ABSTRACT
A shape grammar is a collection of visually defined geometric rules that could be used to automate the generation of formal representations of designs for buildings, cities, products and more. We offer an extension of the shape grammar formalism based entirely on voxel space instead of vectors, which we used for the generation of schematic architectural designs. We describe a method using playability to increase human agency and designer control over the outcome of the generative phase of voxel-shape grammars. The method is presented with an implementation in the environment of Minecraft and employs three guidance mechanisms. To conclude we list a few considerations from our experience in the design of a playable, voxel-shape grammar and point to future work.
INTRODUCTION
The ability of shape grammar generative formalism to capture design intentions and expert knowledge is what interests us in the pursuit of engaging non-experts in the architectural design process. We base our work on this large field of research and present a novel extension to shape grammars, namely playable, voxel-shape grammars that could have an enabling effect upon both experts and non-experts by letting them interactively and immersively explore design options and learn by making. With our shape grammar method we target use scenarios in the early schematic design phase, when all stakeholders explore the relationship between the elements of a project in an informal and loose manner, often in the form of massing models.

From very early on in the work we aimed to:

1. Offer user-friendly visual modelling of both the shapes in the shape grammar, as well as the grammar rules, in order to afford the expert users easy access to the tooling framework—this was possible by redefining the typically vector-based shape grammars for a voxel space and using Minecraft as the implementation environment.

2. Open the generative process to human agency in an interactive, manageable and surmountable way—by making the grammar playable, we could add direct participation and guiding mechanisms to the otherwise purely algorithmic generative process.

We developed and tested an extendable framework implementing playable, voxel-shape grammars in the Minecraft environment. This paper presents:

- a definition of playable, voxel-shape grammars (Section 3.1);
- test cases at urban and building scales (Section 4);
- three types of guiding mechanisms that experts could use to control the play-driven generative process (Section 5.3).

PROBLEM STATEMENT
Shape grammars have been successfully used to encode aspects of expert architectural knowledge, and thus help their users, mostly designers, to explore a large set of design solutions or quickly generate two-dimensional representations of designs with desired formal qualities (Stiny and Gips 1972; Stiny and Mitchell 1978; Chase 2002). The application of shape grammars consists of mostly two activities:

1. designing the grammar, which consists of shapes and rules;
2. iterating multiple times through the rules to generate designs.

The second activity maintains the usefulness of shape grammars as a design tool, subject to their implementation as automated systems run by a computer (Ruiz-Montiel et al. 2013). However, shape grammars have been mostly vector-based (Stiny 1980; Woodbury 2016), and the shapes generated with grammars are only notational graphs or abstract 2D compositions. (Martin 2006; Stiny and Mitchell 1978; Duarte 2005). This poses two main challenges in the attempt to use them as design tools:

1. computer implementations of shape grammars are difficult to define;
2. computer-automated generation of shapes with shape grammars is difficult to explore and control.

Computer Implementations of Shape Grammars are Difficult to Define
An important aspect of a successful shape grammar implementation is that the users are able to model the shapes and rules themselves.

Usually, shape grammars have been defined in a closed, expert coding environment with an interface defined solely for the purposes of the shape grammar. Therefore, a potential new user of the shape grammar cannot rely on computer skills they already have, and must learn new, advanced ones. The use of generic, accessible and easy-to-use 3D-modelling environments to implement shape grammars has not been well explored. The following 18-year-old quote from Terry Knight states the problem rather well, and unfortunately not much has happened since then to address it:

Currently though, there are few computer implementations that are practicable for students or practitioners. Most do not have interfaces that make them easy for non-programmers to use. More efforts have gone to computational problems than to interface ones. Implementations of simple, restricted grammars that are visual and require only graphic, nonsymbolic, nonnumerical input are needed. (Knight 1999)

We address this by implementing our method for playable, voxel-shape grammars in a popular and accessible 3D environment with more than 40 million worldwide users of all ages and backgrounds: the game Minecraft. The size of a user’s Minecraft avatar in relation to the voxel shapes they model—and subsequently iterate through in the game world—creates an immersive perception for a one-to-one architectural scale.

Computer-Automated Generation of Shapes with Shape Grammars is Difficult to Explore and Control
Most existing research on shape grammars uses a computer-automation approach to the generative phase, leaving no control, or very little, mostly global control, to the designer during the
process of shape generation. This has been done “as pencil-and-paper execution might be tedious and therefore useless for the sake of discovering many new solutions.” (Ruiz-Montiel et al. 2013). However, an automated generation process is most often opaque and non-iterable—the user cannot influence the result while the process runs and is simply presented with an outcome. Our method uses a combination of playability and guiding mechanisms to make the iteration process more navigable.

A METHOD FOR USING PLAYABLE VOXEL-SHAPE GRAMMARS

Voxel shape

George Stiny (1980) defines a vector shape as follows: “A shape is a limited arrangement of straight lines defined in a cartesian coordinate system with real axes and an associated euclidean metric.” In a similar manner let us define voxel shape as a limited arrangement of voxels in a discrete voxel grid and an associated euclidian metric (Figure 3).

A voxel-shape will define all further terms we find in Stiny’s “Introduction to Shape and Shape Grammars” (1980), but for the discrete voxel space instead of continuous cartesian vector space. For example, a scale operation on a voxel shape needs to have an integer number for the scale factor—for example 2x—so that it turns one voxel into a cube of eight voxels: 2x2x2. Rotations would go only in 90 degree increments and so on. A less strict definition, allowing real number scale factors and rotation, is also possible, but it would need a post-processing voxel detailing routine, such as marching cubes, and thus introduce unnecessary complexity for the current purposes of the method (Marais and Crumley 2012).

The original shape grammars formalism uses vectors that are simple representations of architectural elements such as walls. Because of its 3D nature, the proposed voxel-shape grammar could be used to represent both elements (walls, slabs) as well as massing volumes of a building’s or city’s elements.

An important aspect of shape grammars is the labelled shape, consisting of shape and symbols, used as a matching mask for the grammar rules. The marker symbols in vector space shapes are notational. In our voxel shapes we use voxels with special attributes for markers (Figure 3).

Voxel-Shape grammars

Shape grammars perform computations with shapes in two steps: a recognition of a particular shape and its possible replacement. Rules specify the particular shapes to be replaced and the
Sample playable voxel-shape grammar (left: rules, right: initial shape)—besides the voxels’ state, the rules take into account also the player’s position.

Generation of a voxel shape using the playable voxel-shape grammar shown in Figure 5.

A guiding mechanism provides a desirable advantage to the players for their next moves and thus could influence their decisions.
The urban scale voxel-shape grammar from IBA_GAME has rules that place pre-designed buildings. A schematic design for a residential building created using a voxel-shape grammar with 6 rules.

A shape grammar is defined by rules and an initial shape, and is in essence a language of shapes. Shape grammar rules consist of a mask shape on the left and a replacing shape on the right (Figure 2).

For the shape rules’ implementation in voxel space we define a voxel shape as a mask, containing at least one marker voxel and a replacing voxel shape (Figure 4). A voxel-space 3D version of the 2D vector-shape grammar from Figure 2 is seen on Figure 4.

The use of shape grammars in voxel space has only been previously explored on two occasions. Marais and Crumley (2012) presented an extension to shape grammars called voxel-space shape grammars, in which the shapes and rules are defined in vector form and only the generated shapes—after the iteration is complete—are voxelized in a post-processing step. A more appropriate naming for their method would be voxelized, vector-shape grammars. Our method is therefore still novel, as the entire grammar is defined and iterated in voxel space.

Friedman and Stamos (2013) introduce the notion of a voxel grammar in their work on procedural tree generation. However, their grammar definition is not visual but numerical, and as such, lacks many of the advantages of the original shape grammar formalism which “… allows for algorithms to be defined directly in terms of […] shapes” (Stiny 1980). Our implementation here suggests that the entire voxel-shape grammar (shapes and rules) is defined in a visual, user-friendly way.

**Playable Voxel-Shape Grammars**

As such, voxel-shape grammars would be easy to model but would have all the problems of shape grammars defined in section 2.

To address the control and ease-of-use issue described in 2.2, we introduce human agency in the form of game mechanics. It adds one additional step in the generative iteration of the grammar rules—a player needs to place some of the markers in order to complete a mask shape found in the rules (Figure 5). Besides the voxel state, the rules in the playable version of voxel-shape grammars also take into account the player position.

Differing from shape grammars, where the generation is automated and each step of the iteration produces the conditions for the next, playable voxel-shape grammars require that every iteration needs to be initiated by a player, thus making conscious choice (i.e., a design decision) an important part of the exploration process (Figures 1, 5, and 6).
On the other hand, player decisions could be influenced by a guidance mechanism. A guiding mechanism is defined by the author of the grammar and is able to encode a design intention or expert knowledge. A well-defined guiding mechanism would be integrated in the game and would in essence be a means to provide a desirable advantage to the players for their next moves, and thus could influence their decisions (Figure 7).

EXPERIMENTS AND CASE STUDIES
We are testing the concept of playable voxel-shape grammars in the project 20.000 BLOCKS. Our implementation consists of a Minecraft version with the ComputerCraft mod and a set of custom scripts which drive rule detection, player/goal tracking and results export. See Savov, Buckton, and Tessmann (2016) and www.20000blocks.com for further details on the project.

We investigated the use of playable voxel-shape grammars at two scales. At the urban scale the shapes in the grammar represent pre-designed whole buildings (or mass models of buildings), and the rules represent the possibilities for these buildings to exist next to each other in a city. We tested this approach between August 2016 and March 2017 in the IBA_GAME—a 20.000 BLOCKS version that we created for IBA Heidelberg, where players can create small neighbourhoods. Hundreds of these neighbourhoods form a new quarter for the German city of Heidelberg. The game featured a voxel-shape grammar of 40 buildings of six types: houses; housing extensions; businesses; science and tech spaces; gathering spaces; and public plazas (Figure 8). It had 140 players who played it 820 times. The shape grammar rules went through about 10 major revisions in the course of development.

At the building scale the shapes in the grammar could represent physical elements of a building, such as walls and slabs or the massing of the volume a room or an area would take in the building, while the rules represent the logic for those elements to aggregate next to each other to form a building. To test this approach we ran a SmartGeometry workshop in April 2016 focusing on residential, multi-apartment buildings. Three teams of architects and engineers used our implementation of playable voxel-shape grammars and defined components of the buildings within the grammar (Figure 9). Each variation had up to 16 shape rules and was played by attendees of the conference.

MINECRAFT IMPLEMENTATION OF PLAYABLE VOXEL-SHAPE GRAMMARS
Based on our two-year experience with this ongoing project, we offer the reader the following thoughts on voxel shapes, voxel-shape rules and guidance mechanisms.

Features of the Voxel-Shapes
Design
The abstracted modelling environment with a voxel size of 1 x 1 x 1 m could be limiting. However, since our aim is to generate schematic architectural designs, this level of abstractness works in our favour.

Walkability
An important requirement in order to achieve an immersive experience in the generative phase—i.e., in the play phase—is that the resulting structures are walkable for players, so they can reach the newly added markers, otherwise the iteration process is blocked. That means that in both urban or building scales, all
possible sequences of the grammar’s shape rules need to create a walkable combination when placed.

**Color Coding**
The points of possible further grammar growth need to be very clearly visible to the player while in play. Therefore, the textures of the marker voxels and the building blocks need to be different. Furthermore, with larger sets of rules, such as in IBA_GAME, where we used six types of rules with 2–4 variations each, a color coding of the types of rules helps the player learn the grammar faster.

**Rotation Options**
As we are operating in a strict orthogonal voxel grid, we had four possible rotations for each shape rule in the grammar. We found out that modeling them to be the same in behavior, but only similar in looks, decreases the rigidity of designs and introduces visual complexity (Figure 10).

Thus far, we have not used scaling or rotating of the shapes, i.e., the masking shapes that are modeled in the grammar rules are absolute in size and orientation. Due to the discrete quality of a voxel space, scaling and rotation in free increments could be detrimental to the readability of a shape’s design.

**Features of the Rule System**

**Markers Placed by the Grammar**
We used two types of marker voxels:

1. the activator block on which a player steps to activate the detection of a masking shape they had built around it (Figure 5);
2. the extension block, which is a marker that replaces the activator block upon successful detection.

Some of the grammar rules require the extension block in their masking shapes. The extension block ensures that structures grow out of themselves coherently instead of populating the design space with too many scattered, independent structures. That means some shapes in the grammar act as seeds, for example the shape representing a house in IBA_GAME, and others as extensions, for example the shapes representing commercial buildings in IBA_GAME (Figure 8).

**Markers Placed by the Player**
Player-placed marker voxels, i.e., resources, are in essence the material with which the player “draws” or models the masking shape. For example in IBA_GAME, we used two resources that encode a set of meta design features into the voxel shapes:

1. an urban resource -> shapes are more solid, with vertically proportioned windows and enclosed;
2. a green resource -> shapes are lighter, with horizontally proportioned windows and more open (Figure 11).

When using both types of resources to build the same masking shape, the player would expect the same type of resulting shape, with the same behavior during the iteration process, but with different design qualities.

**Markers in a Grid or Freely Spaced**
When designing the playable voxel-shape grammar, the expert needs to make sure that all rules are spatially compatible with each other and create a walkable design. Walkability can be obstructed by unforeseen shape overlaps.

One strategy we enforced in the rules is growth along, or within, the constraints of a three-dimensional grid. The markers were always placed in a shape according to a grid of 9 x 9 x 8 blocks (Figure 10). We also developed an in-house design style guide.
to regulate, across our team, how and when sections of a shape rule could overlap with another (Figure 12). This saved the work of having to try all combinations separately.

Another strategy, which avoids the rigidity of using a grid, is for the shapes to come with the markers in a position that allowed for the making of stepped structures and richer designs in spatial terms. To avoid too many unwanted overlaps, new structures only grow from existing ones (seeds), to allow us to detect rule adjacency (Figure 9).

Features of the Guidance Mechanisms
We implemented playability in the form of an economy system: if a player owns the resources and the territory needed to create a masking shape, then they could activate a grammar rule with it and so gain the rewards given by the rule. We used playability as a generative force and at the same time as a control mechanism over the outcome. Ruiz-Montiel et al. (2013) make the case for control mechanism as means to guarantee feasible designs.

For the designer to be able to encode a design intention into the playable voxel-shape grammars, we tried three types of guidance mechanisms: a quantitative game-goal system; limiting choice with the shape design; and performance-based feedback.

Quantitative Guidance Mechanism
We used a soft goal system that qualitatively guided the growth as different players chose to go for different goals. For example in IBA_GAME we defined four server-wide goals: collect the most green points, build the most buildings, build the tallest structure, build as many different buildings as possible (Figure 14). Each voxel-shape rule in the grammar rewards the player differently on all four metrics.

Limiting Choice with the Shape’s Design
If players pursue a soft goal such as height, then they will aim to build as high as possible with as little effort and expense as possible. To prevent the stacking of one and the same house on top of itself, we designed the roof of the house so that players cannot model the masking shape for a new house on top of it (Figure 13).

Performative Guidance Mechanism
While still in the early works in terms of seamless integration, we tested a guiding system based on structural analysis. It exports the model to Grasshopper every time a rule is activated, runs an analysis routine on it, and then "reports" back the results to the players by color-coding certain blocks of the structure (Savov, Buckton, Tessmann 2016). We used a similar approach in the project Sensitive Assembly, documented in Savov, Tessmann, and Nielsen (2016), where this type of feedback loop is described in more detail.

CONCLUSIONS AND FUTURE WORK
We presented an extension to the shape grammar formalism for voxel spaces, which we used for the generation of schematic architectural designs. We described a method using playability to increase human agency and designer control over the outcome of the generative phase of voxel-shape grammars. At the end we shared a few thoughts on the design of a playable voxel-shape grammar in Minecraft.
**Potential Contribution to the Field**

With our method architects can encode architectural logic and principles in the grammars and easily modify them. The grammars are volumetric and thus open to immediate interpretation and analysis for their architectural potential without further transformation or translation steps.

Furthermore, the method allows for the integration of design and fabrication. An additional reason to use voxel-shape grammars instead of vector-shape grammars is to come one degree closer to embedding material specification rules in the grammar (Rossi and Tessmann 2017; Stiny and Gips 1971).

**Challenges**

The method relies on players using the Minecraft map to model grammars and generate structures. Currently we have had 800 plays, 150 play structures, and 150 downloads. There are the following bottlenecks to increasing participation:

1. The need to download and use a customized Minecraft version, which reduces the number of potential players. We are looking for ways to implement the grammar logic entirely on the server side so that players can participate with a Vanilla (unmodified) Minecraft client.
2. Closed server access: currently the data collection requires us to run the server ourselves. We are exploring options to let users run their own Minecraft server with the map and save results in the cloud.

**Future Work**

The research could benefit from the following future developments:

- A generative visualizer, which could simulate a game being played so that a designer can capture some of the most common conflict/bugs in advance, like mismatching shapes and broken walkability, without having to play the game over and over again.
- Implementing automated rules that would not need the user’s input at every generative step. There needs to be a better balance between the effects of the player’s intentions and the generative rules over the final designs. To address this, in some cases, when the right conditions occur, the grammar rules might allow for several iteration steps to happen automatically one after the other, in the form of a bonus effect or what Sanchez (2015) defines as synergies.
- The collected data (grammar definitions plus the generated shapes) can be used to train a machine learning algorithm to generate either grammar sets or schematic designs.

The test game can be downloaded at: www.20000blocks.com.

**ACKNOWLEDGEMENTS**

The project 20,000 BLOCKS is led by Anton Savov and developed at the DDU (Digital Design Unit, Prof. Dr. Oliver Tessmann) at the architectural faculty of Technische Universität Darmstadt with the following team: Steffen Bisswanger, Ben Buckton, Theo Gruner, Jörg Hartmann, Thomas Valentin Klink, Sebastian Kotterer, Marios Messios, Lorena Müller, Max Rudolph, Alexander Stefas, Alban Voss. The IBA_GAME version of 20,000 BLOCKS was made possibly through the partnership with IBA Heidelberg and sponsored by Eternit GmbH.

**REFERENCES**


Savov, Anton, Ben Buckton, and Oliver Tessmann. 2016. “20,000 Blocks.”


**IMAGE CREDITS**

Figure 2: George Stiny, 1980

All other drawings and images by the authors.

Anton Savov works as a research associate at the Digital Design Unit (DDU) – TU Darmstadt and is the founder of architectural studio AWARE. He initiated the platform “20.000 Blocks” as a means for collaborative architectural design and successfully partnered with IBA Heidelberg to create the IBA_GAME based on it. Previously, Anton has worked at Bollinger+Grohmann Ingenieure and taught at the Städelschule Architecture Class. His work has been exhibited at the Venice Biennale, MAK Center in Los Angeles, NODE Frankfurt and others. If you want to get in touch, email to savov@dg.tu-darmstadt.de

Oliver Tessmann is an architect and Professor for Digital Design at TU Darmstadt. His research and teaching is located in the field of computational design and digital fabrication. After receiving his doctorate from the University of Kassel in 2008, he served as the director of the performative Building Group at Bollinger+Grohmann. From 2010 to 2012 he was Visiting Professor at the Städelschule Architecture Class in Frankfurt and, from 2012 to 2015, held the position of Assistant Professor at KTH Stockholm.