Flying Bricks

Algorithmic Design Studio

Wolfgang E. Lorenz¹, Gabriel Wurzer²
1,2 Vienna University of Technology | Institute of Architectural Sciences | E259.1 Digital Architecture and Planning
¹,² {lorenz|wurzer}@iemar.tuwien.ac.at

The design studio 'Flying Bricks' was held during the summer semester 2015. Its main objective was to redesign an existing building with the use of facing bricks algorithmically, utilizing algorithmic thinking and programming as a means for form generation. The purpose of having students express their designs in terms of code was to emphasize problem thinking over solution generation, which has several advantages but also disadvantages which we would like to share in this paper. Furthermore, we would like to show how our implementation process worked, so that others can leverage that for their own algorithmic design courses.

Keywords: NetLogo, Digital Design, Bricks, Education

MOTIVATION

In this paper we discuss our experience of teaching algorithmic thinking during a design studio using a multi-agent simulation software. Moreover, we will have a look on how to promote problem thinking over solution generation.

'Flying bricks - algorithmic design studio' was held in the summer semester 2015. The objectives of our course were threefold:

1. We wanted our students to digitally design facades (or facade elements),
2. to learning algorithmic thinking and
3. to implement their ideas in form of a program (script).

In a first step, students had to choose an existing building which would have then to be redesigned with facing bricks. In that process, they had to constantly think about possible implementation of their ideas with the help of algorithmic methods which we taught them during the design studio. The actual programming work happened in NetLogo (Wilensky 1999), a multi-agent simulation which offers a relatively "easy" scripting language plus built-in 3D visualization. Most students had no background in programming, and thus we also gave an introductory workshop on that subject beforehand.

The course was accompanied by the question how to promote problem thinking over solution generation, which is still underrepresented in architectural practice. To illustrate that point further, we wish quote Lawson (2005) who did an experiment in which students from architecture had to compete against postgraduate science students. Their task was to arrange Tetris-like blocks according to some "hidden" rules which were only known to a computer which they could ask whether their proposed solu-
tion was valid. In doing so, they showed strikingly different and consistent strategies:

"[...] while the scientists focused their attention on understanding the underlying rules, the architects were obsessed with achieving the desired result. Thus we might describe the scientists as having a problem-focused strategy and the architects as having a solution-focused strategy." (Lawson 2005, 43)

Thus, we can summarize that the problem which our algorithmic design course tackled was to shift the students' attention - at the very least to an extent - from the solution space to the problem space.

**EDUCATIONAL METHOD**

In the first phase, students analyzed historic buildings and innovative brick systems in order to find new creative application areas. The design of facing brick facades is a complex issue in architecture. While facing bricks are still common and an integral part of architectural form in many countries of the European Union (compare The Netherlands, Belgium, Germany, Italy), application in our own country (Austria) has continuously decreased since the epoch of historicism. Such a development has been assisted by the introduction of Exterior Insulation and Finishing System (EIFS). The success of EIFS lies in cheaper constructions, light weight and thinner walls that result in an increase of space (and thus in an increase of revenues). Thus, the first phase of the design studio was characterized by the understanding of the brick format, its restrictions and the requirements on a design regardless of actual implementation. Students analyzed important impacts on the algorithm and on the design, such as the neighbouring building, the shape of the building itself, the proportion of the facade, the place and the orientation. "Wienerberger AG", a global manufacturer of bricks, acted as our industrial partner. They let us visit their factory for clinker and supported us with information about the technical requirements of brick construction from the very beginning.

In the second phase, algorithmic methods that would support the students in their design were studied. In that context, the students were faced with two questions:

1. How well would algorithms provide them with new perspectives on the problem and
2. to what extent would they be able to generate alternative solutions with algorithmic help, in the same fashion that they had done by hand in their previous design studios.

As they had to find out, there is no overall answer to these questions. Algorithmic thinking means, above all, to break down the design process into separately solvable subareas (Alexander et al. 1977; Alexander 1979). These areas included the overall form on a broad level, colour schemes on a mid-level and the brick format itself on a low level.

The third phase was occupied with the final designs. It required exporting the outputs of the students' programs for further elaboration in a CAAD (Computer Aided Architectural Design) tool, plus rendering in a 3D modelling platform. All of these materials were then combined in presentable format (slideshow, poster, short paper) in a final step.

**Algorithmic design**

We made the experience that from the very beginning of the design process students tends to have an almost finished picture - or at least a sketch - of their design ready in mind. This picture may change during the process of analysis (location and task) and/or design; however, an interim result is present. Our design studio 'Flying Bricks' focused on the problem itself rather than on the result. The problem was, so to speak, within the design influencing factors (parameters). Thus, rules had to be defined that solve the problem and influence the result.

The initial problem, to re-design an existing facade with facing bricks, was, first of all, divided into solvable sub-problems. In order to bring these sub-problems to the surface, the main focus was placed on certain influence factors. 'Site analysis' by Edward T. White (1983) provides an insight into the numerous factors that influence architectural design. Some
of them were incorporated or expanded by the students. The existing building itself was recognized as initial group of influences, including its geometry (cubic or free form), its proportion, its rhythm (sequence of openings) and the relation between open and closed spaces. The neighbouring buildings defined another group of influences, including the position of the cornice, zoning (base zone and roof zone), proportion and size of openings (windows and doors). But also public space served as inspiration, such as the course of the street. Location and orientation were also considered as an important group of influences, including shadowing, protection from street-noise, wind, permission of outlook and insight, distance to the street or position above ground level.

Finally, the material used comprised another group of influences with its own problems (and their solutions); e.g. the small scale of bricks and the resultant specific need of static load-related arrangement. Bricks laid bonded in a structure not only increases stability and strength but also leads to a varied picture; furthermore, small scale provides flexibility of geometry. In short, size was discovered to allow different possibilities of brick bonding that ask for certain rules (minimum overlapping of bricks for stability). The section about bricks was enlarged by colour psychology (different possibilities of bricks colour), the rain security of bricks that makes further weather protection unnecessary (protection from environmental influences, such as sun, wind, rain and their impact on geometry of the composition) and the extraordinary durability of bricks that provides benefits in the preservation of buildings.

In summary, the how and why was already present in the approach of solving the problem, which influenced the final form. Therefore the definition of design influences was most important. In short, problem solving was considered to take up influences to guide towards the solution. This yields two possibilities for a program:

1. To generate an optimum; this includes less material consumption or the claim for the smallest number of different shaped bricks; e.g. the Johnson Wax Building by Frank Lloyd Wright uses over 200 different shapes of bricks to form the angles and curves of the design (Levy 2004), and
2. to generate a set of possibilities, from which the "best" solution can be selected.

We hoped to find new aspects, that otherwise would not have been discovered or only with increased personal effort. In summary the design process included several phases:
- Drawing of the existing facade in a CAAD program and obj-export,
- import of the obj-file and voxelization in Net-Logo,
- application of the own script to laying bricks,
- export of the result as obj-file and import in a CAAD program and
- further work on the result.

IMPLEMENTATION METHOD
Learning from Lawson (2005) and White (1983) two questions arises: What if the mentioned influences are stored as values inside a three-dimensional grid? And what if this stored information influences the way of how an object evolves? This kind of, so to say, information driven design has successfully implemented during a workshop in Białystok (Wurzer and Lorenz 2014). The advantage of an agent based programming resides in the possibility to control agents (entities) externally or internally. External controls comprise the information stored in the grid. This supports the idea of problem thinking solution. The solution is inherent in the stored information.
Agent Based Programming
Agent based programming offers the advantage that results of a simulation can directly provide the information comprised in the model (e.g. simulation of wind, the sun’s course and noise dispersion). The results of these simulations are immediately available to influence e.g. the path of agents that form a specific design, leave traces, or, as in our case, simply place bricks. Agents are, so to speak, seekers, driven by the places they pass. One could almost say, agent based programming provides an “all in one solution”.

The key issue of our model is how and why agents move and who controls or influences the agent on his way from a starting point to its goal. In the case that the agent does not know its path per se, we speak of external control. A well defined playing field (the outer wall of certain dimensions) provides such a restriction. The agent may be free to walk (randomly) but is thrown back before passing the border line. In an advanced model the current location tells the agent where to move next, as in a pinball machine. Relevant information that affects the path includes differentiation between solids and openings, distances to specific points, heights above ground or shadowing by other buildings.

NetLogo Programming
Since NetLogo is a multi agent programming language with integrated modelling environment, it served as a basis for all of our programming work. The tool is more than a programming language, since it offers several useful metaphors that are already in-built, ready to be used when implementing a design:

Voxel space. NetLogo’s virtual world is a 3D regular grid made of voxels (i.e. cubes), each having an extensible set of properties (e.g. colour).

Agents can move freely within voxel space. Each agent also carries an extensible set of properties (e.g. position, heading shape, colour and type; the latter is useful for defining different ‘flavours’ of agents, such as builder, seeker and so on).

Writing a program encompasses two steps:
1. Setting up the world (assigning initial colours to voxels, creating agents) and
2. calling a step function continuously, until some stopping criterion is satisfied.

The step function is occupied with moving agents through voxel space, and letting them alter their environment as they pass through (e.g. laying a brick). In that piece of the program, they also need to make use of sensory capabilities - i.e. "What is the colour of the voxel ahead", in order to decide what action to perform.

Our design studio was occupied with redesign of existent building using bricks. Thus, we had to somehow load these into NetLogo during the setup phase, in order to have a "substrate" along the facade where agents could do their (brick-laying) job. In more detail, we imported the 3D geometry as faces made out of three vertices, which we then subdivided until they were of voxel size (see left in Figure 1). Then, we simply asked each voxel to colour itself if it contained a vertex. The results of that process is shown right in Figure 1 (for more details, refer to Wurzer 2014).

Figure 1
Principle of voxelization (left) and result (right).

It was now the agents’ job to lay bricks layer by layer, moving from one coloured voxel to the next coloured voxel. In order to get into even more detail, we wish to briefly present some examples that show how this was done (see Lorenz and Wurzer 2015 for an in-depth explanation).
DISCUSSION OF OUTCOMES

The playing field, as described before, was defined by the outer walls of a specifically selected building. In order to import a wall into the world of NetLogo, two possible ways exists: The wall can be discrete or continuous. The discrete world, a three-dimensional grid, is preferable, since a division into voxels offers the possibility of additional storage of information in each voxel (which, in turn, influences the path of the agent or properties of the brick).

Our participants used NetLogo's possibilities for agent-based programming in different ways. Some used voxels to store information, which then influenced agent movement along the facade and/or properties of the built bricks. Other students simply set bricks in a given wall without reacting to context. Some projects even went as far as to incorporate actual simulation within their design. We are now going to present a few examples in more detail.

Simply set bricks: This first program served as inspiration. The design emerges exclusively out of the algorithm itself. The basic idea is to simply offset bricks in a stretcher bond along a given orthogonal outer wall of a building. The particular challenge lies in well defined corners with bricks exactly positioned one above the other (without any offsets). Therefore, first of all, a "spy" agent analyses the geometry of the outer wall. Later, all bricks are arranged smoothly between the fixed corner bricks (let's call them "reference" bricks). The "spy" agent also checks if a full brick or a half brick can be laid at corners and at the edges of openings. The starting position of a single agent is set automatically and the custom property "offset" of the agent is set to "right". At each step, the agent determines whether its offset is "right" or "left". In case it is "right", it tries to place a brick two centimetres to its right (opposite goes for "left"). If this position is not coloured, then the agent is beyond the facade and may not place a brick there. Thus, the offset is set to its opposite value (left becomes right, right becomes left) and the process starts again. If this is not possible either, then the program stops. In all other cases, the agent moves one step along the voxelized wall, adjusting its heading if it is necessary to turn (e.g. at corners). If, after taking a step, it reaches a voxel it has already visited, it changes its position to one level up. If there is no coloured voxel there, the agent is beyond the facade and stops the program. In all other cases, the step function is repeated (see Figure 2a for a result).

Viewpoints: The basic idea of this program was to create a reference oriented design that reflects certain viewpoints in a specific combination of headers and stretchers. At the beginning, the user defines specific viewpoints from outside the building. The algorithm then automatically detects four main corners of the building (referenced diagonal to the four corners of the NetLogo world). Initially, the facade is covered with a stretcher bond. Subsequently, previously defined areas (e.g. above/below windows, layers with/without openings and n-th layers) are specifically handled: Some stretchers are replaced by rotated headers and offset outward (leading to a picture similar to a course of diagonal bricks; Figure 2b).
The replacing of bricks starts with a stretcher followed by a rotated header. The sequence continues with a stretcher and three headers, two stretchers and five headers, four stretchers and seven headers and vica versa. The special feature is that all headers are oriented to one of the given viewpoints, which were deemed to be "important". This means that the design exclusively depends on the geometry of the building (including openings), the viewpoints and the given sequence of rotated headers and straight stretchers. In other words, the layout is exclusively defined through the rules defined at the very beginning.

**Light and shadows:** This program based on the idea to map light and shadow on facades depending on orientation (and the surrounding environment). For this purpose direction of light and intensity is calculated and stored in each voxel. The orientation of the facade, the height in relation to the street level and the shading by neighbouring buildings influences the degree of offset (from one brick to the next). In other words, roughness depends on exposure. For the sake of simplification, sun-intensities are summarized in definite ranges of offset. The offset of each brick operates like described in "Simply set bricks" with a maximum displacement of six centimetres (Figure 2c). This example demonstrates how simulation (of light) and design can take place in one and the same program. However, technical problems prevent current realization. Apart from that, the vulnerability of the algorithm mainly occurs at corners with a non-right angle due to imprecise transitions.

**Simulation as input:** The reasoning underlying this approach was the facade’s function to protect. In more detail, areas of higher pollution should be reflected by darker colour; furthermore, colour graduation and different offsets should provide more variety. A simulation of passing cars, leading to pollution, serves as input for the design. The density is added up and stored within each voxel of the facade (together with the maximal distance to the source of noise). Agents (brick layers) ask for this information and reacts by altering their colour and their offsets (Figure 3a-3b). More precisely, colour, on the one hand, depends on the maximal distance to the source of noise, classified into three colour spectra. Offset, on the other hand, depends on intensity, again, classified into three ranges. In general, displacement is restricted to two centimetres to keep the maximal difference between two layers below four centimetres (recommended by the manufacturer). In this example NetLogo was considered to confer a large benefit in terms of simulation, but also in terms of the possibility to export colour information.

**The urge to turn:** The basic idea was to cover the facade with continuously rotated bricks. Bricks at corners of the building and of openings serve as fixed points oriented in 45 degrees to the outside. All bricks between two corners change their orientation in order to complete a rotation of 90 degrees. The bond’s front view displays a continuous development from a rather "smooth" appearance, with the only structure given by bed and head joints, to a rougher appearance of strongly protruding bricks at
Despite rotation, bricks of the next layer have to centrically cover head joints of the layer below. In a first step, a so-called “spy” agent walks along the facade in order to capture the geometry. This includes to identify fix points (corners and edges of openings), to calculate the orientation of the angle bisector of fix points and to identify successive levels with abrupt changes of orientation. The latter occurs above and below edges of openings, when a layer of continuous rotation between corners meets a layer with many rotating areas between openings. Moreover, the anchorage is carried out by horizontal consoles with straight stretcher bonds. These new layers influence again the rotation of the brick layers above and below. All the information is stored in the voxels and the agent is guided by this information. The algorithm is designed for rectangular buildings only. It is an example that reminds us of design intent (which does not always need context; Figure 3c-3d).

**Brick patterns with bonds:** An intensive examination of different bonds led to the idea of this program. Patterns or pictures should be projected on facades, visualized by different structures. The template might derive from an image or from the structure of other facades. Both, the information about the section of wall (opening or solid) and about the pattern are stored in the voxel. The agent just reacts on the information it gets from the current position. A special challenge is the transition between different areas of specific bonds. They may not be abrupt but continuously. This is solved by creating new intermediary bonds (Figure 4a). Additional feature concerns colouring and/or offset of headers.

**Chameleon:** The basic idea was to take up the bonds of surrounding environment and to colour and offset bricks depending on their orientation. The more a facade is oriented to the sun, the lighter the colour of the brick and the larger the offset (to increase self-shadowing). Both, calculation of the orientation and the setting of a simple header bond are done automatically. Subsequently, single bricks are offset depending on the orientation of the facade (Figure 4b). Bricks at the corner and the edge of openings are not offset.

**DISCUSSION**

During the course, scale differences between NetLogo and CAAD programs turned out to cause the main difficulties. While the latter have a well-defined longitudinal scale (e.g. meter or centimetre), the scale of NetLogo is defined by the number of voxels in x-, y- and z-direction. This is bound up with the fact that voxelization entails the disadvantage of simplification. The original shape is transformed into boxes of a three-dimensional grid with the resolution defined by the box size. The sizes of the outer wall and of openings may be distorted (e.g. windows of the same size may turn into openings of different size). To increase resolution of the grid means to reduce the size of the boxes. However, this comprises performance. During the import process parts of the wall can be imported separately, e.g. parts of the wall, windows and doors. Each part can be associated with a different colour of the voxel and a different type (wall or opening). Since overlapping of

---

Figure 4

‘Brick patterns with bonds’: transition between bonds (a), 'Chameleon' (b).
these elements is possible, the order of importation into NetLogo is important.

The key benefit of NetLogo is the simulation-based programming itself, since it can be used directly as source of information (simulation of noise distribution). Moreover, agents (the placement of bricks) are directly driven by forces contained in voxels (a certain place) and not by the outside. This opens up possibilities for future research.

Concerning problem thinking over solution generation it turned out that in most cases students still have certain results in mind. Sometimes the algorithm was even developed to fit this idea. What was helpful, on the one hand, was to think about certain influences and to implement their impact on the design. To store information in voxels that lead agents and subsequently set bricks was, on the other hand, the main aha-experience. We will focus on this aspect in our future work.

CONCLUSION

In this paper, we have presented our algorithmic design studio 'Flying Bricks' which had the purpose of teaching students algorithmic thinking and problem-oriented thinking through agent-based programming. The context of the design studio was building with bricks, supported by our industry partner 'Wienerberger AG' who advised on practical questions in brick construction. Results obtained by our students show that such an approach is feasible even when starting from scratch, both programming-wise as well as in brick technology. We are convinced that similar lecture formats which let students assemble "simple elements" to form "complex shapes" are didactically advantageous and guarantee a steep learning curve.

ACKNOWLEDGEMENT

We thank Mr. Staudinger and his team from Wienerberger AG who provided insight and expertise. Furthermore, we would like to thank the following students, since their contribution is part of this paper: B. Arendt, M.A. Schiltz, A. Malhotra, B. Hanser, B. Blank-Landeshammer and N. Jasinski.

REFERENCES

Alexander, Ch. 1979, The Timeless Way of Building, Oxford University Press
Levy, H.H. 2004, Artists and Architects (Famous Wisconsin), Badger Books Inc
Wilensky, U. 1999, NetLogo, ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.