Playful Programming: Higher Order Design as Shaping Emergence – A life-like work in progress

Luke Church, Alan F. Blackwell

Computer Laboratory
University of Cambridge
luke@church.name, Alan.Blackwell@cl.cam.ac.uk

Keywords: POP-II.A. Neat / Scruffy, POP-II.B. Program comprehension, POP-III.B. Game of Life, POP-IV.A. Exploratory

Abstract

We consider approaches to the design of software, comparing rational design with the exploration of emergent behaviour. We discuss the challenges of attribution of behaviour to code, the formulation of patterns within code and some of the social processes that affect such systems.

Introduction: Rational Design vs. Emergence

Is playing with Conway's Game of Life anything like programming in C#? Trivially, they are both capable of emulating a Turing machine. However, in terms of what their users are trying to achieve, they are apparently nothing like each other. C# is typically used for Windows application development, whereas the Game of Life is usually played for entertainment, or exploring the behaviour of cellular automata.

This difference is reflected in the way in which programmers express their desired behaviour. C# and Visual Studio, as with most commercially used languages, encourages 'rational design'. The programmer is encouraged to think very hard about what they want to happen and describe it to the computer. The manner of the description influences the programming experience. For example, the structuring of C# (and Visual Studio) encourages a decomposition of the program into objects, as opposed to aspects.

The Game of Life[1] is somewhat different. It was explicitly designed to result in interesting behaviour from a starting condition. The user is encouraged to explore the interaction space and see what develops. Perhaps then going back and changing the starting conditions, or even modifying the program during execution. Despite one being able to express the rules of the Game of Life in a paragraph, it can result in enormously complex behaviour. This phenomenon, the generation of complexity from simple rules, is often referred to as emergence.

However, these two behavioural extremes don't really tell the whole story. Some C# programmers go about their work in a more life-like manner than might be expected. For example, performance issues are fixed by monitoring how long things take and poking them, rather than formally understanding everything and optimising it. However, emergence isn't without criticism either. Downie (2005) is strongly critical of emergence in computer art, describing it as an 'anti-methodology'. His position is that emergence is trivial to generate, the difficulty is in constraining it into a form where it can be used to generate 'interesting' behaviour.

[1] We're using Golly (Trevorrow and Rokicki, 2007) as our reference implementation

We suggest that end user programming to date has tended to align itself with one of these extremes. Exploring the middle ground seems an interesting problem, however there are many difficulties, not least of which, is how would users understand the programs they were creating?

Comprehension by Attribution

As mentioned above, formal consideration is not the only way of understanding a system. Consider playing a game like Command and Conquer 3 (CnC). CnC is characterised by players building and placing structures, which are used to 'harvest' resources, as well as allowing the construction of military units. The units are commanded in real time, via mouse and keyboard commands, to manoeuvre and/or engage in combat with opponents. Strategies emerge out of compound sets operations of these operations. (EA, 2007)

Few of the players of CnC could literally describe the rules that the game engine executes, and yet most manage successful play. The 'meaningful play literature' (e.g. Salen (2003), Koster, (2005)) stresses the importance of players being able to attribute their success or failures to their actions within the game. This is often not easy to achieve in games like CnC where success or failure is usually determined by the player's strategy, not their micro-actions. We believe that this process of attribution is crucial to end user programming, and an area that would benefit from further study.

Prosaic Patterns

Game play strategies look like a program of interactions that the player is expressing to the computer. Where do such strategies come from? Much of the behaviour is highly sophisticated, involving deciding how to allocate resource well ahead of time, executing multiple tactical operations in parallel, predicting points of conflict and responses from adversaries. All of this is the kind of problem we'd like to be able to solve in 'serious' programming languages.

Characteristic forms of behaviour seem to emerge within games. Long term strategies in CnC have even been given names, such as 'steamroller'. We argue that such prosaisms also occur in languages used for professional programming where they tend to be referred to as 'design patterns' (Gamma et al, 1995). Such patterns seem to be something of a double edged sword, they are generally unwelcome in computer based art, and may limit how engaging the interaction is in other domains. However in professional software development tools the response of the vendor is typically to encapsulate the patterns themselves into a new framework, such as the Microsoft Presentation Foundations navigation support system. How might we support such behaviour in end user programming tools?

Metadesign

The SER (Seeding, Evolutionary growth, Reseeding) process (Fisher and Ostwald, 2002) is a model for exactly how clichés might be integrated into the platform. It advocates a process where a domain is 'seeded' with a platform that allows end-user innovation. This is then allowed to grow organically, to determine what kinds of features are actually desired before being reseeded with support for these features.

We can see this kind of behaviour in games like CnC, where the previously mentioned user strategies have been developed into AI strategies in later versions. However, relationships are by no means always this happy. CnC 3 has gone through rapid patching cycles, tweaking the strengths of units to prevent 'degenerate strategies' emerging in which one side can almost always guarantee success. Downie's suggestion is to use an online machine learning algorithm to automatically stabilise the system when such behaviour emerges: how would users' ability to comprehend what is occurring cope with such online adaptation? Further, such behaviours are discovered only through very

widespread play-testing (millions of hours worth). How might the developers of the game have predicted and shaped the behaviour of the players without this very extensive testing?

One possibility is through the use of technological toys, as described by Turner et al. (2005). Turner stresses the importance of programmers who are working with artists creating technological toys for the artists to play with in order to help the developer become attuned with the artist, and to help the artist understand the possibilities offered by computational artefacts. Another possibility is through the development of Formal Abstract Design Tools (FADT) (Church, 1999), as advocated by the game design community. FADTs are an attempt to form a common vocabulary of the properties of interaction that are common across many games. Building such a vocabulary might help us to choose particular strategies for shaping emergence. Further work is being undertaken to analyse these possibilities.

Conclusion

We have outlined the beginnings of an agenda to understand how to build computational models and their associated user interfaces that provide novel experiences through emergent behaviour, as opposed to requiring the user to precisely specify what the system is intended to do up front. We have identified a number of challenges with this approach and drawn on examples from the computer-art and computer games literature. Finally, we have determined some questions, which we will expand upon in further work.

Acknowledgements

Luke's work is supported by the Eastman Kodak Company.

References

Church, D. (1999) Formal Abstract Design Tools. http://www.gamasutra.com/features/19990716/design_tools_01.htm, accessed 03/01/08

Downie, M. (2005) Choreographing the Extended Agent: performance graphics for dance theater, Doctoral thesis, MIT Media lab

EA, (2007) Command & Conquer Battlecast. http://www.commandandconquer.com/ accessed 13/01/08

Fischer, G. and Ostwald, J. (2002) Seeding, Evolutionary Growth, and Reseeding: Enriching Participatory Design with Informed Participation, Proceedings of the Participatory Design Conference (PDC'02), Malmö University, Sweden, 135-143.

Gamma, E. Helm, R. Johnson, R and Vlissides, J. (1995) Design Patterns: Elements of Reusable Object-Orientated Software, Addison-Weslet, Reading, Mass

Koster, R. (2005) A Theory of Fun for Game Design, Paraglyph Inc.

Salen, K. (2003) Rules of Play: Game Design Fundamentals, MIT Press

Trevorrow, A., Rokicki, T. (2007) Golly, a Game of Life Simulator, http://golly.sourceforge.net accessed 13/01/08

Turner, G., Weakley, A. Zhang, Y. and Edmonds, E. (2005) Attuning: A Social and Technical Study of Artist-Programmer Collaborations, In Proceedings of the 17th Annual Workshop on the Psychology of Programming Interest Group (PPIG 2005), 106-119.