Auto-Generated Game Levels Increase Novice Programmers' Engagement*

Michael J. Lee
Department of Informatics
New Jersey Institute of Technology
Newark, NJ 07102
mjlee@njit.edu

Abstract

A significant number of novices are learning programming using various online resources. Unfortunately, it is highly likely that these first-time learners will encounter obstacles that are too difficult to overcome on their own, especially as they reach more complicated concepts. Keeping these online learners engaged with the content is essential for them to learning programming, as their experience may have long-term effects on the way they view computing. One possible way to address this issue is to detect when a learner is having difficulties with a concept, provide them with automated help and encouragement, and present them with opportunities for more practice. For struggling learners, more practice will help them better understand the concept(s), and prepare them for later topics. We tested this by modifying an existing programming game, building on its existing frustration detector to provide learners with extra levels for more practice when necessary. We ran a controlled experiment with 400 participants over the course of 1.5 months. We found that the users who received extra levels when frustrated completed more core levels than their counterparts who did not receive these additional opportunities for more practice. Based on these findings, we believe that adaptive opportunities for more practice is essential in keeping educational game learners engaged, and propose future work for researchers and designers of online educational games to better support their users.

1 Introduction and Related Work

Over the last decade, there has been a significant increase in the number of novices learning programming on their own using various online resources such as Massive Open Online Courses (MOOCs) and educational games. However, critics have long claimed that online learning is not as effective as traditional classroom learning because of the absence of face-to-face interactions [5]. Attrition rates in introductory programming (CS1) courses may support these criticisms, with a worldwide survey showing that in-person CS1 classes and MOOCs have a 33% [3] and 95%+ [6] dropout rate, respectively.

The relatively higher retention rate for in-person courses may be partially explained by the personalized feedback students receive from their teachers. In a classroom, teachers can use various cues to determine the best way to help their students, such as giving them additional questions or practice. Unfortunately, many of these chances to help learners go unfulfilled in online contexts, which can negatively affect learning outcomes [7] and engagement [13]. In fact, studies have identified that a lack of motivation is a root cause for high dropout rates in online courses [13]. Without the help of more experienced individuals such as teachers, keeping students motivated to learn can be challenging, especially if they encounter obstacles that they cannot overcome on their own.

We may be better able to provide online learners with additional help and practice by adapting to their needs. For example, a system could detect when learners are having difficulties with a problem (i.e., frustrated), give them encouraging and helpful feedback hints, and provide them with an opportunity for more practice on the same type of problem(s). The idea of adaptive technology in educational systems is not new. Standardized testing such as the Graduate Record Examination (GRE) and the Graduate Management Admissions Test (GMAT) use Computerized Adaptive Testing (CAT) that adapts to test-takers' ability level by determining which question to ask next based on the correctness of the previous question [15, 22]. However, this type of on-the-fly adaptation might work well for individual users, but may be difficult to implement in classes with multiple students learning the same content at the same time. Course curricula are usually preplanned and difficult to deviate from once a class has started. Although teachers may have some flexibility to modify in-person course content in an ad-hoc manner, they have to consider that changing something will affect all of the students in the class. On the other hand, while online learning resources such as MOOCs, tutorials, and games also typically have fixed content/curriculum, many also have options for learners to access content at their own leisure and pace, without affecting other learners' experience in the same course.

We set out to create and evaluate a system that overcomes some of these limitations in current learning resources, providing an online curriculum that

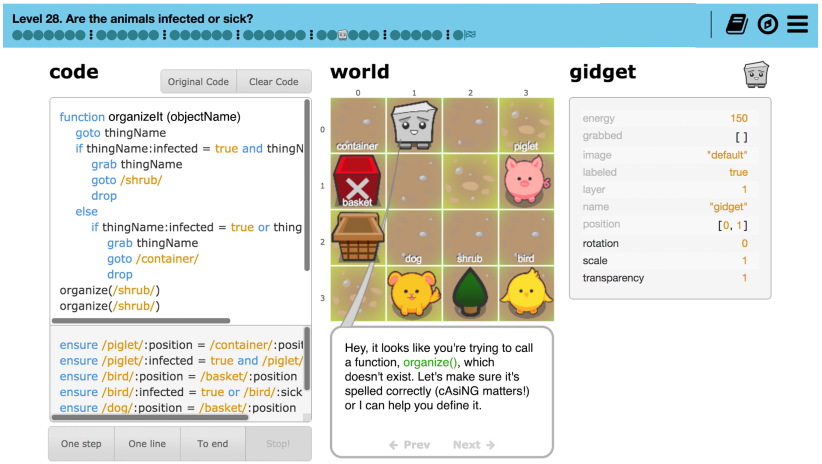

Figure 1: *A screenshot of the Gidget introductory programming game.*

adapts to the needs/skill of each individual player within an educational game. This project explores if extra automatically-generated levels, triggered by users' performance on previous levels, affects their motivation to complete more levels in an online educational programming game. We tested our game with and without this feature with 400 new users, tracking their progress through the game for one week (7 days) each, spanning a total of 1.5 months.

2 Method

2.1 The Gidget Educational Game

We modified our free, introductory coding game, Gidget *(helpgidget.org)*, for this study. The game has a total of 37 levels, where each level teaches a new programming concept (e.g., variable assignment, conditionals, loops, functions, objects) using a Python-like, imperative language [8, 11]. For each level, a player must fix existing code to help the game's protagonist. The objective of each level is to pass 1–4 test cases (i.e., statements that evaluate to `true`) after running the code. After the code runs, the game shows which test cases failed and succeeded. Each level introduces at least one new programming concept, becoming progressively harder in subsequent levels. Therefore, users are exposed to more programming concepts the farther they progress through the game. Finally, the game also includes a set of help features to help players overcome obstacles while coding on their own [8], including a coarse-grained frustration detector that provides encouraging hints/messages to those that are struggling with a level [9]. For this study, we extended the game's existing frustration detector (described in Section 2.2) to auto-generate and provide

learners with an extra level covering the same concept after they complete the current level. Details of the level generation process are detailed in Section 2.3.

2.2 Frustration Detection

We reused the frustration detector from our past work [9], which utilized coarse-grained predictors [17] to detect learners' frustration. When the system detects frustration, it provides customized feedback (and hints) to help re-engage the learner. For this study, we extended this frustration detector to additionally trigger an extra level-generator (described in 2.3). As outlined in [9], we defined frustration as, *deviations from the...*

1. ... average number of consecutive code executions with the same edit location
2. ... average time between code executions
3. ... average number of code executions
4. ... average time spent on a level
5. ... average time without any activity (idle time)

Deviation was defined as values exceeding two standard deviations from the calculated mean of any measure, as this threshold can be considered "unusual." Using a data set of 15,448 past users' game logs, we calculated all of the means and standard deviations for each of the frustration measures above. This data set was detailed, including individual players' time spent on level, idle time, all of their code edits, clicks, keystrokes, and execution button usage in the game.

2.3 Extra Level Auto-Generation Program

Once we determined that a learner was struggling with a level (reaching the threshold value of any of the measures listed in the previous section), we created a system to auto-generate an extra level for them to play *after* completing the current level. To help learners reach this newly generated level, the game includes various help features, including customized messages that are triggered by the frustration detector, which have been shown in our past works to help learners successfully overcome the issue(s) with their current level [9, 10].

The creation of our level auto-generator was guided by the work done in adaptive curricula within the intelligent tutoring and educational communities (e.g., [4, 12, 16]). For example, Baker et al.'s intelligent tutoring system gave off-task learners up to three additional exercises covering the same material, which led to learning gains that were comparable to their on-task peers [2]. To create the actual levels, we used program synthesis, which has been used by others in generating mathematical problems and proofs [1, 19], educational games [20], and introductory program auto-graders [18]. Since the programs we need to generate are mostly straightforward/uncomplicated, we create a repository of predefined programming patterns for different programming concepts,

and created a simple program synthesizer that generates levels that cover a specific concept, along with unique variable and function names, and different characters and obstacles that the player can interact with in the Gidget game.

In addition to the predefined programming patterns, we included a feature to create different world layouts for each level to differentiate and make them visually interesting for learners. The layout and puzzles of Gidget are similar to that of Sokoban puzzles—problems where a character must move items within a grid world to specific locations while avoiding environmental obstacles. We used existing procedural content generation algorithms for making Sokoban puzzles [14, 21] to generate solvable (i.e., there is at least one valid path to get an item from one position to another) Gidget levels with obstacles and objects for the character to interact with. Finally, because each generated level must contain broken code for the learner to fix (which serves as the game's instructional material), after verifying the new level is solvable using automated checks [14, 21], we used a scrambler (algorithm defined in [8]) to intentionally "break" the working code by injecting a certain number of bugs.

2.4 Pilot Study

Our goal was to provide extra practice for concepts that players struggled with, but not too many as to further disengage them. To better understand how many additional levels the system should provide a player, we ran a pilot study with 90 participants and up to 3 extra auto-generated levels for each original level (as suggested by Baker et al. [2]). We recruited these participants through the sign-up page of the game. When a new player was creating an account, they had the option to opt-in to a research study, with a high-level description of the research goals as not to prime them to the feature we were testing. Participants were randomly assigned to a condition having 1, 2, or 3 levels, where the number indicates the number of extra levels each participant would receive if they triggered the frustration detector. Each condition had 30 participants, with similar demographic characteristics.

We found that there was a statistically significant difference in levels completed among the three conditions ($\chi^2(2, N = 90) = 6.1398, p < .05$). Further post-hoc, pairwise analyses revealed that the players in the 1-level condition completed significantly more levels than both their 2-level condition ($W = 4.505, Z = -2.050, p < .05$) and 3-level condition ($W = 4.503, Z = -2.213, p < .05$) counterparts. This suggests that giving learners too many additional levels covering the same concept disengaged them from the game, leading them to quit. Based on these results, we modified our game to only include a maximum of one extra auto-generated level per original game level.

2.5 Participant Recruitment

We evaluated our system with a group of 400 new users of the game who were randomly assigned to the unmodified version of the game (the control condition) or the modified version of the game including the extra level-generator (the experimental condition). The sign-up screen asked users for their age, gender, e-mail address, a checkbox indicating whether they have prior programming experience, and a checkbox (with link to consent form) asking if they were willing to participate in a research experiment. For this study, we only selected users that indicated they were 18 years old or older, had no prior programming experience, and willing to participate in a research experiment. For selected participants, the game recorded their game condition so that they would only see the game version they were assigned to, even when coming back to play at a later time. Adapting the methodology from one of our prior studies [9], we set the observation time to 7 days (168 hours) per user to have a consistent evaluation window for all users. To promote quick account creation, we did not collect other demographic information such as ethnicity, geographical location, or education level. Participants were required to read and digitally sign an online consent form that briefly described the study. We were intentionally vague in our description of the additional levels, stating that we were "testing how extra practice can help with learning and engagement" to minimize any potential leading or biasing of participants who might be sensitive to potentially receiving additional levels (e.g., a user who perceives they are getting many levels covering the same concept might become disengaged if they feel this is a reflection of poor performance on their part). However, we e-mailed all participants a copy of the study procedures 7 days after the end of their individual observation window to debrief them, regardless of the condition they were assigned to. Prior to the debrief message (one day after their observation window ended), we sent an e-mail with a link to an optional online questionnaire. This was only sent to the experimental group participants, as only they experienced the extra auto-generated level feature. It asked them to rate their agreement to the following statements about their experience with the game on a scale from 1 ('strongly disagree') to 7 ('strongly agree'):

1. Having multiple levels covering the same concepts kept me engaged with the game.
2. I would have preferred fewer levels covering the same concepts.
3. Having multiple levels covering the same concepts helped me me better understand these concepts.

3 Results & Discussion

We report on our quantitative results comparing the outcomes from our two groups using nonparametric Wilcoxon rank sums tests, with a confidence of

$\alpha = 0.05$, as our our data were not normally distributed. The study used a between-subjects design, with 200 participants in the control condition group (aged 18-57; median 20), and 200 participants in the experimental condition group (aged 18-60; median 20). We compared demographics between the two groups, and found no significant differences between the control and experimental conditions by age or gender (99 males, 92 females, and 9 other or decline to state; and 102 males, 89 females, and 9 other or decline to state, respectively). We operationalized our key dependent variable, *engagement*, as the number of levels completed. Because those in the experimental condition saw more levels by design (i.e., a player in the experimental condition might be given extra auto-generated levels for extra practice), for all comparisons between the two conditions, we do not count the extra auto-generated levels, only comparing the core levels (i.e., the main levels that *all* players would see, regardless of condition) to measure how far they progressed in the game. We also report on the participants' responses to the optional questionnaire.

3.1 Experimental Group Complete More Core Levels

All participants in both conditions completed at least four levels. After adjusting for the auto-generated levels, the range of the core levels completed in the control and experimental conditions were 4-37 (median 11) and 4-37 (median 13), respectively. We used the game logs to verify that all experimental condition participants received at least one extra auto-generated level during their play time (with more occurring in later, more difficult stages). There was a significant difference in the number of levels participants completed between the two conditions ($W = 42401.5, Z = 1.8993, p < .05$), with the experimental group participants completing more core levels.

Since all of the participants indicated they were novice programmers, these results suggest that something about interacting with the extra levels had a significant positive effect on the experimental condition participants' engagement and ability to complete more core levels (and therefore expose themselves to more concepts) in the game compared to their control condition counterparts.

3.2 Unable to Compare Differences in Play Times

Next, we had planned to measure the differences in completion times for the core levels that participants completed. However, because everyone completed a different number of levels, we would only able to compare the levels that all 400 participants completed (i.e., Levels 1-4) to see if there were any differences in play times. Our logs indicated that only 9 of our 200 participants in the experimental condition received at least one of extra practice level during the first four levels, likely because the first few levels are very low in difficulty.

Therefore, we were unable to compare the differences between the control and experimental group play times since the majority of the experimental group (191/200, or 95.5%) did not experience anything differently from the control group for these common, completed levels.

3.3 Experimental Group Agrees Extra Levels Kept Them Engaged

For our optional questionnaire responses to the experimental group, we had a response rate of 25 out of 200 (12.5%). In our analysis, we we flipped the scale for Question 2 as it was stated negatively. Although we did not have a comparison group and response rate was low, we did find interesting trends.

For Question 1, which asked if the extra levels kept them engaged with the game, our learners gave a median score was 6 (range 4-7). This is promising, as the main goal for providing extra auto-generated levels was to give learners extra practice so they would be better prepared to complete subsequent levels. This also addressed one of our main concerns that giving learners more levels covering the topics they had just been struggling with, might be disengaging.

For Questions 2 and 3, our median scores were 4 (range 2-7) and 3 (range 1-5), respectively. Question 2 asked if learners would have preferred fewer levels covering the same concepts. Our data indicated that learners were neutral about preferring more levels, confirming our findings in our pilot study, where people did not necessarily want more levels. This makes sense, as too many levels covering the same concept can become disengaging, but too few might not help them with learning (even if they were not aware that the purpose of the next level covering the same concept was to give them more practice). This is further demonstrated in the outcome for Question 3, which asked learners if having levels covering the same concept actually helped them learn. Our participants were slightly negative from neutral, indicating that they might not have *believed* that having the extra levels were useful for learning. However, our game logs show otherwise, as the experimental group participants who experienced extra auto-generated levels completed significantly more levels than the control group participants who did not have this feature (see Section 3.1).

4 Conclusion

Our findings show that extra practice levels, triggered by a frustration detector, can significantly improve users' performance in an educational game. In our study, our experimental group participants (those who received extra practice levels for concepts that they struggling with) completed significantly more levels than their control group counterparts (who played the game without this feature). Designers for online resources teaching programming may

benefit from detecting when learners are struggling with a concept, and give them customized opportunities for extra practice on those same concepts.

We have several limitations to our study. We recruited participants who opted into a research study while signing up for an educational game. These participants may already have high motivation, and therefore may not be completely representative of the larger population. Next, we used on-the-fly auto-generated levels, where different users received completely different levels even for those covering the same concepts. Though these were generated by the same algorithm, the unique differences between these levels may have had an effect on learners, which also means it affected our results. However, we found that most learners were able to finish the extra levels the system gave them, and that they completed more subsequent levels, suggesting that the extra practice helped them. Next, we only surveyed experimental group participants and had a low overall response rate, which may limit the generalizability of the findings. In future studies, we might ask participants to complete all levels and questionnaire (adding free-response questions and/or conducting interviews to collect qualitative data). Finally, for future studies, we could use pre-post tests to measure how these new features affect players' learning outcomes.

Our study results show that extra practice levels, triggered by coarse measures to detect frustration, are sufficient in increasing online learners' level-completion performance. Our future work will examine these outcomes in more detail to isolate features that are causing these effects, and also how different types of extra practice levels might contribute to further potential differences.

5 Acknowledgements

This work was supported in part by the National Science Foundation (NSF) under grants DRL-1837489 and IIS-1657160. Any opinions, findings, conclusions or recommendations are those of the authors and do not necessarily reflect the views of the NSF or other parties.

References

[1] Chris Alvin, Sumit Gulwani, Rupak Majumdar, and Supratik Mukhopadhyay. Synthesis of geometry proof problems. In *AAAI Artificial Intelligence*, 2014.

[2] Ryan Baker, Albert T Corbett, Kenneth Koedinger, Shelley Evenson, Ido Roll, Angela Wagner, Meghan Naim, Jay Raspat, Daniel Baker, and Joseph Beck. Adapting to when students game an intelligent tutoring system. In *International Conference on Intelligent Tutoring Systems*, pages 392–401. Springer, 2006.

[3] Jens Bennedsen and Michael E Caspersen. Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2):32–36, 2007.

[4] Peter Brusilovsky. A distributed architecture for adaptive and intelligent learning management systems. In *Workshop "Towards Intelligent Learning Management Systems", Artificial Intelligence in Education*. Citeseer, 2003.

[5] Mark Bullen. Participation and critical thinking in online university distance education. *Int. Journal of E-Learning & Distance Education*, 13(2):1–32, 2007.

[6] Wenzheng Feng, Jie Tang, and Tracy Xiao Liu. Understanding dropouts in moocs. *Association for the Advancement of AI*, 2019.

[7] Starr Roxanne Hiltz. Collaborative learning in asynchronous learning networks: Building learning communities. 1998.

[8] Michael J Lee. Teaching and engaging with debugging puzzles. *University of Washington, Seattle, WA*, 2015.

[9] Michael J Lee. (re)engaging novice online learners in an educational programming game. *Journal of computing sciences in colleges*, 35(8), 2020.

[10] Michael J Lee, Faezeh Bahmani, Irwin Kwan, et al. Principles of a debugging-first puzzle game for computing education. In *IEEE VL/HCC*, 2014.

[11] Michael J Lee, Amy J Ko, and Irwin Kwan. In-game assessments increase novice programmers' engagement and level completion speed. In *ACM ICER*, 2013.

[12] Yanyan Li and Ronghuai Huang. Dynamic composition of curriculum for personalized e-learning. *FAIA*, 151:569, 2006.

[13] Lin Y Muilenburg and Zane L Berge. Student barriers to online learning: A factor analytic study. *Distance education*, 26(1):29–48, 2005.

[14] Yoshio Murase, Hitoshi Matsubara, and Yuzuru Hiraga. Automatic making of sokoban problems. In *PRICAI*, pages 592–600. Springer, 1996.

[15] Gregor M Novak, Andrew Gavrin, Christian Wolfgang, and Just-in-Time Teaching. Blending active learning with web technology, 1999.

[16] Neil Peirce, Owen Conlan, and Vincent Wade. Adaptive educational games: Providing non-invasive personalised learning experiences. In *IEEE DIGITEL*, pages 28–35, 2008.

[17] Ma MT Rodrigo and Ryan S Baker. Coarse-grained detection of student frustration in an introductory programming course. In *ACM ICER*, 2009.

[18] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. In *ACM SIGPLAN*, pages 15–26, 2013.

[19] Rohit Singh, Sumit Gulwani, and Sriram Rajamani. Automatically generating algebra problems. In *AAAI Conference on Artificial Intelligence*, 2012.

[20] Adam M Smith, Eric Butler, and Zoran Popovic. Quantifying over play: Constraining undesirable solutions in puzzle design. In *FDG*, pages 221–228, 2013.

[21] Joshua Taylor and Ian Parberry. Procedural generation of sokoban levels. In *INM Conference on Intelligent Games and Simulation*, pages 5–12, 2011.

[22] Brenda Cantwell Wilson and Sharon Shrock. Contributing to success in an introductory computer science course: a study of twelve factors. In *ACM SIGCSE Bulletin*, volume 33, pages 184–188. ACM, 2001.