Literature Survey

Robbie Jakob-Whitworth, L3 Natural Sciences

Introduction

Conway's Game of Life was first popularised after appearing in a Scientific American article (Gardner 1970) and is commonly described as a zero-player game. Its evolution is determined by its initial state, with no further input required. A player interacts with the Game by creating an initial configuration and allowing the simulation to run, observing how the state evolves.

The Game starts with a two-dimensional infinite grid of cells, where each cell can be alive or dead. The living cells comprise the current population. On each iteration of the Game, the following simple genetic laws (McIntosh 2010) apply:

1. Survival – every cell with exactly two or three neighbouring live cells survives for the next iteration.
2. Death – every cell with four or more alive neighbours dies due to overpopulation. Every cell with one or zero live neighbours dies from isolation.
3. Birth – each dead cell with exactly three live neighbours becomes alive.

Existing Game of Life Visualisations

There exist already a number of visualisations of the Game of Life. Having been popularised for close to 50 years, these range from implementations using modern web technology to Python visualisations or even older ones.

A simple, elegant visualisation (Weldon & Laventure 2018) was submitted as a project to a hackathon, HackUVic 2018. This web-based visualisation uses a 100x100-cell grid and is rather rudimentary in its approach (which is to be expected, given that it is a hackathon project). All computation happens on the client-side, with 10,000 nodes in the DOM comprising the grid, which is not particular performant when the iteration speed is set to the minimum option, 100 seconds.

A more complete and polished visualisation (Bettilyon 2018) is also a client-side, web-based visualisation. However, it utilises the modern HTML5 canvas and, as such, is able to achieve a vastly more responsive visualisation than that of Weldon and Laventure.

Another popular visualisation (Vanderplas 2013) uses Python and NumPy to compute iterations for the Game of Life. It then passes the computed results to the author's own package, JSAnimation, which is responsible for visualising the computed results on a webpage. This approach allows complex populations to be rendered smoothly and promptly as no computation takes place during render-time.

Arguably the most well-known and respected visualisation is the application Golly (Trevorrow & Rokicki 2006), which has native applications for Windows, macOS, Linux, Android and iOS. It contains support for many different types of cellular automaton rules, as well as implementations of many different algorithms for simulating against those rules. Moreover, Golly contains scripting support for Lua or Python, with some common scripts enabling it to produce animated GIFs of patterns.

While there are a number of web-based implementations of Game of Life visualisations, these tend to be fairly basic implementations with little consideration given to the performance or suitability of a particular visualisation. Typically, all computation occurs client-side, leaving the browser to do complex calculations for large grids.

Computing New Iterations

While the rules of the Game of Life are simple, it is shown that small, finite patterns can produce emergent structures of great complexity, given enough iterations (Gotts 2010). The popular “glider” structure uses just five live cells and a bounding box of 3x3 (Sapin 2010) but translates itself diagonally across the board infinitely. Moreover, the “glider gun” (36 live cells, contained within a 36x9 bounding box) produces gliders, which as they themselves move, have the effect of growing the non-empty part of the board, potentially with no upper bound.

It follows that, over time, each iteration of the simulation could increase in complexity. The aforementioned Golly application implements an algorithm called Hashlife (Trevorrow & Rokicki 2006). Hashlife is a memoised algorithm which computes the long-term outcome of a given starting configuration in Conway’s Game of Life (and related cellular automata) and is known for doing so with great speed and comparatively small computational load.

Memoization (Abelson, Sussman & Sussman 1997), also called tabulation, refers to storing previously computed values in a lookup table. When a memoised procedure computes a value, it first checks the table to see if the value is already computed, otherwise, it computes the value in the ordinary way and stores it in the lookup table for future use. This approach is used in Hashlife by splitting the grid into subpatterns. For example, a commonly reused pattern (like the aforementioned “glider gun”, or large regions of empty space) can be hashed and mapped to the same position in the lookup table, meaning many copies of the same subpattern can be stored just once using the same hash table entry (Gosper 1984).

Web-Based Client-Server Communication

The most basic architecture for retrieving data from a server using the web is through a simple page load. The user requests a web page, and the server responds with some (static) contents (Berners-Lee & Cailliau 1990). After the server responds, additional data must be requested by requesting a new web page. In this architecture, the client must request some data from the server in order for the server to respond; the server is unable to “push” data directly to the client without the client having requested it. While it’s technically possible to

have a web page periodically self-reload, the delay is too great for many applications, and decreasing the delay would lead to unnecessarily many network requests. In the context of a Game of Life visualisation, this approach would be a very poor user experience as the page would have to reload for every iteration!

This “stateless” architecture is dated and much of the modern web now heavily utilises Ajax (Zepeda & Chapa 2007). This technology enables a client to request additional data from a server without having to reload the entire web page. Google Maps, for example, heavily relies on Ajax to dynamically load new regions of the map as the user pans around the region, without having to reload the page. Such a technique could be used for a Game of Life visualisation if the server were to, say, send blocks of computed iterations in response to a client making successive requests. However, Ajax still faces the limitation that the client must request data from the server in order for the server to respond.

One previously popular model that attempts to circumvent this limitation is Comet, also known as Ajax 2.0 (Crane & McCarthy 2008). This utilises a “long-polling” technique. Because the client must initiate a connection to the server, the server is unable to send data directly to the client without the client having first requested it. “Comet” itself is an umbrella term (Russell 2006), encompassing several techniques in which a long-held HTTP(S) request allows a web server to push data to a browser (the client), without the browser having explicitly requested it. The client must make the first initial Ajax request, but the server does not respond until it wishes to. This allows a server to effectively push data to a client only when it needs to, and for a long time, this was considered the best way to send data asynchronously to a browser (Kachhwaha & Patni 2012). For example, Facebook Messenger used a variant of this approach when it first launched (Letuchy 2008). Implementing Comet, however, is non-trivial and can make scalability difficult (Mesbah & van Deursen 2008).

WebSocket (Puranik, Feiock & Hill 2013) is a modern standard that seeks to address these issues. Long-polling itself can be memory and network intensive and is an anti-pattern to how Ajax was originally designed. Unlike HTTP, WebSocket provides full-duplex communication, enabling a server to send a stream of data to the client with greatly reduced overhead and significant performance improvements (Lubbers & Greco n.d.). It is ideal for situations where a web server needs to continuously send data to a browser – perfect for constantly sending new states and iterations of a Game of Life for a browser to then visualise.

References

Abelson, H., Sussman, G.J. and Sussman, J. (1997) Structure and interpretation of computer programs, (second edition). *Computers & Mathematics with Applications*. [Online] 33 (4), 133. Available at: doi:10.1016/S0898-1221(97)90051-1.

Berners-Lee, T. and Cailliau, R. (1990) *WorldWideWeb: Proposal for a HyperText Project*. [Online]. 12 November 1990. Available at: https://www.w3.org/Proposal.html (Accessed: 31 October 2019).

Bettilyon, T.E. (2018) *The Game Of Life (Simulation)*. [Online]. 2018. Available at: https://tebs-game-of-life.com/single-large/single-large.html (Accessed: 21 October 2019).

Crane, D. and McCarthy, P. (2008) *Comet and Reverse Ajax: The Next-Generation Ajax 2.0*. Berkely, CA, USA, Apress.

Gardner, M. (1970) Mathematical Games. *Scientific American*. 223 (4), 120–123.

Gosper, R.Wm. (1984) Exploiting regularities in large cellular spaces. *Physica D: Nonlinear Phenomena*. [Online] 10 (1), 75–80. Available at: doi:10.1016/0167-2789(84)90251-3.

Gotts, N. (2010) Emergent Complexity in Conway's Game of Life. In: Andrew Adamatzky (ed.). *Game of Life Cellular Automata*. [Online]. London, Springer London. pp. 389–436. Available at: doi:10.1007/978-1-84996-217-9_20 (Accessed: 20 October 2019).

Kachhwaha, R. and Patni, P. (2012) Ajax Enabled Web Application Model with Comet Programming. *International Journal of Engineering and Technology*. 2 (7), 7.

Letuchy, E. (2008) *Facebook Chat*. [Online]. 14 May 2008. Facebook. Available at: https://www.facebook.com/note.php?note_id=14218138919 (Accessed: 31 October 2019).

Lubbers, P. and Greco, F. (n.d.) *HTML5 WebSocket - A Quantum Leap in Scalability for the Web*. [Online]. Available at: http://www.websocket.org/quantum.html (Accessed: 31 October 2019).

McIntosh, H.V. (2010) Conway's Life. In: Andrew Adamatzky (ed.). *Game of Life Cellular Automata*. [Online]. London, Springer London. pp. 17–33. Available at: doi:10.1007/978-1-84996-217-9_3 (Accessed: 20 October 2019).

Mesbah, A. and van Deursen, A. (2008) A component- and push-based architectural style for ajax applications. *Journal of Systems and Software*. [Online] 81 (12), 2194–2209. Available at: doi:10.1016/j.jss.2008.04.005.

Puranik, D.G., Feiock, D.C. and Hill, J.H. (2013) Real-Time Monitoring using AJAX and WebSockets. In: *2013 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*. [Online]. April 2013 pp. 110–118. Available at: doi:10.1109/ECBS.2013.10.

Russell, A. (2006) Comet: Low Latency Data for the Browser *Infrequently Noted*. [Online]. Available at: https://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/ (Accessed: 31 October 2019).

Sapin, E. (2010) Gliders and Glider Guns Discovery in Cellular Automata. In: Andrew Adamatzky (ed.). *Game of Life Cellular Automata*. [Online]. London, Springer London. pp. 135–165. Available at: doi:10.1007/978-1-84996-217-9_9 (Accessed: 20 October 2019).

Trevorrow, A. and Rokicki, T. (2006) *Golly Game of Life Home Page*. [Online]. 18 June 2006. Available at: http://golly.sourceforge.net/ (Accessed: 31 October 2019).

Vanderplas, J. (2013) *Conway's Game of Life in Python | Pythonic Perambulations*. [Online]. 7 August 2013. Available at: https://jakevdp.github.io/blog/2013/08/07/conways-game-of-life/ (Accessed: 31 October 2019).

Weldon, J. and Laventure, A. (2018) *Conway's Game of Life Visualization*. [Online]. 11 February 2018. Devpost. Available at: http://devpost.com/software/game-of-life-tkn5zw

(Accessed: 31 October 2019).

Zepeda, J.S. and Chapa, S.V. (2007) From Desktop Applications Towards Ajax Web Applications. In: *2007 4th International Conference on Electrical and Electronics Engineering*. [Online]. September 2007 pp. 193–196. Available at: doi:10.1109/ICEEE.2007.4345005.