ABSTRACT

Block-based programming environments have been widely used to introduce K-12 students to coding. To guide students effectively, instructors and platform owners often need to understand behaviors like how students solve certain questions or where they get stuck and why. However, it is challenging for them to effectively analyze students’ coding data. To this end, we propose BlockLens, a novel visual analytics system to assist instructors and platform owners in analyzing students’ block-based coding behaviors, mistakes, and problem-solving patterns. BlockLens enables the grouping of students by question progress and performance, identification of common problem-solving strategies and pitfalls, and presentation of insights at multiple granularity levels, from a high-level overview of all students to a detailed analysis of one student’s behavior and performance. A usage scenario using real-world data demonstrates the usefulness of BlockLens in facilitating the analysis of K-12 students’ programming behaviors.

CCS CONCEPTS

• Human-centered computing → Visual analytics; • Social and professional topics → K-12 education; Computational thinking; Software engineering education.

KEYWORDS

Visual analytics, block-based programming, learning analytics

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:


1 INTRODUCTION

Block-based programming (programming by dragging and dropping blocks) has been widely used to ignite K–12 students’ interest in coding and foster their computational thinking skills [21]. Compared to text-based programming, block-based programming is more suitable for K-12 students since it reduces the cognitive workload associated with the programming syntax. To effectively guide students, instructors and platform owners often need to explore and understand student coding behaviors. For example, how do most students solve certain questions? Where do they get stuck and why? To answer these questions, the detailed steps of how students are dragging and dropping the blocks need to be analyzed instead of just the final blocks submitted. However, it is challenging for instructors and platform owners to analyze and extract meaningful insights from the raw data, since raw coding log data (e.g., drags and drops) are often large in scale and difficult to interpret.

Many prior studies about programming behavior analysis [4, 15] focus on solution analysis instead of procedure analysis. For example, OverCode [4] uses the variable renaming the abstract syntax tree methods to cluster different correct solutions in text-based programming, which is not sufficient for understanding how students make mistakes.

Some studies have analyzed the programming procedures, but they focus on the problem-solving strategy instead of the fine-grained steps. In terms of text-based programming, Piech et al. [17] took several screenshots during students’ programming process and clustered the images to analyze the different strategies students use. PathViewer [24] analyzes how students pass predefined test cases. In terms of block-based programming, many studies [6, 8, 9, 11, 23] propose computational ways to measure and cluster different events...
that happen during the programming process. For example, Kesselbacher and Bollin [8] calculated the relationship between different event types (e.g., using a loop or not) in block-based programming and student success. However, none of these studies can provide a fine-grained step-wise level analysis, which is essential for instructors or teachers to analyze common mistakes or critical steps and give accurate and personalized support. BlockLens enables this by proposing a method of identifying important outcome-associated code snapshots within a sequence of step-wise snapshots and classifying them as being associated with either success (such snapshots are called checkpoints in this paper) or failure (called warning signs).

Information visualization has proved to be an effective way to analyze large-scale student learning data. Some of these visualizations focus on individual learning path analysis. For example, iSnap [18] and Dr. Scratch [14] propose visual systems aimed at individual students, rather than providing educators with analytical results from the perspective of common problem-solving strategies among multiple students. Others analyze the behaviors of a group of students. They have demonstrated their advantages for teachers’ observation and analysis of students’ behavior in MOOC platforms [1, 13, 20], interactive online question pools [25, 26] and online exams [12]. However, they cannot be directly applied to block-based programming questions due to the questions’ different characteristics and interactions. For example, as opposed to math questions [25], block-based programming questions can have multiple solutions. Inspired by these studies, we proposed a modified Sankey diagram with contextual information (e.g., programming blocks, time) to demonstrate the important outcome-associated code snapshots and step-wise transitions for instructors to analyze the fine-grained programming behaviors of students.

We propose BlockLens, a novel visual analytics system that allows educators and platform owners to view the block-based coding behaviors of K–12 students through an interactive Lens. We first formulate the process of solving a block-based programming question as a sequence of code modifications (i.e., moving, creating, deleting, or modifying blocks). Then we generate the code snapshot after each code modification. Inspired by the methods for analyzing event sequences outlined by Gotz et al. [5], we identify and classify key snapshots as checkpoints and warning signs. Such checkpoints and warning signs enable a meaningful characterization of students’ programming progress and subsequently facilitate the visual analysis of problem-solving strategies. The interface of BlockLens consists of four views. Question Selection View allows instructors to see key statistics for each question, and select a question they would like to focus on. Path Summary View provides an overview of all students’ progress on the selected question, as well as the checkpoints, warning signs, and frequent snapshots which appear in the students’ answers. Student View groups students by their progress on questions according to the number of checkpoints and warning signs that appear in their code snapshots. Finally, Sequence View shows the detailed snapshot sequences. The main contributions of this work are as follows:

- We propose BlockLens, a novel visual analytics system for interatively monitoring and analyzing student programming behaviors at multiple levels of details.

2 DATASET

Anonymized event sequence data was collected from a bilingual e-learning platform where K–12 students complete basic coding exercises using Scratch 3.0, a block-based programming language [19]. Our dataset consists of data from robot programming questions. In these programming questions, students are asked to instruct the robots to finish some tasks such as collecting items. After submitting the code, a simulator runs, showing an animation of the robot’s response to the student’s program. The score is then computed based on whether the code passes the test case.

3 REQUIREMENT ANALYSIS

To gain an in-depth understanding of the requirements and challenges of understanding student coding behaviors, we worked closely with four educational experts (E1–E4) who have 5–20 years’ experience in designing and teaching K–12 computational thinking curricula. We had a series of online meetings with them and recorded their opinions towards developing a visual analytics system. Then all authors discussed and summarized five major requirements for analyzing students’ coding behaviors (R1–R5).

R1. Monitor students’ overall performance on a certain question. An overview of students’ performance is desired by the experts for easy selection of detailed questions. The overview should present important performance metrics to demonstrate whether students have been struggling with a question.

R2. Identify the problem-solving behavior and strategies of groups. The experts stated that it is not easy for teachers to observe each student’s behavior and learn common reasons for success or failure. As a result, they suggested that our system should allow educators to differentiate between the coding behavior of different groups of students, such as two groups who reached a correct answer using different strategies, and show how the groups are distributed based on multiple performance metrics.

R3. Identify checkpoints and warning signs. E1–E3 believe that identifying positive or negative key points within students’ problem-solving paths is helpful for educators to provide assistance to students. Thus, our system should be able to identify such key points and present them to teachers intuitively.

R4. Display a summary of problem-solving paths. All the experts believed a visual summary of the problem-solving paths taken by multiple students was needed. This would allow educators to monitor and compare the coding behavior of multiple students.

R5. Show an individual student’s detailed problem-solving process. E4 pointed out an educator’s need to view the detailed information (e.g., a students’ actual code) and further understand how they are doing on a question or support them. Thus, the detailed sequence of coding steps should be available in our system.

4 IDENTIFYING KEY SNAPSHOTS

Since the block-based programming questions allow students to use and arrange blocks freely, the quantity of possible code snapshots...
In practice, the identified checkpoints are significant common steps that students may take in a correct solution, while the warning signs are a strong indicator that does not lead to a correct solution. In addition, since checkpoints are markers of success and warning signs are markers of failure, plotting warning sign count vs. checkpoint count separates the students who got the question correct from those who did not, and can further group them into smaller subdivisions based on how many checkpoints/warning signs their snapshot sequences contain.

5 VISUAL DESIGN

BlockLens consists of four views: Question Selection View, Path Summary View, Student View, and Sequence View.

Question Selection View. Question Selection View (Fig. 1A) allows users to select a question to inspect within the other views (R1). To facilitate question selection, simple and intuitive visualizations of three key statistics are provided for each question. The proportion of students who attempted a given question who got the question correct, i.e., the question’s success rate, is shown using a clock-inspired visualization [16], wherein each rectangle represents a step. Finally, the average time taken is represented using a clock-inspired [2, 3, 10] unit visualization.

Student View. Student View (Fig. 1B) shows how students are distributed based on performance metrics (R2, R3). It contains a student checkpoint–warning plot and two distribution plots.

The student checkpoint–warning plot is a unit visualization showing groups of students with different numbers of key snapshots that are identified using the algorithm introduced in Section 4 (R2, R3). The motivation of applying unit visualizations is that they enable the effective identification of outlying students within a group [16]. The plot has warning count on the y-axis and checkpoint count on the x-axis. Within the plot, green points represent students who got the question correct; orange points represent students who did not. The size (area) of each point encodes the student’s step count. The interactive plot operates in two modes. To facilitate easy selection of both single student and multiple students (R2, R5), we allow the teachers to switch between “select individual” and “select group” modes. When the teacher selects a student in this view, either the student or the entire group that student belongs to will be shown in the Sequence View according to the mode. Additionally, in “select group” mode, the group’s transition behaviors are also highlighted in Path Summary View.

We also include two distribution plots below the student checkpoint–warning plot, one for time taken from start to submission and one for step count. For both, we overlay the distribution curves (generated using Kernel Density Estimation 1) of students who got the question correct versus those who did not (R2).

Path Summary View. Path Summary View (Fig. 1C) consists of a Sankey-based visualization where each node represents a code snapshot and the links represent one-step transitions between two snapshots (R2, R4). The width of each link represents the number of transition instances. The nodes (snapshots) are arranged in columns from left to right according to their length in the number of blocks. Snapshots that appear frequently and/or are identified as checkpoints or warning signs (R3) (see Section 4), which we call keyframes, are shown in their entirety (all code blocks displayed). Other snapshots are shown using a compact representation which only shows the number of blocks within the snapshot. We use Sankey diagrams with circular links 2 to account for links between nodes in the same column and links going “backwards” from right to left across columns. A backwards (right-to-left) link occurs when a student deletes blocks, resulting in a snapshot that has fewer blocks than the previous snapshot (and is thus in a column further to the left). To avoid visual clutter, we only highlight the starts and ends of circular links between nodes in the same column by setting the opacity of all other parts of the links very low, and we

1https://en.wikipedia.org/wiki/Kernel_density_estimation
2https://github.com/tomshanley/d3-sankey-circular/
label the sources and destinations of the nodes. To represent code snapshots, we show code blocks as color-coded rectangles placed within a larger rectangle representing the coding area. Unconnected groups of blocks are separated with a gap. A short description is displayed within each code block, as are shadow blocks (i.e., “block-within-a-block”) and their values. The type of a given snapshot is encoded using the fill and stroke of the rectangle representing the coding area. Green snapshots are checkpoints; orange snapshots are warning signs. Other snapshots are grey, and any snapshots with a border are the ones that occur frequently in snapshot sequences.

Finally, a step count distribution bar is included on every snapshot visualization (except for those marked “other”). The step count distribution bar is a horizontal stacked bar chart showing the proportion of instances in which the snapshot in question appeared in each quartile of steps within a student’s snapshot sequence.

**Sequence View.** Sequence View (Fig. 1D) displays the complete snapshot sequences, as well as the events linking each snapshot, of any students selected in Student View or Path Summary View (RS). The snapshots are similar to those in Path Summary View, but have two key differences: the step count distribution bar is not included, and “other” snapshots are shown in full, unlike in Path Summary View. These differences align well with Sequence View’s purpose, i.e., showing the complete problem-solving paths of individual students, instead of only a summary of multiple students.

### 6 USAGE SCENARIO: IDENTIFYING A COMMON PROBLEM-SOLVING STRATEGY

In this usage scenario, the user uses BlockLens to explore a question and identifies a common strategy for solving it.

The user first browses all questions using Question Selection View (Fig. 1A). Then he identifies Question 1_6 as an interesting question since the correct rate of the question is moderate. The correct rate indicates that some students may meet difficulties while others can finish it without problems. As the result, the user decides to conduct an in-depth analysis of the problem-solving behaviors.

The user then looksting at Student View. Looking at the step count distribution plot, the user sees a narrow peak in the green distribution curve at close to 5 steps (many students in a narrow range), but a much shorter peak further to the right shows a few students took >10 steps. On the other hand, the step counts of students who get the question wrong fall in a wider range, and many students submitted their answers very early, perhaps indicating that the students meet difficulties in the early stage. The user then carefully observes the student checkpoint–warning plot. There are many more green points (correct students) to the right of the plot, which suggests the presence of more checkpoint instances is strongly correlated with success. Furthermore, the presence of a warning sign is a strong indicator of failure. Looking at Path Summary View, the user observes the warning sign usually appeared during the first half of the problem-solving process, such that it could act as an early warning sign (see the circled step count distribution bar in the right part of Fig. 2), which matches his findings from the step distribution plot. He also observes that most correct students appeared to have gone from Frequent (1) to Checkpoint (2) to Checkpoint (5) to Solution (8). Back in Student View, the user selects the group of students with three checkpoint instances using the Checkpoint–Warning plot. All the points in the group are green (all correct). Upon doing so, the user notices that the path from (1) to (2) to (5) to (8) is highlighted, and that most of the people in this group have followed this path, which can be viewed as a common strategy. On the other hand, the presence of the warning sign indicates that a student has deviated from the recommended steps, and almost all students with warning sign instances got the question wrong. Finally, in Sequence View, the user sees all paths in detail and confirms the common strategy.

This usage scenario confirms the usefulness of BlockLens. First, our key snapshot detection and classification algorithm can successfully extract the snapshots that are closely associated with the final result. This can help the instructors to notice common errors and also provide early intervention. Second, the visual analytics dashboard provides a convenient and efficient approach for instructors to select questions and then conduct an in-depth analysis. They can identify common problem-solving strategies easily and offer targeted help to students.

### 7 DISCUSSION

Our usage scenario demonstrates the effectiveness of our system. However, we also identify some areas worth discussing. In this section, we discuss potential limitations and points to consider including the generalizability and scalability of the system.

**Generalizability.** Although our prototype system focuses on questions using Scratch, we expect our method to be generalizable to other block-based programming environments, e.g., those based on Snap! 3 or Blockly 4. The checkpoint and warning sign identification methods used are even more widely generalizable. They can in principle be applied to any event sequence with a binary outcome (e.g., success vs. failure). BlockLens is not designed for completely open-ended project-based coding without any right or wrong answers, such as projects where block-based programming is used to make animations or games. Additionally, the current prototype supports a binary correct/incorrect scoring scheme. In cases where there are multiple possible scores but there is a clear divide between low and high scores, scores could be thresholded and categorized as high or low. Fully generalizing to multiclass scoring systems would require modifying the algorithm and redefining checkpoints and warning signs, but would not necessitate major design changes.

---

3https://snap.berkeley.edu/about
4https://developers.google.com/blockly
Scalability. The scalability of the unit visualizations used within our system is currently limited to a user’s ability to discern individual visual elements. There are also usability concerns when the number of units represented is large, especially in regard to features like selecting a single unit, e.g., selecting a student in the student checkpoint–warning Plot. However, these concerns can be mitigated by adding a zoom feature or through aggregation methods, e.g., replacing multiple units with a glyph. Another scalability issue is representing long and complex code block sequences. We have been able to control the amount of horizontal space occupied by Path Summary View and the maximum number of blocks drawn within a snapshot (these are linked; see Section 5), but complicated questions necessitating a large quantity of code blocks would require further measures, such as a short, ‘compact’ version of long snapshots with some code details hidden.

8 CONCLUSION AND FUTURE WORK

We propose a novel visual analytics system for analyzing student behaviors in block-based programming environments. BlockLens allows the interactive exploration of student coding behaviors at multiple levels of detail. It allows educators to monitor students’ overall performance on questions, identify groups of students which differ in their coding progress, performance, and behavior within a question, and identify both common problem-solving strategies and common mistakes or pitfalls. We describe a usage scenario to show the usefulness of our system.

In the future, we would like to further test our approach with more programming questions which require more steps and more types of blocks, e.g., conditionals and loops. In addition, it would be interesting to explore how the key snapshot identification algorithm can be generalized to text-based programming.

ACKNOWLEDGMENTS

We would like to thank TrumpTech for the dataset and valuable feedback, and all the anonymous reviewers for their constructive comments.

REFERENCES