Teaching Programming and Urban Complexity to Architecture Students

Lukas Treyer\textsuperscript{1}, Daniel Zünd\textsuperscript{2}
\textsuperscript{1,2}Chair of Information Architecture, ETH Zürich
\textsuperscript{1,2}http://ia.arch.ethz.ch
\textsuperscript{1,2}\{treyer|zuend\}@arch.ethz.ch

In this paper we present a framework that allows to introduce architecture students to agent-based simulations in the context of urban planning. It provides them with an understanding of how such simulations work by instructing them to learn how to program and develop an agent on their own. Along with the framework we explain our didactic concept of teaching complexity-science-methods to students from other fields such as architecture. In the discussion we report on that theory and practise should be alternated at very short intervals. Additionally we emphasize the importance to teach a good understanding of the capabilities of modelling and simulation tools, since uneducated students tend to trust them too blindly.

**Keywords:** Programming, Crowd Dynamics, Urban Planning

**INTRODUCTION**

Urban planning is a task dealing with many uncertainties (Mustafa et al. 2015). It is hard to predict the impact of decisions and the consequences of decisions made. With modern technology, it is possible to use the increasing computing power to reduce uncertainties and to better understand the urban system of interest.

This paper introduces a course held to architecture students from different semesters and with different backgrounds. The aim of the course is to introduce the students to simulations and give them insights into how simulations can improve the urban design process, but also rising awareness of the disadvantages of simulations. To be able to run the simulations on a student’s computer we limit the scope of the urban planning perimeter to the area of a latin music festival in Zürich. The agents in the crowd simulation reflect, amongst others, visitors of the festival with different needs.

In the course of the lecture series, students learn how to use an agent-based modelling and simulation framework, and all the tools that are needed to use it. This means that students first are taught the programming language that is later used in the agent-based framework. Later in the lecture series, basic concepts of complexity science are taught and the change of perspective, introduced by it, is explained. In the last part of the lecture series, the students have to implement their own agent.

The implemented agents are finally run in the framework and compared against each other. To motivate the students to invest some time into their implementation, the simulation framework is set up in a way that it is possible to directly compare the different implementations of the students. This is used
to run a competition with all agents in one simulation. The students with the strongest, longest living agents receive a price.

**DIDACTIC CONCEPT**

In this section we describe the structure of the course and explain the interplay between the three main parts "programming for beginners", "design of the festival layout", and "simulating complexity with agents". The course breaks down into ten lessons, four of which dedicated to the introduction of programming for beginners, two to the design of the festival layout, three to the agent programming and one to a final presentation.

The course addresses architecture students as an addition to their regular design courses. It focuses strongly on the programming rather than the design aspect. Nevertheless, we conceive it as important that the course includes a short design exercise to relate the simulation tools to traditional planning methods. Also, this exercise introduces the few students from other disciplines such as Mechanical Engineering and Integrated Building Systems to basic concepts of architecture and design.

**Programming For Beginners**

We start the introduction to programming by pointing out the different way of how programming tasks are tackled, compared to the traditional way architects cope with a task. By comparing computer science with architecture we implicitly mention the debate on the relation of design and science (Cross, 2001) in the sense that computer science emerged out of physical / mathematical science as an engineering discipline while architecture in the 20th century gradually got more and more backed by engineering sciences such as civil engineering. This means that both disciplines require a domain knowledge in order to design (software or buildings). For the beginners in the course, we think it is important to mention, that the skills they learn are not sufficient to do software design. Yet they start to learn the art of programming, the craftsmanship of computer science, comparable to drawing plans in architecture; even though way less expressive when comparing the immediate media "code" and "drawing". So we need to prepare the architects to deal with a textual representation of a language that is by far less expressive than a spoken language or the graphical language they know from architectural drawings and that on top of that depends on many very strict syntactical rules.

After explaining the building process in programming, its concepts are introduced by focusing on the key aspects "data types", "control structures", "grammar rules". The taught programming language is Processing/Java [1] and the students are also introduced to Eclipse [2] in the first interactive step-by-step exercise. In the second lesson we provide an alternative approach. Having heard about the concepts that are key to the flow of a program, students are presented the syntactical elements: a list of keywords and operators, an introduction to blocks and statements and an explanation of the meaning of selected characters and keywords such as ":=" and "class". The third lesson introduces object oriented programming concepts since we heavily depend on it using the simulation framework later in the course. The third exercise accompanying this lesson asks students to animate an object in the Processing framework. The fourth lesson on programming introduces the simulation framework, its structure and how to use it in the course. In the sequence of the course this is actually the fifth lesson with the fourth lesson giving an introductory overview on simulation methods and complexity science.

**Simulating Complexity With Agents**

Before introducing the simulation framework, which is later used in the course, students are introduced to modelling, simulation, and complexity science. The goal is that the students start to understand how simulations work, and how to interpret and use them in urban planning. A key aspect of this part is to encourage a critical attitude towards computational urban simulation methods and make students aware
of what complex simulations are able of and what not - similar to weather forecasts. They are taught to treat simulations like sketches: imperfect images that communicate an idea. And we raise the awareness of the verification problem of simulation models. Thus they learn how to interpret the results of a simulation and not just blindly trust the computer. It became clear during the design exercise, that this is a crucial part, when teaching modelling and simulation to laypeople. This crucial part of educating architectural students and introducing them to digital simulation tools is further discussed in the discussion section.

The last part of the course consists of implementing agents for the agent-based simulation. Each student has to implement his own agent. Implicitly it should become apparent that they develop an agent not as close to a real person as possible but an agent that deals with the constraints set by the framework in the most efficient way.

**Design of a Festival Layout**

To introduce the usefulness of modelling and simulation to support design and planning task, students have to plan the festival in a traditional way. The students have to place 20 drink stands, 20 food stands, 8 toilets, 3 music stages as if they would in urban planning when defining land use zones. Additionally, the students are divided into four groups, each of them trying to optimize the festival layout according to one of the following criteria: (i) flow of people; (ii) overall profit of all stands; (iii) number of happy people; (iv) security. As not only architects may attend the course and since the design task is very simplified, the design can be visualized using office tools such as MS PowerPoint or a scanned hand-drawn sketch; it does not need to be a precise drawing. The designs are then presented by explaining why they think their design fulfills the assigned criteria best. The idea of this exercise is to relate the traditional planning methods to the subsequent task of programming an agent that is competing against other agents in the given simulation framework. As it turned out during the course and as discussed in the discussion section this exercise is also a valuable test for the critical attitude we hope to induce earlier in the course.

**SIMULATION FRAMEWORK**

In this section we introduce the agent-based framework for crowd simulations that was specifically implemented for this course. We are aware that already many agent-based modelling frameworks exist [4] that simulate crowds, but we decided against using an existing one for several reasons.

The main reasons to implement our own agent-based framework are due to its educational purposes. With our own implementation we are able to show and explain the students important concepts of software architecture, without the overhead of already existing implementations in terms of number of classes and configuration files. The simplicity of the framework and the number of lines of code gives the students a chance to understand the whole framework, and motivates them to do so. The framework was implemented using Java/Eclipse [2] with the Proclipsing plugin [3], using Processing [1] as the visualization library.

**Software Architecture**

The architecture of the framework is implemented to directly reflect important concepts of Object Oriented Programming (OOP). Because the course did not require any prerequisites from the student, the focus was mainly on inheritance and polymorphism, which we understand as the two most important concepts of OOP. If the festival visitors are not counted, the whole framework consist of 15 classes, divided into three packages: the geometry package, which contains all the classes that are directly related to geometry; the agents package, which contains all the different types of agents that are used in the simulations; and the main package, which only contains the main class.

The main class sets up the whole program and implements the functions that are needed for processing, such as the setup and the draw methods,
which set up the visualisation window and draw the state of the simulation, respectively.

The agents of the simulation are all implemented within the agents package. The implementation of the simulation is such, that all entities that are present in the simulation are agents, i.e. the wall elements (Walls), the entities that provide the needs (Needs), and the visitor of the festival (Visitors). The base class for all the agents in the simulation is the Agent class. It defines the basic functions that all the agents in the framework must implement.

The Walls are agents that are immobile and they only limit the liberty of action of the Visitors. They are used in the simulation to imitate the street canyons and force the visitors to stay within them. The Needs are agents that are immobile too, but have the capability to serve certain types of needs the visitors have, and, in the here described version of the framework, represent drink stalls, food stalls, music stages, and toilets. The Visitors represent the people that visit the festival and are the only mobile agents. They have the four needs which are served by the four Needs agent type. The needs have to be fulfilled within a certain time, otherwise they will leave the festival again.

**Simulation Dynamics**

The framework we implemented models in the micro level. This means that every agent is simulated atomically. We do not assume any social interaction between the agents, they are only self-serving and try to fulfill their needs. When the agents enter the festival, they choose the location they want to go and then walk into the goals direction until they reach it. Only then they choose a new location, to which then they walk. This process is repeated until one of the needs can not be fulfilled in a certain time. The basic algorithm of the visitors is depicted in Figure 1. The variables that are initialised at the beginning describe the happiness score of the agents with the four needs. At the beginning they are satisfied with all the needs, but they slowly decrease as the simulation progresses.

```plaintext
initialize needs scores;
while all needs satisfied do
    select location;
    while location not reached and all needs satisfied do
        step towards location;
        decrease needs scores;
    end
    if location reached then
        increase specific needs scores;
    end
end
leave the festival
```

The direction the visitors walk during the simulation is calculated with a simplified version of the social force model (Helbing and Molnár, 1995). We drop most of the terms and use only the ones that are needed to calculate the before described behavior. Each agent \( i \) has to solve the following system of equations to know his new direction of walking, thus his location after the next step:

\[
\frac{d\vec{x}_i(t)}{dt} = \vec{v}_i(t) \tag{1}
\]

\[
\frac{d\vec{v}_i(t)}{dt} = \vec{F}_i + \sum_{j \neq q i} \vec{F}_{ja} \tag{2}
\]

where \( \vec{x}_i \) is the location of agent \( i \) at time \( t \), \( \vec{v}_i \) the velocity, \( \vec{F}_i \) the force that pulls the agent towards his desired location, and \( \vec{F}_{ja} \) the force that repels one agent from another. \( \vec{F}_{ja} \) models the mass of the agents and ensures that they cannot walk through each other, but are compressible. This is an important property for crowd simulations.

Figure 2 depicts the landscape of what a visitors sees if he wants to go to the center of the bottom left edge. Ten other agents are present in the square (peaks) and four needs providers (hills). The layperson can read the image as if the agent was a marble put at any location on the square, and gravity pulls on it until it reaches an exit.
The problem that arises with the above described method is when the geometry gets more complicated. The festival addressed in the lecture, is not only on one square but consists of two squares and several streets that connect them. If the agents would walk only according to equations 1 and 2 they would get stuck in corners. To circumvent this problem, we implement the agents in such a way that they don't walk directly to their desired location. They rather walk along a graph that guides them through the streets, i.e. the graph is set in such a way, that it is always possible to directly walk from one vertex to another, without hitting a wall. After choosing a new location yet before starting to walk they calculate the shortest path from their current location along this graph, and then go from one vertex to another, until they reach the desired location. To prevent that agents walk to not needed vertices, we additionally implemented an algorithm that removes vertices on the path if they are not needed. For example in Figure 3, if an agent is at a location close to vertex 27 and chooses to go to vertex 29, the agent is directly attracted by vertex 29, without going through vertex 28. The reason to have this is twofold. First it gives the agents more degrees of freedom when they walk on the streets, because they are not attracted by too close vertices and thus are more agile. The second reason is that if the agents walk strictly along the graph, also within squares, it can happen that they walk around their final destination or along walls. This can happen when the closest vertex to a final destination is not directly on the shortest path to it, in an Euclidean sense.

**Analysis Methods**

To teach the students the value of simulation, we implemented some basic analysis methods into the framework. The framework allows to have real-time feedback on how dense the crowd is at any point in the festival, visualizes the locations of where visitors decided to leave the festival again, counts the number of customers served at each service provider, and visualises the walking paths the agents took.

To get an overview on festival security it is important to visualize congestions. It allows to plan in advance and understand the possible risks a layout of the festival has. To understand the locations which do no serve the visitors good enough, we implemented a visualisation of the locations of where people decided to leave the festival. It helps the planners of the festival to identify the locations where visitors are not satisfied anymore, be it by the lack of needs providers or due to congestion.

The third visualisation the frameworks provides is the walking paths of all the locations all the agents have been to. This visualisation technique gives the planners an idea of where people walk and highlights high frequented paths and streets. The last visualisation method depicts the number of customers the need providers served. It is of high interest for the festival organiser to know the frequency of customers for one location, for economic reasons. It allows them to estimate the rent prices for the locations.
EVALUATION

For the evaluation of the course we asked the students to answer a questionnaire. It consists of 33 questions, out of which 16 are standard questions taken from our university's official course evaluation survey. If not otherwise stated the questions are answered using a scale from 1 to 5, where 1 means total disagreement and 5 total agreement. Out of the 34 students 14 participated in the survey. The survey was conducted 5 months after the end of the course. Out of the 17 specific questions we show the results of the following 7 in Figure 4. As shown by the answers to D3 students had heterogeneous previous knowledge. Also, five of the 14 survey participants where no architecture students. In addition to Figure 4 we show the results categorized by beginners/no beginners in Figure 5 and Figure 6. All the beginners where architecture students.

The results of the questions concerning the quality of the course, its hand-outs and its teachers can be summarized as rather good. Hence any bad results regarding the teaching goal correlate with the course design rather than its execution.

- **D2** Did you profit from the Q&A sessions after each lecture?
- **D3** Did you take another programming course before this one? (1 = no, 2 - 4 programming was involved a bit or you didn’t finish the course, 5 = yes)
- **D4** Can you imagine to use the learned programming skills in another project in the future?
- **D5** Do you conceive programming more like a language or more like mathematics? (1 = language, 5 = mathematics)
- **D7** How would you rate your skills when confronted with a new programming task such as editing a webpage or filter data in a csv/excel file?
- **D14** Are you able to solve programming problems by yourself / by searching in the internet?
- **D15** Do you prefer to learn programming by understanding the concepts underneath or by trying out and follow syntactical rules?
DISCUSSION
This lecture series tries to reach the challenging goal of combining three different topics into one storyline. By conducting the course in fall 2015 we found a few details worth discussing.

Generally, the students were very motivated throughout the course. Which can be seen in the number of drop-outs during the semester. By experience, about fifty percent of students do not finish courses that are teaching programming. In the case of this course, the fraction of people finishing the lecture series was above 75%.

Nevertheless, during the course of the lecture series, it became clear, that our strong division of fundamental concepts of programming and the syntactical form in Java code is not the best approach for beginning programmers. As Aguiar and Gonçalves stated in 2015 (Aguiar and Gonçalves, 2015), the two should be introduced to early programmers more in connection to each other, so that the theoretical concepts can immediately be tested and hopefully better understood. This also complies with Lahtinen et al. (Lahtinen, Mutka, Järvinen, 2005) finding that students perceive programming structures and language syntax equally hard to learn.

It is also shown by answers D5 and D15 (see Figure 4), that the alternation between a focus on fundamental concepts of programming and a focus on its pure syntactical rules is appreciated by a heterogeneous group of students. While beginners prefer to hear about concepts as shown in Figure 6, more advanced students prefer to learn the syntax first, see Figure 5. Yet, the division of a lesson into one hour lecture and one hour solving the exercise in an interactive Q+A session was highly appreciated by both of the groups as indicated by D2.

The main goal of teaching programming to architects must be considered as failed since the answers to D7 and D14 are very low among beginners, see Figure 6.

A special finding we would like to highlight are the results of the design exercise. We clearly stated, that the students should justify their design arguments, in a qualitative way. Even though the categories do align with the values calculated by the simulation framework, we thought the students would understand that they should find qualitative arguments for their layout, that in the best case would not be covered by the values calculated by the framework. Nevertheless, when doing the exercise almost all the students ran the simulation they were introduced a lesson before, and then stated that they chose the layout, because it produced the best simulation results. This shows that, even thought we discussed the significance of this kind of simulations in an earlier lecture and pointed out that they should design the layout in a traditional way, the students still used the simulation to justify it.

CONCLUSION
The lecture series, introduced in this paper, offers a way to introduce architecture students to complexity science, programming, modelling and simulation. We were able to keep the students interested in the topic, until the end, even though it was an elective course.

The evaluation of the course shows that not all goals of the lecture series were met. Students with an already basic knowledge in programming were able to strengthen their skills. For students with no prior knowledge in programming, the course seemed to be too difficult, even though most of them were able to solve all the exercises without any help from the assistants.

The didactic concepts, in its general form, was able to lead most of the students to the goal of understanding basic concepts of complexity science, programming, modelling and simulation; with the need for only few adaptations.

REFERENCES
Aguiar, R and Gonçalves, A 2015 'Programming for Architecture: The Students’ Point of View', Proceedings of the eCAADe Conference, pp. 159-168


[1] https://processing.org