Interactive programming paradigm for real-time experimentation with remote living matter

Peter Washington[a], Karina G. Samuel-Gama[a], Shirish Goyal[a], Ashwin Ramaswami[a], and Ingmar H. Riedel-Kruse[a,1]

[a]Department of Bioengineering, Stanford University, Stanford, CA 94305

Edited by David A. Weitz, Harvard University, Cambridge, MA, and approved January 15, 2019 (received for review September 6, 2018)

Recent advancements in life-science instrumentation and automation enable entirely new modes of human interaction with microbiological processes and corresponding applications for science and education through biology cloud laboratories. A critical barrier for remote and on-site life-science experimentation (for both experts and nonexperts alike) is the absence of suitable abstractions and interfaces for programming living matter. To this end we conceptualize a programming paradigm that provides stimulus and sensor control functions for real-time manipulation of physical biological matter. Additionally, a simulation mode facilitates higher user throughput, program debugging, and biophysical modeling. To evaluate this paradigm, we implemented a JavaScript-based web toolkit, "Bioty," that supports real-time interaction with swarms of phototactic *Euglena* cells hosted on a cloud laboratory. Studies with remote and on-site users demonstrate that individuals with little to no biology knowledge and intermediate programming knowledge were able to successfully create and use scientific applications and games. This work informs the design of programming environments for controlling living matter in general, for living material microfabrication and swarm robotics applications, and for lowering the access barriers to the life sciences for professional and citizen scientists, learners, and the lay public.

human–computer interaction | cloud laboratory | augmented reality | swarm programming | interactive biotechnology

Life-science research is increasingly accelerated through the advancement of automated, programmable instruments (1). Nevertheless, many usage barriers to such instruments exist, primarily due to physical access restrictions, advanced training needs, and limitations in programmability. Equivalent barriers for computing (2–4) have been solved through application programming interfaces (APIs) (5), domain-specific applications, and cloud computing (6–9). Consequently, cloud laboratories to remotely experiment with biological specimens have been developed and deployed for academia and industry (10), with applications including citizen science games (11, 12) and online education (13–16). Different approaches have been taken to make automated wet laboratory instruments programmable: Roboliq (17) uses artificial intelligence (AI) to ease the development of complex protocols to instruct liquid-handling robots; BioBlocks (18) and Wet Lab Accelerator (10) are web-based visual programming environments for specifying instrument protocols on cloud laboratories like Transcriptic (10).

Beyond programming automated laboratory experiments, we propose that there is an emerging need for a more general programming paradigm that allows users to develop applications that enable real-time interaction with the living matter itself. In analogy to conventional computers, this can be seen as the difference between numerical calculations by mathematicians vs. truly interactive applications like word processing (19), interactive graphical programs (20), and computer games (21) used by all strata of society. In other words, first-hand interactive experience with microbiology should become accessible for everyone. Such concepts of "human–biology interaction" (HBI) have been explored previously through interactive museum installations (22) and educational games (23), but both the software and the hardware always had to be developed from the ground up. Swarm programming abstractions for easier development of interactive applications have also been proposed (24). Other potential future applications include living material microfabrication through light stimulation, self-assembly, and swarm robotics, e.g., with engineered bacteria or molecular motors (25–31).

Specifically, we conceptualize and implement an integrated development environment (IDE) (32) and API for the creation of both interactive and automated applications with living matter hosted on a cloud laboratory (Fig. 1) (14). This paradigm enables real-time interactive applications and spatial separation of life-science instruments, programmers, and end users. As a specific implementation, we develop the JavaScript-based web toolkit Bioty that uses the phototactic behavior of *Euglena* cells (14). We conduct on-site and remote user studies with domain experts and novices to test the usability of the system and of applications developed.

Results: System

Overview. We determined through iterative design, development, and user testing that a system for remotely programming living matter should ideally have the following minimal set of components (Fig. 2): (*i*) end user and programming environments supporting online and event-driven application conventions (Fig. 2 *A*, *B*, and *H*); (*ii*) a set of programming functions for manipulating and sensing biological matter and integrating with standard programming logic (Fig. 2*C*); (*iii*) biotic processing

Significance

Biology cloud laboratories are an emerging approach to lowering access barriers for life-science experimentation. However, suitable programming approaches and interfaces are lacking for both domain experts and lay users, especially ones that enable interaction with the living matter itself and not just the control of equipment. Here we present a programming paradigm for real-time interactive applications with remotely housed biological systems which is accessible and useful for scientists, programmers, and lay people. Our user studies show that scientists and nonscientists are able to rapidly develop a variety of applications, such as interactive biophysics experiments and games. This paradigm has the potential to make first-hand experiences with biology accessible to all of society and to accelerate the rate of scientific discovery.

Author contributions: P.W. and I.H.R.-K. designed research; P.W., K.G.S.-G., S.G., A.R., and I.H.R.-K. performed research; P.W., K.G.S.-G., S.G., and A.R. contributed new reagents/analytic tools; P.W. analyzed data; and P.W. and I.H.R.-K. wrote the paper.

The authors declare no conflict of interest.

This article is a PNAS Direct Submission.

Data deposition: The link to all code used to implement the cloud laboratory is publicly available on GitHub (https://github.com/hirklab/euglenalab).

[1]To whom correspondence should be addressed. Email: ingmar@stanford.edu.

This article contains supporting information online at www.pnas.org/lookup/suppl/doi:10.1073/pnas.1815367116/-/DCSupplemental.

Published online March 1, 2019.

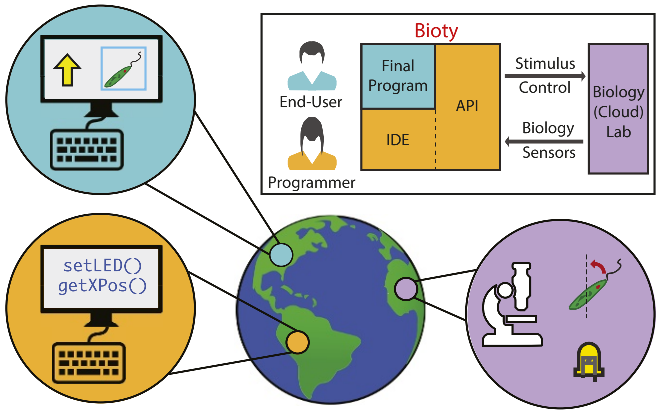


Fig. 1. We conceptualize a web programming paradigm that enables the creation of automated, real-time interactive applications with living matter for expert and nonexpert programmers and end users. An integrated development environment (IDE) enables programmers to rapidly develop versatile applications. The resulting applications can then be run by end users (e.g., experimenters). The underlying application programming interface (API) includes stimulus commands that are sent to a biology cloud laboratory affecting the living matter, e.g., shining light on phototactic *Euglena gracilis* cells. The API also contains biology sensor commands to detect properties of the living matter, e.g., tracking cellular movements. We termed the specific implementation of this paradigm "Bioty." The programmer, end user, and cloud laboratory can be spatially separated across the globe (depicted positions are of illustrative nature).

units (BPUs) to digitally interface with the biological specimen and where a cluster of such BPUs is hosted on the cloud (Fig. 2*E*); (*iv*) virtual BPUs that simulate all real BPU functionalities, allowing the users to switch between virtual and real BPUs (Fig. 2 *D* and *F*); and (*v*) real-time conversion of BPU raw output into high-level accessible data structures (Fig. 2*G*). We developed Bioty as a specific implementation that integrates all of these components. Humans working with this system fall into the two categories of end users and programmers, i.e., the latter developing applications in Bioty that the former then use.

BPUs. In analogy to electronic microprocessors like GPUs (33), BPUs (14, 15, 34) are devices that house, actuate, and measure microbiological systems. Computing is defined by the Association of Computing Machinery (ACM) as a series of "processes that describe and transform information" (35). A BPU performs biological computation by transforming a digital input into a physical stimulus affecting analog biological behavior, which is then converted back into digital output. A BPU can be programmed like a conventional microprocessor, with a domain-specific instruction set where the "computational algorithms" are realized through the nondeterministic biological behavior and responses of living matter (24).

Here, we use a previously described BPU architecture (14) (Fig. 2*E*): Photophobic *E. gracilis* cells (36, 37) are housed in a quasi-2D microfluidic chip, and the modifiable light intensity of four LEDs placed in each cardinal direction can stimulate cells to swim away from light, which is recorded by a microscope camera (Fig. 2*E*). More complex responses are also possible (24).

Biology Cloud Laboratory. A cloud laboratory has the advantage of making biology experiments accessible from anywhere. However, the presented programming paradigm is equally suitable for a local implementation.

We developed Bioty over the existing cloud laboratory architecture, described previously in ref. 14, which provides real-time interactive access to a cluster of BPUs (Fig. 2*E*), e.g., through a virtual joystick.

Biological Data Structures. The raw data stream from a BPU should be preprocessed in real time into higher-level data types that enable direct access to state variables about the biological material. Such abstractions allow programmers to treat biological objects (e.g., cells) like sprites (38) or objects in a database, whose state (e.g., position or gene expression level) can be queried and manipulated in real time (24).

In Bioty, we implemented a continuous image-processing layer where cells are continuously tracked (Fig. 2*G*). Information about individual cells, such as position and orientation, is extracted and associated with a cell index. The resulting data structures can be queried directly.

Programming Abstractions. There are three fundamental categories of functions for programming living matter: stimulus control (actuation), organism sensing, and application creation

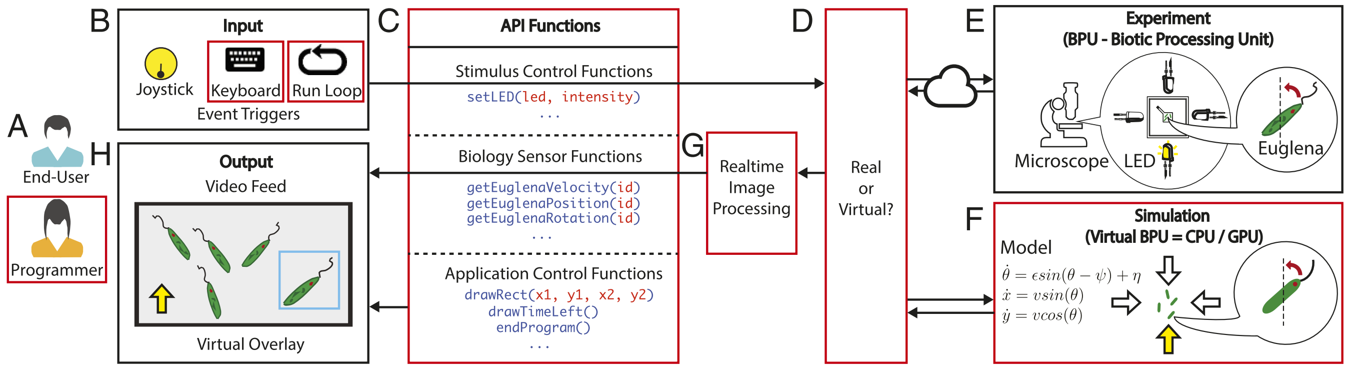
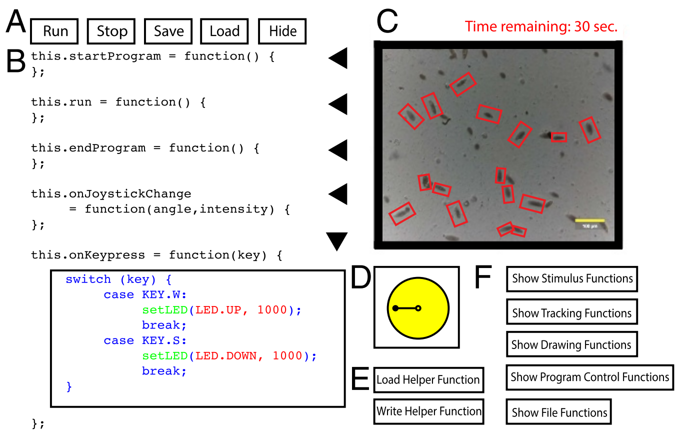


Fig. 2. System architecture of Bioty, which transforms a biology cloud laboratory (14) into a platform for programming living matter. (Bioty-specific features that have been implemented here compared to ref. 14 are highlighted with a red box.) (*A*) The users of Bioty fall into two categories, end users and programmers, where the latter develop applications for the former. (*B*) End users have a multitude of input modalities for interacting with the remote interactive microscopes, including a virtual joystick and the user's keyboard. The run loop allows timed events to be programmed as well. (*C*) The programmers can develop applications using Bioty's API functions for stimulus control, biology sensing, and application control. (*D*) The developed programs can be run either on the live experimentation platform (BPU) or on the simulation thereof (virtual BPU, i.e., CPU/GPU). (*E*) When running on the experimentation platform, the users can control biological stimuli in real time, i.e., using light emitted from an LED, which affects the swimming direction and other behaviors of living *Euglena* cells inside the BPU. (*F*) Alternatively, the users can execute their programs on a simulation that models *Euglena* behavior. (*G*) The real-time image processing system can operate on either the live video stream from the real BPU or the image stream from the simulation (i.e., virtual BPU). (*H*) The final output consisting of a processed live video stream and overlaid virtual objects is displayed to the users.



```
this.startProgram = function() {
};

this.run = function() {
};

this.endProgram = function() {
};

this.onJoystickChange
    = function(angle,intensity) {
};

this.onKeypress = function(key) {
    switch (key) {
        case KEY.W:
            setLED(LED.UP, 1000);
            break;
        case KEY.S:
            setLED(LED.DOWN, 1000);
            break;
    }
};
```

Fig. 3. The Bioty user interface enables users to program applications and to observe the program's effect on living cells. (*A*) The tool bar at the top allows users to start/stop the program and save/load their code. Users can also hide/show their code to preview the final prototype without seeing the underlying code. (*B*) User programming area. Each text box corresponds to a particular event: the program starting, the program ending, a millisecond passing, a user keypress, or the movement of the joystick control by the user. Code boxes can be expanded and collapsed. (*C*) Live microscope video feed, with virtual objects overlaid on the frames. This is the primary end-user program created by the user. (*D*) The joystick provides another method of user input beyond keypresses, for example by mapping the joystick's angle to LED direction and the joystick's drag length to LED intensity. (*E*) Users can write helper functions which can be used across programming areas and user programs. (*F*) API calls are displayed on the interface. The functions are organized by type and can be expanded and collapsed. (This is a schematic of the actual user interface, placed here for legibility; *SI Appendix*, Fig. S3 shows a screenshot of the Bioty user interface.)

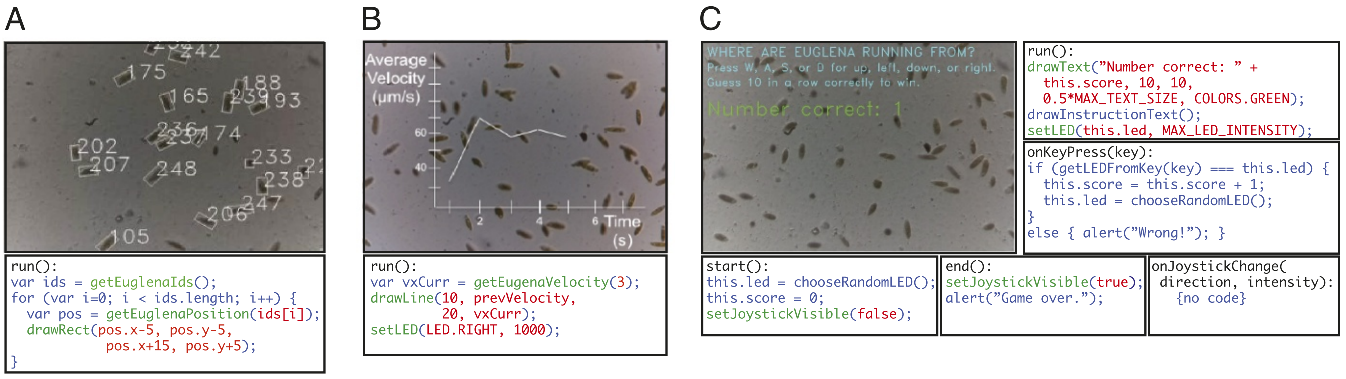
(Fig. 2*C*). The actuating ("writing") functions affect the state of biological matter via a physical stimulus inside the BPU. The sensor functions "read" the state of the biological matter. The application creation functions consist of all other standard programming functionalities.

We implemented Bioty in JavaScript with the needed functions in each category. Stimulus control functions like "setLED" allow manipulation of light intensity and evoke *Euglena* responses like negative phototaxis. Biology sensor functions like "getEuglenaPosition" provide information regarding the position of a tracked *Euglena* cell with a given ID; functions for velocity, orientation, and regional cell count assessment were also implemented. Application creation functions are Bioty specific, e.g., drawing virtual shapes on the live video feed. Functions can be combined into new, more advanced functions. The full set of Bioty functions is detailed in *SI Appendix*, Fig. S4. Users can also use the standard built-in JavaScript libraries.

Interface Design. An accessible user interface and programming environment is required to reflect the standards of online (39, 40) and event-driven (41, 42) development environments. Interactive applications can then be developed and executed by any programmer and end user.

The client side of Bioty (Figs. 2 *B* and *H* and 3 and Movie S1) has a programming interface (Fig. 3*B*) and program output that includes the live BPU camera feed with virtual overlays generated by the program (Fig. 3*C*). The programming interface has distinct areas (Fig. 3*B*) for five separate event-driven functions: (*i*) "startProgram" runs at the beginning of program execution, (*ii*) "endProgram" runs after program termination, (*iii*) "run" continuously operates during program execution at a rate of 1 kHz, (*iv*) "onKeypress" runs when the user presses a key, and (*v*) the "onJoystickChange" function runs when the user operates the virtual joystick (Fig. 3*D*). The joystick angle and magnitude map to LED intensity inside the BPU. Standard features supporting programmers and end users are in place (Fig. 3*A*): "Run" and "Stop" buttons trigger the "startProgram" and "endProgram" events, and "Save Code" and "Load Code" buttons enable file handling. A user can also run the same program on different BPUs.

Simulation Mode—Virtual BPU. A virtual BPU should be integrated that can be programmatically accessed equivalently to a real BPU, using the same programming commands. It should be simple for a user to switch between the real and simulated BPUs. Here, the actual "biological computation" will likely not be the same for the real and virtual BPU, as the fidelity of the underlying model is typically limited by incomplete knowledge of the biological system and by computational power. The virtual BPU is useful as it (*i*) allows fast and cheap testing and debugging of programs, (*ii*) enables application development even if there are more developers than available real BPUs, and (*iii*) enables life-science research involving developing models of the



WHERE ARE EUGLENA RUNNING FROM?
Press W, A, S, or D for up, left, down, or right.
Guess 10 in a row correctly to win.

```
run():
var ids = getEuglenaIds();
for (var i=0; i < ids.length; i++) {
  var pos = getEuglenaPosition(ids[i]);
  drawRect(pos.x-5, pos.y-5,
           pos.x+15, pos.y+5);
}
```

```
run():
var vxCurr = getEugenaVelocity(3);
drawLine(10, prevVelocity,
         20, vxCurr);
setLED(LED.RIGHT, 1000);
```

```
run():
drawText("Number correct: " +
  this.score, 10, 10,
  0.5*MAX_TEXT_SIZE, COLORS.GREEN);
drawInstructionText();
setLED(this.led, MAX_LED_INTENSITY);
```

```
onKeyPress(key):
if (getLEDFromKey(key) === this.led) {
  this.score = this.score + 1;
  this.led = chooseRandomLED();
}
else { alert("Wrong!"); }
```

```
start():
this.led = chooseRandomLED();
this.score = 0;
setJoystickVisible(false);
```

```
end():
setJoystickVisible(true);
alert("Game over.");
```

```
onJoystickChange(
 direction, intensity):
   {no code}
```

Fig. 4. Versatile biological applications can be created using 100 lines of code or fewer (study S1a). The code in *A* is completely depicted. The code presented in *B* and *C* is simplified for illustration; the full code can be found in *SI Appendix*, sections S10 and S11, respectively. (*A*) *Euglena* can be tracked by their ID in real time. The code obtains all of the organism IDs, iterates through them, and draws a rectangle around them. This code is further abstracted into a helper function that can be reused across programs. (*B*) Real-time data visualization, such as the average velocity of the organisms over time, can be plotted. The code extracts every organism's velocity and uses the drawing functions to make a real-time plot of the average velocity. (*C*) The "guess the LED" game asks the user to guess which LED is shining based on the movement direction of the *Euglena* swarm. The code makes use of four of the five supported event handlers. The start handler initializes global variables and hides the joystick input; the end handler displays an end message to the player and unhides the joystick; the run handler draws text on the screen and randomly selects one of the four LEDs to shine light from one direction; the "onKeyPress" handler contains the logic for user key input for guessing which LED is on.

Table 1. Mean completion times for the two preliminary tasks and the free-form task during the on-site study (study S1b); *n* = 7 participants

Task	Average completion time ± SD, min
Modify first program	29.02 ± 14.19
Modify second program	38.15 ± 21.39
Program two new applications	50.32 ± 21.03

Details about the programming tasks are in *SI Appendix*, section S3).

biological system that can be tested side by side against the live experiment.

We implemented the virtual BPU (Fig. 2*F*) with a simple model where animated *Euglena* respond to simulated "setLED" stimuli with negative phototaxis (*Materials and Methods*). We integrated the virtual BPU on the client side, but it could also be implemented on the server side, depending on computational resource limitations on either end. The virtual BPU feeds simulated images into the real-time image processing module.

Results: Use Cases and User Studies

Overview. To evaluate Bioty from the perspective of the programmer and end user, we undertook the four studies in the following order: (S1a) We programmed various example applications ourselves (Fig. 4), (S1b) we let HBI novices (who had no prior experience with interactive biology but prior JavaScript programming experience) develop applications (Table 1 and Fig. 5A), (S1c) we let HBI experts (who had worked with or developed interactive biology before) program applications (Table 2 and Fig. 5 *B* and *C*), and (S2) we let end users interact with an educational program we developed (Fig. 6).

S1a: Example Applications. We first illustrate the programming potential and versatility of Bioty through three applications created by the research team. These applications also provide use cases of the stimulus control, biology sensor, and application creation functions while requiring comparably few lines of code (20, 71, and 30 lines, respectively).

Real-time tracking. This program tracks all live cells on screen and draws a box around each with the tracking ID displayed next to the box (Fig. 4*A* and Movie S2). This program provides direct visualization of the underlying tracking algorithm.

Velocity plot. This program provides a visualization of the instantaneous average *Euglena* velocity (Fig. 4*B*). This program might be useful for real-time data visualization during biophysics experimentation.

Guess the LED game. This program is a "biotic game" (43) where the player has to guess which of the four LEDs is switched on based on the observed direction of *Euglena* swarm movement (Fig. 4*C* and Movie S3), scoring one point for every correct guess. This game might be used to teach about phototactic behavior, also highlighting the few seconds of delay between light stimulus and cell reorientation.

S1b: HBI Novices as Programmers. To assess the accessibility of the paradigm, we recruited participants without previous experience developing interactive biology. Seven programmers aged 21–24 y (mean = 22.6 y, SD = 1.1 y) participated in the study. The only inclusion criterion was previous programming experience. On a scale of 0 ("no experience") to 5 ("expert"), the participants described their experience regarding programming (mean = 3.6, SD = 0.8), JavaScript (mean = 2.6, SD = 1.5), and biology (mean = 2.0, SD = 1.2). The primary purpose of this study was to let these participants develop a variety of example applications which we could then evaluate qualitatively.

All participants worked on site in our laboratory. To first familiarize the participants with this programming paradigm, they completed two structured tasks where they modified existing programs (see Table 1 for completion times as well as *SI Appendix*, section S3: Further description of study S1b). They then performed two free-form programming tasks to determine the types of applications that novices may create (Table 1). All seven participants completed all required structured programming tasks (detailed in *SI Appendix*, section S3) and developed at least one

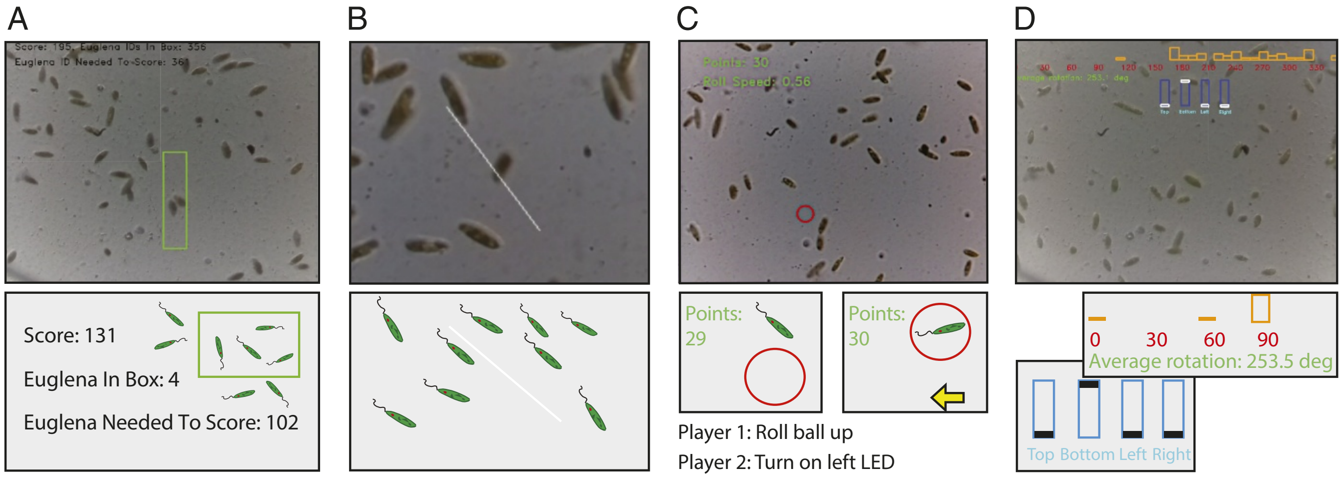


Fig. 5. Study participants (studies S1b and S1c) were enabled to create a variety of interactive biological applications (*A*–*D*, *Top* is screenshot and *A*–*D*, *Bottom* is illustrative picture of program). (*A*) (novice user, study S1b) A video game where the player must get specific *Euglena* into a moving virtual green box, controlled by the player, while the user-directed LEDs shine in the direction of the moving box, making the *Euglena* move away from the target goal. (*B*) (expert user, study S1c) A continuously rotating line visualizing the average orientation of all organisms detected by the BPU. (*C*) (expert user, study S1c) A two-player game where the first player shoots the red ball from a particular position with the aim of hitting as many *Euglena* as possible, while the second player then tries to steer the *Euglena* with the arrow keys (mapped to LEDs) so that the ball avoids as many *Euglena* as possible. (*D*) (semiexpert user) A histogram of *Euglena* rotations on the screen, grouped into bucket sizes of 10°. This program allows the user to change the intensity of the four LEDs by dragging four sliders on the screen. This program was created by an undergraduate who did not participate in any study and who later joined the research team. Mouse click event listeners were not explicitly supported by the current version of Bioty but were added through native JavaScript mouse event handlers on the HTML5 canvas.

Table 2. Coding time and length of the free-form applications expert participants built (study S1c)

Participant	Time, min	Lines of code	Organism API calls	Sensor API calls	Application API calls
1	120	41	2	2	8
2	269*	121	14	12	2
		104	16	2	4
3	137	57	8	5	11
4	351	110	10	5	19
5	77	68	5	0	5

Also included is the number of API calls made in each application, split up by function type. Application details for each program are detailed in *SI Appendix*, section S9.

*Participant 2 created two applications; the reported time is the combined time to create both programs.

free-form application, although some programs had some bugs due to the 2-h time constraint. Task completion times appear in Table 1; no single programming task took more than 1.5 h to complete.

To simulate BPU timing constraints with an increased number of users, we implemented a fixed session time, after which the participants were locked out of the BPU and had to log back in, either to the same BPU or to a new one. The first two participants commented on the inconvenience of this time pressure (example quote: "I have to admit, having the timer count down while I'm writing my code is pretty stressful"). This suggested to us that it would be beneficial to address these physical resource limitations with a virtual development and execution mode. We then implemented a virtual BPU that simulates the main aspects of the real BPU; from both an end-user and a programmer perspective, the virtual and real BPUs are handled equivalently (*Simulation Mode—Virtual BPU* and Fig. 2*F*). All of the following study participants had access to this virtual BPU, including in subsequent studies.

Two notable programs developed by the participants demonstrate biotic games (23) and applications for data collection.

Moving box game. One participant (programming = 4, JavaScript = 3, biology = 1) created a game (Fig. 5*A*; Movie S4) where the player must "capture" one *Euglena* that is specified by its ID into a virtual green box that is controlled by the user. The *Euglena* are stimulated to move away from the box via the joystick-controlled LEDs, which shine in the direction of the moving box's trajectory, making the *Euglena* move away from the target goal.

Swarm movement statistics. Another participant (programming = 4, JavaScript = 4, biology = 2) kept a running average of *Euglena* velocity, acceleration, and rotation over time while randomly varying the direction and intensity of light. The user was able to visualize these aggregate movement statistics on the screen while observing the moving *Euglena* and seeing the effects of the shining LED.

Overall we found that these HBI novices were able to successfully develop versatile applications. In the poststudy questionnaire, seven of seven participants mentioned the ease of use (example quote: "The API was very straightforward and simple to use. It does not take much time to ramp up on the API, which made it fun and way faster to move toward actually using the program"). Optional feedback also indicated that programmers learned new biology during the development process (example quote: "I realized their individual behaviors are really variable; some of them barely respond to light, and some of them respond really quickly"). This suggests that programming with living matter can facilitate experimentation and education online.

S1c: HBI Experts as Programmers. To benchmark the affordances of this development methodology to previously established approaches for creating interactive biology applications (such as refs. 13, 22, 23, and 44), we recruited participants who had developed such applications in the past. Five HBI experts aged 26–33 y (mean = 31.2 y, SD = 3.0 y) were recruited for this study. On a scale of 0 (no experience) to 5 (expert), the participants had a range of backgrounds in programming (3–5, mean = 3.6, SD = 0.9), JavaScript (0–4, mean = 2.4, SD = 1.5), and biology (3–5, mean = 3.4, SD = 0.9).

All participants worked remotely and successfully developed applications of their own choosing, spending between 77 min and 351 min (mean 190.74; median 137 min), using between 42 and 123 lines of code, and using all types (organism, sensor, and application) of available API calls (Fig. 2*C*) with different frequencies (Table 2 and *SI Appendix*, section S9). Participants used both the live BPU and the simulation mode, spending the majority of time (77.6%, SD = 11.4%) in the former. Two notable programs developed by the HBI experts demonstrate how they applied the paradigm to real-time data visualization and game design (23).

Orientation visualization. To provide a way of determining the overall response of the organisms to light stimuli, one of the programmers (programming = 3, JavaScript = 4, biology = 3) created a visualization with a line that rotates in the direction of the

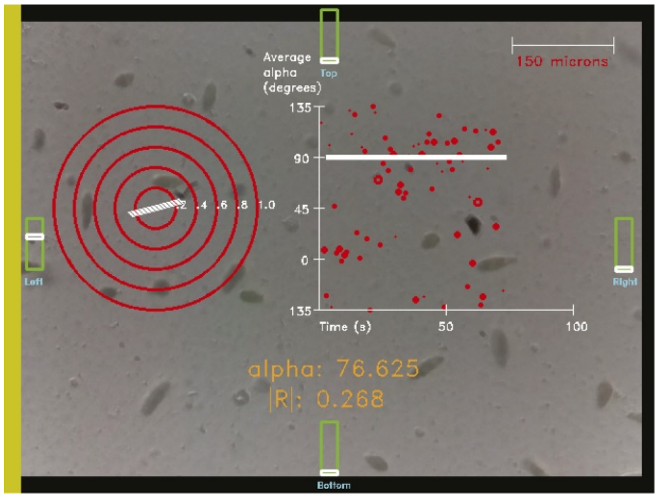


Fig. 6. End users during study S2 tested applications that enabled the interactive interrogation of *Euglena* responses augmented by real-time data visualization. Applications progressed in five stages of complexity. The image shows the final program (for the full progression applications, see *SI Appendix*, Fig. S1). These applications were developed by the research team and were implemented as iterations of the program in Fig. 5*D*. Remote study participants were evaluated based on how they could interact with the developed applications. The guided experimentation platform displays a real-time radial plot of the average orientation of all cells as well as a time series plot of average orientation. Users can change the sliders to adjust the intensity of the LED lights.

average orientation of all detected *Euglena* (Fig. 5*B* and *SI Appendix*, section S9). The program also records the average orientation every 10 s and saves the results to a file.

Basketball game. Another programmer (programming = 3, JavaScript = 3, biology = 5) created a two-player game (Fig. 5*C* and *SI Appendix*, section S9) where the first player lines up a virtual ball using the "A" and "D" keys and shoots it with the space key, aiming to hit as many *Euglena* as possible and scoring a point for every cell that is hit. The second player then uses the W, A, S, and D keys to control LEDs to try to steer the *Euglena* away from the ball in an attempt to minimize the number of points the first player scores.

When asked whether developing their application with Bioty was easier or harder than it would have been using previously existing HBI approaches and tools, all five participants stated that Bioty was much easier. This is also indicated by the relatively low development time and few lines of code (Table 2 and *SI Appendix*, section S12).

S2: HBI Novices as End Users. To test whether programs written in Bioty are ultimately usable for end users, we developed a set of educational applications centered around self-guided science experiments (Fig. 6). These applications were developed by the research team as an iteration of a program created by a semiexpert user (Fig. 5*D*), who was not a member of any study but used Bioty in a hackathon at Stanford University. These applications present moving sliders to control the LED stimulus and real-time visualizations of the average orientation of all organisms depicted through the circular variance (Fig. 3). The main learning goals were as follows: *Euglena* respond to light, *Euglena* orient with the direction of the light, *Euglena* response depends on light intensity, and it takes a few seconds until the *Euglena* have fully responded and reoriented with the light. These applications build up in complexity over five phases (*SI Appendix*, Fig. S1). Fig. 6 and Movies S6 and S7 show the final application in the sequence.

Seventeen remote HBI novices aged 22–55 y (mean = 26.0 y, SD = 8.2 y) were recruited to participate in the guided experimentation platform. Eleven of the participants identified as female and six as male. The participants were asked for their prior familiarity with *Euglena* on a scale from 1 to 5, where 1 corresponds to having never heard of *Euglena*, 3 corresponds to having read about *Euglena* before, and 5 corresponds to working with *Euglena* regularly. The participants' responses ranged from 1 to 4 (mean = 1.8, SD = 0.4). Among these participants were also three regular Eterna (11) players (>30 h of play per month over the past 1.5 y), who were recruited to gather preliminary insight about Bioty's potential as a citizen science platform.

After the first activity, 9/17 participants reported movement toward or away from light. The others reported spinning around their axis. After the last activity, 14/17 participants reported that *Euglena* move away from light. Eleven of 17 participants reported that the intensity of light affects the speed at which the *Euglena* move away from light. Fourteen of 17 participants reported a response time, ranging from 2 s to 1 min (mean = 31.8 s, SD = 8.7 s). The concept of circular variance was harder to grasp for the participants: Only 9/17 correctly stated that "alpha" meant the average orientation of the cells. Overall, participants stated that they learned certain concepts from the applications, such as the fact that *Euglena* have a negative response to light. This demonstrates that applications written in Bioty could support science education.

In the poststudy questionnaire, participants were asked free-form questions about their experience. Regarding the difference between using live and simulated experiments, 11/17 found the simulation to be more predictable and reliable. Nevertheless, 13/17 preferred the live mode over simulation, 1/17 preferred the simulations, and the others did not have a preference. Participants also pointed out that while the live mode interacts with real organisms, the simulation is easier to use. When asked about the advantages of a live experimentation platform, two of the Eterna players responded that it is "more factual" and "What you see is what you get." Hence the feedback combined from all participants is consistent with previous findings [e.g., from the educational literature (45–47)] that both experiments and simulations synergistically motivate scientific inquiry, and the presented IDE supports both.

Discussion

We demonstrated a programming API and IDE to perform real-time interactive experiments with living matter, both locally and on a cloud laboratory and for experts and novices alike. Bioty allows for the specification of interactive experimentation, the program execution to adjust to the biological response via real-time feedback, the integration of a simulation mode, and the creation of interactive programs for remote end users. The API allows for organism sensing via real-time object tracking, organism control through user-controlled highly precise light stimuli enabled through communication with a remote web server hosting BPUs, and application development through a user-friendly drawing and program control library.

The usability and ease of application development using this paradigm were successfully evaluated through multiple user studies. Participants from a variety of backgrounds, both HBI novices and experts, mastered the familiarization tasks and developed applications of their choosing. The applications developed in study S1a demonstrated the feasibility of versatile use cases, e.g., data visualization, automated scientific experiments, interactive scientific experiments, art, and games (Fig. 5). Furthermore, the applications were created rapidly (less than 6 h, with less than 150 lines of code; Table 2). The novice HBI users in study S1a indicated that programming had a low barrier to entry, and the HBI experts in study S1b confirmed easier and faster development than previous approaches (14, 15, 22, 24, 48) (*SI Appendix*, section S5). Study S1c demonstrated that end-user applications (e.g., for science education) can be implemented that leverage the real-time interactivity with the biological substrate. Hence, the Bioty paradigm follows Seymour Papert's vision of interfaces with "low-floor/high-ceiling/wide-walls" (49, 50) and constitutes a significant step toward making experimentation, engineering, and interaction with living matter more accessible to a broader community.

This programming paradigm for living matter and its implemented architecture (Fig. 2) enables versatile future applications in research, fabrication, and education. It generalizes beyond the domain-specific *Euglena* biocomputing used here to other biological, chemical, and physical systems with different control capabilities, e.g., chemically responsive bacteria or molecular motors for swarm robotics and living material fabrication (24, 25, 29–31, 35, 51). Higher spatiotemporal manipulation through more complex light fields and programming abstractions is possible, (e.g., "move cell i right by 5 μm") (24). Citizen science projects like Eterna (11) and Foldit (54) could be supported with enhanced real-time laboratory capabilities. The parallel integration of simulation and live experiments (Fig. 2 *E* and *F*) not only reduces load on the physical experimentation resources but could also afford direct model validation, such as with a whole-cell model as described in ref. 55. Formal and informal science, technology, engineering, and mathematics (STEM) education is in significant need of new approaches and technologies to enable inquiry-based science learning (13, 14, 22, 34, 44, 56–60), which could be supported as well. Augmented reality (61) could deliver versatile and rich worlds as "μAR" in a cost-effective and scalable manner given the small footprint of microbiology. Advancement in high-throughput life-science technologies (62, 63) will increasingly facilitate such applications, considering that

Bioty is run on a cloud laboratory that could already support millions of 1-min long experiments per year at a cost of \$0.01 each (14). Just as personalized computers and programming APIs revolutionized the accessibility and mass dissemination of interactive computing (64), we believe that programming toolkits like Bioty could stimulate equivalent innovations for the life sciences.

Code Availability

The link to all code used to implement the cloud laboratory is publicly available on GitHub at https://github.com/hirklab/euglenalab (65). The code for Bioty is in the feature/bioticGameProgramming branch.

Materials and Methods

Technical Implementation.

Cloud laboratory implementation. The Bioty system is developed over the existing cloud laboratory architecture described in ref. 14. The original implementation contained a joystick which allowed users to control remote LEDs on a BPU. A standard web socket connection is used to send the user joystick commands from the web server and the remote BPU, allowing for real-time interaction.

Each BPU consists of a Raspberry Pi which controls four LEDs. The LEDs are placed over a microfluidic chamber that houses the organisms. The Raspberry Pi is also connected to a Raspberry Pi camera that is placed over a microscope lens facing the microfluidic chamber. The frames from the camera are sent back to the web server and displayed to the user in real time.

Real-time image processing. The client continuously manipulates the frames returned from the microscope's live feed using the Chrome browser's Portable Native Client (PNaCl) toolchain. PNaCl executes native C++ code directly in the browser, performing multiple object tracking via the Kalman filtering algorithm for motion prediction (66) in conjunction with the Hungarian algorithm (67) to continuously match detected object contours. The application control functions render over the HTML5 canvas displaying the live video feed.

Application development support. The programming interface follows an event-driven real-time programming mechanism. When any of the five event programming blocks ("start," "end," "run," "onJoystickChange," or "onKeyPress") are triggered through one of the end-user events, the code is filtered through a parser that removes any nonapproved function calls. The set of approved function calls is the set of API calls plus the standard JavaScript built-in functions. The program throws a compile-time error if a nonapproved function is called via the Caja compiler. For each API function that the parser evaluates, the corresponding backend code is injected into the user code, replacing the user call to the API function. If the input code block passes the prechecks, then the modified code is evaluated as normal JavaScript code, with all built-in language constructs such as looping and program control. If the evaluation of the code throws a runtime error, the execution of the entire script terminates and the error message is displayed to the user.

Some API calls communicate directly with the microscopes, while others perform image processing on the video frames that are returned from the cloud laboratories. This distinction in function implementation is not seen by the user.

When code is saved by a user, the JavaScript code is saved in a formatted file on the system that contains the user code. When a user loads a previously saved program, the formatted file is parsed and placed into the corresponding code blocks on the Bioty user interface.

Simulation mode. The following equations are used as a toy model for the motion of the *Euglena* simulation, where $x(t)$ and $y(t)$ are the positions of a *Euglena*, v is the velocity of a *Euglena* (which is assumed constant but can vary between individual *Euglena*), $\theta(t)$ is the angle of a *Euglena* in the 2D plane, and $\phi(t)$ is the angle of the LED light stimulus, all at time t; δt is the frame rate, and η is random noise:

$$x(t+\delta t) = x(t) + v\cos(\theta(t))\delta t$$
$$y(t+\delta t) = y(t) + v\sin(\theta(t))\delta t$$
$$\theta(t+\delta t) = \theta(t) + [\epsilon\sin(\theta(t) - \phi(t)) + \eta]\delta t.$$

Each *Euglena* is given a random initial position on the screen, a random initial orientation angle, and a constant velocity v sampled from a uniform distribution between 0 and 10 pixels per frame. To calibrate this range of velocities, videos of *Euglena* were analyzed to determine how many pixels on the HTML5 canvas the *Euglena* tended to move through per frame. The frame rate is set to 1 frame per 10 ms. ϵ is the coupling strength, set to -0.3. Each simulated cell is an ellipsoid with a 5:1 major-to-minor axis ratio.

We used periodic boundary conditions: When a Euglena's x position moves past the left or right edge of the screen, it retains its y position, velocity, and orientation, appearing on the other side of the screen. The same method is used when its y position moves past the top or bottom edge of the screen. If a *Euglena* collides with another, as defined by their x and y positions being within 2 pixels of each other, then both *Euglena* are assigned a new random θ.

This is a simple model capturing the basic idea of *Euglena* dynamics in response to light. More sophisticated parameter matching between real and simulated *Euglena* behavior is possible. For example, the *Euglena* model is not currently dependent on light intensity. Furthermore, more complex models capturing the subtleties of *Euglena* movement, such as it 3D polygonal motion, helical swimming pattern, and spinning at high light intensities, are possible, but beyond the scope of this work.

User Studies. All user studies were conducted according to Stanford University IRB-18344. Informed consent was obtained from all participants.

Study S1b—HBI novices as programmers. To evaluate the remote programming of organisms, we started with an on-site study with programmers performing two structured and two free-form programming tasks. Participants were recruited through online mailing lists and self-described programming ability.

Participants were limited to 2 h of total coding time, including familiarizing themselves with the interface and API. The version of Bioty used in this study did not include a virtual simulation mode for the first two participants, but it was provided to the remainder of the participants (including in subsequent studies) in response to feedback about physical resource limitations. Participants worked on site (instead of remotely), as it allowed us to directly observe their actions and interview them.

Seven programmers aged 21–24 y (mean = 22.57 y, SD = 1.13 y) participated in the on-site study. Three of the participants identified as female and four identified as male. Participants were required to be fluent in English and came from a variety of academic backgrounds. Three participants were undergraduate students at Stanford University, three were full-time software developers, and one was a clinical researcher with some coding experience.

In the recruiting form, participants were asked to describe their general programming experience, their JavaScript programming experience, and their general biology experience on a scale from 0 (no experience) to 5 (expert). The participants' stated programming experience ranged from 2 to 4 (mean = 3.57, SD = 0.84), JavaScript experience ranged from 0 to 4 (mean = 2.57, SD = 1.51), and biology experience ranged from 1 to 4 (mean = 2.0, SD = 1.17). One participant (biology = 3) had prior experience working with *Euglena*. Five of the seven participants had little to no biology knowledge (rating of 1 or 2).

Before starting the familiarization tasks (*SI Appendix*, section S3), participants were shown a working demonstration of the applications that they were asked to modify. The study researchers were available to answer questions about the API and web interface logistics, but answers to the programming tasks were not provided. After the completion of the first set of structured tasks, participants were asked to complete two free-form programming tasks. Data were recorded on an online Google Form. See *SI Appendix*, section S3 for more details about the study, including the full set of questions participants were asked.

Study S1c—HBI experts as programmers. To gather information about system use by domain experts, we recruited participants with prior *Euglena* HBI development backgrounds to program applications on the platform over an extended period. We aimed to compare their prior experiences with their experience with the remote paradigm.

Five HBI experts (i.e., people who have previously developed *Euglena*-based HBI applications) aged 26–33 y (mean = 31.2 y, SD = 3.0 y) were recruited for the remote portion of the study. The participants were known by the authors beforehand as established HBI experts. Two of the participants identified as female and three identified as male. Three participants were graduate students at Stanford University, one was a full-time software developer, and one was a postdoctoral scholar in biophysics. The participants' stated programming experience ranged from 3 to 5 (mean = 3.6, SD = 0.9), JavaScript experience ranged from 0 to 4 (mean = 2.4, SD = 1.5), and biology experience ranged from 3 to 5 (mean = 3.4, SD = 0.9).

The procedure for the study with HBI experts was identical to that for the study with novices (study S1b), except for where noted here. The participants were asked to perform two structured and two free-form programming tasks. However, this time the participants were not limited to 2 h of total coding time, instead having 1 wk to complete their programs. The HBI experts were also provided a simulation mode, in response to the feedback from study S1b. Logs of the total time actually spent on the IDE were recorded. See *SI Appendix*, sections S3 and S4 for more details about the study, including the full set of questions participants were asked.

Study S2—HBI novices as end users. To test whether programs written using this paradigm are ultimately usable for end users, 17 remote HBI novices aged 22–55 y (mean = 26.0 y, SD = 8.2 y) were recruited to participate in the guided experimentation platform. Participants were recruited through the Stanford University email lists and the Eterna news feed. The remote participants were provided with instructions for interacting with the guided experimentation platform. To verify the usability of the programs developed in an independent online setting, no help with the interface was provided by researchers at any point during the study. The simulation mode was implemented for this study. Participants were asked to use both the simulation and live modes.

The guided experimentation platform was broken up into six submodules. The first five submodules corresponded to a developed application. These submodules built off of each other to progressively teach the student more about *Euglena* movement patterns through increasingly interactive programs (see *SI Appendix*, Fig. SI1 for details). The final submodule asked students to perform an experiment to determine how long it takes *Euglena* to respond to light, as defined by the time it takes for the average orientation of the organisms to have a consistently low circular variance. Note that the current implementation of Bioty (Figs. 2 and 3) does not directly support mouse touch events, but the touch event functionality can be implemented within Bioty indirectly by manipulating the HTML5 canvas using native JavaScript code, which was done here. Adding an explicit touch event handler will be added as a feature in a future iteration of Bioty.

To analyze the qualitative results, two raters categorized all quotes. A first rater initially categorized the quotes, followed by a second rater who confirmed the first rater's categorizations. When there was disagreement, the two raters discussed the categorization of quotes.

See *SI Appendix*, section S5 for more details about the study.

ACKNOWLEDGMENTS. The authors thank Z. Hossain, T. R. Stones, R. Das, and members of the I.H.R.-K. laboratory for suggestions; all volunteers who tested the system in various stages; and NSF Cyberlearning (1324753) for funding.

1. Sia SK, Owens MP (2015) Share and share alike. *Nat Biotechnol* 33:1224–1228.
2. Wang L, et al. (2008) Scientific cloud computing: Early definition and experience. *Tenth IEEE International Conference on High Performance Computing and Communications, 2008. HPCC'08* (IEEE), pp 825–830.
3. Hoffa C, et al. (2008) On the use of cloud computing for scientific workflows. *IEEE Fourth International Conference on eScience, 2008. eScience'08* (IEEE), pp 640–645.
4. Keahey K, Figueiredo R, Fortes J, Freeman T, Tsugawa M (2008) Science clouds: Early experiences in cloud computing for scientific applications. *Cloud Computing and Applications, 2008*, pp 825–830.
5. Bloch J (2006) How to design a good API and why it matters. *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications* (ACM), pp 506–507.
6. Corbató FJ, Merwin-Daggett M, Daley RC (1962) An experimental time-sharing system. *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference* (ACM), pp 335–344.
7. Murty J (2008) *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB* (O'Reilly Media, Sebastopol, CA).
8. Zahariev A (2009) *Google App Engine* (Helsinki Univ of Technology), pp 1–5.
9. Wilder B (2012) *Cloud Architecture Patterns: Using Microsoft Azure* (O'Reilly Media, Sebastopol, CA).
10. Check Hayden E (2014) The automated lab. *Nature* 516:131–132.
11. Lee J, et al. (2014) RNA design rules from a massive open laboratory. *Proc Natl Acad Sci USA* 111:2122–2127.
12. Khatib F, et al. (2011) Algorithm discovery by protein folding game players. *Proc Natl Acad Sci USA* 108:18949–18953.
13. Hossain Z, et al. (2017) Design guidelines and empirical case study for scaling authentic inquiry-based science learning via open online courses and interactive biology cloud labs. *Int J Artif Intell Educ* 28:478–507.
14. Hossain Z, et al. (2016) Interactive and scalable biology cloud experimentation for scientific inquiry and education. *Nat Biotechnol* 34:1293–1298.
15. Hossain Z, et al. (2015) Interactive cloud experimentation for biology: An online education case study. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (ACM), pp 3681–3690.
16. Hossain Z, Riedel-Kruse IH (2018) Life-science experiments online: Technological frameworks and educational use cases. *Cyber-Physical Laboratories in Engineering and Science Education* (Springer, Cham, Switzerland), pp 271–304.
17. Whitehead E, Rudolf F, Kaltenbach H-M, Stelling J (2018) Automated planning enables complex protocols on liquid-handling robots. *ACS Synth Biol* 7:922–932.
18. Gupta V, Irimia J, Pau I, Rodríguez-Patón A (2017) Bioblocks: Programming protocols in biology made easier. *ACS Synth Biol* 6:1230–1232.
19. Zinsser WK (1983) *Writing with a Word Processor* (Harper & Row, New York).
20. O'rourke TC, et al. (1994) Graphical user interface. US Patent 5,349,658 (September 20, 1994).
21. Heidel R, Snider RE (1994) Touch screen video gaming machine. US Patent 5,342,047 (August 30, 1994).
22. Lee SA, et al. (2015) Trap it!: A playful human-biology interaction for a museum installation. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (ACM), pp 2593–2602.
23. Cira NJ, et al. (2015) A biotic game design project for integrated life science and engineering education. *PLoS Biol* 13:e1002110.
24. Lam AT, et al. (2017) Device and programming abstractions for spatiotemporal control of active micro-particle swarms. *Lab Chip* 17:1442–1451.
25. Jin X, Riedel-Kruse IH (2018) Biofilm lithography enables high-resolution cell patterning via optogenetic adhesin expression. *Proc Natl Acad Sci USA* 115:3698–3703.
26. Glass DS, Riedel-Kruse IH (2018) A synthetic bacterial cell-cell adhesion toolbox for programming multicellular morphologies and patterns. *Cell* 174:649–658.
27. Frangipane G, et al. (2018) Dynamic density shaping of photokinetic E. coli. *eLife* 7:e36608.
28. McCarty NS, Ledesma-Amaro R (2018) Synthetic biology tools to engineer microbial communities for biotechnology. *Trends Biotechnol* 37:181–197.
29. Tabor JJ, et al. (2009) A synthetic genetic edge detection program. *Cell* 137:1272–1281.
30. Yao L, et al. (2015) Biologic: Natto cells as nanoactuators for shape changing interfaces. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (ACM), pp 1–10.
31. Nakamura M, et al. (2014) Remote control of myosin and kinesin motors using light-activated gearshifting. *Nat Nanotechnol* 9:693–697.
32. Rehman RU, Paul C (2003) *The Linux Development Platform: Configuring, Using, and Maintaining a Complete Programming Environment* (Prentice Hall Professional, Upper Saddle River, NJ).
33. Owens JD, et al. (2008) GPU computing. *Proc IEEE* 96:879–899.
34. Hossain Z, Bumbacher E, Blikstein P, Riedel-Kruse I (2017) Authentic science inquiry learning at scale enabled by an interactive biology cloud experimentation lab. *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale* (ACM), pp 237–240.
35. Comer DE, et al. (1989) Computing as a discipline. *Commun ACM* 32:9–23.
36. Diehn B (1969) Phototactic response of Euglena to single and repetitive pulses of actinic light: Orientation time and mechanism. *Exp Cell Res* 56:375–381.
37. Tsang ACH, Lam AT, Riedel-Kruse IH (2018) Polygonal motion and adaptable phototaxis via flagellar beat switching in the microswimmer Euglena gracilis. *Nat Phys* 14:1216–1222.
38. Resnick M, et al. (2009) Scratch: Programming for all. *Commun ACM* 52:60–67.
39. Wong J, Hong J (2006) Marmite: End-user programming for the web. *CHI'06 Extended Abstracts on Human Factors in Computing Systems* (ACM), pp 1541–1546.
40. Myers B, Ko A (2003) Studying development and debugging to help create a better programming environment. *CHI 2003 Workshop on Perspectives in End User Development* (Fort Lauderdale, FL) , pp 65–68.
41. Opher E, Niblett P, Luckham DC (2011) *Event Processing in Action* (Manning Greenwich, Shelter Island, NY).
42. Overmars M (2004) Teaching computer science through game design. *Computer* 37:81–83.
43. Riedel-Kruse IH, Chung AM, Dura B, Hamilton AL, Lee BC (2011) Design, engineering and utility of biotic games. *Lab Chip* 11:14–22.
44. Kim H, et al. (2016) LudusScope: Accessible interactive smartphone microscopy for life-science education. *PLoS One* 11:e0162602, and erratum (2016) 11: e0168053.
45. Doerr HM (1997) Experiment, simulation and analysis: An integrated instructional approach to the concept of force. *Int J Sci Educ* 19:265–282.
46. Ma J, Nickerson JV (2006) Hands-on, simulated, and remote laboratories: A comparative literature review. *ACM Comput Surv* 38:7.
47. de Jong T, Linn MC, Zacharia ZC (2013) Physical and virtual laboratories in science and engineering education. *Science* 340:305–308.
48. Kim H, Gerber LC, Riedel-Kruse IH (2016) Interactive biotechnology: Building your own biotic game setup to play with living microorganisms. *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (ACM), pp 1000–1002.
49. Papert S (1980) *Mindstorms: Children, Computers, and Powerful Ideas* (Basic Books, New York).
50. Resnick M, Silverman B (2005) Some reflections on designing construction kits for kids. *Proceedings of the 2005 Conference on Interaction Design and Children* (ACM), pp 117–122.
51. Katz E (2015) Biocomputing—Tools, aims, perspectives. *Curr Opin Biotechnol* 34:202–208.
52. Ozasa K, Lee J, Song S, Hara M, Maeda M (2011) Two-dimensional optical feedback control of Euglena confined in closed-type microfluidic channels. *Lab Chip* 11:1933–1940.
53. Beni G (2004) From swarm intelligence to swarm robotics. *International Workshop on Swarm Robotics* (Springer), pp 1–9.
54. Eiben CB, et al. (2012) Increased diels-alderase activity through backbone remodeling guided by Foldit players. *Nat Biotechnol* 30:190–192.
55. Karr JR, et al. (2012) A whole-cell computational model predicts phenotype from genotype. *Cell* 150:389–401.

56. Dede CJ, Jacobson J, Richards J (2017) Introduction: Virtual, augmented, and mixed realities in education. *Virtual, Augmented, and Mixed Realities in Education* (Springer), pp 1–16.
57. Huang A, et al. (2018) Biobits^{TM} explorer: A modular synthetic biology education kit. *Sci Adv* 4:eaat5105.
58. Stark JC, et al. (2018) Biobits^{TM} bright: A fluorescent synthetic biology education kit. *Sci Adv* 4:eaat5107.
59. Cybulski JS, Clements J, Prakash M (2014) Foldscope: Origami-based paper microscope. *PloS One* 9:e98781.
60. Auer ME, Azad AKM, Edwards A, de Jong T (2018) *Cyber-Physical Laboratories in Engineering and Science Education* (Springer, New York).
61. Milgram P, Takemura H, Utsumi A, Kishino F (1995) Augmented reality: A class of displays on the reality-virtuality continuum. *Telemanipulator and Telepresence Technologies* (International Society for Optics and Photonics), Vol 2351, pp 282–293.
62. Balagaddé FK, You L, Hansen CL, Arnold FH, Quake SR (2005) Long-term monitoring of bacteria undergoing programmed population control in a microchemostat. *Science* 309:137–140.
63. Skilton RA, et al. (2015) Remote-controlled experiments with cloud chemistry. *Nat Chem* 7:1–5.
64. Nichols LM (1992) The influence of student computer-ownership and in-home use on achievement in an elementary school computer programming curriculum. *J Educ Comput Res* 8:407–421.
65. Washington P, et al. (2018) Data from "Bioty source code." GitHub. Available at https://github.com.hirklab/euglenalab. Deposited January, 18, 2018.
66. Li X, Wang K, Wang W, Li Y (2010) A multiple object tracking method using kalman filter. *2010 IEEE International Conference on Information and Automation (ICIA)* (IEEE), pp 1862–1866.
67. Huang C, Wu B, Nevatia R (2008) Robust object tracking by hierarchical association of detection responses. *European Conference on Computer Vision* (Springer), pp 788–801.