To appear in the Proceedings of the 9th International Conference on Applied Human Factors and Ergonomics, July 21-25, 2018

Name Tags and Pipes: Assessing the Role of Metaphors in Students' Early Exposure to Computer Programming using Emoticoding

Angelos Barmpoutis[1] and Kim Huynh[1]

[1] University of Florida, Digital Worlds Institute, Gainesville, Florida 32611, United States of America
angelos@digitalworlds.ufl.edu, huynhtina123@ufl.edu

Abstract. This paper presents a case study for assessing the effect of emoticoding during the students' first encounter with text-based coding interfaces, in which period a student could have a deeply disappointing experience that may lead to "blank page trauma" as well as negative attitude towards the subject. A prototype metaphor-based source code editor was developed using novel human-computer interaction mechanics based on the concept of emoticon-like scripting. Similarly to the use of shortcuts for typing emoticons in social media, visual or textual replacements appear in the proposed text editor when the user types complete valid tokens from a given programming language. Appropriate metaphors can be used in the design of the token replacements so that they are appealing to a particular age, gender, or cultural groups of users. Quantitative analysis of data from 5th-grade students (n=40) shows that metaphor-based emoticoding improves significantly the students' performance in terms of syntax recall when they transition from block- to text-based programming in comparison to transitioning without emoticoding.

Keywords: Computer Science Education · Computer Programming · Source Code Editors · STEM Education · Emoticoding.

1 Introduction

It is known in computer science education that learning to program can be a challenging task for beginners of all ages [1] and may lead to "blank page trauma" [2]. Among the several methods and teaching instruments that have been developed to address this issue, the three most notable types of tools for learning computer programming are Tangible User Interfaces (TUI), Graphical User Interfaces (GUI, also known as visual or block-based coding environments), and Text Editing Interfaces (TEI) for source-code editing. TUIs are experiential tools for early computer education (Fig. 1) that build connections with real world skills and experiences [3,4]. GUIs are the virtual equivalents of TUIs and can improve students' understanding especially in middle learning stages, as they have greater flexibility in content (Alice [5], Scratch [5,6], Tynker [7], iVProg [8], Greenfoot [5], etc.).

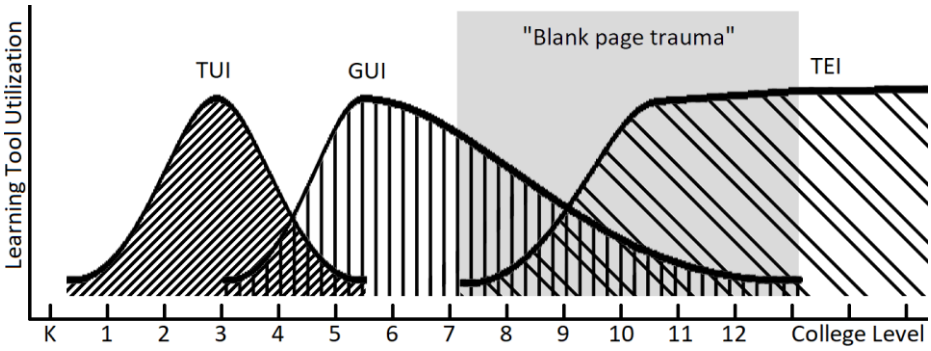

Fig. 1. Illustration of the utilization of learning tools across school levels. The most probable age for experiencing "blank page trauma" is when students transition to TEI (in gray box).

Contrary to the "toy"-looking interface of TUI and GUI tools, TEIs offer a text-editing interface that resembles professional integrated development environments [9,10]. Several studies indicate that the key advantage of TEIs is that they do not underestimate the detail and complexity of more comprehensive programming languages [11,12], while TUIs and GUIs have limited utility in teaching advanced concepts [6]. Furthermore, students learning to program in GUIs may show certain habits that are contrary to the basic programming recommendations [13].

However, transitioning to TEIs can be a challenging task for a beginner [1,2], since they provide a significantly larger number of degrees of freedom to the students compared to GUIs. A simple deviation from the syntactic rules produces compilation errors, which may not be easy to be detected and resolved by beginners. Such an early disappointing and frustrating experience caused by repetitive errors may lead to negative attitude towards the subject. A deeply disappointing experience is one of the many different factors that can cause emotional and psychological trauma, which can affect sustained and focused attention, though selective attention, which is used when processing sensory memories into short-term memories, is not affected [14,15].

In order to bridge the gap between GUIs and TEIs, a novel method is presented and assessed in this paper, which is a hybrid method that combines the visual effectiveness of GUIs with the coding complexity of TEIs using a real programming language. More specifically, it maintains the detail and capabilities of professional programming languages and at the same time adds a human-readable layer on top of the typed code using age-appropriate visual or textual replacements. Our research is based on the hypothesis that a smooth transition between GUIs and TEIs can improve students' success during their first encounter with TEI environments, and subsequently increase the probability of developing a positive attitude towards the subject. The hypothesis was tested on 40 5th-grade students who used the proposed TEI environment, which is a text-editor with emoticon-like text replacement capabilities that allows a novel coding interaction dubbed "Emoticoding" [16].

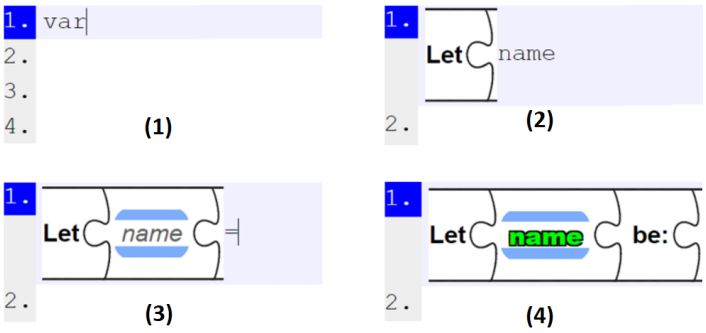

Fig. 2. The prototype editor in four instances during the typing of the JavaScript code: var name=... 1) During the typing of a new token, the typed code appears as in a conventional editor. 2) As soon as a complete valid token is typed, it is interactively being replaced by a visual metaphor. 3) The user can continue typing more tokens or erase existing ones in an intuitive emoticon-like typing. 4) Example of three tokens in the proposed text editor with replacements designed for K-12 students using metaphors such as puzzle pieces, name tags, and others.

2 Method

The use of metaphors in education is a common and well-established practice that allows students to build strong associations between a learning topic and familiar concepts or experiences from a properly chosen metaphor. In computer science education, metaphors have been used in the majority of the aforementioned learning methods. TUIs are based on direct associations between computer programming concepts and tangible hands-on experiences from familiar gaming interfaces such as building blocks or jigsaw puzzles. Similar metaphors are used in GUIs that offer virtual interfaces for connecting components (blocks) such as puzzle-like connections or directed wires that connect the output of a block with the input of another one.

However, metaphors have not been significantly utilized within text editing environments, which creates a significant disconnect between GUIs and TEIs despite their adjacency as steps in the learning path of computer programming. In this paper we propose the use of metaphors such as jigsaw puzzle pieces as part of a mechanism that interactively replaces individual programming tokens typed in the text editor using the framework of emoticoding [16] (see Fig. 2).

The method presented in this section is based on the theory of brain-activating text replacements that was introduced in [17] and has been shown to improve the performance of college-level beginner programmers in terms of syntax recall and logic comprehension. For the purposes of the present study, a prototype source-code editor was developed that has novel human-computer interaction elements that deviate from the conventional text editing interfaces. More specifically, the re-designed interface of the text editor has the following features:

Block Sequence. The body of the text consists of rectangular areas that are arranged sequentially along each line of the typed text.

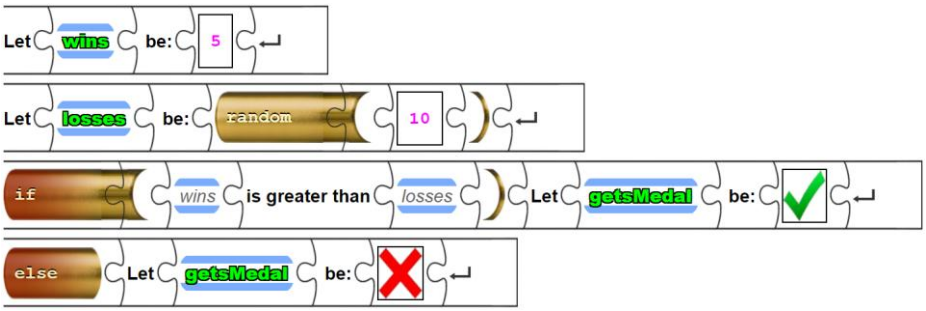

Fig. 3. An example of a 4-line program in JavaScript visualized using the metaphors designed for this study. Each token was composed by typing a valid token in JavaScript syntax.

Dual Mode Interaction. Each block can be either in edit mode or rigid mode. During edit mode the user can type text, while in rigid mode the block is decorated with a graphic or text that replaces the user-typed input. At most one block is in edit mode at a given time.

Active Tokenization. As the user types computer code, an active tokenization algorithm splits the text into individual programming tokens, which turn into blocks that are decorated with a corresponding graphic or text description (Figs. 2 and 3). Because of the resemblance of this interaction with the use of shortcuts for typing emoticons in social media, we name this coding process "emoticoding".

Compatibility with Conventional Text Editing. The user can still perform all standard text editing tasks, such as erasing blocks with backspace, selecting, copying, pasting blocks and their underlying computer code, etc.

Group Representation. Finally, several consecutive blocks can be replaced by a higher-level block if they represent a self-contained programming entity that was identified through active tokenization. For example all the blocks contained within "/*" and "*/" (comment delimiters) can be grouped together by one larger block that replaces the entire typed comment. Examples of other self-contained elements are functions, classes, etc.

The text replacements that appear when a block is in rigid mode can be designed for specific user profiles based on their age, gender, language/ethnicity, etc. Figures 2 and 3 show various examples of replacements that were designed for K-12 students using various metaphors such as puzzle pieces for separating individual tokens, name tags for variable names, and English words/phrases such as "Let" and "be:" for the tokens "var" and "=" respectively. A complete set of more than 90 text replacements was designed for the purposed of this study.

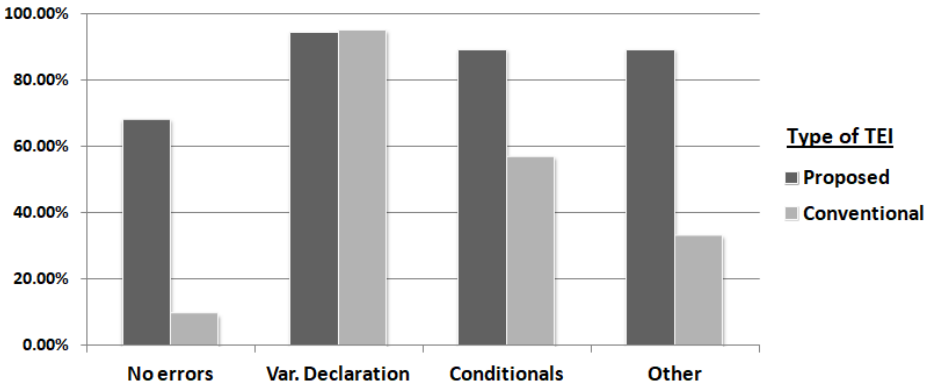

Fig. 4. The percentage of successful students for each type of TEI using the four predefined evaluation metrics.

3 Case Study

The proposed editor was implemented as a web application [16] with an active tokenizer for JavaScript and tested by 40 5th-grade students from Gainesville, Florida who volunteered to participate in this study. To enroll in this study the student should have no prior experience in coding, which was determined in the enrollment questionnaire that was filled by all the participants. In order to assess the effectiveness of the proposed metaphor-based method as a tool for smooth transition to TEI environments, we followed a curriculum that involved GUIs for early exposure of the students to computer programming, which was then followed by transition to TEIs.

More specifically, before the study the students were exposed to computer programming using a GUI (Tynker [7]) for a period of 10 weeks following a curriculum from "Hour of Code" (hourofcode.com) for 1hr/week. After the end of this training period, the students were split into two same-size groups, and each group transitioned to a different TEI environment. The students of each of these two groups were exposed to the same basic programming syntax such as variable declaration and conditionals for approximately 45 minutes. During the same time the students of each group practiced using their assigned TEI; group A used the proposed TEI with metaphor-based emoticoding; group B used a conventional TEI (source code editor) without emoticoding.

At the end of the study all students had to complete the same programming assignment in JavaScript based on the material that was covered in the study. More specifically, the students were asked to declare a few variables (such as "score" and "medals" with assigned values 5 and 0), and use a few conditional statements (such as "if the value of score is equal to 5 then set medals to 1", etc.), with complexity similar to the example shown in Fig. 3. The students had to complete the assignment with pen and paper in order to test if they could recall the syntax and logic of the programming language without the assistance of the compiler or the proposed metaphor-based token replacements.

4 Experimental Results

The students' solutions to the programming assignments were quantitatively evaluated for accuracy using four different metrics: 1) The submission had no errors (i.e. can be successfully compiled), 2) The variable declarations had no errors, 3) The conditionals had no errors, 4) There were no other errors. Note that these are four independent metrics with binary value, which were uniformly applied to all students.

The first column in Fig. 4 shows the percentage of the students who completed the assignment without errors (metric 1). According to this metric, the students who transitioned to the proposed metaphor-based TEI performed better compared to the students who used a conventional source code editor. More specifically, the use of metaphor-based emoticoding increased the success of the students by a factor of 7.5, i.e. 68\% vs. 9\%.

The remaining columns in Fig. 4 show the detailed percentages of the students who performed successfully in metrics 2, 3, and 4. Overall, the percentages indicate that the use of metaphor-based emoticoding yields equal (metric 2) or better results (metrics 3 and 4) compared to conventional TEIs. More specifically, the percentage of success was increased by a factor of 1.6 (from 57\% to 89\%) with regard to the use of conditionals, and by 2.7 (i.e. from 33\% to 89\%) with regard to other syntactical structures respectively. The latter category included all other observed syntactical errors, such as missing semicolons at the end of individual commands, unbalanced parentheses or braces, etc. Additionally, the following observations were made:

The students who transitioned to the proposed metaphor-based TEI typed more examples of variable declarations and conditionals during the allotted 45 min. of training. The responsiveness of the text editor with emoticoding increased the level of student engagement, who wanted to play more with the JavaScript source code, compared to the conventional source code editor.

Emoticoding helped students identify incorrect programming syntax by means of English syntactic errors in the phrases composed from the token replacements. For example, a correctly defined condition will read "if score is equal to 5..." contrary to the grammatically incorrect "if score be: 5..." caused by the erroneous use of the symbol "=" instead of "==", which correspond to "be:" and "is equal to" respectively.

Students who used emoticoding acquired additional knowledge from what was covered in the tutorial by simply interacting with the system. For example, the erroneous use of a space within a variable name produces two tokens (emoticon blocks) instead of one variable name (one block) which is the goal. After this observation, they corrected the error by removing the space, which indicated that they learned that spaces are not allowed within variable names.

Only in one student's submission there was confusion between the elements of JavaScript with the text-replacements of emoticoding. For example, instead of the keyword "var" the student typed "Let", which was the corresponding token replacement shown in the proposed metaphor-based emoticoding editor.

5 Discussion and Future Directions

The aforementioned findings indicate that the proposed method improves the performance of 5th-grade students, which is in agreement with the results obtained from a previous study on non-Computer Science college-level students [17]. The findings from these two studies indicate that overall emoticoding facilitates the learning of computer programming across ages, from elementary school to college, an argument that should be verified through long-term studies in the future.

A key difference between GUIs and the proposed TEI-based environment is that the former constitutes a new programming language (a visual or block language), while the latter facilitates the learning of an existing programming language. As a result, an important benefit from using the proposed method is that the student acquires a skill set that is directly applicable in many areas (game, web, app development), i.e. the knowledge of a professional programming language (in this study JavaScript).

This distinction makes it difficult, or better irrelevant to compare these two methods in terms of syntax recall (for example compare how many students know the difference between "=" and "==" after using these two methods). The different goals and objectives between GUIs and TEIs make them two distinct but equally important steps in a sequence of steps that leads beginners to coding.

In this sequence, the discontinuity between GUIs and TEIs due to their differences in complexity, degrees of freedom, and human-computer interaction can be smoothed by the use of emoticoding in TEIs. More specifically, the results from this study suggest that by introducing metaphor-based emoticoding to source code editors, the students' success during their first encounter of TEIs is significantly increased by a notable factor of 7.5. The students who transitioned from GUI to the proposed text editor demonstrated superior knowledge of programming (in JavaScript) compared to the students who transitioned from GUI to a conventional source code editor. These findings support the hypothesis expressed earlier.

Furthermore, if "blank page trauma" is related in some degree to the disappointing experience that a beginner may have, the proposed method can be reasonably considered a helpful tool for reducing this problem. Subsequently, the proposed method can increase the probability of a student developing a positive attitude towards the subject, an argument that will also be tested more thoroughly in a long-term study in the future.

Finally, various different sets of visual or textual replacements can be designed with focus on specific target populations that might benefit significantly from this tool, such as minority students, female students, and other groups. More specifically, emoticoding can be used as a culturally responsive teaching method that can appeal to the specific interests of target populations within a specific spatiotemporal context. This can be implemented by using visual replacements with metaphors from athletics, music, nature, fantasy, comics, urban life, science, and others that could be appealing to different individuals or groups of individuals. Such metaphors can increase the effectiveness of emoticoding for these populations and subsequently serve as a catalyst for gaining essential technology-oriented skills that can be applied to all STEM areas in general as it was recently shown in [18].

Acknowledgments. The authors would like to thank the students who participated in this study. Angelos Barmpoutis would also like to express his appreciation to the University of Florida College of the Arts for honoring him with the "Best Teacher of the Year" award in 2017 for inventing and applying the method discussed in this paper.

References

1. Lahtinen, E., Ala-Mutka, K., Järvinen, H. M.: A study of the difficulties of novice programmers. ACM Sigcse Bulletin 37(3), 14--18 (2005)
2. Morgado, C., Barbosa, F.:A structured approach to problem solving in CS1. In: Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education. pp. 399--399 (2012)
3. Wang, D., Zhang, L., Xu, C., Hu, H., Qi, Y.: A tangible embedded programming system to convey event-handling concept. In: Proceedings of the TEI'16, Tenth International Conference on Tangible, Embedded, and Embodied Interaction. pp. 133--140 (2016)
4. Sapounidis, T., Demetriadis, S.: Educational Robots Driven by Tangible Programming Languages: A Review on the Field. In: Int. Conference EduRobotics. pp. 205--214 (2016)
5. Utting, I., Cooper, S., Kölling, M., Maloney, J., Resnick, M.: Alice, greenfoot, and scratch–a discussion. ACM Transactions on Computing Education (TOCE) 10(4), 17--17 (2010)
6. Sivilotti, P. A., Laugel, S. A.: Scratching the surface of advanced topics in software engineering: a workshop module for middle school students. ACM SIGCSE Bulletin 40(1), 291--295 (2008)
7. García-Peñalvo, F.J., Rees, A. M., Hughes, J., et al.: A survey of resources for introducing coding into schools. In: Proceedings of the 4th International Conference on Technological Ecosystems for Enhancing Multiculturality. pp. 19--26 (2016)
8. de Oliveira Brandão, L., Bosse, Y., Gerosa, M. A.: Visual programming and automatic evaluation of exercises: An experience with a STEM course. In: Frontiers in Education Conference (FIE). pp.1--9 (2016)
9. Reas, C. and Fry, B.: Processing: A Programming Handbook for Visual Designers and Artists. MIT Press, Cambridge (2014)
10. Freeman, J., Magerko, B, Verdin, R.: EarSketch: A web-based environment for teaching introductory computer science through music remixing. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education. pp. 5--5 (2015)
11. D. J. Malan and H. Leitner. 2007. Scratch for Budding Computer Scientists. ACM SIGCSE Bulletin 39, 1 (2007), 223–227.
12. Meerbaum-Salant, O., Armoni, M., Ben-Ari. M.: Learning computer science concepts with scratch. Computer Science Education 23(3), 239--264 (2013)
13. Moreno J., Robles.G.: Automatic detection of bad programming habits in scratch: A preliminary study. In: Frontiers in Education Conference (FIE). pp. 1--4 (2014)
14. Robinson, L., Smith, M. Segal, J.: Emotional and Psychological Trauma: Healing from Trauma and Moving On. Helpguide.org (2017)
15. Jenkins, M. A., Langlais, P. J., Delis, D., Cohen, R. A.: Attentional Dysfunction Associated with Posttraumatic Stress Disorder Among Rape Survivors. The Clinical Neuropsychologist 14(1), 7--12. (2000)
16. Emoticoding, http://emoticoding.org
17. Barmpoutis, A., Huynh, K., Ariet, P., Saunders, N.: Assessing the Effectiveness of Emoticon-Like Scripting in Computer Programming. Advances in Intelligent Systems and Computting 598, 63--75 (2017)
18. Barmpoutis, A.: Integrating Algebra, Geometry, Music, 3D Art, and Technology using Emoticoding. In: Proceedings of the 8th IEEE Integrated STEM Education Conference (ISEC), 1--4 (2018)