Teaching architecture students to code

Thrills and spills

Tim Ireland[1]
[1] *Kent School of Architecture*
[1] *t.ireland@kent.ac.uk*

This paper will present the introduction of computer programming for design to students at the Leicester School of Architecture (LSA). It will describe the course and teachings, explain the trials and tribulations, and illustrate the results. An important weight on students of architecture, when it comes to the inclusion of coding into their architectural education, is the pressure of meeting certain professional criteria. The MArch Architecure course results in a professional level award that is prescribed by the ARB, and accredited by the RIBA for Part II exemption from their examinations. Consequently, students are required to articulate through their design work that they have met the learning outcomes associated with the stipulated professional criteria. Given the task of meeting the learning outcomes is challenging enough, the pressure of then learning to code, and to apply that skill to the design process in the course of the traditional process is a pressure few students of architecture seem willing to take on. The paper will conclude with a discussion as to the merits of coding and reason why students of architecture should learn to code.

Keywords: *Programming, Code, Processing, Teaching, Architectural Education*

INTRODUCTION

This paper will present the introduction of computer programming for design to students at the Leicester School of Architecture (LSA). It will describe the course and teachings, explain the trials and tribulations, and illustrate the results. The course has been taught for three years, commencing as part of a specialised stream of the MA Architectural Design course (2014-15), before becoming a component of the MArch Architecture course in 2015-16, whereby the course has become a module adjunct to the design studio; promoting algorithmic design thinking in correlation to traditional and contemporary methods of production and design thinking. An important weight on students of architecture, when it comes to the inclusion of coding into their architectural education, is the pressure of meeting certain professional criteria. The MArch Architecture course results in a professional level award that is prescribed by the ARB, and accredited by the RIBA for Part II exemption from their examinations. Consequently students are required to articulate through their design work that they have met the learning outcomes associated with the stipulated professional criteria. See the QAA

Subject Benchmark Statement for Architecture (QAA 2010). Given the task of meeting the learning outcomes is challenging enough, the pressure of then learning to code, and to apply that skill to the design process in the course of meeting these criteria is a pressure few students of architecture seem willing to take on. This is understandable because learning to programme is a challenge that requires a mindset quite different from the traditional design studio (see Vannini 2016), and why would someone wish to struggle with connecting two random points to draw a line when they can quite easily do it with a pen and paper; and thereby focus on developing and articulating their design proposal. The paper will conclude with a discussion as to the merits of coding and reason why students of architecture should learn to code.

BACKGROUND

Computer coding was introduced to students at LSA in the academic year 2014-15. It was offered as an option to learn coding for the purpose of using the computer as a tool to simulate parallelism and thereby generate architectural scenarios through a decentralised process of production. Until that time the manner in which computation was utilised in the school was quite traditional. Whilst it was incorporated and applied in the design process the computer was typically used for 3D modelling, graphics and visualisation. Rhino, the favoured modelling software, was, and remains to be, encouraged and students are adept in its use through studio tutorials, workshops and self-directed learning. Whilst workshops in Grasshopper are delivered students tend to adhere to 3D modelling software as a means to produce representations of their designs, rather than use the capacities of such tools to enable them to explore and form find: or as Ranulph Glanville would say to produce "models of, as opposed to models for". A specialist studio was introduced in the MArch Architecture course this academic year (2016-17), concerned with morphogenesis and parametric modelling techniques (led by Gudjon Thor Erlendsson), so this trend is changing and LSA students are becoming more computer savvy. The school emphasises the craft of making, and promotes, through studio teaching, craft-oriented and digital production techniques. Its mix of analogue and digital fabrication techniques, with a strong design-led ethos has yielded a number of student awards, putting the school in company with internationally renowned leading schools of architecture. (See note 1). The intention behind introducing students to algorithmic design thinking and teaching them how to code is to promote the generation of architectural propositions through decentralised methods of pattern formation and form production (see Coates 2010), so as to (a) open up their way of thinking about architecture and the world and (b) experiment and test the potential of the computer as a tool for design.

IMPLEMENTATION

Coding is taught as a short course, composed of several lessons, introducing up to 10 students with only basic computer skills and no coding experience to programming. Processing is the programming language taught. By the end of the course they are able to produce their own agent-based-modelling programs. To date no students have failed the module, and the majority of students achieve a merit or higher: due in large to the format and premise that the students are required to write their own program that generates some spatial composition. The nature of code is such that no mistakes can be made, else the program does nothing. So, so long as a student is able to write code and produce a short program that generates some spatial formation they are unlikely to fail the module. This is not to say it is an easy course. The majority of students experience a breakdown (see note 2) until their mind-set clicks: after all programming is a way of thinking. This short course has evolved over three years. It was implemented at first as part of a specialist computing stream to students on the MA course, during the academic year 2014-15, before it migrated the following year to the MArch Architecture course, as a specialist technology module. As part of the MA Architectural Design course the students coding ability became more advanced (compared with the following year when the course transferred to the MArch) and the manner in which they took to coding was more literate. Having learnt from the MA-MArch transition the course was adapted and the relevance to design studio revised for the current year (2016-17). The coding module took place in the first term of the MArch course; completed before the Christmas break. This differs from what was taught when the course was part of the MA. The MA students where first taught (in the Autumn term) NetLogo and given theory lectures introducing them to key concepts: self-organisation, emergence, decentralisation and agency. They then took the Processing course in the Spring term (after the Christmas break). So the MA students were better prepared for programming, as they were clued up: in terms of having both a theoretical grounding and a basic understanding of code before commencing with the Processing programming module.

COURSE OUTLINE

The course starts with an introduction to the basics of computer programming, before going through a series of lessons to teach the Processing programming language, agent-based modelling and Object-Oriented Programming (OOP). The intention is that the students write a short OOP programme which generates 3-dimensional architectural spatial formations. Asked to investigate simple organisms, students are required to model a simple artificial organism that responds to differences in its environment which can be utilised to generate architectural spatial compositions.

Essentially students are taught coding through a series of weekly workshop oriented seminars, during which they are introduced to key concepts and taken through a series of exercises. Each of these workshop-seminars concludes with a task, requiring the student to produce their own processing sketch to illustrate they have learnt the principles presented in each lesson. This is also a mechanism prompting them to work through the lesson independently, and to urge them to complete the tasks as the course progresses because the module concludes with the above mentioned assignment: to produce an object-oriented agent-based program, whereby a cyber-organism of their devising generates a pattern, structure or form that has spatial qualities and is an emergent outcome of their agent's autonomous interaction with its environment. This agent-based modelling assignment is the peak of this short course which they present in the form of a report, including their outputs for the weekly tasks. The report thus illustrates their progression through the course, articulating their understanding and development. Consequently, as the weekly tasks form the basis of this report, students are encouraged to complete each task promptly, not only to ensure they are absorbing the lessons but also as a scheduling mechanism to provide them the time to focus on the concluding assignment.

Each workshop-seminar starts with a confab whereby students present their previous weeks task output, explaining their code and what it does. The group discusses the output, problems highlighted and successes are congratulated. Given each task must, in actuality be completed for the submission of the report, students are able to take on comments to revise and rework their weekly task codes. Students are strongly encouraged to review the lesson independently, and that an hour a day of coding is necessary, else the code-muscle weakens and each week will quickly become groundhog like; and it quickly becomes apparent to the students that the workload snowballs if they do not keep abreast of the developing syllabus. Students are also encouraged to meet up to go through lessons and exercises together, either with a code-buddy or as a group to prompt discourse about the tasks, and problems encountered; to counter them getting trapped by internal processes that prohibit seeing the wood for the trees. The confab and workshop-seminar environment promotes an inner-circle atmosphere and students tend to work openly with each other, supporting one another in the various glitches they encounter. At the end of the session the week's task is revealed and example output(s) presented.

The course runs for five weeks. Starting with the fundamentals of programming students learn how to write short programs that create dynamic patterns and then, having grasped the fundamentals of coding, they move to interaction and movement before learning about agents and finally object-oriented programming (OOP). The course outline presented below is as delivered this academic year (2016/17). It is a reincarnation of a course originally authored by Alasdair Turner, which the author took in 2009 at the Bartlett School of Graduate Studies. The course has been altered and adapted over the three years it has been taught at LSA; in response to the students, the authors intentions and how it aligned with the design studio module. An underlying aspect of the course is that it has been shoehorned into existing programme structures. As part of the MA it was offered as an aspect of the design studio module, whilst under the MArch it is offered as a specialist option in the History, Theory & Criticism module, which has both a humanities side and a technology side. In both circumstances the timetable did not allow for the course - at least not to provoke and prompt the students to go from zero to agent-based modelling coder in five weeks. The two-hour weekly seminar timetabled for five weeks was insufficient. The course is an elective and consequently, it was stressed at the outset of week 1 that the enthusiastic students had drawn the short straw and that a deal was to be struck if they were willing to "feel the pain". To achieve said transition the author was willing to put in the time if the students were, and proposed the weekly sessions took as long as needed: on the basis that he was keen to share and teach coding skills with the students, it would not only be fun, but that it was a new skill and would open the students up to another way of doing things, with the view that it could feed into their studio work: as well as being part of a pioneering way of doing architecture at LSA. Over the three years all students have voted to take the plunge, signing up to extended weekly sessions that generally last three-and-a-half to four hours.

Week 1. The first week is a crash course in the fundamentals of programming before explaining repetition and variation. The lesson starts with a short presentation and discussion regarding "Why Code?", before covering fundamentals of 'functions' and 'data', the Processing co-ordinate system, and an explanation of 'loops' and 'conditions'. Using colour and co-ordinates as variables students are taught how to control data. The first week's task is twofold, requiring one sketch that produces a composition of points and lines, and another that produces colourful patterns. See Figure 1.

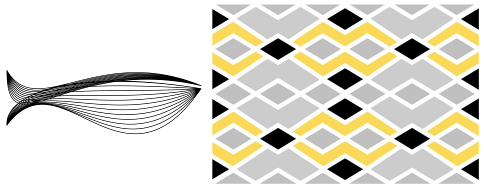


Figure 1
Week 1 sketches by MArch student Ka Leong (2015/16) that produces a composition of points and lines (left) and another (right) that produces a colourful pattern.

Week 2. The following week is about movement and change. Having learnt how to draw a shape primitively (opposed to using, say the "rect" command, to do so) in week 1, students are shown how to write their own function, enabling them to replicate their shapes more easily: i.e., less code. They then study how to manipulate these shapes through loops, altering local and global variables so that they shift and rotate. The output is to produce a sketch that generates a dynamic artwork. See Figure 2.

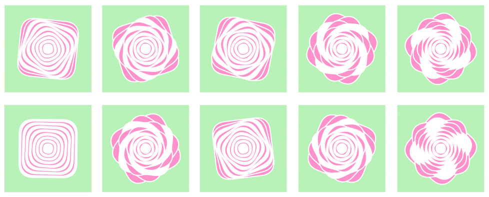


Figure 2
Week 2 sketch by MArch student Ka Leong (2015/16) that generates a dynamic artwork

Week 3. Students are shown how to stop and start shapes rotating use an event according to mouse clicks and key presses, so that their sketches become interactive, before they are introduced to classes. The concept of grouping information into routines and sub-routines is emphasised so that they are able to organise their short programmes, grouping certain bits of data and functions together, to enable them to add and interact with shapes/objects that have different behaviours and respond differently to external events: like the press of the keyboard or click of the mouse to change direction. The aim being that they view their coded objects as characters that "know" what and where they are, and so respond accordingly to an event. They are required to write a short program in which an "object" has behaviour and responds to mouse/keyboard interaction. See Figure 3.

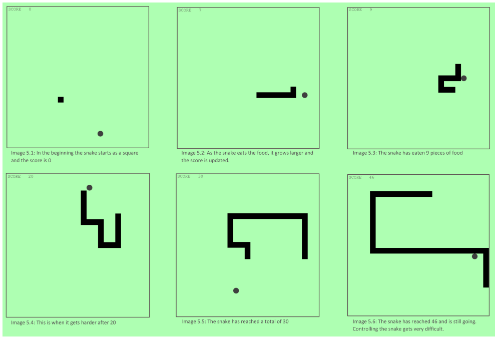

SCORE 0
SCORE 7
SCORE 9
SCORE 20
SCORE 30
SCORE 46
Image 5.1: In the beginning the snake starts as a square and the score is 0
Image 5.2: As the snake eats the food, it grows larger and the score is updated.
Image 5.3: The snake has eaten 9 pieces of food
Image 5.4: This is when it gets harder after 20
Image 5.5: The snake has reached a total of 30
Image 5.6: The snake has reached 46 and is still going. Controlling the snake gets very difficult.

Figure 3
Week 3 task by 2016/17 student Aashiv Shah who created a simple two dimensional game, making his own version of the classic game 'SNAKE'.

Week 4. This lesson introduces arrays. The notion of arrays is introduced, starting with fixed length arrays, first for lists of integers, and then for classes. Variable length arrays are then introduced so that they can add an arbitrary number of objects, which they can manipulate and interact with, before arrayLists are introduced. They are given a short sketch of variable 2D objects with behaviour which they are required to modify, amending the objects behaviour and transforming the sketch into 3D. They are introduced to libraries, at the end of the lesson so that they can either independently convert the sketch into 3D or use a plug-in: such as peasyCam.

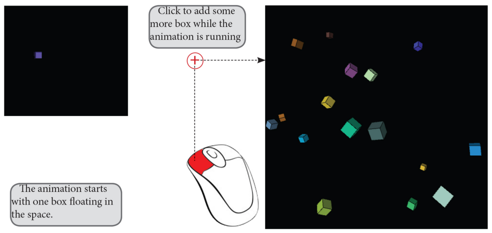

Click to add some more box while the animation is running
The animation starts with one box floating in the space.

Figure 4
Week 4 sketch by Florian Mouafo (MArch 2016/17) in which spinning boxes that are attracted to/repelled by other boxes are added at the click of the mouse, to create an asteroid field.

Week 5. Students are shown how to manipulate and interact with their objects personally (with the click of the mouse) and with one another. They are introduced to the idea of agents, and agency to extend their coding to agent-based modelling. The object-oriented programming concept of composition is introduced to present the technique for creating more complex objects, and they study how we define an agent. The notion of an agent and agency crosses many disciplines, but the focus is on the computational notion of an agent; because that's what the course is concerned with. Jacob von Uexkull's concept of the "functional Cycle" (1957) is introduced to highlight an agent must be able to detect its environment in order to react to it. A taxonomy of "agent" is presented (see Franklin and Graesser (1996), and Ingham (1997)) with the conclusion that an agent is situated and exhibits autonomous and cogent behaviour with respect to its environment and to each other. The definition presented by Franklin and Graesser (1996) is adopted as a working definition, that: "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." Uexküll's functional cycle is thus used as a model to emphasise detector-effector interplay, explaining the congruence between an organism (agent) and its environment.

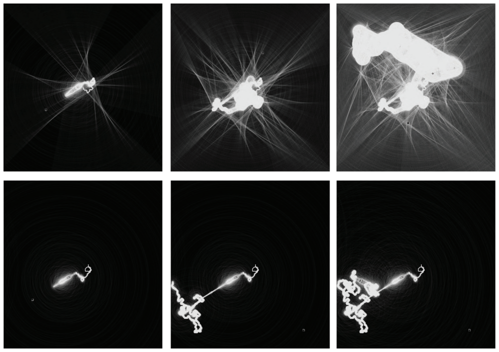


Figure 5
Two spider-agents; one that spins a web as it moves, thereby enhancing its sensorial domain, the other with just a field of view. The spider with the web (top) lasts longer and explores the majority of its environment.

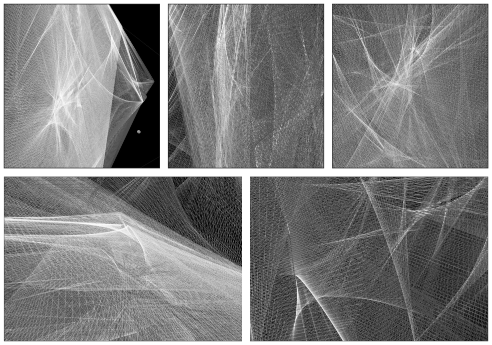


Figure 6
And transcribed into 3D .

Looking at a simple ant inspired agent model we look first at a primitive method of movement using Moores Neighbourhood as a method of governing movement of a random walker: akin to how a turtle moves from patch to patch in NetLogo. We then convert the method of walking to using PVectors. The lesson closes with a look at the object-oriented programming concept of composite objects and how an agent can interact with its environment. Students are then given the code for a random walking agent sketch and tasked to alter the agent's behaviour to generate an alternative output arising from the agent's interaction with its environment. They are also issued with the concluding assignment to write a programme, for which they are required to characterise an agent, which creates something through interaction with its environment. By detecting differences in its environment which it reacts to, their autonomous agent must alter or affect its environment through its motion (shaping its environment as it moves) or by brute-force (moving and changing elements in its environment). The aim is that their cyber-organism produces some pattern, structure or form which has spatial qualities and is an emergent outcome of its autonomous interaction with its environment. See figures 5 and 6, by 2016/17 MArch student Michael Roden, who, interested in how spiders sense vibrations, explored how the web acts like an amplifier enhancing spiders' capacity to detect, and replicated the spiders coupling with its web to locate food and survive.

CORRELATION WITH THE DESIGN STUDIO

The coding course outlined above was for the first two years adjoined to the design studio. In its first year (MA students 2014/15) the course was integral to the design studio. With its transition to the MArch in 2015/16 students, who signed up to the authors design studio, were compelled to sign-up to the coding course, and in its third year (MArch 2016/17) the course was detached from the design studio; as the emphasis to design architectural projects through code appeared daunting for MArch students; who are compelled to respond to the constraints of the Part 2 architectural curriculum (QAA 2010). The MA is not vetted by the RIBA and so the design studio is at liberty to experiment and explore, with the luxury of being able to promote a more theoretical discourse given it is not constrained by vocational targets. During its application as part of the MA the course was submerged into the studio. The design studio commenced with Processing lessons, and so it drove the design thinking and process. The final output was thus not a report but a design project. The MA students were thus not bound to producing outputs each and every week; though in large they did. It meant that the learning was more fluid and embedded in a design discussion. Interestingly MA students seemed to struggle a little initially with some of the more advanced concepts (such as arrays and classes), even though they had the benefit of having worked with NetLogo in the previous term: during which they were introduced to key principles (self-organisation, emergence, distributed representation) and "played" with agents; so they had a heads-up. This was only appreciated in hindsight, once the course was delivered to MArch students (2015/16), who seemed to lap up the initial weeks of the course, producing dynamic art sketches more impressive than their previous counterparts. However, having "twigged" the MA students use of Processing as a tool for their design project became ubiquitous and their coding projects surpassed the course content. The MA students design projects were driven by a code mind-set, and so their design projects were motivated by their coding abilities. They explored methods, utilised libraries and overcame technical problems independently: like how to enable an agent to recognise a 3D plane so that it changes direction and doesn't walk through a wall. The (2015/16) MArch students found the transition from using Processing as a drawing tool (to generate artsy dynamic outputs) to producing models that generate some sort of architectural/spatial composition through agent interaction more difficult. This is likely down to the fact that the MA cohort (2014/15) had more time and where not constrained by vocational criteria. The course they followed and the final output was the same but the format and schedule differed. The Processing course outlined above was delivered in MA studio time (for which a day was available!) and bled directly into the studio project which was to model a paramecium (i.e., build a 2D artificial creature that responded to differences in its environment), which they used to generate two-dimensional spatial compositions and developed into three-dimensions, to generate architectural compositions that express structure, space and form. (See Figure 7: top). The MArch students had the same brief as the MA students, but seemed to freeze when it came to using processing to drive their architectural projects, as they struggled to "let loose" and perceive, or consider, the output of their processing model as a basis for architectural thinking and a vehicle for designing. Consequently, the transition between their Processing work and their design project became moot: more an interesting exercise than an embedded aspect that informed their initial design process. (See Figure 7: bottom).

This third year the above Processing course was decoupled from the design studio, and students wishing to use their new coding skills were given the flexibility to test and try, whilst ensuring they progressed their design work by other methods in tandem. The result of this decoupling was that the MArch students appeared to relax; with regards the need to drive their design studio projects through Processing. The emphasis in the design studio was instead placed on algorithmic thinking: i.e., their design projects were driven by a rule based process (to emphasise results derived from cause and effect "systems"), which they were welcome to implement through analogue methods if they wished. The result of this decoupling was that the students were less troubled with regards the design studio and were more focused on the Processing course. The detachment seemed to allow them the space to focus on both individually and not worry about the coupling. Consequently, a few of the students freely utilised Processing in the early stages of their design studio projects to drive their conceptual thinking, and as an aid to generate outputs, which they exported into Rhino for further development. (See figure 8). Whilst few actually developed this further, to generate their final output, the fact that they freely utilised code in their design process was encouraging and suggests that there is an interest in using the computer in a more creative manner (i.e., "models for"), but that the vocational constraints and schedule, discourage architecture students because the vocational constraints pressure them to thinking more traditionally - or using more tried and tested methods. This is perhaps why (digital) architecture schools promote computational design through Masters courses as opposed through the vocational Part 1 and 2 (or equivalent courses). Whilst one might say this works, its a deterrent to experimenting with the machine

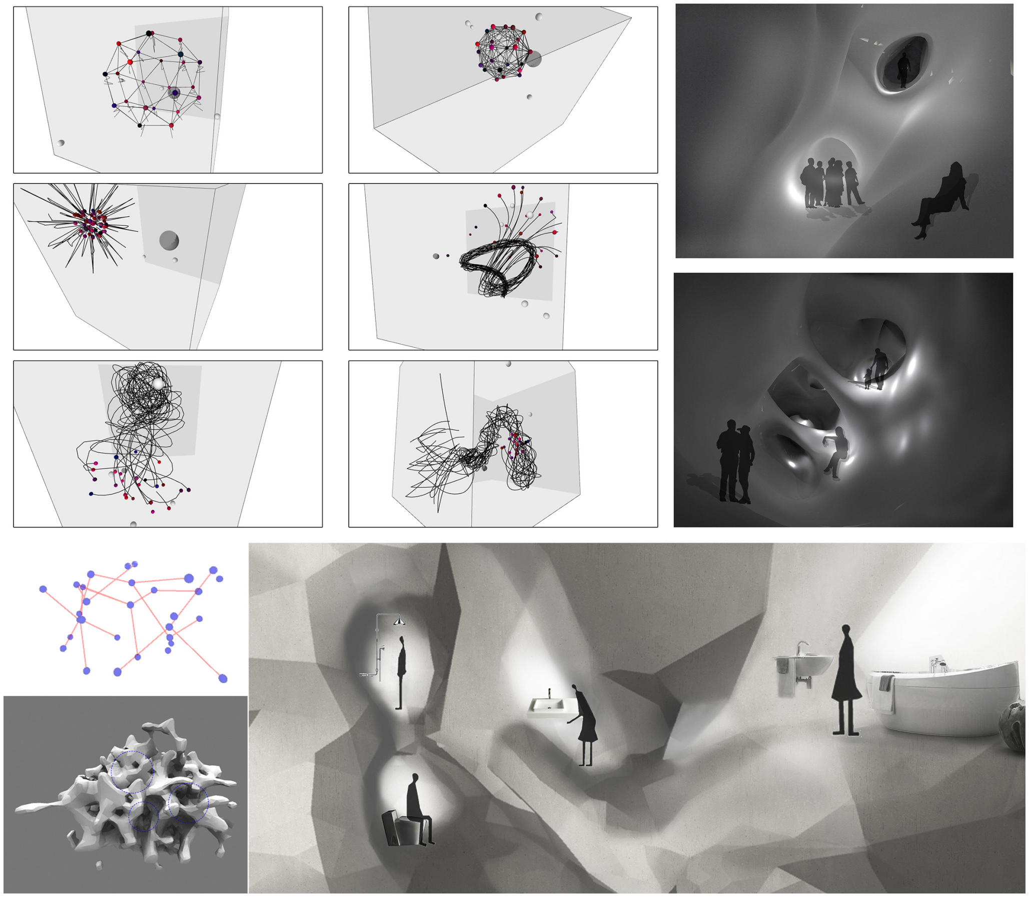


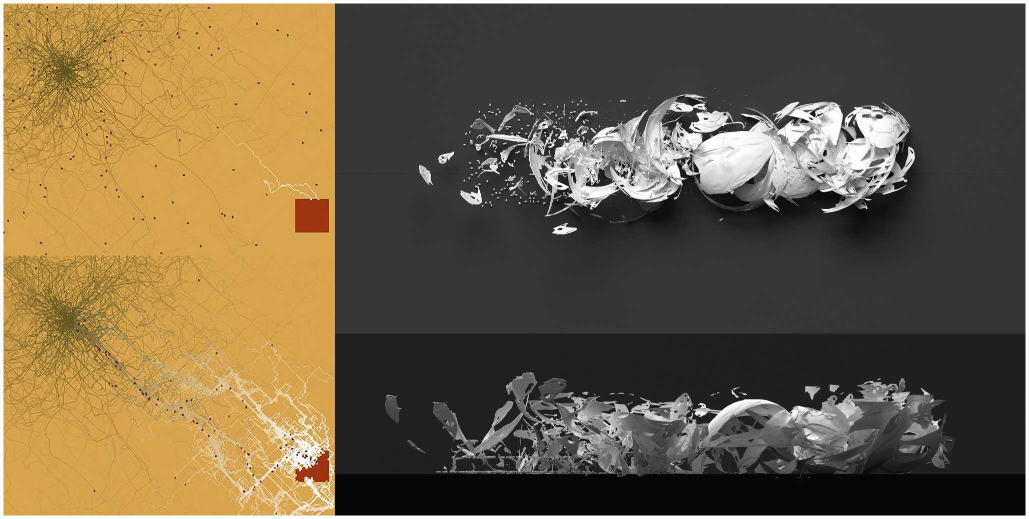
Figure 7
(Top) "Paramecium Drawing" by MA student (2014/15) Shen Guanlong (aka Jerry) derives inspiration from single-celled organisms that are attracted to light. The behaviour of which was used as a mechanism to draw and generate spatial compositions that materialise bottom-up. (Bottom) MArch student (2015/16) Ka Leoung uses an attract-repel network of nodes to generate a simple structure which a mesh is wrapped around (using toxi.geom) to inform a spatial scenario.

as a creative device/engaging at the low level because completing a Masters thereafter (Part 2) not only amalgamates further time and expense, but is by that time somewhat detached from the "day job".

TO CODE OR NOT TO CODE?

The fundamental question anyone interested in using computers as a medium for design must ask is whether to learn to code, or not. The underlying concern to this question is whether they should engage with the machine at a level typically reserved for computer scientists and nerds. Computers are a part of our everyday life, and computation has become an intrinsic aspect of architectural design today. But computers are typically background artefacts. Something we use in the process of doing something, which enables and enhances production and streamlines the way we do it. The vast majority of

Figure 8
March student (2016/17) Mark Ngo uses the pheromone trails generated by ant-like agents searching for food to generate a trail network which he exported into Rhino to produce a 3D spatial composition.



architects and designers use computers as a production tool. Fewer use computers as a tool for design. Fewer still use computers as a creative device - even in this age of digitalisation.

As a tool for design a user typically engages at the level of a CAD package (such as Rhino), to build a three-dimensional form using the in-built functions and procedures provided. Whilst using a 3D CAD package enables and enhances our design abilities it also constrains; keeping the user within a well-defined "design space". The user has the freedom to generate and build models but the manner in which they do so pertains to particular procedures and means dictated by the functionality of the program: i.e., the software designer felt they knew a better way than another program to do something in a particular way. A typical question a student may have when seeking to learn 3D modelling is which CAD software they should learn - because each package has its merits, functionality and thus particular ways of doing things. One may answer "learn any", because once you've learnt one you might crossover; and having understood the nuances of each platform be able to use whichever package best suits what you want to achieve. Whilst this is true, and not new, the issue is that the different packages require one to think in a particular way. 3D modelling packages, generally, do the same thing. They enable a user to build three-dimensional models, which they can animate and manipulate, but the functionality and ways and means of doing something is aligned to the developer's view of how something should be done. The same thing goes for visual programming languages, which enable students to construct diagrammatic programmes in an interactive way. Having learnt the graphic interface and functionality students can produce "something" quickly, with relative ease, and then simply play around. This provides a systems-oriented approach to "generating" something, but what the user gains in ease of programming they lose in being able to manipulate the "finer details". It is important to understand "algorithmic thinking" and the basics of programming, quite simply because it makes you build from the ground up. (See Burry 2011). Textual programming is empowering. It is the language of the computer scientist and it is necessary to "speak" this language. It is a question of communication and not of simply doing "something".

Steve Jobs said, "I think everybody... should learn how to program a computer because it teaches you how to think". (See URL reference: one minute in). The ability to code is liberating, because it enables you to get to the base of how to do something. To draw a line with a pen and paper is "natural", but using a computer to do the same thing is a complicated task. There is no kidding - code is difficult, time consuming and often daunting. But once you have figured out how to draw a line, and to copy, rotate, and offset that line the capacity to draw lines is enriched. The computer cannot and will not replace the qualities of a line drawn by hand but the capacity of the computer to do something repetitively and automate the process provides the "drawer" a freedom otherwise unknown. Speed and automation are key benefits, but the capacity to surrender to the process is liberating. For example; the addition of lines may change colour and an element of randomness may be combined into the mix. The lines could be animated, such that they move and have the capacity to affect and be affected by other lines. Once the "drawer" has figured out how to draw a circle the same process can be adapted to drawing circles, and then lines and circles; and so forth. In short the drawing of lines becomes an animated, dynamic and potentially self-organising and emergent process, which opens up a world of possibilities for the "drawer". In this way the machine is used creatively, because the only constraint is the "drawer". Additionally, the problem of having to work out how to draw a line in the first instance requires the "drawer" to think about the process of drawing lines. In short, programming how to draw a line mirrors the thought process and action of drawing lines. Learning to code empowers because it makes one think from the ground up.

Notes:

1. LSA students have achieved 3 x RIBA awards at the RIBA president's medals in the past 5 years. A significant achievement paralleled only by four other schools of architecture: Bartlett School of Architecture, Royal College of Art, University of Sydney and London Metropolitan University.
2. Not literally - what occurs is that the students experience considerable frustration and fear that they cannot get to grips with code. They (reportedly ....and the author bears witness to the tired unkempt zombie looking beings the students become as the course progresses) spend long hours starring into space, wanting to throw their machine (and most likely the author) out the window.

ACKNOWLEDGEMENTS

The author would like to acknowledge Alasdair Turner (author of the Processing course on which the course presented is founded) and the reviewers for their comments and suggestions, which have helped to improve the content of this paper. Not forgetting the students, who opted to take the course, and whose effort and input have made the course a delight to teach.

REFERENCES

Burry, M. 2011, *Scripting Cultures: Architectural Design and Programming*, John Wiley & Sons, Chichester, UK

Coates, P. 2010, *programming.architecture*, Routledge, London

Franklin, S. and Graesser, A. 1996 'Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents', *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Budapest, Hungary, pp. 21-35

Ingham, J. 1997 'What is an Agent', *no source given*, Centre for Software Maintenance, University of Durham

QAA, initials missing 2010, *Subject Benchmark Statement for Architecture.*, The Quality Assurance Agency (QAA) for Higher Education 2010

von Uexkull, J. 1957, 'A Stroll Through the Worlds of Animals and Men', in Schiller, C. H. (eds) 1957, *Instinctive behaviour; The development of a modern concept*, Methuen & Co. Ltd., London

[1] https://www.youtube.com/watch?v=mCDkxUbalCw