A Bricklayer-Tech Report

VICTOR WINTER
Department of Computer Science
University of Nebraska-Omaha
PKI 174C, 1110 South 67^{th} Street
Omaha Nebraska, USA

HUBERT HICKMAN
Topy Creek Ventures, LLC
Omaha Nebraska, USA

ISABELLA WINTER
Winter Code School
Omaha Nebraska, USA

Abstract: Technology is playing an increasingly prominent role in all human endeavors, including education. Tech enables the realization of educational environments that are adaptive, interactive, and immersive. Such environments are well-suited for appropriately engaging student populations comprised of digital natives.
Bricklayer is an educational ecosystem whose focus is on the development of visuospatial, mathematical, and computational abilities foundational to computer science. This article gives an updated report on the core (Ed)Tech elements comprising the Bricklayer educational ecosystem.

Key-Words: computational thinking, mathematical thinking, visual thinking, spatial reasoning, functional programming

Received: February 10, 2021. Revised: July 1, 2021. Accepted: July 13, 2021. Published: July 23, 2021.

1 Introduction

In a 2015 Wall Street Journal article, Christopher Mims wrote "in the future, no profession is untouched by machines". Since then, advances in computer technology have increasingly impacted markets to align with this prediction in ever broadening terms. The imbalance in labor markets between jobs requiring tech skills and personnel qualified to fill those jobs continues to grow.

This article presents evolutionary developments in Bricklayer, an educational tech-based ecosystem created to positively contribute to the solution of the educational challenges articulated in the previous paragraph. Bricklayer is a *low-threshold infinite-ceiling* system designed to teach math and computer science in ways that are engaging as well as technically meaningful. This is accomplished through the integration of a variety of web apps and tools that facilitate a thoughtful and systematic exploration of construction techniques underlying 2- and 3-dimensional block-based art (e.g., pixel art).

As a domain, the visual arts are extremely well suited for the study of foundational ideas underlying both computer science and certain forms of mathematics. Stated in its simplest terms, to leverage the power of the machine, one must be able to create a *small* program (measured in lines of source code) whose execution yields a *large* number of computational steps. In order for this to occur, there must exist a *pattern* in the computational sequence that is captured in the program. Thus, it can be argued that the study of patterns is an integral part of both computer science and mathematics. Visual domains are extremely suitable for both the informal and formal study of patterns. Furthermore, the richness of the visual domain makes it possible for such study to be appropriately tailored to a wide range of audiences (e.g., across the K-16 spectrum).

The rest of this paper is organized as follows. Section 2 outlines Bricklayer's educational goals. Section 4 takes a look at some educational systems that share attributes (e.g., computer science, math, interactive web apps, gamification) with Bricklayer. Section 5 describes advancements to the web apps that lie on the periphery of Bricklayer's educational ecosystem. Section 6 describes advancements to Bricklayer's core. Section 8 gives an overview of efforts to create high-end 3D digital experiences and games integrated with Bricklayer's educational goals. Section 9 describes how Bricklayer's extended library can be used in modeling and simulation. Section 10 concludes.

2 Educational Goals

At the core of the Bricklayer educational ecosystem is a graphical library implemented in the functional programming language SML. This library provides a set of primitives suitable for the construction of 2D and 3D block-based artifacts within a discrete Cartesian coordinate system. The primitives provided include functions to create a variety of geometric objects such as lines, rectangles, rings, circles, spheres, cones, rectangular prisms, and cylinders. Iterators are also provided supporting the property-based con-

struction of artifacts. For example, an 8×8 checkerboard pattern can be created by using an iterator to traverse an 8×8 square, placing a black brick in cells whose coordinates sum to an even number and placing a white brick otherwise. It should be noted that property-based construction of artifacts is extremely powerful and can require non-trivial use of logic.

The rich visual domain accessible through the combination of the Bricklayer library combined with the general-purpose computing capabilities of SML allows for the creation of educational content suitable for a wide range of audiences. The result (by design) is a *low threshold infinite ceiling system*.

2.1 Low Threshold

One of Bricklayer's primary goals (the *low threshold* part of the system) is to provide novices with a gentle introduction to computational thinking, spatial reasoning, and coding. This objective was the basis for choosing a functional programming language in general, and SML, in particular, as the language in which Bricklayer programs would be written. As a language paradigm, functional programming languages have a clean and clear semantics. Functional code is well-suited to algebraic manipulation and other forms of equational reasoning (e.g., program transformation). Novices are not confronted with the need to resolve the mathematical paradox of $x = x + 1$ allowing their mathematical background to be more comprehensively leveraged. In addition, an orthogonal aspect of central importance is that the discussion and exploration of algorithms for creating visual patterns (e.g., tessellations) can oftentimes be easily understood in non-verbal terms. This separates the exploration and understanding of *algorithms* from the challenges associated with their expression in code. In this way, the cognitive load associated with the intersection of these two activities is reduced.

The Bricklayer ecosystem provides extensive scaffolding specifically targeting novice programmers. For novices, Bricklayer programs are written in a subset of SML. Level 1 Bricklayer programs are flat programs in which artifacts are created via a sequence of *put-function* calls. The Level 1 Bricklayer library provides put-functions capable of creating a limited set of rectangle shapes (e.g., 4×2, 2×3, etc.) and position them at locations within a grid. By design, the shapes and colors provided in Level 1 align with standard LEGO bricks. This aligns with tactile (pseudo-code like) activities where students create a physical artifact on a LEGO baseplate and then recreate the artifact in code. For elementary school students, this may be the extent of their engagement with Bricklayer coding. That being said, it should be noted that pixel art can be created using Bricklayer's Level 1 library, and pixel art spans a wide range of complexity. The creation of complex pixel art should be approached in a systematic and disciplined manner, and the difficulty of its construction should not be underestimated.

At present, Bricklayer provides 5 levels of programming with the first programming level (Level 1) described in the previous paragraph. The subset of SML in these levels collectively includes, val-declarations, user-defined curried function declarations, let-blocks, conditional expressions, integer expressions, boolean expressions, relational expressions, tuples, and lists. Using these language constructs, most (but not all) of the artifacts described in section 2.2 can be created.

2.1.1 Scaffolding

Bricklayer's scaffolding technology consists of 6 interactive web apps: Vitruvia, Quill, Lynx, Mystique, the Grid, and BLite. All apps were created in response to observed coding difficulties exhibited by novices spanning the K16 spectrum. Each app develops and strengthens a particular set of skills related to the construction of (text-based) SML programs that create Bricklayer artifacts. Vitruvia introduces Cartesian coordinates and SML language constructs aligned with Bricklayer's programming levels. Vitruvia exercises involve code comprehension. In contrast, Quill exercises involve code creation, in particular (1) writing code in general, (2) writing code that adheres to specific artifact construction algorithms (e.g., a row-major algorithm for creating pixel art), and (3) code optimization (e.g., creating an artifact using the fewest number of function calls). Lynx focuses on artifact sequences. Specifically, how artifact relates to or can be constructed from previous artifacts in the sequence. Lynx seeks to help students answer the question: "How could I create a Bricklayer artifact a_{n+1} given a function that creates artifact a_n?". Mystique focuses on developing an understanding of symmetry. More specifically, horizontal and vertical reflection symmetry as well as 2-fold and 4-fold rotational symmetry. Oftentimes the understanding of positional calculations (e.g., the endpoints of a line) as well as more specific code-based activities like parameter passing (e.g., (0,max) and (max,0)) and the number of function calls needed can be more clearly understood though an awareness of symmetry.

The Grid serves as electronic graph paper with some enhanced features that allows students to sketch out and thereby develop a clearer understanding of salient aspects of an artifact before writing a Bricklayer program that creates the artifact. And finally, BLite is a block-based programming environment in which a subset of Bricklayer programs can be created in an environment relatively free from the complexity and dangers resulting from syntactic errors.

The BLite programming environment employs a continuous execution model and provides extensive visual and verbal feedback regarding how to create well formed programs. BLite provides a gentle transition to text-based Bricklayer programming.

2.2 ...Infinite Ceiling

From a technical standpoint, Bricklayer programs are SML programs that make use of the Bricklayer graphical library. Since SML is a general-purpose programming language, the Bricklayer library can be (and has been) used to create a wide variety of artifacts including (but not limited to): (1) pixel art, (2) geometric patterns, (3) buildings and other forms of architecture, (4) 2D and 3D space-filling curves, (5) minimal surfaces like the Moebius strip, and (5) a variety of fractals including Wolfram's *nested patterns*, sponges (e.g., the Menger sponge), and Julia sets (e.g., the Mandelbrot set). The general-purpose computing capabilities of SML also admit the possibility of creating artifacts within polar coordinate systems such as Mystic Roses and Cardioids.

In addition, Bricklayer's cell-based framework is well suited for creating cellular-automata models and simulations allowing for the exploration of various phenomena such as percolation, heat diffusion and the spread of infectious diseases. Bricklayer can also be used to study Lindenmayer Systems (L-systems) as well as elementary cellular automata.

3 Bricklayer Highlights

To date, Bricklayer curriculums have been used in eastern Nebraska in over 100 schools across the K-16 spectrum. Thus far, engagement with elementary and middle schools has primarily involved gifted students.

At the elementary level, Bricklayer curriculums have been used to teach coding as well as math [11]. These Bricklayer curriculums have been shown to increase spatial abilities in students [9].

At the middle school level Bricklayer inquiry-based learning activities have been developed to teach various aspects of geometry [13].

At the high school level, preliminary findings indicate that Bricklayer has appeal to broad audiences [20]. High school level engagements include a year-long Bricklayer course offered under the title of Math Analysis. This course culminated with a VR experience (using the HTC Vive) where student teams created Bricklayer artifacts that were then placed in a Minecraft world. The high school course was offered for 2 consecutive years and was discontinued after the instructor was recruited by the tech industry. It should be noted that such recruiting represents a major concern to hiring and retaining teachers having appropriate technical backgrounds necessary for the educational needs of the future.

Beyond K-12, two courses are being offered at the University of Nebraska-Omaha (UNO). The first course is titled *Introduction to Mathematical and Computational Thinking* and can be used to satisfy the GenEd math requirement (e.g., an alternative to College Algebra) for non-STEM majors [23]. Currently, four sections, each section having a cap of 40 students, is being offered every semester at UNO. Omaha's Metro Community College also offers this same course which is recognized (from the perspective of transfer credit) by UNO as an acceptable substitute for the UNO course. The second course is titled *Introduction to Computational Science* and can be counted towards the fulfillment of the GenEd science requirement. Currently, one section, with a cap of 60 students, is offered annually at UNO with an anticipated significant increase in demand. Plans are underway at the Metro Community College to offer this course as well.

4 Related Work

Prodigy is a commercially available online multiplayer web-based game that is comprehensively integrated with math learning standards for grades 1-8. The math concepts encountered during gameplay are broadly determined by the indicated grade level of the player, and the specifics of problem selection (e.g., problem difficulty) is determined by an adaptive algorithm, whose goal is to keep students in their zone of proximal development [17].

Prodigy gameplay revolves around the completion of quests. In order to complete a quest, a player must battle their way through pokemon-esque monsters. A player can get monsters to join their side and be their pets. These pets can be used to fight other monsters. The outcome of battles are determined by math problems, which have a read aloud feature as well as associated embedded educational videos. If a player gets the given math problem right, their attack is successful; otherwise the attack does nothing. Prodigy also provides comprehensive feedback in the form of a teacher/parent dashboard. Feedback includes: (1) mastery level assessments, (2) individual concept status, and (3) overall grade-level completion status (i.e., curriculum progress).

Beast Academy is an educational ecosystem, created by the Art of Problem Solving, focusing on math for ages 8-13. Originally, educational content was offline in comic-book form. However, in 2018 Beast Academy went online. The homepage of the online ecosystem shows a small 2D mountain town comprised of four buildings: a centrally located Class building, a Theater, a Library, and a Lab. The Class building contains lessons, reading material and exer-

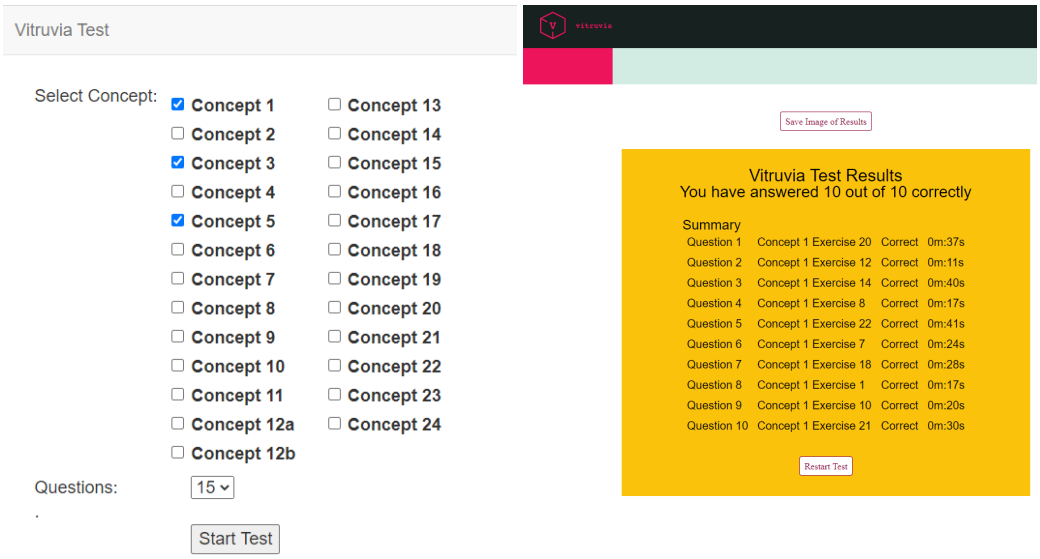


Figure 1: The Vitruia test generator: on the left generating a test, and on the right, displaying test results.

cises. The Theater contains instructional videos. The Library contains Beast Academy comic books, and the Lab contains interactive puzzles. Beast Academy is an excellent example of an engaging and comprehensive embedding of educational content into a tech environment.

Originally, **Bootstrap** [12] was a programming language, environment and curriculum designed primarily for middle and high school students. Programs were written in a subset of Racket, a functional programming language, and focused on the creation of video games involving four principle abstractions: (1) a player, (2) a target, (3) a danger, and (4) a background. Bootstrap coding is designed to achieve very precise learning objectives, namely, to improve abilities in solving algebra word problems of the kind found in standardized algebra tests given to middle school students in the US. The Bootstrap philosophy is based on the premise that in order for transfer of learning to occur between domains (e.g., coding and algebra) study must be carefully guided. Towards this end, students are taught how to use a process called the *(Bootstrap) Design Recipe* [8] to design and develop the functions that underly their video games. This approach has strong ties to key concepts defined in algebra learning standards.

Bootstrap has evolved in recent years to a larger educational ecosystem consisting of four modules: an Algebra module, a Reactive module, a Data Science module, and a Physics module. Computer science plays a foundational role in all modules. The original Bootstrap is now the Bootstrap:Algebra module. The Bootstrap:Reactive module explores more sophisticated elements of programming, thereby enabling extensions of the Bootstrap:Algebra game. The Bootstrap:Data Science module focuses on computational techniques and statistical methods used to analyze large datasets. The Bootstrap:Physics module focuses on building computational models of the physical world.

Alice[4] is a visual programming language and environment supporting object-based and event driven programming. Though visual in nature, Alice 3 programs can be converted into Java [3]. Alice programs produce interactive 3D computer animations whose goal is to tell a story. This storytelling objective distinguishes Alice[1] from traditional environments whose objectives center around the creation of a game or puzzle. Storytelling is appealing to a broad audience, in particular to females. Due to its general appeal, a strength of storytelling is that is it highly motivational. The hope being that this will result in learners spending significant time on task thereby improving learning outcomes. In general, people have considerable experience with and exposure to storytelling. This knowledge level of the learner facilitates the introduction of storyboarding as a design technique. Thus, Alice positively impacts learner motivation and its instructional design is able to leverage a higher knowledge level than might otherwise be expected.

CodeSpells [6][7] is an extensible educational platform designed to teach Java programming in the context of video game play (a JavaScript/Blockly ver-

[1] Recently, scope of Alice has been expanded to support the creation of simple games.

sion is available on Steam). CodeSpells targets learning spaces outside of the traditional classroom. The authors believe that it is in such non-institutionalized settings that lifelong passion for coding is most likely to develop.

In contrast to educational systems like Bootstrap[12], which focuses on *building* video games, the focus of CodeSpells is on *playing* a video game. More specifically, CodeSpells is a non-competitive role playing game in which the student is a wizard in a 3D world populated by gnome-like creatures. The game provides 7 primary spells which include levitation of objects, fire, and flying. Programming consists of modifying and creating spells, which are written in Java. Game play is largely unstructured, though quests can be undertaken to earn badges. The first level of the game introduces students to Java basics like method parameters, conditional statements, and iteration.

5 Advancements in Bricklayer's Periphery

This section gives an overview of five web apps: Vitruvia, Mystique, Quill, Lynx, and the Grid. These web apps are interactive and have been designed (redesigned in the case of the Vitruvia web app) to be mobile friendly. As discussed in Section 2.1.1, these apps provide scaffolding for the acquisition of skills beneficial to Bricklayer programming. In the subsections that follow, overviews are provided for each of these apps.

5.1 Vitruvia

Vitruvia is a web app that focuses on developing an understanding of Bricklayer library functions as well as SML programming constructs. A detailed discussion of Vitruvia can be found in [22]. Automatic test generation is a new capability in which tests can be created consisting of exercises randomly selected from a given set of concepts. The image on the left in Figure 1 shows the creation of a 15 question test consisting of exercises randomly selected from Concepts 1, 3, and 5. The image on the right shows the results of a test consisting of 10 questions selected from Concept 1. It should be noted that Mystique, Quill, and Lynx all have test generators similar to Vitruvia.

5.2 Mystique

Mystique is an interactive web app whose purpose is to develop a student's understanding of various forms of symmetry. Its creation was inspired from an interaction with a student who was working on a Bricklayer program to produce an artifact having various forms of symmetry. Rather that leveraging the properties of symmetry within the code, the student was recalculating coordinates for each symmetric element in the artifact (e.g., calculating the midpoint of the bottom of a square, then calculating midpoint of the left side of a square, and so on).

In Mystique, exercise sets exist for (1) horizontal and vertical reflection symmetry, (2) 2-fold, and 4-fold rotational symmetry, and (3) a special category of problem called a *completion*. Reflection and rotation exercise sets have two levels of difficulty. Difficulty level I involves artifacts occupying a 3×3 grid, and difficulty level II involves artifacts occupying a 5×5 grid.

The general form of a reflection/rotation exercise is as follows. An exercise is presented in a 2 column format with an image of an artifact displayed on the left and an empty interactable grid displayed on the right. The objective of an exercise is to create (via mouse clicks) a flipped/rotated version of the displayed artifact in the grid. Cell-based algorithms for reflecting or rotating artifacts can be employed when solving such problems. However, in order to encourage the development and exercise of a student's spatial abilities, artifacts have been selected to facilitate flipping or rotating the entire artifact in the "minds eye". The exercise shown on the left in Figure 2 is an example of artifact designed to encourage such manipulation. The intuition here is that the letter-h is a familiar shape and can therefore easily be flipped in its entirety. Other such shapes include arrows, the letter-s, diagonal lines and so forth.

The last and most difficult problem type in Mystique is called a *completion*. The exercise shown on the right in Figure 2 is an example of completion problem. In a completion problem, a student is given a *partial view* of a *completed artifact* belonging to one of the following symmetry groups: (1) horizontal reflection symmetry, (2) vertical reflection symmetry, (3) 2-fold rotational symmetry, (4) 2-fold and 4-fold rotational symmetry, (3) horizontal, vertical, and 2-fold symmetry, and (4) horizontal, vertical, 2-fold, and 4-fold symmetry. The objective of a completion exercise is to create in the grid (via mouse clicks) the *smallest* artifact (measured in terms of occupied cell count) having the desired symmetries. Sufficient information is provided in the partial artifact view to enable the completion of the artifact (i.e., solutions are unique).

5.3 Quill

Quill is an interactive web app designed to provide a stepping stone between Vitruvia and Bricklayer-lite. The key difference between Vitruvia and Quill is that while Vitruvia focuses on reading and executing programs, Quill focuses on writing/creating programs.

Quill exercises are presented in a 2 column format with an image of an artifact displayed on the left and an empty interactable grid displayed on the right. The

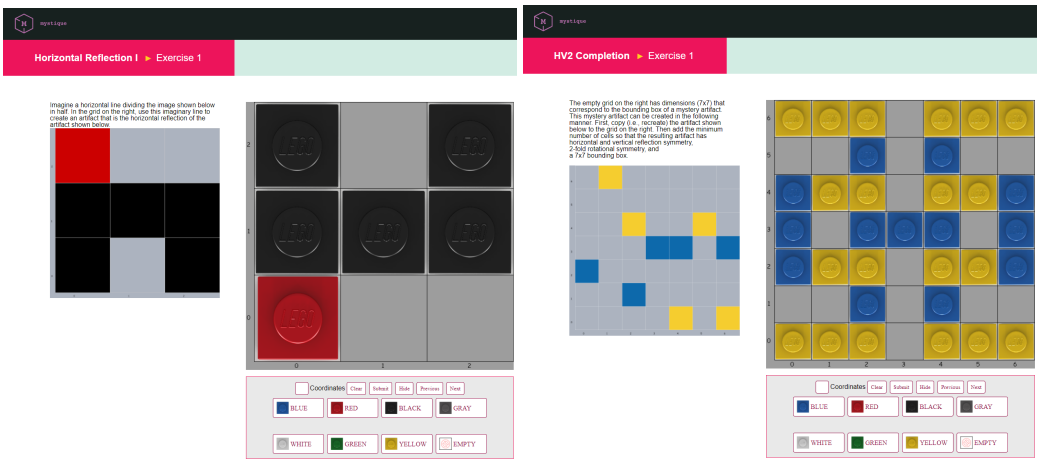


Figure 2: Mystique: On the left, flipping an artifact horizontally. On the right, solving a completion problem.

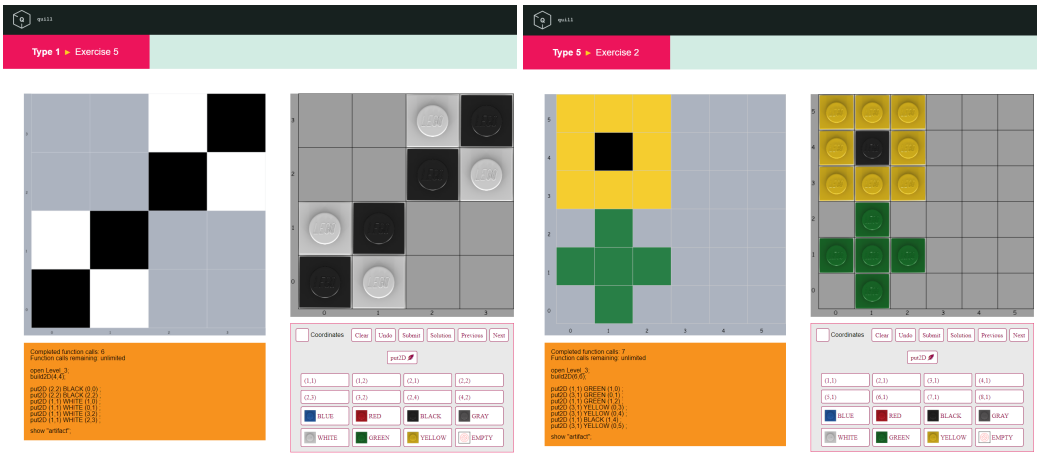


Figure 3: Quill: On the left, a Quill artifact creation using overwriting. On the right, enforcement of a row major construction algorithm.

objective of an exercise is to create (via mouse clicks) the displayed artifact in the grid. Quill exercises provide a set of brick shapes (e.g., 1×1, 2×3, and so on) and a set of brick colors (e.g., BLUE, YELLOW, and so on). A brick of a given size and color can be placed in the via the following mouse click sequence. First, click on the *put2D* button. Then click on the desired brick dimensions. Then click on the desired color. And finally, click on the cell in the grid where the lower left corner (the reference point) of the brick should be placed. In addition to creating the specified rectangle in the grid, a corresponding text-based put-function call will be created in the lower section of the right column. In this way, Bricklayer programs consisting of a linear sequence of put-function calls can be created.

Function Call Optimization. Bricklayer does not place restrictions on the positioning of bricks. For example, bricks can be placed so that they overlap (i.e., overwrite) one another or even go off the edge of the grid. Such capabilities, in particular overwriting, can be leveraged to optimize the number of put-function calls needed to create an artifact. Consider the creation of the artifact shown on the left in Figure 3. Without overwriting, 8 put-function calls (each placing a 1×1 brick) are needed to create the artifact. However, with overwriting the artifact can be created using using 6 put-function calls: 2 put-function calls placing 2 black (or white) 2×2 bricks followed by 4 put-function calls placing white (alternately black) unit bricks.

Quill tracks such optimization metrics and provides problem sets for which artifacts must be created using no more than a given number of put-function

calls. It should be noted that artifact creation involving such optimizations can require non-trivial analysis.

Pixel Art Creation Algorithms. Empirical evidence indicates that the creation of pixel art in Bricklayer is a very popular activity among students of all ages [24]. In a typical pixel art assignment, the student is given considerable freedom in the selection of the pixel art image they wish to create. A quick Google search of pixel art reveals that the range of complexity for pixel art images is extremely broad. In cases involving non-geometric art (e.g., portraits), the mathematically patternless nature of pixel placement requires a scribe-like approach to software construction that is heavily dependent upon reliable human-centered processes. For the creation of such pixel art in code, column/row major construction algorithms are very effective. For a detailed discussion of this topic see [25].

Quill provides problem sets that enforce the creation of artifacts through both column and row major construction algorithms. This assures that students have a correct understanding of these algorithms before attempting the construction of complex pixel art in code. In Figure 3, the image on the right shows the result of artifact creation using row major (left-to-right and bottom-to-top) construction algorithm.

5.4 Lynx

The website `http://www.visualpatterns.org/` provides a repository of artifact sequences. An entry in this repository consists of a sequence of three artifacts followed by one or more questions and answers about properties of the 43^{rd} artifact in the sequence. The most common question asks how many unit elements (e.g., 1×1 bricks) make up the 43^{rd} artifact in the sequence.

The Lynx web app was inspired, in part, by `http://www.visualpatterns.org/`. Artifact sequences provide a visual format for developing and understanding of incremental change and problem decomposition. For example, given and artifact sequence a_0, a_1, a_2 one can write a Bricklayer function that creates a_1. One can then analyze how a_2 might be constructed by *extending*, in some way, a single instance of a_1. In other words, what bricks need to be *added* to a_1 to obtain a_2? A corresponding mathematical analysis can then be done involving the size of the artifacts. This type of analysis underlies mathematical series (e.g., arithmetic series and geometric series).

Lynx has two types of exercises. The first type of Lynx exercise, shown on the left in Figure 4, consists of an image showing the first three artifacts of an artifact series where each artifact a_{i+1} in the series can be created by adding bricks to a_i. In other words, the size of the artifact a_{i+1} is increased through a visual union involving a_i. The student is then asked to create, in the empty grid on the right, the next artifact in the sequence. The second type of Lynx exercise, shown on the right in Figure 4, focuses on artifact series where the size of an artifact a_{i+1} is increased by expanding the size of an initial seed artifact a_0. For example, the size of a seed artifact a_0 can be increased by replacing each cell in a_0 by an $n \times n$ square. The resulting larger artifact will have similar shape to a_0.

5.5 The Grid

The *Grid*[19] is a Bricklayer web app that allows users to choose a color by clicking on a palette, then click on selected cells in the displayed grid in order to fill them with the chosen color. The Grid's standard capabilities include: (1) saving images of Grid artifacts, (2) saving and loading json files containing models of Grid artifacts, (3) undo and redo operations, and (4) repositioning an artifact within a Grid (e.g., moving an artifact to the left, right, up or down).

The original goals of the Grid were modest: it was designed to replace the graph paper and colored markers that students were using to create pixel art. The process of creating an image is iterative and often labor intensive. However, since then the feature set of the Grid has been extended as discussed in the sections that follow.

5.5.1 Exploring Static and Dynamic Elements of Pattern

Grid artifacts are static in nature in the same sense that paintings, drawn on canvas, are static in nature. In more technical terms, an image of a Grid artifact captures the state of the Grid (with respect to cell occupancy) at a given point in time. In such images, dynamic elements of the construction process (e.g., the order in which cells are occupied to create the artifact) are tacit as is the algorithm used to construct the artifact, which is left to inference.

To address this limitation, the Grid was extended to include an advanced set of features and interactions to (1) make explicit various dynamic elements underlying artifact creation, and (2) facilitate the exploration and examination of the exposed dynamic elements [21].

When exploring dynamic elements of artifact creation, three Grid features are of central importance: *Create Frame* (copy), *Frame Mode* (similar to paste), and *Show Graph*.

It should be noted that the Grid's frame metaphor is similar to, though different from the typical copy-paste metaphor used in text editors. The notion of frame draws on abstractions from film-making where the central focus is on animation. A frame is a finite function (i.e., an array) from integers (i.e., indexes) to

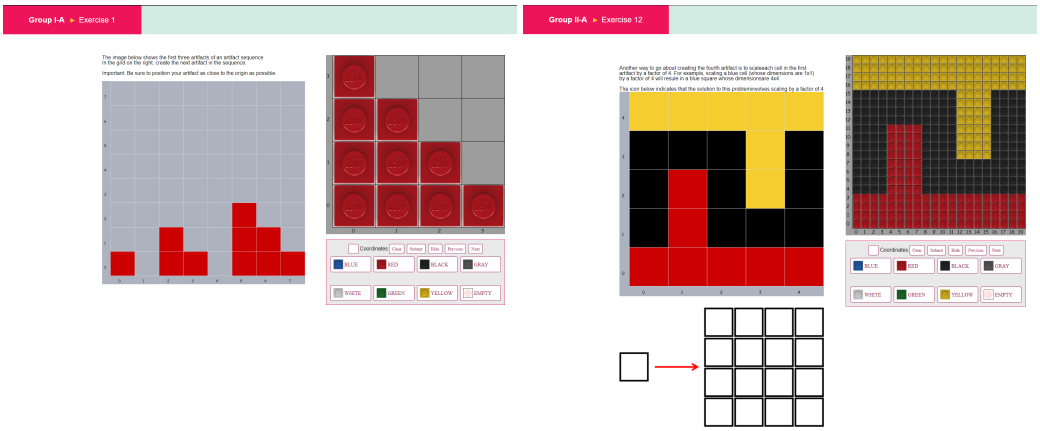


Figure 4: Lynx: On the left, an example of type 1 artifact sequences. On the right, an example of type 2 artifact sequences.

grid states. At present, the input domain of the frame function is $\{0, 1, \ldots, 5\}$. The Grid also supports rudimentary animation of artifact construction (which is not discussed in this article).

5.5.2 Frames

When selected, the *Create Frame* button will take a snapshot of (i.e., capture) the current state of the grid and associate this state with a particular index (i.e., input) of the *frame* function. The first snapshot will associate the current grid state with frame 0, the second snapshot will associate the current grid state with frame 1 and so on. We use the term *frame artifact* to denote a grid state that is stored in the frame function.

Complimenting the Create Frame button is a *Frame Mode* button which can be toggled on and off. When the Frame Mode is *off*, mouse clicks can be used to create an artifact using the standard "one mouse click for each cell to be colored" approach. However, when the Frame Mode is *on*, a mouse click will result in the placement of the entire contents of the most recently created frame artifact at the position selected by the mouse click. More specifically, the lower-left corner of the bounding box of the frame artifact will be positioned at the cell corresponding to the mouse click.

The frame function facilitates the systematic exploration and development of artifact construction techniques based on abstraction and inductive thinking. An example is shown in Fig 5. Beginning with an empty grid, one can use standard construction techniques to create an initial artifact. One can then use the *Create Frame* feature to take a snapshot of the artifact. Taking this initial snapshot will have the effect of updating the frame function so that the input 0 is associated with the artifact (i.e., frame(0) = artifact). With the Frame Mode on, one can then place copies of the captured artifact using single mouse clicks. Thus, the creation of the captured artifact has been abstracted to a single mouse click. When a desired grid state has been reached (e.g., the next artifact in a sequence of imagined artifacts), one can again capture the (entire) artifact using the Create Frame feature. In this way, abstract thinking is exercised through the association of single mouse clicks with artifacts of increasing size and complexity.

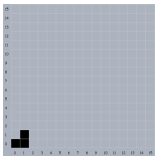
5.5.3 Artifact Graphs

The grid captures all mouse click sequences associated with the construction of an artifact. This includes mouse clicks associated with the creation of a frame artifact. Figure 6 shows the construction graphs resulting from three different lace creation algorithms. The graph on the upper right results from the frame-based inductive construction algorithm described in Figure 5. The graph on the lower left results from a top-down (recursive) construction algorithm based on the creation of a sequence of wire-frame triangles having smaller and smaller sizes. And the graph shown on the lower right results from a cellular automata construction algorithm based on rule 102.

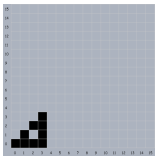
6 Advancements in Bricklayer's Core

The creation of 2D and 3D block-based artifacts through text-based SML programs making use of the Bricklayer graphics library represents the core of the Bricklayer ecosystem. The ecosystem's primary objective is to develop abilities suitable for artifact creation in this setting. As a result, all tool and educational content design and development efforts are oriented towards this goal.

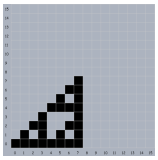
This section describes some key advancements to Bricklayer's coding infrastructure.



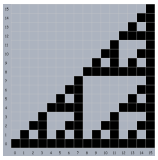
Step 1**:** Using the standard approach, construct the artifact shown using mouse clicks on cells $(0, 0)$, $(1, 0)$, and $(1, 1)$. Use the Create Frame button to capture the artifact.



Step 2**:** With *Frame Mode* set to on, the artifact shown here can be constructed by extending the artifact created in the previous step through two mouse clicks, one mouse click on cell $(2, 0)$ and the other on cell $(2, 2)$. After the construction is complete, use the *Create Frame* button to capture the artifact.



Step 3**:** Use an approach similar to Step 2 to create the artifact shown here using through two mouse clicks, one mouse click on cell $(4, 0)$ and the other on cell $(4, 4)$.



Step 4**:** Use an approach similar to Step 2 to create the artifact shown here through two mouse clicks, one mouse click on cell $(8, 0)$ and the other on cell $(8, 8)$.

Figure 5: Frame-based artifact construction of the reverseL Lace.

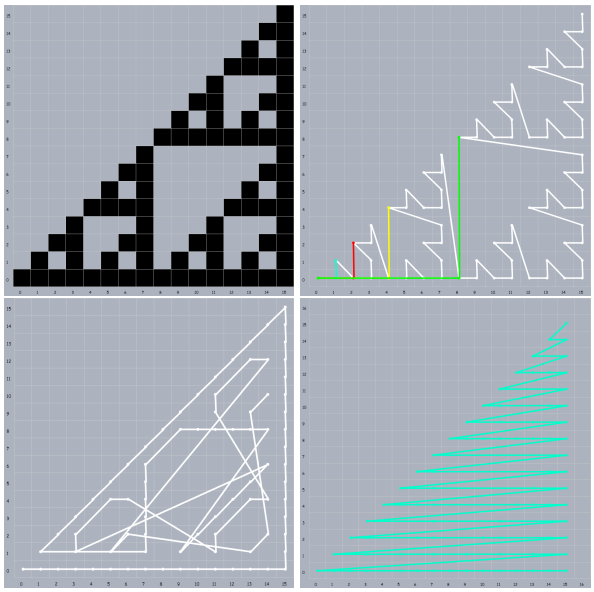


Figure 6: The construction graphs resulting from three different lace creation algorithms.

7 BLite

BLite is a Google blockly-based visual programming environment for creating Bricklayer artifacts. BLite, currently under development, is a significant improvement over its predecessor Bricklayer-lite. It provides a gentle transition from skills developed through engagement with Bricklayer webapps to the complexity of text-based programming. In Bricklayer-lite, which is not particularly mobile friendly, a run button must be clicked in order to see the results of evaluation. Error messages, which are only displayed after the run button has been clicked, are text-based, somewhat cryptic, and not prominently displayed (one would have to scroll down to see the error message). BLite, on the other hand, is mobile friendly, provides continuous evaluation and comprehensive visual feedback. Figure 7 gives a screenshot of an artifact produced by a well-formed BLite program as well as an example of an error message.

The benefit of continuous evaluation is that a clear cause-and-effect is established between function parameters (e.g., rectangle dimensions, color, and position) and their (visual) semantics. In BLite, an error message is displayed the moment a user places a block resulting in an ill-formed program. Furthermore, the error message displayed is contains both text based and visual information. Specifically, the visual information displays the error in terms of BLite program blocks. Error messages are sufficiently comprehensive that a person having no prior background can create a well-formed BLite program by simply fixing all program errors in the manner described by the error messages.

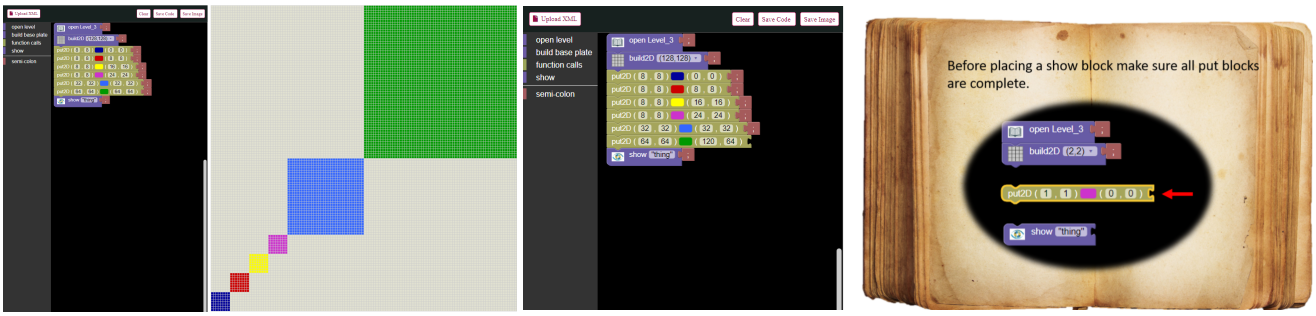


Figure 7: BLite: On the left, an artifact produced by a well-formed BLite program. On the right, an example of an error message.

7.1 The BricklayerIDE

Previously [18][22], text-based Bricklayer programming required a third party text editor. A Bricklayer install was required to create file associations and perform system level actions associated with program execution. The install for Windows machines was fairly standard, but the install for MAC OS machines was complex. Within this infrastructure, updates to the Bricklayer library required a complete reinstall. And finally, error messages as well as standard output was displayed in a system window (e.g., command prompt for windows).

When viewed as a whole, the infrastructure for executing Bricklayer programs, viewing error messages, and updating the Bricklayer library was cumbersome and unwieldy. These issues were the impetus behind the development of a Python-based Bricklayer development environment called the *BricklayerIDE*. The BricklayerIDE installs easily on both Windows and MAC machines without requiring administrator privileges. It embeds a SML-NJ compiler [2] and is packaged to be a self-contained installation with zero external installation requirements.

The *BricklayerIDE* provides a custom text editor (with features similar to notepad++) for creating Bricklayer programs. A Help tab provides links to online Bricklayer documentation for each Bricklayer programming level, and also allows the user to check for Bricklayer library updates. If a newer version of the Bricklayer library exists, the BricklayerIDE asks the user if they would like to install the update. If the user replies "yes" the Bricklayer library is automatically updated (i.e., no reinstall of the BricklayerIDE is required).

The BricklayerIDE also has a run button and a console window below the text editor in which error messages as well as standard output is displayed. To facilitate debugging, error messages (produced by SML) are scanned for line number information and the corresponding lines in the program text are flagged. This has proven to be extremely helpful for novice programmers.

7.2 Bricklayer's Color Palette

In the past, LEGO Digital Designer (LDD) and LDraw served as the primary and secondary tools for viewing Bricklayer artifacts. Using these tools, unit bricks were rendered as LEGO bricks. LDD, whose development has been discontinued by LEGO, is resource intensive. A number of classroom settings where Bricklayer was being taught had thin clients that were unable to support LDD and as a result the use of LDraw gained prominence. However, both these tools require their own software installs, are LEGO based, and were limited by the LEGO color palette. For these reasons, a custom browser-based tool, called WebViewer, was created to serve as the primary tool for viewing Bricklayer artifacts. The WebViewer is implemented in javascript using the three.js graphics library and is bundled with the BricklayerIDE install. It therefore requires no additional software installation. In the WebViewer, a unit brick is displayed as a cube (not a LEGO brick). Furthermore, the WebViewer supports the full RGB color palette. To leverage this capability, the Bricklayer library was extended to enable the contents of cells to hold RGB values in addition to LEGO bricks and Minecraft blocks.

RGB values make the expression of color gradients possible, which in turn allow for visually meaningful mappings between numbers and colors. For example, the RGB color cube shown in Figure 8 was created by mapping 3D coordinates to RGB values. Specifically, $(x, y, z) \rightarrow rgb(x, y, z)$ with an arithmetic operation (e.g., multiplication) thrown in to adjust for the dimensions of the cube. Julia sets represent a class of artifacts requiring mappings from real numbers to a color model such as RGB or HSV. The Julia set shown in Figure 8 was created with $c = -0.835 - 0.2321i$.

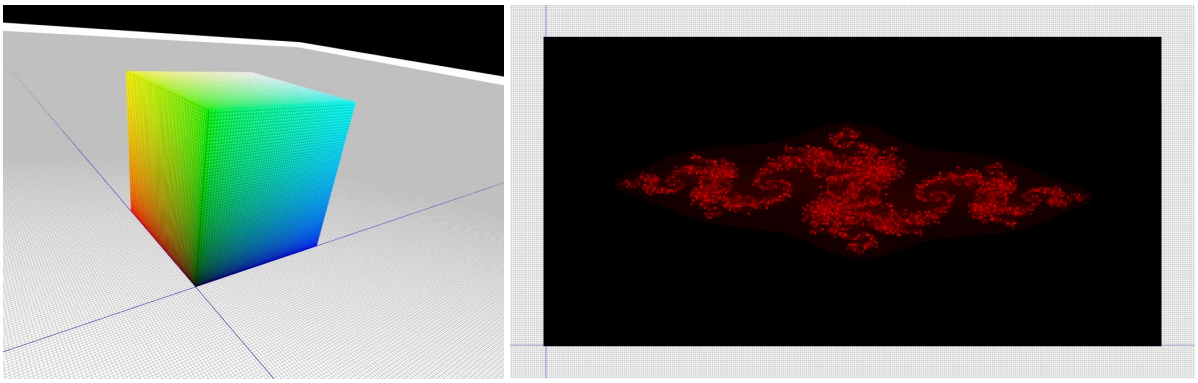


Figure 8: WebViewer: On the left, an RGB color cube, and on the right a Julia Set with $c = -0.835 - 0.2321i$.

7.3 Output Formats

One of the strengths of Bricklayer is that all artifacts are composed of unit cubes placed at discrete coordinates. The advantages of imposing such constraints on a design are numerous and widely recognized. In Minecraft, standard blocks and their placement conform to these constraints. In Fortnite, the placement of ramps, floors, and walls also conform to these constraints.

Unit cubes have relatively simple specifications. This significantly reduces the complexity associated with implementing a show function capable of outputting Bricklayer artifacts to a targeted file format (e.g., FBX). Table 1 lists the display capabilities of Bricklayer.

8 Digital Experiences and Gamification

Presently, significant development efforts are focused on the gamification of Bricklayer's learning objectives. These efforts revolve around the creation of third-person digital games/experiences developed in Unity. The *showUnity* display function and the Kessel Run are two Unity games that are in the early release stage.

8.1 The UnityViewer

The Bricklayer library has been extended with a *showUnity* display function that can be used launch a 3D digital experience via an application called the *UnityViewer*. The UnityViewer offers a third-person Unity digital experience (currently under development) in which the user enters a digital world where they can interact with educational content and summon and destroy their Bricklayer artifact. A short YouTube video of an early version of the UnityViewer can be viewed at `https://youtu.be/hRCj7g3gnZU`. Figure 9 shows the current version of the UnityViewer which contains (1) a variety of player spell casting emotes for player teleportation, as well as for summoning and destroying artifacts, (2) dance emotes with selectable music, and (3) and magic surfaces that when triggered appear underneath the players feet allowing them to walk into space.

The showUnity function requires that the UnityViewer has been previously installed (i.e., the UnityViewer lies outside of the BricklayerIDE). Under these conditions, an artifact that was created via the BricklayerIDE can be viewed using the *UnityViewer*.

8.2 The Kessel Run

The Kessel Run is a third-person Unity game framework in which Vitruvia, Mystique, Quill and Lynx exercises can be embedded. The Kessel Run incorporates versions of Bricklayer web apps that have been slightly modified so that information from the web app (e.g., whether an exercise has been solved correctly) is communicated from the web app back to the Unity game. Inspired by Han Solo's Kessel run, the game involves a trek across a sequence of platforms located in a galaxy far far away (see Figure 10). Teleportation from one platform to the next is enabled by correctly completing the platform's activity, which is contained in the embedded (and interactive) web app, and then walking through a portal which only appears after the exercise has been completed correctly. Completion of the final platform's exercise initiates an awards ceremony in which NPCs perform a dance (i.e., an emote). A narrated YouTube video showing game play can be accessed at `https://youtu.be/huJVMSs_QIA`.

Each platform in the Kessel Run is associated with a specific set of exercises. For example, the first platform is associated with the exercises from Vitruvia Concept 1, the second platform is associated with the exercises from Mystique horizontal reflection level 1, and so on. During gameplay, an exercise is randomly selected from the platform's exercise set and presented to the player.

show	The Bricklayer artifact will be displayed using a browser.
showLDD	If LEGO Digital Designer (LDD) is installed, the Bricklayer artifact will be displayed in LDD.
showLDR	If LDraw is installed, the Bricklayer artifact will be displayed in LDR.
showSTL	The Bricklayer artifact will be displayed using an available STL viewer (e.g., 3D Viewer or Print 3D in Windows)
showFBX	The Bricklayer artifact will be displayed using an available FBX viewer (e.g., 3D Viewer or Paint 3D in Windows)
showBinvox	While any viewer supporting the binvox file format can be used, this output format was specifically created for use with Brickr. To view with Brickr, first install Brickr (one time), create an "open with" file association (one time), and then manually load the desired binvox file from within Brickr (every time).
showMC	This requires that both a Minecraft client and a special Minecraft server are running on your machine. The Bricklayer artifact will then be created in the Minecraft world that is currently on the server.
showArduino	This show function is in an experimental stage and requires the installation of the Arduino IDE as well as specific Arduino libraries.
showUnity	This show function is not yet publicly available and requires the installation of the UnityViewer.

Table 1: Bricklayer's output functions.

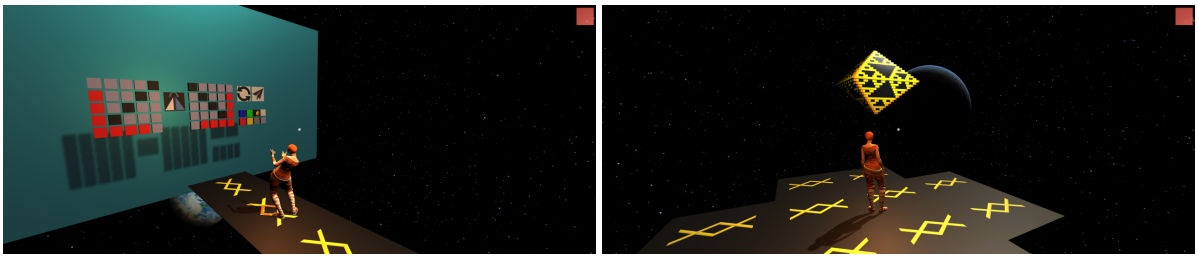


Figure 9: Scenes from the UnityViewer. On the top left, solving a reflection exercise and then doing an emote. On the top right, summoning a Bricklayer artifact.

The scoring function for the Kessel Run is currently being refined and its design draws upon considerable research that has been conducted regarding how to best measure performance [16]. Intuitively, performance (i.e., the scoring function) should, at a minimum, take into account mouse clicks and time. It should be noted that exercises (which are randomly selected) require the creation of artifacts of various sizes. Thus, the following properties should hold if a scoring function is to be fair. The scoring function should not penalize a player for (1) having to create an artifact that requires more mouse clicks than the artifact created by another player, and (2) requiring more time to create an artifact that is larger than the artifact created by another player. To ensure the first property, the *accuracy* (i.e., the number of mouse clicks for the optimal solution divided by the number of actual mouse clicks used to create the current solution) of a solution is measured. To ensure the second property, a *speed* function that measures the number of correct mouse clicks per second is used. And lastly, the value of *experience*, measured in how many times a player has completed the Kessel Run, is taken into account. The experience metric rewards *practice* which has been shown to positively affect stress relating to test performance [1].

9 Computational Science

A variety of scientific models and simulations are based on cellular automata [10]. Diffusion models and simulations can be used to study heat-diffusion, spreading of fire, ant behavior, and biofilms [14]. Cel-

Figure 10: Kessel Run: Player in front of a Vitruvia exercise (top). Player in front of a portal (lower left), and player at the awards ceremony at the end of the Kessel Run (lower right).

lular automata have also been used to model and study the formation of biological patterns (e.g., molds), Turing patterns, as well as adhesive interactions of cells [5]. Calculations associated with such models and simulations involve storing numerical values in cells and performing cell-based calculations using those values (e.g., computing the average value for a von Neumann or Moore neighborhood). To support such models and simulations, the Bricklayer library has been extended to enable the contents of cells to hold fixed-point values[2].

9.1 The Heat Diffusion Library

The Bricklayer library contains a Heat Diffusion library providing abstractions and functionality that support exploring and experimenting with heat diffusion on 2D surfaces.

A basic heat diffusion experiment begins with a Bricklayer model of a 2D surface (e.g., a table top or cutting board) and a heat source with a given temperature and a heat diffusion rate. Next, neighborhood and boundary models must be specified. A neighborhood model can be either a von Neumann model (4 neighbors) or a Moore model (8 neighbors). A boundary model can be either an absorbing model, a reflecting model, or a periodic model. A heat diffusion simulation can then be run for a given number of simulation steps. The result of a simulation is a cellular automaton whose cells contain fixed-point numerical values representing temperature. To visually display temperature in terms of color, the Heat Diffusion library provides users the ability to specify *color palettes*. In this context, a color palette is a list of $(temperature, color)$ pairs, where colors are expressed as RGB values. Given a color palette, linear interpolation is then used to map all temperature values in a specified range to corresponding RGB colors. The Heat Diffusion library also provides some predefined color palettes.

Figure 11 shows three stages of a heat diffusion experiment. In the first stage, a $1''$ heated brass cube

[2] A fixed-point representation is needed to permit equality comparisons on the contents of Bricklayer cells and a detailed discussion of this topic lies outside the scope of this article.

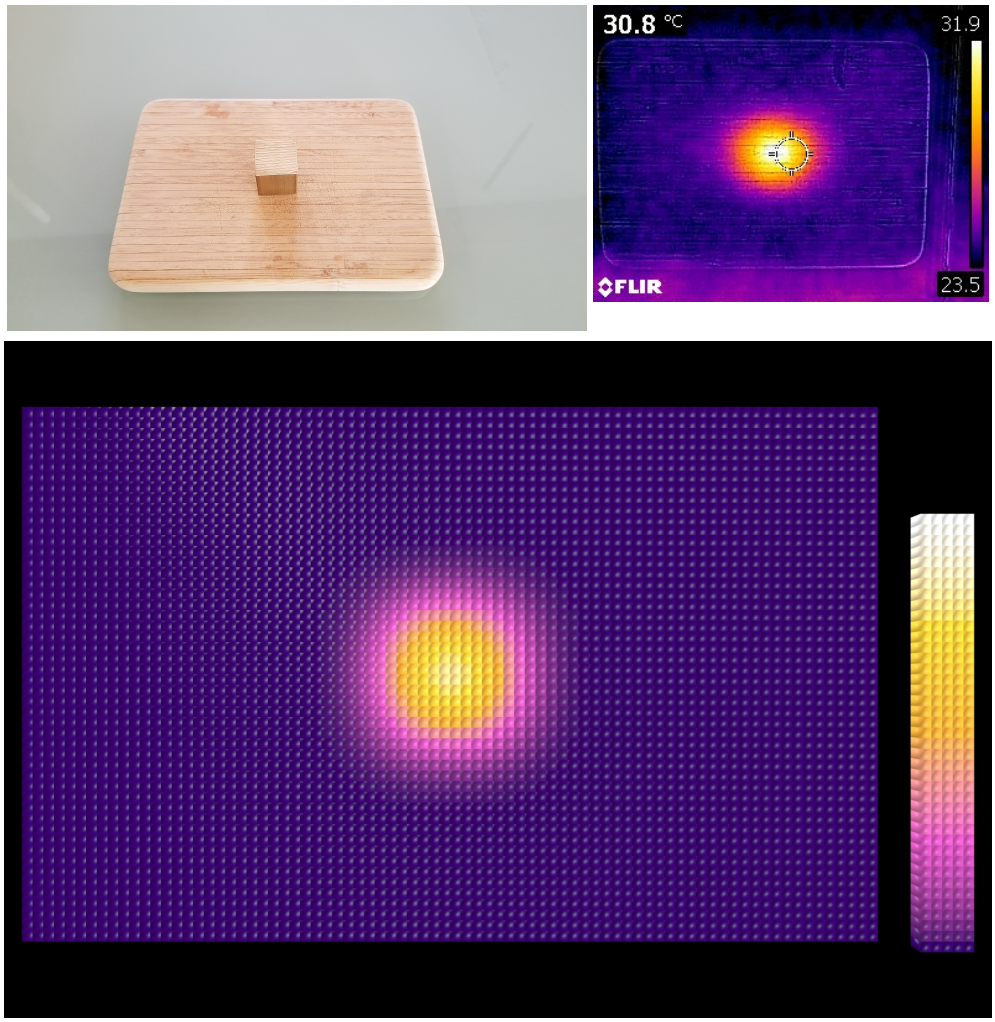


Figure 11: Heat diffusion: A bamboo cutting board on which a heated brass cube has been placed (upper left). A thermal image of the heat diffusion in the cutting board (upper right). A Bricklayer simulation of the heat diffusion for a model of the cutting board (bottom).

is placed on an $8'' \times 5.5''$ bamboo cutting board. After a period of time, the cube is removed and a thermal image is taken of the bamboo cutting board. Next, an RGB image color picker is used to reverse engineer the color mappings in the thermal image. The resulting information is used to create a color palette, which is then used to display the result of the heat diffusion simulation.

9.2 The Percolation Library

Percolation theory[15] provides an example of the importance of probabilistic cell-based computational models and simulations. A representative question in Percolation theory is as follows. Assume that a liquid is poured on top of some porous material. Will the liquid be able to make its way from hole to hole and reach the bottom? Percolation is an important scientific model because of its numerous applications to chemistry, biology, statistical physics, epidemiology, and materials science. For example, a composite system comprised of metallic (open) and insulating (blocked) materials is an electrical conductor if there is a metallic path from top to bottom. In Bricklayer, such materials can be modeled as rectangles (typically squares) in which some cells are occupied (i.e., blocked) and others are empty (i.e., open). Simulations involving such 2D models can be useful in studying the percolation phenomenon. Of particular interest is the application of Monte Carlo simulations to approximate *percolation thresholds*. The determination of such thresholds is an example of a calculation for which no mathematical solution has yet been derived. In practice, a standard approach to answering the classical threshold question involves a search for a probability p having the property that

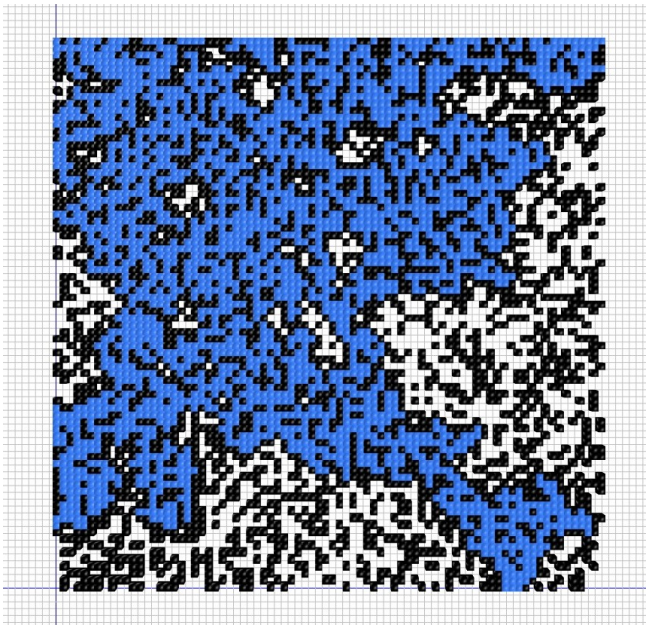


Figure 12: A randomly generated Bricklayer 2D cell structure that percolates.

when *p-open* $= p$ half of the randomly generated instances percolate. It should be noted that the graph of $f(\textit{p-open}) = \textit{percolation-probability}$ is a sigmoid function that represents a tipping function. What this means is that for large squares, small departures in p-open from the percolation threshold will result in instances that almost always/never percolate.

The Bricklayer library contains a Percolation library providing abstractions and functionality that support exploring and experimenting with percolation in the context of 2D rectangles. Figure 12 shows an artifact (displayed using the WebViewer) that percolates in which (1) black blocks represent closed cells, (2) empty cells represent open cells, and (3) blue blocks represent water (a flowing liquid). The percolation functions provided enable both the manual and automated search for percolation thresholds. A key abstraction here is an individual *percolation test* – a rectangle (typically a square) that is created in which cells are populated according to a given p-open probability and where water is then poured on the top row. Bricklayer provides functionality supporting the specification and analysis of individual tests. This includes metrics related to the number number of occupied cells, the number of open cells on the top/bottom rows, and whether the rectangle percolates (which can also be determined by visual inspection). Using just this functionality, a question that can be asked is "For a given test specification[3], how many tests must be run before a percolating rectangle is encountered?"

The Percolation library allows individual percolation tests to be grouped and their test results aggregated (e.g., for a given p-open probability a rectangle of a given size percolated x-percent of the time). Test groups enable a manual search for percolation thresholds and also demonstrate the power/value of simulation. And finally, a Monte Carlo search function is provided that can be used to automatically search for threshold values to within a desired tolerance.

10 Conclusion

In order to appropriately engage student populations of increasingly technological societies, technology should be intentionally and comprehensively integrated with educational objectives. The interactive and responsive potential of tech provides a rich environment for culturally relevant personalized learning that is well suited for mastery-based learning. Bricklayer is a growing educational ecosystem in pursuit of the previously stated objectives. On Bricklayer's periphery are a growing number of webapps with expanding capabilities especially designed to address knowledge gaps related to the Bricklayer coding. Bricklayer's core technology begins with BLite whose purpose is to provide a gentle transition to text-based programming. The Unity game engine provides a cross-platform technology supporting the creation of increasingly engaging games and digital experiences encompassing Bricklayer tech. And lastly, Bricklayer's block based visual domain is well suited for a broad range of educational objectives including: art, geometric patterns, fractals, math, computer science, and computational science.

References:

[1] S. L. Beilock, C. A. Kulp, L. E. Holt, and T. H. Carr. More on the Fragility of Performance: Choking Under Pressure in Mathematical Problem Solving. *Journal of Experimental Psychology: General*, 133(4):584–600, 2004.

[2] Standard ML of New Jersey. `https://www.smlnj.org/`, 2020.

[3] T. Daly and E. Wrigley. *Learning Java Through Alice 3*. CreateSpace Independent Publishing Platform, second edition, 2014.

[4] W. Dann, S. Cooper, and R. Pausch. *Learning to Program with ALICE*. Pearson Education, 501 Boylston Street, Suite 900, Boston Massachusetts 02116, third edition, 2012.

[3] The specification of a percolation test includes a seed value for the random number generator used by p-open.

[5] A. Deutsch and S. Dormann. *Cellular Automaton Modeling of Biological Pattern Formation*. Birkhauser, 2^{nd} edition, 2017.

[6] S. Esper, S. R. Foster, and W. G. Griswold. CodeSpells: Embodying the Metaphor of Wizardry for Programming. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 249–254, New York, NY, USA, 2013. ACM.

[7] S. Esper, S. R. Foster, and W. G. Griswold. On the nature of fires and how to spark them when you're not there. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 305–310, New York, NY, USA, 2013. ACM.

[8] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. *How to Design Programs: An Introduction to Computing and Programming*. MIT Press, Cambridge, MA, USA, 2001.

[9] M. Friend, M. Matthews, V., Winter, B. Love, D. Moisset, and I. Goodwin. Bricklayer: Elementary Students Learn Math Through Programming and Art. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 628–633, New York, NY, USA, 2018. ACM.

[10] A. Ilachinski. *Cellular Automata - A Discrete Universe*. World Scientific, 2001.

[11] P. Jakopovic, M. Friend, B. Love, and V. Winter. Changing the game: Teaching elementary mathematics through coding. In K. Graziano, editor, *Proceedings of Society for Information Technology & Teacher Education International Conference 2019*, pages 55–60, Las Vegas, NV, United States, March 2019. Association for the Advancement of Computing in Education (AACE).

[12] E. Schanzer, K. Fisler, S. Krishnamurthi, and M. Felleisen. Transferring Skills at Solving Word Problems from Computing to Algebra Through Bootstrap. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 616–621, New York, NY, USA, 2015. ACM.

[13] K. Sherwin, V. Winter, and B. Love. Inquiry-Based Learning Activities for Geometry. In *National Council of Teachers of Mathematics (NCTM)*, San Diego, California, 2019. (workshop).

[14] A. B. Shiflet and G. W. Shiflet. *Introduction to Computational Science - Modeling and Simulation for the Sciences*. Princeton University Press, 2^{nd} edition, 2014.

[15] D. Stauffer and A. Aharony. *Introduction to Percolation Theory*. CRC Press, 1994.

[16] A. Vandierendonck. A comparison of methods to combine speed and accuracy measures of performance: A rejoinder on the binning procedure. *Behavior Research Methods*, 49(2):653–673, 2017.

[17] L. S. Vygotsky. *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, 1978.

[18] V. Winter. Bricklayer: An Authentic Introduction to the Functional Programming Language SML. *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, 2014.

[19] V. Winter. The Grid, 2015.

[20] V. Winter and J. M. Diaz-Kelsey. In Pursuit of CS-based Educational Content Suitable for Broader Audiences. In *2020 ACM Special Interest Group on Information Technology Education (SIGITE)*, October 2020.

[21] V. Winter, M. Friend, M. Matthews, B. Love, and S. Vasireddy. Using Visualization to Reduce the Cognitive Load of Threshold Concepts in Computer Programming. In *2019 IEEE Frontiers in Education Conference (FIE)*, Oct 2019.

[22] V. Winter, B. Love, and C. Corritore. The bricklayer ecosystem - art, math, and code. *Electronic Proceedings in Theoretical Computer Science*, 230:47–61, Nov 2016.

[23] V. Winter, B. Love, M. Friend, and M. Matthews. A computer scientist teaches gen ed math. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI), Symposium on Education (CSCI-ISED)*, Dec 2019.

[24] V. Winter, B. Love, M. Friend, and M. Matthews. A computer scientist teaches gen ed math. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 793–799, 2019.

[25] V. Winter, B. Love, and C. Harris. Delphi: A source-code analysis and manipulation system for bricklayer. In *Proceedings - SEKE 2017*, Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, pages 456–461. Knowledge Systems Institute Graduate School, 2017. 29th International Conference on Software Engineering and Knowledge Engineering, SEKE 2017 ; Conference date: 05-07-2017 Through 07-07-2017.