Let's Work Together: Improving Block-Based Environments by Supporting Synchronous Collaboration

Jennifer Tsan*, Fernando J. Rodríguez†, Kristy Elizabeth Boyer†, and Collin Lynch*
*Department of Computer Science
North Carolina State University, Raleigh, NC 27606
Email: {jtsan, cflynch}@ncsu.edu
†Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida 32611
Email: {fjrodriguez, keboyer}@ufl.edu

***Abstract*—Block-based programming environments are a good way to teach beginners how to code, in part because they eliminate syntax errors and provide visual feedback. However, many of the existing environments do not explicitly support synchronous collaboration. Collaboration is a critical component of computer science practice and CS education. We therefore argue that features to support collaboration could significantly enhance existing and new block-based programming environments. We review existing block-based programming environments, suggest design ideas for supporting synchronous collaboration, and evaluate environments that currently support some of these features.**

I. Introduction

Block-based programming environments are becoming increasingly common in introductory programming courses. These environments have been used for many age groups in both formal and informal settings [1], [2]. Research has shown that integrating collaboration into learning activities can foster good practices, such as teaching concepts to less experienced peers and actively contributing to the solution of a given task [3]. Unfortunately, most block-based programming environments do not explicitly support synchronous collaboration. In this paper, we review existing programming environments that support synchronous collaboration—either co-located or remote—, focusing on computer or desktop applications, and suggest design ideas for future systems.

II. Collaboration in Computer Science Education

Collaboration plays an important role in fields such as computer science, and many curricula encourage the use of collaborative activities in the classroom. Prior research has shown that collaboration can have a positive impact on students' learning experience. In computer science classrooms, approaches such as peer instruction [4], process-oriented guided inquiry learning (POGIL) [5], and pair programming [6] encourage students to work together on computer science tasks, providing opportunities for novices to learn from their more experienced peers and for advanced students to solidify their content knowledge by explaining it to others [7]. Students also routinely collaborate with TAs and instructors in debugging, modifying, or analyzing their code. Prior research has shown that this one-on-one tutoring can be more effective than traditional classroom instruction [8].

Our research group studies one of these approaches: *pair programming*. In pair programming, two developers collaborate on a single coding task. The collaborators work by sharing a single computer or utilizing screen-sharing software. The programmers generally alternate between taking the roles of driver and navigator. The *driver* has control of the computer and enters the code, while the *navigator* provides suggestions and feedback. The programmers switch roles over the course of the collaboration. Prior research has shown that using pair programming in computer science classrooms can improve retention even for non-majors [9], [10] and that it can promote good programming practices such as code-planning [11].

III. Collaboration in Block-Based Environments

One common approach to teaching computer science to programming novices in classroom settings is to use block-based programming environments. Some of the most well known environments are Scratch [12], Snap! [13], Alice [14], and Blockly [15]. While some of these environments include features to support asynchronous collaboration (e.g. the Scratch online community), they can support synchronous collaboration when combined with other technologies.

In the Scratch online community, users can share their projects and comment on each others' work. The environment also includes a feature called the "backpack" where users can store code, sprites, and backdrops, which is helpful when merging projects with their partners. Scratch users often engage in remixing, that is modifying projects that other users have shared [16]. The Snap! community has its own discussion board within the Scratch forum, where programmers can ask questions, report bugs, and help their peers with Snap!-related problems. Both Scratch and Snap! allow users to import and export code, sprites, and background images, making it easy to share code. We suggest that this support could be enhanced by providing explicit scaffolding for synchronous collaboration, which would avoid the time-consuming steps involved with asynchronous sharing.

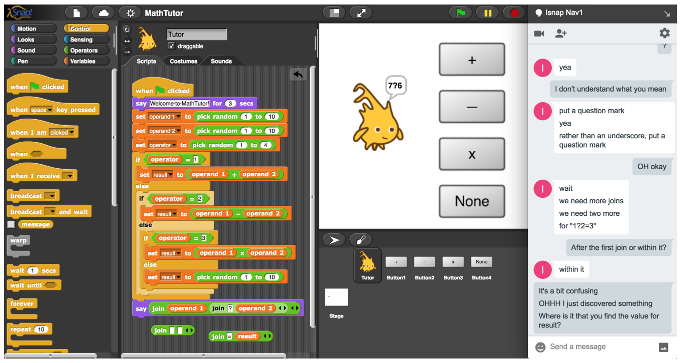
Alice is a block-based programming language based on Java. It is commonly used to teach younger students how to code animations, games, and interactive narratives. Alice has an online forum for users, which includes a section for educators. The Alice community also provides a listserv for teachers to quickly answer each others' questions and start discussions regarding lesson plans and activities. Similar to Scratch and Snap!, we argue that Alice could be improved if it included features that supported remote users who would like to work together synchronously.

Google's Blockly [15] is a code library that supports users in implementing their own block-based languages. A number of block environments built on Blockly have been released. Two of the most popular implementations are App Inventor and BlockPy. App Inventor is designed to support rapid development of mobile applications. The language abstracts away a number of the communication tasks associated with app development. The developers have also provided a forum for users to seek advice. BlockPy [17] is a hybrid block and text-based environment that allows users to develop applications either in block form or with Python code, and it supports users in translating between the two languages. BlockPy is relatively new, so a strong online community has not developed yet. The Blockly library provides features that can easily be used to facilitate remote collaboration, such as Mirrored Blockly, which supports screen sharing. To our knowledge, however, none of the existing blockly-based languages have made use of these features.

IV. Problems Encountered during Pair Programming

As part of our research on pair programming, we have worked with groups of students at the elementary and undergraduate levels in order to understand the issues that they face when engaging in pair programming. We have found several obstacles that limit the effectiveness of the process.

First, when elementary-school students engage in pair programming they often fail to balance time equally between roles. This can lead to students struggling with one another to gain control of the mouse and keyboard. Second, when students reference something on the screen or point at the screen with their fingers, it is not always clear to their partner what they are discussing. This can be frustrating for students on both ends because one student will feel ignored while the other may feel misled or confused. Third, as part of our studies with undergraduate students, we have investigated the limits of existing collaborative tools. In a prior study, we facilitated collaboration by requiring drivers to share their screens via Google Hangouts and to communicate via a Google Chat window. Figure 1 shows what the driver's screen looked like during the study. The drivers coded using the Snap! programming environment and communicated with the navigators through the chat window. Navigators also had access to the task instructions in addition to the shared screen and chat



?

yea

I don't understand what you mean

put a question mark
yea
rather than an underscore, put a question mark

OH okay

wait
we need more joins
we need two more
for "1?2=3"

After the first join or within it?

within it

It's a bit confusing
OHHH I just discovered something
Where is it that you find the value for result?

Fig. 1. Snap! interface (left) with Google Chat window (right).

window. This arrangement forced each student to take a single role (driver or navigator) throughout the entire activity. Several of the students in this study also mentioned that they wished to point directly to particular parts of the code to let their partner know what code segment was being discussed. While the drivers could highlight code via their mouse, the navigators had no comparable mechanism.

V. Design Recommendations

Based upon our research and our examination of prior systems, we make the following recommendations for future implementations of block-based environments.

A. *Collaborative Features*

- **Shared Workspace**: In order to collaborate on a program remotely, all users require access to the shared program. This can be done by allowing a user to create programs and invite collaborators to edit them together.
- **Synchronous Update**: The collaborative programming environment should have the ability to update the program synchronously across all collaborators. This will allow users to follow each others' changes without any confusion due to lag.
- **Multiple Mouse Pointers**: When users are collaborating remotely, they may need to reference certain blocks of code. By giving each user the ability to point to or highlight blocks, the confusion about which blocks are being discussed will be reduced.
- **Different Modes of Control**: By locking code or even limiting direct interaction to a single user, we can use the system to enforce role taking and support tutorial interactions.
- **Synchronous Editing**: Although it may be useful to have a mode that enforces roles in the programming environment, there are also cases where it may be useful for multiple programmers to edit different parts of the same program synchronously in order to complete tasks more efficiently.
- **Planning Space**: Some existing collaborative environments include a 'whiteboard' area for planning and brainstorming that does not affect code. One such example

is shown in Figure 2 below. Prior work has shown that users may move blocks within the programming interface without connecting them as a form of planning [18], and a whiteboard area may help them better organize their thoughts and communicate their ideas to their partners.
- **Communication Channel**: Whether the users are working next to each other or from opposite sides of the world, communication is one of the most important aspects of collaboration. Therefore, environments should offer a way for users to communicate while working remotely. This could include textual chat, voice chat, and video.

B. *Designing for Data Collection*

The preceding design requirements focused on user-facing functionality. In order to support research on collaboration in computer science, it is also important for programming environments to have the ability to automatically log collaboration data. In order to support such research, the following data could be collected:

- **Programming Actions**: Examples of these actions include block creation or deletion, switching between block categories, running the program, and switching between sprites or scripts. These actions will help researchers identify compare common programming patterns different pairs of users follow.
- **Dialogue**: Analysis of the dialogue (typed or spoken) between collaborators can provide information about how students communicate their ideas to their partners and how they approach the problem-solving tasks together.
- **Videos of Collaboration**: Analysis of the users' physical actions (e.g., pointing) and posture can give us insight into their engagement and affective state, as well as how they communicate using body language in face-to-face interactions.
- **Other Interface Actions**: Other important actions in the interface, such as mouse highlighting and drawing in the whiteboard, should be recorded for analysis of how users communicate and plan.

C. *Challenges of Designing a Collaborative Block-Based Environment*

While there are many benefits to supporting collaborative problem solving in block-based environments, there are also design challenges that should be considered, especially when these environments are used in the classroom:

- **Logistical Overhead**: Some environments require the installation of a local program or specialized browser plugins, which may not be possible for students or instructors in a controlled school environment.
- **Data Collection & Privacy** Other environments require users to create accounts which often require an email address or other separate point of contact to confirm the account. This step, may present logistical difficulties for students who lack private email. And it may violate district or school policies regarding data privacy particularly where younger students are concerned.
- **Connection Reliability**: With any online environment, it is important to consider the challenges that may arise with network reliability. This is especially important when designing for a synchronous collaborative environment. Schools vary in their connectivity, and high-bandwidth connections cannot be assumed. System designers should consider alternative methods to support unreliable environments such as allowing for asynchronous updates or by minimizing network demand. Unfortunately, this problem is non-trivial and typically takes years to solve.

VI. Synchronous Collaboration Features in Current Programming Environments

We found and reviewed block- and text-based programming environments that were designed to support synchronous collaboration. While we were only able to find two block-based environments with such features, we found many text-based environments that have been developed for the purpose of synchronous collaboration. From all of the text-based environments we have found that explicitly support collaboration, we chose some of the most highly cited environments to review.

A. *Block-based*

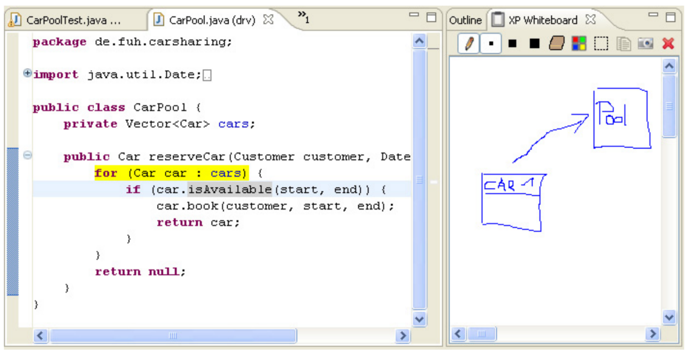
AliCe-VilLagE [19] is built on the Alice code base. In this environment, roles are enforced by restricting edit privileges to the driver. Changes are displayed to all users in real-time, and the users can chat with each other via text or video calls. Navigators may add comments to the code. Initial results suggest that using AliCe-VilLagE is more efficient than using Alice with traditional methods of communication (e.g., emailing or text) [20]. While AliCe-VilLage has many of the features that are important to support collaboration, it does not include multiple mouse pointers or a planning space.

One of the more recent environments is NetsBlox [21]. NetsBlox was built using the Snap! code base and it allows users to build distributed applications, and to synchronously modify a program from separate computers. These collaborative features were only recently added to the interface, so further studies are needed to assess their effectiveness.

B. *Text-based*

Codechella [22] is an online Python programming environment that supports collaboration. In it, users can modify code simultaneously, chat with each other through text, jointly visualize the execution state, and see each other's mouse clicks. An review of collaborative dialogue from an initial study suggests that users enjoyed Codechella features [22]. Missing features include different modes of control and providing a planning space for users.

Saros allows for collaborative programming between multiple users [23]. It is a plug-in for Eclipse that includes features designed to help navigators. When users select text, their selection is highlighted in different colors, and the position of each user in the text is identified. The editor also highlights the last twenty edits made by the driver and includes a "follow-mode," which allows the navigator to see the same view as



```
package de.fuh.carsharing;

import java.util.Date;

public class CarPool {
    private Vector<Car> cars;

    public Car reserveCar(Customer customer, Date
        for (Car car : cars) {
            if (car.isAvailable(start, end)) {
                car.book(customer, start, end);
                return car;
            }
        }
        return null;
    }
}
```

Fig. 2. Screenshot of XPairtise.

the driver. Navigators can disable this mode at any time to traverse the code independently. Saros does not include a communication channel or a planning space.

XPairtise [24] was developed specifically for distributed pair programming. It includes many of the features found in other tools as well as novel features, such as the integration of Skype for verbal communication, and a whiteboard space (displayed in Figure 2) where users can use their mice to sketch ideas. Additionally, XPairtise allows users to join as a "Spectator, which may support tutoring in a distributed setting" Spectators may not make any changes to the code, but they can contribute to the conversation. XPairtise is the only system we reviewed that includes a whiteboard.

VII. Conclusion

Collaboration is essential to computer science and has been shown to be helpful for learning. Block-based programming languages and environments are great for beginners; however, they generally lack features that explicitly support synchronous collaboration. In this paper, we surveyed existing block-based systems, discussed the collaborative affordances of each one, and presented specific recommendations for features that should be included. The implementation of such features will enhance the utility of these tools by increasing the ease of programmer collaboration. It has the potential to impact the way students learn to program and the ways developers approach their development process.

Acknowledgments

The authors thank the anonymous reviewers for their insightful comments. This work is supported in part by the Wake County Public School System, the National Science Foundation through grants CNS-1453520, DRL-1721160, CNS-1622438, DUE-1625908, a Graduate Research Fellowship, and Google through a CS Capacity Research Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] B. Xie and H. Abelson, "Skill Progression in MIT App Inventor," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) Proceedings*, 2016, pp. 213–217.

[2] G. Ota, Y. Morimoto, and H. Kato, "Ninja Code Village for Scratch: Function Samples/Function Analyser and Automatic Assessment of Computational Thinking Concepts," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) Proceedings*, 2016, pp. 238–239.

[3] D. L. Jones and S. D. Fleming, "What Use Is a Backseat Driver? A Qualitative Investigation of Pair Programming," in *2013 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) Proceedings*, 2013, pp. 103–110.

[4] C. H. Crouch and E. Mazur, "Peer Instruction: Ten Years of Experience and Results," *American Journal of Physics*, vol. 69, no. 9, pp. 970–977, 2001.

[5] J. J. Farrell, R. S. Moog, and J. N. Spencer, "A Guided Inquiry General Chemistry Course," *Journal of Chemical Education*, vol. 76, no. 4, pp. 570–574, 1999.

[6] L. Williams, "Integrating Pair Programming into a Software Development Process," in *Proceedings of the 14th Conference on Software Engineering Education and Training (CSEE&T '01)*, 2001, pp. 27–36.

[7] M. T. H. Chi, N. de Leeuw, M.-H. Chiu, and C. LaVancher, "Eliciting Self-Explanations Improves Understanding," *Cognitive Science*, vol. 18, no. 3, pp. 439–477, 1994.

[8] 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.

[9] L. Porter and B. Simon, "Retaining Nearly One-Third more Majors with a Trio of Instructional Best Practices in CS1," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*, 2013, pp. 165–170.

[10] C. O'Donnell, J. Buckley, A. E. Mahdi, J. Nelson, and M. English, "Evaluating Pair-Programming for Non-Computer Science Major Students," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*, 2015, pp. 569–574.

[11] Z. Li and E. Kraemer, "Social Effects of Pair Programming Mitigate Impact of Bounded Rationality," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*, 2014, pp. 385–390.

[12] Scratch. [Online]. Available: https://scratch.mit.edu/

[13] Snap! [Online]. Available: http://snap.berkeley.edu/

[14] Alice. [Online]. Available: http://www.alice.org/

[15] N. Fraser. (2013) Google Blockly. A Visual Programming Editor. [Online]. Available: https://developers.google.com/blockly/

[16] S. Dasgupta, W. Hale, A. Monroy-Hernández, and B. M. Hill, "Remixing as a Pathway to Computational Thinking," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing (CSCW)*, 2016, pp. 1438–1449.

[17] A. C. Bart. (2017) BlockPy. [Online]. Available: http://think.cs.vt.edu/blockpy/

[18] F. J. Rodríguez and K. E. Boyer, "Discovering Individual and Collaborative Problem-Solving Modes with Hidden Markov Models," in *Proceedings of the 17th International Conference on Artificial Intelligence in Education (AIED '15)*, 2015, pp. 408–418.

[19] A. Al-Jarrah and E. Pontelli, ""AliCe-ViLlagE" Alice as a Collaborative Virtual Learning Environment," in *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, 2014, pp. 1–9.

[20] A. Al-Jarrah and E. Pontelli, "On the Effectiveness of a Collaborative Virtual Pair-Programming Environment," in *International Conference on Learning and Collaboration Technologies*, 2016, pp. 583–595.

[21] B. Broll and A. Ledeczi, "Distributed Programming with NetsBlox is a Snap!" in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*, 2017, pp. 640–640.

[22] P. J. Guo, J. White, and R. Zanelatto, "Codechella: Multi-User Program Visualizations for Real-Time Tutoring and Collaborative Learning," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) Proceedings*, 2015, pp. 79–87.

[23] S. Salinger, C. Oezbek, K. Beecher, and J. Schenk, "Saros: An Eclipse Plug-in for Distributed Party Programming," in *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*. ACM, 2010, pp. 48–55.

[24] T. Schümmer and S. Lukosch, "Understanding Tools and Practices for Distributed Pair Programming," *Journal of Universal Computer Science*, vol. 15, no. 16, pp. 3101–3125, 2009.