Swarm Grammars GD: Interactive Exploration of Swarm Dynamics and Structural Development

Sebastian von Mammen, Sarah Edenhofer

Organic Computing, University of Augsburg, Bavaria, Germany
{sebastian.von.mammen, sarah.edenhofer}@informatik.uni-augsburg.de

Abstract

We present an interactive simulation of Swarm Grammars (SGs). SGs are an extension of L-Systems, where symbols of the production system are considered agents, whereas the given production rules determine their differentiation or reproduction. Assigning boid properties to the SG agents yields spatial dynamics apt to building structures in space and to collaborate stigmergically. In the presented interactive simulation, we put an emphasis on accessible interactive visuals for shaping the initial configuration of the simulation, to program the agents' perceptual and productive behavioural abilities, to dynamically drive developmental stages and to fine-tune visual structural properties such as colouring and scaling of the utilised developmental building blocks. Our system has been successfully deployed to promote swarm dynamics and developmental processes as important aspects of Artificial Life in a playful way. We present results from deploying the simulation in the context of an event to promote STEM research among high-school girls.

Introduction

Confronted with the challenge of providing an engaging entrance point to Artificial Life research to young high-school students, we decided on implementing an interactive Swarm Grammar (SG) simulation (von Mammen and Jacob (2009)). SGs seemed to be an adequate choice, as they integrate aspects of complex system dynamics—bridging from the behaviour or simple individuals to the emergent properties of large populations—and developmental processes—yielding unique artefacts that allow to trace spatial interaction processes over time.

In order to kindle intrinsically motivated engagement (Koster (2013)), the simulation was devised to (a) establish a relationship between the students, the software and its artefacts. (b) We had to provide accessible means to defining rules and configuring agents so the students could challenge their competence in the actual simulation processes and explore the outcome of their high-level programming ingenuity. (c) The intensity of the students' explorations and design efforts has to result from their voluntary engagement. Accordingly, the simulation provides an open-ended, directly manipulatable playground environment that is freely and widely accessible as a public, WebGL-driven website[1].

In the remainder of this paper, we present the following aspects of the project. In the next section, we briefly summarise the key concepts from related works that we utilised in the agents' flocking definition and their (re-)production rules. Afterwards, we explain the simulation concept with an emphasis on its visual programming assets. Before concluding this work with a brief summary and an outlook on possible future work, we dedicate one section to presenting select simulation artefacts as well as preliminary user feedback.

Related Work

Our simulation concept has been developed to support the definition of perception and actuation properties of 'boid' agents and the construction, reproduction or differentiation of swarm grammar agents. The deployed mechanics of user interaction and visual programming techniques that will be outlined in the next section, are a translation of the following, underlying geometrical and grammatical relationships.

Boid Flocking

Reynolds (1987) presented the concept of 'bird-oids', or boids. It implements smooth, decentralised steering of swarms of agents based on several 'flocking urges' that emerge from the relationships between an agent and its neighbours. The neighbourhood is determined by the agent's radial field of view (FOV) defined with a maximal distance d_{max} and an angle α, see the annotated screenshot from our simulation in Figure 1.

Neighbours within a minimal distance d_{min} are considered 'too close' and trigger an evasive manoeuvre away from their centre (*separation urge*). The agent further synchronises its velocity (heading and speed) with the other neighbours (*alignment urge*) and it accelerates towards their centre (*cohesion urge*). Coefficients of these acceleration urges determine the emergent flocking behaviour. In addition to

[1] `http://www.vonmammen.org/SG-GD/`

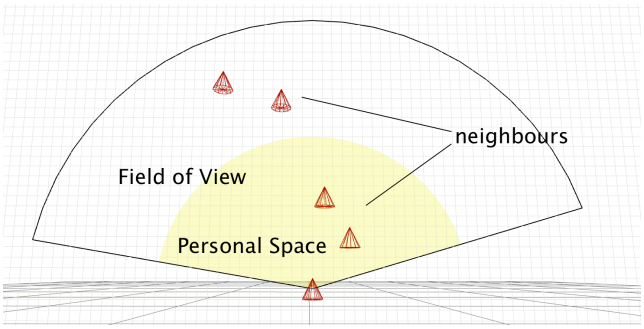


Figure 1: Perception of a 'boid' agent. Agents within its field of view are perceived as neighbours, those within the 'Personal Space' are considered too close, triggering an evasive manoeuvre.

separation, *alignment* and *cohesion*, we utilised a correspondingly weighted random vector (*random urge*) and a global direction vector (*direction urge*) to steer the individual agent. The acceleration $\vec{a}_i$ of an individual i totals its j neighbour-dependent urges $\vec{u}_{ij}$, scaled by the individual's weights w_{ij}. The agents' acceleration and flight are kept within reasonable boundaries by maximal values for acceleration, a_{max}, and velocity, v_{max}. For simplicity sake, we chose integration step size $\Delta t = 1$ to infer the updated position p'_i of individual i. The corresponding equation system is listed below; markings denote the sequence of variable updates, $|\vec{x}|$ denotes the norm and $\hat{\vec{x}}$ the versor of vector $\vec{x}$.

$$
\begin{aligned}
\vec{a}_i &= \sum_j w_{ij}\vec{u}_{ij} \\
\vec{a}'_i &= max(a_{max}, |\vec{a}_i|)\hat{\vec{a}}_i \\
\vec{v}'_i &= \vec{v}_i + \vec{a}_i \Delta t \\
\vec{v}''_i &= max(v_{max}, |\vec{v}'_i|)\hat{\vec{v}}'_i \\
\vec{p}'_i &= p_i + \vec{v}''_i \Delta t
\end{aligned}
$$

In Figure 2, all flocking parameters are shown that the user is encouraged to alter. Similarly to Reynolds' later extensions (Reynolds (2000)), we give the user the ability to introduce novel agents, to change their FOV and flocking weights on the fly and, thus, to interactively guide the emerging flocks.

Swarm Grammar (Re-)Production

Originally, a swarm grammar $SG = (SL, \Delta)$ was conceived as a combination of a rewrite system $SL = (\alpha, P)$ and a set of agent specifications $\Delta = \{\Delta_{a_1}, \Delta_{a_2}, ...\Delta_{a_n}\}$ for n types of agents a_i (von Mammen and Jacob (2009)). The rewrite system SL loosely followed the concept of an L-system with axiom α and production rules P, as described by Prusinkiewicz and Lindenmayer (1990). In the simplest

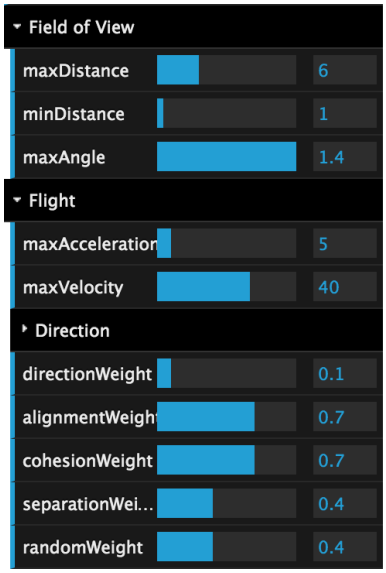


Figure 2: Boid paramters as displayed to the user and offered for alteration. Changes to the Field of View parameters are immediately reflected by the neighbourhood visualisation of the introspected agent.

form of context-free 0L-systems, each rule has the form $p \rightarrow s$, where $p \in \Omega$ is a single symbol over an alphabet Ω, and $s \in \Omega^*$ is a word over Ω. The application of the replacement rule can be conditional, for instance upon a successful stochastic experiment (with specified probability θ) or repeatedly over time (with a specified time period Δt).

Agent specifications may include the flocking parameters described above such as the agents' FOVs and urge weights, as well as characteristic parameters of the geometrical objects they leave behind during their flight. Figure 3(a) shows a representative artefact produced by an early swarm grammar, emphasising the branching structure emerging from the reproduction rules $SL = \{A, \{A \rightarrow BBB, B \rightarrow A\}\}$ in combination with a moderate separation urge.

Later, the rule representation of Swarm Grammars was extended towards quantitative stigmergy (von Mammen and Jacob (2008a)), which (a) allowed to trigger (re-)production, in case a specific environment is perceived and (b) to utilise various static building blocks as well as agent progeny as product. Hence, spatial structures became the outcome of the agents' behavioural interactions rather than simply tracking their flight. A structure built by an extended SG is shown in Figure 3(b).

Interactive Simulation Concept

Our application offers several user interaction mechanisms that support the population and configuration of the simulation space. In order to keep the user interface free from clutter, we decided to omit certain functionalities altogether,

(a) (b)

Figure 3: (a) An early SG definition emphasising branching (von Mammen and Jacob (2009)). (b) Artefact built by an extended, stigmergic SG (von Mammen et al. (2009)).

such as the rotation of geometric bodies, and we stripped the UI of any data that would not immediate benefit the target audience, such as the bodies' coordinates. At the backend, too, we pursued a simple but still ambitious management of agent specifications and geometric templates.

Basic Scene Manipulation

Figure 4 shows a close-up of the top-left corner of the main screen. Here, the user can start and stop the simulation. In the pull-down 'templates' menu an agent specification or a static geometry can be chosen to populate the simulation space. Clicking on any object in the scene (initially, there is the ground) places the selected template on top—in this way, the user can stack objects and populate in all three dimensions (Figure 5). Click and drag of an object moves it parallel to the ground. Hovering above an object and pressing the minus key removes an object (we found that the delete or backspace key is frequently assigned to other tasks in standard internet browsers).

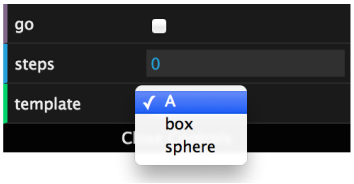


Figure 4: The menu of the simulation's main screen. The simulation process can be started, feedback about the current simulation step is provided and a template can be selected to populate the simulation scene.

Right-clicking an object exposes its properties, as seen in Figure 5, and allows the user to change them. Property changes apply to all objects of the same name, which is shown as the top-most entry in the introspection menu to

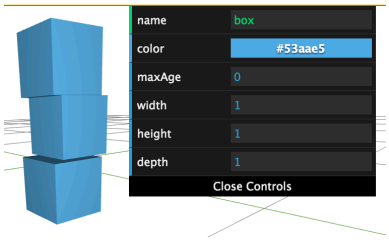


Figure 5: Templates are placed on top of any clicked, existing objects. Properties of objects in the scene can be introspected and change on right click; an according menu appears in the upper-right corner of the screen.

the right. An alteration of a name triggers the creation of an according, new template in the 'templates' drop-down menu (Figure 4). In this way, the user can create a diverse set of static geometric objects and agent specifications.

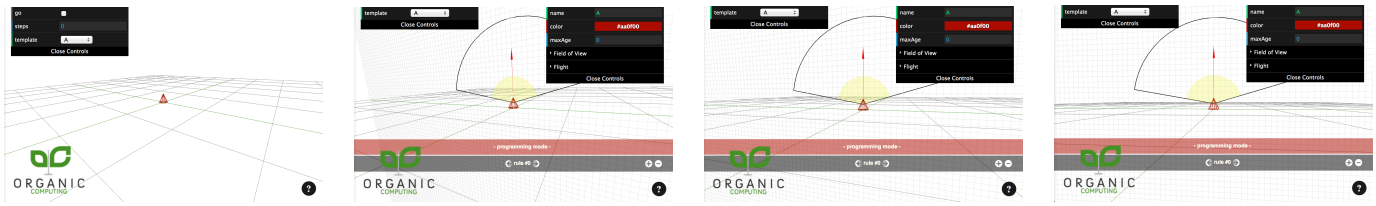
Boid flocking parameters, including the field of view, and the (re-)production rules are part of an agent specification. As we offer a visual programming interface to configure some of these properties, the camera positions itself at a predefined distance from the agent when introspected. The dolly animation closing in on an introspected agent is shown in Figure 6.

Rule Editor

During introspection of an agent specification, alterations of the field of view parameters, d_{min}, d_{max}, α, result in an updated visual representation. This immediate reflection helps the user relate the parameter values to actual geometric dimensions and to quickly grasp the variables' relationships. Yet, the visualisation of the field of view plays another, more important role.

Figure 7 shows the view of an introspected agent. Production rules can be specified directly in the vicinity of the agent. In particular, dice, timers, and and arrow-enclosed timers can be dropped, which represent the abstract rule conditions and associate probability θ, point in time t, and time interval Δt variables, respectively. Platonic solids and agents that are placed outside of the introspected agent's FOV are products of a behavioural rule, those inside are considered quantitative, stigmergic conditions. Any such stigmergic conditions are fulfilled, if the according number of objects is perceived by the agent in the neighbourhood or in the personal space, respectively. For production objects, the minute geometric offset from the introspected agent is taken into account. Their displacement as expressed visually in the rule determines their relative placement in the simulation, which resolves the issue of potentially conflicting product placements.

As a consequence of the semantics associated with differently located platonic solids and agents, their placement and movement are diligently tracked and registered. The red bar



(a) (b) (c) (d)

Figure 6: (a) A Swarm Grammar agent placed on the ground. When right-clicked, the camera automatically positions itself at a predefined distance (b-d).

towards the bottom of the screen provides feedback about the user's programming efforts and any corresponding rule-affecting changes. The grey bar below provides the user with information about the currently displayed rule, to create new rules, to remove existing ones and to browse the complete set of rules of the introspected agent (and its namesakes).

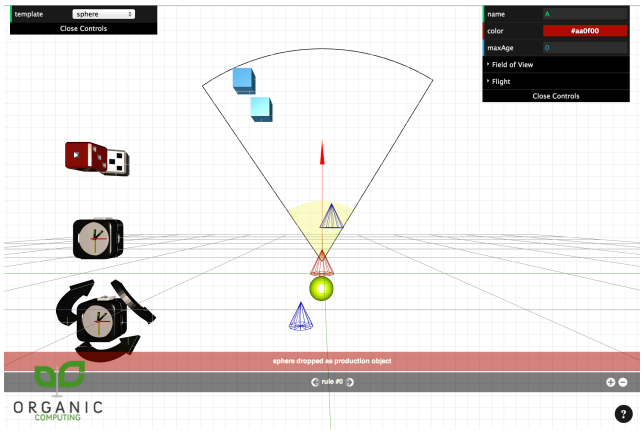


Figure 7: Introspecting an agent specification, sets of production rules may be visually programmed. Abstract conditions, stigmergic conditions, and products of the rules can be placed in the local vicinity of the agents.

Preliminary Feedback & Example Outcomes

In this section, we introduce the circumstances of the first deployment of the interactive simulation presented in this paper, *Swarm Grammars GD*. We provide details on the preliminary feedback we have gathered and we give examples of the artefacts built in this context.

Girls-in-STEM Programme

The concept was conceived while developing contributions to an entertaining and informative programme that aims at encouraging girls to develop and follow their passion for Science, Technology, Engineering and Mathematics (STEM) at the Faculty of Applied Computer Science at the University of Augsburg, Germany. As part of this programme, four groups of roughly ten girls between the ages twelve to fifteen attend four slots of 45 minutes each, offering different contents and activities: 'Autonomous Vehicles', 'Sight-Finder', 'Touch-Robots', and our entry 'Artificial Life'.

Aspects of Artificial Life Research

We identified the following aspects of Artificial Life research that our implementation makes accessible to interested novices in a playful manner.

Similarly to popular simulation environments such as NetLogo (Wilensky and CCL at Northwestern University (2014)), *Swarm Grammars GD* promotes an agent-based modelling approach. More specifically, it provides direct access, often supported by visual cues and interactive elements, to the parametric properties and sets of "if-then"-rules that describe the behaviour of deterministically or stochastically acting, spatially interacting reactive agents (Wooldridge (2009)). When occurring in greater numbers, the interaction of such agents may result in complex feedback cycles, which in turn might lead to emergent phenomena, such as flocking dynamics (von Mammen and Jacob (2008b)) or complex built constructions (Bonabeau et al. (1999)). The means to directly program an individual agent or to simultaneously modify all agents of a certain type allows one to observe and experiment with the relationship between local behaviours and such global emergent patterns, as for instance portrayed by Johnson (2001).

As described in the section on Related Work, the interaction mechanisms that individual agents can perform in *Swarm Grammars GD* are limited to neighbourhood-dependent boid flocking (Reynolds (1987)) and rule-based production as in advanced swarm grammar concepts (von Mammen and Jacob (2009)). Both can be considered concrete concepts of two important Artificial Life themes, namely *collective locomotion* and *developmental models*. We make the first theme accessible by offering the means to visually program an agent's perceptional abilities. It is further promoted as simple construction rules that merely place single three-dimensional objects behind the agents at each simulated step effectively trace the resulting flight dy-

namics, as for instance seen in Figure 8(a).

Regarding the latter theme, *developmental models*, *Swarm Grammars GD* touches on the aspects of production, reproduction and differentiation, whereas these processes are triggered by the agents' internal states, effected by timers and stochastic experiments, as well as external stimuli (see our explanations of the visual rule editor above). Motivating differentiation based on locally perceived stimuli, such as the presence of a specific construction template or of a peer of a specific type, enables modelling a mechanism similar to task assignment in social insect societies (Camazine et al. (2003)). Stimulus-dependent construction efforts, on the other hand, allow one to implement sigmergic lines of communication (Grassé (1959)), i.e. indirect communication through the environment. All elements in *Swarm Grammars GD*, whether they are agents or built objects, have a lifespan attribute which determines the respective element's timely removal from the simulation (the elements are not removed, if this attribute is set to the value 0). Utilising such a timed appearance in combination with stigmergic construction and boid-based cohesion, a simplistic model of Ant Colony Optimisation can be retraced Dorigo (2007).

Presentation Sequence

Despite the intricate modelling options *Swarm Grammars GD* provide the user with, in the context of a short introduction to young novices, we directed our introductory STEM session to Artificial Life to the foundations of agent-based programming and discussed the intuitively accessible basics of swarm dynamics, construction and reproduction. After brief examples of L-Systems (Prusinkiewicz and Lindenmayer (1990)) and boids (Reynolds (1987)) and their utilisation as special effect techniques by the movie industry (schematic slides and movie snippets), we introduced *Swarm Grammars SG* hands-on. Following the structure of the above section 'Interactive Simulation Concept', we first explained the main view of the simulation space and basic user interactions to populate it. Next, we briefly demonstrated the exploration potential merely arising from altering various boid parameters. Finally, we detailed the composition of production rules (in this order: producing static geometries, initialising other agent specifications, and adding conditions to the rules). At this point, each group had approximately 25 to 30 minutes at its disposal for exploring and design artificial artefacts and swarm dynamics, under guidance and with feedback if desired. Our offer to print out screenshots of the individually generated artefacts was in good demand, we handed out 24 of them.

Leeway for Improvement

Some weaknesses of the current simulation became obvious during the supervised sessions—especially with respect to choosing reasonable parameter values, including boid urge weights, and configuring abstract production rule conditions such as chance or time steps. One could mitigate the issue of conflicting or ineffective boid parameter sets by offering several presets such as the ones evolved by Kwong and Jacob (2003). Regarding conditional values, if one does not want to drastically limit the expressiveness of rule compositions, warnings could be issued that hint at potentially unreasonable parameters. For instance, agent multiplication at high frequencies quickly exhausts the host computer, if the maximal life span of the agents is relatively high.

While exploring, one student asked how one could program the cubes (as opposed to the agents) to reproduce themselves. This question made clear that the distinction between producing agents and static geometries is arbitrary, not necessary, and possibly not even beneficial for the sake of functional distinction. At least one could expand the computational representation and nullify this rigid distinction. Other, more frequently asked questions were related to increasing the diversity of templates: although creating new templates by entering new names was quickly understood, this mechanism seemed to be too lengthy for supporting the creative diversity in colour and scale some users would have liked to deploy.

Despite the shortcomings of the current implementation, twelve out of a total of 42 participants declared our session and the use of *Swarm Grammars GD* to be the highlight of the whole introductory programme.

Example Artefacts

Figure 8 shows an array of six different Swarm Grammar artefacts programmed by participants of our session. We can identify different classes of structural complexity based on the flocking and production rule complexities of the SG agents. With only minor changes to the default boid flocking parameters (see Figure 2) and continuously dropping geometries, swarm motion is captured by the built artefacts (Figures 8 (a-c)). Agents with distinct construction behaviours—either resulting from differentiated reproduction or from interactively adjusting agent specifications—yield more visually complex structures (Figures 8(d-f)).

Perspective Shots

Unfortunately, as the students did not have much time to explore, play and create, their artefacts were mostly captured from a global perspective, which usually does not emphasise their appealing peculiarities. Figure 9 displays several Swarm Grammar configurations, snapshots of the emergent generative processes and close-ups of the final products set in scene.

In particular, three different Swarm Grammars are shown. The first one, with captions subtitled *cloud*, works based on a simple, unconditional production rule that traces an upwards-flocking agent with a cluster of spheres. The twists and turns triggered by the interplay of several flocking

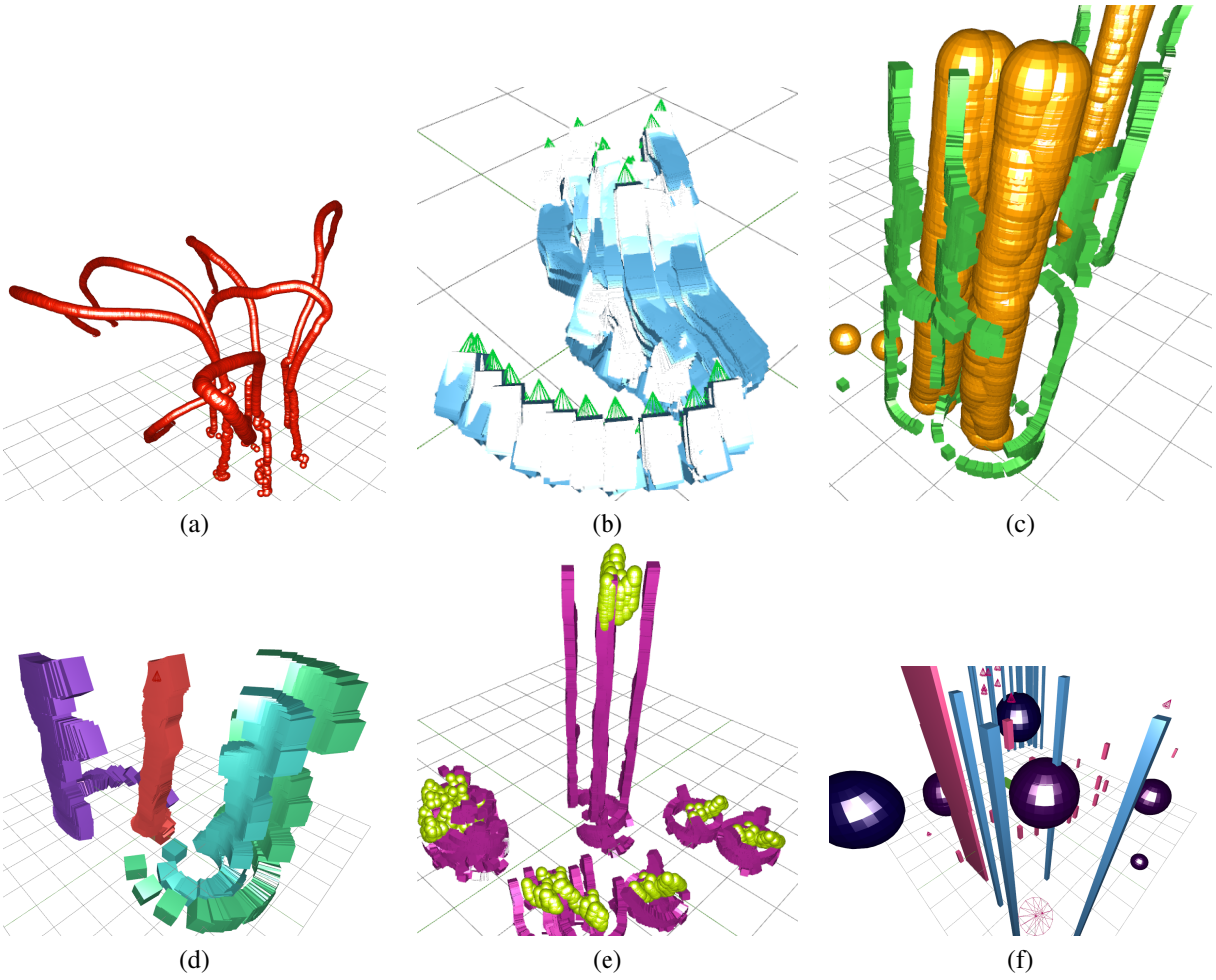


Figure 8: (a-b) Flocking SG agents leaving traces of one platonic solid (red spheres and blue cubes, respectively). (c) Individuals with a strong directional upwards urge leave a trail of two solids, golden spheres and offset green cubes. (d-f) More intricate and diverse structures emerge from building efforts by heterogeneous agent sets.

agents (as seen in Figure 9(b)) are exalted in the close-up by an upwards perspective and light shading. The second example deploys two agents, the first one simply flies upwards, repeatedly creating an offspring of a different kind (rule^0_{wall}). The latter one is pushing hard to the right while continuously dropping cubic solids (rule^1_{wall}) but it is also distracted by its neighbours and thrown off its path by some randomness. The resulting, aligned traces pave a solid uneven wall (close-up_{wall}); In the third example, a single agent is equipped with two rules, one to establish a continuous trace (rule^0_{tree}) and the second one triggering periodic branching to two sides (rule^1_{wall}). The repeated branching process (process_{tree}) yields a tree-like structure (close-up_{tree}).

Conclusion

In this paper, we presented an interactive Swarm Grammar simulation. It has been conceptualised and implemented in order to engage a young audience in Artificial Life concepts, namely swarm dynamics and developmental processes. The simulation attempts to intrinsically motivate the users by keeping the learning-curve as low as possible. At the same time, we challenge the users' competence by attaining a relatively expressive programmable representation, including boid flocking behaviour as well as (re-)production behaviour of Swarm Grammars. The gap between expressive representation and simplicity is bridged by means of visual programming interfaces for configuring the simulation space as well as individual agent behaviours.

We exhibited some of the artefacts designed by a number of high-school students at the age of twelve to fifteen. We suggested several possible improvements to the software based on feedback by the students but also based on observations during supervised hands-on sessions with the simulation. We complemented the display of the students' works by three additional Swarm Grammar examples that explicitly rely on (a) multiple construction elements in single rules, (b) differentiated reproduction, and (c) branching production rules.

In order to further the presented work, we suggest to translate all boid parameters into meaningful interactive visuals. For instance, the line-width of arrows representing various flocking urges could stand for the according, relative weights. Field of View and other visually represented parameters should be readily manipulatable, and not only be altered by means of textual GUIs. The maximal Age of an agent could be visualised by projecting a faded out geometry along its current trajectory. Also, the composition of (re-)production rules could be improved by relating them to the actual environment: the production objects could be projected on top of the actual simulation environment in order to facilitate precise definitions of environmental alterations.

Programmatically, we suggest switching from the current, class-based architecture to a component-based perspective that allows to aggregate behaviours. This would simplify the template management and provide the flexibility to assign behaviours to arbitrary objects, as asked for by the students.

References

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, New York.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2003). *Self-Organization in Biological Systems*. Princeton Studies in Complexity. Princeton University Press, Princeton.

Dorigo, M. (2007). Ant colony optimization. *Scholarpedia*, 2(3):1461.

Grassé, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–80.

Johnson, S. (2001). *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. Scribner, New York.

Koster, R. (2013). *Theory of fun for game design*. O'Reilly Media, Inc.

Kwong, H. and Jacob, C. (2003). Evolutionary exploration of dynamic swarm behaviour. In *Congress on Evolutionary Computation*, Canberra, Australia. IEEE Press.

Prusinkiewicz, P. and Lindenmayer, A. (1990). *The Algorithmic Beauty of Plants*. Springer, New York.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34.

Reynolds, C. W. (2000). Interaction with groups of autonomous characters. In *Game Developers Conference*, pages 449–460.

von Mammen, S. and Jacob, C. (2008a). Evolutionary swarm design of architectural idea models. In *Genetic and Evolutionary Computation Conference (GECCO) 2008*, pages 143–150, Atlanta, USA. ACM Press.

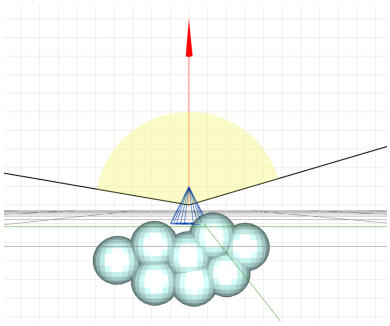
von Mammen, S. and Jacob, C. (2008b). The spatiality of swarms — quantitative analysis of dynamic interaction networks. In *Proceedings of Artificial Life XI*, pages 662–669, Winchester, UK. MIT Press.

von Mammen, S. and Jacob, C. (2009). The evolution of swarm grammars: Growing trees, crafting art and bottom-up design. *IEEE Computational Intelligence Magazine*, 4:10–19.

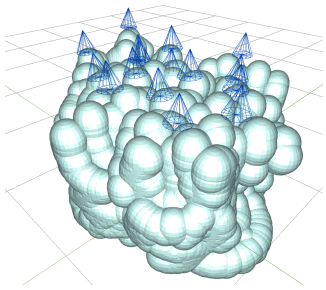
von Mammen, S., Novakowski, S., Hushlak, G., and Jacob, C. (2009). Evolutionary swarm design: How can swarm-based systems help to generate and evaluate designs? *DESIGN Principles & Practices: An International Journal*, 3:371–386.

Wilensky, U. and CCL at Northwestern University (2014). Netlogo: A cross-platform multi-agent programmable modeling environment. `http://ccl.northwestern.edu/netlogo/`.

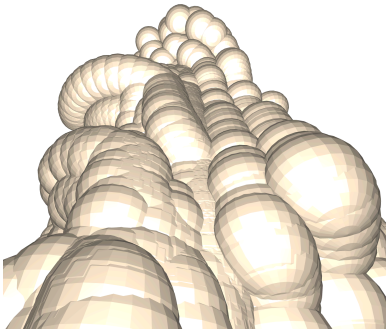
Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems*. John Wiley and Sons Ltd, West Sussex, UK, 2nd edition.



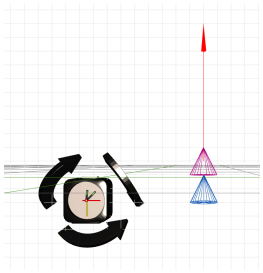
(a) rule$_{cloud}$



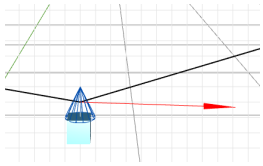
(b) process$_{cloud}$



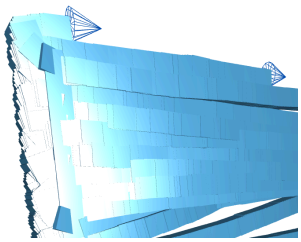
(c) close-up$_{cloud}$



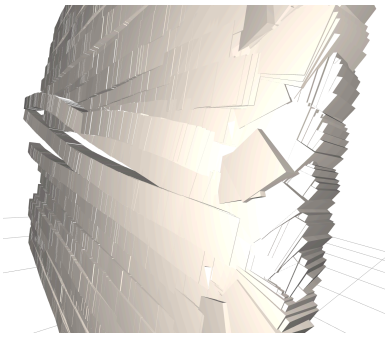
(d) rule$^0_{wall}$



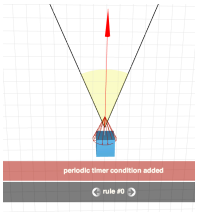
(e) rule$^1_{wall}$



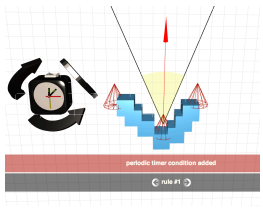
(f) process$_{wall}$



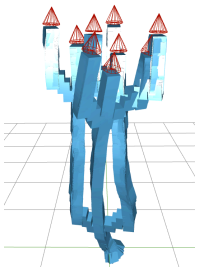
(g) close-up$_{wall}$



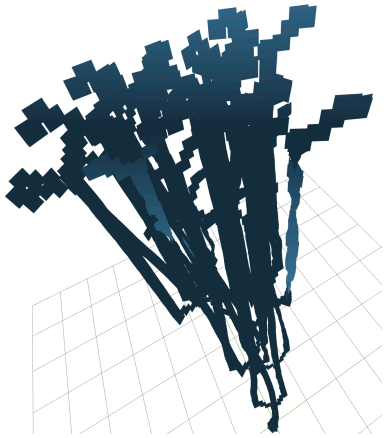
(h) rule$^0_{tree}$



(i) rule$^1_{tree}$



(j) process$_{tree}$



(k) close-up$_{tree}$

Figure 9: (a) An unconditional multi-solids production rule to trace an upwards-flocking agent. (b) A snapshot of the resulting developmental process, given a small set of initial agents. (c) A close-up of the final artefact. (d) An agent that flies upwards without considering any distractions and periodically produces (e), an agent heading to the right and leaving a cubic-trail behind. Its trajectory is influenced by its neighbours and chance. (f) The resulting developmental process, and (g) the final artefact close-up. (h) A simple trail production rule, combined with (i) a periodic branching rule, resulting in a (j) branching developmental process. (k) The final artefact set in scene with a pixelated 2D style.