

A Gamified Multi-Format Assessment for High School Programming Camps

Dr. Seth Polsley, University of Nebraska-Lincoln

Seth Polsley is an Assistant Professor of Practice in the Jeffrey S. Raikes School of Computer Science and Management and the School of Computing at the University of Nebraska-Lincoln. He teaches the core data science curriculum for the Raikes School, as well as general department- and college-level courses focused on teaching AI technologies to students across the university. Coming from Texas A&M University, where he earned a PhD in Computer Engineering, and the University of Kansas, where he earned his BS in Computer Engineering, his interests span human-computer interaction and educational technologies to brain-computer interfaces and speech systems. He is open to exploring any technologies that may be applied to assisting humans in their learning, development, as well as daily interactions.

Devadarshan Pushkaran, University of Nebraska - Lincoln

Zohaib Shaikh, University of Nebraska - Lincoln

A Gamified Multi-Format Assessment for High School Programming Camps

Summer camps offer valuable opportunities for broadening participation in computing, connecting institutions with communities while helping students explore new ideas and learn. For instance, high school computer science camps focused on skill development can better prepare students for technical degrees, yet organizers of large multi-section camps face a specific challenge: how to place students in the right section.

Our coding camp at a large midwestern university faced this issue with two sections: a Java portion for beginners and a Python portion for intermediate data science applications requiring AP CSA-level knowledge. Placement based purely on AP course completion may struggle to capture true programming ability, but the students themselves are generally unequipped to judge their own readiness. If a student is initially placed in a misaligned section for their coding competency level, that student can begin to have a very negative experience, either becoming bored or falling rapidly behind, but there is no easy remediation by offering switching sections as few students would choose to draw any attention or leave a new potential friend group.

We developed “Ricky Robot’s Journey Home,” a multi-format gamified assessment and learning tool to evaluate programming concepts through both paper and digital experiences. The activity follows Ricky the Robot’s story after his spaceship crashes. First, students complete paper-based Java tasks (e.g., demonstrating string processing through a ‘signal decoder’ task), aligned with AP CSA outcomes. Then, after a brief Python lecture covering critical concepts and syntax, they transition to an online game where they navigate Ricky through levels to locate ship components. Each component requires implementing Python functions evaluated via unit tests, with sufficient solutions unlocking the path to victory.

To our knowledge, this is the first implementation of a dual-format (paper and digital) gamified assessment for programming skill evaluation in high school summer camps that specifically uses: (1) a unified narrative across both formats, (2) cross-language transfer assessment, and (3) an auto-graded digital platform to support sustained learning. Supporting the value of the narrative and gameplay, qualitative observations indicated that students enjoyed the game so much they requested additional levels to practice advanced concepts. Quantitatively, we aggregated across the paper-based Java assessments and digital-based Python coding interactions (e.g., number of submissions) to explore trends; we found that there is a statistically significant difference in the way students across competency levels engaged with the software, with those exhibiting higher code proficiency being far more willing to try ideas and test often, informing potential pedagogical enhancements.

Introduction

The Need for a Summer Camp Assessment

College summer camps provide students of all ages with a variety of advanced computing opportunities. Ranging from hands-on activities and events, like robotics competitions, to purely code-focused boot camps, students benefit from additional skill building and knowledge of potential career paths. Of course, summer camps have a significant emphasis on student enrichment and learning but there is an additional pressure for organizers to maintain a sense of fun as the timing overlaps with summer break for most students. Meeting these requirements can be a challenge for large camps where students will have diverse needs and expectations, especially for camps that have multiple sections that places students according to their experience.

We host an annual, two-week coding camp at a large midwestern university for high school students between their junior and senior year. It has two sections: a Java section oriented to beginners and a Python one designed for those who have already completed the Advanced Placement Computer Science A (AP CSA) course. AP CSA is generally a Java course that gives students early exposure to CS1 concepts. Due to the significant overlap between the Java section and AP CSA material, the Python camp is provided as an alternative to more advanced students and offers an applied introduction to data science concepts, through language processing, visualization, and artificial intelligence. The emphasis on data science was selected for its approachability through Python and application to visualizations, along with the contemporary relevance of AI-related topics.

As the camp is considered part of an early recruitment activity to introduce students to our honors program, many of the participants are high achievers, and camp organizers seek the aforementioned balance of pushing them to learn while having fun. Historically, students have been placed solely on their completion of AP CSA, but this has proven ineffective in some cases due to many students having self-studied and developed skills beyond AP CSA without taking a course. In recent years, we have allowed some of them to switch sections in order to feel more engaged and have an opportunity to demonstrate their skill. However, we have observed that students' overall experience may suffer if they seek to join the more advanced section before they are truly ready, which can happen as students are not necessarily equipped to assess their own competency. We saw the need for some kind of assessment that would enable instructors to make more informed decisions about section placement but wanted to find a solution that would not feel like a test.

In this work, we discuss a gamified, multi-format approach to assessing students' programming competency levels. The described assessment is given on the first day of camp only to those who have already taken AP CSA, or an equivalent course, or those who express interest in joining the Python section. Student feedback and instructor findings suggest the approach as a viable solution to the section placement problem.

Based on our review, this is the first of a dual-format gamified assessment for programming skill evaluation in high school summer camps using (1) a unified narrative across both formats, (2) cross-language transfer assessment, and (3) an auto-graded digital platform to support sustained learning.

In the next section, we will explore more background about gamified learning as it has been used to boost engagement and support assessment, with a key distinguisher being our use of multiple languages and formats. Then, we discuss more about the activity itself before covering qualitative observations and early analyses on behaviors across languages and platforms based on competency levels, along with their potential implications for future curricular and pedagogical design of summer camps such as our own.

Background

Gamified Learning

Gamified learning has the potential to enhance both the student perspective of the camp and improve programming skill. Previous work that has aggregated results across multiple computer science courses at universities has found that implementing gamified elements such as badges and level progression can increase students' overall performance from their enjoyment and willingness to continue in programming focused activities¹. Furthermore, there is empirical evidence that suggests these motivational benefits can turn into measurable gains. Students who participated in gamified programming environments showed significantly higher cognitive engagement. Also in comparison to students in non-gamified environments students in gamified settings showed statistically significant improvements in post-test performance². Many other studies focused in the classroom support the claim that gamified activities can breakdown complex computer science topics into a simpler form while increasing student enjoyment and comprehension³.

In programming-specific contexts, gamification mechanisms have been shown to produce measurable differences in observable coding behaviors. For example, the introduction of gamified elements in programming environments has been associated with statistically significant improvements in cyclomatic complexity, suggesting that gamified systems can influence how students approach and structure their code, not merely their motivation to participate⁴. Such findings reinforce the idea that gamified environments can generate quantifiable behavioral signals, a pattern echoed in our own results showing that students with higher coding competency engaged with the system in distinctly different ways.

Gamified Assessment

Taking one additional step with gamified assessment, we can engage with students early by recognizing where they are at conceptually. Game-based assessment provides deeper insight into students' knowledge, skills, and attributes that other current methods of assessment may not have the power to measure⁵. Recent literature reviews further emphasize that game-based assessments provide valuable data to educators^{6,7}. In computer science contexts in particular, prior work has shown that game-based assessment environments can offer meaningful insights into students' computer science algorithmic thinking by analyzing in-game problem-solving behaviors rather than solely final answers⁷.

Beyond descriptive insight, recent research demonstrates that game-based environments can generate predictive models of student knowledge using in-game behavioral traces. For instance, early prediction models in game-based learning environments have demonstrated the ability to

infer fine-grained student performance on individual assessment items by combining gameplay logs with semantic representations of assessment questions⁸. These findings suggest that gamified systems hold promise for estimating student performance on specific competencies during gameplay, potentially complementing rather than replacing traditional post-test evaluations⁸. The ability to infer knowledge early and at a granular level provides a foundation for adaptive instructional decisions and skill-based placement mechanisms within gamified environments. Together, these results support the use of gamified approaches as an effective means of enhancing both learner experience and programming skill development in informal learning environments such as summer computing camps.

Placement Assessment

Programs that teach programming often face wide variation in students' incoming preparation, which makes it difficult to align students with an appropriately paced track. When novices are placed alongside more experienced peers, they can face unnecessary obstacles and confidence setbacks. This provides motivation to offer distinct beginner and advanced entry points into the program and use an evidence-based placement to direct students accordingly⁹. To address this, computing-education research has examined how to design placement instruments and decision rules that go beyond transcripts or self-reporting. Nugent et al. describe the design, implementation, and early analysis of a computer science placement test intended to discriminate readiness for introductory coursework¹⁰. More recent work has continued to study strategies for directing incoming students into an appropriate first computing course when multiple starting points exist¹¹. Longitudinal reports of track-based introductory programming courses further show that voluntary placement exams can be iteratively refined over multiple years to support enrollment in an advanced track while maintaining a better fit between student preparation and course expectations¹². Taken together, this literature suggests that low-stakes diagnostic assessment is a standard and defensible response to heterogeneous preparation; however, relatively less work has explored how placement assessments can be embedded in informal contexts in ways that preserve enjoyment and reduce the perception of "taking a test."

Multi-form Assessment

A second design consideration for placement-style programming assessments is assessment mode: while programming is typically learned and practiced in an IDE, many assessments are still administered on paper, raising concerns about "test mode effects" and construct underrepresentation. Studies comparing pen-and-paper and computer-based programming exams report mixed results in overall scores but show that the environment can change the kinds of errors students make and the opportunities they have to revise¹³. Hybrid designs that deliberately combine modes have been used to capitalize on complementary evidence. For example, splitting an assessment across paper and computer¹⁴ or using a two-phase approach where students begin on paper or in an IDE and then all rework solutions with computer support has shown to be beneficial in capturing programming competence¹⁵. Beyond comparability, computer-based formats can reduce transcription burdens and produce analyzable data. Exam environments such as BlueBook illustrate practical mechanisms for administering secure, laptop-based programming exams while controlling access to execution¹⁶. This prior work motivates multi-format

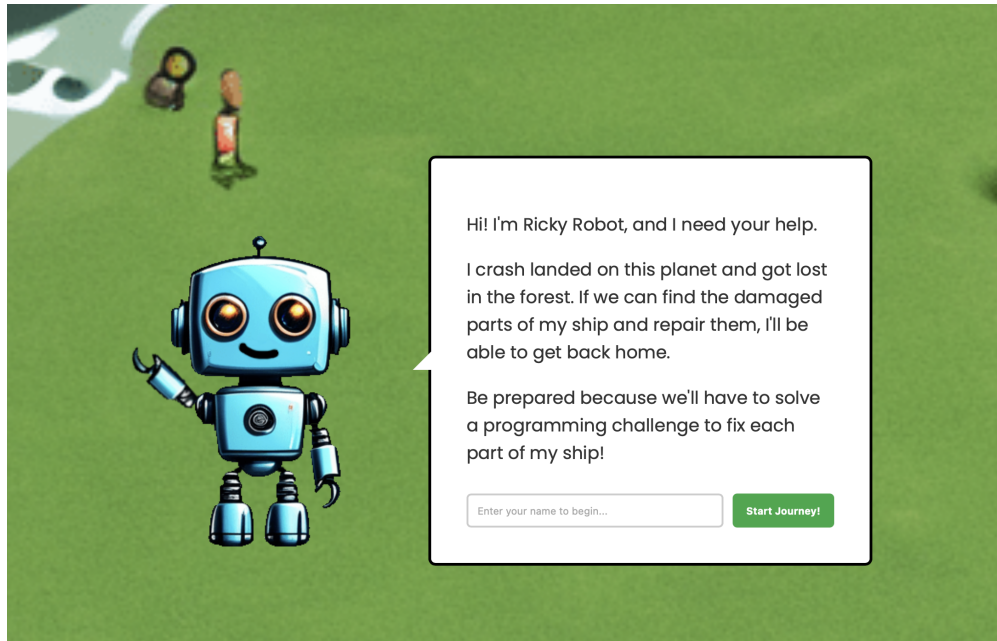


Figure 1: The landing page for “Ricky Robot’s Journey Home” introducing the character and scenario, allowing students to input their name and begin.

assessment designs that intentionally triangulate evidence: a paper-based component can elicit conceptual reasoning, while a computer-based component can capture student signals and process data that are useful for judging readiness programming tracks.

Methods

Activity

We present the game “Ricky Robot’s Journey Home,” a themed programming activity in which players must solve coding challenges to help Ricky the Robot repair parts of his ship and find his way home. The activity exists in two primary modalities, paper and digital, and over two languages, Java and Python.

Java Paper Portion

The first part is a paper-based set of 6 programming questions in Java and presented as a “Programming Skill Builder Challenge.” Java is emphasized initially because of the need for students to have completed AP CSA, which is taught in Java, or an equivalent. This stage of the assessment is only themed around the scenario of helping fix the spaceship without other significant aspects of gamification. Three questions cover debugging and code reading, and the remaining three require code writing, with allowance for pseudocode or flowcharts. Concepts include logic and control flow, math operations, variable usage, and limited arrays and lists. The theme is present throughout each problem’s design, e.g., writing a signal decoder as string processing practice.

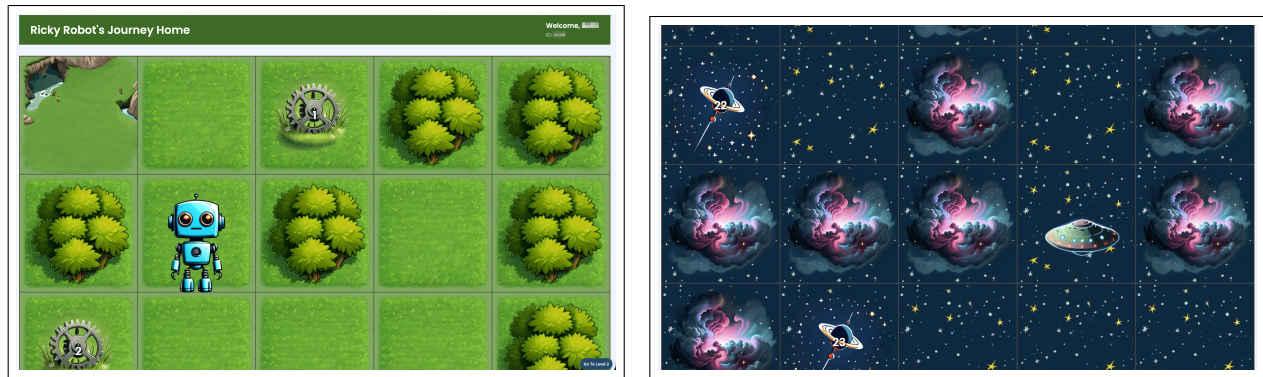


Figure 2: Portions of the level maps with the top showing level 1 set on a forest planet's surface and below with level 2 in space.

Python Digital Portion

Following completion of the Java paper, students were given a short lecture on Python principles. This lecture provided direct mappings from basic concepts in Java to Python syntax (e.g., if-else statements, variable creation and assignment, mathematical operators, etc.). After this, students opened the Ricky Robot game on their machines to begin practicing Python. Figure 1 shows the landing page to the game in which students can read the setting and enter their name to start.

A key distinction of the digital portion from paper is the use of gamification alongside programming problems. The students take control of Ricky the Robot to move around the level map, locating and 'repairing' parts (i.e., completing a problem), and traveling to other challenges as they are unlocked. Figure 2 shows portions of the two levels, the first taking place on a forest planet's surface and the second in space controlling Ricky's spacecraft.

Also unlike the paper, the game used an auto-grading engine with the Javascript-based PyScript. Client-side evaluation was selected to reduce server load and retain a focus on personal learning and development rather than managing student grades or coursework. That said, the results of the unit tests and student code were logged for the instructor to monitor progress.

Problems were designed to be progressively more difficult, covering concepts from simple calculations up through manipulation of lists and objects. Not all problems had to be solved. For instance, level 1 featured seven problems in all across 5 difficulties: 1; 2 and 3; 4; 5 and 6; and 7. Thus, students all had to complete problems 1, 4, and 7 but could make a choice to skip or only do portions of 2, 3, 5, and 6. All that was required to unlock or 'locate' the next problem(s) was successful completion of 3 unit tests across the current difficulty, which may be split across problems. This design was adopted to encourage students to try different problems, and even move between them, in the manner that felt most comfortable.

Students click on a problem to open a modal with a code viewer, test status report, and output console for printing and debugging. Ricky or the spaceship appear alongside, with Ricky's expression and pose changing based on progress. Figure 3 shows several examples of questions and corresponding status of tests in the problem modal.

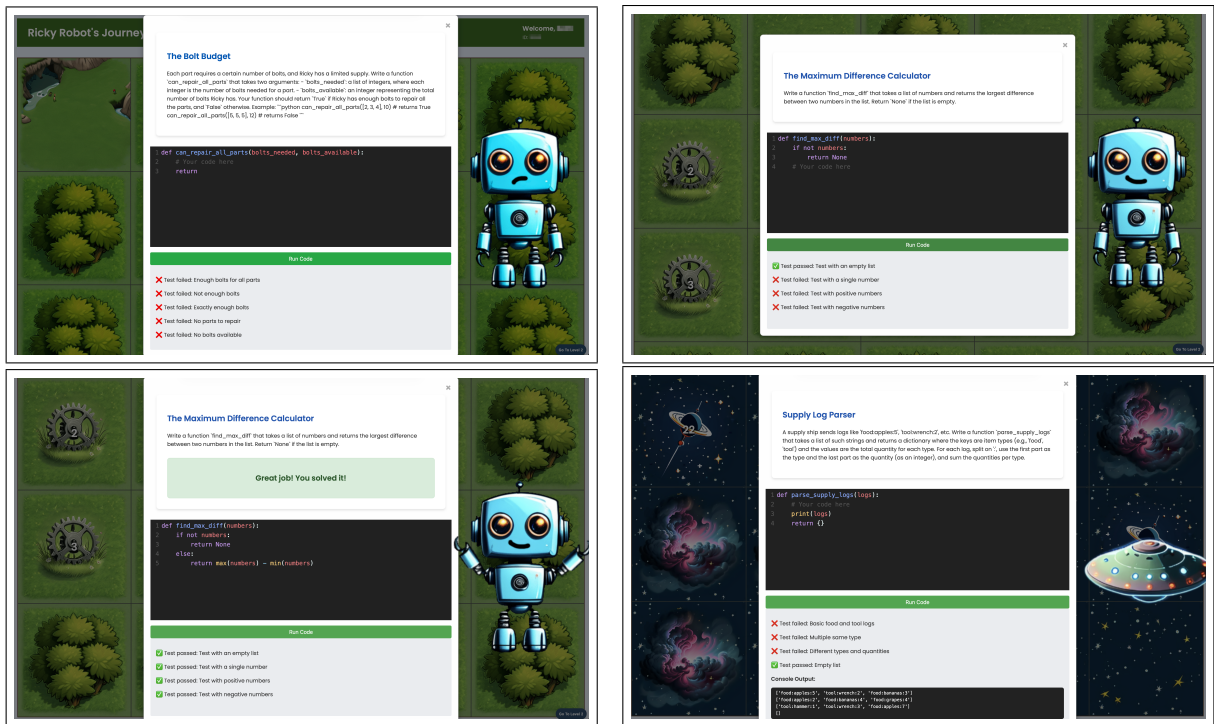


Figure 3: Examples of the problem modal. From upper left, going clockwise, we see a question with no completed tests, a test with one completed test, a test with all completed tests, and a test from the second level showing debug info printed in the console.

Completing the last problem moved students to the next level. Level 2 follows the theme with Ricky having repaired the ship and taken off from the planet to now be searching message buoys in the nebula to locate the wormhole home. While the first level is conceptually similar to the Java test in terms of topics, level 2 builds more on fundamental data structures, notably working with dictionaries as that is an important construct for the data science curriculum in the camp. For this reason, level 2 is actually locked until the second day of camp after a lecture on dictionaries is given. In this way, the activity aims to build from purely focusing on assessment to sustaining engaged learning through gamified challenges.

Analysis

After the initial paper submission, the instructor reviewed all students responses to the 6 programming questions. From a high level, these were grouped into buckets based on comparative completion, correctness, and conciseness of code, creating ‘Beginner,’ ‘Intermediate,’ and ‘Advanced’ levels of demonstration of coding competency. It is important to note that these findings were never shared to students, and for those who had already completed AP CSA, they had no bearing on camp section. They were only used to assess the few students who were seeking to move into the accelerated section, and in all cases, we found they had enough background to do so; simply the presence of a short set of problems seemed to address the primary purpose of dissuading many students from wanting to move. A compounding factor that bears mention is that the first day of the Java camp is a no-code introduction that sometimes

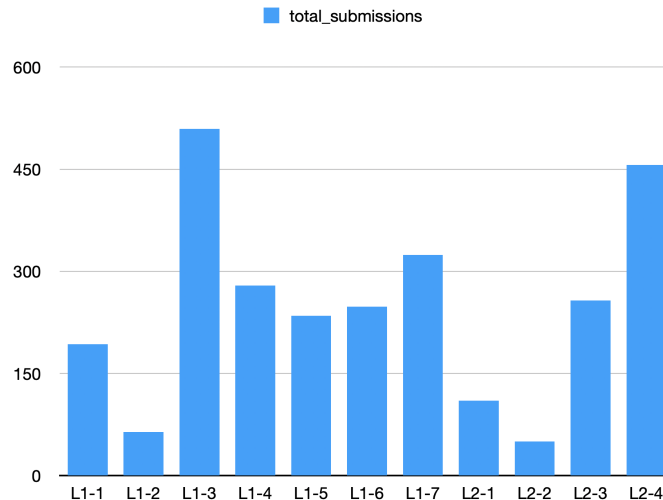


Figure 4: Submissions to the system grouped by problem number.

misleads students with even a little coding experience about the nature of the remaining days. The only impact to students who were already in the Python camp was combining these observations with how they engaged during the lectures and activities, leading to one or two check-ins with students to make sure they didn't feel they were falling too far behind.

In the next section, we share a few aggregate results based on submissions to Ricky Robot and qualitative observations from the instructor about the experience. Charts are derived from logged data in aggregate, and we focus only on time spent and number of submissions, while we leave a deeper dive into the specific code progression analysis for a future work.

Results and Discussion

Aggregate Submission Metrics

In Figure 4, we show the submissions received by problem number. Clearly, there was a difficulty spike at problem 3 in level 1, which may have been caused by students seeking to skip problem 2 as both 2 and 3 unlock simultaneously. This is an interesting phenomena as it is perhaps due to our desire to let students choose the challenges they complete but the numerical ordering accidentally encouraging them to skip ahead. Average time spent matched fairly closely with the trend shown in total submissions, with times ranging from about 19 minutes on level 1, problem 3 to only about 2 minutes on level 2, problem 2, with the beginning of level 2 being quite easy as it focuses on basic dictionary syntax and access.

Next, we consider the submissions and time spent across the aggregated three competency levels (Beginner, Intermediate, Advanced) observed from the Java portion of the activity. Figure 5 shows these results. Interestingly, we see an increasing trend in number of total submissions while there is inconsistent behavior in the time spent.

Shapiro-Wilk testing confirmed normality of all data points (total time spent and total submissions across all cases), so we applied single-factor ANOVA to both dimensions

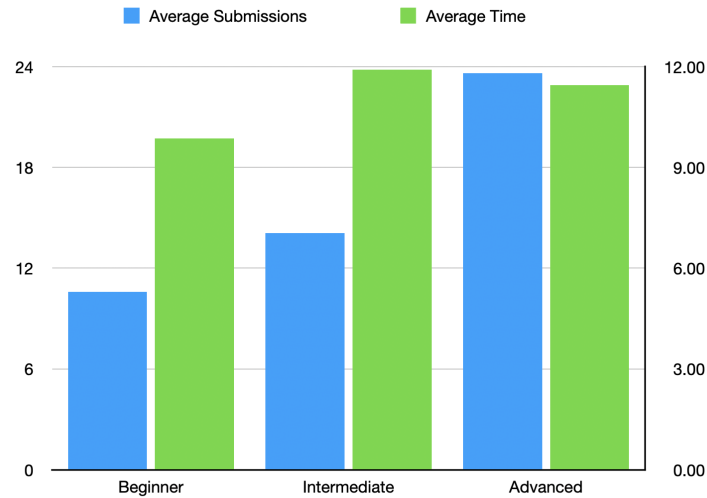


Figure 5: Submissions and time spent in Python across the competency levels exhibited in Java.

individually. ANOVA revealed that there was a statistically significant difference across the total submissions in the three groups, $F(2, 13) = 3.86, p < 0.05, \eta^2 = 0.37$, while there was no statistically significant difference across the total time spent,

$F(2, 13) = 2.94, p \approx 0.09, \eta^2 = 0.31$. In the case of submissions, post-hoc pairwise Welch's T-tests showed that the only statistically significant difference was between Beginner and Advanced, $t(8) = 2.65, p < 0.05$, with a mean difference of 160 submissions. There were no pairwise comparisons that were statistically significantly different for time, although the mean differences are notable: 3.21 minutes between Intermediate and Advanced but 44.73 and 41.51 between Beginner and Intermediate and Advanced, respectively. Of course, these are extremely high-level aggregate metrics, and it is important to acknowledge the limited sample size, with data of only 16 summer campers represented in all. Nevertheless, they point to some noteworthy trends that may be worth future investigation in terms of analyzing code progression.

For now, we would make the case that the students who were more familiar with coding were less afraid to submit frequently to the system. The lack of statistically significant differences in total time while the number of submissions increased seems to indicate that those with stronger coding competency were more optimal with their time, by rapidly checking to see if their code worked. From an instructor's perspective, there is value in beginner students taking more time to understand the problem and submitting only when they feel ready. Concurrently, given that the platform uses unit tests, it may be worth encouraging students to practice problem solving by dividing the challenge into smaller pieces; this may encourage them to be more willing to solve parts of a challenge, ask for feedback, and iterate rather than attempting to tackle the entire challenge at once.

Classroom Observations

The coding assessment was specifically presented as a skill building challenge in an effort to avoid the sense of taking a test. For the most part, this strategy seemed to work, but we benefited

from doing it at the beginning of the first day to rapidly get any of the work that could feel like an exam out of the way. By the time we moved on to the digital activity, students had begun to meet each other and work with neighbors. We made no explicit effort to stop this and rather preferred to watch the campers develop their own sense of work, accomplishment, and responsibility over the course of the exercise. At first, there may have been some gender differences, with more of the girls opting to work together as there were slightly fewer of them in the camp overall. However, by the second level, after we had covered the concept of dictionaries, almost everyone found themselves enjoying the activity enough that they preferred to work through the problems on their own.

In fact, the original plan for the exercise was that there would only be the Java problems on paper and single Python level. At the end of the first day, one of the girls in the camp specifically requested that we do more Ricky Robot activities. Several students expressed that they saw these as a good way to practice the concepts, indeed viewing it as more of a skill builder activity, in alignment with the overall goal. There may be something to be said about paper versus digital in terms of how the coding problems are viewed, with an interactive coding modal capable of auto-grading having distinct advantages over handwritten code on paper. There may also be the influence of how we utilized this activity, with it more as a support for monitoring student progress than reporting any scores or grades. In either case, the comments from students, the requests for more levels, and the observed shift to more of them wanting to ‘play’ the character for themselves, supports that wrapping the problems in the game scenario helped reduce the sense of pressure. The sample size is admittedly small, so our claims in this work are more to the general positive nature of the experience and serve to motivate further exploration of game-based assessment and learning, with the two at least partly intermingled through a system such as the one used in our approach.

Next Steps and Conclusion

In this work, we discussed the implementation and application of a two-part set of coding challenges, themed around the scenario of helping a robot find the way home. One part practiced Java on paper with the next using Python on a web-based tool capable of interactive coding and auto-grading. While we observed some promising trends and received positive feedback, there were several limitations in terms of lack of control for external factors and a small sample size. In some ways, the activity has already achieved the purpose for which it was created—to help manage student placement into different camp sections. However, there are many possible next steps. One would be the exploration of code progression itself if we were to look beyond high-level, aggregate observations from instructors, and another would be re-running the activity in the upcoming summer camp with a larger pool of participants and potentially more levels.

In closing, we will reiterate that the second level of the game was created specifically at the request of campers. This perhaps more than any other metric speaks to the potential of including such activities alongside lectures, especially as they introduce new content. Also, teaching students more about unit testing and encouraging breaking a problem into parts could be a worthwhile enhancement based on the patterns we observed with average number of submissions. We look forward to exploring more ways to engage the high school students through games and code-centered skill building challenges!

References

- [1] Risna Oktaviati and Adam Amril Jaharadak. The impact of using gamification in learning computer science for students in university. *International journal of engineering & technology*, 7(4.11):121–125, 2018.
- [2] Maria-Blanca Ibanez, Angela Di-Serio, and Carlos Delgado-Kloos. Gamification for engaging computer science students in learning activities: A case study. *IEEE Transactions on learning technologies*, 7(3): 291–301, 2014.
- [3] HM Jawad and S Tout. Gamifying computer science education for z generation. *information*, 12 (11), 453, 2021.
- [4] Remin Kasahara, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa. Applying gamification to motivate students to write high-quality code in programming assignments. In *Proceedings of the 2019 acm conference on innovation and technology in computer science education*, pages 92–98, 2019.
- [5] Sebastiaan de Klerk and Pamela M Kato. The future value of serious games for assessment: Where do we go now? *Journal of Applied Testing Technology*, pages 32–37, 2017.
- [6] Manuel J Gomez, José A Ruipérez-Valiente, and Félix J García Clemente. A systematic literature review of game-based assessment studies: Trends and challenges. *IEEE Transactions on Learning Technologies*, 16(4): 500–515, 2022.
- [7] Qi Luo and Shuhan Zhang. Game-based assessment for computational thinking: A systematic review. In *2024 IEEE Integrated STEM Education Conference (ISEC)*, pages 01–08. IEEE, 2024.
- [8] Andrew Emerson, Wookhee Min, Roger Azevedo, and James Lester. Early prediction of student knowledge in game-based learning with distributed representations of assessment questions. *British Journal of Educational Technology*, 54(1):40–57, 2023.
- [9] Daniel Archer, Leo C. Ureel II, and Charles Wallace. The impact of placement strategies on the success of students in introductory computer science. *Proceedings of the Frontiers in Education Conference (FIE)*, 2017.
- [10] Gerald Nugent, Leen-Kiat Soh, Ashok Samal, and Jeff Lang. A placement test for computer science: Design, implementation, and analysis. *Computer Science Education*, 2006.
- [11] Leo C. Ureel II, Charles Wallace, and Daniel Archer. Directing incoming cs students to an appropriate introductory computer science course. *Proceedings of the Frontiers in Education Conference (FIE)*, 2020.
- [12] Stefan Podlipnig. A four-year study of a placement exam for an introductory programming course. *Proceedings of the ACM SIGCSE Technical Symposium*, 2022.
- [13] Johan Öqvist. Coding by hand or on the computer? evaluating the effect of assessment mode on performance of students learning programming. *Journal of Computers in Education*, 2018.
- [14] Mohammad M. Haghighi, James Peterson, and Michael Reilly. Summative computer programming assessment using both paper and computer. *Proceedings of the International Conference on Computers in Education*, 2005.
- [15] Ville Lappalainen, Antti-Jussi Lakanen, and Heikki Högmander. Towards computer-based exams in cs1. *Proceedings of the International Conference on Computer Supported Education (CSEDU)*, 2017.
- [16] Chris Piech and John Gregg. Bluebook: A computerized replacement for paper tests in computer science. *Proceedings of the ACM SIGCSE Technical Symposium*, 2018.