How to Catch Novice Programmers' Struggles: Detecting Moments of Struggle in Open-Ended Block-Based Programming Projects using Trace Log Data

Benyamin T. Tabarsi[1], Ally Limke[1], Heidi Reichert[1], Rachel Qualls[2], Thomas Price[1], Chris Martens[1], Tiffany Barnes[1]
[1]North Carolina State University
[2]The University of Alabama
[1]{btaghiz,anlimke,hreiche,twprice,crmarten,tmbarnes}@ncsu.edu
[2]rlqualls@crimson.ua.edu

ABSTRACT

Regardless of skill level and background, programming can be challenging for all students. However, in the early stages of learning, challenges may particularly lead to a decrease in students' sense of self-efficacy and interest in computer science. Hence, finding the moments when novices struggle during programming will help us provide support and intervene at the proper time. Some efforts have been made to find out when students struggle during a specific assignment, but none of them (to our knowledge) have targeted open-ended tasks, i.e., tasks that have no fixed solutions or processes to reach the objective. This study aims to determine how students' coding traces in a block-based programming environment relate to their struggles while completing an open-ended project. We ran a study in an introductory programming course that used a block-based language for the first 8 weeks of a semester-long class, culminating in a 2-week-long project. Students were given class time for two sessions to work on their projects in pairs, during which we collected students' coding traces. Based on experts' hypotheses and two prior studies, we developed detectors to parse coding traces and find struggle moments automatically. We also conducted a survey at the end of each session to ask students about their satisfaction with their programming and feelings when encountering programming challenges for which we defined detectors. We investigated how well moments identified by our detectors associated with students' immediate survey responses. Our results show that students' perceptions of the experienced challenges significantly correlate with detectable patterns in their coding traces.

Keywords

block-based programming, open-ended project, cs education, novice programming, struggling

B. T. Tabarsi, A. Limke, H. Reichert, R. Qualls, T. Price, C. Martens and T. Barnes. How to Catch Novice Programmers' Struggles: Detecting Moments of Struggle in Open-Ended Block-Based Programming Projects using Trace Log Data. In B. Akram, T. Price, Y. Shi, P. Brusilovsky and S. Hsiao, editors, Proceedings of the 6th Educational Data Mining in Computer Science Education (CSEDM) Workshop, pages 57–65, Durham, United Kingdom, July 2022.
© 2022 Copyright is held by the author(s). This work is distributed under the Creative Commons Attribution 4.0 International (CC BY 4.0) license.

1. INTRODUCTION AND BACKGROUND

Programming is a difficult subject to learn for many students and especially novices [6, 17]. While tutoring and professor support can greatly aid students in learning, this help is increasingly limited. Computer Science (CS) enrollment rates have risen significantly since 2006, but professor and faculty staffing has failed to keep up [9]. This has produced elevated student-to-faculty ratios in computer science courses, especially introductory computer science courses where students might need the most support [8]. A novice programmer's inability to access help when needed could be detrimental to their computer science learning and potentially give rise to a higher dropout rate in CS [10]. Gorson and O'Rourke pointed out that in struggling moments, students negatively self-assess their programming ability, leading to lower self-efficacy and thus higher drop-out rates in computer science [4].

This demand-supply imbalance sets off a great need for an intelligent interface to determine when a student needs help or intervention [10]. If struggle moments could be detected early in novice programming, a timely intervention, such as automated help, would benefit students[12, 11]. This has made detecting struggle moments an interesting topic for many researchers to investigate. Some prior studies focused on finding novices' coding characteristics and behaviors by analyzing their coding traces [1, 5]. However, their concentration is not in moments of struggle. Another group of papers focused on finding patterns in coding traces that can specify struggling students based on their grades. To do that, they used machine learning or statistical models to find the relationship between traces and students' grades or performances [3]. Some other papers used the mapping cost or the similarity score between students' coding traces and correct solutions to detect struggle and progress moments or to provide hints and feedback [2, 12, 18].

Our work bears two major differences from the mentioned studies. First, although in previous studies the assignments were open-ended, students were given specific questions that could have multiple solutions with a relatively clear structure. In contrast, we used code traces of students' projects for our detectors, where students were asked to choose a topic and create a program in Snap*!*. Second, our paper does not fully rely on students' grades or experts' reviews to evaluate proposed struggle detectors. We defined struggle detectors based on students' perceptions of negative self-assessment moments and gauged the performance of detectors mainly against students' reported feelings towards each of the mentioned moments. This is particularly significant since being context-sensitive is an important aspect of automated help in Intelligent Tutoring Systems (ITS), especially for open-ended assignments and projects [15]. Also, it is helpful for professors/assistants if they want to determine why their students may be struggling and how to better help them. No struggle detectors, to our knowledge, have been developed and shown to work for open-ended programming projects where every student project can be different.

This work is driven by the following research questions:

- **RQ1:** What detectable patterns in students' coding traces are associated with self-reported struggle moments during doing an open-ended block-based project?
- **RQ2:** How do these detectable patterns correlate with novices' perceptions of the experienced challenges in completing an open-ended block-based programming project?

We propose, create, and test detectors that can identify moments of student struggle when programming in a block-based environment. The final goal of the detectors is to work for any assignment and project in Snap*!*, a block-based programming language. Moreover, we aim to design detectors that not only identify moments of student struggle but also give insight into why the student may be struggling.

2. DATA CONTEXT

2.1 Trace Log Data

The dataset we analyzed comes from an undergraduate introductory computer science course with 45 non-computer science major students at a public university in Spring 2022. Students worked in Snap*!* for the first 8 weeks of a semester-long class. This study was conducted while students were completing their open-ended midterm projects, which required them to create an interactive program in Snap*!* using concepts they learned throughout the semester, such as loops and conditionals. After this project, the block-based portion of the class ended, and students began learning Python. Students were allowed to work in groups of 1-3 people. They had two class periods to work on their project and were expected to finish any remaining work outside of class. We collected code traces from these two days when students were working on their projects in class.

In this course, students programmed their project in iSnap[16], which is a variation of the Snap*!* programming environment. iSnap generates a record for each action that students take in the programming environment, such as adding sprites and clicking on the run button. A set of these records relating to a student or a group, which is also called a student's coding trace, allows researchers to track and analyze students' coding patterns and progress. iSnap also creates a snapshot of the whole environment at specific timestamps that makes it possible to rebuild and replay the steps of a student during a coding period.

2.2 Survey

After each coding session, we administered a survey asking students about their overall satisfaction with their progress that day and self-assessment moments they encountered.The surveys on each day were identical and inspired by a survey originally developed by Gorson and O'Rourke, in which the researchers identified 13 common moments during which students negatively self-assessed their programming abilities in an introductory course [4].

As shown in Table 1, the first portion of the survey asked students, "How did you feel about your programming experience/progress today?" and offered a six-point Likert scale from "Very dissatisfied" to "Very satisfied". The second portion of the survey asked the students if they experienced 15 scenarios and if so, how each made them feel. The first 13 scenarios corresponded to the 13 moments when students self-assessed, as identified by Gorson and O'Rourke. The last two scenarios were the negation of two of the previous self-assessment moments. For each scenario, students could choose between the options of "Didn't happen," "I felt bad/frustrated," "It was normal and I felt OK," and "It was good/useful or helped me learn." We modified the language of the scenarios to fit the perspective of a student using a block-based language instead of a text-based language[4]. Of the entire class, 16 students responded to the survey on the first day and 13 participated on the second day.

2.3 Data Cleaning

We cleaned data via the following three steps:

Step 1: We removed the traces of pairs who did not consent to and complete at least one survey.

Step 2: We matched students' traces to their survey responses. Connecting each survey to its corresponding trace log was highly challenging since students were asked to answer the survey individually but were allowed to work in groups. To handle pair programming, the iSnap environment has two input boxes in which students are asked to enter their name and their partner's name before entering the programming environment. In all cases where a student had multiple traces, the trace in which their name was entered as the first programmer was used to link to their survey responses. Except for one student, all who participated in our study had a code trace in which their name was specified as the first programmer. In that one exceptional case, the student whose name was entered as the partner did not have any separate trace with their name as a first programmer. Therefore, the trace of this pair was allocated to both students.

Step 3: We combined disjointed traces. Some students worked across multiple computers during a class period, resulting in what we call disjointed coding traces. By looking at the timestamps of traces for each student, we specified traces that overlapped in terms of starting and finishing time. Then, by scrutinizing the actions in each trace, we found any code traces that were scratch work, where students were clearly trying out something minor on a separate Snap*!* window but not in the main project. We discarded such scratch traces and put the remaining disjointed traces together based on their timestamps to obtain a complete trace.

Table 1: Survey Questions and Answers about Self-Assessment Moments in Programming

Survey Questions	Survey Answers
How did you feel about your programming experience/progress today?	1. Very dissatisfied 2. Somewhat dissatisfied 3. Mildly dissatisfied 4. Mildly satisfied 5. Somewhat satisfied 6. Very satisfied
Think about your programming experience today. How did you feel if or when you experienced the following things? 1. Got a simple error, like something not working as desired 2. Started over or erased a significant portion of code to try again 3. The code was not working as expected 4. Stopped programming to plan 5. Got help from others, (e.g. partner or TA) 6. Spent a long time working on a feature 7. Felt unsure where/how to start programming 8. Looked up how to do something 9. Spent time planning before starting to program 10. Spent a long time looking for a simple error or mistake 11. Struggled to fix errors and get the program to work 12. Took longer than expected to finish a feature 13. Felt unsure about what to work on 14. Finished a feature quicker than expected 15. Fixed an error faster than expected	1. Didn't happen 2. I felt bad/frustrated 3. It was normal and I felt OK 4. It was good/useful or helped me learn

Finally, these steps resulted in the construction of 19 coding traces matched to the survey responses. These 19 traces comprised aggregately 12262 rows each showing a student's action in iSnap. Table 2 shows the summary of 19 students' demographics.

Table 2: Summary of Students' Demographic Data

Demographic Category		Count	Percentage
Gender	Female	7	36.8%
	Prefer not to say	1	5.3%
	Male	11	57.9%
Race	Asian	1	5.3%
	Black or African American	1	5.3%
	Mixed	3	15.8%
	Prefer not to say	2	10.5%
	White	12	63.2%

3. METHODOLOGY

3.1 Creating Detectors

The goal of our research was to create detectors that identify when students are struggling. We considered seven scenarios that may be indicative of student struggle and created a detector for each of these scenarios. We designed these detectors using Gorson and O-Rourke's research on moments in which students negatively self-assess [4], the research of Dong et al. on analyzing tinkering behaviors [1], and the suggestions of four experts, comprised of a professor who is a CS education specialist and three Ph.D. students, all included as the authors of this paper. Two of these experts have conducted extensive studies on Snap*!* and two others are experts in Snap*!* programming.

We primarily used Gorson and O'Rourke's research to design detectors because we intended to create detectors that identify when students negatively self-assess, thus catching students at these moments to provide aid and reduce negative self-assessments [4]. We additionally considered research on tinkering proposed by Dong et al. because they proposed identifiable coding actions that corresponded with students' tinkering moments during block-based programming. There are many definitions for tinkering, but the definition of Petre and Blackwell is concise and complete. They described tinkering as "a non-goal-oriented (or a theoretical) exploration of a problem space"[14]. Uncertainty or hesitation are common critical themes in the definition of tinkering, and these characteristics in code are useful to detect moments when students may be struggling [1].

In general, detectors read and analyze each students' coding trace to find the desired patterns. In the following paragraphs, we describe each of the seven detectors and the reason behind their designs. Table 3 also shows the names of the detectors, the negative self-assessment moments that correspond to each of them, and a brief description about them.

Sprite Deletion: This detector counts the number of moments when a student removes one or more sprites. Removing a sprite results in removing all the code inside that sprite and could possibly indicate that the student is deleting a large portion of code, realizing that a significant portion does not perform as intended or is not salvageable and cannot be fixed. We hypothesized that this action could be indicative of a student starting over or erased a significant portion of the code, a negative self-assessment moment [4].

Overly Idle: This detector identifies the number of times when there is a gap of more than five minutes between stu-

Table 3: Struggle Detectors and their Corresponding Self-Assessment Moments and Description

Detector Name	Survey Corresponding Moment	Detector Description
Sprite Deletion	2. Started over or erased a significant portion of code to try again	Deleting a whole sprite
Overly Idle	4. Stopped programming to plan 7. Felt unsure where/how to start programming 8. Looked up how to do something 9. Spent time planning before starting to program 10. Spent a long time looking for a simple error or mistake 13. Felt unsure about what to work on 14. Finished a feature quicker than expected	More than 5 minutes with no programming activity
Scant Blocks	7. Felt unsure where/how to start programming 9. Spent time planning before starting to program 13. Felt unsure about what to work on	5 minutes passing at the beginning with all scripts having fewer than 3 blocks
Minor Change	1. Got a simple error, like something not working as desired 3. The code was not working as expected 5. Got help from others (e.g. partner or TA) 8. Looked up how to do something 10. Spent a long time looking for a simple error or mistake 11. Struggled to fix errors and get the program to work	Tweaking parameters of blocks or reordering blocks, adding or removing 1-2 blocks between code runs
Last Rows Count	2. Started over or erased a significant portion of code to try again 6. Spent a long time working on a feature 7. Felt unsure where/how to start programming 10. Spent a long time looking for a simple error or mistake 12. Took longer than expected to finish a feature 13. Felt unsure about what to work on 15. Fixed an error faster than expected	Counting the number of rows within the last 10 minutes of programming
Excessive Runs	3. The code was not working as expected 11. Struggled to fix errors and get the program to work	Running the code several times without making changes
Blocks Per Minute	All moments can be relevant	The average number of blocks within one minute of programming

dents' actions in iSnap. Being overly idle could conceivably mean that a student is planning or thinking about what to do next, is off-task, or is not sure how to continue. While planning code and being off-task are not necessarily indications that a student is struggling, we aimed to capture the moments when a student is not sure how to progress. We also thought idleness could detect negative self-assessment moments such as feeling unsure how to start programming, looking up how to do something, planning, or finishing a feature quicker than expected [4].

Scant Blocks: This detector identifies if, after five minutes of beginning a project, all coding scripts have less than three blocks. Having a few blocks at the beginning of a coding session can possibly suggest that a student does not know how to start or has trouble starting, but may also indicate that the student was planning before programming [4].

Minor Change: This detector recognizes tweaking parameters of blocks, reordering blocks, or adding or removing 1-2 blocks between code runs and counts the number of times they occurred. This action is a form of test-based tinkering identified by Dong et al. and signifies debugging behavior [1]. Therefore, making small changes between runs can conceivably indicate the following negative self-assessment moments proposed by Gorson and O'Rourke: the student had a simple error; the code was not working as expected; the student got help from others; the student looked up how to do something; the student spent a long time looking for a simple error, or struggled to fix errors [4].

Last Rows Count: This detector counts how many coding actions were logged for a student in the last ten minutes of their programming session. We predicted the detector, which essentially quantifies how active a student is behaving at the end of their programming session, could indicate how a student feels about their coding session or if they had significant struggles earlier in their coding session such as starting over on a coding section, spending longer than expected on a feature, or being unsure how to initially start programming [4].

Excessive Runs: This detector recognizes and counts when the code is run more than two times without making any changes. This action is a form of test-based tinkering identified by Dong et al. and could signify either frustration or that a student is trying to understand the behavior of their code output [1]. Therefore, we hypothesized that running the code several times without making changes could indicate negative self-assessment moments such as the code was not running as expected, or the student was struggling to fix errors. [4].

Blocks Per Minute: We included a general detector that calculated the average rate of how many blocks a student placed per minute. The goal of this detector was not to identify any specific struggle moment, but to serve as a general indicator of how the student was progressing. Based on the work of Dong et al. [2], we believe that the amount of code added per minute, compared to the values across an entire class, may be used as an indicator of progress or struggle.

4. RESULTS AND DISCUSSION

4.1 Analysis Protocol

To determine how accurate the detectors are at identifying when a student may be struggling, we ran a post hoc analysis to find out if the detectors correlated with students' survey responses and project grades. We also looked through code traces to determine when and why detectors had a high or low correlation with certain survey responses.

The survey responses were converted to numerical values before analysis. The six-point Likert question that asked students "How did you feel about your programming experience/progress today?" was converted into a numerical response that ranged from '1' to '6', corresponding with "Very dissatisfied" to "Very satisfied," respectively. As mentioned earlier and shown in Table 1, for each of the 15 scenarios, students indicated if they did not encounter this scenario or how they felt about it if they did. We used different methods for handling "Not Applicable" or "Didn't Happen" responses when analyzing Likert scale surveys. We separately analyzed the occurrence (if the scenario occurred) and the valence (how the student felt given the scenario occurred). The idea of separating responses into two metrics was inspired by the zero-inflated model [7] which was mainly proposed for handling a high number of zero observations in a count data.

For each scenario on the survey, if the scenario occurred for the student, their response was given a value of '1'. If the scenario did not occur, the response was given a value of '0'. On a separate metric, for each scenario, if the student said that the scenario did not occur, the response was dropped during the analysis. If the student selected "I felt bad/frustrated", they were given a value of '1'; for "It was normal and I felt OK", they were given a value of '2'; and for "It was good/useful or helped me learn", they were given a value of '3'.

We additionally imported the grade each student received for their project related to the corresponding code trace. We then found how closely the frequency of each detector was activated or the respective metric of the detector correlated with the survey responses and project grades by finding the Spearman's rank correlation coefficient and the p-value. We used Spearman's correlation since it covers the ordinal data type of the survey responses and other data types that we had in our analysis, such as ratios. We consider a p-value of below 0.05 to be significant.

Finally, two experts stepped through and tagged students' code traces to determine if the detectors correctly identified struggle moments and which struggle moments each detector missed. The first expert focused on a single trace, examined all actions the student took during class time, and compared that to the student's final submission for the project. The first expert tagged actions based on the definition of detectors in two general categories: 1) actions and patterns that could trigger each of the detectors, and 2) actions/patterns that seemed to reflect struggling but were missed by all detectors. The second expert reconstructed the coding period of five students with Snap Playback to check how and why some detectors did not work, but did not do any action-by-action analysis. To avoid hindsight bias, the second expert did not look at students' final submissions. The goal was to see how these detectors work given the various coding styles of different students. In summary, this step gave experts insight into why each detector correctly or incorrectly identified or failed to identify a moment as a struggling moment.

4.2 Results

Here, we will go through each detector, determine if it was a good indicator of the scenario we hypothesized it would signify, and discuss why the detector did or did not achieve its goal. We will also discuss the relationship between the detectors and two more general variables, i.e., project total grade and general feeling about programming. Table 4 shows the correlation coefficients between the valence and occurrence of the output of each detector and their corresponding moments. The p-value of each correlation coefficient is also stated. For project grade and feelings about programming that do not have valence and occurrence, we wrote the correlation coefficient and p-value in the occurrence column and N/A (not applicable) in their valence-related cell.

4.2.1 Sprite Deletion

We predicted that Sprite Deletion could be an indicator that a student started over or erased a significant portion of code to try again, as students who deleted a sprite would delete all the code within the script for that spite, which may be a significant portion of their code. However, the correlation coefficient and the p-value for this pairing are not significant.

After looking through code traces and going through portions of students' coding snapshots, we found that most times when students deleted a sprite, it was a sprite with little or no code. A student might delete a sprite they added accidentally which will trigger the detector but by no means be a sign of struggling. While students sometimes did delete a sprite with significant code and try to recreate it in a different way, most times when students deleted a sprite, it resembled tinkering or playing with the interface. Through adding and deleting sprites, students may have learned more about the concept of sprites, but made no significant change in the overall code script.

4.2.2 Overly Idle

We found significant p-values for the correlation between the Overly Idle detector and the occurrence of planning before starting to program for students. Furthermore, there was a significant correlation between students who were idler and reporting feeling negative about looking up how to do something, spending a long time looking for an error, and not knowing what to work on.

This suggests that students who are more overly idle, i.e., performing no coding actions within five minutes or more, felt worse about having to look up how to do something, spending a long time trying to find an error, and not knowing where to start. Students may be negatively self-assessing while idle, thus feeling worse when they have looked up how to do something (a normal programming action), or not knowing what to work on. Additionally, this result could imply that students with less developed debugging skills who do not perform coding actions while debugging and trying to find an error, feel worse about taking a long time to find an error.

Table 4: Correlations between Struggle Detectors and their Corresponding Self-Assessment Moments and Project Grade

Detector Name	Survey Corresponding Moment	Valence	Occurrence
Sprite Deletion	2. Started over or erased a significant portion of code to try again	0.3165, p=0.3162	0.1349, p=0.5818
Overly Idle	Project Total Grade	N/A	0.5066, **p=0.0269**
	4. Stopped programming to plan	-0.4287, p=0.2163	0.1944, p=0.4251
	7. Felt unsure where/how to start programming	-0.5094, p=0.1095	-0.0103, p=0.9665
	8. Looked up how to do something	-0.5595, **p=0.0468**	-0.0879, p=0.7204
	9. Spent time planning before starting to program	-0.1975, p=0.5177	0.5166, **p=0.0235**
	10. Spent a long time looking for a simple error or mistake	-0.6299, **p=0.0282**	-0.0424, p=0.8633
	13. Felt unsure about what to work on	-0.6753, **p=0.0226**	0.2277, p=0.3485
	14. Finished a feature quicker than expected	-0.0782, p=0.7904	0.4293, p=0.0666
Scant Blocks	7. Felt unsure where/how to start programming	0.2472, p=0.4636	-0.0795, p=0.7462
	9. Spent time planning before starting to program	0.4192, p=0.154	0.338, p=0.157
	13. Felt unsure about what to work on	0, p=1	0.1364, p=0.5778
Minor Change	1. Got a simple error, like something not working as desired	0.1377, p=0.6246	0.3448, p=0.1482
	3. The code was not working as expected	-0.1896, p=0.5349	-0.0864, p=0.725
	5. Got help from others, (e.g. partner or TA)	0.4703, p=0.1229	-0.2706, p=0.2625
	8. Looked up how to do something	-0.144, p=0.6387	0, p=1
	10. Spent a long time looking for a simple error or mistake	0.2184, p=0.4952	0.0312, p=0.899
	11. Struggled to fix errors and get the program to work	0.0389, p=0.9044	0.0312, p=0.899
Last Rows Count	2. Started over or erased a significant portion of code to try again	0.6464, **p=0.0231**	0.3988, p=0.0908
	6. Spent a long time working on a feature	0.5455, p=0.0666	-0.0199, p=0.9354
	7. Felt unsure where/how to start programming	0.6648, **p=0.0256**	0.1169 , p=0.6337
	10. Spent a long time looking for a simple error or mistake	0.6504 , **p=0.022**	-0.0199, p=0.9354
	12. Took longer than expected to finish a feature	0.2854, p=0.3227	0.131, p=0.5928
	13. Felt unsure about what to work on	0.6257, **p=0.0395**	-0.0974, p=0.6916
	15. Fixed an error faster than expected	0.0098, p=0.9771	-0.3701, p=0.1188
Excessive Runs	3. The code was not working as expected	0.0238, p=0.9386	0.0753, p=0.7594
	11. Struggled to fix errors and get the program to work	0.2866, p=0.3665	0.1451, p=0.5535
Blocks Per Minute	** For this detector only significant correlations are mentioned.*		
	Feeling about Programming	N/A	-0.5212, **p=0.0221**
	14. Finished a feature quicker than expected	-0.3052, p=0.2887	-0.5684, **p=0.0111**
	15. Fixed an error faster than expected	-0.4454, p=0.1698	-0.4874 ,**p=0.0343**

Aligning with our hypothesis, the Overly Idle detector has a strong positive correlation with students who report planning before programming, suggesting that students who were overly idle were more likely to spend time planning before starting to program. Although at first glance, we may expect less idleness from students who plan, our impression is that planning has increased idleness because planning itself can increase the number of times students become idle, especially if students do planning before implementing each component of their program. Additionally, planning can lead to finishing the programming tasks sooner so they might have implemented everything they had planned for and become idle after a while.

In sum, this detector divides students into two distinct groups. The first category is those who feel worse while struggling with certain moments, such as not knowing what to work on next or spending a long time looking for an error. The second category is students who plan their code at the beginning and earn higher grades. This means one portion of the population recognized by this detector may benefit from intervention, and the other portion may not need intervention. In its current shape, this detector cannot distinguish between two populations, but perhaps the combination of this detector and another detector such as Last Rows Count or a detector that would determine idleness at a specific time period of programming could result in a more useful detector for automated help.

4.2.3 Scant Blocks

We found no significant correlation between the Scant Blocks detector and the moments we hypothesized that the detector would identify. While we do not have a strong explanation for not seeing expected correlations, we think that students who were not placing blocks at the beginning of their programming session could have been setting up their project, creating variables, communicating ideas to their programming partner, and doing their activities that do not have any relationship to the moments we hypothesized.

4.2.4 Minor Change

None of the correlation values between the Minor Change detector and the moments we hypothesized that the detector would capture were significant. This could be because the detector only checks specific actions between a student's click on run button (presented as a green flag in Snap*!* environment). In the tagging process, we realized that minor changes do not necessarily start and end by clicking on run button. Therefore, an improved version of this detector may be time-based, i.e., it captures the length of time a student makes minor changes.

4.2.5 *Last Rows Count*
Out of the hypothesized scenarios we thought the detector Last Rows Count would predict, four had a significant p-value. There was a positive significant correlation between the number of rows generated in the code log in the last ten minutes of programming and students reporting that they felt positive about starting over or erasing a significant portion of code to try again, being unsure where/how to start programming or what to work on, and taking longer than expected to finish a feature. Students who had a higher last rows count, or performed a higher number of coding actions in the last ten minutes of the coding session, and indicated experiencing these moments, either experienced them before or during the last ten minutes of their coding session.

If the student experienced one of these four moments before the last ten minutes of the coding session, the student may have found a remedy to their struggle, indicated by the student moving on and making good progress in the last ten minutes of their coding session, which possibly increased their self-efficacy, and thus made them able to see the benefit of their time struggling. For example, if a student felt unsure what to work on or how to start programming, the indication of having a high number of coding actions at the end of the session suggests that the student did find something to work on and was actively programming.

If a student experienced one of these four moments during the last ten minutes of the coding session, they would be performing a high number of coding actions, supporting the idea that even though a student is working through a struggling moment, they feel better if they are not idle. For example, if a student was trying to reimplement deleted code in the last ten minutes of their coding session, the fact that they were actively coding could indicate that they are making progress and feel good about reimplementing code, and the time spent deleting and recreating code is worth it. Additionally, if a student mentioned they felt good when it took longer than expected to find a simple error and they had performed a high number of coding actions in the last ten minutes, this could indicate that the student found the error and started working on other parts of the program, seeing the positive return on investing a lot of time into correctly fixing an error. If a student were to be looking for an error in the last ten minutes and had a high Last Rows Count, it could mean the student was performing multiple coding actions while debugging, revealing active debugging which may cause the student to feel better about spending a long time looking for a simple error or mistake. We continue to find that students performing coding actions while trying to find errors often feel better about their programming than being idle.

In addition to the aforementioned points, it is worthwhile to investigate the other side of this metric, i.e., students with a lower Last Rows Count may have more negative self-assessments. Given the four survey items that were significant, it seems Last Rows Count can distinguish students who performed fewer coding actions at the end because they finished early from those who performed fewer coding actions because they were stuck. Deleting a lot of code, feeling unsure and spending a lot of time looking for errors can relate to feeling stuck, which would lead to a more negative self-assessment. As a matter fact, they are also less likely to be experienced by a high-achieving student who finishes their work early.

4.2.6 *Excessive Runs*
We hypothesized that students who excessively ran the program without making changes to the program script were trying to understand why the program performs the way it does, which may be a sign for students finding their code was not working as expected or struggling to fix errors and get the program to work [1]. However, the correlation coefficient and p-values for these pairings were not significant. After looking through code traces in the tagging process, we found that students often run the program repetitively without making changes simply because they are enjoying the results of their programming, playing with what they have programmed so far. While the Excessive Runs detector could help identify when students are trying to understand the output of their program, any possible correlation is clouded by the fact that students run their program multiple times because they find what they created fun.

4.2.7 *Blocks Per Minute*
We did not focus on any particular survey scenario for the Blocks Per Minute detector. Blocks per minute is a very general gauge of how a student may be doing in their programming, so we found correlations and p-values for each survey question. We found a significant negative correlation between blocks per minute and students responding they finished a feature or fixed an error quicker than expected.

The results indicate that a higher block per minute rate can signify that the student did not finish the feature or fix an error quicker than expected, or inversely, a lower block per minute rate can indicate that the student finished the feature or fixed an error quicker than expected. This finding denotes that students who use fewer blocks per minute are more intentional and thoughtful of their coding actions, and thus may finish a programming task or fix an error faster.

4.2.8 *Feeling about Programming*
At the beginning of the survey that students took after each programming session, they were asked ”How did you feel about your programming experience/progress today?” We tested responses to this question against all detectors to see if any had a significant correlation and may be a good indicator of how a student felt about their programming. The only significant correlation was with the detector Blocks Per Minute. The correlation reveals that students with a lower Blocks Per Minute rate felt better about their programming experience/progress in the coding session. This continues to back the idea that students who created fewer number of blocks were more purposeful, and thus they feel better about their programming progress.

4.2.9 *Project Total Grade*
We tested if any of the detectors were a good indicator of the total grade a student received for the project. We found that only the overly idle detector values had a significant positive correlation with the project's grade, which suggests that students who were more idle during class time received

higher grades. Once more, this reinforces the relationship between using fewer blocks and being more thoughtful during programming. Additionally, the students' pace of doing the project may be another factor that led to this result. As found in [13], among two groups of students who spend the same amount of time on assignments or projects, those who distribute their workload in the time frame considered for their project show better performance than those with longer sessions of work.

In general, it was difficult for detectors to give an accurate indication of the grade that a student would receive because only data from two coding sessions that contributed to the project, about one hour each, was collected. However, students were allowed to complete their projects when they were not in the classroom. In other words, if data was collected for the entirety of the coding project, the detectors might be able to better correlate with student grades for the project.

4.2.10 Summary

From our results, we find two main themes. The first is that students who were more active (i.e., performing more coding actions) often felt better when they experienced certain moments in which students negatively self-assessed. In other words, students who were more idle felt negative about a couple of struggle moments that students who were less idle felt positive about. In addition, we noticed that students who performed more coding actions at the end of their session, or were less idle, felt more positively about having experienced restarting code or being unsure how or what to code compared to students who did fewer coding actions at the end of their coding session.

Second, we observe that students who used fewer blocks when programming felt better about their programming, since they may plan more and do programming more intentionally and be more thoughtful in their coding process. Also, students who reported planning before programming were identified as more overly idle compared to students who did not plan beforehand. Additionally, students with a lower blocks per minute rate more often identified finishing a feature or fixing an error faster than expected. Lastly, there was a significant correlation between students who were overly idle and students who received a better grade for the project. This signifies that students who are idler or program slower planned more and were more intentional in building their project and thus gained a higher project grade. In a nutshell, students who were more active felt better when encountering specific struggling moments than their more idle counterparts; however, being less active can suggest having plans, performing better (as measured by higher grades), and an overall better general feeling towards programming.

5. LIMITATIONS AND FUTURE WORK

This study has some limitations. First, our work is limited by a low participation rate and its generalizability needs to be verified. However, it showed great potential for exploring the connection between detectable programming patterns and moments of negative self-assessment. Second, we found a significant number of p-values to evaluate our correlation coefficients, but this could lead to the multiple comparison problem. We consider addressing this issue in our future work by using methods such as Bonferroni or Benjamini-Hochberg correction.

Future work may consist of improving our current detectors and creating new ones, for which a larger population would be highly beneficial. Additionally, a think-aloud study can provide further insight into the detectable patterns that students produce in their code traces when encountering a struggling moment, as well as the reasons behind these patterns. Moreover, implementing these detectors in a Snap*!* programming environment to provide real-time intervention is another plan of ours which will be investigated in the future.

6. CONCLUSION

This work presented an analysis of students' struggle moments when working on an open-ended project in a block-based language. We investigated whether detectors could be used to identify these struggle moments and how a student might be feeling. We considered code traces and post-surveys from two coding sessions with students working on an open-ended midterm project for an introductory programming course. Post-surveys asked students how they felt if/when they experienced 15 specific moments in their programming. These moments were based on what Gorson and O'Rourke identified as moments when novice programmers negatively self-assess [4]. Based on two prior studies [4, 1] and experts' suggestions, we created seven detectors to identify struggle moments in the collected coding traces of students and matched each detector to survey responses we hypothesized they would detect. We found correlation values for how well each detector reflected specific responses from students' surveys. While we created some detectors such as Overly Idle and Last Rows Count that significantly correlated with some of their corresponding struggle moments, some detectors did not have any strong correlations. We determined and discussed why certain detectors were successful or unsuccessful indicators of certain scenarios. Our analysis shows that first, there are detectable patterns in students' coding traces that signify when students are struggling regardless of the programming assignment, and second, these patterns are associated with students' self-reported challenges, which paves the way for realizing why they are struggling. We thus addressed our two research questions.

7. REFERENCES

[1] Y. Dong, S. Marwan, V. Catete, T. Price, and T. Barnes. Defining Tinkering Behavior in Open-ended Block-based Programming Assignments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 1204–1210, New York, NY, USA, Feb. 2019. Association for Computing Machinery.

[2] Y. Dong, S. Marwan, P. Shabrina, T. Price, and T. Barnes. Using Student Trace Logs to Determine Meaningful Progress and Struggle during Programming Problem Solving. *International Educational Data Mining Society*, 2021.

[3] G. Gao, S. Marwan, and T. W. Price. Early Performance Prediction using Interpretable Patterns in Programming Process Data. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 342–348, Virtual Event USA, Mar. 2021. ACM.

[4] J. Gorson and E. O'Rourke. Why do CS1 Students Think They're Bad at Programming?: Investigating Self-efficacy and Self-assessments at Three Universities. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, pages 170–181, Virtual Event New Zealand, Aug. 2020. ACM.

[5] M. Kong and L. Pollock. Semi-Automatically Mining Students' Common Scratch Programming Behaviors. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, pages 1–7, Koli Finland, Nov. 2020. ACM.

[6] D. Krpan, S. Mladenović, and M. Rosić. Undergraduate Programming Courses, Students' Perception and Success. *Procedia - Social and Behavioral Sciences*, 174:3868–3872, Feb. 2015.

[7] D. Lambert. Zero-Inflated Poisson Regression, with an Application to Defects in Manufacturing. *Technometrics*, 34(1):1–14, 1992. Publisher: [Taylor & Francis, Ltd., American Statistical Association, American Society for Quality].

[8] K. J. Lehman, J. R. Karpicz, V. Rozhenkova, J. Harris, and T. M. Nakajima. Growing Enrollments Require Us to Do More: Perspectives on Broadening Participation During an Undergraduate Computing Enrollment Boom. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, pages 809–815, New York, NY, USA, Mar. 2021. Association for Computing Machinery.

[9] S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter. A Robust Machine Learning Technique to Predict Low-performing Students. *ACM Transactions on Computing Education*, 19(3):1–19, Sept. 2019.

[10] S. Marwan, A. Dombe, and T. W. Price. Unproductive Help-seeking in Programming: What it is and How to Address it. page 7, 2020.

[11] S. Marwan, G. Gao, S. Fisk, T. W. Price, and T. Barnes. Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, pages 194–203, Virtual Event New Zealand, Aug. 2020. ACM.

[12] S. Marwan, J. Jay Williams, and T. Price. An Evaluation of the Impact of Automated Programming Hints on Performance and Learning. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 61–70, Toronto ON Canada, July 2019. ACM.

[13] J. P. Munson and J. P. Zitovsky. Models for Early Identification of Struggling Novice Programmers. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 699–704, New York, NY, USA, Feb. 2018. Association for Computing Machinery.

[14] M. Petre and A. F. Blackwell. Children as Unwitting End-User Programmers. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, VLHCC '07, pages 239–242, USA, Sept. 2007. IEEE Computer Society.

[15] T. W. Price, Y. Dong, and T. Barnes. Generating Data-driven Hints for Open-ended Programming. page 8.

[16] T. W. Price, Y. Dong, and D. Lipovac. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 483–488, New York, NY, USA, Mar. 2017. Association for Computing Machinery.

[17] T. W. Price, Z. Liu, V. Cateté, and T. Barnes. Factors Influencing Students' Help-Seeking Behavior while Programming with Human and Computer Tutors. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 127–135, Tacoma Washington USA, Aug. 2017. ACM.

[18] R. Zhi, S. Marwan, Y. Dong, N. Lytle, T. W. Price, and T. Barnes. *Toward Data-Driven Example Feedback for Novice Programming.* International Educational Data Mining Society, July 2019. Publication Title: International Educational Data Mining Society.