Codechella: Multi-User Program Visualizations for Real-Time Tutoring and Collaborative Learning

Philip J. Guo, Jeffery White, Renan Zanelatto
Department of Computer Science
University of Rochester
Rochester, NY 14627
pg@cs.rochester.edu, {jwhite37,rzanelat}@ur.rochester.edu

***Abstract*—An effective way to learn computer programming is to sit side-by-side in front of the same computer with a tutor or peer, write code together, and then discuss what happens as the code executes. To bring this kind of in-person interaction to an online setting, we have developed Codechella, a multi-user Web-based program visualization system that enables multiple people to collaboratively write code together, explore an automatically-generated visualization of its execution state using multiple mouse cursors, and chat via an embedded text box. In nine months of live deployment on an educational website – www.pythontutor.com – people from 296 cities across 40 countries participated in 299 Codechella sessions for both tutoring and collaborative learning. 57% of sessions connected participants from different cities, and 12% from different countries. Participants actively engaged with the program visualizations while chatting, showed affective exchanges such as encouragement and banter, and indicated signs of learning at the lower three levels of Bloom's taxonomy: remembering, understanding, and applying knowledge.**

Keywords*—*program visualization, collaborative visualization, CS education

I. Introduction

Despite the recent proliferation of free online resources for learning computer programming such as MOOCs, Codecademy [1], and Khan Academy CS [2], decades of computing education research has shown that programming is incredibly hard to learn alone [3]. Learners face fundamental technical challenges such as developing mental models of invisible execution state [4] as well as motivational challenges such as wanting to give up prematurely because they feel like they are somehow "not cut out for coding." [5] Two effective ways to overcome such challenges include one-on-one tutoring from an expert [6] and learning collaboratively with peers [7].

In a face-to-face setting, people often teach and learn programming by sitting in front of a single computer, writing code together (i.e., pair programming), running the code, and verbally discussing what they see on-screen [8]. Since internal execution state is invisible, people also draw *program visualizations* on paper or on the whiteboard to hone their mental models. These visualizations include elements such as stack frames, variables, pointers, and data structures. Researchers have developed many tools to automatically visualize code execution state [9]. They found that an effective use for such tools is to augment in-person tutoring or peer learning sessions so that people can *talk about dynamic properties of execution state* in addition to static properties of code [8]. Such high-fidelity, face-to-face interactions are hard to surpass.

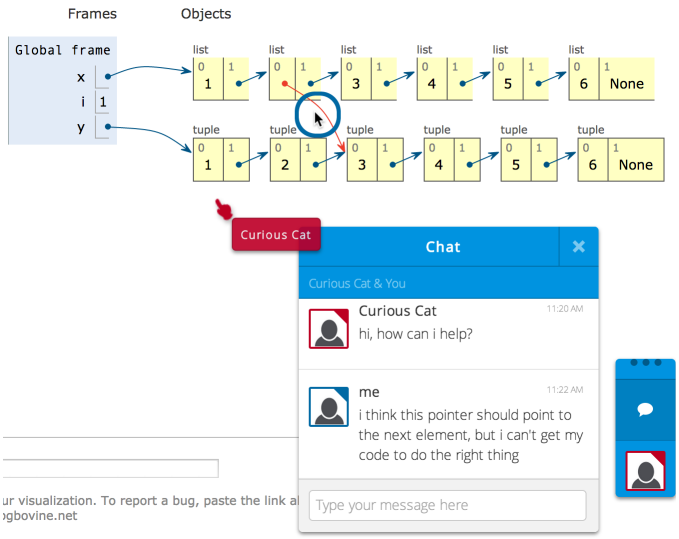

Fig. 1. Codechella is a Web-based system that enables two or more people to collaboratively write code together, explore a detailed visualization of its execution state, and chat via text. All participants within a session see a real-time view of one another's mouse cursors as colored hand icons (e.g., the red hand for the user named "Curious Cat") and mouse clicks as bubbles.

However, many people around the world – especially those taking MOOCs or using other free online resources – do not have access to tutors or peers to help them learn in person, so they are missing out on this valuable pedagogical interaction.

We hypothesize that we can effectively bring this kind of in-person interaction to an online setting by adding real-time synchronized shared sessions to automatically-generated program visualizations. To explore this idea, we developed a prototype called *Codechella*. Figures 1 and 2 show its features:

- Two or more participants join a Codechella session by visiting a unique URL in each of their Web browsers.
- Participants collaboratively write code, run it, and explore a step-by-step visualization of its execution state. All participants see a real-time synchronized view of the code and visualization, as though they are looking at the same screen.
- Participants also see one another's mouse cursors as colored hand icons and mouse clicks as visual bubbles.
- Participants chat about the code and visualizations within an embedded textual chat window.

There are now dozens of program visualization tools for education, but they are all single-user applications [9]. And general-purpose chat applications such as Google Chat or Skype are not integrated into coding or visualization environments. To our knowledge, Codechella is the first multi-user program visualization system, which combines real-time synchronized visualizations with embedded chat to support online tutoring and collaborative learning of computer programming.

We implemented Codechella as a component within Online Python Tutor [10], a popular Web-based program visualization tool (pythontutor.com). Our findings from deploying Codechella for nine months on that website show that people voluntarily used it for both tutoring and collaborative learning in a naturalistic online setting. Specifically, people from 296 cities across 40 countries participated in 299 Codechella sessions. 57% of sessions involved participants from different cities, and 12% from different countries, which shows that Codechella connected people who could otherwise not meet up to work together face-to-face. 69% of logged actions were interactions with the visualization, while 31% were chats, which indicates that participants took advantage of the program visualization features. Participants showed affective exchanges such as encouragement and banter, and indicated signs of learning at the lower three levels of Bloom's taxonomy [11]: remembering, understanding, and applying knowledge.

Codechella's Web-based nature allows it to easily integrate into online educational resources such as MOOCs, digital textbooks, discussion forums, and Q&A sites. It is hard to surpass the fidelity of face-to-face interactions, but Codechella strives to bring some of the benefits of face-to-face learning to those who cannot meet up in-person.

This paper makes three contributions:

- The novel idea of creating a multi-user program visualization system for CS education by combining real-time synchronized visualizations with embedded chat, which enables multiple remote participants to talk about the dynamic execution state of code.
- The design and implementation of Codechella, a prototype Web-based system that implements this idea.
- A case study of 299 sessions during a nine-month-long deployment, which shows high engagement with visualizations and chat in a naturalistic online setting.

II. Related Work

Limitations of online education: One common criticism of online educational resources such as MOOCs is that they mostly serve self-directed autodidacts: people who already know how to learn well by themselves [12]. For instance, many people who successfully complete MOOCs are highly-educated professionals with graduate-level degrees in technical fields [13], [14]. Critics argue that a key missing ingredient is human guidance to sustain motivation and engagement for less independent learners [15]. Discussion forums serve that role in principle, but in practice are dominated by a minority of vocal elite participants, can feel unwelcoming or too public for novices, and lack real-time interactivity [14], [16], [17]. In contrast, Codechella aims to facilitate private, synchronous human-to-human interactions in online educational settings.

Tutoring and collaborative learning: As opposed to the impersonal nature of online educational resources, one-on-one human tutoring is a highly personal and effective way to learn. It is more effective than group-based instruction [18], self-directed learning [15], and computer-based tutoring [6]. Also, collaborative learning can motivate people to learn better than they would alone [7]. Although it is hard to emulate the fidelity of face-to-face interactions, several studies have shown that tutoring and collaborative learning sessions held using text-only chat led to similar learning gains as those using face-to-face interactions [19], [20]. These findings suggest that Codechella's text-based chat interface can potentially bring these personal forms of pedagogy to an online setting.

Collaborative learning interfaces: Codechella's design is inspired by Web-based interfaces for collaborative learning. The most relevant asynchronous interfaces include anchored discussion forums [21], [22] and the NB system for collaborative PDF annotations [23]. The most relevant synchronous interfaces include Talkabout [24], [25] and MOOCchat [26], [27], which coordinate structured small-group discussions in MOOCs over video and text chat, respectively. In addition, Coetzee et al. integrated asynchronous and synchronous interfaces by combining forums with real-time text chat in MOOCs [28]. These chat systems are all general-purpose and not specialized for a particular domain of learning. In contrast, Codechella focuses solely on computer programming by integrating chat with automatically-generated program visualizations.

Program visualization tools: One fundamental challenge in learning programming is developing a viable mental model of code execution [4], [29]. To help students construct mental models, instructors draw diagrams of execution state on the board or in PowerPoint slides [30]. Over the past four decades, computing education researchers have created many *program visualization tools* to automate this drawing procedure. Sorva et al. provide a comprehensive survey of 44 such tools from 1979 to 2013, which he calls "generic program visualization systems." [9] These tools all allow the user to write code in a language such as BASIC, Pascal, C++, Java, or Python, run their code, and then step forward and backward through execution points to see a visual representation of run-time state, including stack frames, pointers, and data structures.

However, to our knowledge, no existing program visualization tool supports multiple remote users connecting to a shared real-time session, which is crucial for tutoring and collaborative learning in situations where people cannot meet face-to-face. Codechella adds a real-time collaborative layer atop traditional single-user program visualization tools.

Studies have found that interacting with visualizations is more pedagogically effective than passively viewing them. Most notably, Hundhausen et al. performed a meta-study of 24 algorithm visualization tools (similar to program visualizations except that the user cannot write arbitrary code) [31] and concluded that these visualizations led to the most learning gains when students actively engaged and interacted with them. Also, Myller et al. showed that these tools are effective for collaborative learning in the classroom where students study visualizations together in front of a single computer [8]. They found that visualizations can be a medium for getting students to collaborate more with one another in class. Codechella strives to bring these benefits to an online setting.

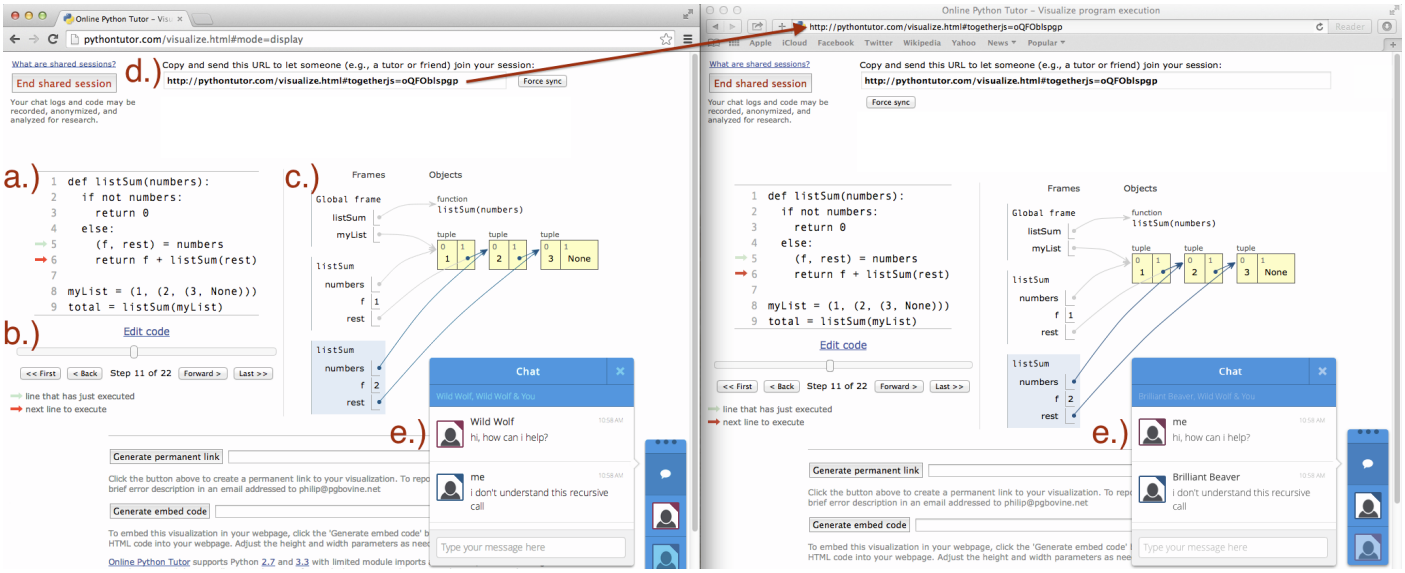

Fig. 2. Overview of our Codechella system, which is built upon the Online Python Tutor program visualization tool [10]. Here is a typical use case: a.) The user writes code in an ordinary Web browser, b.) runs their code and steps forward and backward through execution points, c.) sees a visualization of stack frames, variables, data structures, and pointers at each execution point, d.) clicks the "Start a Codechella session" button and sends a unique URL to a tutor or friend, and then e.) chats with other participants in the Codechella session while navigating the visualization and writing code together in-sync.

III. Formative Observations and Design Goals

To formulate Codechella's design goals, we observed TAs in our university's introductory programming course as they helped students in the computer lab. Nothing can match the fidelity of these in-person interactions, but how can we bring some of its benefits to people who are learning online? Based on our observations, we came up with these design goals:

Chat: In a typical tutoring interaction, the tutor would sit next to the student as they are debugging a piece of code together. The two would continually talk back and forth, with the tutor making suggestions for the student to try. Thus, real-time chat embedded within a code editor is a necessary base component.

Shared Multiple Cursors: The tutor would sometimes point to relevant areas on the student's screen for emphasis, but it was a commonly-accepted best practice *not* to take over the student's keyboard to write code for them. Most of the time, the student would be editing code in a text editor or IDE. The tutor would encourage them to add lots of debugging `print` statements, execute frequently, and then inspect, point to, and discuss the resulting textual output together. Thus, Codechella users must be able to see each other physically point to their shared display.

Synchronized Code Editing and Visualizations: Sometimes the tutor would manually draw diagrams of execution state on a piece of paper to clarify concepts that were not clear from studying the textual output alone. Also, aside from receiving tutoring, students often worked together on programming assignments in the computer lab, since they said that it helped foster a sense of camaraderie. They would often pair-program on a single computer and also draw diagrams on the whiteboard to illustrate algorithms or program execution state. Thus, Codechella users must be able to see and edit a synchronized view of code and visualization diagrams as though they were pair programming on a single computer.

IV. Background: Online Python Tutor

We built Codechella upon an existing Web-based program visualization tool called Online Python Tutor [10]. Figure 2 shows how a user can use Online Python Tutor to write code and visualize its runtime state: The user first visits www.pythontutor.com and writes code directly in their browser (Figure 2a). Despite its legacy name, Online Python Tutor supports coding in five popular languages: Python, Java, JavaScript, TypeScript, and Ruby. When the user presses the "Visualize Execution" button, their code is sent to the Online Python Tutor server to execute in a sandbox. The server sends a complete execution trace back to the user's browser, usually in less than two seconds. The user can step forward *and backward* through all execution points using a navigation slider (Figure 2b). At each execution point, the user sees a detailed visualization of their code's runtime state, which includes stack frames, variables, data structures, and pointers (Figure 2c).

Online Python Tutor can visualize arbitrary heap graphs consisting of custom nested and linked data structures. These automatically-generated visualizations mimic what people normally draw on the board or on paper when explaining code execution state. Over 1.2 million people have used this tool for lecturing, tutoring, debugging programming assignments, creating video tutorials, and self-study to clarify their own understanding [10].

V. The Codechella System

The primary limitation of Online Python Tutor is that it is fundamentally a single-user tool. So far, the main way that people have been using it collaboratively is by sitting together in front a single computer, writing code within it, and discussing the visualizations in person. However, not everyone has direct access to an in-person tutor or peer to help them learn, so we designed a prototype system called Codechella that mimics this sort of human interaction in an online setting.

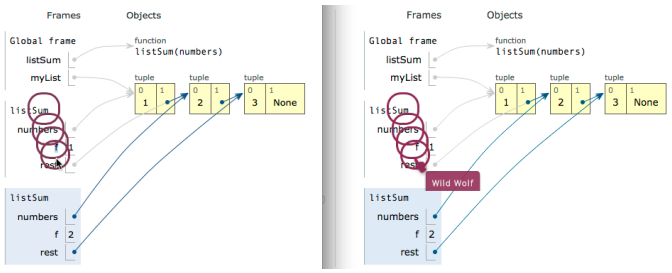

Fig. 3. Codechella displays the positions and clicks of all other participants' mice in real time so that they can see each other pointing to on-screen elements. In the figure above, the left participant moves their mouse downward while clicking four times. The right participant sees that visualized as a downward-moving hand icon that produces four bubbles.

Codechella is embedded within the existing Online Python Tutor website. To activate it, the user:

1) Clicks the "Start a shared session" button at the top of the Online Python Tutor app to start a new session.
2) Copies and sends a unique URL to anybody whom they want to join their session (Figure 2d).
3) Everyone in the session collaboratively writes code and steps through the visualization together while text chatting in an embedded chat box (Figure 2e).

Note that participants still need to know one another to use Codechella; it does not automatically match up participants. People can coordinate when to sign into a session by sending its unique URL via IM, email, or a discussion forum post. Here are the main components of the Codechella system:

Real-Time Collaborative Code Editing: Codechella contains a collaborative text editor, an extension of Ace (http://ace.c9.io/) that works similar to Google Docs with added syntax highlighting or collaborative coding IDEs such as Collabode [32]. This base layer enables multiple people to edit code together as though they were sitting in front of a single computer.

Embedded Text Chat: We chose a text chat format rather than audio or video both due to its relative simplicity, and also to the observation that coding-related terms and external resource URLs (e.g., documentation websites) are far easier to communicate via text than verbally. The chat widget is a pop-up box at the bottom edge of the webpage (Figure 2e).

Shared Program Visualizations: When any user presses the "Visualize Execution" button, Codechella executes the currently-displayed code and shows an interactive visualization of its runtime state using the underlying Online Python Tutor visualization engine (Figure 2c).

All participants in a Codechella session always see the same synchronized view of the code and visualization. This means that if one participant clicks the "Visualize Execution" button, then everyone sees the same visualization. And if someone drags the slider to advance to a different execution point (Figure 2b), Codechella *automatically drags the slider* on everyone else's Web browser and updates their visualization to match that view. If a participant toggles any options such as changing the current programming language, then the respective option elements on all other participants' browsers are toggled as well. And if a participant clicks the "Edit code" button to enter the code editor, all other participants also get switched over to the code editor. Again, this behavior emulates what would happen if multiple people were physically sitting in front of the same computer.

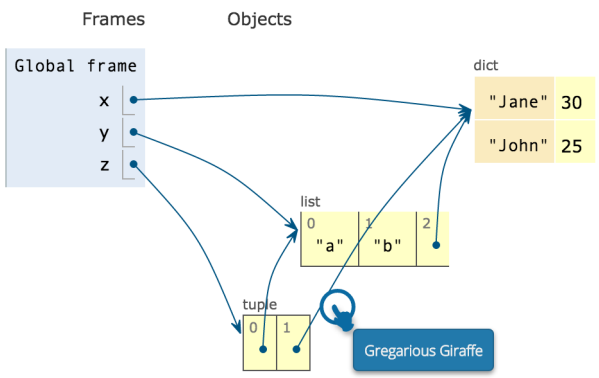

Fig. 4. Codechella enables remote users to point to and discuss complex runtime state such as the following diagram of three global variables, with `z` set to a Python tuple with elements pointing to a list and a dictionary.

Figure 4 shows a detailed view of a participant (with username "Gregarious Giraffe") hovering over and clicking on a specific pointer (arrow) in a complex nested Python data structure. By tightly integrating into a program visualization tool, Codechella enables its participants to talk about dynamic properties of execution state (e.g., "why is this pointer now referring to that element in the dictionary?") rather than simply discussing static properties of the code itself. Without automatically-generated visualizations, people would need to manually sketch these sorts of diagrams using a shared drawing canvas application, which is tedious and error-prone.

Resolution-Independent Multiple Cursors: Studies of computer-supported collaborative learning in the classroom show that seeing each other's mouse cursors in real-time can improve engagement and learning outcomes [33]. When multiple people sit around a single computer screen in real life, they can physically point at the screen. To mimic this interaction in a remote setting, Codechella shows the mouse cursor positions of all other participants as colored hand icons, labeled with their respective usernames. When a participant clicks their mouse, a colored bubble emanates outward from the click position, gradually expanding for two seconds before disappearing. For instance, Figure 3 shows the left participant moving their mouse downward while clicking four times. The right participant sees those clicks visualized as bubbles, along with the left participant's current cursor position. Also, Figure 4 shows the "Gregarious Giraffe" user pointing to and then clicking on a particular arrow in the visualization. Note that every participant still controls their own mouse cursor; they simply see all other participants' cursors.

The hand cursor displays are positioned relative to individual DOM (Document Object Model) elements on the web page, so they are *resolution-independent*. Thus, if a participant clicks, say, a specific linked list element in the visualization, their cursor position and bubbles will show up on top of that exact same linked list element on all other participants' screens, regardless of their current window resolutions. This feature turns a mouse click into a precise way to focus attention

on a particular portion of the code or visualization. Resolution independence is crucial since different users will most likely have their Web browsers set to different resolutions, so synchronizing cursors using absolute x-y positions will not work.

Implementation: We implemented Codechella by modifying Mozilla's TogetherJS library (http://togetherjs.com) and integrating it into the Online Python Tutor codebase. TogetherJS provides JavaScript code for synchronizing multiple users' Web browsing sessions in real time using WebSockets. When a Codechella user performs an action such as sending a chat message or stepping through the visualization, that action gets sent to the WebSocket-enabled Node.js server, which immediately broadcasts it to all other users in the same session. Latency is bound by each user's Internet connection speed, but there is little additional overhead since each action sends minimal amounts of data. All modern Web browsers support WebSockets, so Codechella does not require any plugins.

We designed Codechella for one-on-one and small-group interactions, which mimics a small number of people sitting in front of a single computer in a lab. It has several benefits over screensharing software: 1.) Every user has their own window that they can move and resize to their own screen resolution rather than forcing all users to see one fixed-size display. This independence gives users a greater sense of control and privacy, since only one browser tab is being shared. 2.) Multiple synchronized cursors enables each user to point to different on-screen components. 3.) Encapsulating a shared session within a URL makes it easy to embed our system within a MOOC or online tutor-matching pool (Section VII); it would be logistically harder for novice learners to use desktop screensharing software that requires installation and setup.

VI. Online Deployment Case Study

To demonstrate the efficacy of our Codechella prototype, we deployed it as a live feature on the Online Python Tutor website (www.pythontutor.com) in May 2014 and logged all usage data on our server. We analyzed nine months of server logs to explore the following research questions:

- Is Codechella serving people around the world who are unable to get together to learn face-to-face?
- How do people use Codechella for tutoring and collaborative learning?
- How much do people interact with the visualizations rather than simply using it as a coding and chat tool?
- What kinds of knowledge do people appear to learn during Codechella sessions?

Although this kind of server log analysis lacks the controlled conditions of a user study, it provides naturalistic usage data at a scale that we cannot feasibly replicate in our lab. Thus, the findings we present here represent a case study of how people actually used Codechella in the wild, and are not meant to test specific hypotheses about usability or learning.

A. Codechella Usage Overview

During the nine-month period in our log data between May 29, 2014 and March 2, 2015, Codechella users participated in 384 sessions. We define a *session* as containing at least 2 participants and at least 10 chat messages. This heuristic eliminates thousands of instances of people simply testing out or demoing the system without intending to learn using it.

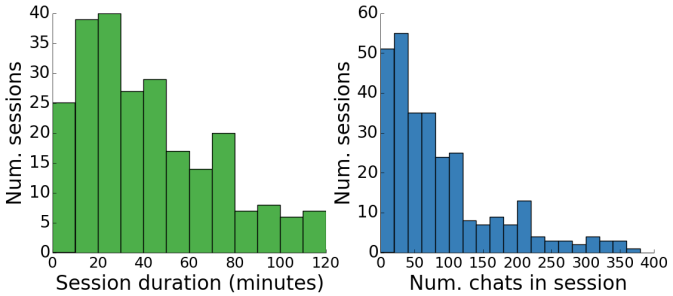

Fig. 5. Number of minutes (mean=58, median=42) and number of chat messages (mean=99, median=68) in each of the 299 Codechella sessions.

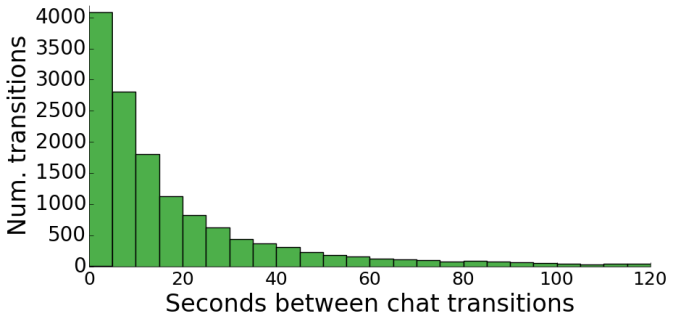

Fig. 6. Seconds between chat transitions (mean=84s, median=10s). A *transition* occurs between two consecutive chats sent by different participants, so it shows how long someone waits before receiving a response to their chat.

English was the most common language used in chats (60% of sessions), followed by Portuguese (11%) and Spanish (5%). Since members of our research team knew all three of these languages, we could understand the contents of those chat logs. For the remainder, we used Google Translate to automatically translate their chats into English so that we could understand them. 85 sessions either could not be translated or contained only chats about non-programming topics or spam, so we filtered those out as well. In the end, we were left with 299 sessions held in 27 different (translatable) languages. Our data analyses in this paper consider only these 299 sessions. Also, all three researchers read over the chat logs together to code for qualitative features such as indicators of learning.

Participants logged on from 296 different cities in 40 different countries, as approximated by IP address geolocation. Most sessions (66%) had two participants, so they were one-on-one interactions; the mean number of participants was 3.

There was a large variation in how long each session lasted, ranging from a few minutes up to two hours. The left half of Figure 5 shows that most sessions lasted under an hour, with a median duration of 42 minutes; the right half shows the number of chat messages exchanged, with a median of 68.

Figure 6 visualizes wait times between chats sent by different participants and shows that they often responded to one another within 10 seconds. This median response time is lower than typical response times of 30 seconds to 2 minutes reported by studies of general instant messaging (IM) interactions [34], which indicates that participants were highly responsive to one

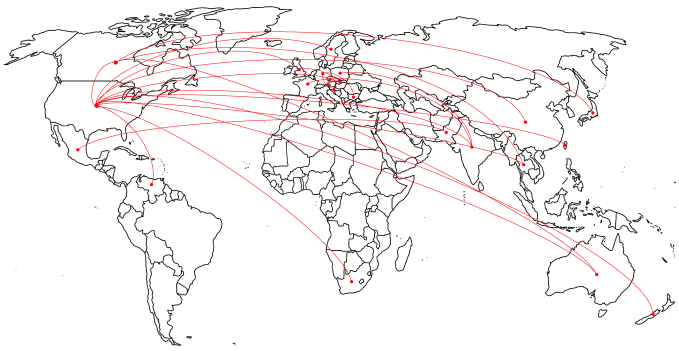

Fig. 7. A line connects two countries if at least one Codechella session involved participants from both of those countries.

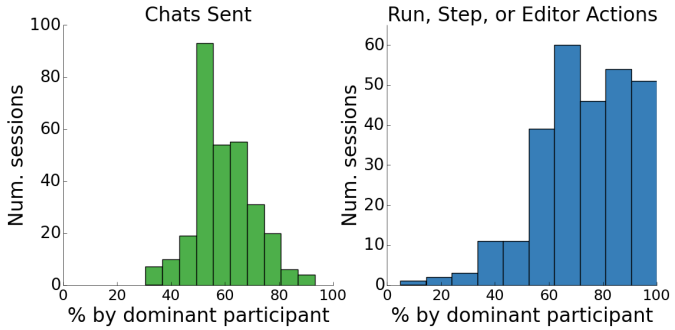

Fig. 8. Each participant sent around half of the chats in a Codechella session, with a median of 57% of chats sent by the dominant participant (left graph). In contrast, one participant usually performed the majority of non-chat actions (run code, step, editor), with a median of 75% by the dominant participant (right graph). Distributions differ with $p < .001$ in a Mann-Whitney U test.

another during Codechella sessions.

B. How Codechella Serves Geographically-Separated Users

Although Codechella can be a convenient alternative to meeting in person, it cannot surpass the fidelity of face-to-face interactions. Thus, a more compelling use case for Codechella is connecting people who cannot possibly meet face-to-face. How often did that occur in our live deployment? To find out, we used the MaxMind geolocation tool [35] to find each participant's geographical location using their IP address.

57% of sessions (171 out of 299) had participants who came from different cities, and 12% (36 out of 299) had participants from different countries. Since these participants did not live in the same place, they could not meet in person to work together. Figure 7 visualizes the international sessions, with a link between two countries if at least one session contained participants from both of those countries. We do not have data on how these participants found one another to start a shared Codechella session, but they could have been fellow students in a MOOC or other online course.

The majority of sessions (57%) involved participants who were geographically separated. But when we read all of the chat logs, we saw that in 35% of sessions (106 out of 299), participants mentioned references to *shared real-life context*. It was common for friends to suddenly start talking about their personal lives in the middle of working on code together. For instance, one pair said: "*A: I will give you more detail about life when you have more awake time. / B: Any decision on Micah's wedding party? / A: I don't think we can go. / B: No? You going to Switzerland?*" These anecdotes indicate that Codechella can be useful even for people who know each other in real life. It can be more convenient to sync up online than to coordinate a time and place to meet in person.

C. Participant Interactions with Program Visualizations

Do Codechella participants interact with the automatically-generated program visualizations? Or are they simply using it as a collaborative text editing and chat tool, in which case it would be no more useful than, say, Google Docs?

To find out, we counted the types of actions present in the server logs. There were 96,866 total actions logged across 299 sessions (324 actions per session, on average). There were four types of actions, each initiated by one participant:

- **Run** – Running (executing) the currently-displayed code and producing a visualization of its execution state. 6% of total actions were of this type.
- **Step** – Stepping to a different execution point in the code visualization, by either pressing the "Previous" or "Next" buttons or by dragging the navigation slider (Figure 2b). 57% of actions were of this type.
- **Open Editor** – Opening the code editor by pressing the "Edit code" button. Note that the server does not log when someone is typing code in the editor itself, only when someone switches from the visualization to the code editor. 6% of actions were of this type.
- **Chat** – A single chat message sent by a participant in the chat box. 31% of actions were of this type.

The majority (69%) of actions involved engaging with the program visualization tool while only 31% were chats, which indicates high engagement with the visualization. Most actions (57%) were of participants stepping to view data structures at different execution points.

In 47% of sessions, participants directly referred to the visualization in their chat messages, which is another indicator of engagement. For instance, one wrote: "*ok, list is the variable declared in line 1 and you see the content of the list in the yellow boxes with the number of their position 0, 1, 2 or 3. this will prove to be very useful when you will have a little more complicated code as you will actually see what is going on in your script.*" Another wrote: "*so if you go back a step ... horse = mask ... that reassigns horse to the lambda function.*"

How did participants balance chatting and interacting with the visualization? Figure 8 shows how participants sent roughly equal numbers of chats in a session (most sessions had two participants), but usually one dominant participant *drove* the interface by interacting most with the visualization (run code, step, and editor actions). This observation is consistent with in-person pair programming and tutoring sessions where one person controls the keyboard and mouse while chatting with their partner. Even though one person is mostly interacting with the visualization, all participants see the same real-time synchronized view.

D. Tutoring and Collaborative Learning Sessions

Even though we did not advertise any specific use case for Codechella, we found from reading chat transcripts that sessions naturally fell into one of two categories:

- **Tutoring** – A one-on-one interaction where one person was clearly in the role of an expert tutor, and the other was the tutee. The tutor either explained a programming concept or helped the tutee debug a specific problem in their code. These comprised 40% of sessions (120 out of 299).
- **Collaborative Learning** – When two or more peers work together on a programming assignment or debugging code, without one person being the clear expert. These comprised 60% of sessions (179 out of 299).

Here is one typical tutor-tutee exchange consisting of a question followed by an explanation:

Tutor: one thing that I think is very helpful is to write "pseudocode" - have you heard of that before?

Tutee: In the discussion space on edx, people wrote this whole code in 14 lines. No I don't know pseudocode

Tutor: Pseudocode is simply writing out the instructions in English, but step by step so that they can be translated easily into code.

Another type of tutoring interaction is the tutor inspecting the tutee's code and asking them specific questions about it. For instance: "*you're passing in a bunch of arguments to the init - do you know the significance of those?*"

From reading tutee reactions, sessions where tutors used Socratic questioning techniques appeared effective. Here is an example series of questions that a tutor asked, which led the tutee to hone in on and fix the bug in their own code (tutee answers omitted for space): "*our program crashed. why do you think it crashed? / what structure is it going to be? / ok how about sorted(t[2]) / how about t[1]? / whats the type? / its going to be an integer / whats the result of adding an integer and a list? / yeah. lets fix the code. how would we fix it?*"

Another effective tutoring technique was for the tutor to prompt the tutee to describe their mental model of how the code works, and then diagnose misconceptions. For example, one session started with the tutor saying: "*well, rather than just trying a bunch of things, let's figure out what you really want to do. talk me through how this function works.*"

One more salient feature of good tutoring sessions was the tutor displaying encouragement, especially when the tutee appeared frustrated. For example:

Tutee: oh my god im so terrible im sorry,

Tutor: not a problem at all. you have writen [sic] little code yet. it will take a while for it to become more comfertable [sic]

In contrast, during seemingly ineffective sessions, the tutor would take over the conversation without asking questions. For instance, one tutor curtly led with, "*i'm going to step through the code, pay attention.*" In those sessions, the tutee passively responded with cursory nods such as "*ok*" or "*sure*" without any indication of engagement. This behavior reminded us of an overbearing in-person tutor taking over the tutee's keyboard and telling them to just watch.

In collaborative learning sessions, participants were usually unsure of their assumptions, so they stepped through the visualization and asked one another questions to figure out the code together. For example:

A: oh i guess basestring is checking if its a string?

B: you can step forward and backward if u want. perhaps?

A: is a string automatically a list in python?

B: lemme see

A: no wonder the stack was weird. i guess it calls printitems each time we have a string

Another salient feature of collaborative learning sessions was friendly banter and cursing. Reading those chat transcripts felt like watching friends working on a programming assignment together while teasing one another. While cursing in tutoring sessions appeared sparingly as a sign of the tutee's frustration, cursing was more liberally used throughout collaborative learning sessions. Taken out of context, such cursing might appear rude, but within those conversations, they felt like a natural and humanizing part of the collaborative learning process. Here is a mild example:

A: I don't feel so bad not getting this now. I thought it was right in front of me and I was being stupid

B: here's an alternative ugly ass function

A: hahaha

B: so basically you first find bob

A: i have the old counting bob function saved so hold on

B: if bob is there it's like kewl. oh you need s

A: i got it. you beautiful bastard

And another one where two participants excitedly stepped through the visualization: "*ok so we're goin into our main loop / rad as sh*t u can see the value of i there it is 0 / amazing / and u can see that guess_clue is an empty motherf**ker*"

Banter and cursing acted as playful encouragement in collaborative learning sessions. Similar lab studies have shown that banter and cursing amongst peers can improve rapport and lead to significant learning gains [36].

E. Indicators of Learning

What did participants learn in Codechella sessions? To address this question, we read all of the chat transcripts and manually coded for *indicators* of learning. However, we cannot make any claims about true learning since we did not run any formal assessments to measure gains, depth, or retention.

We observed both declarative ("what-is") and procedural ("how-to") knowledge being exchanged in Codechella sessions. Collaborative learning sessions focused on procedural knowledge since participants were usually trying to debug or figure out how to solve a class programming assignment together rather than learning brand new concepts. Tutoring sessions contained a mix of declarative and procedural knowledge.

All indicators of learning were at the lower three levels of Bloom's taxonomy: remembering, understanding, and applying knowledge [11]. This finding was unsurprising because we

did not expect these informal chat sessions to result in more substantive learning at the higher levels of Bloom's taxonomy: analyzing, evaluating, and creating.

Some indicators of learning involved *remembering* specifics about Python language terminology. For instance:

Tutor: So, are you familiar with what a "return" statement does?

Tutee: don't you have to do that so you can print it [?]

Tutor: Right, so if you want to assign the output of a function to a varaible [sic], you have to return something inside the definition of that function

Some indicators were at the next level of Bloom's taxonomy: *understanding* how each Python language concept works, usually by *exemplifying* with a specific code example, and also by *comparing* to similar concepts in other programming languages. For instance, when trying to figure out pointers in a collaborative learning session, one participant said: "*reference is deleted, object stays because another reference is still in place, then a new reference (using the same name "b") is put back in place.*" The other participant responded, "*basically python author implemented pointers by default, referencing and dereferencing, took away the complexity for majority of the programmers.*" The participants then compared Python pointers to those in other languages they used in the past such as C++, Java, Objective-C, and PHP.

Finally, in some tutoring sessions, we saw tutees *applying* their newly-learned knowledge to solve a new problem. A good tutor would explain a concept and then get the tutee to apply it in a novel way in the context of their own code. We did not observe any knowledge application in collaborative learning sessions, which were much more focused on debugging a specific piece of code rather than generalizing to other examples.

F. Limitations

These findings come from a nine-month-long online deployment, but we have not yet conducted controlled experiments to formally evaluate the usability of Codechella or how it compares to alternative interfaces. We have not collected any demographic information from Codechella users, so we do not know whether they are a representative sample from a typical online learning community such as a MOOC. We have also not directly compared Codechella to in-person learning. Finally, we can only point out indicators of learning in chat transcripts but cannot make definitive claims about actual learning since we did not personally interview or test the participants.

VII. Future Work

A potential Codechella user must now personally know a tutor or peer who is willing to join a session with them. Then they must coordinate joining using external means such as sending a session URL via email or IM. However, many people do not have access to someone who can tutor or work with them on-demand, so they cannot use this tool.

Since Online Python Tutor is a popular website for learning Python, there are around 60 users concurrently on the website at any given moment. Thus, we want to create a *learner matching interface* hosted on that website where strangers can start Codechella sessions with one another based on their learning needs. The main challenges here are determining appropriate matches based on user skill levels and interests, dealing with potential language barriers due to the international user base (Online Python Tutor users come from over 165 countries), and maintaining high-quality conversations when participants are now strangers instead of personal acquaintances. If successful, this interface could bring the benefits of one-on-one tutoring and peer learning to a much wider population of learners, especially self-directed learners who do not have immediate access to formal schooling or tutoring resources.

Second, people do not use Codechella in isolation. From reading chat logs, we saw that many participants started a Codechella session while they were in the middle of debugging a piece of code written elsewhere, working on a class programming assignment, or setting up a tutoring session via, say, email or instant messenger. Thus, future systems of this sort should not exist as standalone services, but rather be tightly integrated into relevant external applications.

One natural integration is with a computer programming MOOC. A learner could launch a Codechella session when watching lecture videos, posting on the discussion forum, or attempting to solve programming assignments in the Web-based code editor. A potentially powerful benefit of MOOC integration is that the MOOC provider's server keeps a detailed learning profile for each learner, which includes their engagement with content and grades on assignments. It can use this information for more effective skill level matching to find tutors or to form ad-hoc Codechella study groups.

Another potential integration is with production-grade messaging apps such as Skype or Google+ Hangouts. Video chat and drawing on a shared virtual whiteboard could augment Codechella's current text- and visualization-based interface.

More broadly, future Codechella-like tools can be directly integrated into an IDE so that users can leverage collaborative program visualizations to debug production-scale software. Here the main challenge is scaling up the visualizations to support larger pieces of code. Using such an interface, a software engineer can solicit real-time live help from experts or peers on the Web as they are coding in their IDE.

VIII. Conclusion

We have presented Codechella, which is, to our knowledge, the first multi-user program visualization system. Codechella combines real-time synchronized visualizations with embedded chat to emulate the in-person experience of learning programming together. A case study of 299 Codechella sessions initiated by anonymous users on the Online Python Tutor website [10] indicates that people from 296 cities across 40 countries: a.) did both tutoring and collaborative learning with geographically-separated partners, b.) interacted heavily with the visualizations while chatting, and c.) appeared to have gained new knowledge about basic programming. We envision a future where these kinds of tools help humanize online education by connecting learners from around the world.

Acknowledgments

Thanks to Carrie Cai, Logan Gittelson, and Juho Kim for their feedback. This work was supported in part by the National Science Foundation under grant NSF CRII IIS-1463864.

REFERENCES

[1] "Codecademy: Learn to code," http://www.codecademy.com/, accessed: April 2015.

[2] "Khan Academy: Computer programming," https://www.khanacademy.org/computing/computer-programming, accessed: April 2015.

[3] M. Guzdial, "Limitations of MOOCs for Computing Education - addressing our needs: MOOCs and technology to advance learning and learning research (ubiquity symposium)," *Ubiquity*, 2014. [Online]. Available: http://doi.acm.org/10.1145/2591683

[4] B. Du Boulay, "Some difficulties of learning to program," *Jour. Educational Computing Research*, vol. 2, no. 1, 1986. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1207/S15327809JLS0904_3

[5] B. DiSalvo and A. Bruckman, "From interests to values: Computer science is not that difficult but wanting to learn it is." ser. CACM, 2011.

[6] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," *Edu. Psychologist*, vol. 46, no. 4, pp. 197–221, 2011. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00461520.2011.611369

[7] A. King, "Structuring peer interaction to promote high-level cognitive processing," *Theory into Practice*, vol. 41, no. 1, pp. pp. 33–39, 2002. [Online]. Available: http://www.jstor.org/stable/1477535

[8] N. Myller, R. Bednarik, E. Sutinen, and M. Ben-Ari, "Extending the engagement taxonomy: Software visualization and collaborative learning," *Transactions on Computing Education*, vol. 9, no. 1, pp. 7:1–7:27, Mar. 2009. [Online]. Available: http://doi.acm.org/10.1145/1513593.1513600

[9] J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualization systems for introductory programming education," *ACM Transactions on Computing Education*, vol. 13, no. 4, pp. 15:1–15:64, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2490822

[10] P. J. Guo, "Online Python Tutor: Embeddable Web-based Program Visualization for CS Education," ser. SIGCSE '13. ACM, 2013, pp. 579–584. [Online]. Available: http://doi.acm.org/10.1145/2445196.2445368

[11] L. W. Anderson and D. R. Krathwohl, Eds., *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Allyn & Bacon, 2000.

[12] A. M. Paul, "Bill Gates Is an Autodidact. You're Probably Not. Ed tech promoters need to understand how most of us learn." *Slate*, Jul. 2014.

[13] T. Balch, "MOOC Student Demographics (Spring 2013) – http://augmentedtrader.com/2013/01/27/mooc-student-demographics/ Accessed: Sept, 2014."

[14] G. Christensen, A. Steinmetz, B. Alcorn, A. Bennett, D. Woods, and E. J. Emanuel, "The MOOC Phenomenon: Who Takes Massive Open Online Courses and Why?" *(working paper)*, 2013.

[15] P. A. Kirschner and J. J. van Merrienboer, "Do learners really know best? urban legends in education," *Educational Psychologist*, vol. 48, no. 3, pp. 169–183, 2013. [Online]. Available: http://dx.doi.org/10.1080/00461520.2013.804395

[16] J. Huang, A. Dasgupta, A. Ghosh, J. Manning, and M. Sanders, "Superposter behavior in MOOC forums," ser. L@S '14. ACM, 2014, pp. 117–126. [Online]. Available: http://doi.acm.org/10.1145/2556325.2566249

[17] S. F. J. Mak, R. Williams, and J. Mackness, "Blogs and forums as communication and learning tools in a MOOC," in *Proceedings of the 7th International Conference on Networked Learning*, 2010, pp. 275–284.

[18] B. S. Bloom, "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring." *Educational Researcher*, vol. 13, no. 6, pp. 4–16, 1984. [Online]. Available: http://www.eric.ed.gov/ERICWebPortal/detail?accno=EJ303699

[19] R. Schoenfeld-Tacher, S. McConnell, and M. Graham, "Do no harm: A comparison of the effects of on-line vs. traditional delivery media on a science course," *Jour. Science Ed. and Technology*, vol. 10, no. 3, pp. pp. 257–265, 2001. [Online]. Available: http://www.jstor.org/stable/40188615

[20] S. A. Siler and K. VanLehn, "Learning, interactional, and motivational outcomes in one-to-one synchronous computer-mediated versus face-to-face tutoring," *Intl. Jour. A.I. in Education*, vol. 19, no. 1, pp. 73–102, Jan. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1517338.1517342

[21] A. J. Brush, D. Bargeron, J. Grudin, A. Borning, and A. Gupta, "Supporting interaction outside of class: Anchored discussions vs. discussion boards," ser. CSCL '02, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1658616.1658676

[22] M. Guzdial and J. Turns, "Effective discussion through a computer-mediated anchored forum," *Journal of the Learning Sciences*, vol. 9, no. 4, pp. 437–469, 2000. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1207/S15327809JLS0904_3

[23] S. Zyto, D. Karger, M. Ackerman, and S. Mahajan, "Successful classroom deployment of a social document annotation system," ser. CHI '12. ACM, 2012, pp. 1883–1892. [Online]. Available: http://doi.acm.org/10.1145/2207676.2208326

[24] J. Cambre, C. Kulkarni, M. S. Bernstein, and S. R. Klemmer, "Talkabout: Small-group discussions in massive global classes," ser. L@S '14. ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2556325.2567859

[25] C. Kulkarni, J. Cambre, Y. Kotturi, M. S. Bernstein, and S. R. Klemmer, "Talkabout: Making distance matter with small groups in massive classes," ser. CSCW '15, 2015.

[26] D. Coetzee, S. Lim, A. Fox, B. Hartmann, and M. A. Hearst, "Structuring interactions for large-scale synchronous peer learning," ser. CSCW '15, 2015.

[27] S. Lim, D. Coetzee, B. Hartmann, A. Fox, and M. A. Hearst, "Initial experiences with small group discussions in MOOCs," ser. L@S '14. ACM, 2014, pp. 151–152. [Online]. Available: http://doi.acm.org/10.1145/2556325.2567854

[28] D. Coetzee, A. Fox, M. A. Hearst, and B. Hartmann, "Chatrooms in MOOCs: All talk and no action," ser. L@S '14, 2014, pp. 127–136. [Online]. Available: http://doi.acm.org/10.1145/2556325.2566242

[29] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas, "A multi-national study of reading and tracing skills in novice programmers," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '04. New York, NY, USA: ACM, 2004, pp. 119–150. [Online]. Available: http://doi.acm.org/10.1145/1044550.1041673

[30] T. L. Naps, "et al. Exploring the role of visualization and engagement in computer science education," *SIGCSE Bulletin*, vol. 35, no. 2. [Online]. Available: http://doi.acm.org/10.1145/782941.782998

[31] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages and Computing*, vol. 13, p. 259–290, 06/2002 2002.

[32] M. Goldman, G. Little, and R. C. Miller, "Real-time collaborative coding in a web ide," ser. UIST '11.

[33] N. Moraveji, R. Lindgren, and R. Pea, "Organized mischief: Comparing shared and private displays on a collaborative learning task," ser. CSCL'09, 2009.

[34] D. Avrahami, S. R. Fussell, and S. E. Hudson, "IM waiting: Timing and responsiveness in semi-synchronous communication," ser. CSCW '08, 2008. [Online]. Available: http://doi.acm.org/10.1145/1460563.1460610

[35] "MaxMind GeoIP databases and web services," http://www.maxmind.com/en/geolocation_landing.

[36] A. Ogan, S. Finkelstein, E. Walker, R. Carlson, and J. Cassell, "Rudeness and rapport: Insults and learning gains in peer tutoring," in *ITS '12*, 2012. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30950-2_2