Understanding Learning Curves and Trajectories in CSS Layout

Meen Chul Kim
Drexel University
Philadelphia, PA, USA
meenchul.kim@drexel.edu

Thomas H. Park
Codepip
Philadelphia, PA, USA
thomas@codepip.com

Ruixue Liu
Worcester Polytechnic Institute
Worcester, MA, USA
rliu2@wpi.edu

Andrea Forte
Drexel University
Philadelphia, PA, USA
aforte@drexel.edu

ABSTRACT

Web development is a learning context with the potential to support rich computational thinking. Large-scale analysis of compilation and runtime errors have been used in introductory programming courses and similar approaches can be used to understand learning in web development environments. We investigated activity logs of a novel web coding game to uncover learning trajectories and what people struggle with when learning flexible box (flexbox), a collection of new CSS layout features. We designed a game called Flexbox Froggy, in which learners solve challenges by writing a few lines of CSS code, moving from simple levels that require knowledge of one flexbox property, to complex levels combining multiple properties. We investigate learning curves based on the changes in syntactic and semantic errors learners make as they complete the game. Our findings show that people performed better encountering a single new property than combined with properties they had already practiced. Clusters of learners at different levels did not demonstrate expected error rates based on learning curve theory. Also unexpectedly, advanced groups that mastered syntax had higher semantic error rates than the beginner group, especially when attempting new properties or complex use cases. We conclude with implications for designing and developing introductory web programming games and other instructional materials.

KEYWORDS

Clustering, CSS layout, Educational game, Learning analytics, Learning curve analysis, Web development

ACM Reference Format:
Meen Chul Kim, Thomas H. Park, Ruixue Liu, and Andrea Forte. 2019. Understanding Learning Curves and Trajectories in CSS Layout. In *SIGCSE '19: 50th ACM Technical Symposium on Computer Science Education, February 27–March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287387

SIGCSE'19, February 27–March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/2…$15.00
https://doi.org/10.1145/3287324.3287387

1 INTRODUCTION

As learning analytics and educational data mining techniques improve and expand, analyzing code-writing data at scale has become an increasingly productive way to investigate how people learn programming and what they struggle with. Studies in this space have used quantitative methods to explore mastery in introductory computer programming [4, 29, 33], common errors and misconceptions such as compiler errors [2, 13], and learning behavior in virtual learning environments such as MOOCs [11, 21], online communities [1, 7, 10, 34], and games [14, 16].

Web development is one context in which people practice rich and multi-layered computation. For many, web development is a first experience in creative computation [8, 30]. Because of its potential bridging role between early programming exposure and later mastery [27], web development is a fruitful area of inquiry for computing education research, but it's difficult to measure learning in the often informal, idiosyncratic contexts where people attempt to solve web development problems. Learning analytics, though not without limitations, is an interesting approach because we can develop creative proxies for measuring learning in contexts where collecting data on learning outcomes is challenging or not possible.

This work presents our preliminary investigation of web programming data at scale. We designed an online game to create playful, motivating experiences with concrete learning goals [20, 22]. Flexbox Froggy (https://flexboxfroggy.com) supports people in learning Flexible Box (flexbox), a collection of CSS features that makes it easier to design web pages with responsive layouts. In this game, people write a few lines of CSS code to guide frogs to lily pads of matching colors, while gradually exposed to core properties of flexbox such as aligning content and items. We explored the changes in learner errors through completion of the game. Learning curve analysis was employed to model the trajectories of such learning where the likelihood of error-making is expected to decrease over the course of engagement. Clustering was also applied to understand aggregate patterns of learning. Our investigation was guided by the following research questions:

(1) How can we measure learning based on users' successes and failures at different game levels?
(2) How can knowledge components and learning curves be modeled in this context?
(3) What patterns of behavior emerge as people progress to increasingly difficult levels?

2 RELATED WORK

This paper builds on three areas of computing education: 1) web development, 2) educational games, and 3) learning analytics. Literature on web development as a context for computational learning examines formative experiences and misconceptions. Early work examined the established practices and conceptions of web developers [8, 30] and uncovered computational features that novice and non-programmers frequently struggle with such as hyperlinks [9, 28] and nesting [23, 24]. Visual layout was found to be difficult for both beginning and experienced programmers [23, 27, 30]. WYSIWYG interfaces are reported to construct unproductive mental models which could block a later sense of mastery [26].

Educational games research explores how to provide authentic, playful experiences for learning computation. Some research stresses creating gaming environments with explicit instructional goals [16, 18]. Others focus on providing game-like environments where people learn through exploration, tinkering, and creation of personally meaningful projects [6, 19, 32]. These platforms are often designed for informal learning opportunities. Environments that provide open-ended, unstructured environments require learners to sustain high levels of motivation while they master the basic but often challenging coding skills necessary to create interesting projects. In our game design efforts, we aim to provide an enjoyable gaming experience with clear goals [15, 19] to foster motivation and understanding and, in turn, elicit prolonged engagement in learning computational features.

The third literature we leverage is the growing body of learning analytics and educational data mining research focused on understanding computing education through large scale analysis of trace data. Such studies have investigated compilation and runtime errors [2, 13], coding patterns [12, 29, 35] of novice programmers [3, 4, 33] and structured computational learning in various environments [11, 17, 21]. Others have explored learning computation in online communities given manipulative toolkits [1, 7, 10, 34]. In order to uncover significant trajectories of learning, these studies employed a variety of quantitative techniques such as learning curve analysis, error quotient, and machine learning. In this work, we aim to identify quantitative analytics that can be applied to large-scale code-writing data generated by learners.

3 METHOD

3.1 Design of Flexbox Froggy

Flexbox Froggy is designed to challenge learners to solve simple puzzles by writing CSS code. We were inspired by computer literacy games such as Gidget, CSS Diner, and Erase All Kittens where people learn about various computational features while solving problems. We chose flexbox because it is a new module and therefore relevant for both beginners and experienced web developers, additionally, both novice and expert programmers struggle with web layouts [23, 27, 30].

In Flexbox Froggy, users learn flexbox's eight core properties, namely, **justify-content**, **align-times**, **flex-direction**, **order**, **align-self**, **flex-wrap**, **flex-flow**, and **align-content** through 24 levels. Most levels have one or two solutions. The game marks solutions correct based on the position of the frogs. We designed levels to provide a smooth learning curve, flat in the beginning so as not to intimidate learners, and then gradually ramp up the challenge. The last level is treated as the final "boss level" and combines nearly all the CSS properties learners have encountered, providing a tougher challenge and hopefully a sense of accomplishment when solved. The first level introduces justify-content, one of most basic properties and its values. Next, learners are exposed to the same property again in another straightforward use case, which leads to a more complex use case. Learners then explore the next property and its use cases, with subsequent levels eventually combining multiple properties.

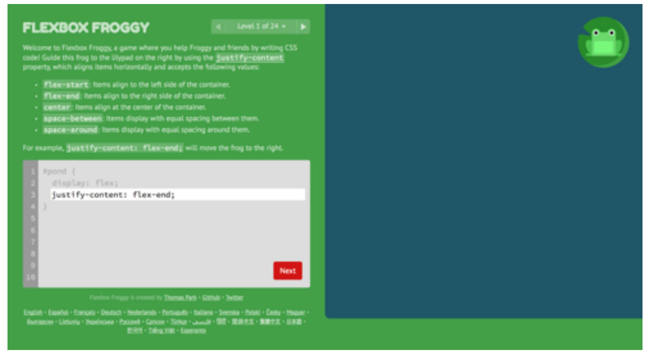

Figure 1: Game Interface with Code Submission

Figure 1 shows the game's interface, which mainly consists of two panes: 1) left pane for in-game reference and code completion and 2) right pane for instant assessment of code-writing. The left pane features a reference guide, instructions and explanations of each stage, property, and value. Reference material is also available in context as tooltips. The code editor provides a partial structure to be completed by users who submit a CSS code snippet by clicking "next." Learners can proceed to next levels only if the current level is solved. Completed levels can be revisited using navigation at the upper-right. The right pane updates as the users edit code to provide within-tool scaffolding in the form of immediate feedback to aid with debugging [18, 25]. If a submitted answer is correct, the frog appears on the lily-pad. If not, the entire screen trembles.

3.2 Data Collection and Processing

Flexbox Froggy is playable in a browser and requires no registration. We did not directly recruit players. Upon release of the game, it was registered to Hour of Code on Code.org. Hacker News and some subreddits featured it, and it is still shared on a regularly on Twitter. We collected anonymous data related to completion of game tasks, input of correct and incorrect answers, and completed levels. To distinguish unique learners, we assigned a random alphanumeric token to each learner based on browser information and epoch time of first access. We collected 4,865,200 submitted answers by 9,980 unique learners. Then, subpopulation who completed all levels was selected, which reduced the submissions to 2,293,681. We considered the level complete once a user submitted a correct answer and discarded subsequent submissions for the same level. Thus, the data was reduced to 2,152,670 answers submitted by 5,520 unique users. We evaluated keystroke-level feedback about

Table 1: Game Levels and Knowledge Components

Level	Properties	Knowledge Components
1	justify-content	horizontal alignment
2	justify-content	horizontal alignment
3	justify-content	horizontal alignment
4	justify-content	horizontal alignment
5	align-items	vertical alignment
6	justify-content	horizontal alignment
	align-items	vertical alignment
7	justify-content	horizontal alignment
	align-items	vertical alignment
8	flex-direction	horizontal direction
9	flex-direction	vertical direction
10	justify-content	horizontal alignment
	flex-direction	horizontal direction
11	justify-content	horizontal alignment
	flex-direction	vertical direction
12	justify-content	horizontal alignment
	flex-direction	vertical direction
13	justify-content	horizontal alignment
	align-items	vertical alignment
	flex-direction	horizontal direction
14	order	horizontal order
15	order	horizontal order
16	align-self	vertical order
17	order	horizontal order
	align-self	vertical order
18	flex-wrap	horizontal wrapping
19	flex-direction	vertical direction
	flex-wrap	horizontal wrapping
20	flex-flow	vertical direction
		horizontal wrapping
21	align-content	horizontal alignment
22	align-content	horizontal alignment
23	flex-direction	vertical direction
	align-content	horizontal alignment
24	justify-content	horizontal alignment
	flex-direction	vertical direction
	flex-wrap	horizontal wrapping
	align-content	horizontal alignment

syntactic and semantic errors through the W3C CSS Validation Service (http://www.css-validator.org); an answer is tagged as having a syntactic error if its result is incorrect and validity returns false. If it is incorrect while validity is true, the answer was regarded as having a semantic error. Based on this validation, the data set ended up having 48 feature dimensions, according to the number of syntactic and semantic errors per level. To build better fitting learning curve and clustering models, we used the interquartile range to remove outlying learners based on the total number of code submissions, which yielded a final data set of 1,775,039 answers submitted by 5,282 unique users.

3.3 Learning Curve Analysis and Clustering

Learning curve analysis is an approach that models learners' performance over time. Based on the power law of practice underlying this approach, the probability that a learner will make an error is expected to decrease as they repeatedly practice a target knowledge component (KC) [5]. In intelligent tutoring where learning curve analysis is often used to estimate student performance, problems are easily broken down into multiple smallest units of action [31]. However, it is less clear what is a meaningful unit of action in our data. Before applying learning curve analysis to our data, therefore, we need to define knowledge components. Table 1 describes the properties expected to be learned at each game level. As described, we designed the game to have learners practice different properties in multiple levels. Failing a given level does not necessarily mean that every knowledge component represented in the code is incorrect, only a subset may be incorrect. Therefore, each level was tagged with one or more knowledge components intended to practice. Just as submissions can be measured for correctness, each of the knowledge components can also be analyzed for changes in correctness over time.

Clustering is an unsupervised machine learning technique that uncovers latent structure in data without ground truth or a priori idea of what should be found. In learning analytics, clustering is used to identify groups of learners that share characteristics. As discussed in Related Work, a range of clustering algorithms such as hierarchical clustering, *k*-means clustering, and EM algorithm have been applied to code-writing data. To investigate collective trajectories of learning, we employ the *k*-means++ algorithm implemented in scikit-learn, a widely used Python machine learning library. We decided to use *k*-means++ over regular *k*-means as the latter is more sensitive to initial cluster centroid seeds. An individual learner's syntactic and semantic errors at each level were considered as feature dimensions. Then, we investigated the existence of different learning trajectories and characteristics per group.

4 RESULTS

4.1 Patterns between Syntax and Semantics

Figure 2 illustrates patterns of syntactic and semantic errors per level. The upper figure shows a stacked mean error rate chart, the lower one depicts a ratio column chart between two types of errors. Syntactic errors take up the largest fraction of incorrect submissions. As expected, learners struggled most with the final "boss" level. Unexpectedly, mean error rates in total do not decrease as the users move on to more advanced levels. Moreover, the ratios of semantic errors tend to increase and jag even though learners have had more opportunities to practice the properties of the flexbox module. These patterns violate our expectations of learning curve analysis described in the methodology section. Instead of a smooth curve, we found that learners struggled more than expected as they encountered familiar properties in new configurations.

4.2 Finding Homogeneous Trajectories

Any variant of *k*-means algorithms requires a manual selection of the number of clusters. To find an optimal *k*, we examined the changes in within-cluster sum of squared error by increasing *k*

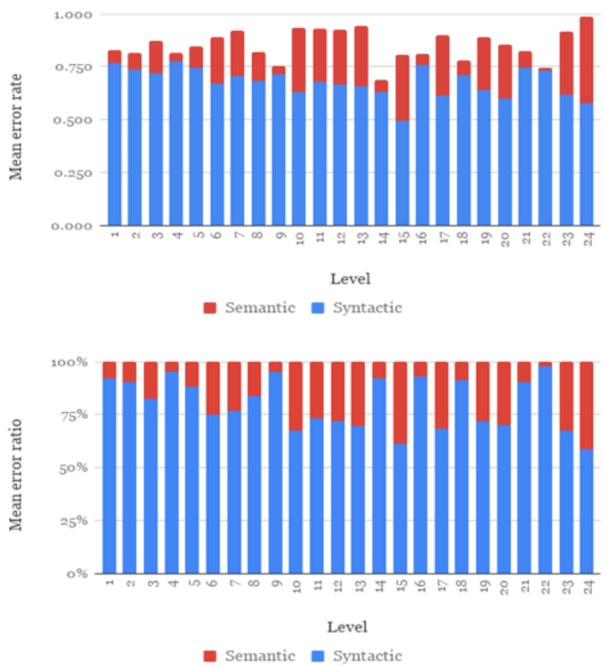

Figure 2: Syntactic and Semantic Errors. Stacked Rate Chart (Upper) and 100% Stacked Ratio Chart (Lower)

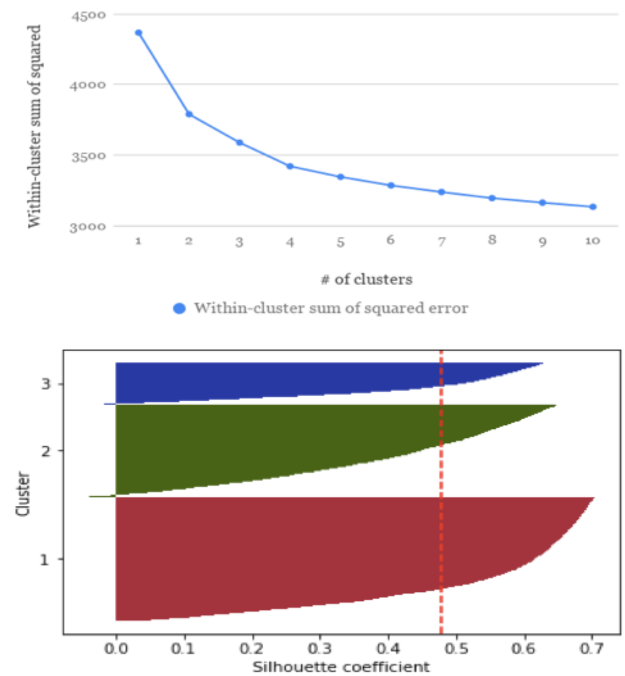

Figure 3: Changes in Within-cluster Sum of Squared Error (Upper) and Silhouette Coefficient at 3 as *k* (Lower)

ranging from 1. As shown in Figure 3 upper, increasing the number of clusters beyond 3 minimally affects the error values. To validate the consistency within clusters of data given 3 and 4 as candidate *k*s, we investigated the silhouette coefficient, using the Euclidean distance (See Figure 3 lower). Given the silhouette ranges between -1 and 1, the learner groups showed a clearer homogeneity when 3 is chosen as the number of clusters. Based on these observations, 3 was chosen as the number of clusters. In order to build a final model, *k*-means++ was run 500 times on the 48 feature dimensions of 5,282 users and the model with the least within-cluster sum of squared error was used.

First, we wanted to see learning trajectories and curves for the whole data set. As described, trajectories were expected to start with a high error intercept, curve downwards, and then plateau near a zero-error rate as users acquire the mastery. Figure 4 top displays the trajectories of learners considering the average rate of all types of errors per level. Based on error trends, we labeled learners in three groups: Gr1 (n=1,892), Gr2 (n=1,546), and Gr3 (n=1,844). Learning curves corresponding to each cluster are also rendered. As illustrated in the figure, three groups of learners show similar patterns of rising-falling-rising learning curves which do not match the general expectations above. They start with an increasing curve over the first seven levels. Even after repeated opportunities to practice the same KC groups, the number of errors tends to increase. One interpretation is that flexbox is a new concept to both the novice programmer and the experienced developer; however, it is surprising that better performing learner groups, *i.e.* Gr2 and Gr3, would not exhibit transfer from experience with other, similar CSS properties. After declining for levels 8 and 9, the errors spike between the 10th and 13th levels. This indicates the learners were most challenged when multiple KCs were combined despite prior exposure to the properties and values. Afterwards, the errors decline and increase again. While Gr1's low error rate persists when practicing a new property (order) at the 14th level, Gr2 and Gr3 error rates spike. Overall, all the groups share indistinct patterns with little arithmetic difference and almost no persistent learning curves.

The rest of the subfigures depict learning trajectories and curves by error types. Syntactic errors show good learning curves that decrease and plateau over time (See Figure 4 middle). In these learning curves, the error rates start between 70% and 80%, high enough to indicate that around three out of four learners struggle with the KC at the first attempt. Although Gr2 and Gr3 have spikes in error rates when moving from level 14 to 15 (See Figure 4 top), their syntactic error rates decreased (See Figure 4 middle). Instead, semantic error rates spike (See Figure 4 bottom). While trajectories with syntactic errors show good learning curves, the learning trajectories and curves derived from semantic errors add richer interpretations to the findings: even though learners get familiar with syntactic operations such as declaring relevant properties and assigning syntactically acceptable values, they still struggle with assigning proper values.

4.3 Learning Curves and Clusters by KCs

Although our findings do not match what we expected based on learning curve theory, this does not necessarily invalidate these

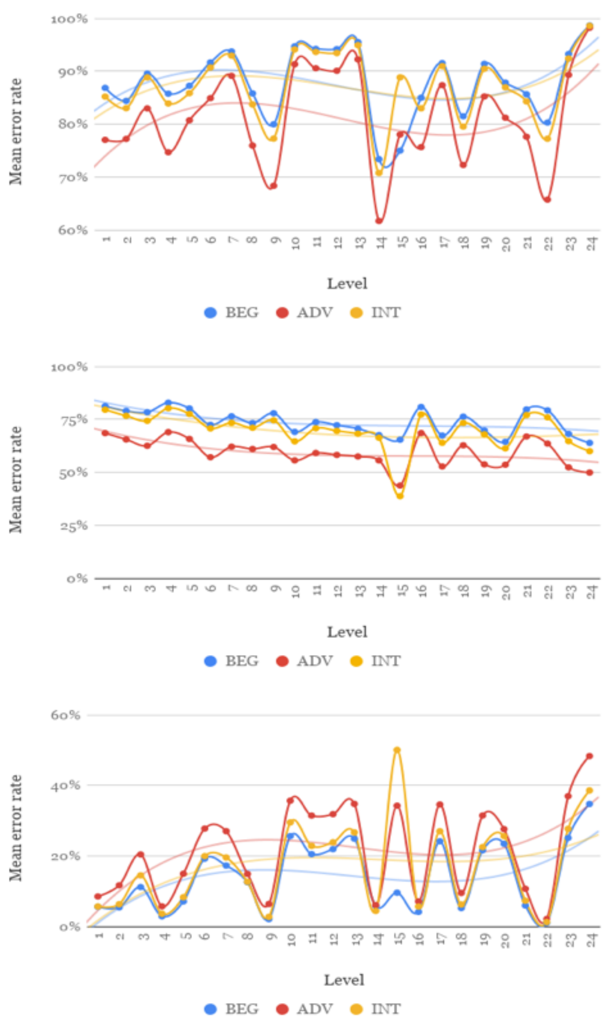

Figure 4: Learning Curves for All Levels: All (Top), Syntactic (Middle), and Semantic (Bottom) Errors

models. Instead, we suggest that investigating learning patterns and trajectories in a game that relies on accumulative knowledge components requires interpreting the game as a more complex learning experience with additional dimensions. Toward that end, we categorized levels in two KC groups based on similarity of visual operations, namely 15 levels that involve **alignment** and 10 that involve **direction**.

Figure 5 shows learning curves for the 15 levels in the alignment KC group, sequentially ordered. The uppermost figure shows rising-falling learning curves including all types of errors for each learner group; notably there are almost no clear learning trajectories that suggest mastery. All learner groups show similar patterns rising at the end. Learning curves by error type are depicted in the middle (syntactic errors) and bottom (semantic errors) for the alignment KC group. In these cases, all of the groups show good learning curves for syntax that slope downwards over time (See Figure 5 middle). It is notable that although syntactic errors decrease, semantic error rates increase as the learners encounter more complex use cases; additionally, Gr1 struggle more with syntax while Gr2 and Gr3 make more semantic errors (See Figure 5 bottom).

Figure 6 illustrates learning curves for the 10 levels in the direction KC group. Again, combining error types shows jagged learning curves for all types of learners, indicating no clear learning trajectory (See Figure 6 top). The middle and bottom figures depict learning curves by error types, demonstrating again that all the groups show good learning curves in mastering syntax (See Figure 6 middle). Semantic error rates keep jaggedly increasing despite opportunities to practice, with Gr3 exhibiting the most errors (See Figure 6 bottom).

5 SUMMARY AND DISCUSSION

In this paper we explored the quantitative generation of KC models, learning curves, and learner clusters with the code-writing data submitted to a web development game. We investigated the efficacy of models generated using all errors versus two discrete error types (syntax and semantic) and explored knowledge component groups as an additional strategy for understanding learning curves. We identified three distinct user groups that we labeled Gr1, Gr2 and Gr3. Models inclusive of both types of errors showed learning curves that tended to rise-fall-rise, indicating no clear learning trajectory. While the different learner groups had similar learning patterns, the learner groups who performed better over the entire course of levels, namely Gr2 and Gr3, struggled more with the semantic operations, especially in levels with new properties or more complex use cases. Overall, all groups of users were generally found to have good learning curves in syntactic manipulation as they were exposed to more opportunities to practice the properties. Contrary to learning curve theory, however, the general learning curves did not show the expected reduction in error rate. This led us to examine patterns by the KC groups, *i.e.* alignment and direction. The results showed Gr1 performed better when challenged with complex combinations of KCs and all learner groups showed clear learning trajectories in practicing syntax.

Our efforts to employ learning curve analysis and clustering in analyzing web programming data can be used to consider the design of instructional materials and games that introduce learners to syntactic and semantic knowledge components. Although learners demonstrated increased proficiency with syntax while learning new knowledge components, our findings suggest that when introducing combinations of components, learning trajectories become less easily identified.

Based on these experiences, we identified several opportunities to refine our instruments as well as data collection. The fact that the learners semantically struggled with complex use cases in spite of multiple exposure to the same KC groups might be a sign that the skill required to use a single layout operation is different from the skill of combining multiple components, and that the two should be separated into different KCs (and taught as separate concepts as well). In addition, to stabilize learning curves, a prep session or lengthier level design as well as greater quantity and variety in the use cases may be required. Finally, we could experiment with features like code mirror and syntax highlighting.

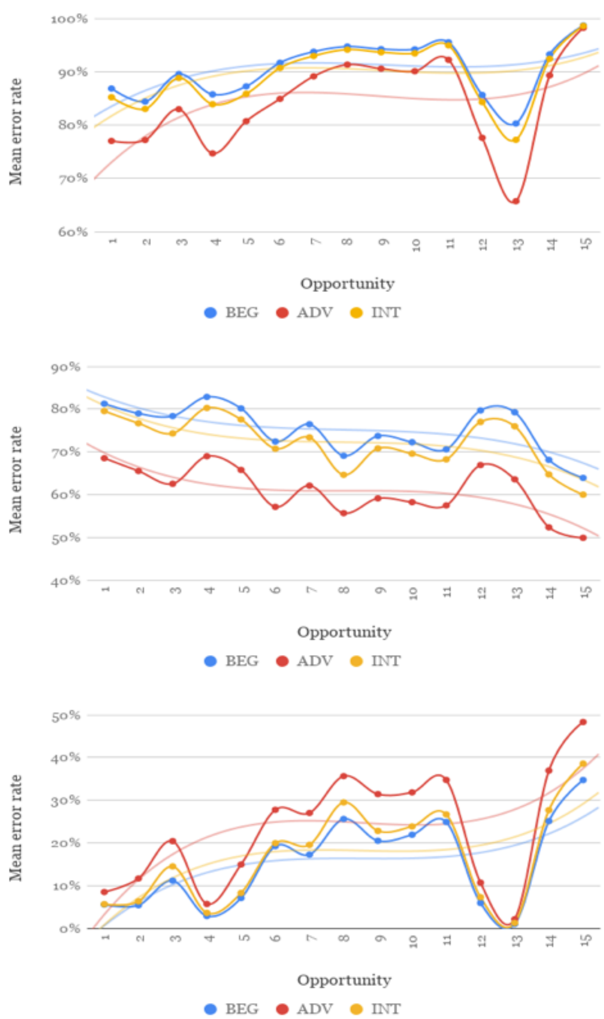

Figure 5: Learning Curves for Alignment: All (Top), Syntactic (Middle), and Semantic (Bottom) Errors

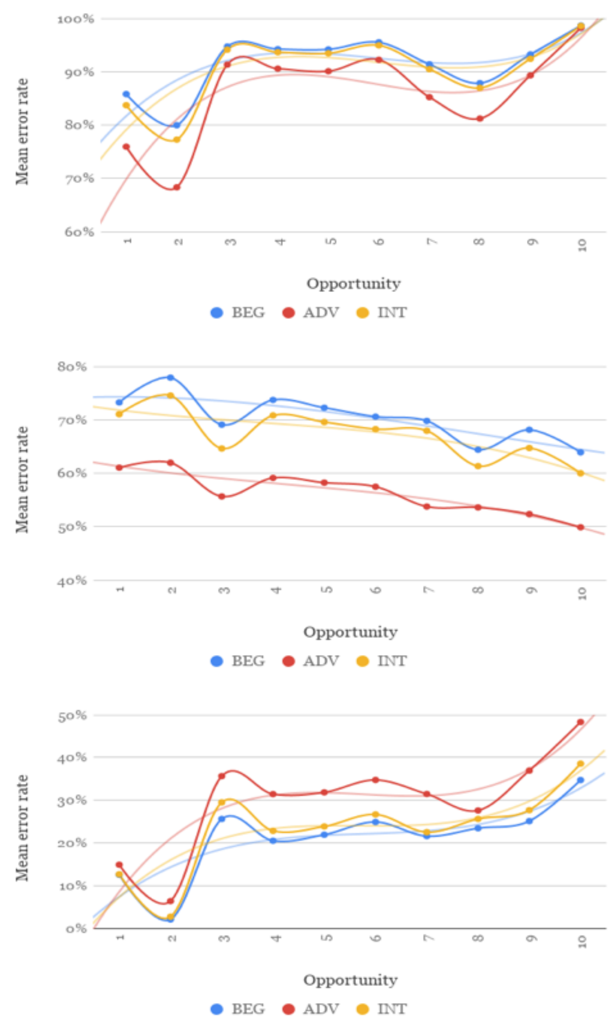

Figure 6: Learning Curves for Direction: All (Top), Syntactic (Middle), and Semantic (Bottom) Errors

Limitations of this work include the absence of observational or other data to aid in our interpretations and the restricted context of Flexbox Froggy as a learning environment. In future work, we plan to expand our work in the context of another introductory web programming game. We also seek a clear explanation for extreme outlier data. In continuation of this work, we aim to compare complete and incomplete sequences of learner actions with keystroke-level learning behavior modeling to further understand learner-generated web coding data as a resource for learning analytics.

ACKNOWLEDGMENTS

This work is supported by the NSF under Grant No.: 1339344.

REFERENCES

[1] Efthimia Aivaloglou and Felienne Hermans. 2016. How Kids Code and How We Know: An Exploratory Study on the Scratch Repository. ACM Press, 53–61. https://doi.org/10.1145/2960310.2960325

[2] Brett A. Becker. 2016. A New Metric to Quantify Repeated Compiler Errors for Novice Programmers. ACM Press, 296–301. https://doi.org/10.1145/2899415.2899463

[3] Matthew Berland, Taylor Martin, Tom Benton, Carmen Petrick Smith, and Don Davis. 2013. Using Learning Analytics to Understand the Learning Pathways of Novice Programmers. 22, 4 (2013), 564–599. https://doi.org/10.1080/10508406.2013.836655

[4] Adam S. Carter, Christopher D. Hundhausen, and Olusola Adesope. 2015. The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior. ACM Press, 141–150. https://doi.org/10.1145/2787622.2787710

[5] Hao Cen, Kenneth R. Koedinger, and Brian Junker. 2006. Learning factors analysis – a general method for cognitive model evaluation and improvement. (2006), 164–175.

[6] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. 15, 5 (2000), 107–116.

[7] Sayamindu Dasgupta, William Hale, Andrés Monroy-Hernández, and Benjamin Mako Hill. 2016. Remixing as a pathway to computational thinking. (2016), 1438–1449.

[8] Brian Dorn and Mark Guzdial. 2010. Discovering computing: perspectives of web designers. ACM Press, 23. https://doi.org/10.1145/1839594.1839600

[9] Alain Désilets, Sébastien Paquet, and Norman G. Vinson. 2005. Are wikis usable? ACM Press, 3–15. https://doi.org/10.1145/1104973.1104974

[10] Deborah A. Fields, Yasmin B. Kafai, and Michael T. Giang. 2017. Youth Computational Participation in the Wild: Understanding Experience and Equity in Participating and Programming in the Online Scratch Community. 17, 3 (2017), 1–22. https://doi.org/10.1145/3123815

[11] Elena L. Glassman and Robert C. Miller. 2016. Leveraging learners for teaching programming and hardware design at scale. (2016), 37–40.

[12] Elena L. Glassman, Rishabh Singh, and Robert C. Miller. 2014. Feature engineering for clustering student solutions. ACM Press, 171–172. https://doi.org/10.1145/2556325.2567865

[13] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. 2017. Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis. ACM Press, 89–98. https://doi.org/10.1145/3051457.3051467

[14] William Jernigan, Amber Horvath, Michael Lee, Margaret Burnett, Taylor Cuilty, Sandeep Kuttal, Anicia Peters, Irwin Kwan, Faezeh Bahmani, Andrew Ko, Christopher J. Mendez, and Alannah Oleson. 2017. General principles for a Generalized Idea Garden Image 1. 39 (2017), 51–65. https://doi.org/10.1016/j.jvlc.2017.04.005

[15] Yasmin B. Kafai. 2006. Playing and Making Games for Learning: Instructionist and Constructionist Perspectives for Game Studies. 1, 1 (2006), 36–40. https://doi.org/10.1177/1555412005281767

[16] Michael J. Lee, Faezeh Bahmani, Irwin Kwan, Jilian LaFerte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, Sheridan Long, Margaret Burnett, and Andrew J. Ko. 2014. Principles of a debugging-first puzzle game for computing education. IEEE, 57–64. https://doi.org/10.1109/VLHCC.2014.6883023

[17] Michael J. Lee and Andrew J. Ko. 2015. Comparing the Effectiveness of Online Learning Approaches on CS1 Learning Outcomes. ACM Press, 237–246. https://doi.org/10.1145/2787622.2787709

[18] Michael J. Lee, Andrew J. Ko, and Irwin Kwan. 2013. In-game assessments increase novice programmers' engagement and level completion speed. (2013), 153–160.

[19] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. 10, 4 (2010), 1–15. https://doi.org/10.1145/1868358.1868363

[20] M. J. Mayo. 2009. Video Games: A Route to Large-Scale STEM Education? 323, 5910 (2009), 79–82. https://doi.org/10.1126/science.1166900

[21] N. Myller, J. Suhonen, and E. Sutinen. 2002. Using data mining for improving web-based course design, Vol. 1. IEEE Comput. Soc, 959–963. https://doi.org/10.1109/CIE.2002.1186125

[22] Harold F. O'Neil, Richard Wainess, and Eva L. Baker. 2005. Classification of learning outcomes: evidence from the computer games literature. 16, 4 (2005), 455–474. https://doi.org/10.1080/09585170500384529

[23] Thomas H. Park, Brian Dorn, and Andrea Forte. 2015. An analysis of HTML and CSS syntax errors in a web development course. 15, 1 (2015), 4:1–4:21.

[24] Thomas H. Park, Meen Chul Kim, Sukrit Chhabra, Brian Lee, and Andrea Forte. 2016. Reading Hierarchies in Code: Assessment of a Basic Computational Skill. ACM Press, 302–307. https://doi.org/10.1145/2899415.2899435

[25] Thomas H. Park, Ankur Saxena, Swathi Jagannath, Susan Wiedenbeck, and Andrea Forte. 2013. openHTML: Designing a transitional web editor for novices. (2013), 1863–1868.

[26] Thomas H. Park and Susan Wiedenbeck. 2010. First Steps in Coding by Informal Web Developers. IEEE, 79–82. https://doi.org/10.1109/VLHCC.2010.20

[27] Thomas H. Park and Susan Wiedenbeck. 2011. Learning web development: Challenges at an earlier stage of computing education. (2011), 125–132.

[28] Ljubomir Perković, Amber Settle, Sungsoon Hwang, and Joshua Jones. 2010. A framework for computational thinking across the curriculum. ACM Press, 123. https://doi.org/10.1145/1822090.1822126

[29] Kelly Rivers, Erik Harpstead, and Kenneth R. Koedinger. 2016. Learning curve analysis for programming: Which concepts do students struggle with? (2016), 143–151.

[30] Mary B. Rosson, Julie Ballin, and Heather Nash. 2004. Everyday programming: Challenges and opportunities for informal web development. (2004), 123–130.

[31] Kurt VanLehn, Kenneth R. Koedinger, Alida Skogsholm, Adaeze Nwaigwe, Robert G. M. Hausmann, Anders Weinstein, and Benjamin Billings. 2007. What's in a step? Toward general, abstract representations of tutoring system log data. (2007), 455–459.

[32] Karen Villaverde and Daniel Jaramillo. 2010. Game design and development course taught with Alice. 26, 2 (2010), 22–29.

[33] Christopher Watson, Frederick W.B. Li, and Jamie L. Godwin. 2013. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. IEEE, 319–323. https://doi.org/10.1109/ICALT.2013.99

[34] Seungwon Yang, Carlotta Domeniconi, Matt Revelle, Mack Sweeney, Ben U. Gelman, Chris Beckley, and Aditya Johri. 2015. Uncovering trajectories of informal learning in large online communities of creators. (2015), 131–140.

[35] Hezheng Yin, Joseph Moghadam, and Armando Fox. 2015. Clustering Student Programming Assignments to Multiply Instructor Leverage. ACM Press, 367–372. https://doi.org/10.1145/2724660.2728695