Using an Instant Visual and Text Based Feedback Tool to Teach Path Finding Algorithms: A Concept

Bhaveet Nagaria, Benjamin C Evans, Ashley Mann, Mahir Arzoky
Department Of Computer Science
Brunel University London
Uxbridge, UB8 3PH
(firstname).(lastname)@brunel.ac.uk

***Abstract*—Methods of teaching path finding algorithms, based purely on programming, provide an additional challenge to students. Indeed many courses use graphs and other visualisations to aid students in grasping concepts quickly. Globally we are rapidly altering our teaching tools to suit the current blended or remote learning style due to the global COVID-19 pandemic. We propose a method that provides instant feedback showing how their programmed path finding algorithm works based upon games. The tool will provide feedback to the student about their code quality. Along with an element of gamification we aim to improve both initial understanding and further exploration into the algorithms taught. This tool aims to provide useful feedback to students in the absence of immediate laboratory support and gives students the flexibility to conduct laboratory worksheets outside of scheduled laboratory slots.**

***Position:* Software tools and teaching assistants heavily assist undergraduate students in learning how to program. In developing enhanced software tools, we can provide immediate feedback to learners. Thus, allowing them to gain an initial understanding of the algorithm before facilitated sessions. This further enriches their experience and learning during contact hours with teaching assistants.**

***Index Terms*—teaching, algorithms, gaming**

I. Introduction

Teaching algorithms effectively proves to be a tricky task for those involved in both teaching and learning. Currently students gain support in learning through formal activities such as lectures, lab sessions and discussion boards. Students gain informal support by means of using online resources including but not limited to StackOverFlow[1], YouTube[2] and Massive Open Online Courses (MOOCs). In addition to this they are able to discuss issues with their peers. Over the years we have seen an evolution of the traditional classroom/lab sessions to include blended learning [1]–[3] and flipped classrooms [4]–[6]. With issues in running laboratory sessions effectively due to the global COVID-19 pandemic students are not able to gain the same learning experience. In this paper we propose a tool to provide immediate visual and text based feedback to students irrespective of where they are studying. The visual feedback elements are visualisation of the algorithms steps and of the generated output path. The text based feedback provides suggestions to students on their code.

At Brunel University London (BUL) undergraduate students are currently taught a module called 'Algorithms and their Applications' in the second year of their Computer Science degree. This module teaches students about data abstractions and algorithms including sorting and routing algorithms. The module employs the use of lectures and laboratory sessions as the means of teaching. Historically the module was assessed using laboratory worksheets in a VIVA form alongside an examination. In recent years, the assessment was replaced with CodeRunner[3]. CodeRunner is a Moodle[4] plug-in that allows tutors to set questions where the students answers take the form of programming code [7]. Using our proposed tool, students will be able to develop, test and submit their program code using an integrated development environment. Students will then be able to receive immediate feedback on their code and resubmit/correct their code under a specific penalty regime. In 2018/2019 we had 191 students and in 2019/2020 we had 279 students enrolled on the course.

We describe a tool that provides students with instantaneous visual and text based feedback while employing some gamification techniques. Students will be able to implement path-finding algorithms and test these against real-world gaming scenarios. Grid based games such as Atari's Pacman offer a platform which provides students with an opportunity to make examples of algorithms in game-like situations. The Ghosts in Pacman as an example use various forms of path-finding algorithms to simulate different behaviours. Path-finding algorithms such as A* [8] and Djikstra's [9] are popular algorithms for explaining and visualizing heuristics to students. Enabling visual feedback of student-written algorithms aims to increase student interest and satisfaction in Computer Science. We aim to assess the tool using a combination of qualitative and quantitative methods by exploring how well they perform in the module, the quality of their code, and their opinions of the tool itself.

The remainder of this paper is organised as follows: Section II outlines the background literature. Section III describes the concept and details our research questions. Section IV discusses how we will evaluate the concept. Section V provides our concluding remarks and directions for future work.

[1]https://stackoverflow.com/
[2]https://www.youtube.com/
[3]https://coderunner.org.nz/
[4]Moodle is an open source learning platform - https://moodle.org/

II. Background

Teaching students is a practice that has evolved over the years. There are many lessons learned and innovations within the landscape. In Section II-A we explore teaching styles, Section II-B explores gamification and Section II-C algorithm visualisation tools.

A. Teaching Styles

The standard teaching style within the Department of Computer Science at BUL typically revolves around lectures followed by laboratory sessions where the student can practice the module's theoretical aspects (taught in lectures). For example, the Algorithms and their Applications module, discussed previously, puts the information learnt in lectures into practise using programming exercises. It uses interactive development environments such as CodeRunner to allow the students to write and test code in an online environment and receive grades based on the code's performance and functionality. CodeRunner is a popular tool being used by several universities when it comes to teaching and assessing students programming ability [7], [10], [11]. Other teaching styles in higher education include active, blended and flipped learning.

The active learning model establishes engagement between the student and the teacher through discussion and activities. The objective of the active learning model is to emphasise higher-level cognitive thought in smaller groups. A review of active learning in 2004 by Prince [12] showed mixed results compared to traditional teaching styles before the teaching method's popularity increased. A 2014 literature study by Freeman et al. [13] conducted on 225 studies examined a notable reduction in failure rate compared to traditional classroom teaching.

The blended learning model combines face-to-face communication with an emphasis on instructional education through an electronic medium. With recent developments associated with the rise of the COVID-19 pandemic, the staff at BUL have adapted to more blended learning based teaching by providing lectures and practical programming sessions through virtual platforms. The concept of using blended learning in computer science has grown in interest recently, supported by publications such as Hadjerrouit [1], Hoic-Bozic et al [2] and Alonao et al. [3].

The flipped classroom model's premise is to provide students access to digital learning materials outside of classrooms to enable students to use in-classroom time for practical and active learning styles [4]. Research by Maher et al. [6] studying the effects of blended learning in computer science found high ratings from the feedback of 213 students. A systematic literature review on 32 flipped classroom studies by Giannakos et al. [5] found high engagement and student satisfaction.

Each of the proposed teaching styles has its advantages regarding student satisfaction and engagement. The increased collaboration proposed by active, blended and flipped learning has shown facilitation for more high-level discussion. Despite this, the aspect of independence and the vast difference from traditional teaching styles, in general, has shown decreased attendance. Furthermore, the cost and time required for complex course changes can be disadvantageous.

B. Gamification

Educational platforms can entice and engage student satisfaction through game-based elements such as points, rewards, and leaderboards known as gamification. Alhammad and Moreno [14] report that these gamification strategies are among some of the most studied concepts. MOOCs provide platforms for learning skills such as programming. Khan Academy[5], Codecademy[6], and HackerRank[7] are such examples. Learning materials produced by MOOCs focus on the core properties of modern programming components such as variables, loops, and methods. MOOCs provide incentives to users in the form of badges to display their skills on account profiles or possibly curriculum vitae in some instances.

C. Algorithm Visualisation (AV) Tools

Algorithm Visualisation (AV) Tools are not new; in fact they have been around for a number of years. We can find examples of these tools both within academic literature e.g. DAVE, an AV tool [15], GreedyEx, a greedy algorithm AV tool [16], GAVEL, a genetic algorithm AV tool [17] and publicly available and free to use e.g. VisualGo[8], Xueqiao Xu's visualier[9] and Clement Mihailescu's Path finding visualiser[10].

Grissom et al. [18] report that AV aided students in learning material. Velazquez-Iturbide et al. [16] develop an active learning AV tool which focuses on the learning of greedy algorithms. Vrachnos and Jimoyiannis [15] present an AV tool, DAVE. DAVE is a web-based tool designed to support secondary education students' learning about basic algorithms. The use of DAVE promoted student engagement and helped students overcome learning barriers, such as understanding sorting algorithms. Hart and Ross [17] report that their tools allow for easy visualisation of important features of the evolution and allows for visual and graphical analysis of the performance.

AV and gamification can be applied to student education to provide the student with a combination of visual and text-based feedback. Popular 1980's arcade games such as the Atari Pacman offer an engaging and visually pleasing environment with potential applications for students education and experimentation. The grid-like visuals of Atari games provide a suitable platform for students to learn different algorithms. Gamification elements such as challenges and leaderboards allow students to learn and optimise their solutions, providing them with valuable workforce skills. AV and gamification provide the core components of this paper's concept designed to engage students and help them learn.

[5]https://www.khanacademy.org/
[6]https://www.codecademy.com/
[7]https://www.hackerrank.com/
[8]https://visualgo.net/en/sssp
[9]https://qiao.github.io/PathFinding.js/visual/
[10]https://clementmihailescu.github.io/Pathfinding-Visualizer/

III. Concept

In the following section, we aim to outline our proposed method for advancing the teaching of path finding algorithms to second-year undergraduate computer science students.

Current teaching methods consist of lectures covering content, laboratories and self study covering reading materials and further practicing concepts by implementing the algorithms taught. Currently, the feedback from student programming is that of errors from the compiler or debug logs placed in the code, along with the assistance of a Graduate Teaching Assistant (GTA).

We propose a new method where the student develops their implementations of various path finding algorithms on a platform that provides instant visual feedback illustrating how the students' algorithms are assessing the paths and provide a comparison with the target algorithm the student is trying to write.

The concept employs two main forms of feedback. The primary method is in the form of visual feedback. We envision a cell-like grid where each cell illuminates as inspected from the students' implementation. Shown alongside the students' visual feedback is that of the correct algorithm they're attempting to implement, allowing the student to identify differences in the procedure more descriptively than that of console logs and compiler errors in the current form of teaching. Refer to Figure 1 illustrating a draft wireframe.

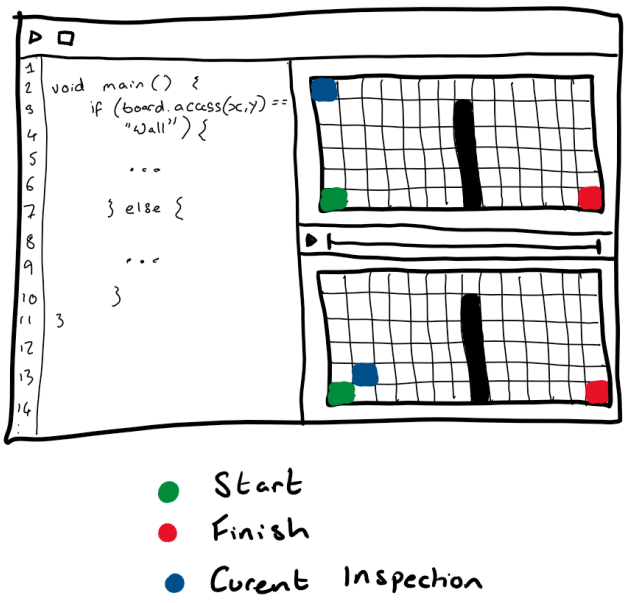

Fig. 1. Draft Wireframe of Visual Interface

To test that the students' implementation is correct, we've identified two methods to explore. The first is we generate all permutations that the current algorithm may take when checking if cells are traversable (no obstacles) and check if the students' algorithm matches one of the known logs. The second method is to identify which cells could/wouldn't be accessed when using a particular algorithm along with the minimum and maximum potential checks for the algorithm and ensuring that the students' code matches.

The secondary method of feedback is in the form of text based feedback. To encourage the use of "clean code" [19], each time the student's code runs, a static code analysis tool checks for code complexity and comments, in turn providing suggestions back. These suggestions being along the lines of "Your code is looking complex, how about splitting the functionality into named functions?". Traditionally this feedback has been given by a GTA inspecting the students' work, though, with the recent increase in demand and complexity due to necessity of remote teaching an automated method should assist in providing faster feedback to students.

Various software metrics [20] will power automated feedback. Including, but not limited to cyclomatic [21] and cognitive [22] complexity to provide feedback on the students' code complexity, and check on variable name consistency to provide feedback on documentation. We will evaluate other metrics in due course to enhance the feedback that is being provided.

Atop the basic implementations, we propose the availability of Atari-like games such as Pacman through the platform where students can build upon their base path finding algorithms to complete the games and their scores ranked on a leaderboard. We hope that this will inspire students further exploration through the use of gamification and by working with a "real world" case, have a better understanding of how and where path finding algorithms may be in use.

We believe this concept differs from pre-existing alternatives such as current offerings from the available MOOCs platforms as it provides an improved view into how a student's algorithm executes in a visual setting. Also including gamification provides a more open environment to explore working with the algorithms beyond theoretical and non-applied tasking.

Researching this concept, we aim to shine further light on the following research questions:

1) Does gamification improve students ability to develop path finding algorithms?
2) Does the instant visual feedback improve understanding during learning?
3) What are student perceptions of real time feedback on code quality?

IV. Evaluation Methods

In this section we detail our desired evaluation methods for the proposed concept. We intend to use an experimental study so we can compare the results from the current teaching style versus the proposed concept. We will employ a combination of qualitative and quantitative methods to gather a diverse range of data from our participants which will allow us to accurately understand whether the proposed concept is more effective than the current methods of teaching routing algorithms to undergraduate computer science students.

We propose an experimental study which will last for two academic years in which we will compare students learning

of path finding algorithms using the current approach (in person lab sheets) and our proposed approach (visual and text based feedback). We will recruit our students from the level 2 undergraduate module, this will mean two rounds of recruitment, one at the beginning of each academic year. In the first year of the study we will run the module as normal using lectures and lab facilities which will be supported by members of academic staff and GTA. In the second year of the study the lab facilities will be adapted to employ use of the visual tool, access to the tool will be made 24/7 so that students can practice with the tool outside of lab worksheets. This is to keep it consistent with current means i.e. a student can work on and or repeat / revisit a lab worksheet outside of the allocated laboratory sessions.

We intend on using the following quantitative measures to aid in the evaluation of the tool; code snapshots and module / lab worksheet results. We will create a snapshot of a students code each time they submit, this will allow us to understand whether the students ability to develop algorithms have increased due to the tool (RQ1). We will be able to infer a students ability to develop algorithms (RQ1) and whether visual feedback aids to improve students understanding from the grades they attain in each lab sheet and from the module as a whole.

We intend on using the following qualitative measures to aid in the evaluation of the tool; student perceptions and insights. These perceptions and insights will be gained through the use of survey instruments [23]. All participants will be requested to complete online questionnaires at set intervals throughout the academic year. We will invite a randomised subset of each population to participate in a semi structured focus group [24], to gain a richer and more detailed insight into student experiences of using the tool. The insights we uncover during the online questionnaires and focus groups will allow us to understand student perceptions of real time feedback on code quality (RQ3).

We are aware that we may have some outliers within our sample datasets. To help us better understand potential anomalous cases we will request basic demographic information from our students e.g. about their prior programming experiences both within and outside of the degree program. We will provide them with a feature to report on positive, neutral and negative experiences during usage of the tool. Finally we will record basic usage analytics e.g. how many hours was the tool used for and over what time frame. We hope that this data will allow us to understand our participants well enough to explore and explain any outliers.

V. Conclusion

In this paper we have highlighted the need for an interactive teaching tool which can support students who learn through the visual learning style at any time irrespective of the academic teaching support at the time. Additionally, we highlighted the need for a tool which allows students to learn effectively in the blended and remote learning settings. We propose a tool in which students can submit their path finding algorithms and gain feedback visually. The students will be able to see the results of their code on real world gaming scenarios such as the ghosts in the Atari Pacman game. The students will gain feedback on their code quality based upon static code metrics which should further boost their development skills and aid distance students who may conduct lab sheets out of taught sessions.

This tool will be evaluated using a combination of qualitative and quantitative methods. Current work sees us implementing the tool and evaluating it using level 2 undergraduate students. Below we explore some future work to be conducted once the tool is developed.

- **Expand on the tools reach** Once this concept tool has been developed and tested, we look to expand on how the tool can be used in three main ways; (a) provide support for other algorithms e.g. sorting algorithm, (b) provide support for other programming concepts which could support other courses e.g. threading within network computing and (c) provide support from students on all levels i.e. levels 1 - 3 of a Computer Science undergraduate degree in the UK.
- **Generalisability** Wagner et al. [25] suggest a sample size of 400 participants to attain generalisability when looking at practising software developers. Many courses at BUL do not have enough students enrolled to meet this sample size. Once we have expanded on the tools reach we will be able to offer the teaching tool on other courses e.g. level 1 and 2 programming courses, network computing, etc. This will increase our sample pool aiding us in reaching a suitable sample size. We are considering making the tool available so that students from other universities can use it and we will be able to evaluate how other students find the experience against our students at BUL.
- **Accessibility Concerns** Work is required to ensure that this learning tool is fully accessible to all users. Given that the tool predominantly focuses on visual aids, we need to consider how best to support students who are blind or partially sighted. Mealin and Murphy-Hill [26] report that some developers use multi line braille displays, however, these are typically single line. A potential solution would be to use multi line braille displays to mimic the visualisation sighted students take for granted.

References

[1] S. Hadjerrouit *et al.*, "Towards a blended learning model for teaching and learning computer programming: A case study," *Informatics in Education-An International Journal*, vol. 7, no. 2, pp. 181–210, 2008.

[2] N. Hoic-Bozic, V. Mornar, and I. Boticki, "A blended learning approach to course design and implementation," *IEEE transactions on education*, vol. 52, no. 1, pp. 19–30, 2008.

[3] F. Alonso, D. Manrique, L. Martínez, and J. M. Viñes, "How blended learning reduces underachievement in higher education: An experience in teaching computer sciences," *IEEE Transactions on Education*, vol. 54, no. 3, pp. 471–478, 2010.

[4] L. Gren, "A flipped classroom approach to teaching empirical software engineering," *IEEE Transactions on Education*, 2020.

[5] M. N. Giannakos, J. Krogstie, and N. Chrisochoides, "Reviewing the flipped classroom research: reflections for computer science education," in *Proceedings of the computer science education research conference*, 2014, pp. 23–29.

[6] M. L. Maher, C. Latulipe, H. Lipford, and A. Rorrer, "Flipped classroom strategies for cs education," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015, pp. 218–223.

[7] R. Lobb and J. Harlow, "Coderunner: A tool for assessing computer programming skills," *ACM Inroads*, vol. 7, no. 1, pp. 47–51, 2016.

[8] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec 1959. [Online]. Available: https://doi.org/10.1007/BF01386390

[10] D. Croft and M. England, "Computing with coderunner at coventry university: Automated summative assessment of python and c++ code," in *Proceedings of the 4th Conference on Computing Education Practice 2020*, 2020, pp. 1–4.

[11] P. Piwek, M. Wermelinger, R. Laney, and R. Walker, "Learning to program: from problems to code," in *Proceedings of the 3rd Conference on Computing Education Practice*, 2019, pp. 1–4.

[12] M. Prince, "Does active learning work? a review of the research," *Journal of engineering education*, vol. 93, no. 3, pp. 223–231, 2004.

[13] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth, "Active learning increases student performance in science, engineering, and mathematics," *Proceedings of the National Academy of Sciences*, vol. 111, no. 23, pp. 8410–8415, 2014.

[14] M. M. Alhammad and A. M. Moreno, "Gamification in software engineering education: A systematic mapping," *Journal of Systems and Software*, vol. 141, pp. 131–150, 2018.

[15] E. Vrachnos and A. Jimoyiannis, "Design and evaluation of a web-based dynamic algorithm visualization environment for novices," *Procedia Computer Science*, vol. 27, pp. 229–239, 2014.

[16] J. A. Velazquez-Iturbide, O. Debdi, N. Esteban-Sanchez, and C. Pizarro, "Greedex: A visualization tool for experimentation and discovery learning of greedy algorithms," *IEEE Transactions on Learning Technologies*, vol. 6, no. 2, pp. 130–143, 2013.

[17] E. Hart and P. Ross, "Gavel-a new tool for genetic algorithm visualization," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 335–348, 2001.

[18] S. Grissom, M. F. McNally, and T. Naps, "Algorithm visualization in cs education: comparing levels of student engagement," in *Proceedings of the 2003 ACM symposium on Software visualization*, 2003, pp. 87–94.

[19] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. USA: Prentice Hall PTR, 2008.

[20] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC press, 2014.

[21] C. Jones, "Software metrics: good, bad and missing," *Computer*, vol. 27, no. 9, pp. 98–100, 1994.

[22] M. M. Barón, M. Wyrich, and S. Wagner, "An empirical validation of cognitive complexity as a measure of source code understandability," *arXiv preprint arXiv:2007.12520*, 2020.

[23] C. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[24] C. Puchta and J. Potter, *Focus group practice*. London;Thousand Oaks;: SAGE, 2004.

[25] S. Wagner, D. Mendez, M. Felderer, D. Graziotin, and M. Kalinowski, "Challenges in survey research," in *Contemporary Empirical Methods in Software Engineering*. Springer, 2020, pp. 93–125.

[26] S. Mealin and E. Murphy-Hill, "An exploratory study of blind software developers," in *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2012, pp. 71–74.

Bhaveet Nagaria is working toward his PhD degree with Brunel University London. He is an hourly paid lecturer on group project, introductory programming and software engineering modules. His research interests include human factors, software engineering and software engineering education. Contact him at Bhaveet.Nagaria@brunel.ac.uk; http://www.brunel.ac.uk/bhaveet-nagaria

Benjamin C Evans is working toward his PhD degree with Brunel University London. He is an hourly paid lecturer on group project and graduate teaching assistant in artificial intelligence and high performance computing. His research interests include machine learning, ecological informatics and software engineering education. Contact him at Benjamin.Evans@brunel.ac.uk; https://www.brunel.ac.uk/people/benjamin-evans

Ashley Mann is working toward his PhD degree with Brunel University London. He is a teaching assistant in introductory programming, algorithms and their applications and artificial intelligence. His research interests include iterated local search, search based software engineering and software engineering education. Contact him at Ashley.Mann@brunel.ac.uk; https://www.brunel.ac.uk/people/ashley-mann

Mahir Arzoky is a lecturer at Brunel University London. He lectures in introductory programming, algorithms and their applications and assist on various other modules. His research interests include intelligent data analysis, software engineering, search based software engineering and software engineering education. Contact him at Mahir.Arzoky@brunel.ac.uk; https://www.brunel.ac.uk/people/mahir-arzoky