education sciences

MDPI

Article

Motivating Students to Learn How to Write Code Using a Gamified Programming Tutor

Simon Grey and Neil A. Gordon *

School of Computer Science, University of Hull, Hull HU6 7RX, UK
* Correspondence: n.a.gordon@hull.ac.uk; Tel.: +44-1482-465038

Abstract: Engagement and retention are widely acknowledged problems in computer science and more general higher education. The need to develop programming skills is increasingly ubiquitous, but especially so in computer science where it is one of the core competencies. Learning to write code is a particularly challenging skill to master, which can make retention and success even more difficult. We attempt to address student engagement within an introductory programming module by attempting to motivate students using a gamified interactive programming tutor application that provides immediate feedback on the student's work. In this paper, we describe the design of the gamified programming tutor application, along with a related topology to characterize student engagement. We discuss the design of the software, the gamified elements, and the structured question design. We evaluate the engagement with the gamified programming tutor of two cohorts of students in the first year of a computer science programme, with over two hundred students taking part. We attempt to frame this engagement in terms of frequency, duration, and intensity of interactions, and compare these engagement metrics with module performance. Additionally, we present quantitative and qualitative data from a survey of students about their experience using the programming tutor application to demonstrate the efficacy of this approach.

Keywords: gamification; motivation; engagement; computer science; programming; learning; teaching

Citation: Grey, S.; Gordon, N.A. Motivating Students to Learn How to Write Code Using a Gamified Programming Tutor. *Educ. Sci.* **2023**, *13*, 230. https://doi.org/10.3390/educsci13030230

Academic Editor: Han Reichgelt

Received: 5 December 2022
Revised: 7 February 2023
Accepted: 14 February 2023
Published: 22 February 2023

1. Introduction

With the advent of the Fourth Industrial Revolution, programming has become an increasingly key competency across many disciplines [1]. However, learning how to program is a challenging skill to master [2]. Becoming competent in programming requires a student to master several challenging skills simultaneously [3]. To solve any significant problem, a student must first understand the domain of the problem they are trying to solve; then, they must break the problem down into a sequence of small steps. In order to express these steps, they need to articulate their solution to the problem using semantics, which can be ambiguous as whilst some terms have very specific meanings, some are used interchangeably; then, they must convert these semantics into specific syntax that a computer can understand. Then, they should test their code to proactively test edge cases and seek out errors, and when these errors inevitably arise, they must embark on a systematic process of debugging and testing their own assertions. Given this process, it is no wonder that many students who are new to programming struggle to grasp the concepts [4].

In this paper, we present the motivation, design, and implementation of a gamified programming tutor. Whilst others have used similar approaches [5–7], this approach is novel in terms of the way that the module content is tied to the activities, which is used to support formative and summative assessment [8], whilst gathering data on students' learning to allow for domain-relevant learning analytics. Here, the approach adopted is really a hybrid or integrated assessment type, where the assessment has an initial formative

role but does contribute towards their final module summative mark [9,10]. We go on to provide an analysis of and reflection on this gamified programming tutor, which is intended to assist students in learning how to program.

For the remainder of this introduction, we set out the background and context presenting a brief overview of the issues of engagement and retention in higher education generally, and when teaching computer science specifically. We set out a definition for gamification and draw inspiration to create metrics to evaluate engagement, and we present some background information on existing research into programming tutors. Following this, in Section 2 we outline the design and implementation of the programming tutor application, and in Section 3 we present and analyse data from two cohorts of students using the tutor. Conclusions and directions for further work are given in Section 4.

1.1. Engagement and Retention

Engagement and retention are both of significant concern in higher education across all disciplines [11], and in computer science especially [12]. Increasing numbers of students are stretching resources thinly, and a more diverse student cohort in terms of ability is making supporting individual students' needs more challenging [13]. Woodfield reports that in the United Kingdom retention is particularly problematic in computer science, where the continuation rate is lower than average at 91%, and 6% of students leave with no award [11]. Computer science is also an area of significant growth, with increasing numbers of students necessitating a more efficient distribution of resources.

The diverse nature of incoming computer science students has been famously described by Dehnadi and Bornat [14]. Whilst they have since retracted some of the controversial claims made in their paper regarding the ability to predict the success of students before they begin, the issue of cohorts that have diverse abilities persists. Moreover, it is likely that the range of abilities is widening as increasingly more students are exposed to computer science at previous levels of education, whilst some still enter higher education with no experience at all. Meeting the competing needs of this community presents significant challenges [15].

1.2. Gamification

Gamification is defined by Deterding et al. as "the use of game design elements in non-game contexts" [16]. In the context of gamification, game mechanics typically include the trinity of points, badges, and leader boards. However, there are other game mechanics that can be included. Allowing multiple attempts of an assessment lowers the risk and is broadly equivalent to games allowing the player to have extra lives. Sheldon takes inspiration from role-playing games in which players cumulatively build experience, and delivers a course where students accumulate grades rather than displaying grades as averages. This results in an identical grade but is subtly distinct from using averages, as when generating grades cumulatively a student can never put effort into their work and have their grade decrease as a result [17]. The interplay between games, learning, and teaching is an interesting one. Gordon et al. [18] describe using natural game mechanics with datasets that predate the popularisation of gamification. This alludes to the fact that good games can be framed as a set of engagement tools that attempt to address the same motivational drivers that teaching does, and common themes can be seen throughout. Specifically, the concept of flow [18] requires that a person's skill level is well-matched to the level of challenge that they face, and that feedback is delivered quickly and accurately. Giving choices allows a degree of autonomy and presenting tasks that are clearly relevant provides a sense of purpose [19].

Engagement, and the measurement of engagement, is very subjective, as is making comparisons between levels of engagement [20]. In this paper, we draw inspiration from Zichermann and Cunningham, who propose that engagement can be scored using a combination of measures of recency, frequency, duration, virality, and ratings [21]. In the case of the programming tutor, recency and virality are not relevant. It could be considered appro-

priate to monitor recency whilst the module is running, but less so after the module has finished. In this paper, we measure frequency and duration. The addition of quantitative and qualitative data via a survey is included as a proxy for ratings.

2. Materials and Methods: Gamified Programming Tutor Design

This section presents the design considerations of a programming tutor application. It is split into three sections, each considering a different aspect of the designed learning experience. Section 2.1 describes the software design, and some considerations that led to the specific implementation of the programming tutor. Section 2.2 describes the design of the gamified elements of the programming tutor. Finally, Section 2.3 presents the approach adopted to design the learning content.

2.1. Software Design

The programming tutor software architecture design is an evolution of the architecture proposed by Gordon et al. [22] to deliver a limited set of functionalities. Practical considerations included the need to deliver content and collect usage data. This was achieved by using an existing source control system (Subversion) that allowed for the collection of rich data every time the student opened the tutor, used the tutor to attempt to solve a problem, and closed the application. Content was delivered from Subversion via a series of xml files. The xml files included a set of teach materials on topics split into logical sections, and a series of challenges that allowed highly scaffolded and restricted code to be edited by the user.

Figure 1 shoes an example of the programming tutor. The material on the left-hand side describes a particular concept, and includes challenges which, when selected, are displayed in the box on the right. The grey areas are not editable, and the white areas are editable. When the user pressed the "Compile" button in the top right, the code was compiled and run locally on the machine, but a series of test cases were exercised by injecting code before and after the editable section. This drove instant feedback that was given in the black box towards the bottom right.

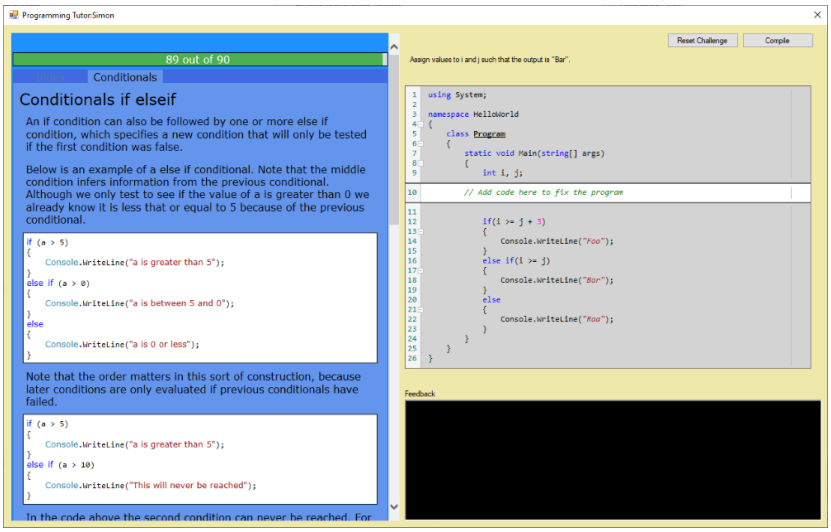

Figure 1. The programming tutor application.

2.2. Gamification Design

The software was gamified in the following ways. Students were allowed unlimited attempts at each problem. As mentioned previously, this lowers the risk faced by the student and is similar to a game allowing players to have extra lives. This can be considered formative, in that the students receive the feedback on their progress and can then revise their answer and try again. However, it is also summative, in that their highest mark from this activity is part of their overall module mark.

The programming tutor tested the code according to a set of test cases and gave instant and immediate feedback as to whether the test passed or failed—including the input and expected output.

When all of the tests for a challenge were passed at the same time, then the student was awarded a badge in the form of a gold star. The student could also see a visual representation of their progress through the programming tutor via a progress bar at the top of the window.

As a further award, at various points, students unlocked new content—so successful attempts to overcome challenges in the programming tutor were rewarded with more challenges. Critically, students did not have to complete all challenges in a previous section to progress, which is a technique used in game design to try to prevent players from disengaging through frustration.

Attempts were made to ensure that the language used throughout the application and the module were consistent with a playful attitude—for example, the students' activities were contextualised as challenges, rather than tests or problems, though they could be considered as directed problems. The opportunity to complete them in different orders, and the flexibility in not having to complete all of a section to progress, were intended to provide a more engaging approach.

2.3. Teaching Design

The module that used the programming tutor application was an introductory programming module. The programming tutor provided students with 90 individual challenges split into six sections. Details of the sections, the topics they covered, and the distribution of challenges is provided in Table 1. The sections can be considered as different aspects of creating software, and thus as problem components in a divide and conquer approach.

Table 1. Sections, topics, and number of challenges in the programming tutor application.

Section	Topics	Number of Challenges
Introduction to Programming Tutor	Introduction to the module and the programming tutor. General advice on learning to program, getting help, and, of course, "Hello, World!" (1)	1
Storing and Manipulating Data Using Variables and Expressions	Declarations (3), types (3), expressions (6), Boolean logic (8), and enumerated types (2)	22
Making Decisions Using Conditionals	If (3), else (5), else if (6), nested conditionals (5), scope (1), and switch statements (3)	23
Repeating Things Using Loops	For (4), while (6), nested loops (3), scope (1), break (1), and continue (1)	16
More Datatypes—Strings, Arrays, and Structs	Arrays (6), strings (5), and structs (5)	16
Reusing Code Using Methods	Method calling (3), method declarations (4), and passing parameters by value and by reference (5)	12

When designing challenges, often, introductory challenges lead to subsequently more difficult challenges. For example, they often begin with a challenge which has code already written, and the student must select values to obtain a specific output. In the next level of the challenge, students had to edit existing code to obtain a specific result. Finally, the most difficult challenges asked that students write code to solve a problem from scratch. This approach was intended to adapt Skinner's [23] behaviourist model, where the challenges provided a framework for students to develop and demonstrate their understanding of programming concepts. As mentioned in the previous section, this did allow for flexibility and choice, in that progression between sections was not dependent on completing a prior section.

Alongside the programming tutor, students also attended regular lectures and were guided in coding workshops and more independent coding lab classes with demonstrator support.

In challenging times when resources are stretched, one goal is to seek out the students who are trying, but struggling, and are reluctant to ask for help, which could be attributed to imposter syndrome [24].

With the growth in higher education, leading to more diverse cohorts—in terms of subject experience, qualifications, and attainment [25]—there is an increasing need for individualised support. Providing such support is made more challenging by this growth in numbers, where teaching staff may not have the time to give routine and direct guidance on the individual level. Identifying students who need assistance, and automating some of that support, is one way to approach this [26]. Figure 2 shows a proposed topology of student engagement and ability and proposes the following questions: How can we identify where a student lies on this topology? What proportion of available resources should we spend in each quadrant? What sort of actions should we be taking in each? It is proposed that it is likely that most of the resources should be spent in the bottom right quadrant, helping students with high engagement but who are struggling to grasp the topic. Students in the top right quadrant should be engaged with different sorts of tasks to provide appropriate challenges that stretch their abilities and prevent them from moving to the left. It is likely that these sorts of tasks require fewer resources, and the students are probably skilled enough to engage in tasks independently. It is also worth mentioning that students in the top right quadrant are probably more likely to seek out assistance and are rewarding for staff to engage with, giving a misleading view of the cohort, and can become a time sink if they are allowed to. The bottom left quadrant is also a significant concern, and it is proposed that different types of interventions should be made to move these students to the right, before attempting to move them upwards.

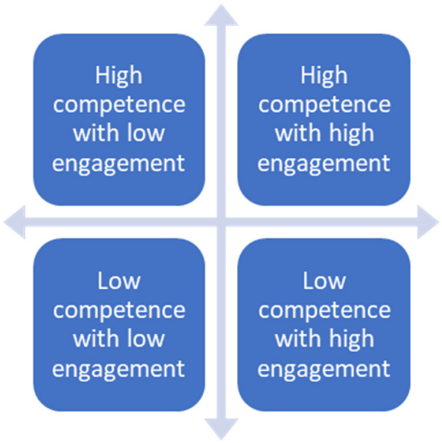

Figure 2. Proposed topology of student engagement and ability.

2.4. Data and Methodology

This work was carried out as action research. This creates some constraints, as discussed in the conclusions. The participants were chosen based on convenience sampling, i.e., the authors applied this work within modules that they were scheduled to teach. The work underwent ethical review within the institutional/faculty research ethics process, which allowed for the collection of the anonymised data and its processing. The data were collected during two sessions—2018/19 (n = 161) and 2019/20 (n = 76). The module was a trimester one module, so it was completed prior to the COVID lockdown of 2020.

2.5. Student Engagement

Measuring the number of successful challenges, however, was not the main goal of the programming tutor. The tutor was designed to help us to measure engagement with learning material during the module, rather than afterwards. In fact, ideally, we did not want to spend a great deal of time on students who were successfully navigating the material. Instead, we wanted to find those students who were highly engaged but struggling. The goal of the programming tutor application was to help better distribute the

limited resources available. We wanted to find and identify students with high engagement but who were struggling, and to offer help where it was needed.

Previously, we identified the concepts of duration and frequency that can be linked to engagement. For duration, we proposed that we estimated the length of time a student used the programming tutor for by measuring the length of sessions. Log data from Subversion give snapshot points in time. Intuitively, a session could be defined as the length of time between the application being opened and closed; however, it was likely that this would give erroneous results, for example, if the application was not recorded as closed because of a network failure, or if a student left the application open whilst performing unrelated tasks. Instead, it was proposed that a session was characterised as a period during which the student was continuously interacting with the tutor, within some threshold (for example, 15 min). It was proposed that the session be measured in units of minutes.

For frequency, we proposed that we counted the number of sessions in each week, and the number of interactions in each week.

3. Results

In this section, we present the results from two cohorts of students. In this section, we present an analysis of the students' interaction with the programming tutor and consider whether it was an effective tool for teaching. The data presented are collected from two cohorts of students who used the programming tutor application in 2018/19 and 2019/20. The programming tutor has not been used since then because students studied at home as a result of the pandemic and supporting the tutor on home machines was prohibitive.

3.1. Analysis of Successfully Completed Challenges

Figure 3 shows a histogram of the number of challenges completed by each student in the two cohorts. Data are presented from 161 students in 2018 and from 76 students in 2019. From the 2018 cohort, 161 students completed challenges in the programming tutor, averaging 78/90 challenges. During this year, the performance in the programming tutor application contributed to 40% of the module marks, replacing a significant piece of assessed coursework that was becoming problematic for staff to mark in a timely manner.

In the 2019 cohort, 76 students completed challenges in the programming tutor, completing an average of just 36.6/90 challenges. During this year, the performance in the programming tutor application contributed to just 4% of the module marks which were presented as part of the significant assessed coursework that had been replaced, in part because of disappointing exam results the previous year. It is proposed that fewer students engaged effectively with the programming tutor because it was worth less marks, and because they were preoccupied by the larger assessment that was worth more marks.

In both years, the remaining 60% of the marks were awarded for performance in two exams. There was a focus on assessment literacy throughout the exams. The exams were open book and focused on the code comprehension of given code, and programming problems that students needed to solve during the exam.

Figure 4 shows the number of challenges completed in the programming tutor application, plotted against their performance in the exams.

3.2. User Rating

As a proxy for a user rating to obtain further information and insight regarding the efficacy of the programming tutor software, students were asked to volunteer to fill out a short survey about their experience as part of a subsequent module. The survey was available to all students in that module from the 2019 cohort, who were the same students as those in the introductory programming module. Twenty students chose to fill out the survey. This subsection will present the results of that survey.

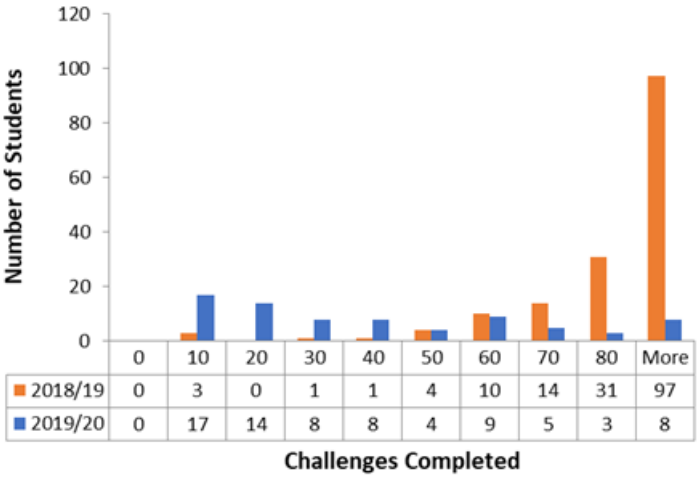

Figure 3. Histogram of number of challenges completed by students in 2018 and 2019 cohorts.

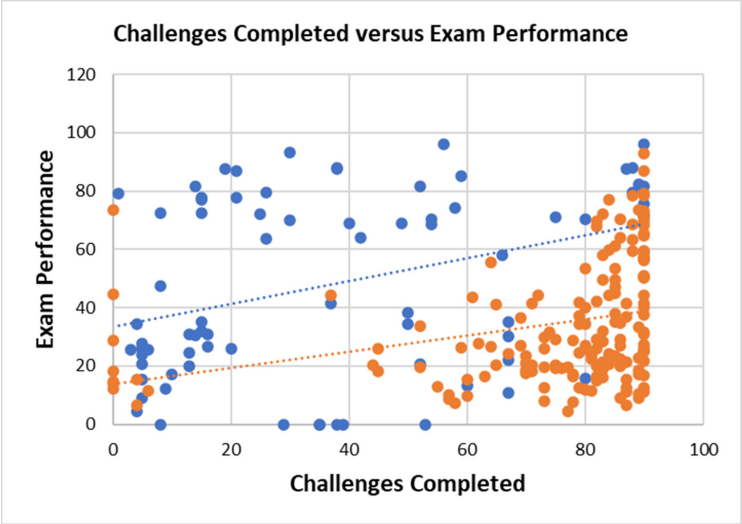

Figure 4. Number of challenges completed versus exam performance of students in 2018 (orange) and 2019 (blue) cohorts.

The survey consisted of nine Likert-style questions, and one question where students rated their preference for different question types.

Table 2 presents the nine Likert statements that were given to students, as well as the average response and standard deviation, where "Strongly Disagree" was encoded to a score of zero and "Strongly Agree" was encoded to a score of four. As described earlier, there was a diverse range of abilities of students upon entry to the degree programme, so it should be of no surprise that the largest amount of variation is manifested in the question concerning initial programming ability. It is reasonable to think that students with very different programming abilities might have experienced the programming tutor application in very different ways.

In Figure 5, students have been divided into two groups based upon their initial abilities, and the average response to each Likert question is displayed. Of the 20 students who filled out the survey, 9 reported having little experience with programming before using the programming tutor, whilst 11 said that they had had previous experience. It is surprising that both groups reported very similar experiences of the programming tutor application, regardless of initial experience. There are some small discrepancies which are not surprising. The whole experience seems like it was more novel and exciting for inexperienced students, and more experienced students seemed to find the written feedback less useful but valued the sense of achievement provided by the progress bar. It is also

perhaps not surprising that the inexperienced students seem to have reported that they were less confident in the code writing problems than students with more experience.

Table 2. Sections, topics, and number of challenges in the programming tutor application.

Statement	Average Response Between 0 (Strongly Disagree) and 4 (Strongly Agree)	Standard Deviation
Prior to taking this course I had little or no knowledge of programming	2.3	1.45
Including challenges for each individual concept helped me to learn each concept in isolation	3.3	0.62
The feedback when my program was incorrect was useful	2.85	0.89
The gold star feedback when my program was correct felt good	2.9	0.81
The overall progress bar motivated me to complete all of the challenges	3.15	0.99
I would have completed the programming tutor even if it carried no marks	2.2	0.96
I found that the programming tutor was easy to use which encouraged me to use it	3.15	0.77
The progression of the questions helped me to understand each concept	3	0.62
Writing code to solve small problems gave me confidence that I understood the concept fully	2.8	0.96

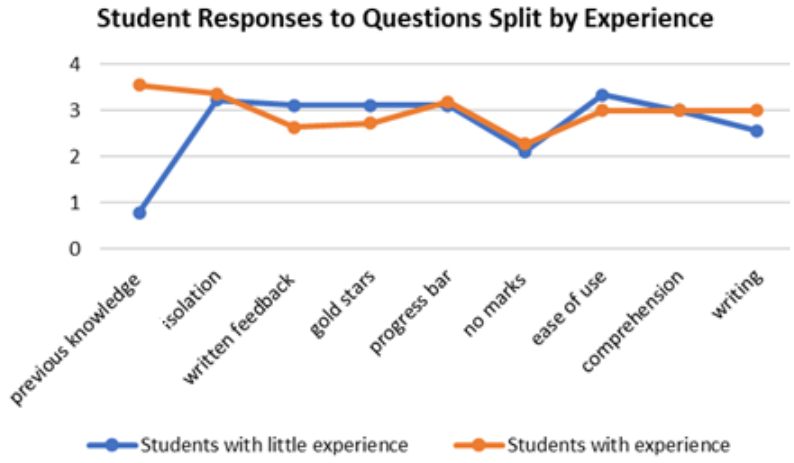

Figure 5. Students' views on the usability/gamified elements of the programming tutor.

In addition to the Likert-style question, the survey ended with a freeform question asking students for any other insights, criticism, or feedback related to the tutor. Freeform feedback was generally positive, and some students touched on the gamified elements and question design.

> *"I really enjoyed it, it helped me to develop the basic skills of programming. I'm still struggling now but, the tutor was definitely a great addition to the module. Very helpful for me."*
>
> *"Overall, I found the programming tutor to be instrumental in my passing the module."*
>
> *"I found that the programming tutor was more useful for learning than the labs were, without it I would have probably failed the coursework."*
>
> *"It was a nice addition to the coursework. Really liked it!"*
>
> *"The star thing and the progress bar make the programming tutor look and feel like a game."*
>
> *"The challenges in which you have to write the code from scratch are very good to finish a topic with."*

The only criticisms of the tutor concerned aesthetics and usability:

"The 'reset' and 'compile' buttons were identical in appearance and very close, which caused me to accidentally press the wrong one a few times."

"Knowing exactly what's left to complete. Maybe accessible via short cuts."

"I think my main criticism of the program is aesthetics."

"Perhaps there could have been links to resources based on each topic if you get stuck to provide some extra support on the specific section that you may be stuck on."

3.3. Further Discussion of Results

Figure 3 shows that the 2018/19 cohort was keen to attempt the challenges/activities, and there is a clear weighting toward the higher number of challenges. The 2019/20 cohort shows quite a different profile, with more of a weighting toward the lower numbers. This may indicate issues with the way that labs were run, and some possible issues with general engagement by students that year. However, as noted above, the key difference seems to have been the lower weighting toward marks for completing the programming tutor challenges. This reinforces the notion that student behaviour is frequently driven by extrinsic factors such as marks.

Figure 4 contrasts the attempts at the challenges with overall final assessment (exam) performance in the module, for the 2018/19 and 2019/20 cohorts. In both cases, there is a small positive correlation between the challenges completed and the exam performance, but it seems unlikely that this is as a result of the programming tutor, and more likely to be due to previous experience or other teaching activities. It is worth noting that the exam performance in 2019 was significantly better, when students completed a larger amount of coursework alongside the programming tutor, and as a result engagement with the programming tutor was diminished.

Table 2 and Figure 5 provide some indication of students' views of using the programming tutor. What seems significant is the relatively negative response to the question as to whether students would have engaged with the programming tutor had it been a purely formative exercise, with no marks to provide extrinsic motivation. This adds to the above discussion about the way that students are responsive to marks as a primary way of encouraging their behaviours.

The free-text comments show that students had some positive views, though these likely came from those who opted to do many of the challenges. The main negative comments were about the design of the user interface for the programming tutor, something that we would look to address in future applications.

4. Conclusions

In conclusion, it seems that this programming tutor application was a popular addition to the programming course that, initial overhead notwithstanding, did not require a great deal of resources to run. However, in 2018 when the tutor replaced a significant assessment with several very small assessments, it was unsurprising that many students engaged more with the programming tutor and scored high marks, because they were allowed unlimited attempts in an open environment. Unfortunately, however, the experience did not seem to result in the same level of comprehension that creating a single larger application did.

Other criticisms included that the small nature of the challenges meant that many suffered from a lack of context, perhaps preventing students from seeing and experiencing the bigger picture. It seems that students suffered from not being able to put the whole thing together.

From the feedback survey, it seems that the tutor was very well received, but it must be acknowledged that the number of students who took the survey was low, and those students were likely to be the more highly engaged students within the group. It is encouraging though that there was an even split between students with previous experience and those with no previous experience, and that their assessments of the programming

tutor application were very similar. In terms of self-selecting, students who engaged more frequently and for longer with the programming tutor did better. Again, a significant result from the survey is the revelation that even though it seemed that they enjoyed the challenges posed by the programming tutor application, and even though the tutor was worth a very small summative amount, it seems that students would be less likely to engage had it been a purely formative exercise.

Throughout the course of those two years, just under 140,000 interactions were recorded, presenting a significant amount of data that provide opportunities for further analysis. Another interesting avenue of investigation would be to investigate the patterns of behaviour. This has been touched upon previously. In one study, it was found that students' engagement with source control peaked just before and after scheduled teaching labs, leading to the hypothesis that more consistent engagement might be achieved by timetabling shorter, more frequent activities [27]. As mental health and well-being become of increasing concern amongst our students, it is also proposed that monitoring the time of day that students interact with teaching material could help identify students who may be struggling with their mental health.

A limitation to the results presented is that—as action research and given ethical concerns about any students suffering from a disadvantage—there was no control group to compare the two cohorts against. The discussion in the paper is based on the qualitative and quantitative data that were collected, and the empirical evidence that shows apparent benefits to this approach over more traditional programming teaching and assessment. This does point towards future work, looking at both the analysis of the data that was—and continues to be—gathered, as well as looking for more direct evidence that student learning was improved.

Author Contributions: Conceptualisation, S.G.; methodology, S.G.; software, S.G.; validation, S.G and N.A.G.; investigation, S.G.; resources, S.G.; data curation, S.G.; writing—original draft preparation, S.G.; writing—review and editing, N.A.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: The study was conducted in accordance with the University of Hull ethical research policy, approved by the Faculty of Science and Engineering Ethics Committee, approval code: FEC_2020_120, approval date: 7 July 2020.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: Data are available by contacting the authors.

Acknowledgments: The authors thank the referees for their helpful suggestions which we have incorporated into the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bikse, V.; Grinevica, L.; Rivza, B.; Rivza, P. Consequences and Challenges of the Fourth Industrial Revolution and the Impact on the Development of Employability Skills. *Sustainability* **2022**, *14*, 6970. [CrossRef]
2. Bennedsen, J.; Caspersen, M.E. Failure rates in introductory programming: 12 years later. *ACM Inroads* **2019**, *10*, 30–36. [CrossRef]
3. Cheah, C.-S. Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemp. Educ. Technol.* **2020**, *12*, ep272. [CrossRef]
4. Arakawa, K.; Hao, Q.; Greer, T.; Ding, L.; Hundhausen, C.D.; Peterson, A. In situ identification of student self-regulated learning struggles in programming assignments. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, Virtual, 13–20 March 2021; ACM: New York, NY, USA, 2021.
5. Venter, M. Gamification in STEM programming courses: State of the art. In Proceedings of the 2020 IEEE Global Engineering Education Conference (EDUCON), Porto, Portugal, 27–30 April 2020.
6. Barrón-Estrada, M.L.; Zatarain-Cabada, R.; Lindor-Valdez, M. CodeTraining: An authoring tool for a gamified programming learning environment. In *Advances in Soft Computing: 15th Mexican International Conference on Artificial Intelligence, MICAI 2016, Cancún, Mexico, 23–28 October 2016, Proceedings, Part II 15*; Springer International Publishing: Berlin/Heidelberg, Germany, 2017.

7. Arawjo, I.; Wang, C.Y.; Myers, A.C.; Andersen, E.; Guimbretière, F. Teaching programming with gamified semantics. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, 6–11 May 2017; ACM: New York, NY, USA, 2017.
8. Gordon, N.A. Enabling personalised learning through formative and summative assessment. In *Technology-Supported Environments for Personalized Learning: Methods and Case Studies*; IGI Global: Hershey, PA, USA, 2010; pp. 268–284.
9. Dolin, J.; Black, P.; Harlen, W.; Tiberghien, A. Exploring relations between formative and summative assessment. In *Transforming Assessment*; Springer: Cham, Switzerland, 2018; pp. 53–80.
10. Houston, D.; Thompson, J.N. Blending Formative and Summative Assessment in a Capstone Subject: 'It's not your tools, it's how you use them'. *J. Univ. Teach. Learn. Pract.* **2017**, *14*, 2. [CrossRef]
11. Woodfield, R. Undergraduate retention and attainment across the disciplines. *High. Educ. Acad.* **2014**, *27*, 1–77.
12. Gordon, N. *Issues in Retention and Attainment in Computer Science*; Higher Education Academy: York, UK, 2016.
13. Nolan, K.; Aidan, M.; Susan, B. Facilitating student learning in Computer Science: Large class sizes and interventions. In Proceedings of the International Conference on Engaging Pedagogy, Dublin, Ireland, 3–4 December 2015.
14. Dehnadi, S.; Richard, B. *The Camel Has Two Humps (Working Title)*; Middlesex University: London, UK, 2006; pp. 1–21.
15. Stephenson, C.; Miller, A.D.; Alvarado, C.; Barker, L.; Barr, V.; Camp, T.; Frieze, C.; Lewis, C.; Mindell, E.C.; Limbird, L.; et al. *Retention in Computer Science Undergraduate Programs in the Us: Data Challenges and Promising Interventions*; ACM: New York, NY, USA, 2018.
16. Deterding, S.; Dixon, D.; Khaled, R.; Nacke, L. From game design elements to gamefulness: Defining "gamification". In Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, Tampere, Finland, 28–30 September 2011; ACM: New York, NY, USA, 2011.
17. Sheldon, L. *The Multiplayer Classroom: Designing Coursework as a Game*; CRC Press: Boca Raton, FL, USA, 2020.
18. Gordon, N.; Brayshaw, M.; Grey, S. Maximising gain for minimal pain: Utilising natural game mechanics. *Innov. Teach. Learn. Inf. Comput. Sci.* **2013**, *12*, 27–38. [CrossRef]
19. Reeve, J.; Cheon, S.H. Autonomy-supportive teaching: Its malleability, benefits, and potential to improve educational practice. *Educ. Psychol.* **2021**, *56*, 54–77. [CrossRef]
20. Csikszentmihalyi, M.; Mihaly, C. *Flow: The Psychology of Optimal Experience*; Harper & Row: New York, NY, USA, 1990; Volume 1990.
21. Zichermann, G.; Christopher, C. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2011.
22. Gordon, N.; Brayshaw, M.; Grey, S. A Flexible Approach to Introductory Programming: Engaging and motivating students. In Proceedings of the 3rd Conference on Computing Education Practice, Durham, UK, 9 January 2009; ACM: New York, NY, USA, 2009; pp. 1–4.
23. Skinner, B.F. Cognitive science and behaviourism. *Br. J. Psychol.* **1985**, *76*, 291–301. [CrossRef]
24. Rosenstein, A.; Raghu, A.; Porter, L. Identifying the prevalence of the impostor phenomenon among computer science students. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education, Portland, OR, USA, 11–14 March 2020; ACM: New York, NY, USA.
25. Klinger, C.; Neil, M. Tensions in higher education: Widening participation, student diversity and the challenge of academic language/literacy. *Widening Particip. Lifelong Learn.* **2012**, *14*, 27–44. [CrossRef]
26. Romero, C.; Sebastian, V. Guest editorial: Special issue on early prediction and supporting of learning performance. *IEEE Trans. Learn. Technol.* **2019**, *12*, 145–147. [CrossRef]
27. Gordon, N.; Grey, S. Motivating and engaging students through technology. In *Student Engagement*; Nova Science Publishers: Hauppauge, NY, USA, 2015; pp. 25–43.