Collaboration Station: Opening up Single-User Software Projects — I-Test & CSforAll

Abstract

The need for collaborative software is more significant than ever in our modern world. Especially in large software companies, it becomes imperative to work efficiently with co-workers to complete large projects. Consider that nearly seven percent of Americans between ages six and eleven have been diagnosed with neurodivergency [1]. Some of these individuals will end up becoming software developers. The problem, though, is that many of these students will not have the practice of effectively collaborating while coding. Scratch, one of the most ubiquitous block-based software tools that aims to teach students basic programming practices, does not support multi-user collaboration[1]. As such, reverse-engineering single-user web programming applications to multi-user applications could help younger students–especially those with neurodivergent social behaviors–learn good collaborative practices early. Moreover, the development of this tool allows a unique case study into the implementation of multi-user features in closed single-user systems and the challenges faced in implementing such a software.

In this paper, we demonstrate the process of developing the software that we built for a summer camp related to teaching around 20 neurodivergent high school students programming concepts under the funding of NSF's Division Of Research On Learning and ITEST. We elaborate on the challenges and potential issues of creating and making such software easily accessible. More specifically, the synchronization problems that arise from turning a closed single-user system into a multi-user system for a neurodivergent programming camp. Additionally, we discuss about the iterative and real-time feedback development of our tool.

Introduction

The importance of collaborative tools in fostering an early software development experience is undeniable. Exposing students to these tools is a great way to introduce these concepts at a stage where many students make their most critical collaborative experiences [2]. However, neurodivergent students often have social impediments that may make it difficult for them to be receptive to working on code or other projects they may deem as personal with their peers [3]. The development of Collaboration Station aimed to create an easily accessible and navigable website online where students are able to collaboratively program, much akin to an application like Google Docs. Ashinof and Abu-Akel assert that the goal of good software should be to minimize the excess "clutter" of the page [4]. Moreover, Kokinda *et al.* emphasize that providing familiar modes of engagement can help promote collaboration and communication between students [5]. Collaboration Station's development strongly follows this principle.

Design Process

This philosophy of keeping Collaboration Station self-contained manifested in many ways. The end product not only synchronized Scratch, but also contained a chat and video bar on the right to

[1] https://scratch.mit.edu

mitigate the usage of other chat media. We wanted to also synchronize an art or music generating tool. However, time did not permit this. The only real unique consideration outside of the main "room" screen was determining how to allow students to join rooms. To simplify the process, users can share a link, enabling collaborators to edit together in a specific room.

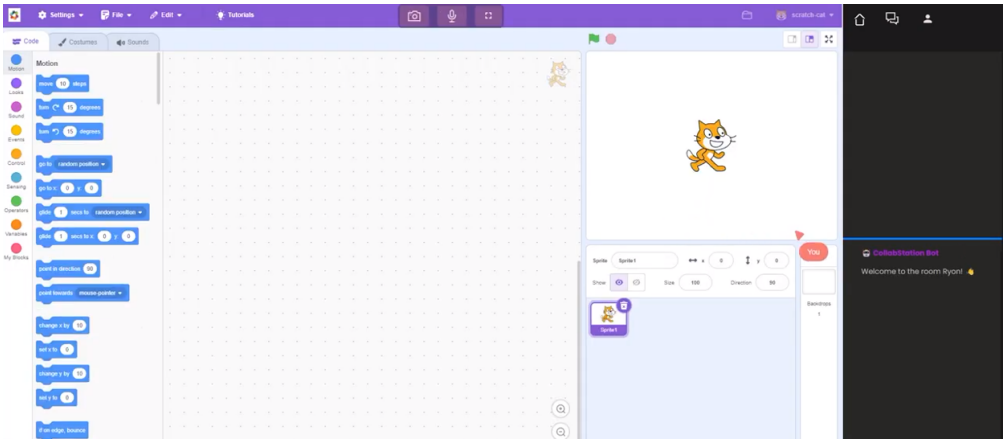

Figure 1: Collaboration Station Main Screen

Understandably, one of the largest barriers to building the collaboration portion of Collaboration Station is synchronization. In this context, synchronization refers to the method where one person may change the state of their project by dragging a block or changing the position of a character. To approach the problem as a potential developer, the reverse engineering process of software almost always begins with background research on existing modifications. Specifically, we examined how other similar add-ons managed to "hack" the tool to synchronize Scratch. For instance, a previous iteration of the camp utilized a Chrome extension called "Blocklive[2]," which added multi-user functionality to Scratch. The code for Blocklive is available directly on GitHub as open-source material. While we did not want to rely on the exact same code—primarily due to the overhead of requiring campers to download an extension and our preference to store information on our servers—we aimed to adapt the overarching mechanisms of Blocklive's approach for own software.

Synchronization

There are numerous ways to mimic this synchronization functionality. For smaller projects, the system can transmit the entire project state after each state change. While relatively easy to implement, this approach requires storing the project in a minimal amount of space, as users can frequently cause state changes, resulting in lots of transmissions and high network overhead. Alternatively, similar to code-sharing services like Git, changes to the project's current state can be transmitted instead [6]. The program might broadcast signals called "events" to each connected user. Each connected user can then parse each event and update the state of their program. However, this method requires significantly more time to implement because it must broadcast each event in a way that allows unpacking and conversion into any possible state change.

[2] A browser extension for collaborating on Scratch. https://sites.google.com/view/blocklive/home

To implement synchronization in Collaboration Station, we had to consider the size of individual Scratch projects. A base Scratch project with no sprites (images) requires roughly 1.5 kilobytes of data, with every additional sprite adding an extra kilobyte. Code Blocks and custom images add even more. Detailed in the following sections, the service we used to send data between connected users, Ably[3], would not be able to send packets over a few kilobytes. To keep the data transmitted within the limit, Collaboration Station instead had to utilize an event-based approach to its synchronization.

Blocklive served as a useful reference codebase. It cannot read internal code and instead uses an event-based approach. It employs a "blackbox" system, intercepting events generated by the application and modifying them before passing them to the rest of the program. A benefit of this approach is that it allows for quick prototyping. Since most of the mechanics of Scratch are already programmed in, we can simulate events and pass them off to the scratch renderer. Reading Blocklive's code is very beneficial in this circumstance as we rarely have to locate the exact bits of code that may act but find the event Blocklive transmits to perform that action. This is especially helpful in large codebases like that of Scratch, where bits of code can be buried deep within files. Instead of finding the exact location of this code, we can simply pass an event to Blocklive and trace its path.

A specific sector in which pathing became useful was the most critical component of this development process: synchronizing the block code between users. Verifying this is important as each user should have the same blocks in the same configurations; a misalignment in blocks or block order would cause the program code to be different for each user. This would result in different behaviors in users' projects. Of special importance is the way Scratch blocks work – code logic is dictated through a vertical "stack" of blocks, where each block has its next block and previous block stored. This allows it to execute all blocks in order from top to bottom when the user clicks the run project button.

However, it is essential in this instance that blocks attached to the end of another block chain require the block ID of the blocks above it. This means that each "event" call must be sent at precisely the same relative time as other messages – as if a message of an ending block referenced a block chain that was not sent yet, it would not attach and the code would never reach that block. Therefore, the project had to utilize a messaging system that could send messages at specific times and in a queue.

A benefit of using Ably was that packet sizes were not of a significant concern. This allows us to batch messages together. Of special note is that Scratch can treat certain events as separate messages. For example, when a user drags a block from the block creation panel to their Scratch workspace, two events are created: one for the block creation and one for the block move to the end position. However, if we synchronize both events at the time they were respectively generated, people connected to the room would see a block randomly appear at the workspace origin and then snap into its end position a second later. There are a lot of small instances of this, where the solution would be to "batch" messages by sending a list of events to be synchronized simultaneously. This also controls how animations can get done; if we know the current position and the following events that a block may end up going to, we can ease between them to reduce jarring jumps in visuals.

[3] https://ably.com/

Another key issue is project preservation. Unlike our model Git, each project state must be the same for reasons described earlier regarding project flow. While this would not be an issue if each synchronization event worked exactly perfectly, the camp was running under the assumption of eventual bugs. Therefore, some users might have slightly different projects than others. So, our goal was to have backups of different users so we could roll a project back to a specific user that may have the least amount of bugs. The way Collaboration Station handled this is by cycling through each camper of a given project and uploading a copy of their file to a central server every minute if a change had been made. Since the central server kept all backups given to it, this guarantees that there would be a copy of the project by each user every few minutes. That way, if an error occurs (of which there were many) where a file becomes otherwise corrupted, a rollback can occur to a different camper's save file. Since a copy is uploaded every minute, a new user that joins between the minute window may not have the most up-to-date project. In this instance, a request to push the latest version is sent to the next queued user before the new user pulls the saved file from the server. Essentially, this copies the save from the next queued user.

Design Reflection

We utilized the first iteration of the camp to act as a debugging and information-gathering tool to test Collaboration Station while it was still in development. This process led to interesting design choices and information gathered. For instance, when users accessed the webpage, they would input solely an email with no password and then proceed to log in. While this was part of our minimalistic-focused approach, we also wanted to test whether this would cause problems with campers logging in to other accounts and whether this was feasible more generally. In the end, no apparent problems stemmed from this addition. While future camps and implementations will certainly have passwords, it is still interesting to note that this camp did not need them. There were also other common bugs that arose from a large group of students utilizing the page at the same time, such as slowdowns and missed synchronizations. However, other bugs gave more insight into how campers themselves think differently from developers. For instance, a few campers were unaware they could move a sprite, object, or character in a project by dragging it on the virtual screen. So instead, they dragged a "move 10 step" block and repeatedly clicked it until the sprite trudged to where the user wanted it to be. If another user modified a block during these sustained synchronizations, it would cause data loss in block order.

The default Scratch Graphical User Interface (GUI) previously had extensive research done into creating software that is easily accessible. We opted then to maintain a similar design to keep accessibility features and prevent mapping mismatches from students who may have used Scratch previously. However, we did make minor changes, such as improving the ease of pressing specific menus and buttons, including the newly added "Save Project" button. Interestingly, though, we noticed some repercussions arise from this. Namely, there was an incident where a camper was trying to import their art to Collaboration Station. However, thinking the rest of the pre-existing art was unnecessary, the camper swiftly deleted all the other sprites. In this case, the camper seemed to "hyperfocus" on the creation of the art, as some neurodivergent people often do when presented with a task that interests them [7]. The camper was nearly entirely focused on their individual contribution to the project and did not consider the implications for everyone else working. For instance, adding hurdles to reduce misclicks like these could be beneficial in this case. As a quick fix, we restricted the deletion ability of sprites to the person who created the

sprite. While it does not remove this problem entirely, it reduces the impact and probability of it occurring by someone who does not directly contribute to the specific sprite.

Discussion & Conclusion

There are certain aspects of Collaboration Station that, given time, should probably have been changed. Implementing a collaborative drawing tool to draw sprites inside of Collaboration Station as well as a music tool would significantly negate the need to use the outside internet during camp. The internet could then primarily be used as references for campers. Another consideration is implementing more features focused on including more neurodivergent-safe compatibilities. For instance, running a possible case study on whether ”disabling” recently moved blocks for campers other than the original creator of the block might aid in reducing accidental deletes or moves that could annoy certain campers.

Overall, Collaboration Station's development represents a step forward to creating accessible, collaborative programming environments for neurodivergent students. The case study in opening up Scratch is an interesting guide to more broadly reverse-engineering software and establishing synchronization tactics. As programming continues to become increasingly collaborative, easily accessible applications like Collaboration Station could play a potential role in helping neurodivergent students develop both technical and social skills that will be essential for their future careers in software development.

Acknowledgements

The authors thank NSF's Divison of Research on Learning in its effort of promoting for its role in fostering this research under NSF Grant #2148720: ”Preparing High School Students with Autism for the Future of Remote Software Development Work.” under NSF programs I-Test and CSforAll.

References

[1] T. Armstrong, *Neurodiversity: Discovering the extraordinary gifts of autism, ADHD, dyslexia, and other brain differences*. ReadHowYouWant.com, 2010.

[2] C. Crook, “Children as computer users: The case of collaborative learning,” *Computers & Education*, vol. 30, no. 3-4, pp. 237–247, 1998.

[3] M. R. SASPORTES, “Challenges and opportunities for neurodivergent software engineers,” 2024.

[4] A. A. D. BIA, “A creativity based goal modeling approach for accessibility of neurodivergent individuals,” 2023.

[5] E. Kokinda, M. Moster, P. Rodeghero, and D. M. Boyer, “Informal learning opportunities: Neurodiversity, self-efficacy, motivation for programming interest.,” in *CSEDU (2)*, pp. 413–426, 2024.

[6] C. Bourke, “Introduction to git,” 2015.

[7] B. K. Ashinoff and A. Abu-Akel, “Hyperfocus: The forgotten frontier of attention,” *Psychological research*, vol. 85, no. 1, pp. 1–19, 2021.