# Bioty: A cloud-based development toolkit for programming experiments and interactive applications with living cells

**Peter Washington, Karina G. Samuel-Gama, Shirish Goyal, Ingmar H. Riedel-Kruse***
Department of Bioengienering, Stanford University
318 Campus Drive, Stanford, CA 94305, USA.
*ingmar@stanford.edu

## ABSTRACT
Life-science research is increasingly accelerated through instrumentation advancements - even supporting the real-time interaction with living matter in the cloud. This enables entirely new applications for research, education, and art. The creation of such applications is currently challenging and time consuming, even for experts, as it requires knowledge of biotechnology, hardware design, and software development in addition to continuous access to reliable and responsive biological materials. We therefore developed Bioty, a JavaScript-based web toolkit enabling rapid prototyping of versatile Human-Biology Interaction (HBI) applications utilizing swarms of motile, phototactic cells hosted on a remote cloud lab. We evaluated and demonstrated the utility of Bioty through user studies with both HBI novices and experts, who used Bioty to create applications such as interactive scientific experiments, biotic video games, and biological art. Our findings inform the design of development toolkits for programming the behavior of biological matter.

### Author Keywords
Life-science automation; Human-Biology Interaction (HBI); Human-Computer Interaction (HCI); Interactive Biotechnology; Prototyping Toolkit; Cloud Lab; Swarm Programming

## INTRODUCTION
Life-science research is increasingly accelerated through the advancement of programmable instruments [42], some of which are even accessible online [18, 20, 42]. This evolution parallels early pioneering computer science research, where time sharing approaches [6, 13], domain specific programming languages [23, 24], and Human-Computer Interaction (HCI) design methods were developed and refined [1]. Living matter is distinct from electronic computation as it allows for chemical synthesis and generation of active physical forms [2, 3, 4]. Hence new user interfaces and programming languages are desired that are tailored to these properties, and that ease the experimentation, engineering, and human interaction with living matter.

Human-Biology Interaction (HBI), in analogy to HCI which aims to enable humans to interact with microscopic biological materials [33]. For example, a human can influence the swimming direction of phototactic *Euglena gracilis* cells in real time by activating light stimuli [20, 33]. This turns classical observational microscopy into an interactive experience [27]. The utility of HBI has been demonstrated through multiple applications for formal and informal education, e.g., as museum exhibits [33], biology cloud labs [19, 20, 21], biotic games [5, 40], and Do-It-Yourself Biology (DIYbio) kits [26]. Future engineering applications such as for biological swarm control have been demonstrated [30], and applications for discovery research are in reach [9, 32, 20].

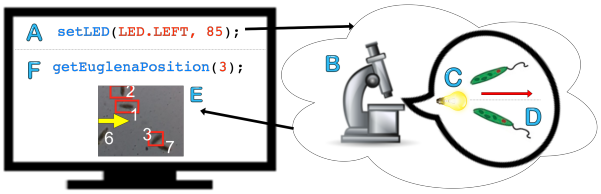


**Figure 1. Bioty is a JavaScript-based web toolkit that allows users to rapidly develop versatile applications (science experiments, biological art, biotic video games) that interact with living microorganisms housed in a remote biology cloud lab. Users program stimulus commands (A) that are sent to cloud lab microscopes (B). These commands actuate one of four LEDs (C) that drive photophobic Euglena cells (D) along the four cardinal directions. The live video stream is displayed to the user (E) and augmented with virtual objects, e.g., outlining cells provide by sensor commands (F).**

All of these applications require 'Interactive Biotechnology,' i.e., specialized hardware, software and bioware that enables continuous access to reliable and responsive biological materials. Mastering all these disciplines poses a high barrier to entry. First steps have been taken toward standardized tools and platforms: (1) The single-celled, phototactic organism *Euglena gracilis* exhibits robust and intricate responses to diverse light stimuli within seconds and can be cultured automatically for weeks [33]. (2) Cloud labs can house biological material at a centralized location and enable remote execution of simple biology experiments [20]. (3) Higher-level programming abstractions and logic for biological swarm control can be implemented to support application development [30]. Consequently, we identified an opportunity to integrate biological swarm programming into a proper user-facing software development environment, and to integrate them into a biology cloud lab.

Here we present Bioty, a novel JavaScript-based web toolkit that allows HBI experts and novices alike to remotely program versatile applications for real-time interactions with swarms of single-celled microorganisms (Fig. 1). The name Bioty is inspired by the Unity game engine [10], as Bioty can be thought of as an Interactive Biotechnology engine.

Our work makes the following contributions:

1. We present the first web toolkit for the development and rapid prototyping of remotely operable HBI applications, thereby lowering the access barriers for experts and novices.
2. On-site and remote user studies reveal preferences for programming with real and virtual swarms.
3. We demonstrate the toolkit's versatility with applications created by the authors and study participants.

## RELATED WORK

To our knowledge, Bioty is the first published development interface that enables programmers to both remotely and rapidly prototype biological applications involving a stimulus and response from living microorganisms in real time. Our work is motivated by numerous examples in previous HBI and swarm robotics literature.

### Broad Visions

Ivan Sutherland's vision for the Ultimate Display as "a room within which the computer can control the existence of matter" [43] can be applied to matter that is not visible to the human eye, such as microorganisms. Hiroshi Ishii's vision of Radical Atoms [25] is popular within tangible user interface research, and describes hypothetical reconfigurable materials described by real-time digital models such that changes in digital information can be reflected by a corresponding change in physical state and vice-versa. We consider Bioty as a substantial step towards making these visions a reality in the life sciences.

### Human-Computer Interaction in the Life Sciences

There are a number of professional, artistic, academic, and laymen intersections between the life sciences and human-computer interaction (HCI). Ken Goldberg's TeleGarden [15] was an art installation that allowed users to interact with a remote garden filled with living plants. Lee's TrapIt! [33] enabled museum visitors to playfully interact with living cells. Commercial and academic cloud labs that enable remote users to design and run life-science experiments have been deployed for research [18, 32, 42] and education [19, 20], e.g., Lee et al. [32] enabled citizen science to help solve RNA structures, and Hossain et al. [19, 20] brought inquiry-based science learning with true experiments to classrooms and massive open online courses (MOOCs). Bioty is built on top of this cloud lab approach.

DIYbio systems are developed for laymen and academic settings and have demonstrated effectiveness in both educational and scientific contexts. Fernando et al. [12] have explored practicing DIYbio in an HCI setting via an autoethnographic study in a functional DIYbio space within an HCI design studio, ultimately identifying three research trajectories for future DIYbio research in HCI: (1) platforms for scaffolding biology expertise, (2) low-cost DIYbio platforms, and (3) systems for crowdsourcing biology research. Kuznetsov et al. [29] have envisioned DIYbio platforms in HCI as rich tools for expertise sharing as well as instruments for public engagement with science. Taylor et al. [44] explored a computational biology tool from an HCI perspective and suggested that the conceptual problems that exist when using tools that model biology can be seen as opportunities to view design and HCI within an orientation that contests science as a monistic endeavor and supports instead an emergent idea of scientific ways of knowing. Kim et al. [26] developed the LudusScope, a smartphone-compatible DIY microscope that enables HBI.

Several best-practice design rules for HBI have been established [14, 19, 33] that are applicable to Bioty. The most central of these rules are: (1) The biological material needs to have obvious and robust responses to the user stimulus; (2) the UI should emphasize to the user that these are real living organisms and not a simulation - augmentation with overlaid virtual objects can help, and (3) inherent biological properties, such as biological variability or natural imagery, should be featured as prominent components of any application, as otherwise a pure simulation might be of similar value to the user while being more easily deployed.

### Swarm Control

Controlling collections of active particle swarms is a wide field, encompassing the programmed self-assembly of swarm robotics [41], novel approaches to tissue engineering [45], and interactions with physical robotic swarm user interfaces [28, 31]. Previous work on Zooids [31] identified four properties of swarm UIs: (1) the UI elements must be physically detached from each other, (2) the elements need to be self-propelled, (3) the elements need to move collectively, and (4) the elements need to be able to react to user input. The biological organisms utilized in Bioty, *Euglena gracilis*, fulfill all of these properties due to their collective response to light stimuli that are provided by human input. Lam et al. [30] explored higher-level programming abstractions with these organisms. Bioty now takes these programming concepts to the next level by extending and integrating them into a proper user-facing software development environment for rapidly prototyping.

Computer-based simulations of swarms have existed for many years, embodied by educational technology systems such as NetLogo [46] and StarLogo [37]. These simulations are based on existing theoretical models derived through observation and analysis of real data, while Bioty allows users to analyze real microorganism swarms in order to construct a working model of organism behavior themselves. Note that Bioty also provides users with the choice to access a simulated cloud lab to lower the time required to access the physical cloud resource.

## PROTOTYPING WITH BIOTY

### Overview

The Bioty user interface (Fig. 2) contains a programming area on the left side of the display (Fig. 2A) and the output program on the right side (Fig. 2B).

The programming interface is separated into five distinct areas (some of which are shown in Fig. 2C) corresponding to an event in the execution of the program, a standard practice in event-driven programming [11, 35]. Correspondingly, all program code then goes into one of five code areas, each of

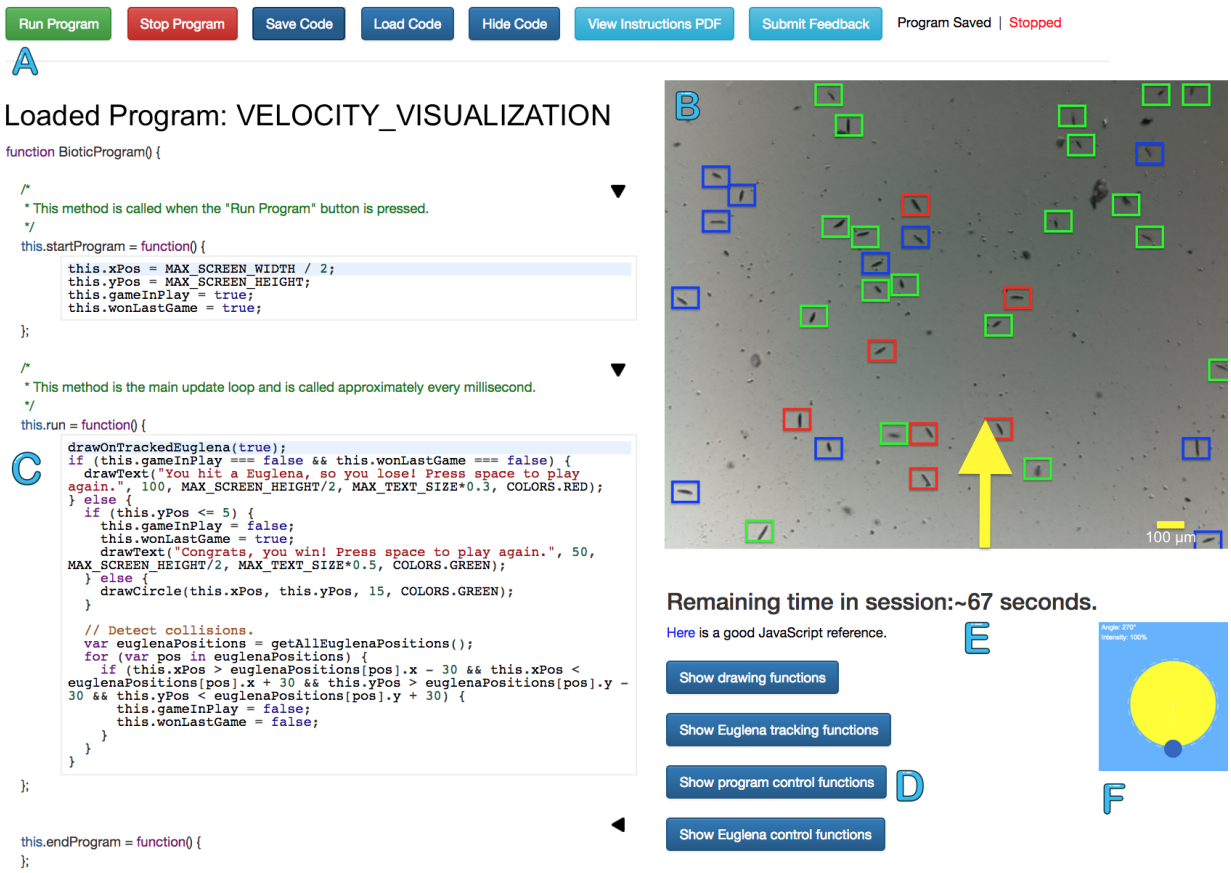


**Figure 2. The primary Bioty interface. (A) The tool bar at the top allows the user to start/stop the program and save/load their code. The user can also hide/show their code in order to preview the final prototype without seeing the underlying code. (B) Live microscope video feed, with virtual objects overlaid on the frames. (C) 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. Not all code boxes are shown here for space. (D) API calls are displayed on the interface. The functions are organized by type and can be expanded and collapsed. (E) Time remaining in the session is displayed to the user. (F) 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.**

which can be displayed or hidden. Each code box corresponds to a separate event-driven function:

*startProgram*(): This function fires at the beginning of program execution.

*endProgram*(): This function fires when the user terminates the program.

*run*(): This function continuously fires during the execution of the user program. The function fires at a rate of approximately once per millisecond, although this rate varies with network connectivity.

*onKeypress*(*key*): This function fires when the user presses a key on the keyboard.

*onJoystickChange*(*angle, intensity*): This function fires when the user moves the virtual joystick and continues firing at the same rate as the *run*() function as long as the user is moving the virtual joystick.

Users have access to a JavaScript Application Programming Interface (API) for manipulating the organisms (see Bioty API section below). The API calls are displayed on the user interface (Fig. 2D) and can be expanded and collapsed by the user. On the toolbar (Fig. 2A), the user has the option to press a "Start Program" button and a "Stop Program" button, which trigger the *startProgram* and *endProgram* events, respectively. The user can save their program for later by clicking the "Save Code" button and can also load a previously saved program via the "Load Program" button. Any saved user code could be loaded onto any other online microscope. This strategy allows the user to test their prototypes on a variety of physical settings, with each cloud lab microscope having different magnifications (4x, 10x, or 20x), different *Euglena* concentrations, and various background lighting conditions.

Given that only a limited number of microscopes are available in the cloud lab, it is necessary to restrict the session time for user - typically to 15 minutes at a time. The corresponding information is provided to the user (Fig. 2E), and the user can directly renew thereafter if a microscope is available, or use a simulation-based sandbox mode instead.

The user is also provided with a virtual joystick (Fig. 2F). The

**Stimulus Control Function**

`setLED(led, intensity)`

**Organism Sensor Functions**

`getAllEuglenaIDs()`

`getAllEuglenaPositions()`

`getEuglenaCount()`

`getEuglenaInRect(upperLeftX, upperLeftY, lowerRightX, lowerRightY)`

`getEuglenaAcceleration(id)`

`getEuglenaPosition(id)`

`getEuglenaRotation(id)`

`getEuglenaVelocity(id)`

**Application Creation Functions**

`drawOnTrackedEuglena()`

`drawCircle(x, y, radius, color)`

`drawLine(x1, y1, x2, y2, color)`

`drawText(text,xPos,yPos, size, color)`

`drawRect(x1, y1, x2, y2, color)`

`endProgram()`

`getTimeLeft()`

`setJoystickVisible()`

`setInstructionText(message)`

`readFromFile(filename)`

`writeToFile(filename, text, mode)`

**Figure 3. The complete Bioty JavaScript API, organized by function type.**

joystick angle and magnitude (the length the user has dragged the joystick) can be mapped to program stimuli, for example, the directional illumination and intensity of the LEDs on the microscopes.

## Bioty API

Programming in Bioty occurs in JavaScript. The user is provided with a set of API functions (Fig. 3) that enable the creation of interactive, microbe-based biotechnologies. We classify the Bioty API functions into three fundamental types:

### *Stimulus Control*

The current implementation of Bioty has one function to control LED stimuli along four cardinal directions, which then affects the organism's movement, as *Euglena* tend to swim away from the light source.

This function is *setLED(led, intensity)*, where *led* is one of LED.UP, LED.DOWN, LED.RIGHT, or LED.LEFT, and *intensity* is an integer between 0 and 1000. More stimulus functions could be added, providing more fine-grained control of LED stimuli or providing another type of stimulus that the microorganisms can respond to (e.g., a chemical stimulus).

### *Organism Sensors*

There are a number of API functions that provide object tracking information about the cells, allowing users to access processed data such as a particular *Euglena*'s X and Y positions, velocity magnitude (in microns per second), acceleration (in microns per second squared), and rotation (in degrees). Users can also access total organism count as well as the organism count in a particular region.

### *Application Creation*

There are several API calls that are not specific to biological programming but are useful for creating an interactive application. These include functions for drawing shapes and text on the screen, functions for file input and output, and timing functions.

## Scenario: Collection of Biophysics Data

We now illustrate how a user may program with Bioty by walking through an example scenario. Amanda is a scientist who wishes to write an automated script to collect data for a biophysics research project. Specifically, she aims to come up with an equation to model the two-dimensional movement of *Euglena* in response to various intensities of light. Further, she wishes to identify in real time which *Euglena* are turning around their short axis in the image plane to align with the direction of light, and she wishes to mark these cells by drawing a red box around them (Fig. 4). We note that such an experiment is realistic, as similar experiments have been published in microbiology and biophysics journals [7, 8, 16].

Amanda first wants to map joystick interactions to LED behavior. She therefore modifies the *onJoystickChange* event handler so that the direction of the joystick determines which LED is shining and the length that the joystick is dragged is proportional to the intensity of light:

```
if (angle >= 45 && angle < 135)
    setLED(LED.UP, intensity);
else if (angle >= 135 && angle < 225)
    setLED(LED.RIGHT, intensity);
else if (angle >= 225 && angle < 315)
    setLED(LED.DOWN, intensity);
else if (angle >= 315 || angle < 45)
    setLED(LED.LEFT, intensity);
```

She next aims to collect the velocity and rotation of each *Euglena* detected on the screen every 5 seconds. She first calls a helper function to verify the initial conditions of the microfluidic chamber and initializes some global variables to keep track of the collected data:

```
verifyInitialState();
this.time = 1;
this.recordTime = 0;
this.velocitiesOverTime = {};
this.rotationsOverTime = {};
```

The code for the helper function is:

```
var initialStateSufficient = true;
if (getEuglenaCount() < 4)
  initialStateSufficient = false;
var euglenaIDs = getAllEuglenaIDs();
for (var i = 0; i < euglenaIDs.length; i++) {
  var id = euglenaIDs[i];
  var eugVel = getEuglenaVelocity(id);
  if (eugVel <= 0.001 && eugVel > 0)
    initialStateSufficient = false;
}
if (!initialStateSufficient)
  alert("Initial conditions failed");
```

Amanda then calls the *getAllEuglenaIDs* function in the *run* update event handler every 5 seconds, iterating through the tracked organism IDs at that time to get their velocity and rotation via the *getEuglenaVelocity* and *getEuglenaRotation* functions, respectively. She also draws a red box around *Euglena* that have rotated more than 15 degrees in the past 5 seconds:

```
this.time++;
if (this.time % 5000 === 0) {
  this.time = 1;
  var euglenaIDs = getAllEuglenaIDs();
  for (var i = 0; i < euglenaIDs.length; i++) {
    var id = euglenaIDs[i];
    var timestamp = Date.now();
    this.velocitiesOverTime[this.recordTime][id]
      = getEuglenaVelocities(id);
    this.rotationsOverTime[this.recordTime][id]
      = getEuglenaRotation(id);
    if (this.recordTime > 1 &&
      Math.abs(this.rotationsOverTime
      [this.recordTime][id] -
      this.rotationsOverTime[this.recordTime-1]
      [id]) > 15) {
      var xPosition = getEuglenaPosition(id).x;
      var yPosition = getEuglenaPosition(id).y;
      drawRect(xPosition-5, yPosition-5,
        xPosition+5, yPosition+5, COLORS.RED);
    }
    this.recordTime++;
  }
}
```

Finally, Amanda wishes to save the experiment results into a JSON file for future data analysis. In the *onKeypress* event handler, she maps the 'S' key to saving the accumulated experimental data:

```
switch (key) {
  case KEY.S:
    writeToFile("velocities.json",
      this.velocitiesOverTime, FILE.OVERWRITE);
    writeToFile("rotations.json",
      this.rotationsOverTime, FILE.OVERWRITE);
    break;
}
```

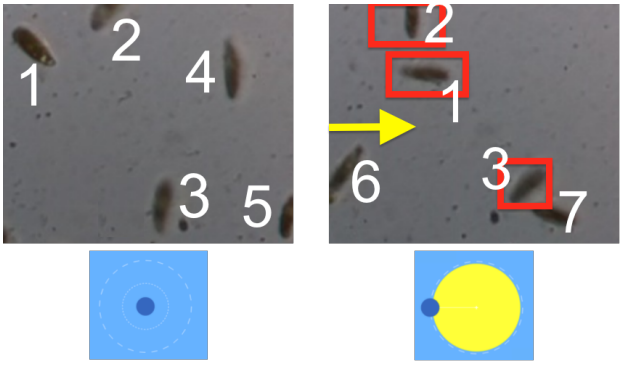


**Figure 4. In the biophysics data collection example, Amanda highlights *Euglena* with red boxes that have moved in response to pulling the joystick. The yellow arrow shows the direction that the LED is shining.**

The total amount of code required for this application is relatively small, at only 38 lines or 50 lines including the helper function to verify the initial state. Amanda can now use her program to interact with the *Euglena* in real time with the joystick to collect data about how *Euglena* move in response to various intensities and directions of light. After collecting the data with her Bioty program, Amanda can analyze the data offline and use it to develop and test her biophysics model about *Euglena* phototaxis for her research project.

## SYSTEM IMPLEMENTATION

The overall system architecture appears in Fig. 5. The API calls that communicate directly with the microscopes do so as in [20], via a standard websocket connection between the web server and the remote lab server. We therefore do not describe this cloud lab implementation in detail here, but briefly, the cloud labs consist of a series of microscopes with a Raspberry Pi camera looking down the lens to a microfluidic chip housing *Euglena*. The microfluidic chip has 4 LEDs placed on each cardinal direction.

When any of the event-driven programming blocks are triggered, the code is filtered through a parser that removes any non-approved function calls. The set of approved function calls are the set of Bioty API calls plus a set of basic JavaScript built-in methods: *Math*, *Date*, *Array*, *String*, *Boolean*, and *Number* methods. If another undefined method is in the code, the program will not run and will instead throw an error message when the user tries to start the program. For each Bioty API function that the parser evaluates, the corresponding backend code is injected into the user code and replaces the user call to the API function.

If the input code block passes the pre-checks mentioned above, then the modified code is evaluated as normal JavaScript code, with all built-in language constructs. 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 by a Raspberry Pi camera attached to each microscope lens. This distinction in function implementation

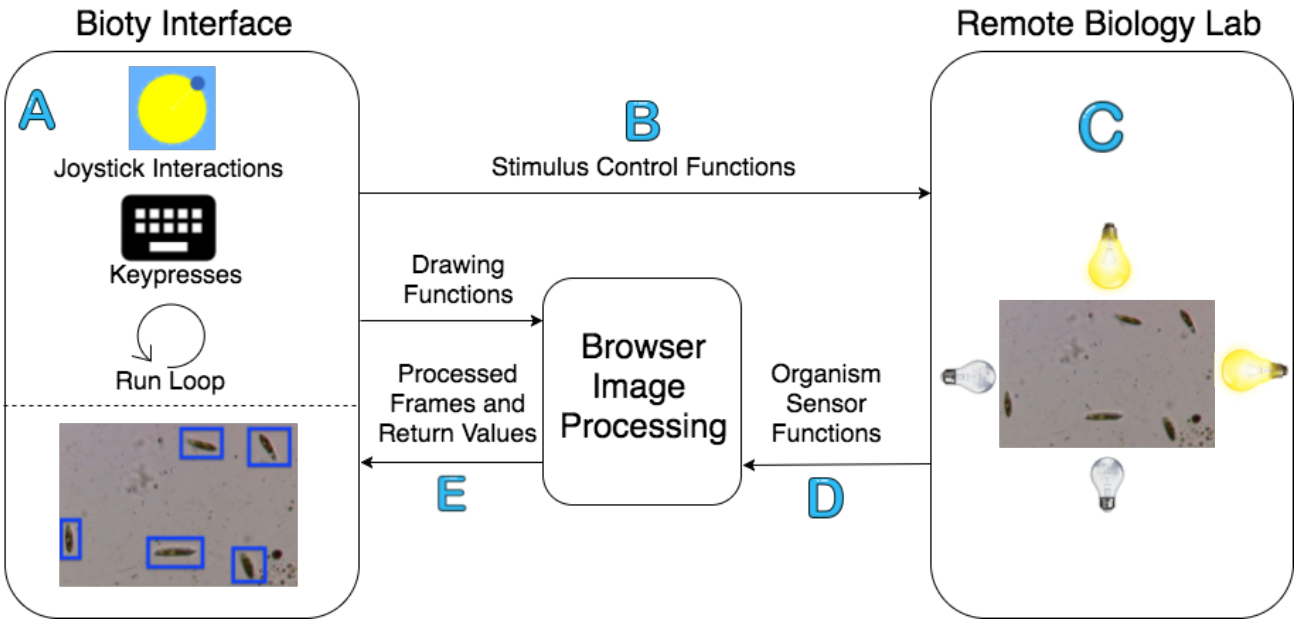


**Figure 5. Overall system architecture of Bioty. (A) Several event triggers can cause a block of code to be executed, including a user moving the joystick, pressing a key, or not doing anything for 1 ms (the run loop). (B) Stimulus control API functions communicate with organism stimuli in the remote labs, such as LEDs. (C) Microscopes in the remote biology cloud lab have positional LEDs placed around the microfluidic chamber that can influence the movement of the organisms. (D) Organism sensor functions capture data from the incoming frames through image processing. (E) Processed frames augmented with virtual images are returned to the interface. Return values from the organism sensor functions are returned as a result of image processing. Yellow arrows indicate to the user the direction in which the light is shining.**

is not seen by the user.

For the API calls that communicate with the frames returned from the microscope's live feed, the Chrome browser's Portable Native Client (PNaCl) toolchain is used to execute native C++ code directly in the browser. All *Euglena* tracking is implemented on the C++ side of the system using OpenCV2. OpenCV2 provides the basic image processing functionality for the implementation of the *getEuglenaPositionByID*, *getEuglenaVelocityByID*, *getEuglenaAccelerationByID*, and *getEuglenaRotationByID* API functions. Multiple-object tracking is performed with the Kalman filtering algorithm for motion prediction [34] in conjunction with the Hungarian algorithm [22] to continuously match detected object contours to tracked objects. OpenCV2 is also used to draw virtual shapes on each frame (such as by the *drawText* and *drawRect* functions, which are wrappers over existing OpenCV2 functions).

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.

## SAMPLE APPLICATIONS

In this section, we describe three applications that we created in order to demonstrate the power of Bioty.

### Velocity Plot

The first example program calculates the average velocity of all detected organisms and overlays a plot of average velocity over time on the screen (Fig. 6A). The plot is updated in real-time. The entire program consists of 50 lines of code.

### Velocity Visualization

Another program provides a visualization of *Euglena* velocities, with boxes drawn around the organisms color-coded based on whether the velocity is slow, medium, or fast (Fig. 6B). The entire program consists of 71 lines of code.

### Euglena Frogger Game

The third program is a "biotic game" where the user controls a virtual green circle with the arrow keys on the keyboard (Fig. 6C). The goal is to get the green circle from the bottom to the top of the screen without hitting a *Euglena*. If the circle intersects with one of the *Euglena*, the user must start over. The entire program consists of 42 lines of code.

## ON-SITE STUDY WITH HBI NOVICES

To evaluate Bioty, we started with an on-site study with programmers performing two structured and two free-form programming tasks. As Bioty is the first of its kind, there was no prior system that could have served as a control in our user study. Participants were limited to 2 hours of total coding time, including familiarizing themselves with the interface and API.

### Methodology

Seven programmers aged 21-24 years (mean=22.57 years, SD=1.13 years) 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

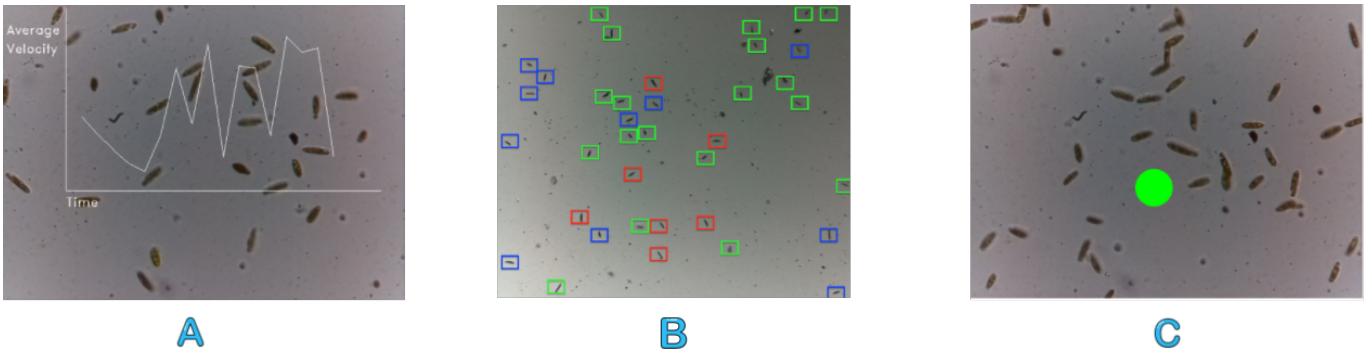


**Figure 6. Several biological applications can be created with 100 lines of code or less with Bioty. (A) A plot of the average *Euglena* speed with respect to time can be virtually plotted onto the screen in real time. (B) Properties of the organism, in this case the velocity of the *Euglena*, can be mapped to properties such as color in order to generate biological art or to visualize data. (C) The Euglena Frogger Game is a "biotic video game" where the user moves a green circle with the arrow keys on the keyboard. The goal is to move the green circle from the bottom to the top of the screen. If the virtual green circle collides with a real *Euglena* cell, then the player must start over. If the player moves the green circle to the top of the screen without colliding with a microorganism, the player wins.**

were undergraduate students at the 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 5-point Likert scale from 1 ("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).

In order to familiarize the participants with the system before programming on their own, they were asked to perform two programming tasks. The programming tasks were as follows, verbatim and in order:

- **Task 1 (T1):** The *avoidEuglenaGame* program codes a "biotic game" where the player uses the W,A,S,D keys to move a virtual green circle from the bottom to the top of the screen. If the virtual circle comes in contact with a real *Euglena* organism on the screen, the player must start again from the bottom of the screen. Modify the *avoidEuglenaGame* program so that the user scores 1 point every time the circle reaches the top of the microscope screen, and display this score on the screen. Save the game as *avoidEuglenaGameModified*. Finally, modify the code so that the green circle increases in size every time the user levels up. This also means that collisions between the circle and *Euglena* are more likely.

- **Task 2 (T2):** The *euglenaCountExperiment* program displays the total count of the detected *Euglena* on the screen and saves this count to a file on the cloud every 10 seconds. After the program is over, the contents of the file are displayed above the microscope screen. Modify *euglenaCountExperiment* so that you get the *Euglena* density in each of the 4 quadrants of the screen. For the purposes of this study, *Euglena* density in a quadrant = (number of *Euglena* in quadrant) / (total number of detected *Euglena*). Save this information in the text file in any format you want, and display the file contents onto the screen when the user clicks the "Stop Program" button.

Before starting, 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. The programming tasks were as follows, verbatim and in order:

- **Task 3 (T3):** Program a new scientific application of your choosing. Your program should teach you something you didn't know before about *Euglena*.

- **Task 4 (T4):** Program any biological application of your choosing. This can be a new scientific experiment, a new biotic game, or some other novel and interesting application that goes beyond the example programs.

As participants provided feedback, we iteratively made improvements to both the Bioty interface and the API. Most of these were non-biology UI improvements, such as making the event code boxes collapsible/expandable and shortening the API function names to be more concise. Usually about four days passed between the individual participants testing Bioty, which provided us with adequate time to make UI changes in between participants based on the feedback received from each participant.

Logs were recorded of all user button clicks on the interface, intermediate versions of their code, any example programs they opened, time spent programming per session, and task completion times.

After completing tasks T1 through T4, participants were asked

| *Task* | *Avg. Completion Time ± STDEV (minutes)* |
|---|---|
| T1 | 29.02 ± 14.19 |
| T2 | 38.15 ± 21.39 |
| T3 and T4 combined | 50.32 ± 21.03 |

**Table 1. Mean completion times for the two preliminary tasks (T1 and T2) and the free-form task (T3 and T4 combined) during the on-site study; N=7 participants.**

to answer a series of questions in an online form about what they liked and disliked about the Bioty interface and the JavaScript API as well as a description of the applications they created for T3 and T4. Participants were compensated 15 US dollars for their participation.

## Findings

All 7 participants were able to complete all tasks (T1 through T4). No single programming task took more than 1.5 hours to complete for any participant. Task completion time averages appear in Table 1.

### *Bioty API*

When asked what they liked and disliked about the API, all participants mentioned the ease of use. P4 stated that "*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 towards actually using the program*".

When asked what could be improved about the API, participants mentioned that they would like additional organism sensor functions. P5 wrote that "*the API could be a little more robust in the information that it provides about each of the Euglena. Size and age and other info tracked by ID could allow for some really cool experiments*".

Other participants desired higher-order API functions. For example, P5 wanted a function to "*guess the closest/average distance between two Euglena*". It should be noted that while such functions could be constructed as helper functions calling several of the core API functions we provided, there are several helper functions like this that participants desired.

### *Bioty Interface*

When asked what they liked about the Bioty interface, participants stated that it was "*very accessible*" (P2) and that "*it was easy to understand, and it was aesthetic*" (P4). Participants also approved of the constrained event-driven programming environment. P6 mentioned that he "*liked how there were text boxes for each function's implementation! Made it more approachable than a giant text box*".

When asked what aspects of the interface could be improved, participants provided general UI advice which we took into account as we iterated the system for the next participant.

### *Learning About Biology*

Several participants in the on-site study had little to no experience with biology. Through interacting with Bioty, participants without knowledge about *Euglena* commented that they learned some fundamentals of *Euglena* behavior through the development process. While all participants learned that

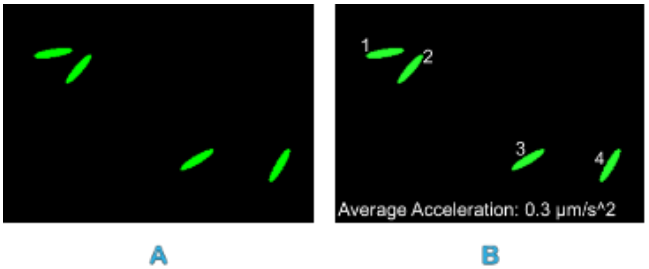


**Figure 7. A simulation-based sandbox mode replaces the live microscope feed with an interactive virtual simulation of the *Euglena*. (A) Particle-based simulation of *Euglena* movement, with *Euglena* moving according to a pre-defined set of rules attempting to mimic the movement of real organisms. (B) The organism sensor and stimulus control functions all work with the simulation.**

the organisms move in response to light, some participants observed subtleties in this behavior. P7 noted, "*from watching the Euglena more and interacting with them on this platform, I realized their individual behaviors are really variable; some of them barely respond to light, and some of them respond really quickly.*"

### *Alleviating Physical-resource Constraints*

The first 2 participants responded negatively to the time pressure introduced by the limited physical biological resources (Fig. 2E). P1 explained, "*I have to admit, having the timer count down while I'm writing my code is pretty stressful*". P2 noted that there were "*moments where the page seemed to crash and I couldn't reload the page without having to check availability on another microscope*".

To address this concern, we created an interactive virtual simulation (Fig. 7). The virtual *Euglena* in the simulation responded to the *setLED* stimulus control function in roughly the same manner as real *Euglena* using basic biophysical modeling principles. There was no clear trend in how much time participants spent in the sandbox mode, with an average of 47.57 (SD=20.54) percent of session time spent in sandbox mode per participant.

## Notable Applications

### *Swarm Movement Statistics*

One participant from the on-site study (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. This program was written on-site in our lab, under time pressure, in 34 minutes and contains 54 lines of code.

### *Moving Box Game*

Another participant from the on-site study (programming=4, JavaScript=3, biology=1) created a video game (Fig. 8A) where the player must get as many *Euglena* into a moving virtual green box while the joystick-controlled LEDs shine in the direction of the moving box, making the *Euglena* move away from the target goal. This program was written on-site,

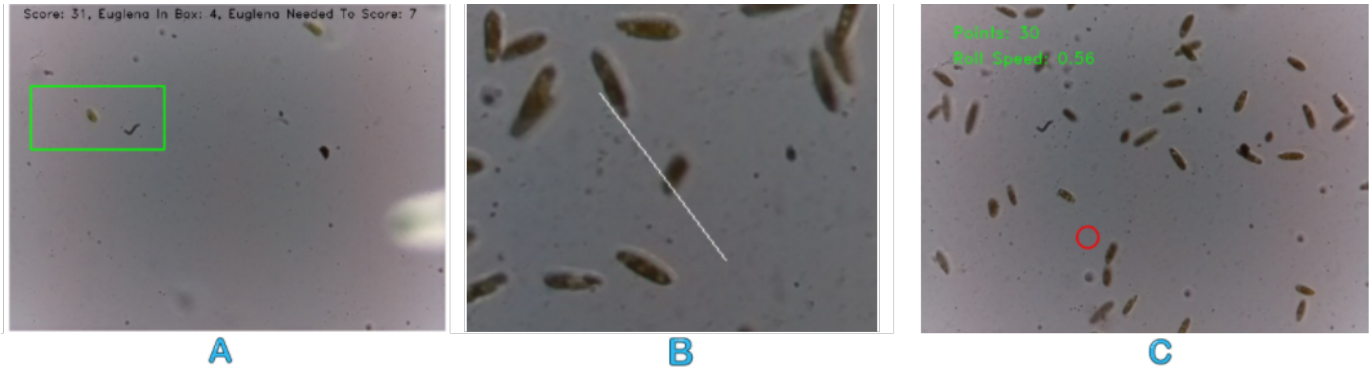


**Figure 8. Participants created a variety of interactive biological applications during the user studies. (A) One participant from the on-site evaluation created a video game where the player must get as many *Euglena* into a moving virtual green box while the user-directed LEDs shine in the direction of the moving box, making the *Euglena* move away from the target goal. (B) A participant from the remote evaluation created an animated visualization of average orientation (rotation) of all detected *Euglena* on the screen, depicted by a white line. (C) Another remote participant created 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.**

under time pressure, in 83 minutes and contains 56 lines of code.

## REMOTE STUDY WITH HBI EXPERTS

Once we had refined the system according to preliminary feedback and added the various sandbox modes, in order to gather information about more casual Bioty usage, we decided to provide participants with the opportunity to develop applications remotely over an extended period of time. Further, we tested with people who had previous experience designing or developing interactive biotechnology applications, as we wanted to compare their prior experiences with their experience with Bioty .

### Methodology

Five HBI experts (i.e., people who have previously developed *Euglena*-based HBI applications) aged 26-33 years (mean=31.2 years, SD=3.0 years) were recruited for the remote portion of the study. Two of the participants identified as female. Three participants were graduate students at the 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 remote study was to conduct tasks T1 and T2 from the on-site study and then to program a free-form application of their choosing (T4). The participants programmed on their own time and had 2 weeks to complete the tasks. All logging was the same as in the lab study, including filling out the same questionnaire at the end of the study.

### Results

Table 2 shows the completion time, lines of code, and API calls broken up by category for each free-form program created in the remote study. Participants spent as little as 33 minutes and no more than 7 hours in active session time developing their applications. Participants spent more time using the simulation mode than the live microscope on the cloud lab, with 77.6 (SD=11.4) percent of aggregate session time spent in live mode. All participants made at least one call to an API function of each type (organism, sensor, and application), as seen in Table 2.

When asked whether developing their application on Bioty was easier or harder than it would have been without it, all participants stated their applications were much easier to develop. P8 also noted the importance of including example programs, stating that "*seeing some examples just gets rid of all the zeroth level things one would need to know to start programming, which might otherwise take a whole day to learn*".

When asked which aspects of developing with Bioty were worse than developing the same application without Bioty, participants universally mentioned the lack of extensive debugging tools on Bioty. Both P8 and P11 noted that the range of applications one can create with the Bioty API are limited, as there are applications that the API does not currently support. However, P8 followed up that this constrained API set "*might give you more creativity.*"

### Notable Applications

#### *Orientation Visualization*

One of the programmers from the remote evaluation (programming=3, JavaScript=4, biology=3) created a visualization with a line that rotates in the direction of the average orientation of all detected *Euglena* (Fig. 8B). The program also records the average orientation every 10 seconds and saves the results to a file. This program was written in 216 minutes and contains 123 lines of code (Table 2). This was the longest program written in our user studies, both on-site and remote.

#### *Basketball Game*

Another programmer from the remote evaluation (programming=3, JavaScript=3, biology=5) created a two-player game (Fig. 8C) 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

| *Application Name* | *Time to Code (mins.)* | *Lines of Code* | *Organism API Calls* | *Sensor API Calls* | *App. API Calls* |
|---|---|---|---|---|---|
| LED Quadrant Count Experiment | 33 | 42 | 3 | 2 | 9 |
| Rotation Experiment | 216 | 123 | 14 | 13 | 3 |
| Average Speed Experiment | 70 | 58 | 8 | 5 | 13 |
| Basketball Game | 52 | 104 | 16 | 1 | 5 |
| Orientation Visualizations | 350 | 109 | 10 | 4 | 15 |

**Table 2. Coding time and length of the free-form applications participants built for the remote study. Also included is the number of Bioty API calls made in each application, split up by function type as defined in Fig. 3.**

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. This program was written in 52 minutes, and contains 104 lines of code (Table 2).

## DISCUSSION

Bioty achieved our goal of lowering access barriers to prototyping and developing HBI applications. Our study participants (on-site and remote) came from various backgrounds, and they were all able to complete the two familiarization tasks and to prototype an application of their choosing with no guidance from researchers. All applications were created quickly and with only a few lines of code ( Table 2). Expert HBI designers and developers confirmed that Bioty significantly speeds up development time and makes development easier than previous approaches. Novice HBI users also indicated that Bioty had a low barrier to entry.

The participants provided several suggestions, some of which we implemented during the on-site study. In particular, we found that virtual sandbox modes solved potential physical-resource constraints that arise with prolonged access to the cloud lab. The time spent across all participants in the sandbox mode and live mode were roughly equivalent. The overall design behind Bioty , including sandbox mode, may be suitable for other swarm-programming applications beyond biology, e.g., [31, 41].

Bioty enabled study participants to build a variety of applications, ranging from data visualization, art, automated scientific experiments, interactive scientific experiments, and biotic video games. While all of these applications are far from polished and significantly more development time could be spent on any of them, the potential is clear.

Out of the five participants without relevant biology experience in the on-site evaluation, all self-reported learning about the *Euglena*'s response to light, and one user inadvertently learned about the biological variability of *Euglena* movement through the process of prototyping on Bioty. This suggests that Bioty could be deployed as educational platform that supports a hybrid approach where students learn about programming and biology simultaneously through developing creative applications.

Two novice users had asked without being prompted about the ethical issues of programming the movement of living cells. We pointed out that these issues were discussed previously, [17], i.e., *Euglena* are non-sentient, hence the present usage does not present a particular issue.

## LIMITATIONS AND FUTURE WORK

The next step is to develop more applications in Bioty and then to investigate the utility for the end user. For example, middle- and high-school students could be enabled to run more complex biology experiments on the platform or play biotic games in order to learn about biology.

The underlying cloud hardware for Bioty could be extended. The current setup only has of 4 LEDs for stimulation along the cardinal axes; more complex programmatic swarm control could be achieved with a projector-based system, as demonstrated previously [30]. This architecture would allow for more stimulus control functions in the API. Additional hardware sensors could also be added, such as temperature and chemical sensors, which would allow for more sensor functions.

Multiple improvements could be made to make Bioty even more user friendly. For example, participants mentioned that Bioty does not have the full debugging capabilities of a traditional development environment, which can inhibit development. Further, programming can be made more accessible to other user groups by replacing JavaScript programming with a simpler, block-based language such as Scratch [38]. This foundation would enable even younger users to quickly design and execute experiments, to program art, and to play games, thereby combining educational aspects of both computer science and life science.

Bioty can be extended for use in other application domains outside biology by replacing the swarm and hardware used on the backend. For example, robotic swarms like Zooids [31] and UbiSwarm [28] could be remotely controlled and applications around these swarms could be developed with Bioty.

## CONCLUSION

We developed Bioty, the first toolkit that allows users to remotely prototype applications that enable humans to interact with living, motile cells in real time. Our qualitative user studies demonstrated that Bioty lowers access barriers to developing HBI applications, and suggested that there can be a variety of types and complexities of applications written with Bioty. Hence, Bioty follows Papert's vision of "low-floor/high-ceiling/wide-walls" [36, 39], and constitutes an important step toward making experimentation, engineering, and interaction with living matter more accessible to a broader community.

## ACKNOWLEDGMENTS

The authors like to thank Z. Hossain, T.R. Stones, all members of the Riedel-Kruse lab and the volunteers who tested earlier versions of Bioty. Financial support was provided by NSF Cyberlearning Grant 1324753 and a James H. Clark Foundation Bioengineering Fellowship. Author contributions: I.H.R-K conceived the projects, all authors contributed ideas; P.W. implemented the software and performed experiments, user studies and analysis; K.G.S-G and S.G. aided implementation and design exploration; P.W. and I.H.R-K wrote the paper.

## REFERENCES

1. William Buxton and Richard Sniderman. 1980. Iteration in the design of the human-computer interface. In *Proceedings of the 13th Annual Meeting of the Human Factors Association of Canada*, Vol. 7281. 37.
2. D Ewen Cameron, Caleb J Bashor, and James J Collins. 2014. A brief history of synthetic biology. *Nature Reviews Microbiology* 12, 5 (2014), 381–390.
3. Allen Y. Chen, Chao Zhong, and Timothy K. Lu. 2015. Engineering Living Functional Materials. *ACS Synth Biol* 4, 1 (2015), 8–11. DOI: http://dx.doi.org/10.1021/sb500113b
4. George M Church, Michael B Elowitz, Christina D Smolke, Christopher A Voigt, and Ron Weiss. 2014. Realizing the potential of synthetic biology. *Nature Reviews Molecular Cell Biology* 15, 4 (2014), 289–294.
5. Nate J Cira, Alice M Chung, Aleksandra K Denisin, Stefano Rensi, Gabriel N Sanchez, Stephen R Quake, and Ingmar H Riedel-Kruse. 2015. A biotic game design project for integrated life science and engineering education. *PLoS biology* 13, 3 (2015), e1002110.
6. Fernando J Corbató, Marjorie Merwin-Daggett, and Robert C Daley. 1962. An experimental time-sharing system. In *Proceedings of the May 1-3, 1962, spring joint computer conference*. ACM, 335–344.
7. C Creutz and BODO DIEHN. 1976. Motor responses to polarized light and gravity sensing in Euglena gracilis. *Journal of Eukaryotic Microbiology* 23, 4 (1976), 552–556.
8. B Diehn. 1969. Phototactic response of Euglena to single and repetitive pulses of actinic light: orientation time and mechanism. *Experimental cell research* 56, 2-3 (1969), 375–381.
9. Christopher B Eiben, Justin B Siegel, Jacob B Bale, Seth Cooper, Firas Khatib, Betty W Shen, Barry L Stoddard, Zoran Popovic, and David Baker. 2012. Increased Diels-Alderase activity through backbone remodeling guided by Foldit players. *Nature biotechnology* 30, 2 (2012), 190–192.
10. Unity-Game Engine. 2015. Unity Technologies. *Available: Unity Technologies* (2015).
11. Opher Etzion, Peter Niblett, and David C Luckham. 2011. *Event processing in action*. Manning Greenwich.
12. Piyum Fernando, Matthew Pandelakis, and Stacey Kuznetsov. 2016. Practicing DIYBiology In An HCI Setting. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2064–2071.
13. Armando Fox. 2011. Cloud computingâĂŤwhat's in it for me as a scientist? *science* 331, 6016 (2011), 406–407.
14. Lukas C Gerber, Honesty Kim, and Ingmar H Riedel-Kruse. 2016. Interactive Biotechnology: Design Rules for Integrating Biological Matter into Digital Games.. In *DiGRA/FDG*.
15. Ken Goldberg, Joseph Santarromana, George Bekey, Steven Gentner, Rosemary Morris, Jeff Wiegley, and Erich Berger. 1995. The telegarden. In *Proc. of ACM SIGGRAPH*. 135–1140.
16. D-P Häder. 1987. Polarotaxis, gravitaxis and vertical phototaxis in the green flagellate, Euglena gracilis. *Archives of microbiology* 147, 2 (1987), 179–183.
17. Hayden Harvey, Molly Havard, David Magnus, Mildred K Cho, and Ingmar H Riedel-Kruse. 2014. Innocent Fun or âĂIJMicroslaveryâĂİ? *Hastings Center Report* 44, 6 (2014), 38–46.
18. Erika Check Hayden. 2014. The automated lab. *Nature* 516, 7529 (2014), 131.
19. Zahid Hossain, Engin Bumbacher, Alison Brauneis, Monica Diaz, Andy Saltarelli, Paulo Blikstein, and Ingmar H. Riedel-Kruse. 2017. Design Guidelines and Empirical Case Study for Scaling Authentic Inquiry-based Science Learning via Open Online Courses and Interactive Biology Cloud Labs. *International Journal of Artificial Intelligence in Education* (31 Jul 2017).
20. Zahid Hossain, Engin W Bumbacher, Alice M Chung, Honesty Kim, Casey Litton, Ashley D Walter, Sachin N Pradhan, Kemi Jona, Paulo Blikstein, and Ingmar H Riedel-Kruse. 2016. Interactive and scalable biology cloud experimentation for scientific inquiry and education. *Nature biotechnology* 34, 12 (2016), 1293–1298.
21. Zahid Hossain, Xiaofan Jin, Engin W Bumbacher, Alice M Chung, Stephen Koo, Jordan D Shapiro, Cynthia Y Truong, Sean Choi, Nathan D Orloff, Paulo Blikstein, and others. 2015. Interactive cloud experimentation for biology: An online education case study. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 3681–3690.
22. Chang Huang, Bo Wu, and Ramakant Nevatia. 2008. Robust object tracking by hierarchical association of detection responses. In *European Conference on Computer Vision*. Springer, 788–801.
23. Paul Hudak. 1996. Building domain-specific embedded languages. *ACM Computing Surveys (CSUR)* 28, 4es (1996), 196.

24. Paul Hudak. 1998. Modular domain specific languages and tools. In *Software Reuse, 1998. Proceedings. Fifth International Conference on*. IEEE, 134–142.

25. Hiroshi Ishii, Dávid Lakatos, Leonardo Bonanni, and Jean-Baptiste Labrune. 2012. Radical atoms: beyond tangible bits, toward transformable materials. *interactions* 19, 1 (2012), 38–51.

26. Honesty Kim, Lukas Cyrill Gerber, Daniel Chiu, Seung Ah Lee, Nate J Cira, Sherwin Yuyang Xia, and Ingmar H Riedel-Kruse. 2016b. LudusScope: Accessible Interactive Smartphone Microscopy for Life-Science Education. *PloS one* 11, 10 (2016), e0162602.

27. Honesty Kim, Lukas C Gerber, and Ingmar H Riedel-Kruse. 2016a. Interactive Biotechnology: Building your own Biotic Game Setup to Play with Living Microorganisms. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 1000–1002.

28. Lawrence H Kim and Sean Follmer. 2017. UbiSwarm: Ubiquitous Robotic Interfaces and Investigation of Abstract Motion as a Display. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 66.

29. Stacey Kuznetsov, Carrie Doonan, Nathan Wilson, Swarna Mohan, Scott E Hudson, and Eric Paulos. 2015. DIYbio things: open source biology tools as platforms for hybrid knowledge production and scientific participation. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 4065–4068.

30. Amy T Lam, Karina G Samuel-Gama, Jonathan Griffin, Matthew Loeun, Lukas C Gerber, Zahid Hossain, Nate J Cira, Seung Ah Lee, and Ingmar H Riedel-Kruse. 2017. Device and programming abstractions for spatiotemporal control of active micro-particle swarms. *Lab on a Chip* 17, 8 (2017), 1442–1451.

31. Mathieu Le Goc, Lawrence H Kim, Ali Parsaei, Jean-Daniel Fekete, Pierre Dragicevic, and Sean Follmer. 2016. Zooids: Building blocks for swarm user interfaces. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM, 97–109.

32. Jeehyung Lee, Wipapat Kladwang, Minjae Lee, Daniel Cantu, Martin Azizyan, Hanjoo Kim, Alex Limpaecher, Snehal Gaikwad, Sungroh Yoon, Adrien Treuille, and others. 2014. RNA design rules from a massive open laboratory. *Proceedings of the National Academy of Sciences* 111, 6 (2014), 2122–2127.

33. Seung Ah Lee, Engin Bumbacher, Alice M Chung, Nate Cira, Byron Walker, Ji Young Park, Barry Starr, Paulo Blikstein, and Ingmar H Riedel-Kruse. 2015. Trap it!: A Playful Human-Biology Interaction for a Museum Installation. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2593–2602.

34. Xin Li, Kejun Wang, Wei Wang, and Yang Li. 2010. A multiple object tracking method using Kalman filter. In *Information and Automation (ICIA), 2010 IEEE International Conference on*. IEEE, 1862–1866.

35. Mark Overmars. 2004. Teaching computer science through game design. *Computer* 37, 4 (2004), 81–83.

36. Seymour Papert. 1980. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

37. Mitchel Resnick. 1996. StarLogo: An environment for decentralized modeling and decentralized thinking. In *Conference companion on Human factors in computing systems*. ACM, 11–12.

38. Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and others. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

39. Mitchel Resnick and Brian Silverman. 2005. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 conference on Interaction design and children*. ACM, 117–122.

40. Ingmar H Riedel-Kruse, Alice M Chung, Burak Dura, Andrea L Hamilton, and Byung C Lee. 2011. Design, engineering and utility of biotic games. *Lab on a Chip* 11, 1 (2011), 14–22.

41. Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. 2014. Programmable self-assembly in a thousand-robot swarm. *Science* 345, 6198 (2014), 795–799.

42. Samuel K Sia and Matthew P Owens. 2015. Share and share alike. *Nature biotechnology* 33, 12 (2015), 1224.

43. Ivan E Sutherland. 1965. The ultimate display. *Multimedia: From Wagner to virtual reality* (1965).

44. Alex S Taylor, Nir Piterman, Samin Ishtiaq, Jasmin Fisher, Byron Cook, Caitlin Cockerton, Sam Bourton, and David Benque. 2013. At the interface of biology and computation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 493–502.

45. Brian P Teague, Patrick Guye, and Ron Weiss. 2016. Synthetic morphogenesis. *Cold Spring Harbor perspectives in biology* 8, 9 (2016), a023929.

46. Uri Wilensky. 1999. NetLogo. (1999).