Learning to code

why we fail, how we flourish

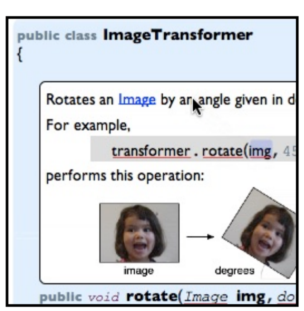
Andrew J. Ko, Ph.D.
Code & Cognition Lab
The Information School
DESIGN USE BUILD
UNIVERSITY of WASHINGTON

Me

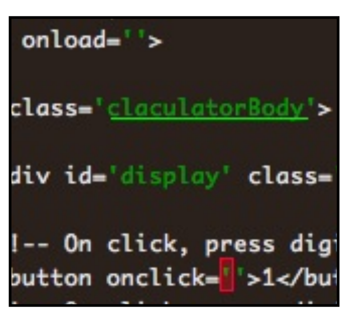
- Professor for the last ~10 years at **UW Seattle**
- Ph.D. from **Carnegie Mellon**'s HCI Institute
- Background in **CS**, **Psychology**, and **Design**

Code is the most powerful, least usable interface we've invented

- Everyone that *wants* to code should be able to
- But there are immense learning barriers
- I spent the first decade trying to lower these barriers by creating more *usable interactive developer tools*

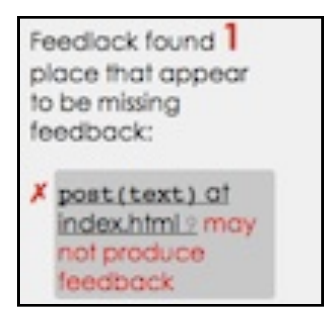


read

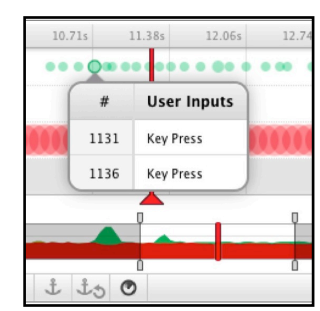


write

test



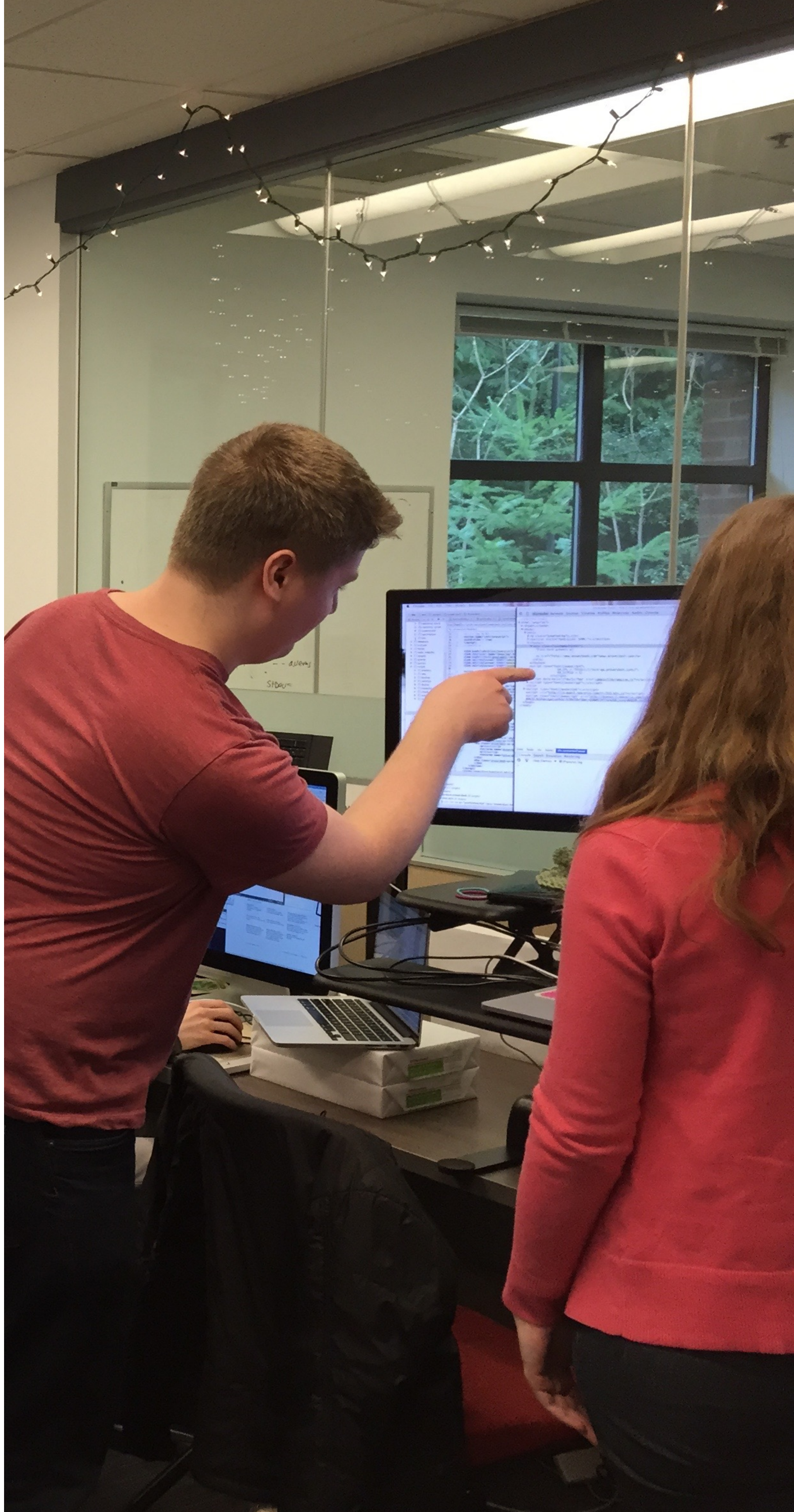
verify



debug

Skills > tools

- I spent 3 years as CTO managing ~8 developers at AnswerDash
- What I saw:
  - Tools only *amplify* skills
  - Skills come from learning
  - Learning comes from teaching
  - I spent most of my time teaching



Millions want to learn to code

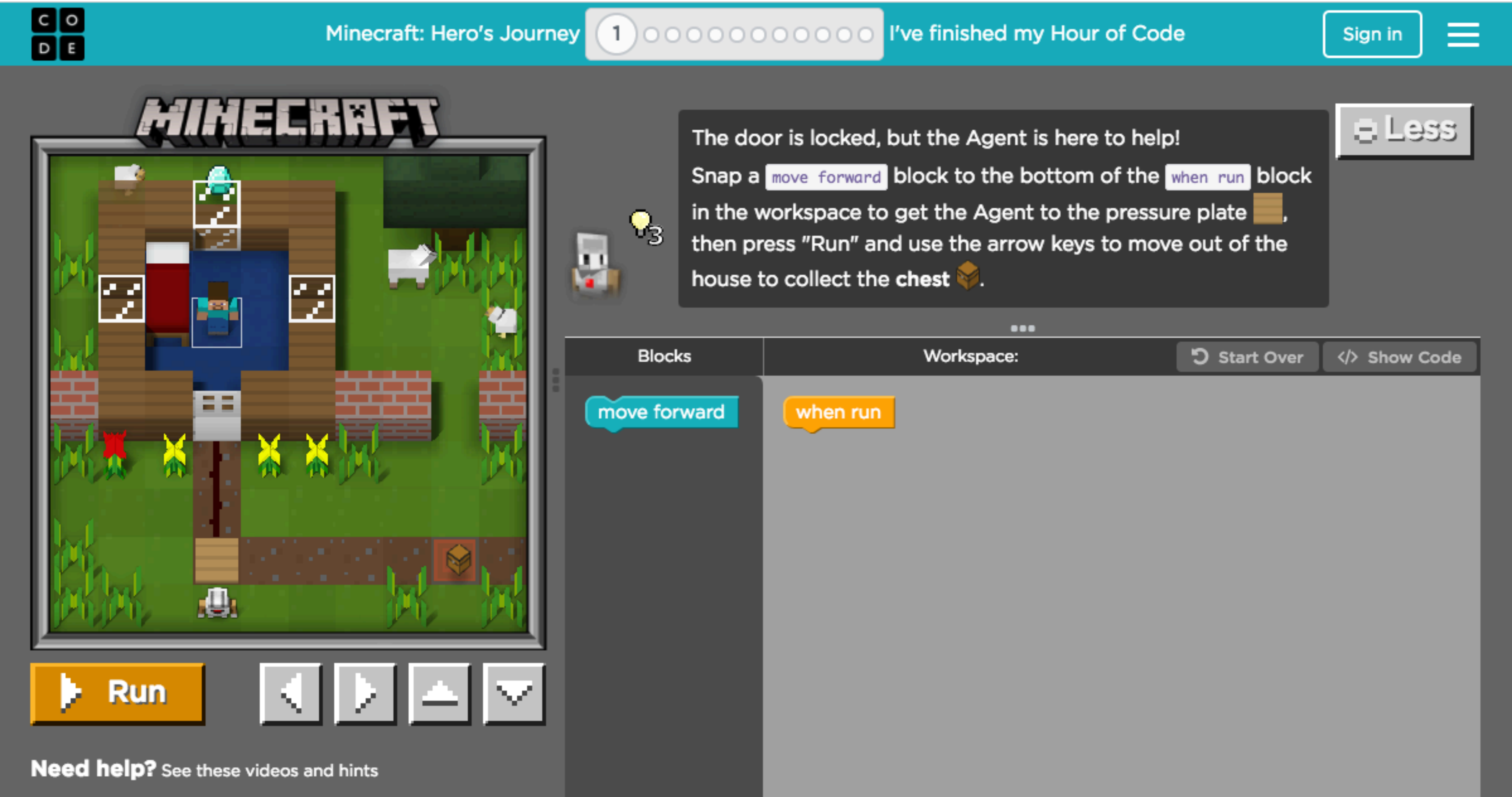
Millions want to learn to code

$1.3 billion

Are people learning?

we have (some) evidence

77% of Code.org's 500 million K-12 learners complete 0-2 puzzles code.org



Record enrollment in AP CS, but most don't take exam, and 60% who do, fail it (especially underrepresented minorities) College Board

After a year of intro courses, most undergrads can't accurately predict the outcome of simple programs, or solve simple programming problems McCracken et al. 2001, Lister et al. 2004 et al. 2013, Seppälä et al. 2015

In 2017, 23,000 adults in 95 U.S. coding bootcamps; 24 report dropout rates of 10-50% CourseReport.com

62% of employers view applicants to entry-level developer positions as lacking basic programming knowledge

Career Advisory Board Survey 2016

So I did some reading.

- Read seminal literature in learning science and education e.g. How people learn: Brain, mind, experience, and school
- Read 30 years of computing education research:
  - *ACM International Computing Education Research Conference (ICER)*
  - *ACM Transactions on Computing Education (TOCE)*
  - *SIGCSE Technical Symposium (SIGCSE)*

Why people fail to learn to code

1. People find computing boring, solitary, unwelcoming
2. People struggle to learn their first programming language
3. People struggle to solve programming problems
4. Teachers struggle to teach these things
5. Teachers blame learners for failure
6. People lose confidence and quit

My goal

1. People find computing boring, solitary, unwelcoming
2. People struggle to learn their first programming language
3. People struggle to solve programming problems

- Why are these hard?
- What are effective, equitable, and scalable ways for people to learn these skills?

This talk

- Why learning to program is hard (3 studies)
- Making programs easier to read (1 theory, 3 ideas)
- Making programs easier to write (1 theory, 1 idea)

Why is learning to program difficult?

Study 1 – 70 high school teens

High school

Ko, A.J. and Davis, K. (2017). Computing Mentorship in a Software Boomtown: Relationships to Adolescent Interest and Beliefs. ACM ICER.

Ko, A.J. et al. (2018). Informal Mentoring of Adolescents about Computing: Relationships, Roles, Qualities, and Impact. ACM SIGCSE.

- Many teens lacked feedback or support about their learning from teachers and family:
  - *He do not spent much time with me to be able to understand my problem in the class or unable to help me on it... throughout the AP class I would cried myself to sleep in silent without letting my older brother know my struggle... (M, Asian, 17)*

High school

Ko, A.J. and Davis, K. (2017). Computing Mentorship in a Software Boomtown: Relationships to Adolescent Interest and Beliefs. ACM ICER.

Ko, A.J. et al. (2018). Informal Mentoring of Adolescents about Computing: Relationships, Roles, Qualities, and Impact. ACM SIGCSE.

- Some teens had *informal computing mentors* who provided encouraging instruction and feedback.
  - Associated with stronger interest in learning to code, independent of gender, socioeconomic status.
- Teens sought teachers and mentors who:
  - Would not judge them for their failures
  - Would inspire them to learn
  - Had the expertise to guide them

Study 2 – 26 Bootcamps attendees

Bootcamps

Thayer, K. and Ko, A.J. (2017). Barriers Faced by Coding Bootcamp Students. ACM ICER.

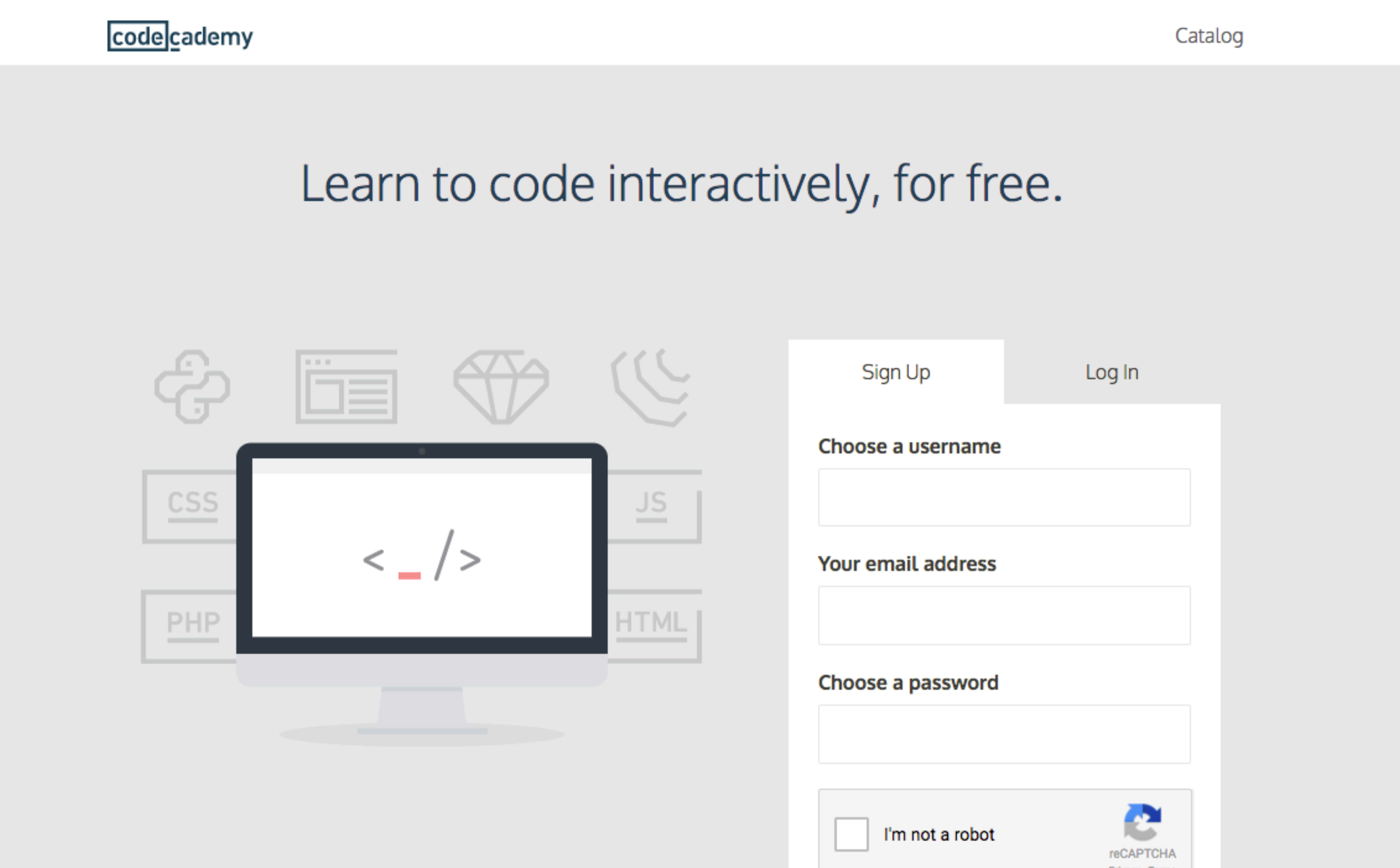
- Some bootcamps were inclusive and encouraging, but many offered no instruction or feedback:
  - *So they're trying to get you into this mentality of you have to read all the documentation. They sit back in the background [to let students read the documentation], and what annoys me is that I've paid a lot of money so that I could have somebody there to teach it to me.*

Bootcamps

Thayer, K. and Ko, A.J. (2017). Barriers Faced by Coding Bootcamp Students. ACM ICER.

- Many bootcamps offered an unwelcoming culture for learners without prior knowledge:
  - *It was divided, the class. Those with experience, I think, they were looking down at [those of us without experience] because maybe there were certain things we were supposed to know and we didn't.*

Study 3 — 30 Coding Tutorials



Tutorials

Kim, A. and Ko, A.J. (2017). A Pedagogical Analysis of Online Coding Tutorials. ACM SIGCSE.

- Four learning science principles
    1. Connect instruction to prior knowledge
    2. Organize declarative knowledge
    3. Offer personalized feedback on practice
    4. Foster self-regulation in problem solving
- Ada completed all 30 tutorials across 100+ hours, judging every lesson against these principles

Tutorials

Kim, A. and Ko, A.J. (2017). A Pedagogical Analysis of Online Coding Tutorials. ACM SIGCSE.

- Most tutorials failed to meet all them:

✗ 1. No connection to prior knowledge

✗ 2. No organization of declarative knowledge about programming languages

✗ 3. No personalized feedback on program correctness or errors

✗ 4. No instruction on how to solve programming problems.

Why is learning to program difficult?

Few of these contexts actually *teach* programming. There are many opportunities to *read and write code*, but learners receive little feedback on whether they are reading or writing *correctly*.

Making programs easier to *read*

One theory, three ideas

Extant theories about why understanding programs is hard

- Wrong programming language
  - Static typing, syntax, and errors matter, but only a little (e.g., Stefik & Siebert 2013)
- Wrong IDE
  - Relative to text, drag and drop “blocks” editors reduce dropout, but don’t improve learning (Cooper et al. 2001)
- Wrong biology
  - No evidence of “geek gene” or bimodal grade distributions (Patitsas et al. 2016)

Extant theories about why understanding programs is hard

- Wrong programming language
  - Static typing, syntax, and editors matter, but only a little (e.g., Stefik & Siebert 2013)
- Wrong IDE
  - Relative to text, drag and drop “blocks” editors reduce dropout, but don’t improve learning (Cooper et al. 2001)
- Wrong biology
  - No evidence of “geek gene” or bimodal grade distributions (Patitsas et al. 2016)

Wrong pedagogy

A new definition of PL knowledge

Knowing a PL means:

1. Being able to reliably and accurately *predict* an arbitrary program's **operational semantics** without the aid of a runtime environment. (Reading a program and knowing what it will do).

2. Knowing **how syntax maps** onto operational semantics.

Note that I'm excluding knowledge of common design patterns, architectures, tools, norms, etc. This strictly concerns the ability to accurately ***read*** *programs.*

An example

Knowing a JavaScript *if-statement* means knowing:

1. Condition is evaluated
2. If it’s true, all of the statements between the first set of braces are executed, and everything between the else braces are skipped.
3. Otherwise, the statements in the first set are skipped, and the statements in the second set are executed.

```
if(dataIsValid && serviceIsOnline) {
  submit();
}
else {
 alert(“Bad error message!”);
}
```

Knowing a entire PL

- Knowing *all* of JavaScript means knowing *all* of the semantics for JavaScript's entire grammar
- That's about **90 non-terminals** in the grammar, each with its own semantic nuances
- Most introductory programming courses never explain any of this:
  - In UW's CS1 course, the 1st homework is to write a Java program with function declarations, function calls, string literals. None of the lectures explain any of this, and, not surprisingly, most students fail.

Four major pedagogies

Learn formal semantics

$$\frac{}{\mathbf{R} \vdash e_{empty} \Rightarrow \mathbf{R}}\ \text{(EMPTY)} \qquad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1 = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}}\ \text{(ASSIGN)}$$

$$\frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1.field* = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}}\ \text{(ASSIGN-FIELD)} \qquad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1[*]* = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}}\ \text{(ASSIGN-ARRAY)}$$

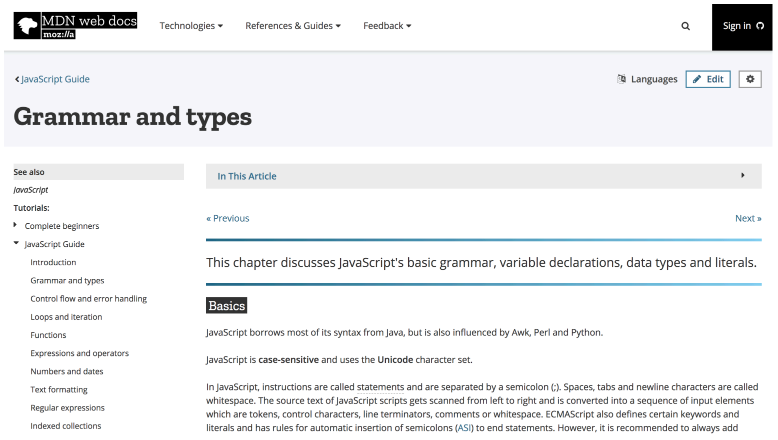
$$\frac{v \in \mathbf{R}\ ;\ \texttt{pointer\_type\_p}(v)\ ;\ \mathbf{R} \vdash e \Rightarrow \mathbf{R}'}{\mathbf{R} \vdash v \oplus e \Rightarrow \mathbf{R}' \cup \{v \oplus e\}}\ \text{(BINOP1)}$$

$$\frac{v \in \mathbf{R}'\ ;\ \texttt{pointer\_type\_p}(v)\ ;\ \mathbf{R} \vdash e \Rightarrow \mathbf{R}'}{\mathbf{R} \vdash e \oplus v \Rightarrow \mathbf{R}' \cup \{e \oplus v\}}\ \text{(BINOP2)} \qquad \frac{\mathbf{R} \vdash e_1 \Rightarrow \mathbf{R}'\ ;\ \mathbf{R}' \vdash e_2 \Rightarrow \mathbf{R}''}{\mathbf{R} \vdash e_1 ; e_2 \Rightarrow \mathbf{R}''}\ \text{(SEQ)}$$

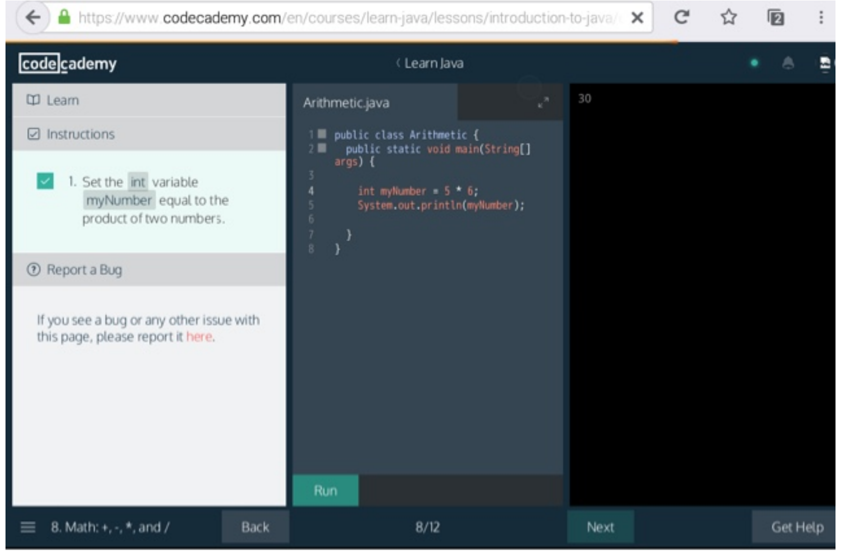
$$\frac{\mathbf{R} \vdash e_1 \Rightarrow \mathbf{R}_1\ ;\ \mathbf{R}_1 \vdash e_2 \Rightarrow \mathbf{R}_2\ ;\ \mathbf{R}_1 \vdash e_3 \Rightarrow \mathbf{R}_3\ ;\ \mathbf{R}_2 \cup \mathbf{R}_3 \vdash e_4 \Rightarrow \mathbf{R}'}{\mathbf{R} \vdash if\ (e_1)\ then\ \{e_2\}\ else\ \{e_3\}\ e_4 \Rightarrow \mathbf{R}'}\ \text{(IF)}$$

$$\frac{\begin{array}{c}\texttt{function\_type\_p}(e) \\ \{e.args[i] \mid 0 < i < e.numargs \wedge \texttt{pointer\_type\_p}(e.args[i])\} \\ \cup\ \{v \mid \texttt{global\_p}(v)\} \vdash e.body \Rightarrow \mathbf{R}\end{array}}{\{\} \vdash e \Rightarrow \mathbf{R}}\ \text{(FUNCTION)}$$

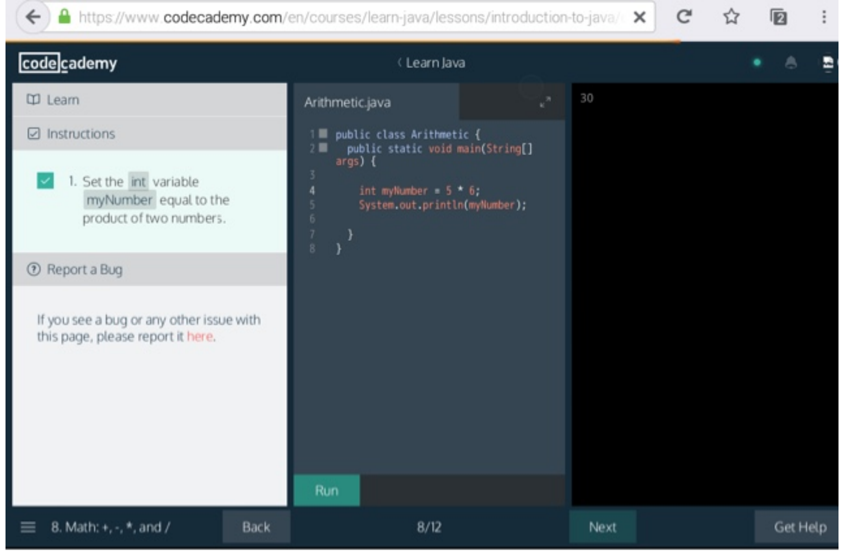
Explain via natural language



Write code

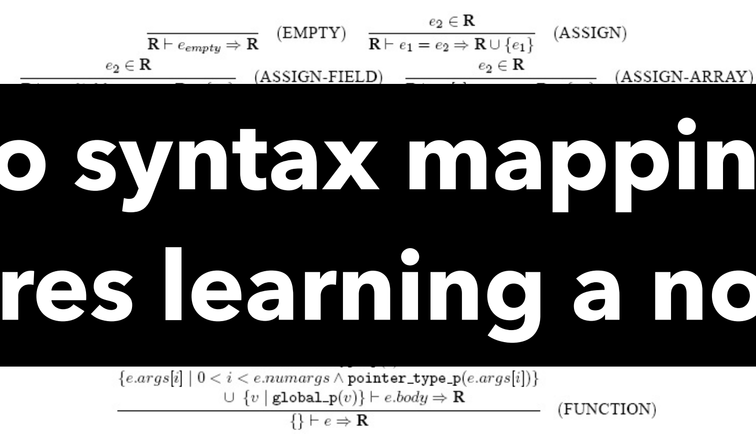


Step through execution



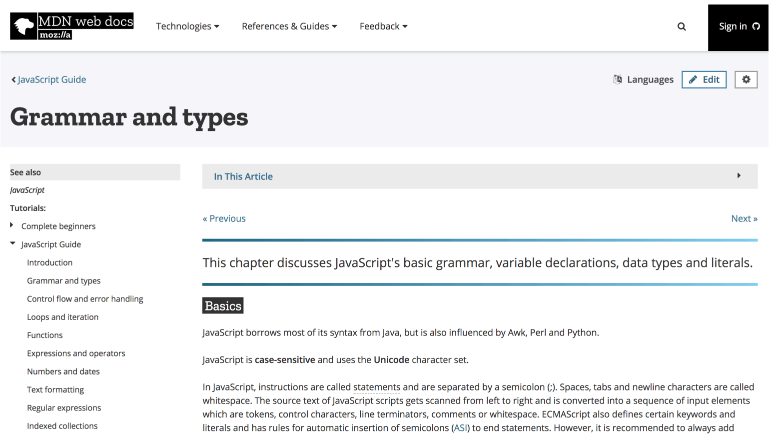
Four major pedagogies

Learn formal semantics

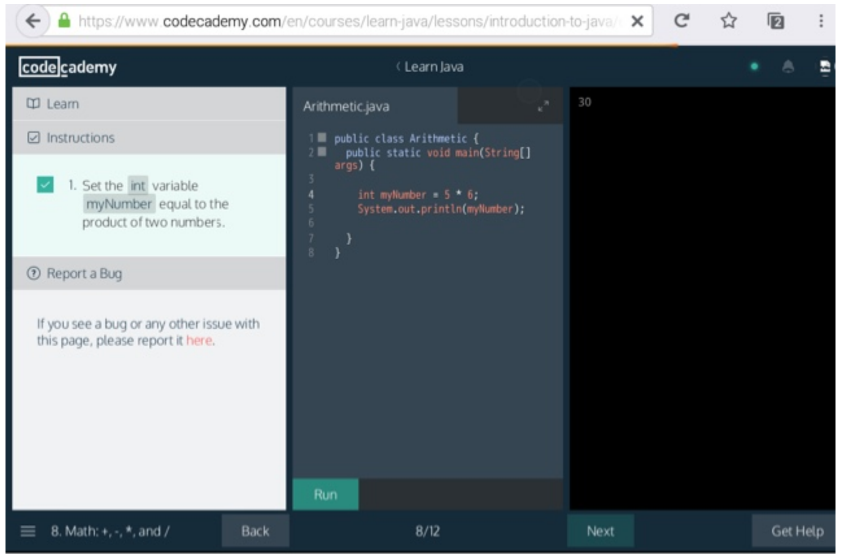


No syntax mapping;
Requires learning a notation

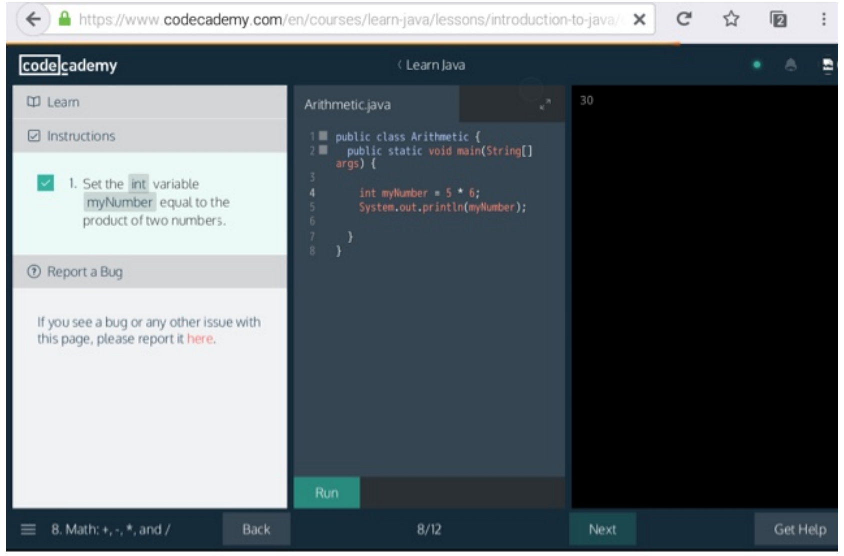
Explain via natural language



Write code

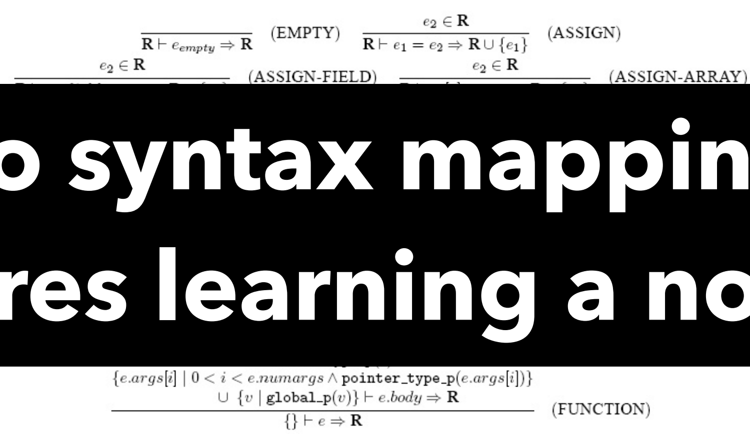


Step through execution



Four major pedagogies

Learn formal semantics

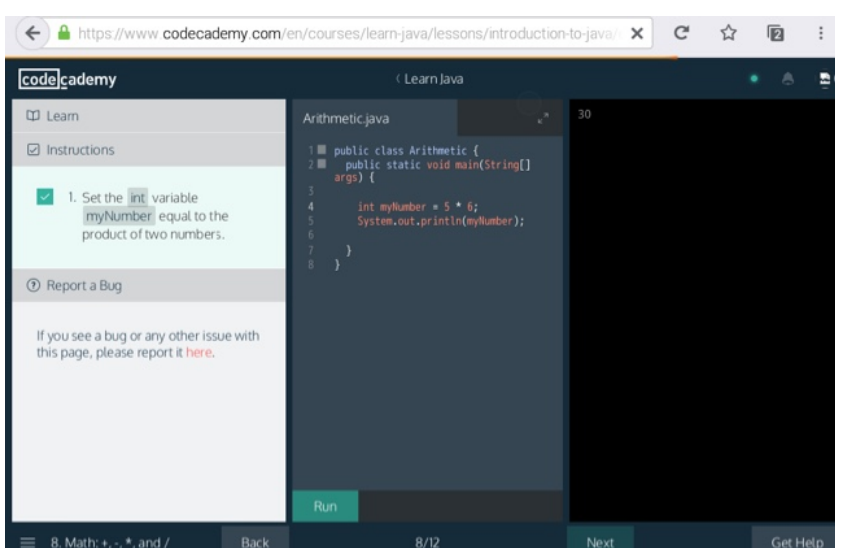


No syntax mapping;
Requires learning a notation

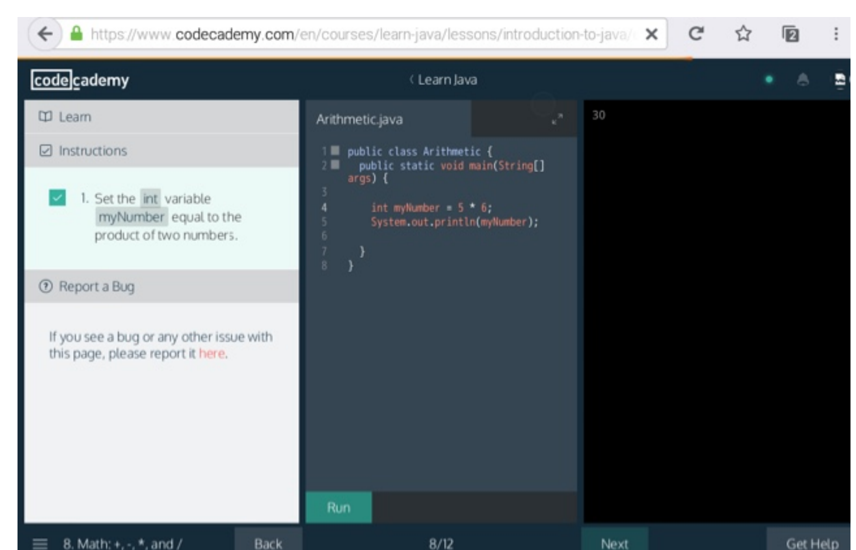
Explain via natural language

Ambiguous, weak syntax mapping

Write code

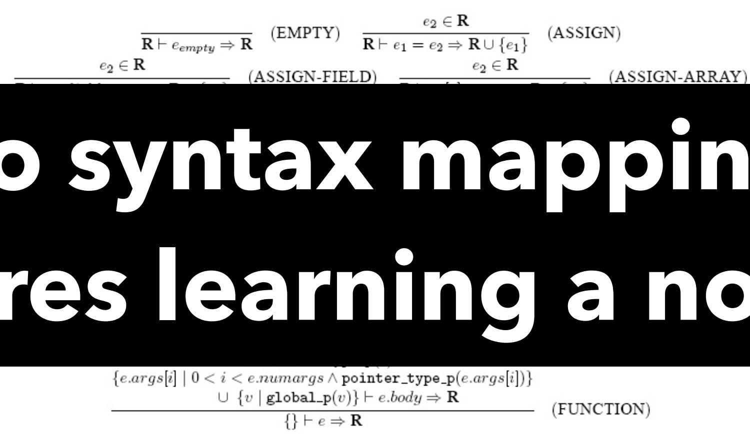


Step through execution



Four major pedagogies

Learn formal semantics

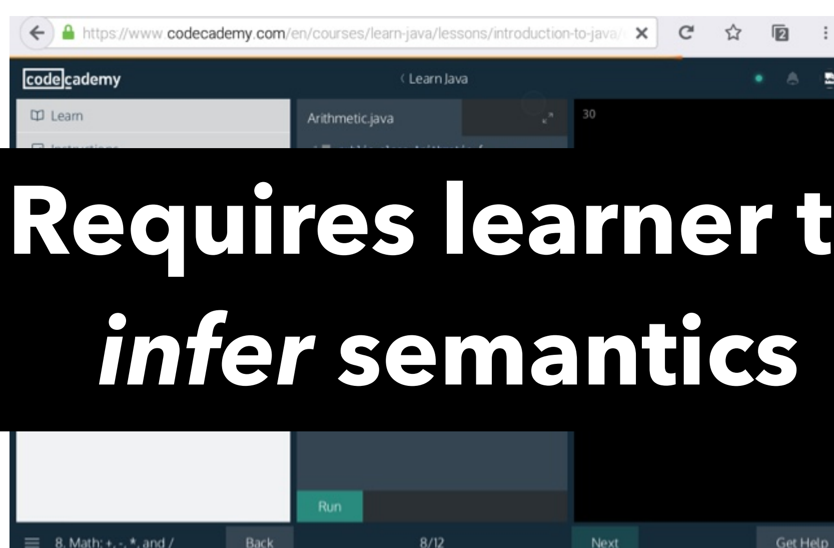


No syntax mapping; Requires learning a notation

Explain via natural language

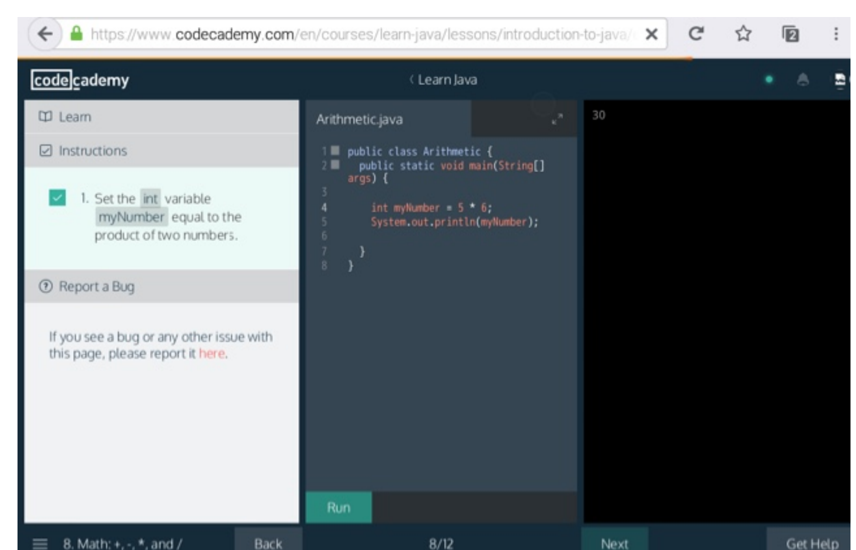
Ambiguous, weak syntax mapping

Write code



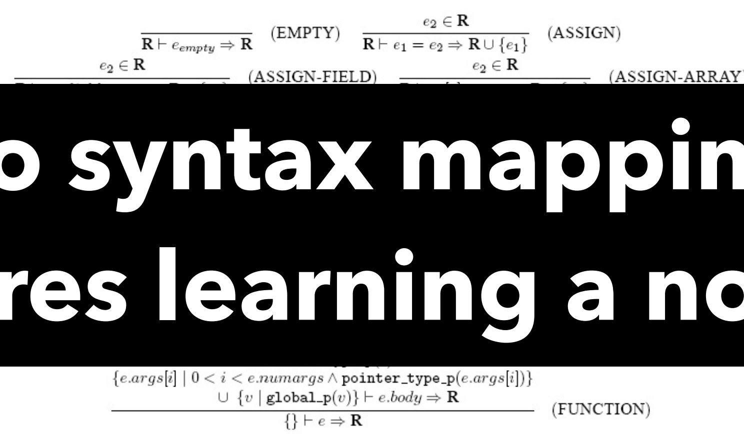
Requires learner to *infer* semantics

Step through execution



Four major pedagogies

Learn formal semantics

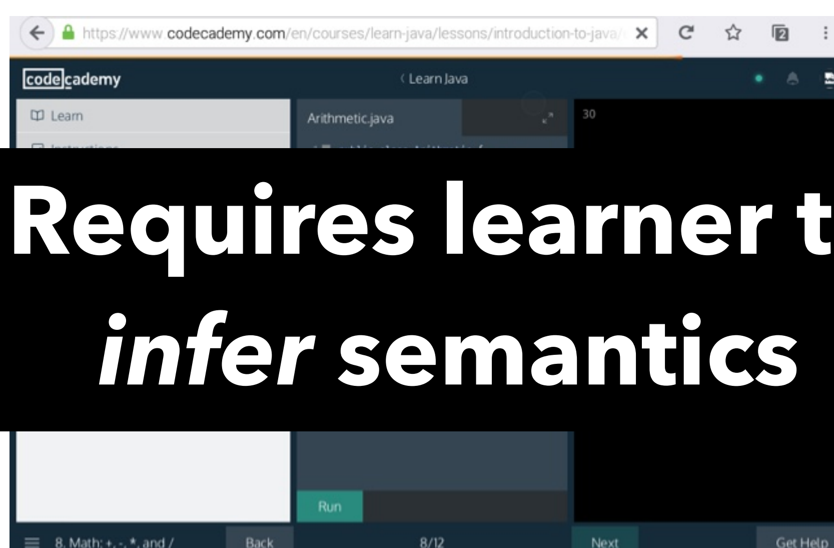


No syntax mapping; Requires learning a notation

Explain via natural language

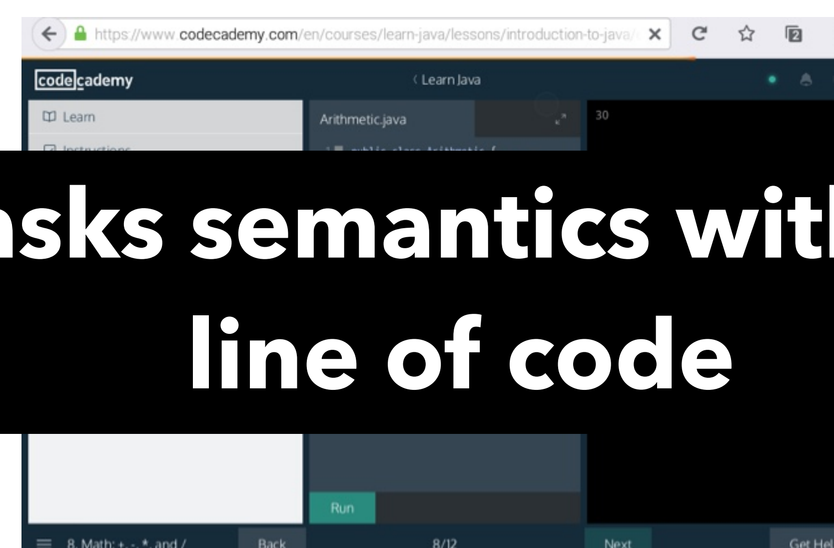
Ambiguous, weak syntax mapping

Write code



Requires learner to *infer* semantics

Step through execution



Masks semantics within a line of code

How *should* we teach syntax semantics?

Teach a “notional machine” du Boulay 1989

1. Show *each step* of semantics and their **explicit effects** on the program counter, call stack, and memory

2. **Map** semantics to concrete syntax, creating an association between syntax and its side effects

Three ideas

Gidget

Mike Lee, Ph.D.

Learners *discover* semantics through debugging

PLTutor

Greg Nelson

Tutor explicitly *teaches* semantics

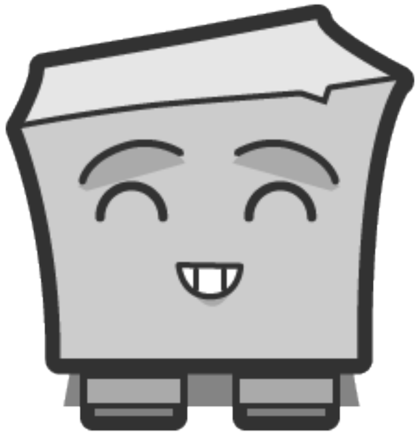
Tracing Strategies

Benji Xie

Learner reminded to follow semantics

Gidget helpgidget.org

Mike Lee



- Frame coding as a **collaboration** between a person and computer
- Give learner a sequence of **debugging puzzles**
- Guide learners' attention to **contextualized instruction on syntax and semantics** of it's Pythonic language

Gidget helpgidget.org Lee et al. 2014, VL/HCC

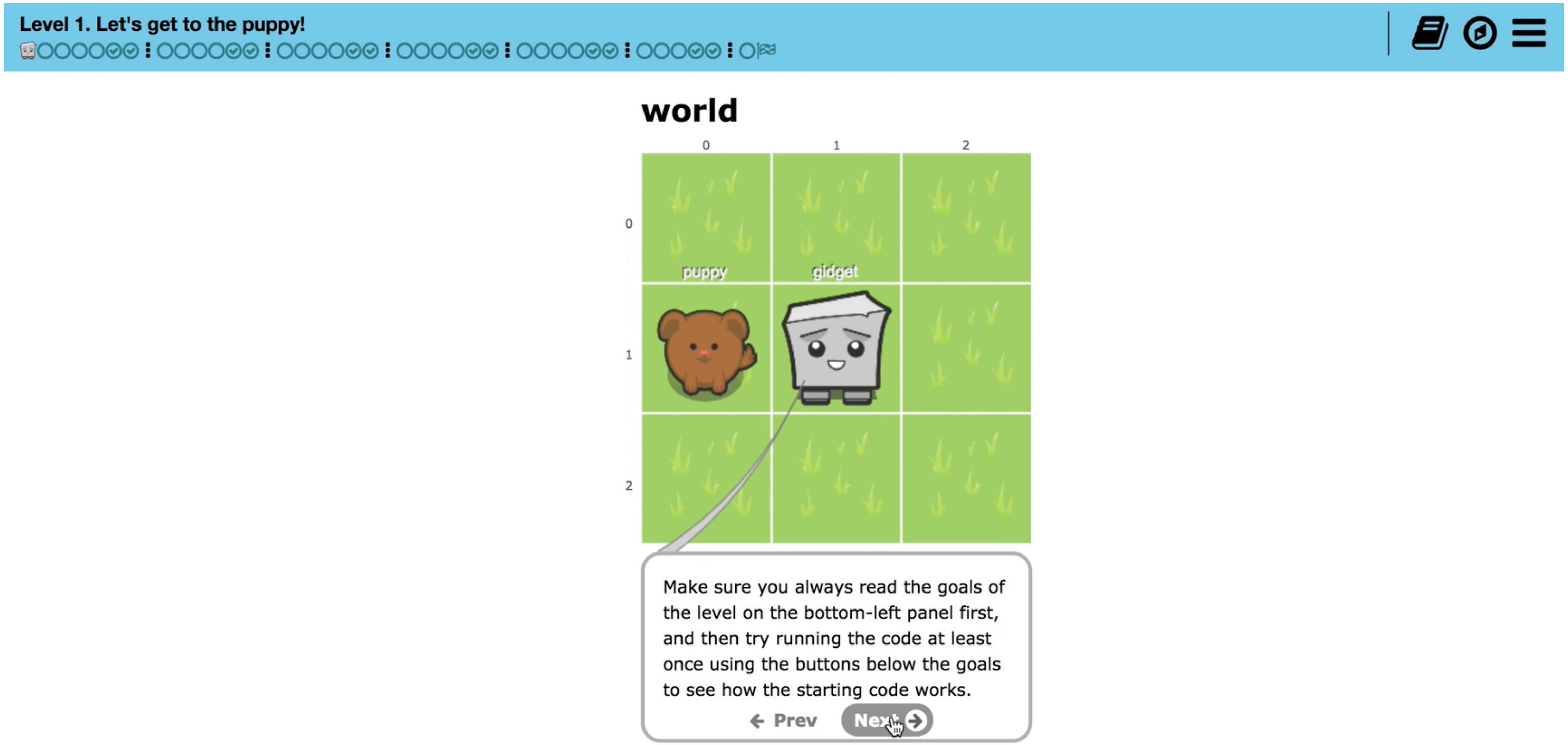
Mike Lee

Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee

Gidget helpgidget.org Lee et al. 2014, VL/HCC

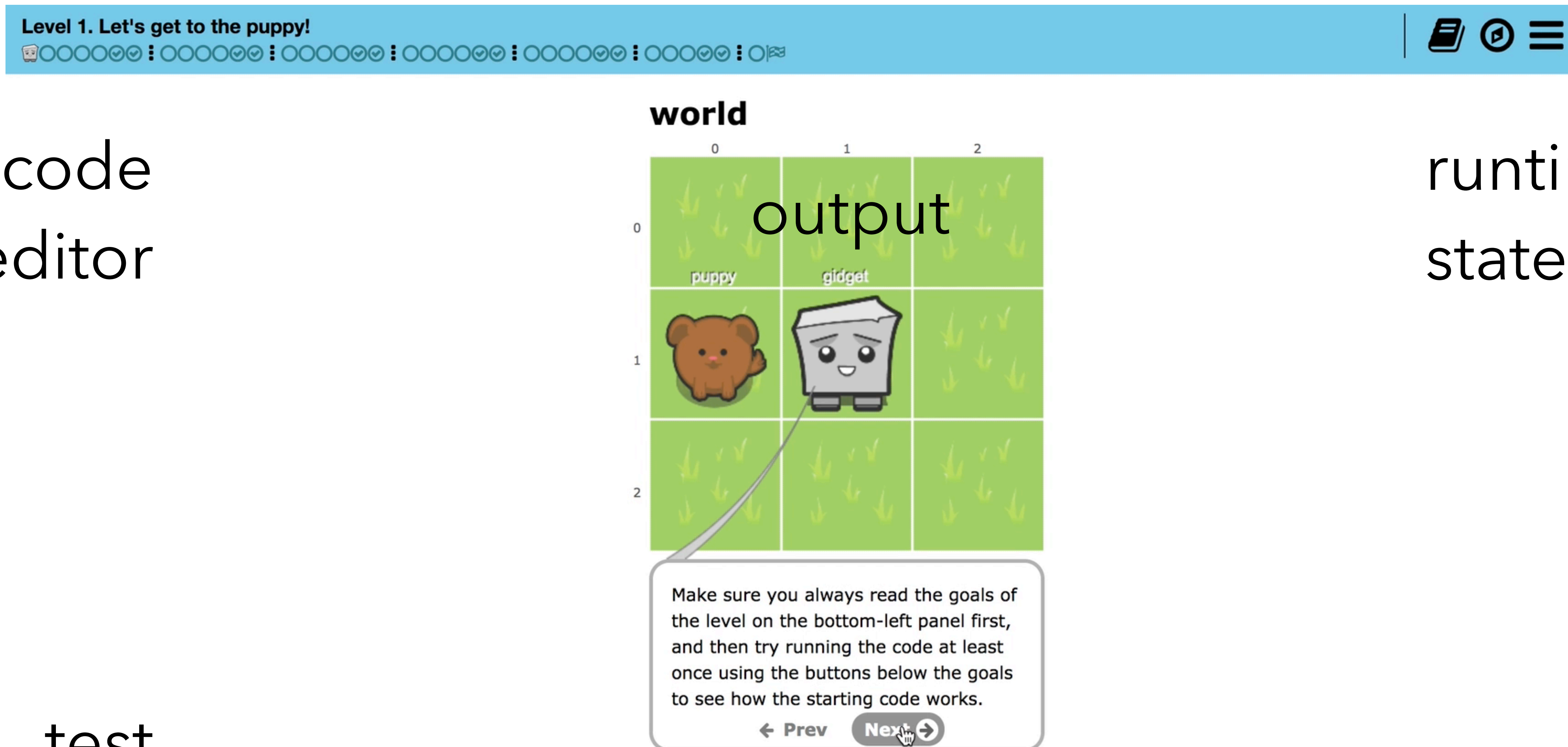
Mike Lee



Q&A

Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee



code editor

output

runtime state

test case

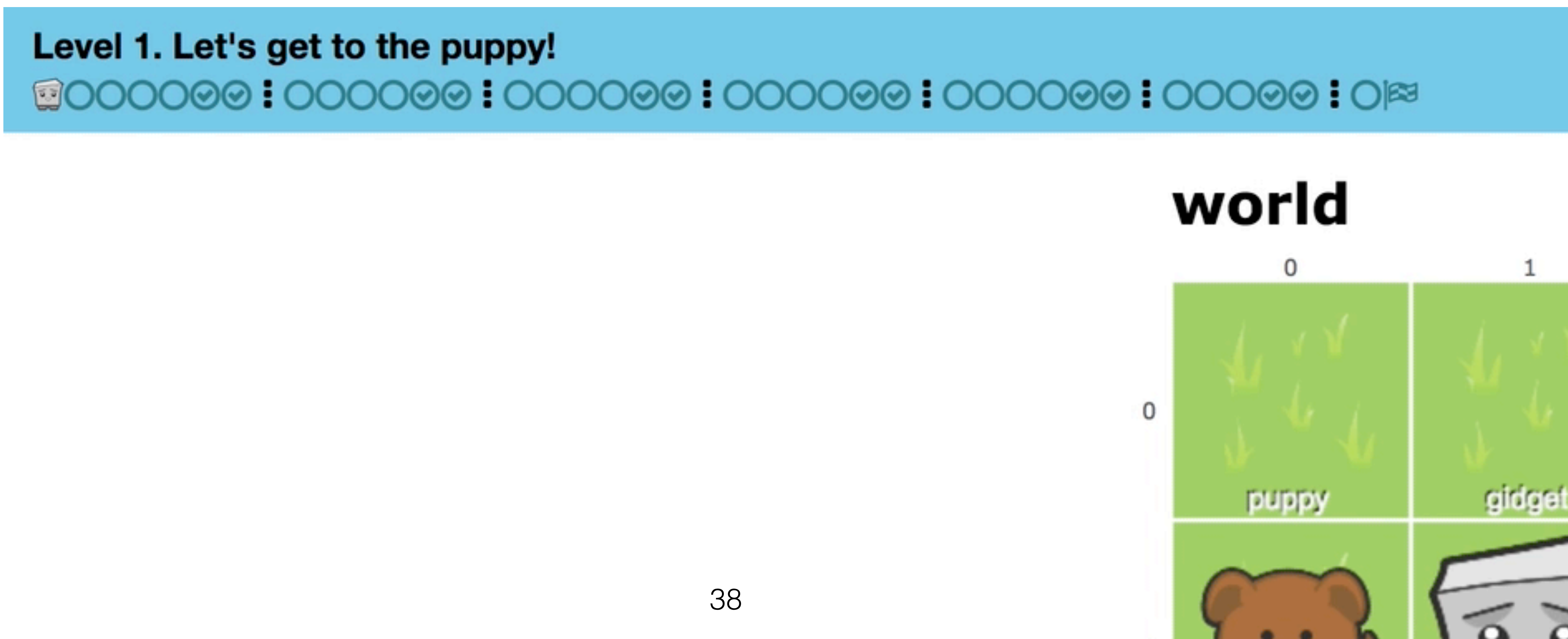
explanation

Q&A

Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee

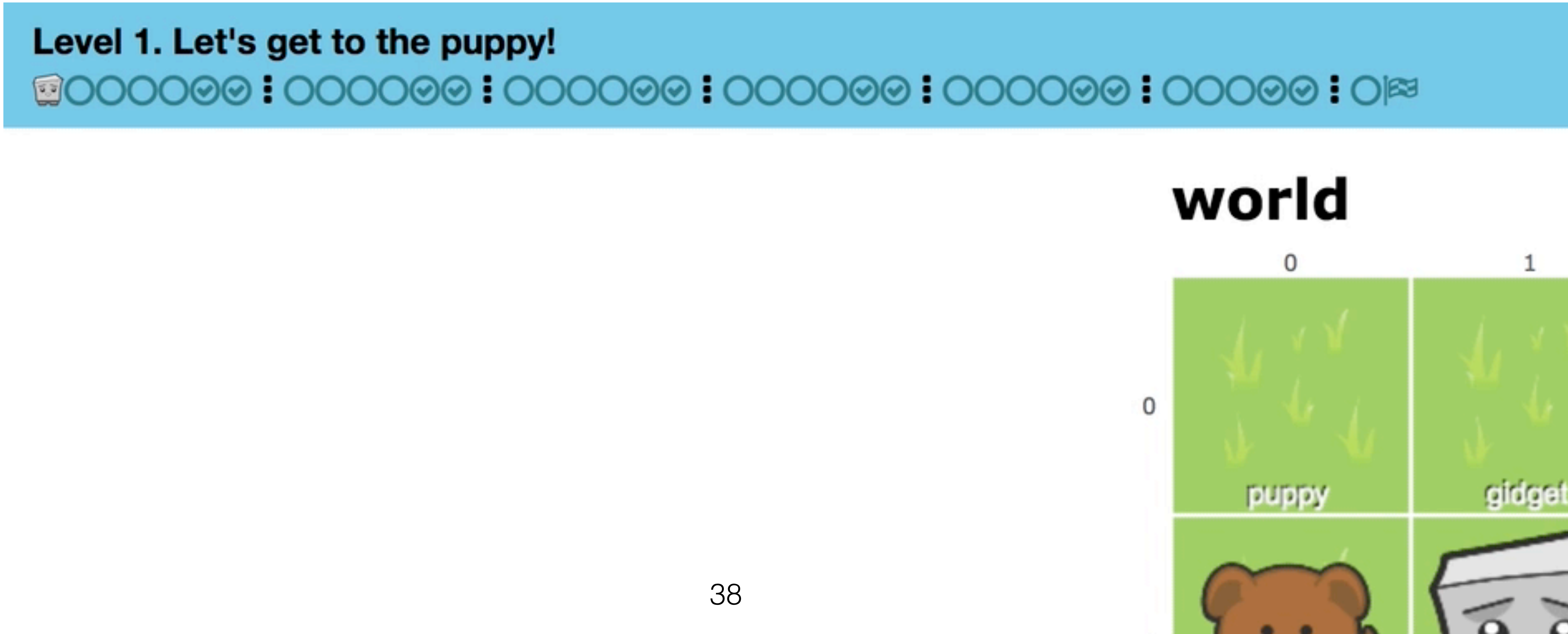
37 levels teaching 12 semantics, including formative assessments to verify understanding



Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee

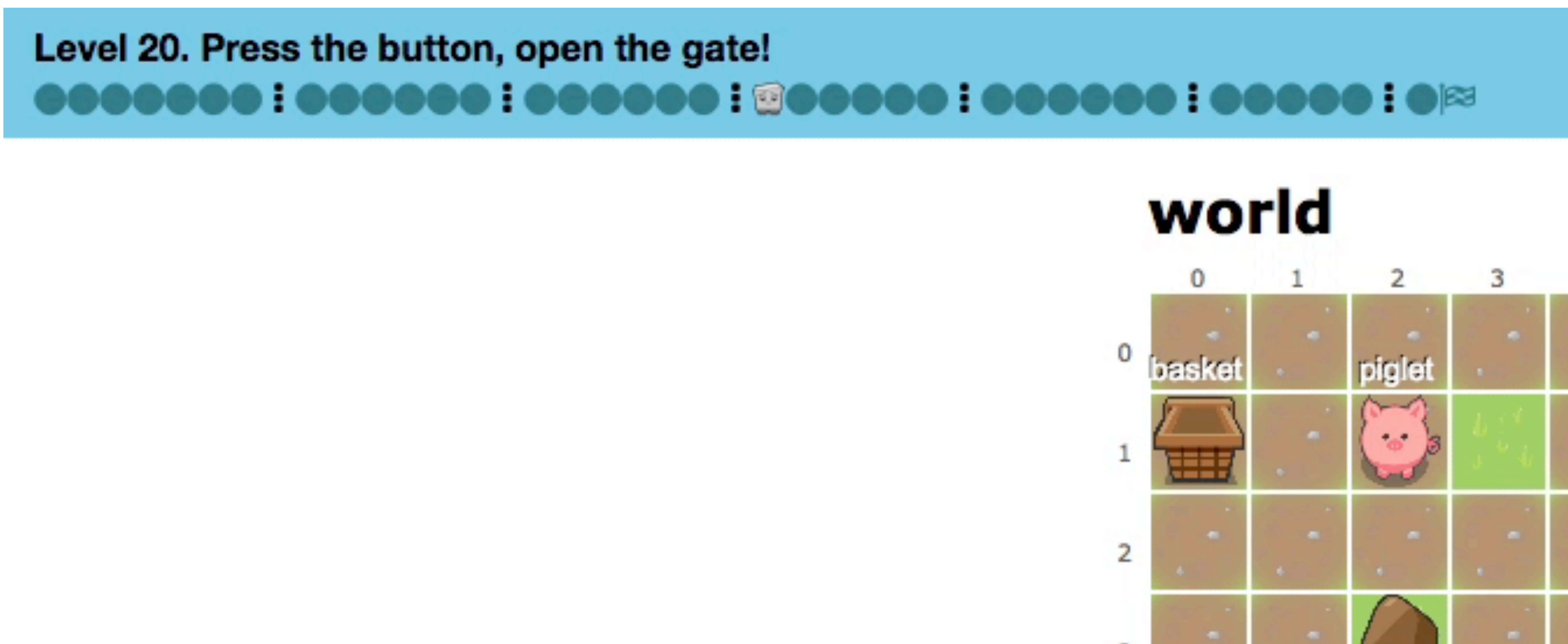
37 levels teaching 12 semantics, including formative assessments to verify understanding



Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee

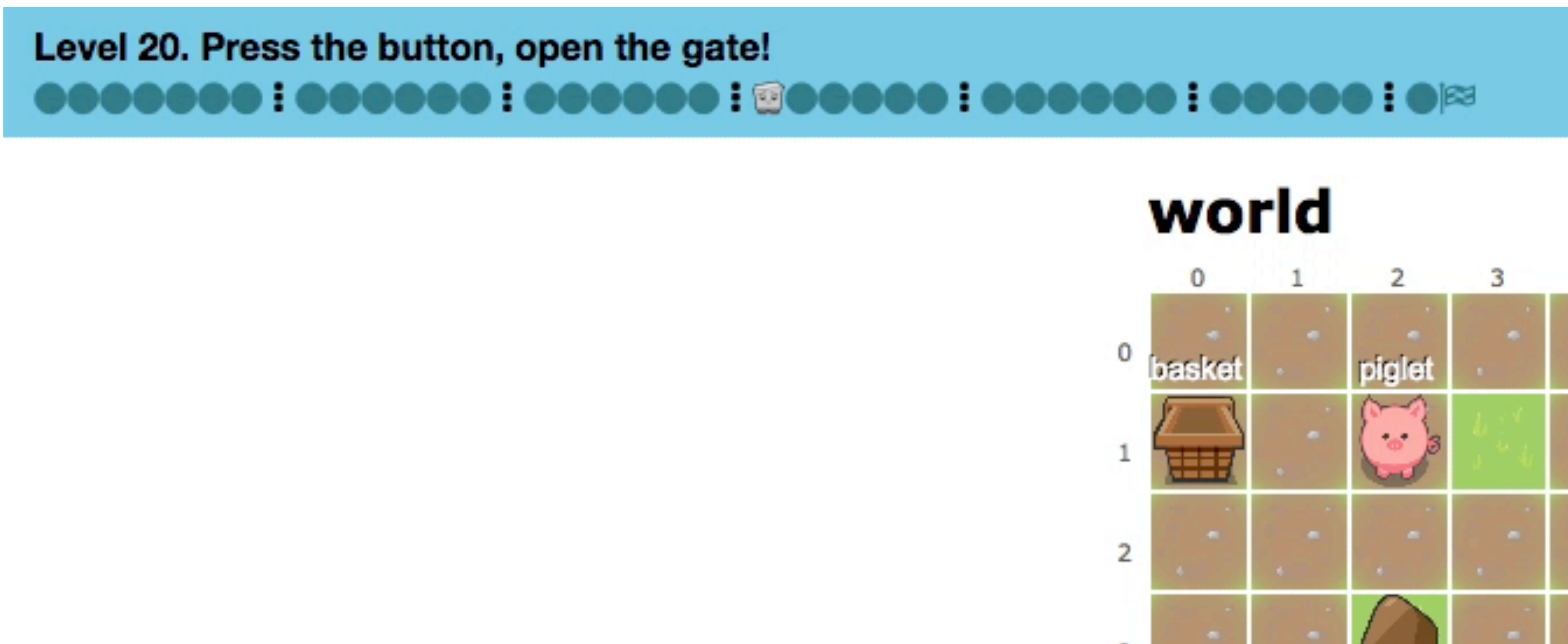
level 20 teaches function calls

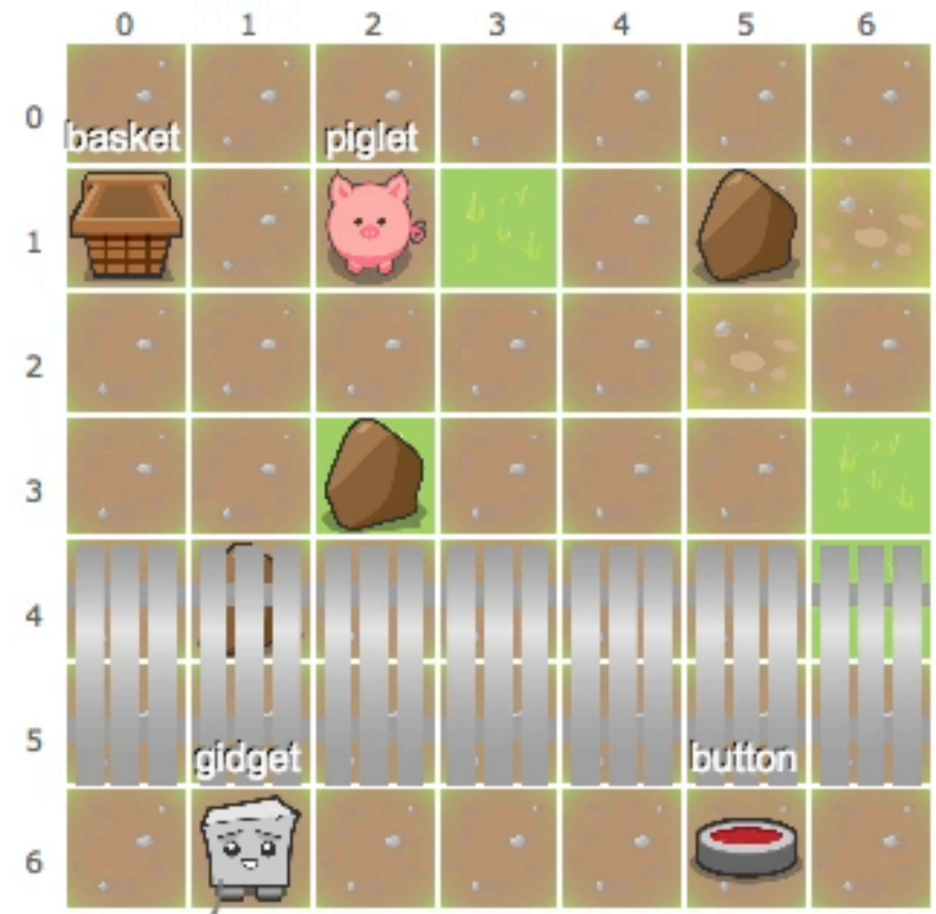


Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee

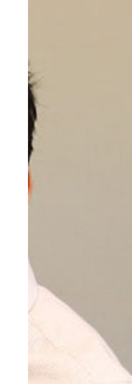
level 20 teaches function calls



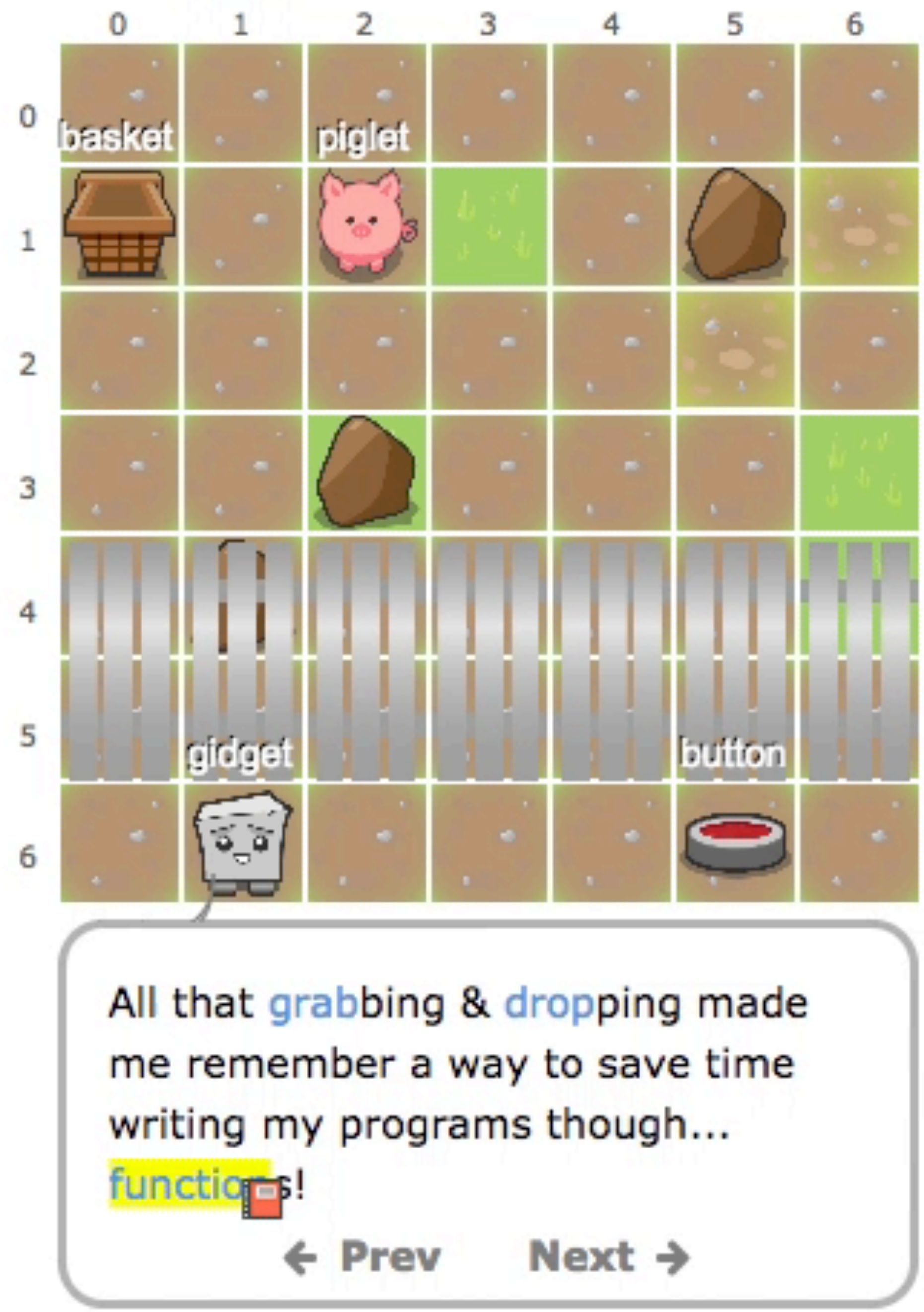


All that grabbing & dropping made me remember a way to save time writing my programs though... functions!

← Prev Next →



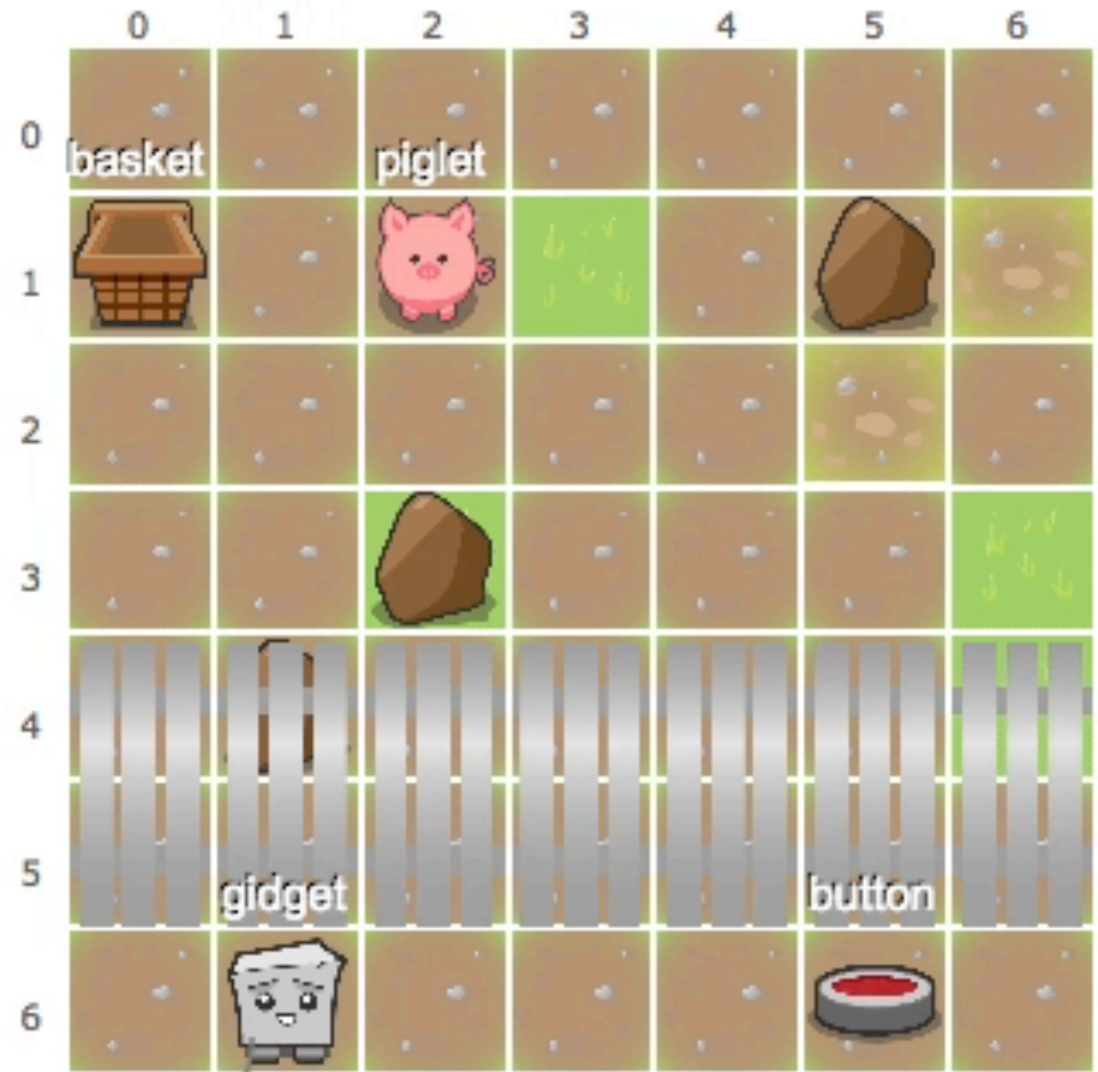
Q&A



Function definition semantics

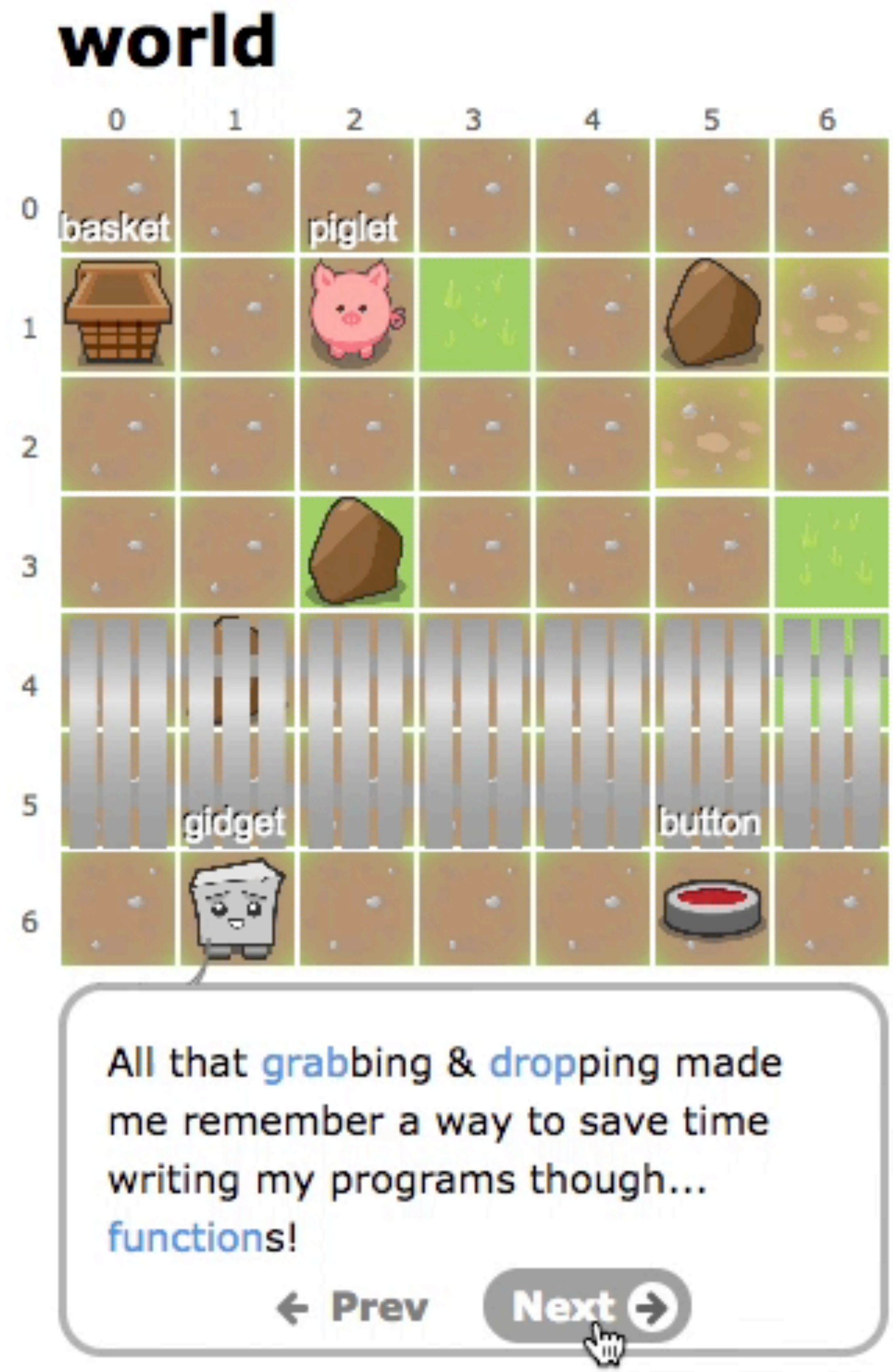
Q&A

world



All that grabbing & dropping made me remember a way to save time writing my programs though... functions!

← Prev Next →

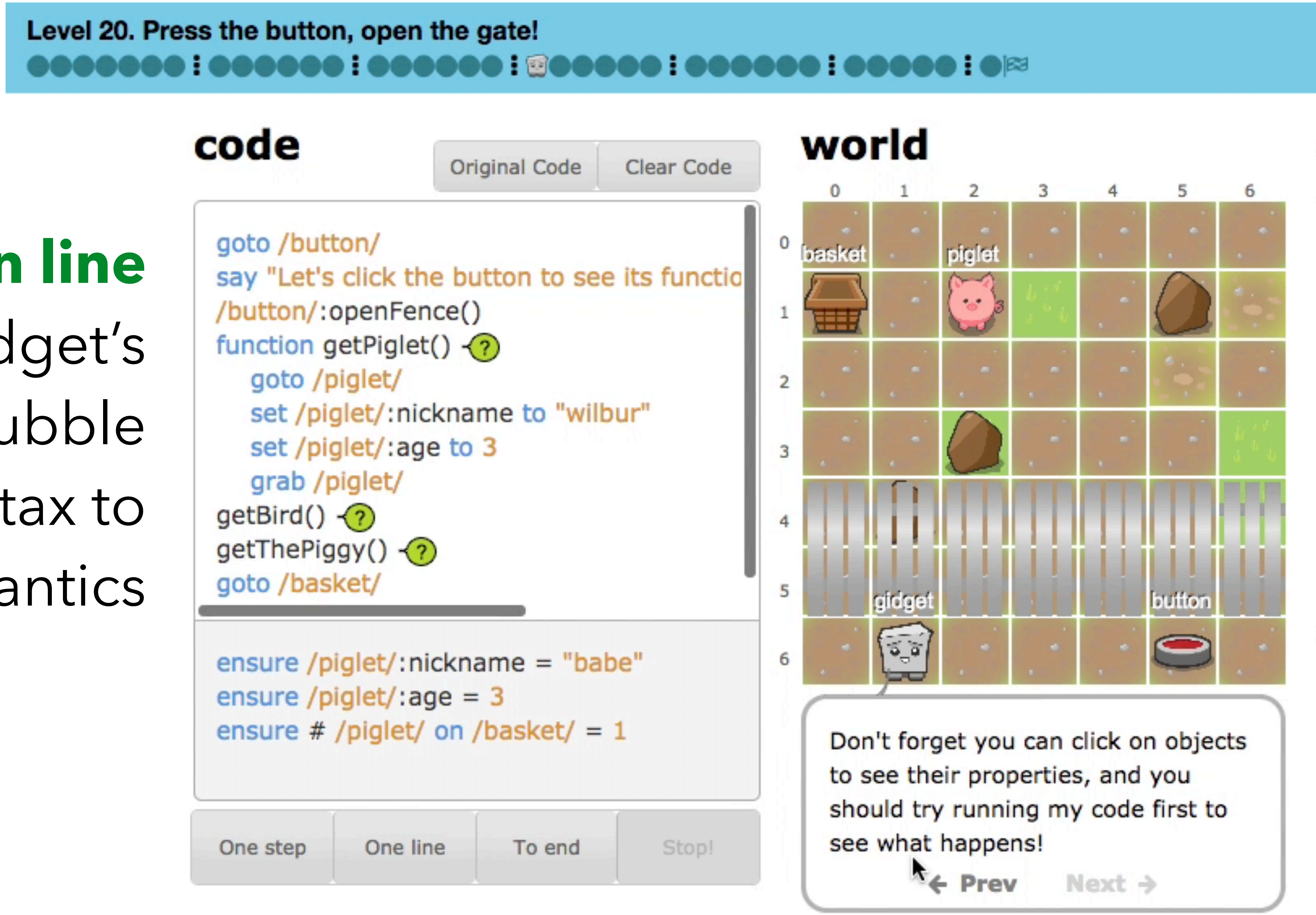


Gidget explains syntax and semantics

Gidget helpgidget.org Lee et al. 2014, VL/HCC

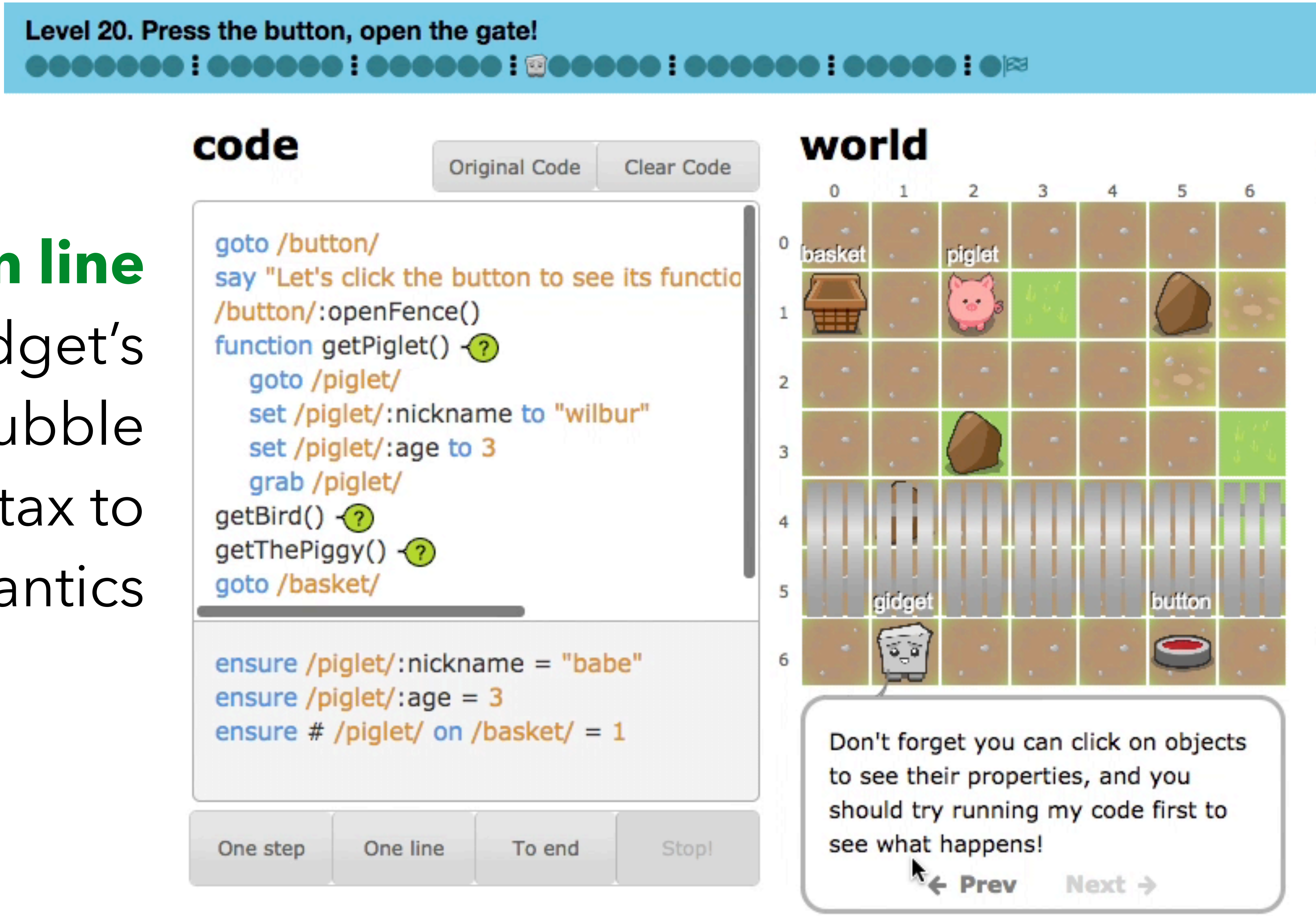
Mike Lee

The **green line** and Gidget's speech bubble maps syntax to semantics



Gidget helpgidget.org Lee et al. 2014, VL/HCC

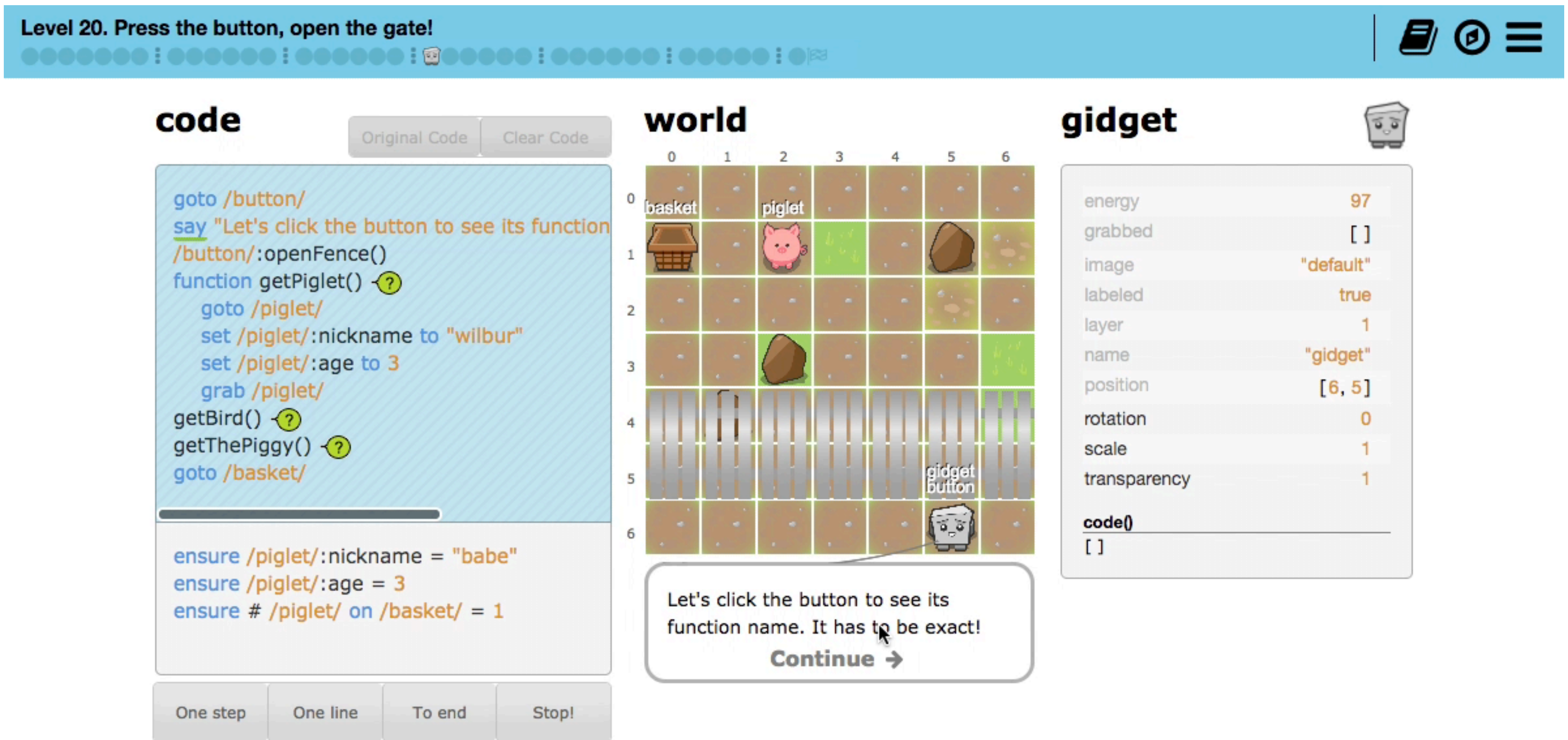
Mike Lee



The **green line** and Gidget's speech bubble maps syntax to semantics

Gidget helpgidget.org Lee et al. 2014, VL/HCC

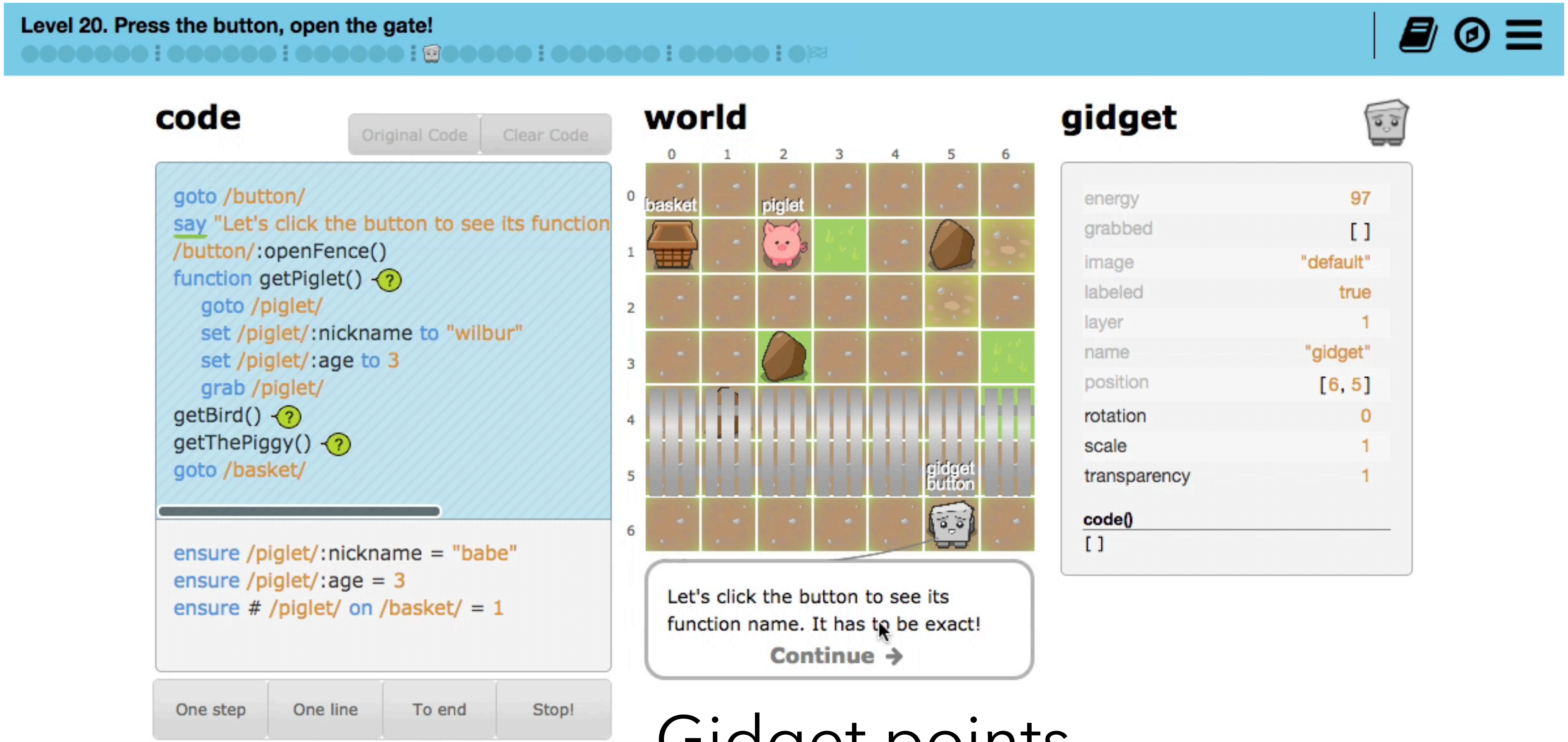
Mike Lee



Q&A

Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee

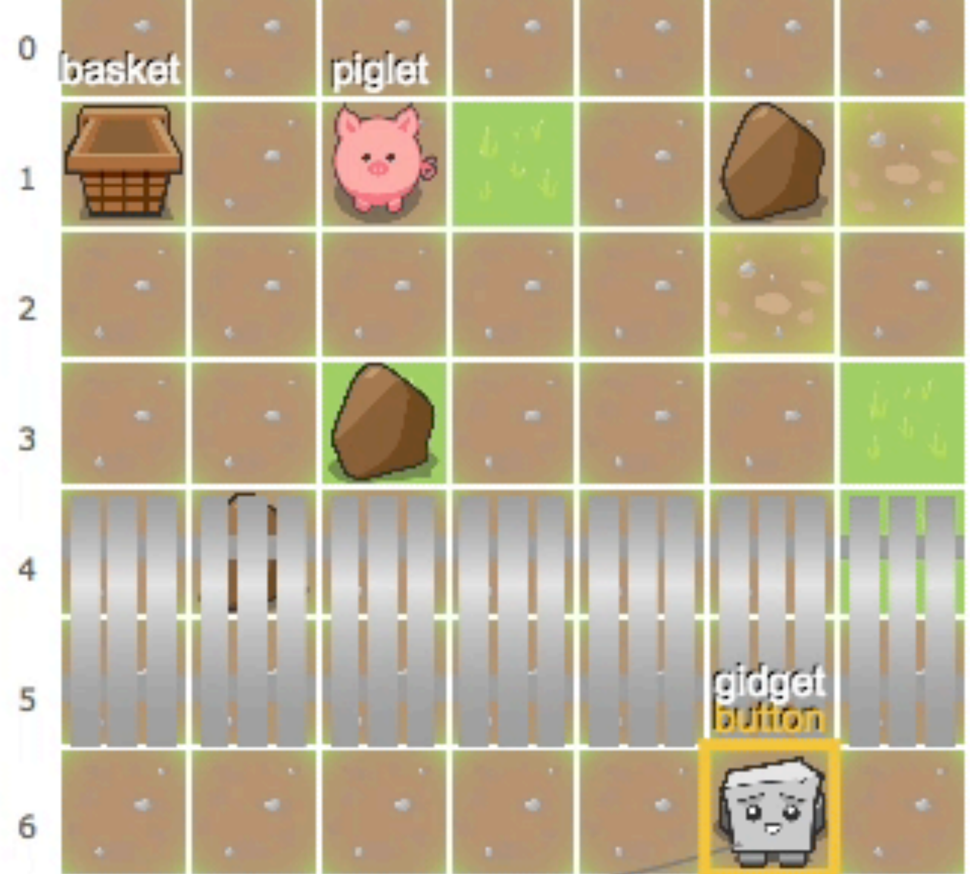


Gidget points out function definitions

```
to /button/
y "Let's click the button to see its function
utton/:openFence()
nction getPiglet() ?
  goto /piglet/
  set /piglet/:nickname to "wilbur"
  set /piglet/:age to 3
  grab /piglet/
tBird() ?
tThePiggy() ?
to /basket/
```

```
sure /piglet/:nickname = "babe"
sure /piglet/:age = 3
sure # /piglet/ on /basket/ = 1
```

e step | One line | To end | Stop!



Let's click the button to see its function name. It has to be exact!

Continue

energy	97
grabbed	[ ]
image	"default"
labeled	true
layer	1
name	"gidget"
position	[6, 5]
rotation	0
scale	1
transparency	1

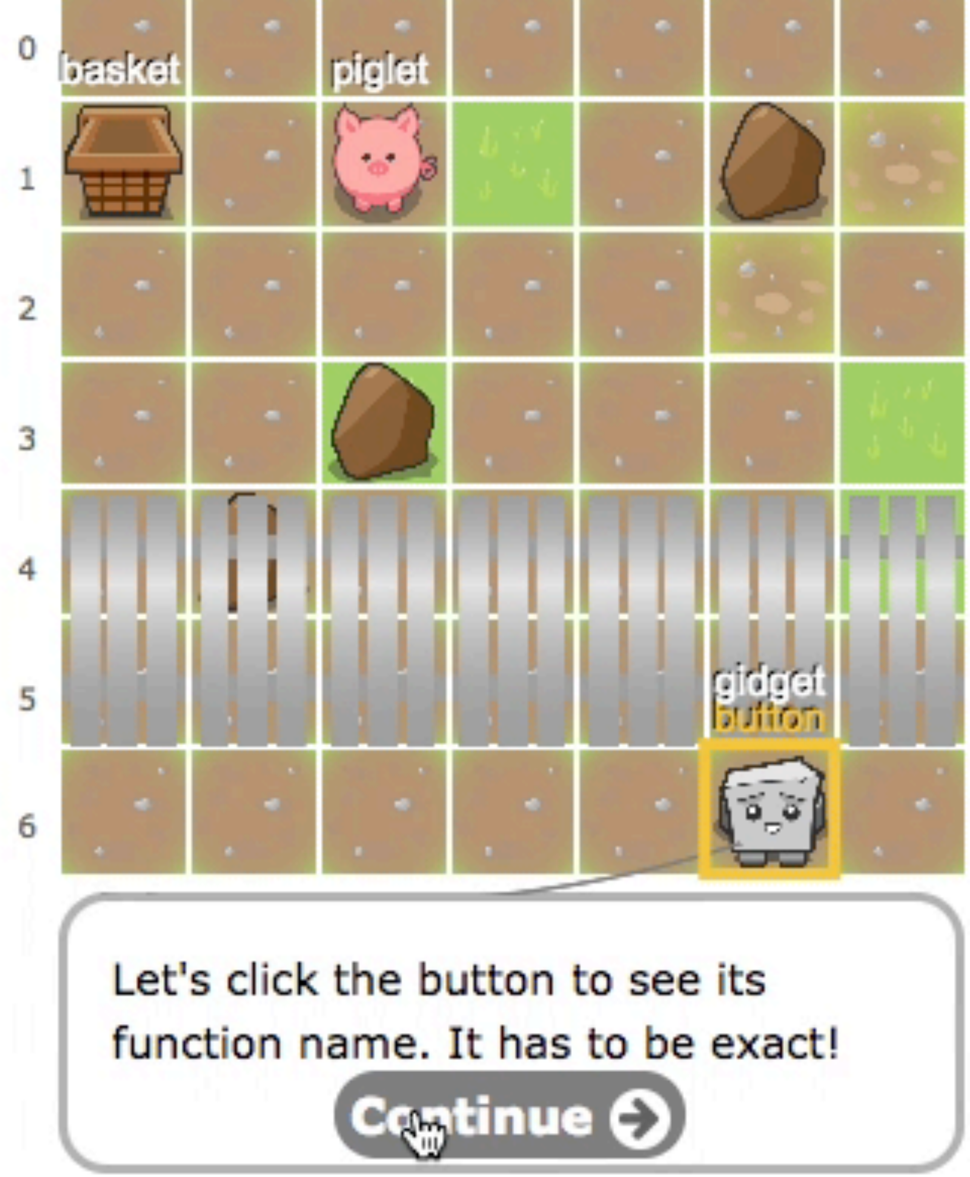
code()

[ ]

button

energy	100
grabbed	[ ]
image	"default"
labeled	true
layer	1
name	"button"
openGate	openGate()
position	[6, 5]
rotation	0
scale	1
transparency	1

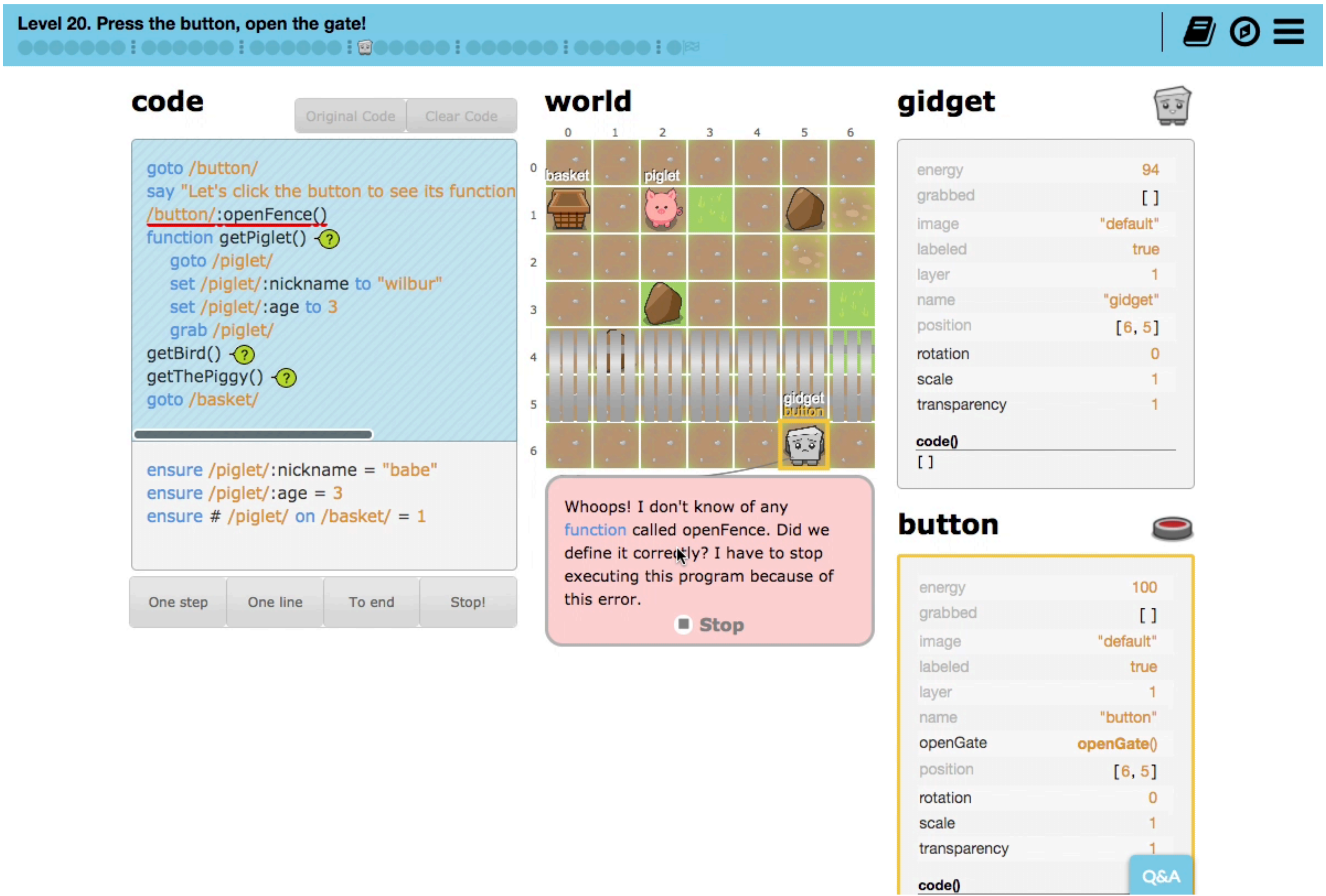
code()



Gidget explains name resolution semantics

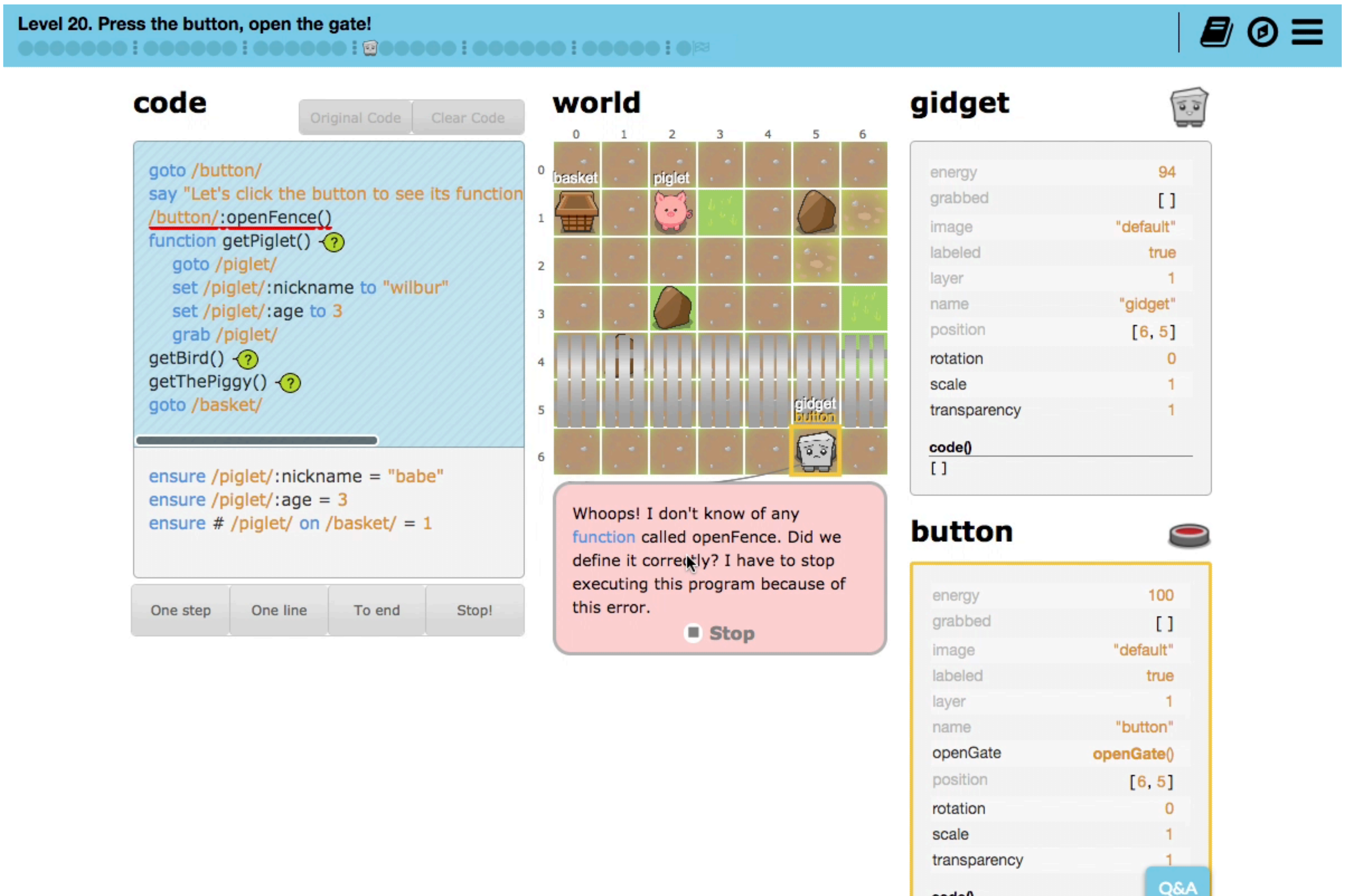
Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee



Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee



Gidget helpgidget.org


Mike Lee

- Four online controlled experiments with over 1,000 adult learners:
  - Learning is **2x as fast** as Codecademy tutorial, **2x as much** as open-ended creative exploration Lee & Ko 2015
  - **Assessment levels** significantly increase learning efficiency Lee et al. 2013
  - Giving compiler a face and using personal pronouns (I, you, we) draws learner's attention to semantics, **doubling learning efficiency** Lee & Ko 2011
  - Changes attitudes about difficulty of learning to code from negative to positive in **20 minutes** Charters et al. 2014

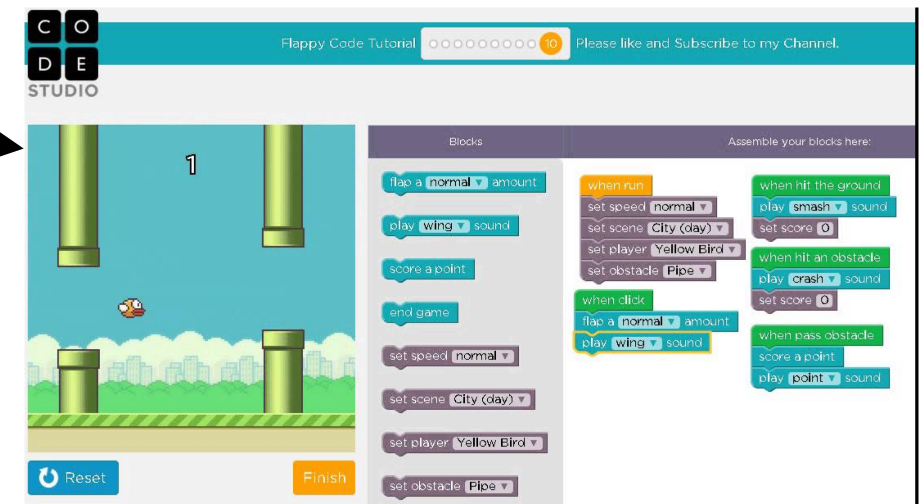
Gidget helpgidget.org

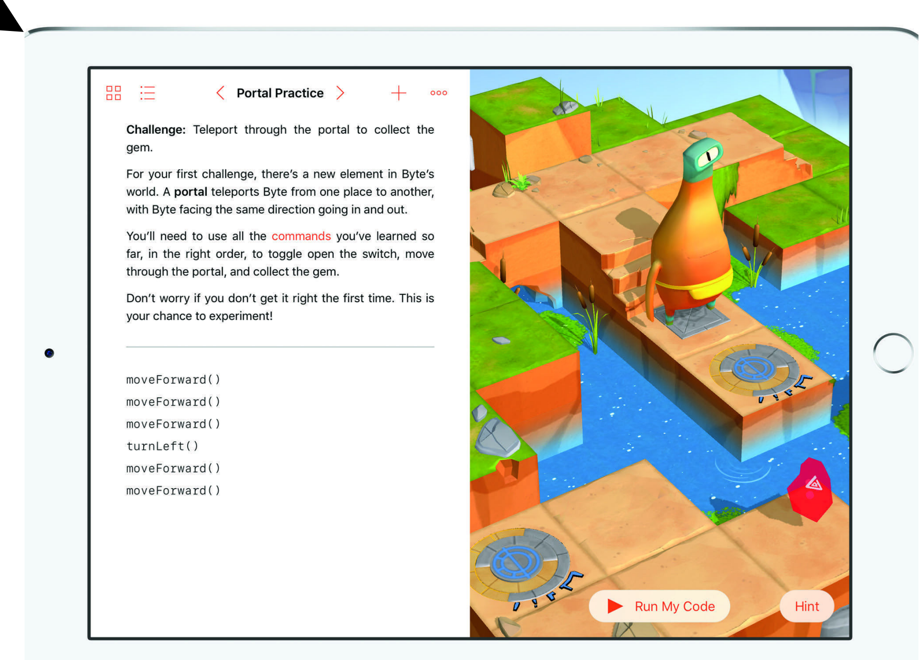
Mike Lee

- Four online controlled experiments with over 1,000 adult learners:
  - Learning is **2x as fast** as Codecademy tutorial, **2x as much** as open-ended creative exploration Lee & Ko 2015
  - **Assessment levels** significantly increase learning efficiency Lee et al. 2013
  - Giving compiler a face and using personal pronouns (I, you, we) draws learner's attention to semantics, **doubling learning efficiency** Lee & Ko 2011
  - Changes attitudes about difficulty of learning to code from negative to positive in **20 minutes** Charters et al. 2014

Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee



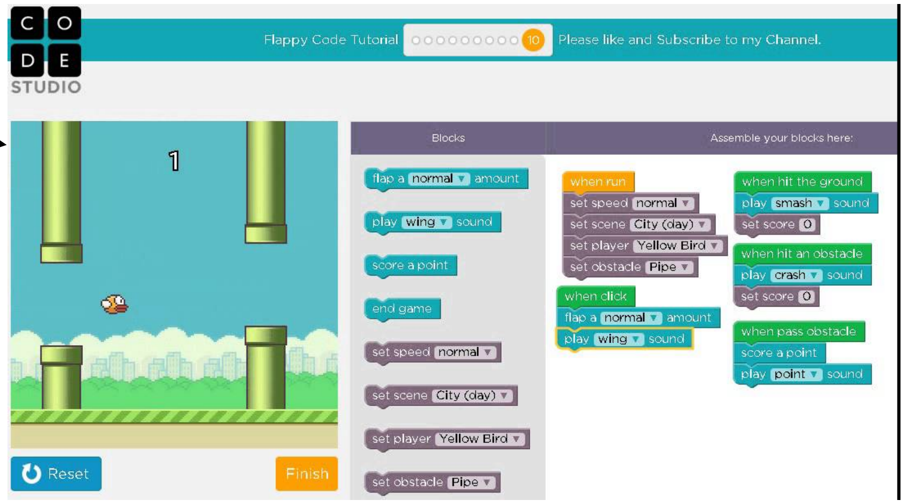


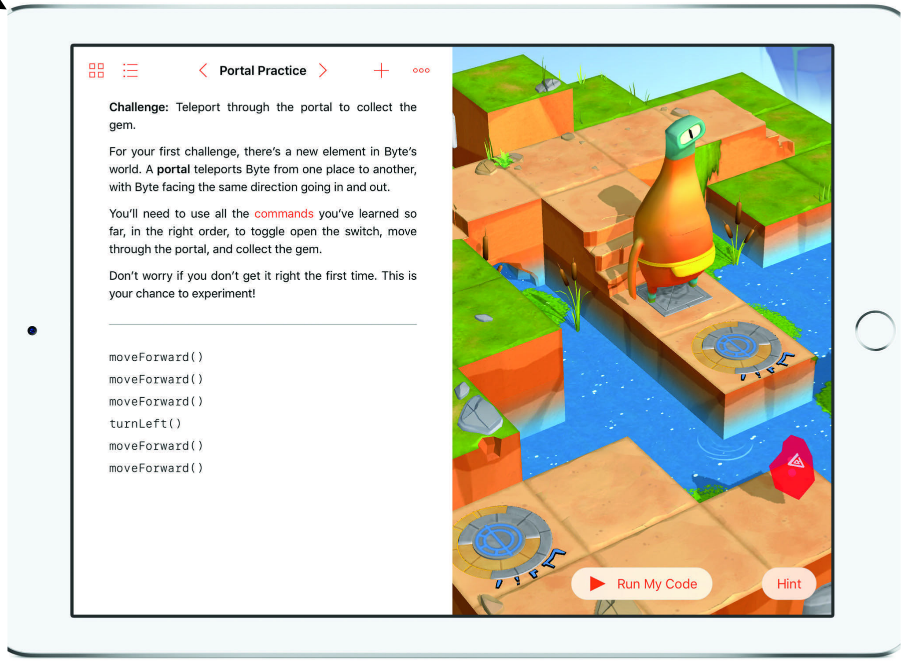
20,000+ have played via word of mouth, including Chicago Public Schools, many retirees (apparently including my mom)

Directly impacted the design of code.org's CodeStudio and Apple's Swift Playgrounds, used by 10+ million learners.

Gidget helpgidget.org Lee et al. 2014, VL/HCC

Mike Lee





20,000+ have played via word of mouth, including Chicago Public Schools, many retirees (apparently including my mom)

Directly impacted the design of <u>code.org</u>'s CodeStudio and Apple's Swift Playgrounds, used by 10+ million learners.

Three ideas

Gidget

Mike Lee, Ph.D.

Learners discover semantics through debugging

PLTutor

Greg Nelson

Tutor explicitly *teaches* semantics

Tracing Strategies

Benji Xie

Learner reminded to follow semantics

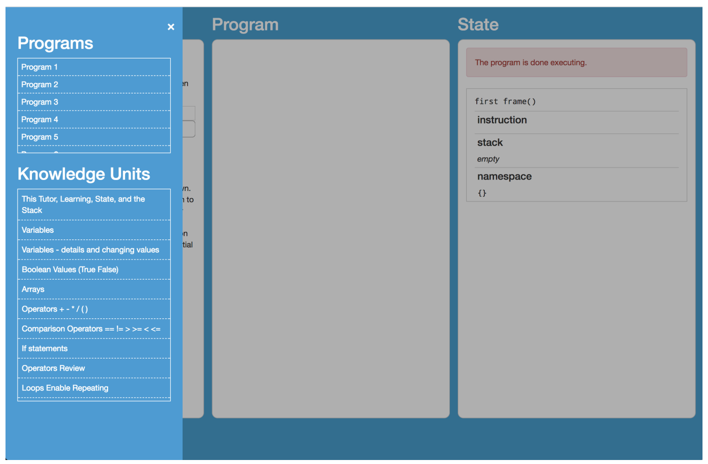
Three ideas

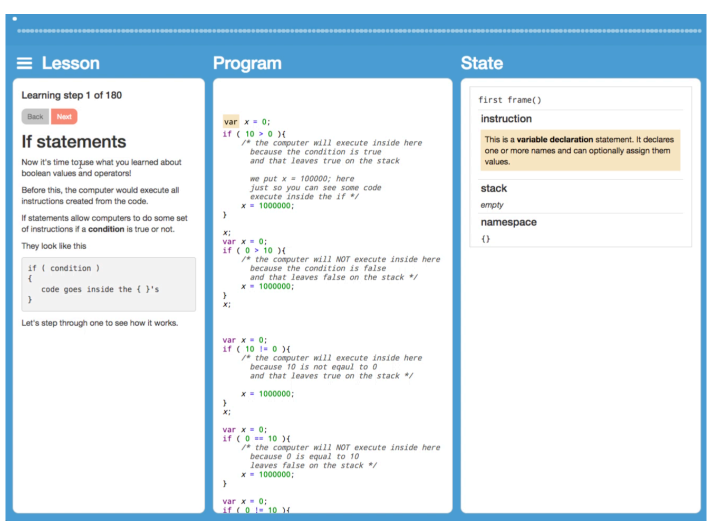
Gidget	**PLTutor**	**Tracing Strategies**
		
Mike Lee, Ph.D.	Greg Nelson	Benji Xie
Learners *discover* semantics through debugging	Tutor explicitly *teaches* semantics	Learner reminded to follow semantics

PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson



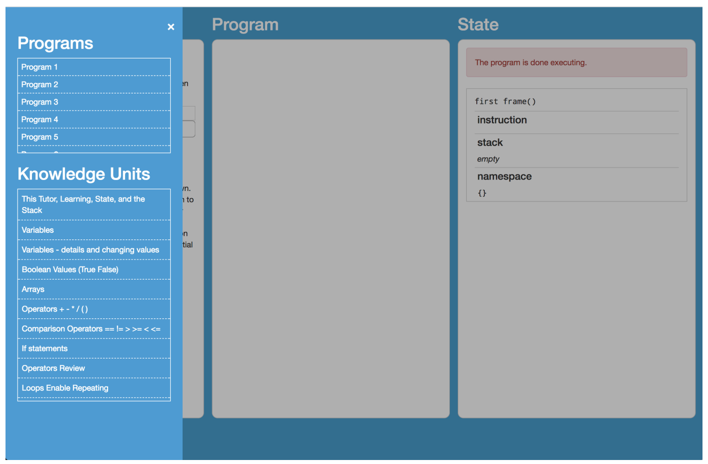


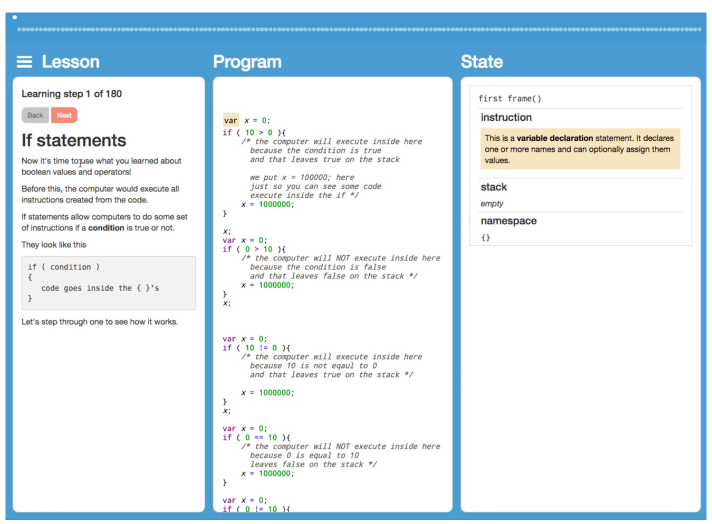
- Convert operational semantics into an **interactive textbook** to be read before learning to write programs
- Covers the entire *JavaScript* semantics in about **3 hours** of practice
- Learner should be able to accurately predict the behavior of *any* JavaScript program

PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson





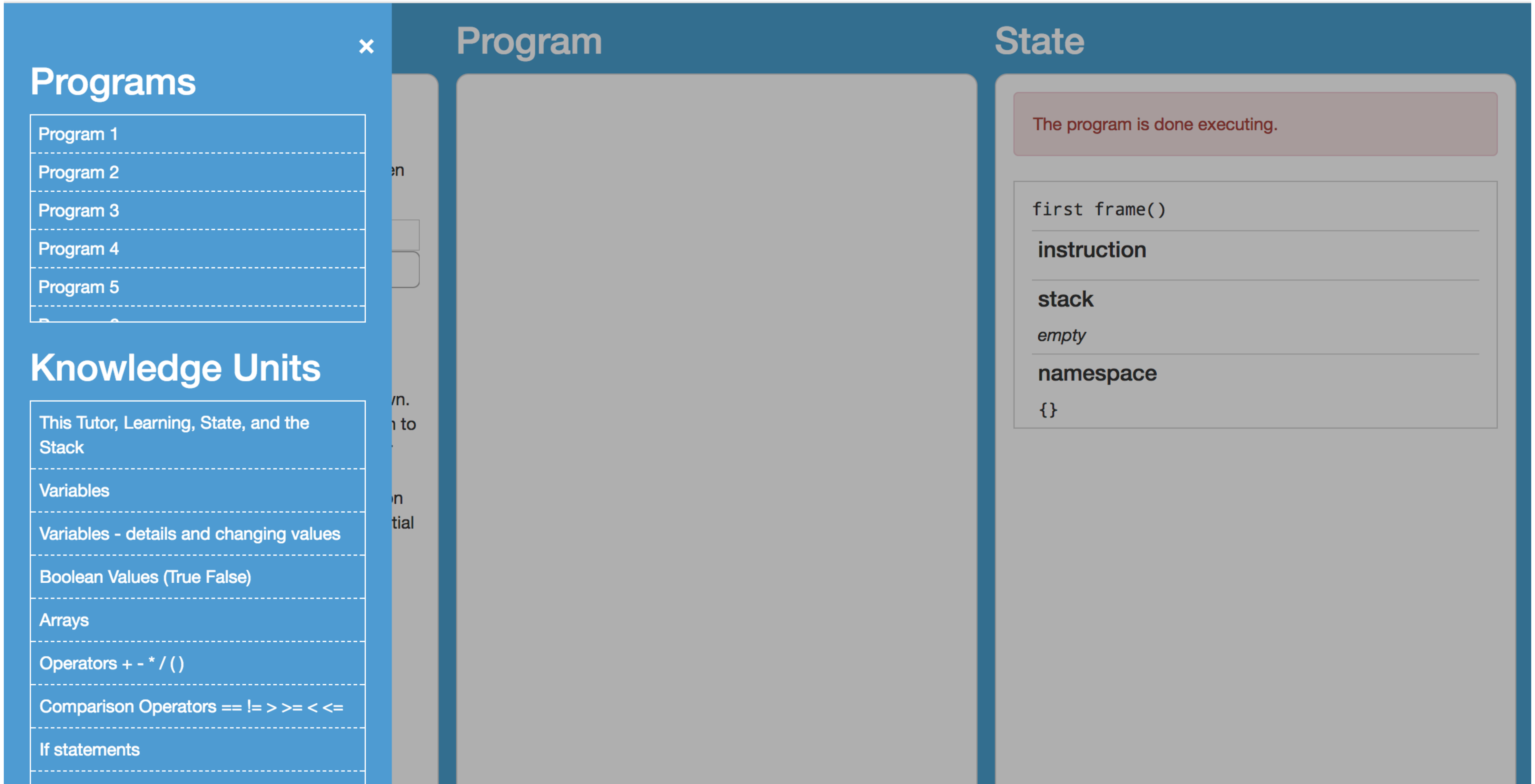
- Convert operational semantics into an **interactive textbook** to be read before learning to write programs
- Covers the entire *JavaScript* semantics in about **3 hours** of practice
- Learner should be able to accurately predict the behavior of *any* JavaScript program

PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Each chapter covers a set of semantics through a series of programs



PLTutor

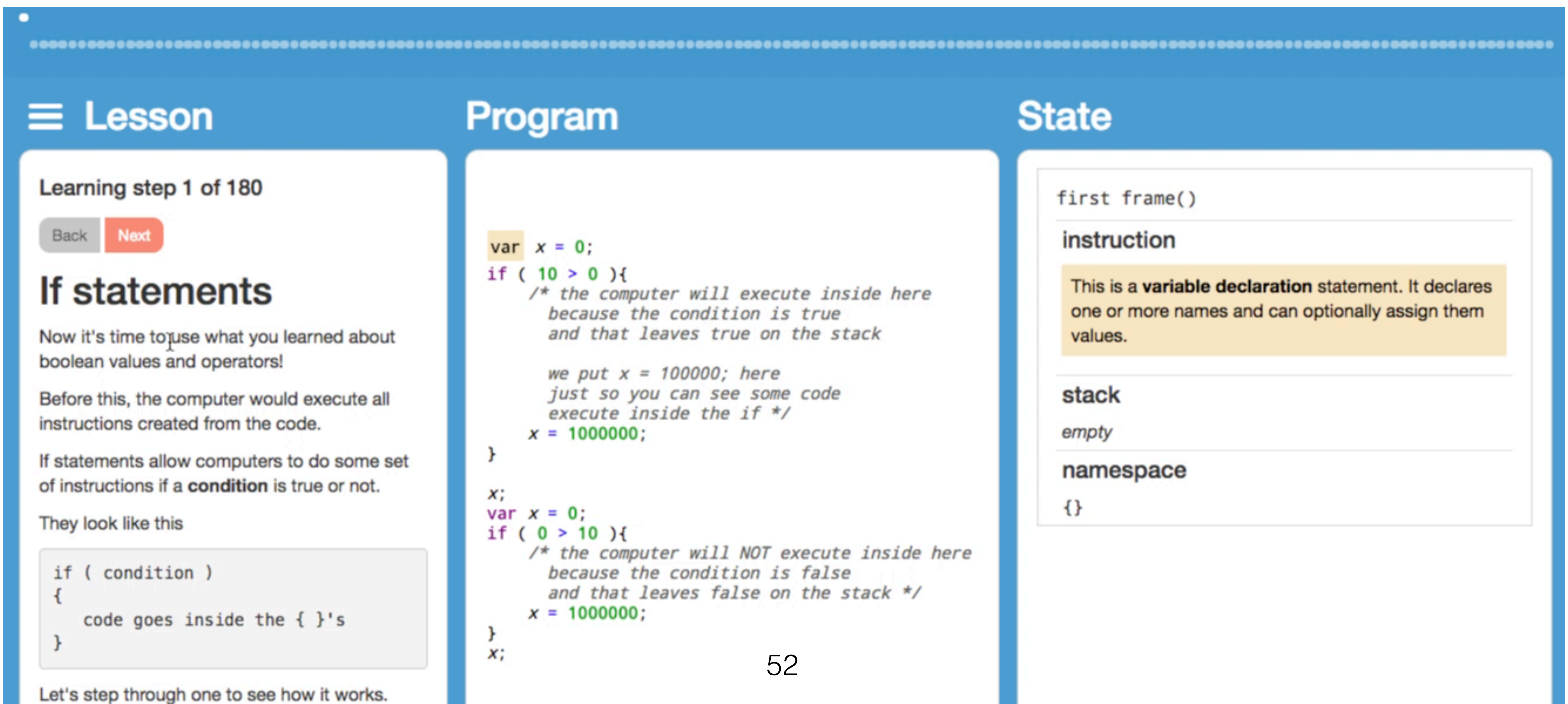
Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Lesson explains purpose of semantics

Program links syntax to semantics

State *explains* semantics



PLTutor

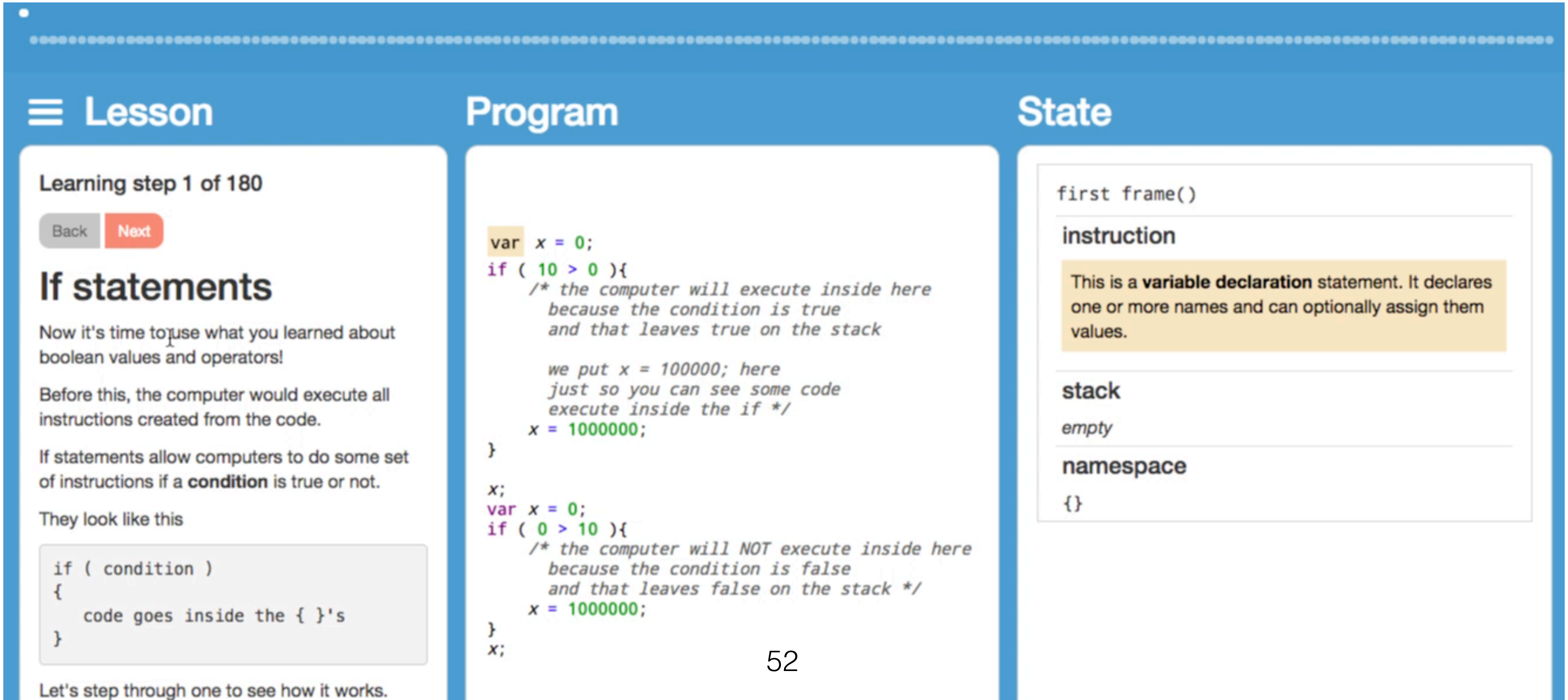
Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Lesson explains purpose of semantics

Program links syntax to semantics

State *explains* semantics

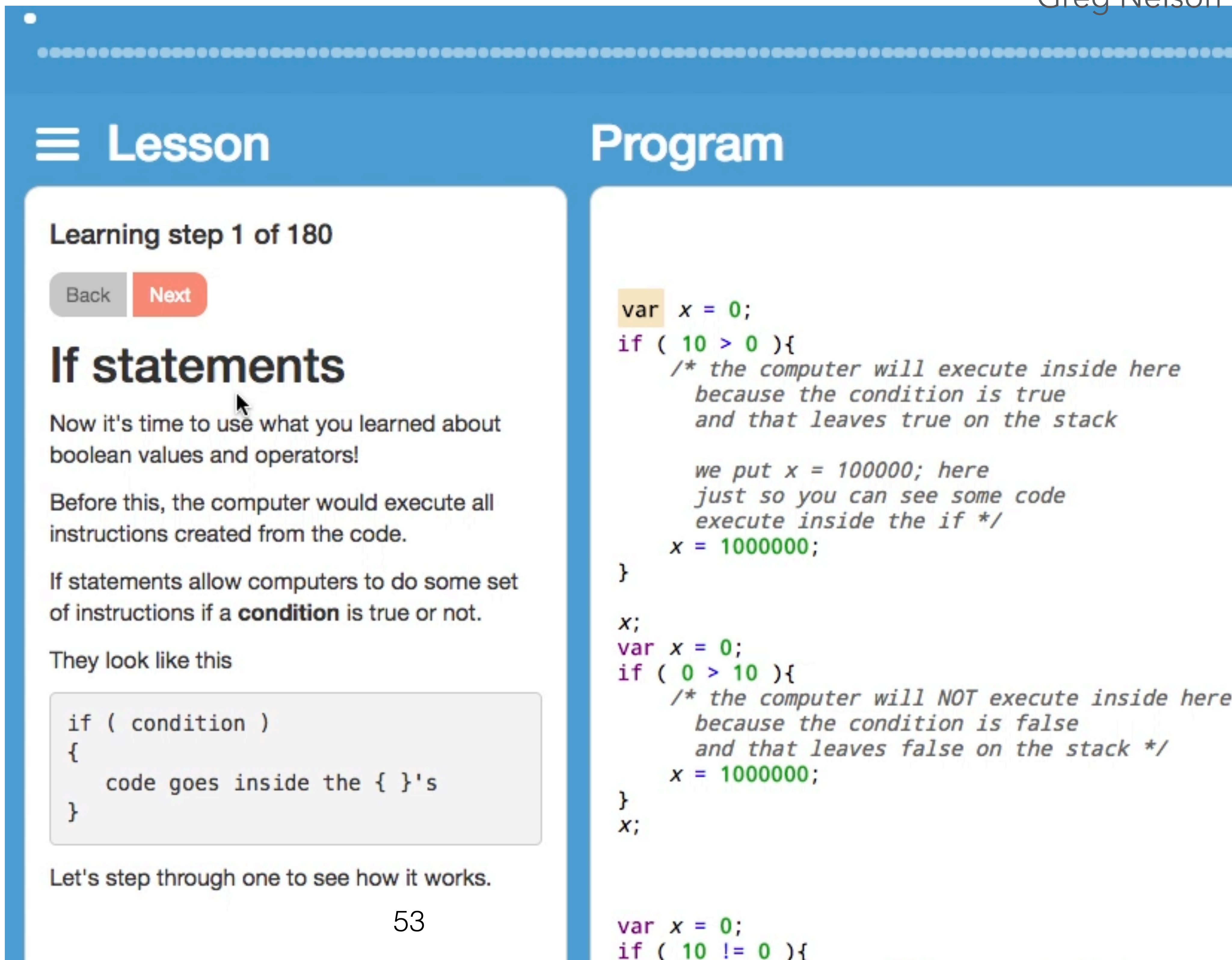


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

purpose of semantics

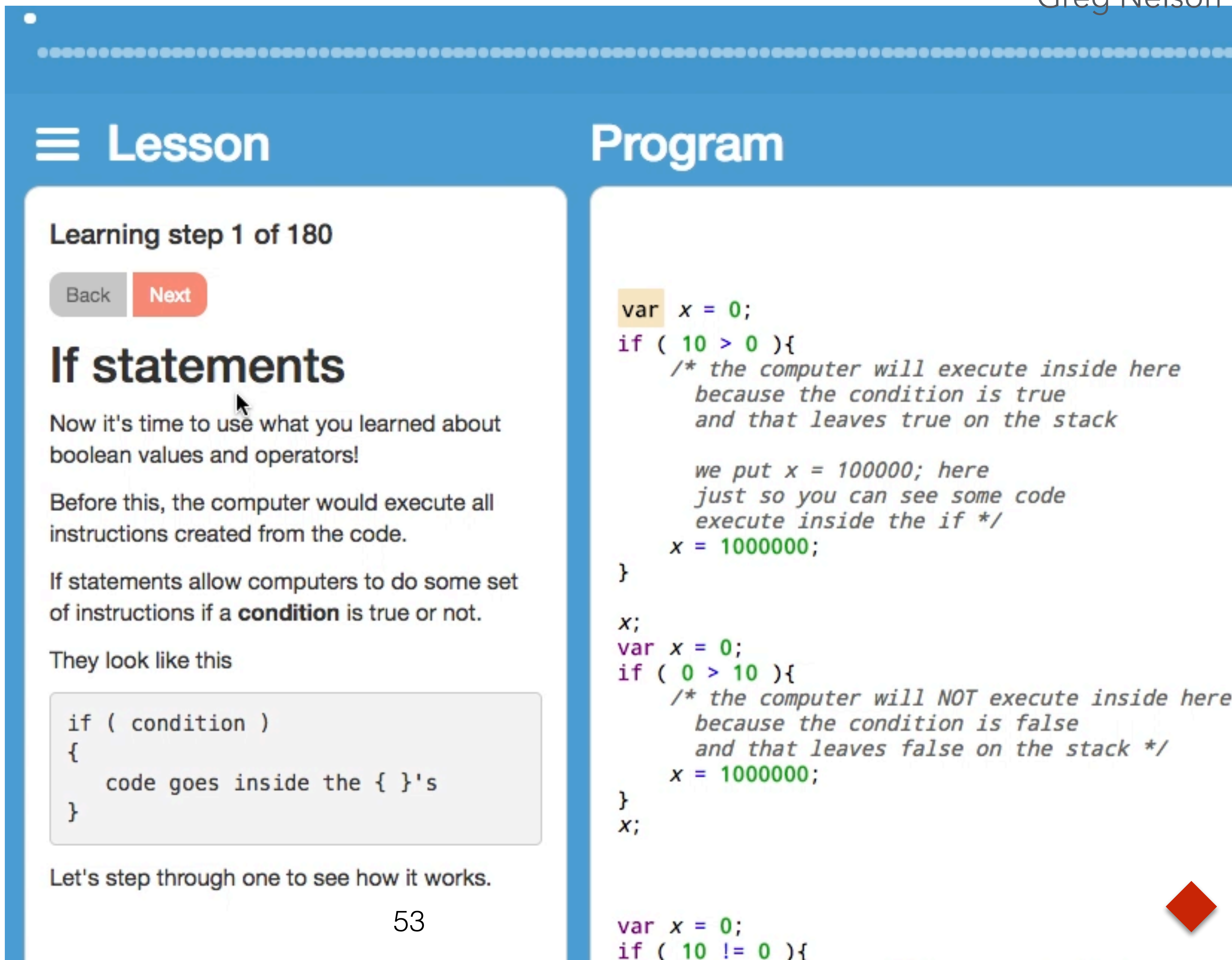


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

purpose of semantics

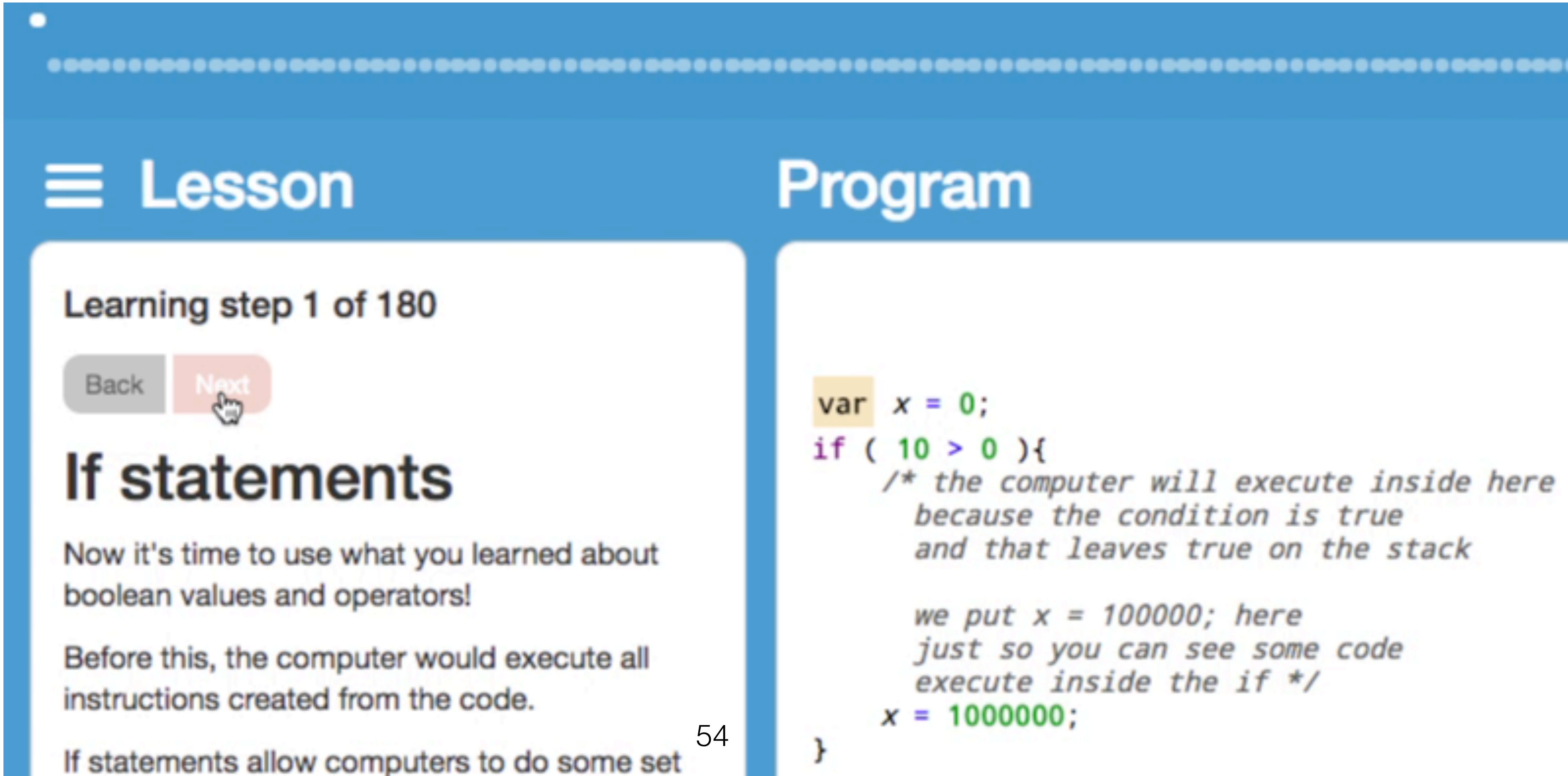


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Next moves through both program execution trace *and* instruction.

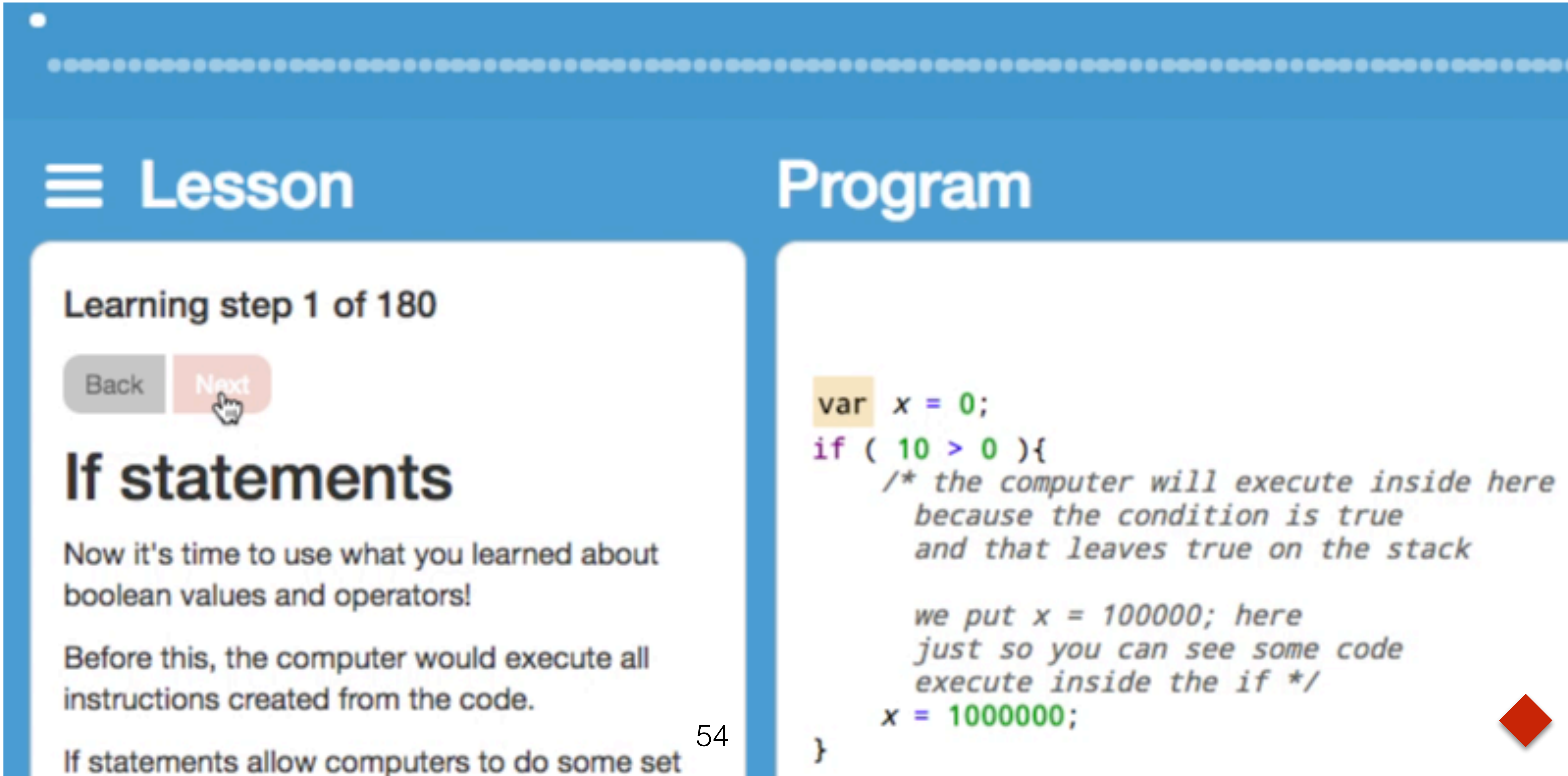


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Next moves through both program execution trace *and* instruction.

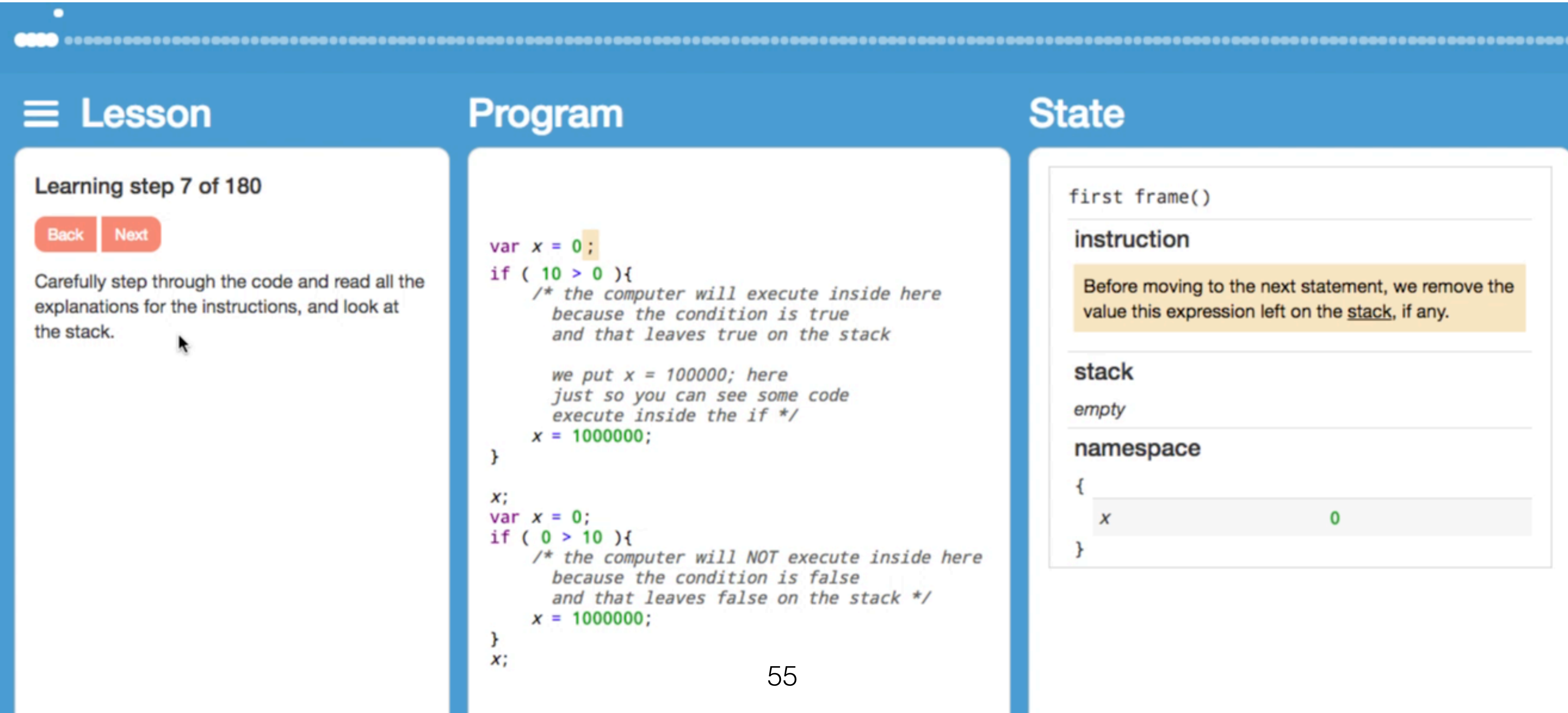


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

State teaches semantics in a runtime context,
Lesson generalizes back to purpose

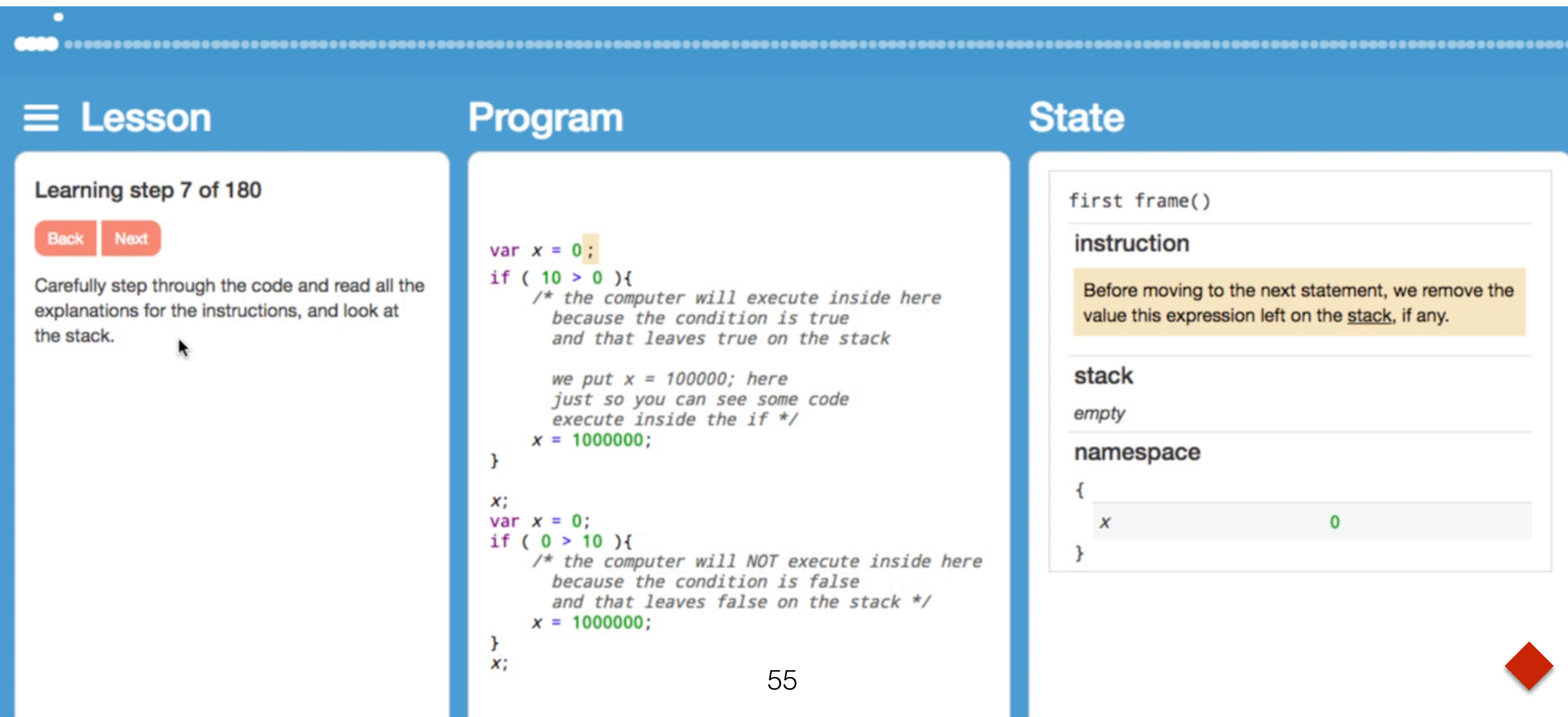


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

State teaches semantics in a runtime context,
Lesson generalizes back to purpose

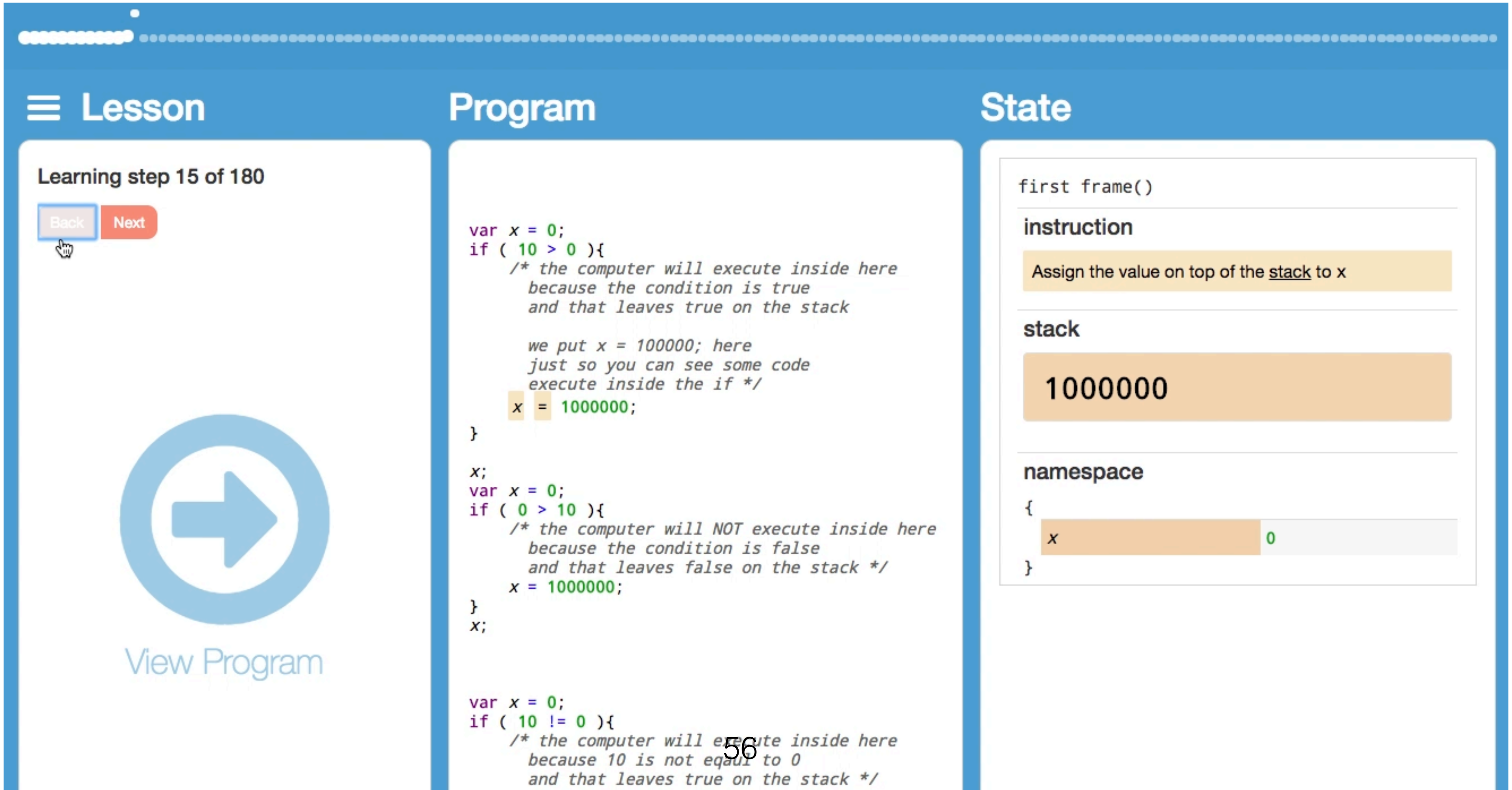


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Reverse execution allows learner to review instruction they didn't understand.

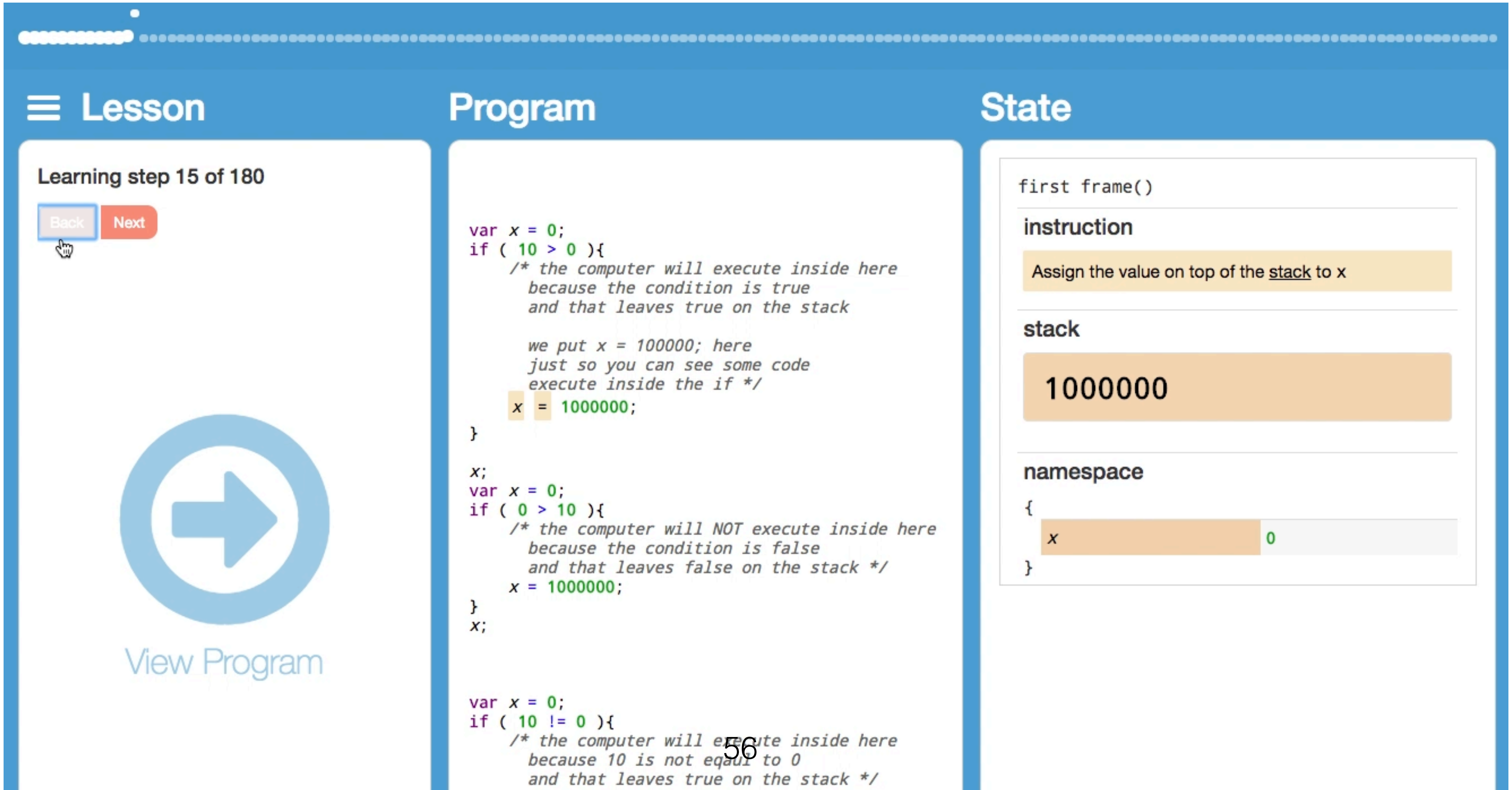


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Reverse execution allows learner to review instruction they didn't understand.

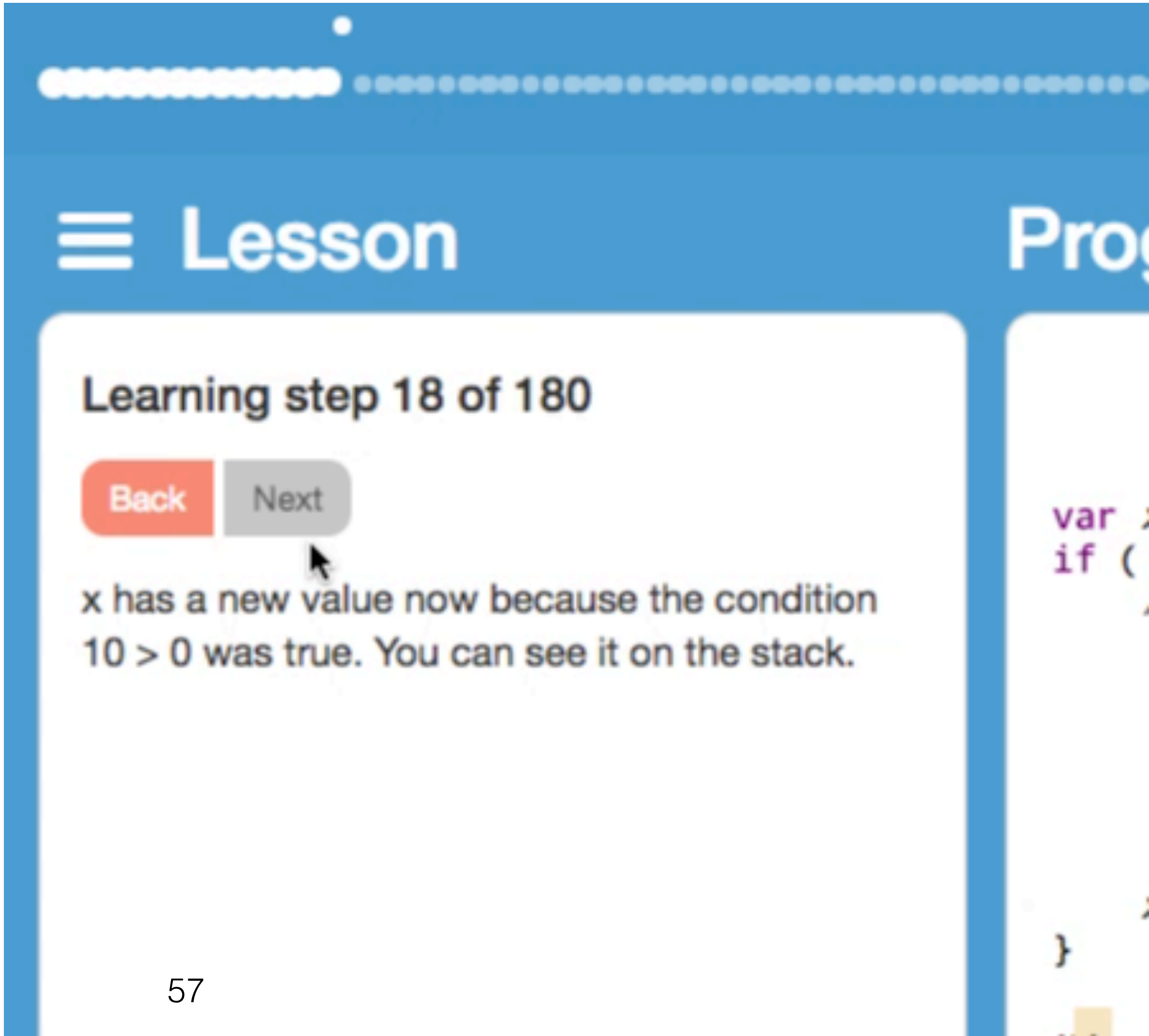


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Lesson explains the side effect of the semantics before proceeding to further examples.

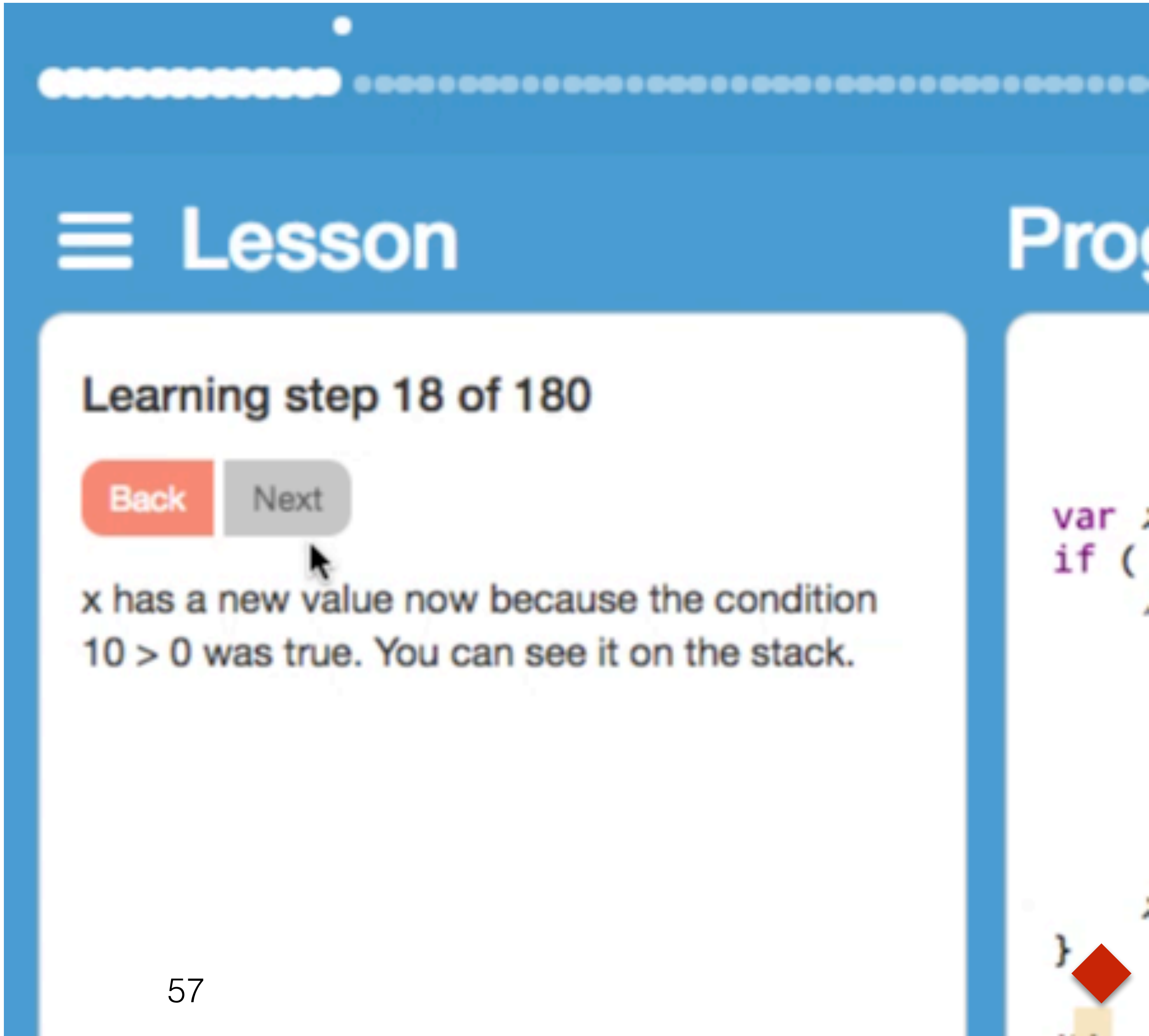


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Lesson explains the side effect of the semantics before proceeding to further examples.

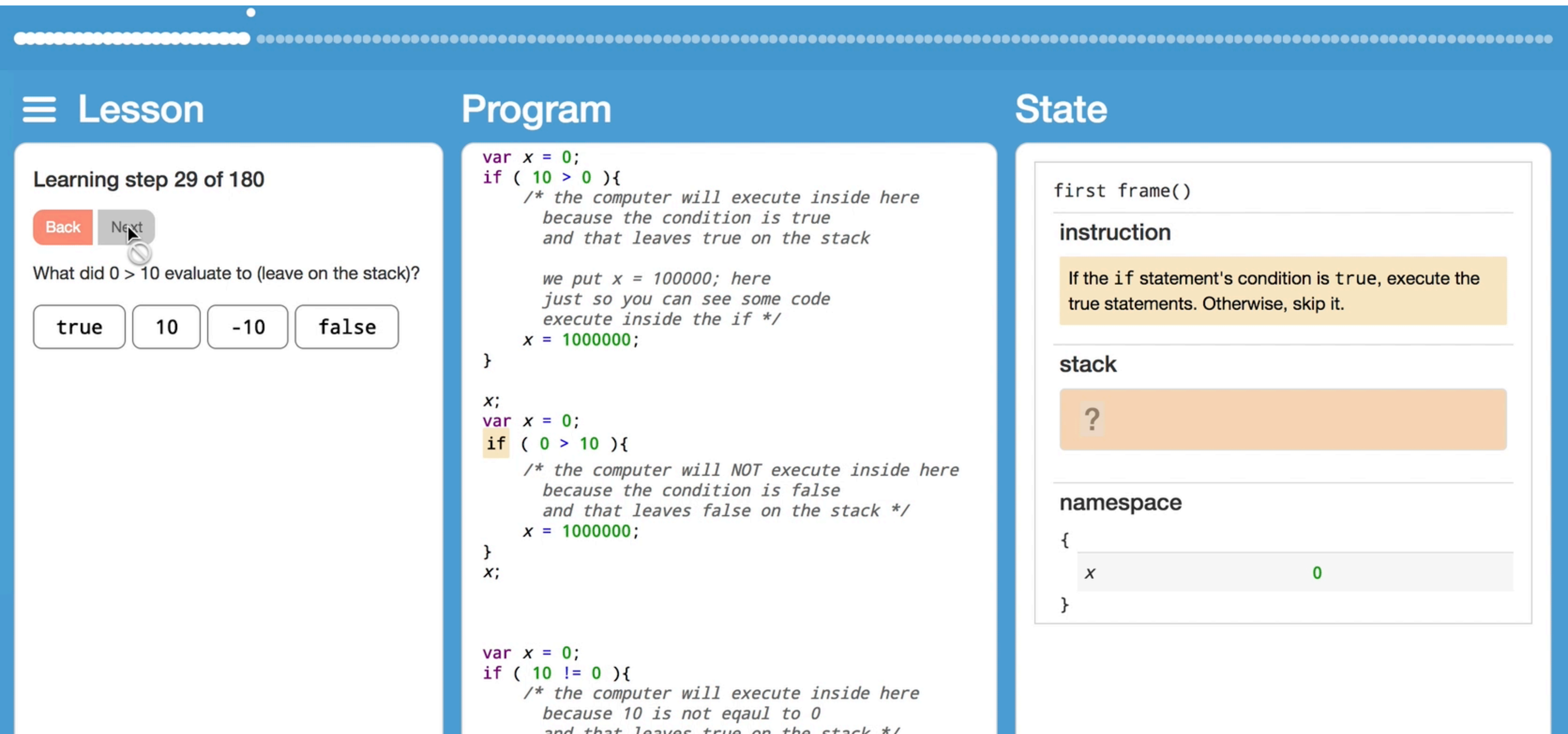


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

Assessments embedded in execution trace require learners to predict side effects of semantics.

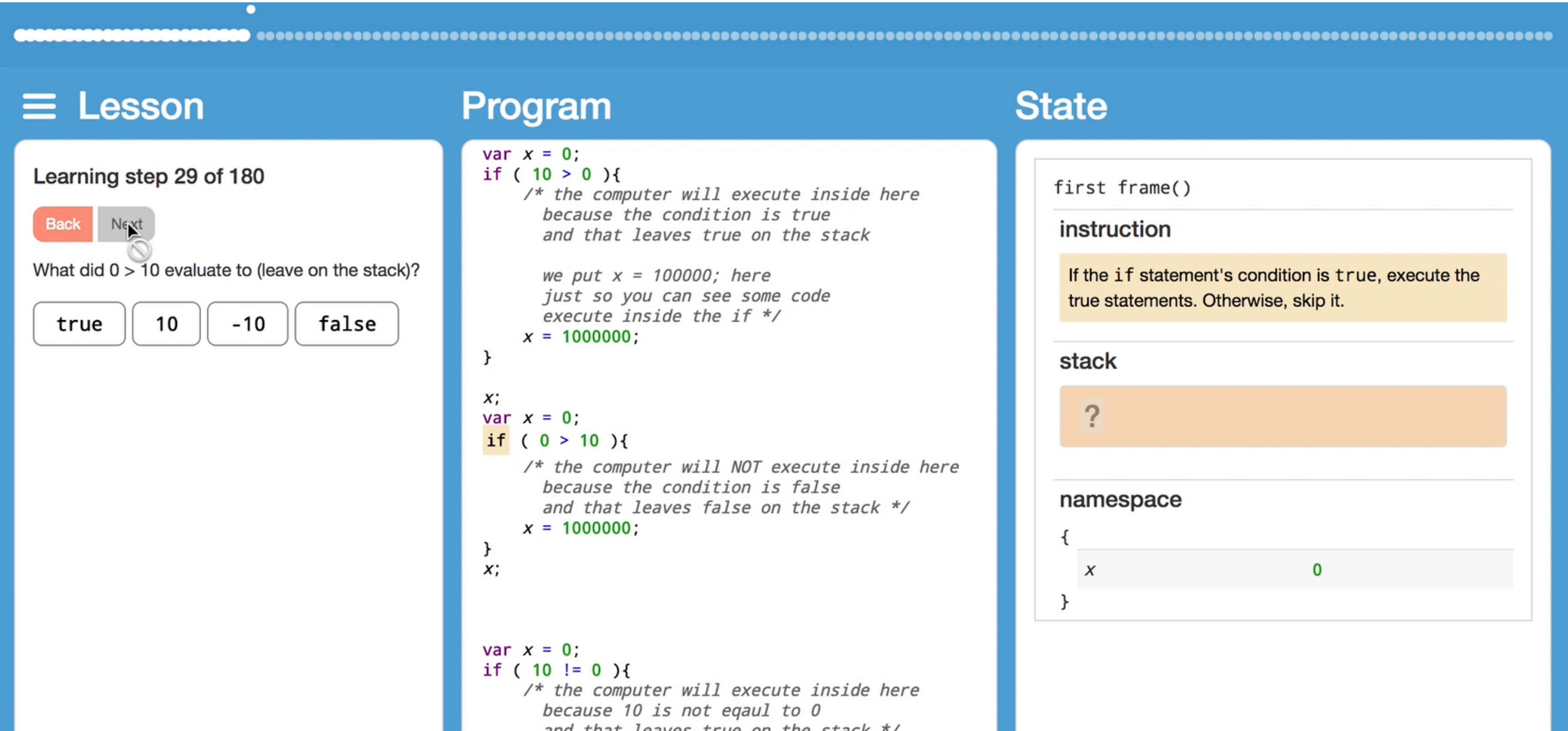


PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

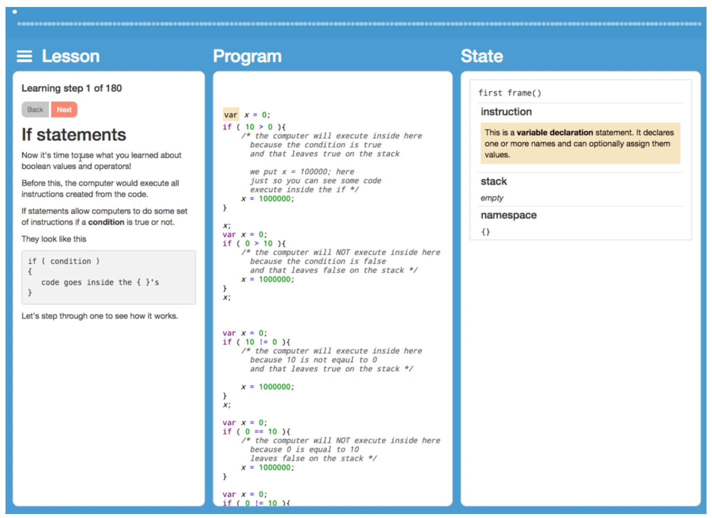
Assessments embedded in execution trace require learners to predict side effects of semantics.



PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

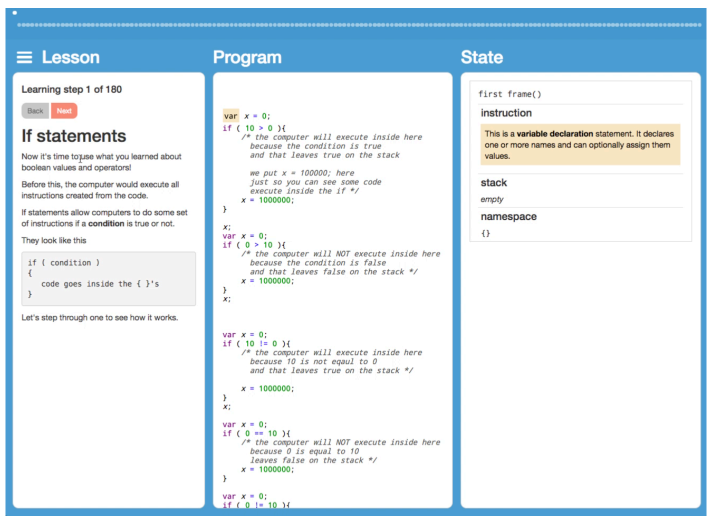


- Required complete re-architecting of language stack
  - Must preserve provenance of all compiler and runtime state to facilitate reversibility and embedded explanations
  - Redesigned grammar to facilitate granular explanations

PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

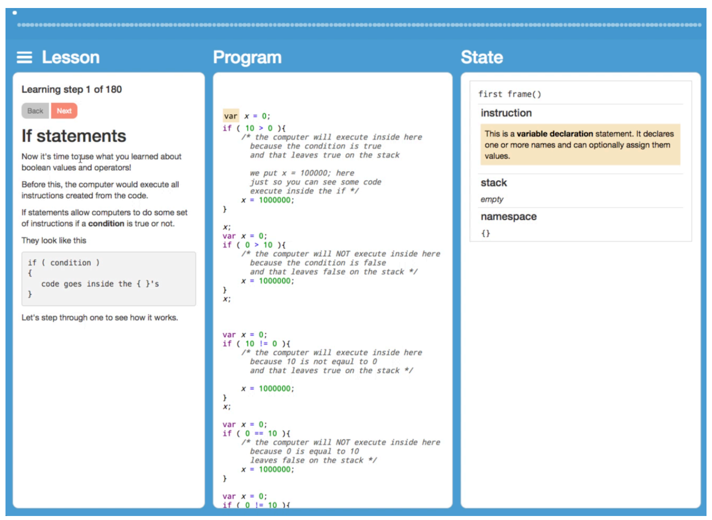


- Required complete re-architecting of language stack
  - Must preserve provenance of all compiler and runtime state to facilitate reversibility and embedded explanations
  - Redesigned grammar to facilitate granular explanations

PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson

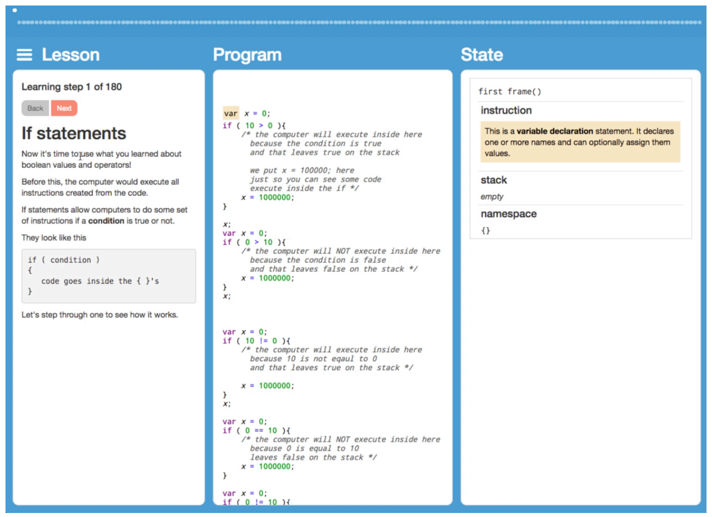


- Compared PLTutor to Codecademy in a 4-hour controlled experiment with **40 CS1** students
- Measured learning with SCS1, a validated assessment of CS1 learning
- PLTutor had **60% higher** learning gains, learning gains predicted midterm scores

PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017). Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.

Greg Nelson



- Compared PLTutor to Codecademy in a 4-hour controlled experiment with **40 CS1** students
- Measured learning with SCS1, a validated assessment of CS1 learning
- PLTutor had **60% higher** learning gains, learning gains predicted midterm scores

Three ideas

Gidget

Mike Lee, Ph.D.

Learners *discover* semantics through debugging

PLTutor

Greg Nelson

Tutor explicitly *teaches* semantics

Tracing Strategies

Benji Xie

Learner reminded to follow semantics

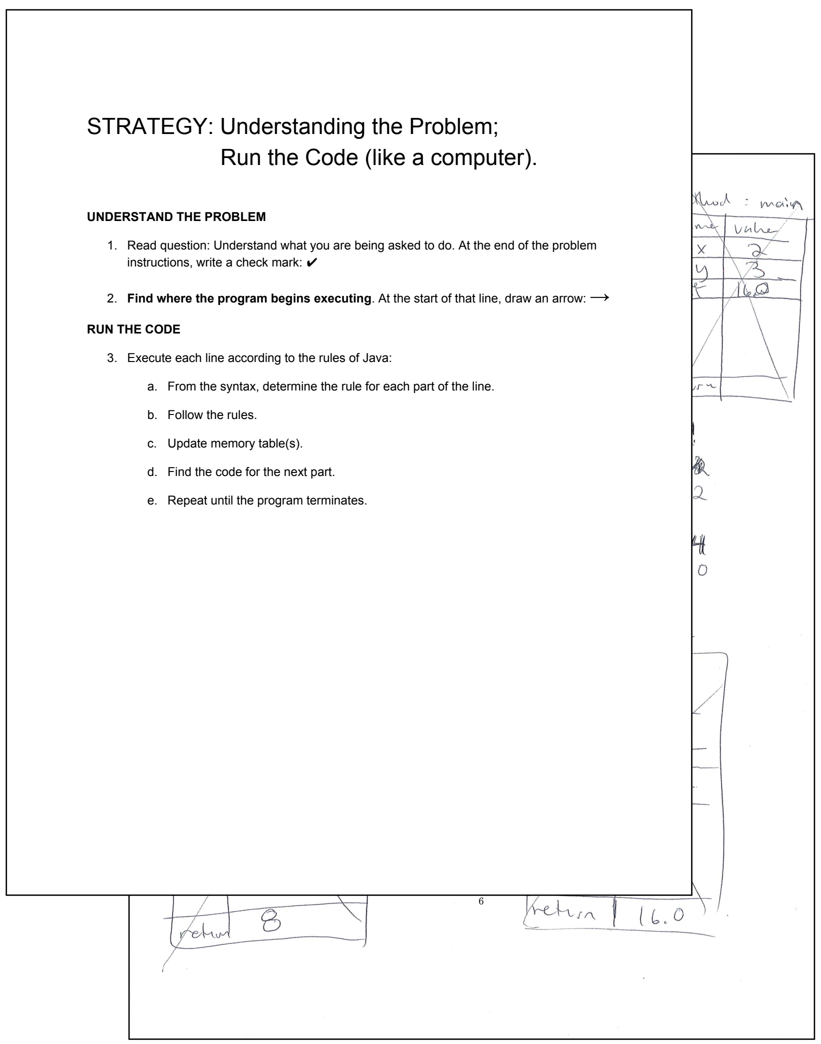
Three ideas

Gidget	PLTutor	Tracing Strategies
		
Mike Lee, Ph.D.	Greg Nelson	Benji Xie
Learners *discover* semantics through debugging	Tutor explicitly teaches semantics	Learner reminded to follow semantics

Strategy

Benjamin Xie, Greg Nelson, and Andrew J. Ko (2017). An Explicit Strategy to Scaffold Novice Program Tracing.SIGCSE.

Benji Xie



STRATEGY: Understanding the Problem;
Run the Code (like a computer).

UNDERSTAND THE PROBLEM

1. Read question: Understand what you are being asked to do. At the end of the problem instructions, write a check mark: ✔
2. **Find where the program begins executing**. At the start of that line, draw an arrow: ⟶

RUN THE CODE

3. Execute each line according to the rules of Java:
   a. From the syntax, determine the rule for each part of the line.
   b. Follow the rules.
   c. Update memory table(s).
   d. Find the code for the next part.
   e. Repeat until the program terminates.

- When learners have brittle knowledge of semantics, they often **guess** how programs will execute
- An explicit strategy for reading programs should outperform guessing

Strategy

Benjamin Xie, Greg Nelson, and Andrew J. Ko (2017). An Explicit Strategy to Scaffold Novice Program Tracing.SIGCSE.

Benji Xie

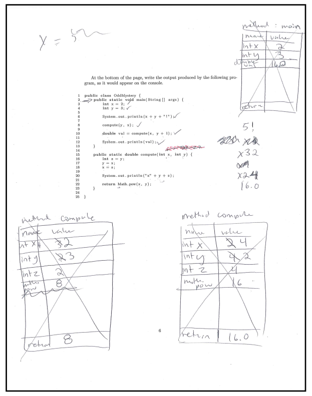
STRATEGY: Understanding the Problem; Run the Code (like a computer).

UNDERSTAND THE PROBLEM

1. Read question: Understand what you are being asked to do. At the end of the problem instructions, write a check mark: ✔
2. **Find where the program begins executing**. At the start of that line, draw an arrow: ⟶

RUN THE CODE

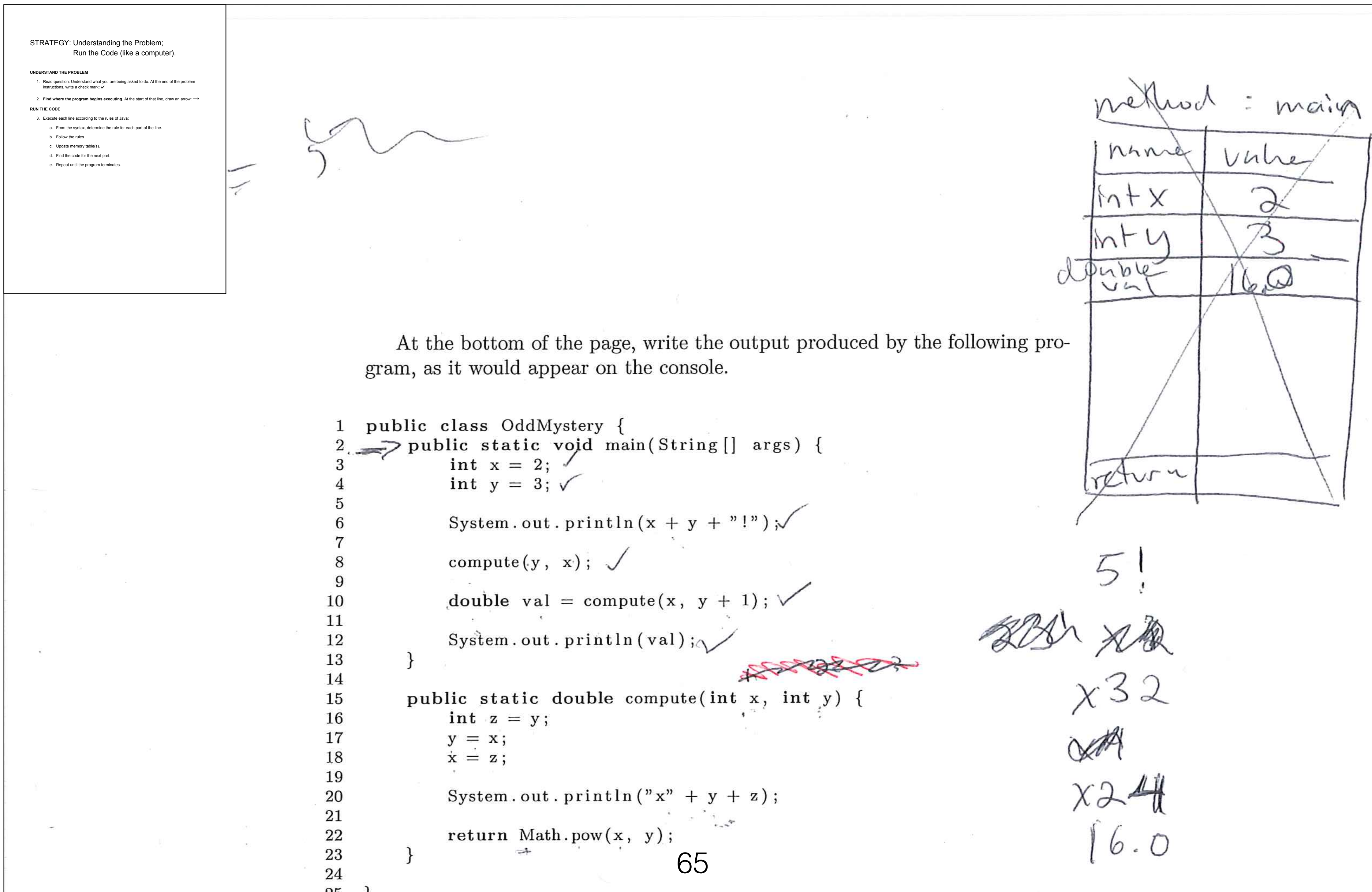
3. Execute each line according to the rules of Java:
    a. From the syntax, determine the rule for each part of the line.
    b. Follow the rules.
    c. Update memory table(s).
    d. Find the code for the next part.
    e. Repeat until the program terminates.



Strategy

Benjamin Xie, Greg Nelson, and Andrew J. Ko (2017). An Explicit Strategy to Scaffold Novice Program Tracing.SIGCSE.

Benji Xie



STRATEGY: Understanding the Problem;
Run the Code (like a computer).

UNDERSTAND THE PROBLEM

1. Read question: Understand what you are being asked to do. At the end of the problem instructions, write a check mark: ✔
2. **Find where the program begins executing**. At the start of that line, draw an arrow: ⟶

RUN THE CODE

3. Execute each line according to the rules of Java:
   a. From the syntax, determine the rule for each part of the line.
   b. Follow the rules.
   c. Update memory table(s).
   d. Find the code for the next part.
   e. Repeat until the program terminates.

At the bottom of the page, write the output produced by the following program, as it would appear on the console.

```
public class OddMystery {
    public static void main(String[] args) {
        int x = 2;
        int y = 3;

        System.out.println(x + y + "!");

        compute(y, x);

        double val = compute(x, y + 1);

        System.out.println(val);
    }

    public static double compute(int x, int y) {
        int z = y;
        y = x;
        x = z;

        System.out.println("x" + y + z);

        return Math.pow(x, y);
    }
}
```

method : main

name	value
int x	2
int y	3
double val	16.0
return	

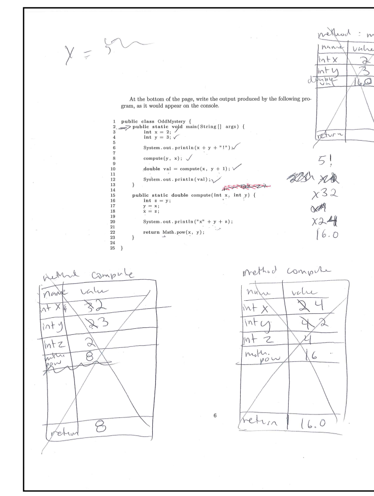
5!
x32
x24
16.0

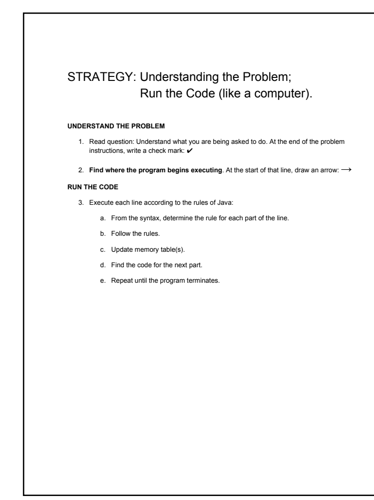
Strategy

Benjamin Xie, Greg Nelson, and Andrew J. Ko (2017). An Explicit Strategy to Scaffold Novice Program Tracing.SIGCSE.

Benji Xie

- In a controlled experiment with **15 minutes** of practice, 12 students who learned the strategy were more systematic than 12 who didn't, resulting in:
  - **15% higher performance** on problems in the lab
  - **7% higher on midterm** that was mostly writing focused
  - **No midterm failures** (compared to 25% failure in control)





STRATEGY: Understanding the Problem;
Run the Code (like a computer).

UNDERSTAND THE PROBLEM

1. Read question: Understand what you are being asked to do. At the end of the problem instructions, write a check mark: ✔
2. **Find where the program begins executing**. At the start of that line, draw an arrow: ⟶

RUN THE CODE

3. Execute each line according to the rules of Java:
   a. From the syntax, determine the rule for each part of the line.
   b. Follow the rules.
   c. Update memory table(s).
   d. Find the code for the next part.
   e. Repeat until the program terminates.

Making programs easier to *read*

Requiring learners to *directly observe* operational semantics and map them to syntax can significantly increase learning outcomes.

Making programs easier to write

One theory, one idea

Program writing

- Little prior work theorizing about what **program writing** skills actually are
    - Most prior work compares to expert and novices, showing that novices are unsystematic, speculative, and ineffective
    - A few papers show that the more developers "**self-regulate**" their problem solving, the more productive they are

Self-regulation

Schraw, Crippen,, & Hartley (2006). Promoting self - regulation in science education. *Research in Science Education* 36, 1-2, 111 - 139.

- From educational psychology, refers to one's ability to reflect on, critique, and control one's thoughts and behaviors during problem solving:
  - Explicit planning skills
  - Explicit monitoring of one's process
  - Explicit monitoring of one's comprehension
  - Reflection on one's cognition
  - Self-explanation of decisions

Great engineers are highly self-regulating

Li, P., Ko, A.J., & Zhu, J. (2015) What Makes a Great Software Engineer? ICSE.

Paul Li

- Interviewed 59 senior developers at Microsoft and surveyed 1,926 about what makes a *great software engineer*:
- Top attributes included:
  - Resourceful
  - Persistent
  - **Self-regulating**

Dastyni's theory of program writing

Dastyni Loksa

- Programming involves iteration through **6 key activities**:
  - Interpreting problems
  - Searching for similar problems
  - Searching for solutions
  - Evaluating solutions
  - Implementing solutions
  - Evaluating implementations

- Programming requires:
  - A **knowledge repository** of problems and solutions (in memory or elsewhere)
  - **Self-regulation skills** to help a programmer:
    - Select strategies for completing activity
    - Deciding when a strategy is failing or successful

Self-regulation is related to success

Loksa, D., Ko, A.J. (2016) . The Role of Self-Regulation in Programming Problem Solving Process and Success. ACM ICER.

Dastyni Loksa

- Observed think aloud of 37 novices in CS1 and CS2 writing solutions to several programming problems.
- Most novices engaged in self-regulation, but infrequently and superficially
- Self-regulation related to fewer errors, but only for novices with sufficient prior knowledge to solve problems

Can we teach it?

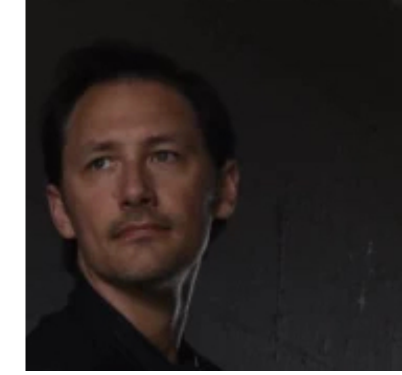
Loksa et al. (2016) . Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. ACM CHI.

Dastyni Loksa

- Taught 48 high schoolers with no prior programming experience HTML, CSS, JavaScript and React for 1 week, then had them build personal web sites for 1 more week
- Treatment group received:
    - Learned Dastyni's theory of program writing
    - Before receiving help, required to practice self-regulation, explaining which activity they were doing, what their strategy was, and whether it was working

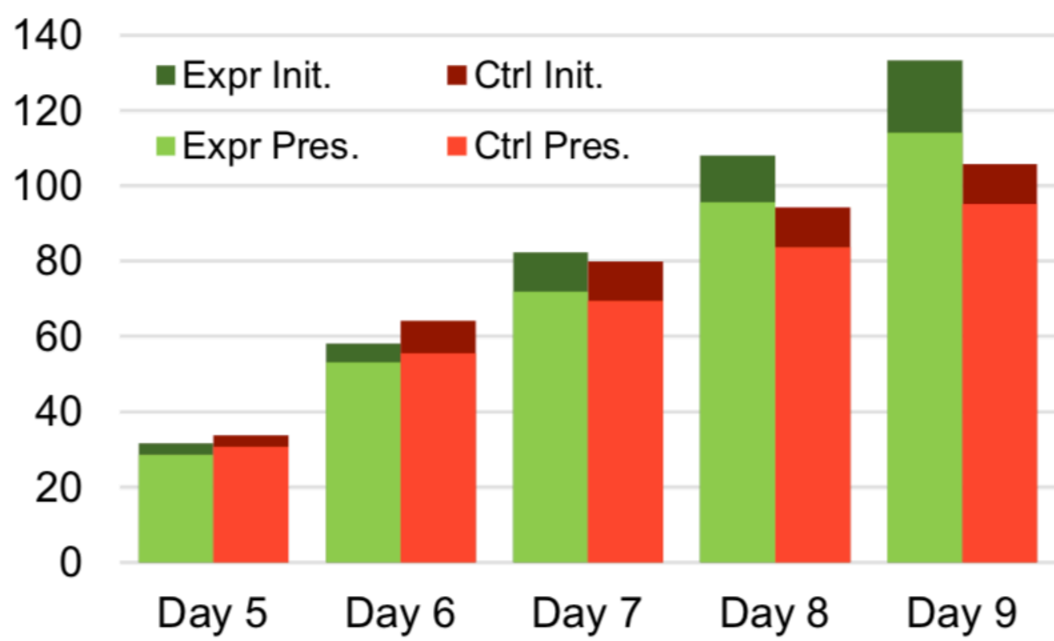
Yes!

Loksa et al. (2016) . Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. ACM CHI.

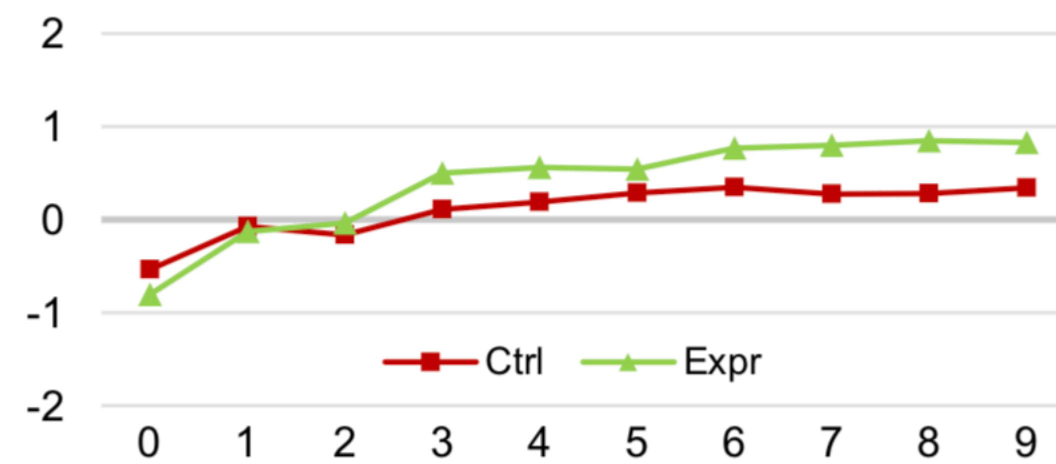


Dastyni Loksa

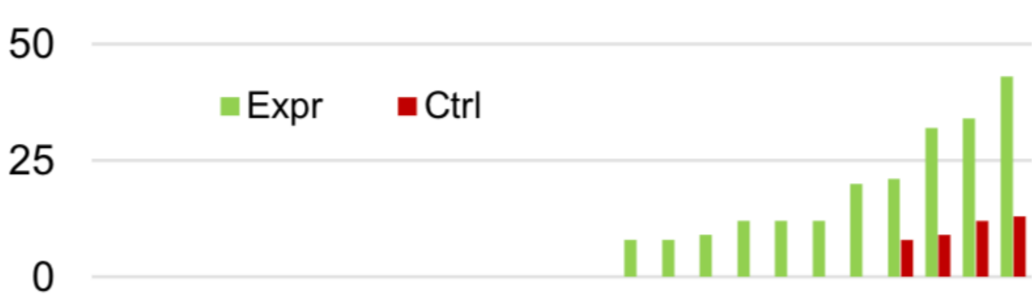
More productive



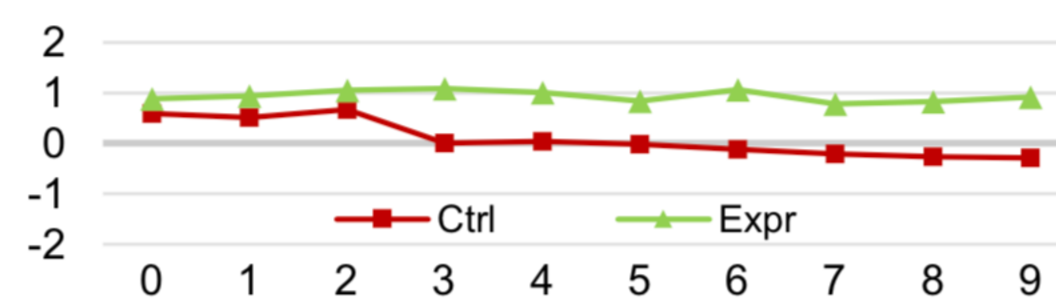
Higher programming self-efficacy



More self-defined work



No growth mindset erosion



Making programs easier to write

Teaching programming self-regulation promotes independence, increased productivity, and higher self-efficacy.

What’s next?

CS1 mastery

New NSF Cyberlearning

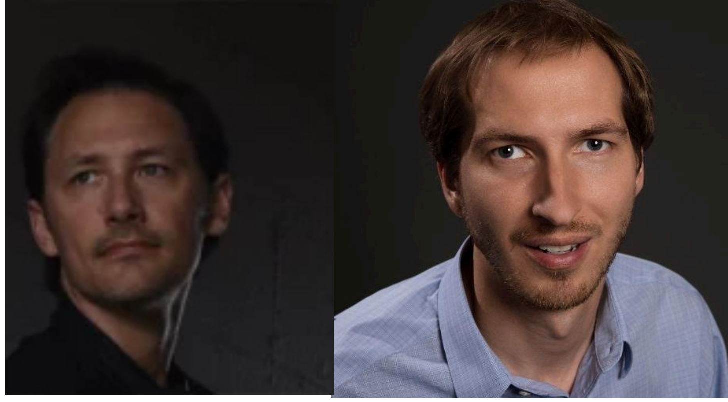
Min Li

Benji Xie

- Prior work shows increased learning, but not *mastery,* which requires **personalized content and feedback**
- Human tutors can provide this, but can't scale it
- We're building a tutor that provides **infinite personalized practice** by applying program synthesis and our theories of programming knowledge
- **Goal**: every student masters CS1 content in 10 hours

Strategies

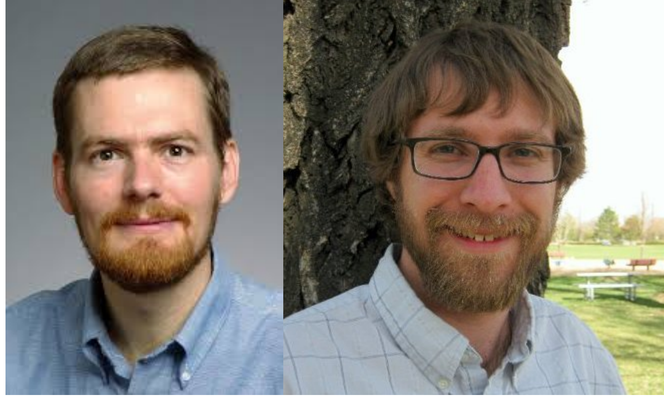
New NSF SHF Medium



Dastyni Loksa Thomas LaToza

- Self-regulation is only useful with good **strategies**
- Defining 1) what programming strategies are, 2) how to describe them, 3) which ones exist, 4) when they're effective, 5) support for learning and executing them.
- **Goal**: A new science of programming strategies analogous to other disciplines' "engineering handbooks," which show how to solve problems in a discipline

Robust API learning



Mike Ernst Kyle Thayer

- New theory of API knowledge as **domain concepts**, **design templates**, and **API execution semantics**
- Techniques to automatically extract this knowledge from API implementations
- Building a tutor that generates on-demand API tutorials using this extracted knowledge.
- **Goal**: rapid, robust API learning at scale

Can’t do it alone…

- Many great faculty contributing to computing education research from PL, Software Engineering, and HCI. **Join us!**

- Our doctoral students need tenure-track positions to continue their work. **Hire them!**

Thanks!

Millions try to learn to code, but fail.

Explicit instruction and feedback on semantics is key.

Learning tech like Gidget and PLTutor are scalable and effective

Pedagogies like tracing strategies and self-regulation prompting are effective and immediately adoptable

Supported by NSF, Google, Microsoft, Adobe, the University of Washington.

Thanks to my wonderful doctoral and undergraduate students, and the hundreds of participants in our studies!