Remote Pair Programming in a Visual Programming Language

Jonathan McKinsey

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-139
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-139.html

May 15, 2015

Remote Pair Programming in a Visual Programming Language

by Jonathan C. McKinsey

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Senior Lecturer SOE Dan Garcia
Research Advisor

(Date)

* * * * * * *

Professor Armando Fox
Second Reader

(Date)

1 Abstract

With the growing popularity and global ubiquity of massive open online courses (MOOCS), the demand for new technology to support remote collaboration has similarly increased. In the domain of computer science, collocated pair programming is common in introductory courses, but not well studied in the remote MOOC context despite its increasing prevalence. Introductory computer science courses also employ visual programming languages that allow students to write code by dragging and dropping blocks on the screen. We wanted to formulate a solution that recreates the same learning experience that collocated pair programming for a visual programming language offers.

First, we examined the remote pair programming environment in existing MOOC edX *CS169.1x: Engineering Software as a Service*, which utilizes a text-based language. Remote pair programming sessions in CS169.1 typically consists of multiple students in a single Google Hangouts session with one student sharing her screen and the rest giving feedback. This differs greatly from the traditional pair programming model of a Driver who writes code and a Navigator who reviews the code, and then switching roles after a certain amount of time. Screen sharing also has a number of issues such as poor video quality and lack of keyboard and mouse access, making it an insufficient tool to support remote collaboration for a visual programming language.

Informed by the feedback in a CS169.1x survey, we designed and implemented *Turtle Tango*, a prototype for a fully-integrated, browser-side visual programming language development environment, featuring real-time collaborative editing, video and audio sharing and code execution to support collocated and remote collaboration. We tested Turtle Tango against *Blockly Games: Turtle* in a pilot study comparing the effects of collaborative editing on development speed. In the experimental scenarios, Turtle Tango had faster completion rates (26.2 minutes) than *Blockly Games: Turtle* (33.3 minutes).

While Turtle Tango is still a prototype, it exemplifies the possibilities for remote collaboration with a visual programming language. It aims to encourage the development of new applications to support wider adoption of remote pair programming within MOOCs and traditional classrooms alike, and inspire new research into the realm of remote pair programming and remote simultaneous programming.

2 Acknowledgements

I want to take a moment to thank everyone that made this research possible:

Teaching Professor Dan Garcia, who served as my advisor and mentor during my time at UC Berkeley, introduced me to the visual programming language Snap*!* and cultivated my passion for computer science education.

Professor Sam Joseph of Hawaii Pacific University and edX *CS169.1x:Engineering Software as a Service* largely influenced my view of remote collaborative education, and collaborated with me on remote pair programming research.

Professor Armando Fox, who introduced me to Professor Sam Joseph and whose feedback has been invaluable.

My wife, without whom none of this would be possible.

Lastly, my son, who brought me so much joy in his short time on this earth.

Table of Contents

1 Abstract **1**

2 Acknowledgements **2**

3 Introduction **5**

4 Background **7**
- 4.1 Pair Programming . . . 7
  - 4.1.1 Pair Jelling and Debugging . . . 8
  - 4.1.2 Free Rider Effect . . . 9
  - 4.1.3 Retention . . . 9
- 4.2 Remote Pair Programming . . . 10

5 Study 1: Surveying CS169.1x **10**
- 5.1 Methodology . . . 11
- 5.2 Results . . . 12

6 Turtle Tango **15**
- 6.1 Libraries . . . 16
  - 6.1.1 Web Real-Time Connection and PeerJS . . . 16
  - 6.1.2 Blockly . . . 17
  - 6.1.3 Skulpt . . . 19
- 6.2 Authorizing the Application . . . 19
- 6.3 Toolbar . . . 19
- 6.4 Turtle Blocks . . . 20
  - 6.4.1 Forward . . . 21
  - 6.4.2 Turn . . . 21
  - 6.4.3 Pen . . . 21
  - 6.4.4 Defining a New Block . . . 22
- 6.5 Establishing a Peer Connection . . . 25

7 Study 2: Turtle Tango Pilot **26**
- 7.1 Methodology . . . 26
  - 7.1.1 Introduction to Turtle Tango . . . 26
  - 7.1.2 Introduction to Skype and Blockly Turtle . . . 28
  - 7.1.3 Turtle Tango Versus Blockly Turtle . . . 28
- 7.2 Results . . . 29

8 Future Work **31**

9 Conclusion **31**

3 Introduction

Education has traditionally consisted of one teacher imparting knowledge to a class, with the number of students constrained by the size of the physical classroom. However, with the advent of free Massive Open Online Courses (MOOCs), education evolved from catering to a relatively small percentage of the population to being freely available to anyone with a computer and Internet access. Traditionally, it is not uncommon for enrollment in popular courses to exceed 1,000 students. However, MOOCs are truly massive with an average enrollment of 43,000 students [11]. In 2014, over 400 universities across world offered a combined 2,400 MOOCs, with 22 of the top 25 US News & World Report ranked universities participating [20]. However, the transition from the traditional classroom to the online one is not an easy task. The constraints of online classroom and its massive scale force educators to rethink their teaching methodologies and create new supporting technologies without sacrificing their curriculum.

CS10: The Beauty and Joy of Computing, an introductory computer science course at UC Berkeley, is on the brink of launching its MOOC instance, but faces a major problem: how do you support pair programming for a visual programming language online? *Pair programming* is a programming practice where two programmers share one computer, continuously collaborating on the same coding problem [22]. Fundamentally, pair programming is designed for both programmers to be physically present in the same room, or collocated. In the MOOC context, pair programming becomes a distributed synchronous collaboration problem on a massive scale as collocation is no longer feasible. This requires the implementation of *remote pair programming* (RPP) technologies to support collaboration in geographically different locations.

We first approached non-collocated pair programming problem for CS10 by surveying remote pair programmers in an existing MOOC, *CS169.1x: Engineering Software as a Service*, using Ruby. CS169.1x students predominately use *screen sharing* technologies where one user shares all or part of her computer desktop as a video stream to another user. Existing communication products such as Google Hangouts and Skype that package webcam video and audio sharing with screen sharing were used. However, these products have many limitations, including: poor video quality, high resource consumption, and the lack of ability for keyboard and mouse sharing. Despite these shortcomings, CS169.1x students embraced Google Hangouts and Skype, because

they were sufficient for the modified form of pair programming prevalent in CS169.1x.

Students in CS169.1x treated these remote pairing sessions like group study sessions, with one person presenting her homework solution and the rest of the students reviewing it without any keyboard or mouse access to the presenter's computer. This contrasts with CS10 pair programming, which is closer to the traditional definition where each programmer alternates between acting as the *Driver* — developing code — or as the *Navigator* — reviewing the code as it is written. When the pair switches roles, there needs to be a seamless transition, or pair programmers could become less efficient or even abandon pair programming all together. In a screen sharing only environment, students have to transfer the code from one computer to the other and deal with the pitfalls of screen sharing. A better solution is to have both members of the pair edit shared code stored in the cloud, commonly known as *collaborative editing*.

Another consideration is that video and audio sharing technologies like Google Hangouts and Skype require client-side installation and setup, which increases the time startup cost of RPP. Google Hangouts and Skype are also not cross compatible with each other, meaning students will probably end up setting up both applications eventually. It would be easier to directly integrate video and audio sharing into an entirely browser-based application with no setup or installation required.

Ultimately, transitioning pair programming with a visual programming language into the distributed MOOC context requires more than just reliance on an off-the-shelf screen sharing technology for the most optimal learning experience for the student. We bundled the essential features needed to successfully enable RPP with a visual programming language and created an all-in-one solution combining collaborative editing, video and audio sharing as well as code execution in an integrated development environment prototype called *Turtle Tango*. We deployed Turtle Tango in a small pilot study where we measured the effect of collaborative editing for a visual programming language on the completion time of drawing activities. Finally, we examine the results and suggest what additional studies should be done in the future.

4 Background

RPP is an effort to recreate the pair programming environment with partners at two different locations instead of one. The CS169.1x survey touches on more than just the common off-the-shelf technologies used during RPP sessions, but also on how these all-purpose technology affect the behavior of students during sessions. To best understand the context and findings of our two RPP studies is to first look at the existing research in pair programming.

4.1 Pair Programming

The typical response when one confronts a problem that cannot be solved alone is to go consult with a peer close by. In pair programming that peer is continuously available and the communication is structured in such a way that both partners benefit. Each programmer alternates between acting as the Driver or as the Navigator. The Navigator is not a passive observer, but is instead actively participating in the development process by looking for defects, considering alternatives and determining code coverage. Both Driver and Navigator work together to produce one artifact and, thus, share equal ownership of the code, regardless of which role she was in when the code was written. Even if one programmer is significantly more experienced than the other, it is important to take turns "driving" so that neither partner becomes complacent in her role. Pair programming can also help a programmer alleviate self-doubt about her own programming skills, because she can improve by constantly observing and obtaining feedback from her partner [22].

Pair programming in the professional world was initially popularized by eXtreme Programming (XP), one of many Agile software development processes. Created by Kent Beck, Ward Cunningham and Ron Jeffries, XP emphasizes a collaborative team environment with continuous informal and immediate communication. In XP, all production code must be continuously reviewed, or, in other words, written while pair programming [21].

It may seem initially counter-intuitive that assigning two people on the same task could be more efficient than putting them on separate tasks. However, existing research already shows that this real-time exchange of ideas between partners significantly decrease the probability of proceeding with an inferior design, and that continuous code reviews reduce the number of code defects [22]. Moreover, collaborative programming is an outlet for each

member of the group to contribute her own unique prior experiences and task relevant knowledge producing more diverse approaches and thus more likely to generate the best solution [12]. In an experiment at Temple University, 15 professional programmers were separated into 5 individuals and 5 pairs and then asked to solve a problem for 45 minutes. The solutions were then assessed in terms of readability — how easily problem solving strategy could be deduced from the code — and functionality — how well the solution addresses the objectives in the problem statement. The pairs completed the task in an average of 30 minutes compared individuals who took an average of 42 minutes. The pairs also significantly outperformed individuals in terms of readability and functionality and also report greater satisfaction and confidence in their solutions [19].

4.1.1 Pair Jelling and Debugging

While the groups in the Temple University study were able to complete their assignments faster than individuals working alone, because the groups were comprised of a pair of programmers, the groups still took 18 more programmer minutes over individuals [19]. This additional time difference may be because pairs that are unaccustomed to working with one another still need to adjust to each other's work habits, programming styles and even ego. This transition period as known as pair jelling typically takes between a few hours to several days, depending on the individuals. A University of Utah study measuring the effectiveness of pair programming with senior software engineering students discovered that while 60% more programmer hours were spent by the pairs on the first assignment, the adjustment period decreased to 15% for the second assignment [21]. Also, because both partners are expecting continuous contribution and input, each partner is far less likely to go off task. Pair programming keeps each participant more productive offsetting the initial startup cost of pair jelling.

Also the Temple University study did not measure the time spent debugging the programs, but found that code produced by individuals had more errors than code produced while pair programming [19]. Similarly, in the University of Utah study, pairs had on average 15% fewer defects than individuals [21]. These studies suggest that the initial time cost of pair programming greatly decreases the time cost of debugging later on.

Despite research indicating pair programming has a positive impact on code readability, functionality and quality, even extending to student enjoy-

ment and confidence, instructors still continue to require students to complete programming assignments independently [17]. The reliance of individual programming largely stems from the difficulty instructors face with measuring accountability and learning in pair programming situations. Essentially, instructors and students alike are concerned if collaboration will give rise to free riders who receive equal credit for work without contributing equally to the task.

4.1.2 Free Rider Effect

There are two types of reasons why groups do not achieve their maximum potential productivity: coordination loss, when a group fails to optimally coordinate or combine the contributions of individual members, and motivation loss, or social loafing, when group members do not exert maximum or equal effort [15]. According to Kerr and Bruun, the free rider effect is largely a motivational issue stemming from self-perception that one's work is unnecessary towards group success and thus people are less motivated to contribute.

To alleviate the free rider problem when assignments are completed collaboratively, each partner must also be tested solo to isolate individual understanding. A study of 86 pair and 141 individuals conducted at UC Santa Cruz measured the effects of pair programming on code quality and individual course mastery. Students were asked to make a list of 3 peers and then randomly assigned a partner. Pairs alternated between driver and navigator roles every hour working on projects. The non-pairing group were required to complete assignments independently. Overall, the pairing group outperformed the non-pairing group significantly on homework assignments in terms of grades [16]. Final examination scores between the pairing and non-pairing groups were comparable, indicating that pair programming does not inherently generate free riders and negatively impact individual learning.

4.1.3 Retention

At first glance, the UC Santa Cruz study seems to conclude that students who pair program master the course materials at least as well as those who program individually. However, the retention rate of pairing section (92%) was dramatically greater than the non-pairing section (76%) and also better than previous semesters that only had non-pairing sections (85%) [16].

McDowell et al. suggest that the difference in attrition rates may have attributed to higher final exam scores for the non-pairing section. If the weaker students in the non-pairing class dropped the course, while their counterparts in the pairing section stayed, the weaker students would have dragged down the final exam average in the non-pairing section.

Retention benefits of pair programming also extended beyond the current course and into the subsequent courses. Both men and women who had pair programmed were significantly more likely to declare a computer science related major [17].

4.2 Remote Pair Programming

Pair programming is typically done as collocated activity with both programmers sharing control of a single physical computer, keyboard and mouse. Remote pair programming (RPP) brings pair programming to the geographically distributed environment where collocated collaboration is not feasible. RPP is supported by technologies such as screen and desktop sharing as well as collaborative editors over the internet. In *desktop sharing*, all users, not only share a single screen, but also have simultaneous keyboard and mouse access. *Collaborative editors* support synchronous read and write operations on shared files. RPP has already been proven to have the same benefits of collocated pair programming citing greater satisfaction of the communication and cooperation within the group [10].

5 Study 1: Surveying CS169.1x

edX CS169.1x: Engineering Software as a Service is a MOOC course inspired by UC Berkeleys upper-division software engineering course. CS169.1x teaches the fundamentals of software engineering by employing Agile techniques to develop Software as a Service (SaaS) using Ruby on Rails. Because students are encouraged to collaborate using pair programming on their homework assignments, CS169.1x was one of the early adopters of remote pair programming in the MOOC world [13].

To take part in a pair programming session, CS169.1x students first join the Google+ edX SaaS community [5] composed of current students, alumni and community teaching assistants. Then the student may either join an existing event or create a new Google Hangouts On Air event at a time of

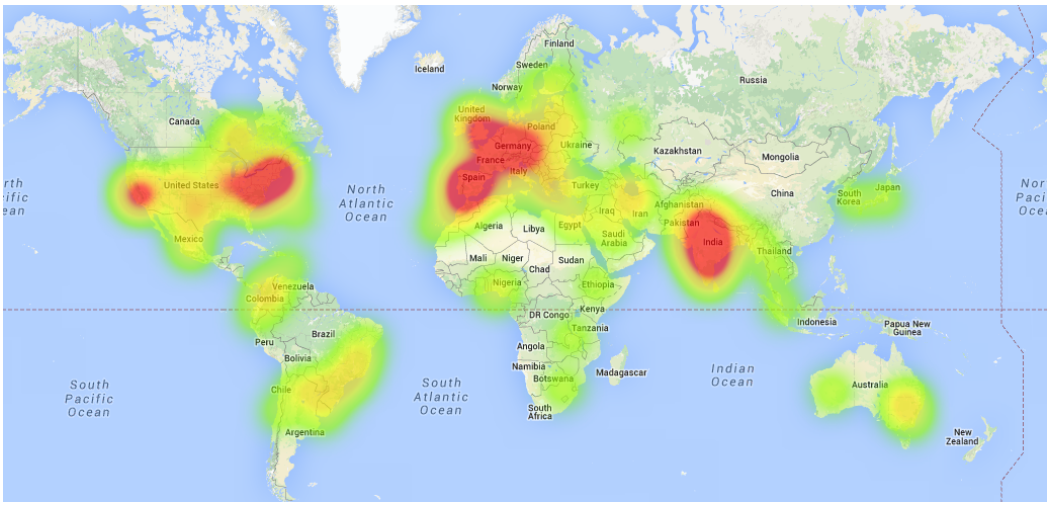

Figure 1: Heatmap of students taking edX CS 169 who responded to a survey on remote pair programming

their choosing. Students may then chose their own sharing technology for the session, but most pick a technology that integrates easily with Google Hangouts (see Table 1). While CS169.1x does not impose a limit on how many people may join a session at any one time, Hangouts On Air can only support up to 10 participants at a time.

5.1 Methodology

In Fall 2013, we sent an informal RPP survey to the Google+ edX SaaS community soliciting feedback on the geographical location of remote pair programmers, the technology used and qualitative feedback on RPP sessions:

- Which Country do you spend most of your time in?
- Which State/Region do you spend most of your time in?
- Which City do you spend most of your time in?
- Please describe your first remote pair programming experience in as much detail as possible.
- Describe a typical remote pair programming session.

- Do you prefer pair programming over individual programming? Why or why not?
- Do you prefer remote pair programming over in-person pair programming? Why or why not?
- What technologies do you use to do remote pair programming?
- Any other thoughts on remote pair programming sessions and how they are connected to MOOCs like edX 169?

5.2 Results

The 225 responses on the RPP survey serve as an initial look at the current MOOC RPP landscape from which to elaborate upon or address for CS10 [14]. The heatmap of survey respondent physical locations (see Figure 1) highlights the geographical disparition between MOOC students and necessity of implementing a viable methodology for these students to collaborate [18]. Pair programming events are predominately initiated with one student serving as the organizer creating the event. These student-led sessions allow pairings to be organic and flexible as sessions can be created with little to no notice and with little involvement from the course staff. However, the success of student-led sections is strongly dependent on the organizer to coordinate the attendees and the willingness of the organizer and attendees to assume the respective Driver and Navigator roles. A pair programming event could attract multiple attendees leaving it up to the organizers or the attendees themselves to separate into pairs. The large sessions may quickly dissolve due to lack of coordination or result in few active participants with numerous passive observers. Here are some student comments from the survey:

> "When I joined the first session there were already 7 people in the session but 2 people were really the only ones working/contributing. I tried to get some people to break out into another session but frankly there wasn't a good or easy way to do that. I ended up leaving after 5 mins and decided to create an event of my own for the next day."

> "5 or 6 people dropping in and out, some lurking while two [program] and I was able to get a helpful screenshot or two as they

> progressed. Even though I was not at their point in the hw, and their code was not done, it helped to have something to look at later."
>
> "Pair programming on the SaaS homework's was a little frustrating, especially in the early homeworks, because many students drop into public 'on air' hangouts, take a screenshot of your code and then disappear. I can only assume they are attempting to get a passing grade on the AutoGrader without doing any work. Also in pairing sessions with more than 3 programmers, a few observers tend to lurk silently without contributing or offering to drive, I find this off-putting."

These free-riding observers may resort to taking *passive screenshots* — capturing screenshots of source code without actively contributing to its creation. These students may believe that merely attending the session entitles them access to the source code. In the extreme case, students commit *passive screenshot drive-bys* — joining multiple sessions, obtaining several screenshots and quickly exiting without participation. Presumably, passive screenshot drive-bys afford the offender the ability to piece together a solution from different groups, and thus, committing a new form of cheating that could implicate multiple unwitting participants. This type of behavior has led some students to indicate that pair programming is incompatible with the course honor code. Some students have already discovered a workaround to protecting themselves against others who take passive screenshots and passive screenshot drive-bys, by using a private Hangout event instead:

> "When we were using public Google Hangout sessions, the result was typically chaotic. Until recently (in part 2 of the course we have switched to a private hangout with just 2 of us), it was quite distracting as different people would come and go with different levels of concerns (ranging from clueless to already completed the assignment and just wanting to help). In fact, it was more of a group study (programming) session than actual pair programming. Often productive and interesting, but not actually pair programming (until our recent switch to a private hangout). "

However, private Google Hangout events are only available to known participants and is not a viable options for individuals unfamiliar with their cohort or have unpredictable schedules:

"I have a very young family with 3 small children and I found that I could not allocate time blocks for working on the course. When I had the chance to get something done I grabbed it"

"I find it extremely hard to schedule time where I can commit to working on the assignments because I have to fit it in with work, family, etc. It is also incredibly difficult being in a timezone much different to most others."

Google Hangout with Screenshare	65.3%
Skype Screenshare	33.3%
TeamViewer	29.9%
Cloud9	11.1%
Other	11.1%
Google Hangout + Floorbits	7.6%
Google Hangout + MadEye	4.9%
GNU Screen	4.2%
ScreenHero	4.2%
Tmux	4.2%

Table 1: Prevalence of RPP Technologies in CS169.1x

For some survey respondents, the barrier to entry into remote pair programming is lacking the requite hardware such as webcam or microphone. For others it is the unfamiliarity with the collaborative technology for screen or desktop sharing. One student mentioned the sheer amount of new software tools and concepts required in the curriculum alone was overwhelming. After adding on new RPP technology, there was too much new information to absorb. Modern screen and desktop sharing technology are still limited in the quality of the images produced. Depending on the screen resolution of the computer of the student sharing her desktop and the computer of the students watching the feed as well as the technology used to share the video feed (e.g. Skype, Google Hangouts), the video image can be blurry and indecipherable. In text-based languages, students can use a collaborative editor such as Floorbits or MadEye instead of using screen or desktop sharing.

When students get over the initial growing pains with sharing technology and scheduling, they quickly discover RPP has the same benefits as collocated pair programming with the added benefit of interacting with peers on a global scale where in-person collaboration would not be possible:

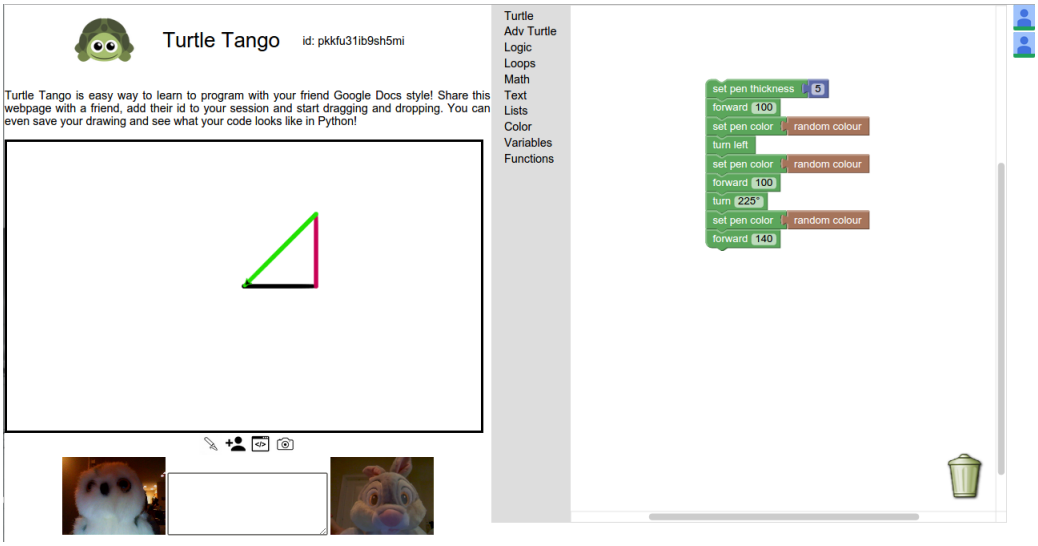

Figure 2: The Turtle Tango interface.

> "I prefer pair programming since it helps in building global contacts interacting with peers all around the world accomplishing a task in good time efficiently and helps making our code universally understandable."

> "During one of the early homeworks I worked with a Spanish guy living in Finland....I enjoyed seeing someone else's approach and definitely learned some new tricks from it. "

6 Turtle Tango

Turtle Tango (see Figure 2) is a browser-side real-time collaborative development environment with integrated video and audio for Logo-style turtle drawing. It uses cloud storage to enable simultaneous collaboration without the blurriness and choppiness of the tools used in CS169.1x (Google Hangouts and Skype). Students are able to move a single turtle sprite around the screen. Because the turtle has a little pen on its belly, it draws a line as it walks. Students can control the pen color and line width, and even retract

the pen so that no line is drawn. Turtle Tango can be used to draw anything from simple shapes to complex fractals.

The CS169.1x survey responses indicate a number of possible considerations for RPP adoption with CS10.:

1. **Collaborative editing with video and audio sharing.** Desktop and screen sharing are needlessly resource-intensive processes and can be blurry or choppy depending on the screen resolution of the group members and the internet speed. Instead, students should be using a collaborative editor for a visual programming language with video and audio support, where the video is optional rather than pivotal to the RPP experience.

2. **Server-side technology with small learning curve.** When a student is already overwhelmed with the new concepts and technologies needed for the existing curriculum, the RPP technology should be seamlessly integrated. There should be little to no setup on individual clients, with the bulk of the work handled on the server side.

3. **Groups of two only.** To reap the full benefits of pair programming and to discourage free riders, group programming sessions must be limited to two participants.

These are all issues that Turtle Tango will address.

6.1 Libraries

Turtle Tango is supported by three main libraries: PeerJS, Blockly and Skulpt.

6.1.1 Web Real-Time Connection and PeerJS

Web Real-Time Connection (WebRTC) is World Wide Web Consortium (W3C) draft API for browser-to-browser applications enabling voice calling, video chat and peer-to-peer file sharing without installing additional add-ons or plugins. WebRTC standardizations are implemented in Mozilla Firefox, Google Chrome and Opera browsers as well as Android and iOS mobile platforms. *PeerJS* is a Javascript library that wraps WebRTC API into a simple peer-to-peer connection API [7]. Each peer is distinguished by an unique

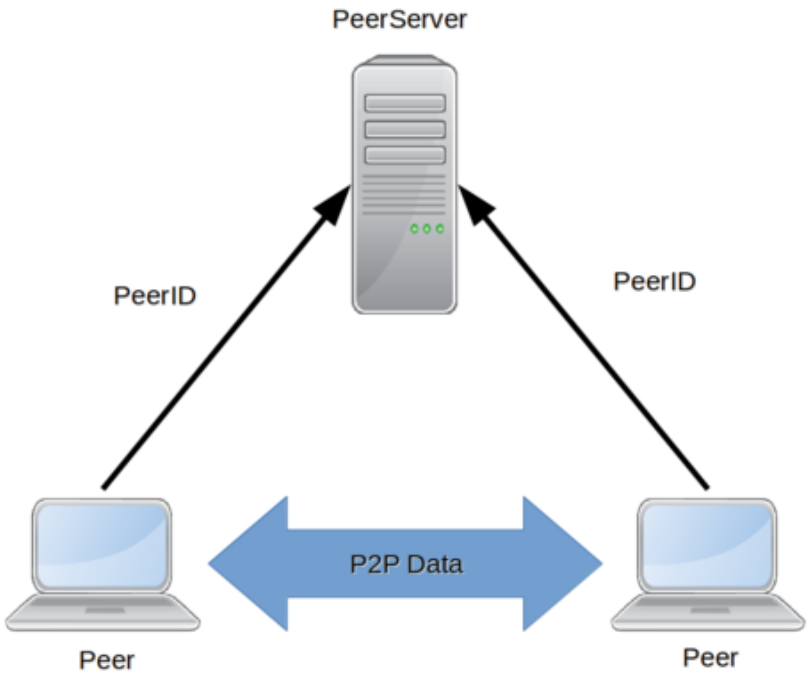

Figure 3: The PeerServer only brokers a connection between the users. Actual data exchange is entirely peer-to-peer.

peer identifier assigned by the brokering *PeerServer*. The brokering server only helps establish the peer-to-peer connection. No peer-to-peer data passes through the connection broker. The PeerJS team provides a PeerServer that supports up to 25 simultaneous connections per API key. See Figure 3 above.

6.1.2 Blockly

Blockly is a Javascript library for creating visual block-based programming editors capable of translating from visual blocks to a text-based programming language such as Javascript or Python [3]. Each Blockly block must be mapped with an equivalent translation in each supported language, which is useful in obscuring the language's syntax from the end user.

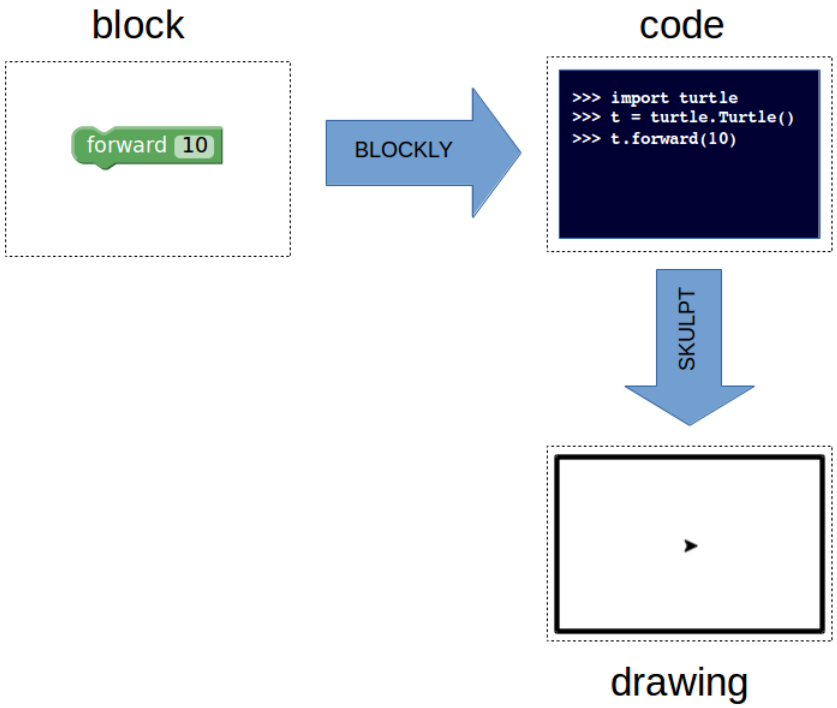

Figure 4: The progression from Blockly block to Python code to turtle drawing

Unlike integrated development environments (IDEs), the Blockly framework does not provide a run environment. It is up to the implementing application or end user to execute the code in their own environment.

Blockly is integrated with *Google's Realtime API* [6], which enables a shared workspace between browser sessions. Google's Realtime API utilizes *Google's Drive API* [4] and currently requires implementing applications to ask users to provide a Google account for access. An additional requirement is that applications that interact with the Google Drive API must be hosted on *Google App Engine* [2].

6.1.3 Skulpt

Skulpt is an entirely browser-side Javascript implementation of the Python programming language [8]. This translation from Python to Javascript allows Python to be run within any Javascript-supported browser. Thus Skulpt enables Blockly blocks to indirectly manipulate the built-in Python turtle library to draw on the screen as shown in Figure 4.

6.2 Authorizing the Application

When a student first enters the Turtle Tango website [9], she is assigned a peer identifier, which will be used to broker a connection between two peers. The student then will give permission to share her webcam and microphone. It is also possible to chose not to share the devices, but some built-in functionalities will be disabled when the session is shared with a peer. If webcam is enabled, the video automatically start to stream at the bottom of the page.

The student then authorizes this application by logging into her Google account and providing her login credentials. This triggers Turtle Tango to load the Blockly workspace as well as populate the appropriate blocks. A shared text chat box at the bottom of the page also becomes enabled. Anyone with access to this page will be able to see and edit the contents of the chat box.

6.3 Toolbar

Turtle Tango has four buttons on its toolbar as shown in Figure 5: pen, add-a-friend, generate code and camera. When the pen button is clicked, it triggers a translation of each Blockly block to its Python equivalent code. These definitions are mapped manually per block within the source code. The Python code is then executed by the Skulpt library, which dynamically renders the image.

The add-a-friend button opens a modal dialog (See Figure 6) where a student can enter the id of another student. Clicking the Ok button will establish a media connection between the two peers and allow webcam video and audio to be shared across that connection.

The generate code button, like the pen button, triggers a translation from blocks to Python. However, instead of running the Python code through Skulpt, the code is displayed in a modal dialog (see Figure 7) for a student

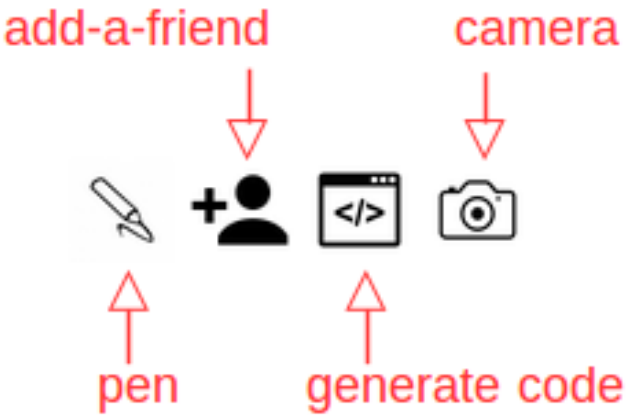

Figure 5: The Turtle Tango toolbar.

to review or save. Running the generated Python code in a Python 2.7 interpreter will produce the same drawing as Turtle Tango.

Lastly, the camera button displays an image of the current turtle drawing (see Figure 8). This version of the drawing will not display the actual turtle, only the turtle pen lines. The drawing can be saved by right-clicking on the image.

6.4 Turtle Blocks

Google Blockly provides a number of built-in blocks for operations such as those involving logic, math and iteration. Turtle Tango defines several turtle blocks designed specifically for drawing on the canvas. There are two types of inputs to turtle blocks, regular and advanced (see Figure 10). *Regular inputs* are designed for beginners by providing default values or selection menus. *Advanced inputs* have connectors to other blocks which enable dynamic values from nested blocks. Because using outputs of one function as an input to another function requires an understanding of return values and types, beginners are encouraged to start with regular input version of blocks first before moving on to advanced blocks. Regular input blocks are found within the Turtle category in the workspace, while advanced input blocks are in the Adv Turtle category.

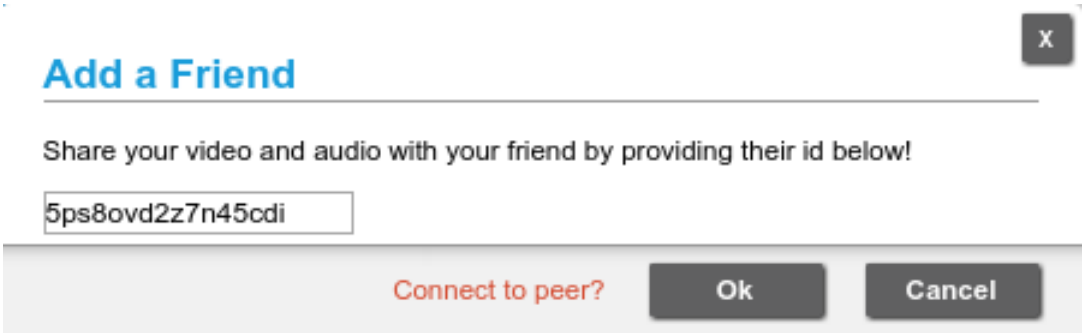

Figure 6: The Add a Friend modal dialog.

6.4.1 Forward

The `forward` block moves the turtle a specified number of steps. If the turtle pen is down, the turtle will leave behind a pen trail as it walks. If a positive number of steps is given then the turtle moves forward. Conversely, if a negative number is given the turtle will walk backwards without changing its heading.

6.4.2 Turn

The `turn` block changes the direction the turtle currently faces as known as the turtle's heading. The `turn left` block changes the turtle's heading 90 degrees to the left where the `turn right` block changes it 90 degrees to the right. The regular input `turn` block allows a student to select an angle using a visual selector (see Figure 11).

6.4.3 Pen

The pen blocks govern whether the turtle leaves a pen trail when it moves, the width of the pen trail and the pen color. The `pen up` block disables the turtle pen trail. Conversely, the `pen down` block enables it. By default, the turtle starts with its pen down. The thickness of the pen trail can be modified by using the `set pen thickness` block. Finally, the `set pen color` blocks change the color of the turtle pen trail. Students are able to click on the default color provided with the regular input `set pen color` block and select a new color from 70 available. Using the advanced input version of the same

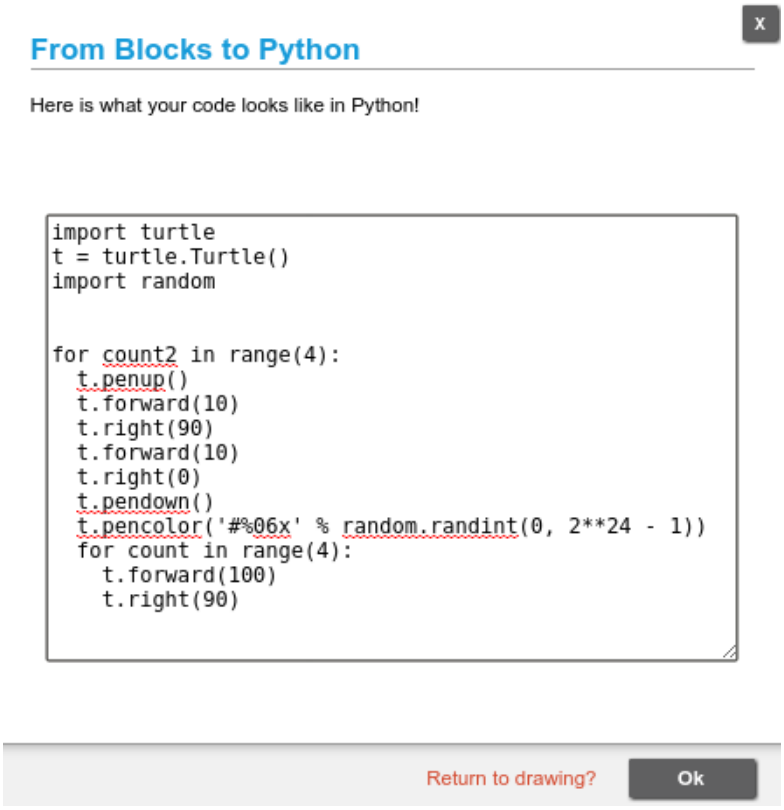

Figure 7: The Generate Code modal dialog displaying Python equivalent of sample program shown in Figure 9.

block and a block from the built-in color category, students can also create their own pen trail colors.

6.4.4 Defining a New Block

Because Blockly is a general-purpose visual editor, custom blocks must be mapped to each supported language for translation. For instance, consider the source code to create a `forward` block with advanced input:

```
Blockly.Blocks['turtle_forward_adv'] = {
    init: function () {
        this.setColour(120);
```

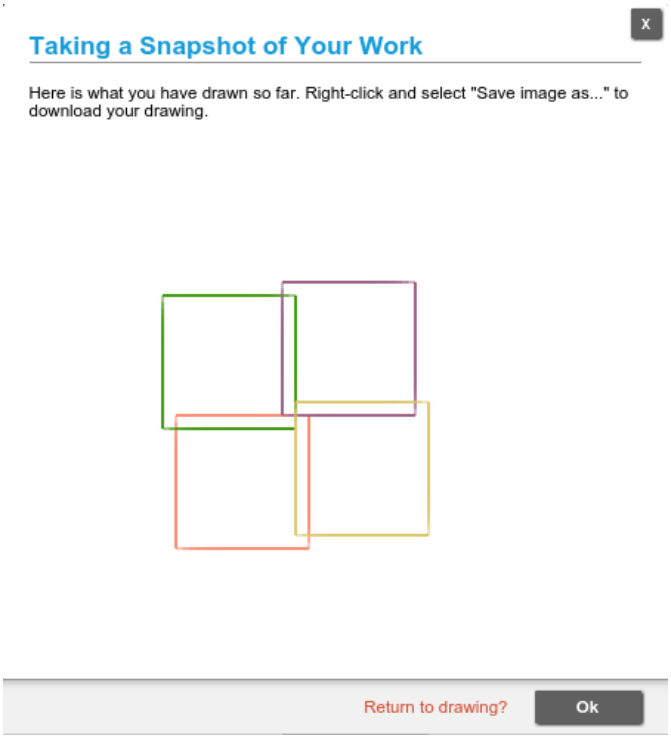

Figure 8: The Camera modal dialog displaying image drawn by sample program shown in Figure 9.

```
        this.appendValueInput("DISTANCE")
            .setCheck("Number")
            .appendField("forward");
        this.setPreviousStatement(true);
        this.setNextStatement(true);
        this.setTooltip('turtle walks forward certain paces');
    }
};
```

First, define a Blockly block with physical color green. Then specify a single input variable `DISTANCE`, which must be a `Number` type. Both `setPreviousStatement` and `setNextStatement` are `true` indicating that the `forward` block is a statement that can have statements before and after it with no restrictions on type. The `setTooltip` function labels the `forward`

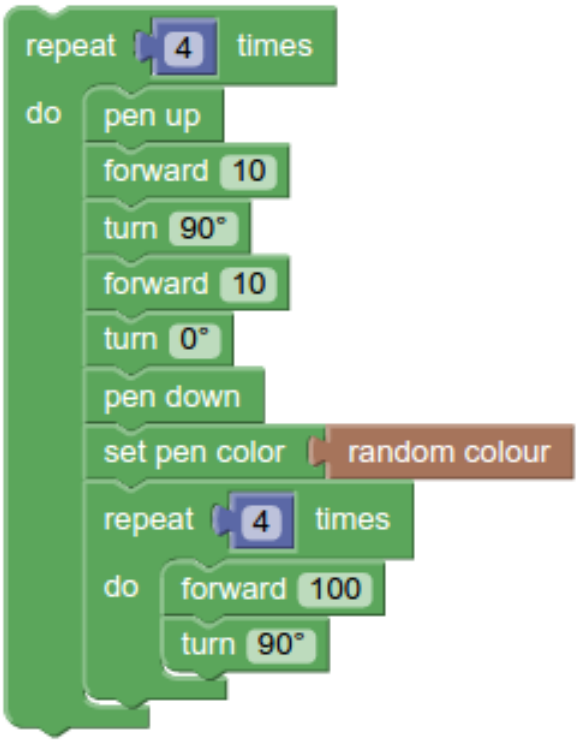

Figure 9: Sample program.

block with a mouse-over tooltip.

Then, implement a mapping from the `forward` block to Python:

```
Blockly.Python['turtle_forward_adv'] = function(block) {
    var value_distance = Blockly.Python.valueToCode(block,
      'DISTANCE', Blockly.Python.ORDER_ATOMIC);
    var code = 't.forward(' + value_distance + ')\n';
    return code;
};
```

`value_distance` represents the input `DISTANCE`, which may be a single value or a function that eventually evaluates to a `Number`. `code` is the Python translation incorporating the input variable `value_distance`.

Figure 10: regular and advanced input versions of forward blocks

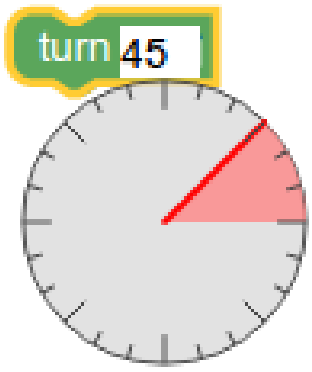

Figure 11: turn block degree visual selector

6.5 Establishing a Peer Connection

First, the student should share the URL of her current session with her peer. To connect to the peer, click on the add-a-friend button on the toolbar (see Figure 5). In the add-a-friend dialog box (see Figure 6), the student specifies the peer identifier of her friend which signals the PeerServer to start brokering the connection. Only one person needs to start the connection. Once the PeerServer successfully brokers the connection, both students will automatically begin receiving their partner's media stream which is displayed in a second window at the bottom of the screen.

The actions of one student in the Turtle Tango workspace is automatically propagated to her peer and vice versa. When a block is selected and dropped in a new location, the peer will see the final location updated on her screen, but the intermediate positions of the block with not be rendered. Workspace and collaborative text box syncing is supported by Google's Realtime API. Through this API, the students are able to both manipulate a shared resource hosted on Google Drive in near real time. However, because

the turtle canvas is actually drawn by Skulpt, each turtle drawing instance is rendered independently on each student's local browser. When one student clicks on the pen button on the toolbar, it triggers the turtle to draw on her local canvas, but also sends a draw message through the PeerJS peer-to-peer connection. On receiving a draw message, her peer's Turtle Tango application will also trigger the turtle to draw, thus syncing the draw event between peers.

7 Study 2: Turtle Tango Pilot

After Turtle Tango was publicly deployed on Google AppEngine and had under gone extensive development testing, it was ready to be used to test the effect of collaborative editing in the context of RPP and visual programming languages. Because collaborative editing could also support simultaneous editing without the roles defined in pair programming, remote simultaneous programming (RSP) was also tested.

7.1 Methodology

Six developers at a major corporation, all holding MS degrees in computer science, volunteered to beta test the Turtle Tango prototype after work. Two different development setup were used: Blockly Games: Turtle [1] with Skype and Turtle Tango. Blockly Games: Turtle, or Blockly Turtle, is an implementation of turtle graphics in Blockly, but without real-time collaborative editing and video and audio built-in.

First, developers were separated into three pairs of two. These pairing remained consistent throughout the entire experiment. Then each team completed two initial assignments designed to familiarize them with the technology used for the experiment, and completed some pair programming challenges.

7.1.1 Introduction to Turtle Tango

Two laptops in two conference rooms on separate floors were used. A new Google account was created prior to the experiment and shared with the participants. In each pair, one participant was assigned the role of Leader and given the following instructions:

1. Navigate to `https://tango-with-code.appspot.com` in Google Chrome
2. Enable mixed content by clicking the shield icon in the address bar and selecting "Load unsafe scripts"
3. Authorize the application and sign-in with the provided Google account and password.
4. Share webcam and microphone with the application.
5. Email the complete URL in the address bar to other participant.
6. In the chat box, provide your peer ID.

The other participant was given these instructions:

1. Wait until you get an email from your collaborator. Then, in Google Chrome, navigate to address provided.
2. Enable mixed content by clicking the shield icon in the address bar and selecting "Load unsafe scripts."
3. Authorize the application and sign-in with the provided Google account and password.
4. Share webcam and microphone with the application.
5. Look for your collaborator's peer id in the chat box and use the add-a-friend button to start a session.

After the peer connection was successfully brokered, each participant was verbally informed of the definition of pair programming and randomly assigned a starting role by a coin toss. The pairs were instructed to explore the Turtle Tango interface and to switch roles every 15 minutes, which was enforced by the moderator who observed the session. Each pairing session only lasted 30 minutes. Directly after the pairing session, each participant was interviewed individually for 10 minutes for their feedback.

7.1.2 Introduction to Skype and Blockly Turtle

The same laptops and conference rooms from Introduction to Turtle Tango were used. Two new Skype accounts were created prior to the experiment and both machines were logged into Skype. The toss of a coin determined which participant would take the role of Leader and be given the following instructions:

1. In Skype contact list, right-click on the only contact in your list and select

   ```
   Share Your Screen -> Share Full Screen
   ```

2. In Google Chrome, navigate to `https://blockly-games.appspot.com/turtle?lang=en&level=10`

The other participant was given these instructions:

1. Wait until you get a Skype call. Click Accept to answer the call and then click Accept to start the video call.

When the call connected, the pairs were instructed to explore the Blockly Turtle interface and to switch roles every 15 minutes, which was enforced by the moderator who observed the session. The Leader who started the screen sharing session acted as the Driver first. Because Blockly Turtle does not have collaborative editing support, when it was time to switch roles, the participants would physically switch rooms. This simulates the interruption necessary to exchange code between programmers using this method. Each pairing session only lasted 30 minutes, which did not include the switching time. Directly after the pairing session, each participant was interviewed individually for 10 minutes for their feedback.

7.1.3 Turtle Tango Versus Blockly Turtle

Now the participants had to solve three fractal drawing problems, each in a different scenario (see Table 2):

1. Skype and Blockly Turtle. Same instructions as Introduction to Blockly Turtle portion.

2. Turtle Tango Remote Pair Programming (RPP). Same instructions as Introduction to Turtle Tango portion.

3. Turtle Tango Remote Simultaneous Programming (RSP). Similar instructions to Introduction to Turtle Tango portion, but both can code at the same time.

Each team member was handed a copy of the fractal drawing to replicate. All three teams solved the same problem number at the same time, but in different conference rooms. Three moderators facilitated the experiment by starting the stopwatch and announcing when to switch roles. Aside for the teams under the RSP scenario, pairs were asked to switch roles every 10 minutes instead of 15 minutes to ensure each participant at least experienced each role once. When the team believed they had the solution, the moderator stopped the stopwatch and checked the solution. If the solution was incorrect, the moderator started the timer again and the team worked until they reached the solution. If the solution was correct, the scenario ended and the moderator recorded the time on the stopwatch. The participants were asked to complete a questionnaire after they completed all of the scenarios.

1. Rank the Turtle Blockly, Turtle Tango RPP, Turtle Tango RSP scenarios from favorite to least favorite.

2. Order fractal problems 1, 2 and 3 in difficulty from high to low.

	Group A	**Group B**	**Group C**	**Average Time (min)**
Fractal 1	BT	TT RPP	TT RSP	32
Fractal 2	TT RPP	TT RSP	BT	29
Fractal 3	TT RSP	BT	TT RPP	24.7
Average (min)	28.7	29	28	

Table 2: Fractal and scenario assignments per group and average completion time. BT = Blockly Turtle and TT = Turtle Tango

7.2 Results

In general, the initial feedback from the pairs was very positive after Introduction to Turtle Tango exercise:

> "This would be good for my kids. Seems like I am playing a game rather than coding."
>
> "Turtle Tango is a novel and interactive way to teach basic programming concepts to students especially those who learn visually."
>
> "I can't believe I was writing programs in Python in less than 30 minutes."

Some of the pairs mentioned that a tutorial video would be helpful in getting started:

> "There were just so many categories of blocks. I didn't know what to start with."

The feedback from Introduction to Skype and Blockly Turtle was negative about the screen clarity:

> "The resolution was just eye wrecking."
>
> "I found myself asking my partner to tell me what blocks he was picking up, because I couldn't read them."

Participants also felt that having to physically switch rooms to simulate code transfer process that would have to occur without collaborative editing support was having them take a step backwards:

> "Showing me Turtle Tango first and then having me use Skype and some bad quality screen sharing is like giving me a car, then taking it away and asking me to take a bus. I am used to using Google Docs to write papers with other people, and I will not willingly go back to the days of emailing drafts back and forth."

In the experimental scenarios of Blockly Turtle and Turtle Tango, Turtle Tango had faster average completion rates (26.2 minutes) than Blockly Turtle (33.3 minutes) as show in Table 7.2. Turtle Tango RSP (24 minutes) was dramatically faster than RPP (28.3 minutes). 100% of the participants ranked Blockly Turtle as their least favorite scenario, while 67% ranked RSP as their favorite. 67% thought Fractal 3 was the hardest and the other problems tied at 50%.

Scenario	Average
BT	33.3
TT RPP	28.3
TT RSP	24

Table 3: Average time in minutes to complete by scenario. BT = Blockly Turtle and TT = Turtle Tango

8 Future Work

The results of the Turtle Tango pilot were promising, indicating it is a strong candidate technology for remote collaboration. The success of RSP over RPP seems to indicate that simultaneous editing might be more efficient than traditional pair programming, but further study is warranted to measure its long term effectiveness.

While Turtle Tango exemplifies the possibilities for remote collaboration with visual programming, it is not without limitations. The Turtle Tango prototype ultimately focuses only on turtle drawing exercises, but foreshadows the potential for an all-purpose collaborative browser-based development environment. UC Berkeley's Snap*!* development team plans to debut shared file access on its cloud infrastructure in Summer 2015 and be the first visual programming language to integrate Turtle Tango's collaborative editing approach.

9 Conclusion

Turtle Tango is a prototype that demonstrates how to build an easy-to-use, collaborative blocks-programming editor with video and audio sharing. It aims to encourage the development of new applications to support wider adoption of RPP within MOOCs and traditional classrooms alike. However, even Turtle Tango is a mere subset of a full-fledged development environment and requires further enhancement to be a more general purpose collaborative visual programming language editor.

RPP in MOOCs is still a very novel idea with little existing research. Current pair programming research have yet to uncover the unusual phenomenon of passive screenshot drive-bys, which suggests strongly that further study is needed. RPP with a visual programming language is still a vastly unexplored

topic, but becoming more and more relevant as the introductory computing classroom expands to the global scale.

References

[1] Blockly Games: Turtle. https://blockly-games.appspot.com/turtle.

[2] Google App Engine. https://cloud.google.com/appengine/.

[3] Google Blockly. https://developers.google.com/blockly/.

[4] Google Drive API. https://developers.google.com/drive/.

[5] Google+ edX Software as a Service Community. https://plus.google.com/communities/101007836695292894562.

[6] Google Realtime API. https://developers.google.com/drive/realtime/.

[7] PeerJS. http://peerjs.com.

[8] Skulpt. http://www.skulpt.org.

[9] Turtle Tango. https://tango-with-code.appspot.com.

[10] Baheti, P., Gehringer, E. F., and Stotts, P. D. Exploring the efficacy of distributed pair programming. In *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002* (London, UK, UK, 2002), Springer-Verlag, pp. 208–220.

[11] Ferenstein, G. Study: Massive online courses enroll an average of 43,000 students, 10% completion. *TechCrunch* (2014).

[12] Flor, N., and Hutchins, E. Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance. In *Empirical Studies of Programmers: Fourth Workshop*, Papers. 1991, pp. 36–64.

[13] Joseph, S. Pair Programming on Air. https://courses.edx.org/courses/BerkeleyX/CS_CS169.1x/1T2014/77935ad32a3943d6b7a177065233358d.

[14] Joseph, S., and McKinsey, J. Remote Pair Programming Survey Analysis. http://www.agileventures.org/remote-pair-programming/analysis/.

[15] Kerr, N. L., and Bruun, S. E. Dispensability of member effort and group motivation losses: Free-rider effects. *Journal of Personality and Social Psychology 44*, 1 (1983), 78–94.

[16] McDowell, C., Werner, L., Bullock, H., and Fernald, J. The effects of pair-programming on performance in an introductory programming course. *SIGCSE Bull. 34*, 1 (Feb. 2002), 38–42.

[17] McDowell, C., Werner, L., Bullock, H. E., and Fernald, J. The impact of pair programming on student performance, perception and persistence. In *Proceedings of the 25th International Conference on Software Engineering* (Washington, DC, USA, 2003), ICSE '03, IEEE Computer Society, pp. 602–607.

[18] McKinsey, J., Joseph, S., Fox, A., and Garcia, D. D. Remote pair programming (RPP) in massively open online courses (moocs) (abstract only). In *The 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14, Atlanta, GA, USA - March 05 - 08, 2014* (2014), p. 725.

[19] Nosek, J. T. The case for collaborative programming. *Commun. ACM 41*, 3 (Mar. 1998), 105–108.

[20] Shah, D. MOOCs in 2014: Breaking Down the Numbers. *EdSurge* (2014).

[21] Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. Strengthening the case for pair programming. *IEEE Softw. 17*, 4 (July 2000), 19–25.

[22] Williams, L. A., and Kessler, R. R. All I really need to know about pair programming I learned in kindergarten. *Commun. ACM 43*, 5 (May 2000), 108–114.