Incorporating real world non-coding features into block IDEs

Mikala Streeter

Georgia Institute of Technology
Atlanta, GA

***Abstract*—We often see block-based coding environments as toy environments that let novice programmers have fun as they learn the basics of programming. While these environments do have an engaging low floor, they are missing out on other aspects of introductory programming that could further engage students and better replicate the real work of software developers. In this paper, I describe real world non-coding features (e.g. collaboration, code reviews, version control, aesthetic appeal, galleries/community) that should be incorporated into block IDEs. There are likely other important non-coding features that are not included.**

Keywords—blocks; programming; collaboration; code reviews; version control; aesthetic appeal; galleries; novice

I. Introduction

We often see block-based coding environments as toy environments that let novice programmers have fun as they learn the basics of programming. While these environments do have an engaging low floor, they are missing out on other aspects of introductory programming that could further engage students and better replicate the real work of software developers. In this paper, I describe real world non-coding features (e.g. collaboration, code reviews, version control, aesthetic appeal, galleries/community) that should be incorporated into block IDEs. There are likely other important non-coding features that are not included.

II. Collaboration

In introductory courses, we often enlist pair programming as a low risk way for students to create projects. This approach gives each student a buddy with whom they can discuss ideas, write code and fix bugs. However, an issue that often arises is when students want to continue working on their projects at home or when their partner is sick. The project was likely created under the account of one student assuming that she'd be there each day for class, but when she's not, if the teacher hasn't written down the student's login information, her partner will either lose a day of work or will have to rush to re-create all of their work. If block IDEs allowed students to add collaborators to a project like professional software developers can do through tools like Github, then this issue would never occur. Both students would have easy access to their shared code because they are both collaborators on the project. Students would also be able to work in larger groups of 3 to 4 people on multiple computers, creating more complex code and experiencing what it's like to work on a real team of software developers building something amazing together.

III. Code Reviews

Another key aspect of collaboration and learning at the introductory level is getting feedback on the quality of your code. In industry, professionals do this through code reviews, which can occur both synchronously and asynchronously. By focusing primarily on the changes made from the previous version, teams can give one another in-line feedback. While computer programs give students immediate feedback on how their code works, a real person needs to intervene to give feedback on the elegance and efficiency of the code. Adding the ability for reviewers and collaborators to do code reviews of the latest changes by tying comments to the new blocks or scripts would speed up grading for educators, decrease wait time for students wanting feedback, and enable asynchronous mentoring of students by industry professionals.

IV. Version Control

In professional work, programmers are able to take deep dives into rabbit holes with their code because they know that their versioning software will easily allow them to backtrack. For them, this flexibility is useful for intentional experimentation with coding approaches. For introductory students, versioning would also be useful albeit for unintentional experimentation, where they thought they had enough time or understanding to try a complex approach, only to later realize that it wasn't the best course of action, but now have gone too far down this path to Undo their work. If block IDEs were

built with version control, preferably through auto-save, students could much more safely experiment knowing that they can go back to an earlier working version with ease. Version control would also enhance code reviews in the event that students are given feedback that the latest version of code is quite poorly written and should be reverted. Version control would additionally support collaboration to simplify merging code from multiple students.

V. Aesthetic Appeal

While some introductory students try computer science because they're interested in the technical challenge or the puzzles, some were intrigued by the potential to make the beautiful mobile apps and video games they use everyday. They're excited about incorporating their own creativity and design eye to their coding work, only to find that this isn't a priority in many block IDEs. These students may feel that their love of design isn't valued in the technical world and be turned off from the field altogether or at a minimum, forgo the possibility that their interests in art and technology can intersect, possibly putting them on an outbound trajectory in the long run. If block IDEs were designed, perhaps like Twitter Bootstrap that makes even the simplest websites beautiful, to ensure aesthetic appeal as students learn to code, beginner students would see how even as novices they can create work that is both complex and beautiful.

VI. Galleries/Community

Scratch and OpenProcessing have shown us the power of worked examples in programming communities. There is a rare day that goes by in my introductory classes that I don't encourage a student to search for an example in either community. These galleries inspire new students about what's possible as they develop their skills and also validate their work as they receive comments and tweaks/remixes from others in the community. And yet, new block IDEs are being built with galleries and community as secondary or even tertiary features. If our focus is on broadening participation, we need to make every student feel as though as they are a part of something much larger than themselves, and that there are lots of people also learning who are willing to help and encourage them along the way. Communities and galleries go a long way for sending this positive message.

VII. Conclusion

As we teach students to code, we need to keep in mind the importance of enculturating them into a computer science community of practice. This community is about much more than being able to write code. It's about being able to write quality code often with other people over many iterations to build something that works well and looks beautiful. In order for students to determine whether they want to stick with computer science, they have to see the full picture and incorporating these non-coding features into the blocks-based programming environments will go a long way in service of this goal.