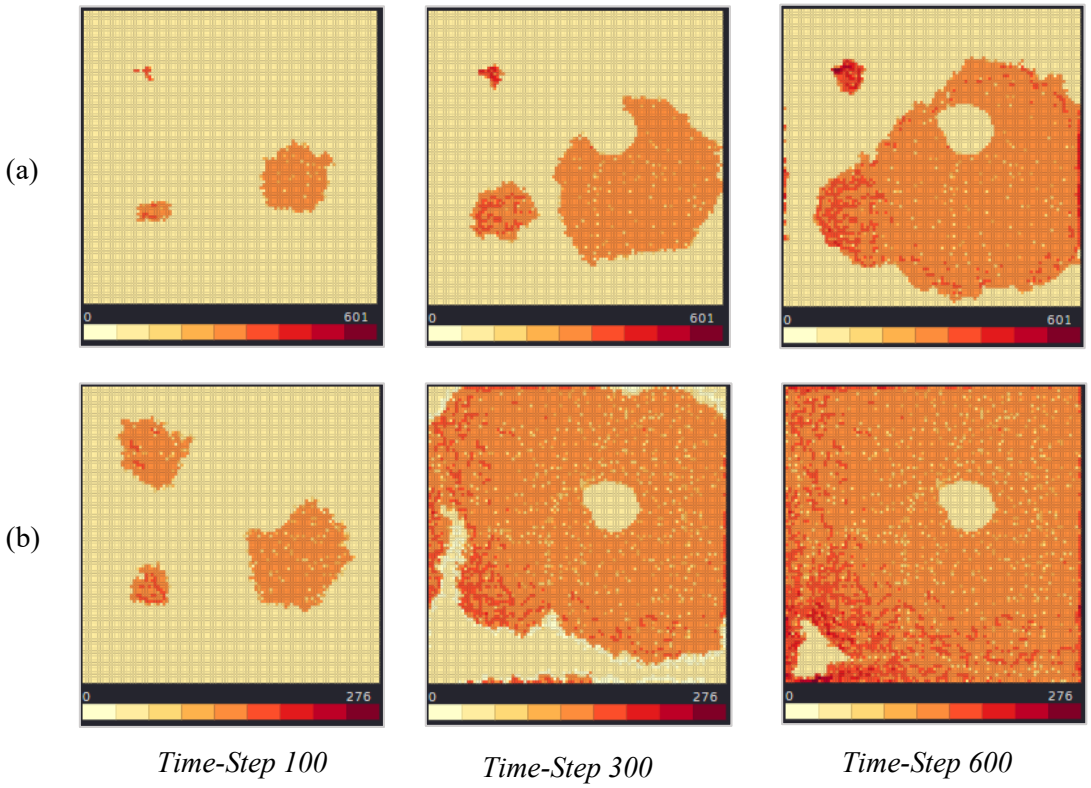

Figure 8 : Transition heatmap illustrating the impact of different P_{di} configurations.

5 CONCLUSION

In this paper, a web-based platform to visualize and analyze results of a CD++ simulation was presented. This platform can be hosted on a server, making it easy to access, but is run entirely on the client side. This eliminates the need to install desktop software, makes it easier to push updates to users and minimizes dependencies on other technologies. Even though it is a web application, it shows good performance due to recent developments in HTML5 and JavaScript. The HTML5 *FileReader* object allows the user to quickly load a very large *.log* file, parse it into chunks and load it in memory for the WebViewer to render, animate and analyze.

Future work on the web viewer should focus on the development of additional data analysis and visualization tools. For example, a cell inspector function could allow the user to study the transitions within a single cell. Such a function could also be expanded to study a range of cells representing an area of interest. Another avenue is supporting a wider variety of charting options. Providing users with a dashboard like application where they can configure their visualization tools according to their needs would provide significant insight into their simulation in a user-friendly manner. Beyond visualization, we also look forward to apply what we learned about handling event-driven simulation data to allow for efficient in-browser modeling and simulation, further streamlining a modeler's workflow and improving the modeling and simulation process, in alignment with our goals as stated in the introduction section.

Data visualization tools focused on analyzing simulation results are infrequent. Dedicated simulation analysis software could increase the productivity of users reducing the need for custom file parsing and data charting solutions. A good visual analysis tool offers deeper insights into the simulation, the capability to identify previously undetected errors in the model and to detect patterns in the state transition process. Developing user-friendly, dedicated analysis tools, is also a way to make modeling and simulation accessible to a wider public.

A APPENDIX

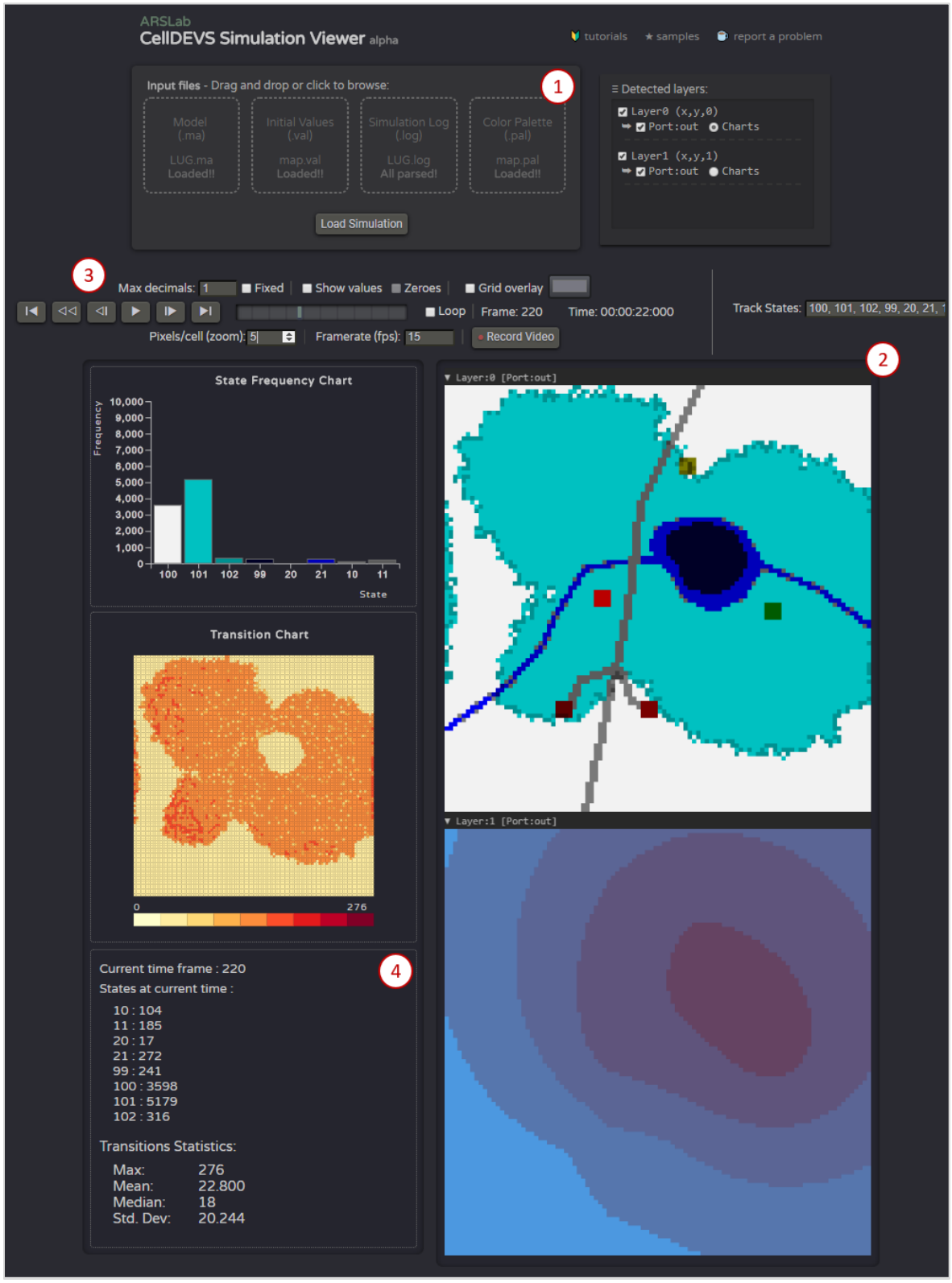

Figure 9 : The Cell-DEVS Simulation Viewer user interface.

REFERENCES

Bostock M., Ogievetsky V., Heer J., 2011, D3: "Data-Driven Documents", IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis).

Brauer F., Castillo-Chàvez C., 2001, "Mathematical Models in Population Biology and Epidemiology", Texts in Applied Mathematics, Vol. 40, Springer, 416 p., ISBN : 0-387-98902-1.

Chidisiuc C., Wainer G.A., 2007, "CD++ Builder: An Eclipse-based IDE for DEVS modeling", Proceedings of the SpringSim '07 simulation multiconference, Vol 2., 235-240.

Feng Y., 2017, "Modeling Dynamic Urban Land-Use Change with Geographical Cellular Automata and Generalized Pattern Search-Optimized Rules", International Journal of Geographical Information Science, Vol. 31, No. 6, Taylor & Francis Group, pp. 1198-1219 .

Franceschini, R., Bisgambiglia, P-A., Touraille, L., Hill, D., 2014, "A Survey of modelling and simulation software frameworks using Discrete Event System Specification", OpenAccess Series in Informatics, Vol. 43, 40-49

Kevin Kwok, "Whammy", https://github.com/antimatter15/whammy, consulted on January 7, 2016.

Microsoft Developer Network (MSDN), 2017, "SVG vs Canvas: How to Choose, https://msdn.microsoft.com/en-us/library/gg193983(v=vs.85).aspx, consulted on Dec 16, 2017.

Vangheluwe, H.L.M., 2000, "DEVS as a Common Denominator for Multi-formalism Hybrid Systems Modelling", IEEE International Symposium on Computer-Aided Control System Design, IEEE.

Van Schyndel M., Hesham O., Wainer G., 2016. "Crowd Modeling in the Sun Life Building", Proceedings of the Symposium on Simulation for Architecture & Urban Design (SimAUD '16). SCS.

Van Tendeloo, Y., Vagheluwe, H., 2017, "An evaluation of DEVS simulation tools", Simulation: Transactions of the Society for Modeling and Simulation International, Vol. 93(2), 102-121

Verhulst P-F. , 1845, « Recherches mathématiques sur la loi d'accroissement de la population », Nouveaux Mémoires de l'Académie Royale des Sciences et Belles-Lettres de Bruxelles.

Wainer, G.A, 2009, "Discrete-event modeling and simulation: a practitioner's approach", CRC Press, Taylor & Francis Group, 485 p.

White R., Engelen G. , 1993, "Cellular Automata and Fractal Urban Form: A Cellular Modelling Approach to the Evolution of Urban Land-Use Patterns", Environment and Planning, Col. 25, pp. 1175-1199.

World Wide Web Consortium (W3C), 2017, "File Reader API", W3C Working Draft, 26 October 2017, consulted on Dec 18, 2017.

World Wide Web Consortium (W3C), 2017, "HTML 5.2 W3C Recommendation – History", https://www.w3.org/TR/html5/index.html, consulted on Dec 16, 2017.

Zeigler, B.P., Praehofer, H., Kim T.G., 1976, "Theory of Modeling and Simulation", New-York: Wiley-Interscience.

AUTHOR BIOGRAPHIES

BRUNO ST-AUBIN. Is pursuing a PhD in Electrical and Computer Engineering at Carleton University where he researches web based simulation and visualization. His email address is bruno.staubin@carleton.ca.

OMAR HESHAM. is pursuing a PhD in Electrical and Computer Engineering at Carleton University, where he researches agent-based simulation and visualization. More info at https://omarhesham.com.

GABRIEL WAINER is a Professor at the Department of Systems and Computer Engineering at Carleton University. He is a Fellow of the Society for Modeling and Simulation International (SCS). His email address is gwainer@sce.carleton.ca.