sCool: Impact on Human-Computer Interface Improvements on Learner Experience in a Game-Based Learning Platform

Martin Sackl\(^1\), Alexander Steinmaurer\(^1\), Christopher Cheong\(^2\), France Cheong\(^2\), Justin Filippou\(^3\), and Christian Gütl\(^1\)

\(^1\) Graz University of Technology, Graz, Austria
\(^2\) RMIT University, Melbourne, Australia
\(^3\) University of Melbourne, Melbourne, Australia

Abstract. As many concepts in programming are difficult for novices to learn, it can be frustrating for them to stay motivated. For this purpose, alternative approaches like game-based learning can help increase their motivation. sCool is a serious game that supports learning computational skills through conceptual aspects and practicing programming, but is also used in revision activities as it provides an engaging way for students to review their understanding of programming. In a revised version of sCool, we focus on obtaining an improved human-computer interface (HCI) by adapting several interface aspects. Thus, our research goal is to determine the effects of the improved interface on novice student programmers’ revision of learning programming, but also how experienced student programmers perceive the interface. In a preliminary study on the revised version of sCool, 11 novice and 9 experienced student programmers played the game and completed a survey providing data to answer the defined research questions. We are able to show that the improved interface has a positive impact on novices’ revision of learning programming as well as on the game-play of sCool.

Keywords: Serious game · Mobile learning · Human-computer-interaction · Coding

1 Introduction

In modern society, the value of programming and computational thinking is more important than ever, as computational methods let us solve problems and build systems accordingly, and this is not only in the area of computer science. Computational thinking enables people to recognize problems and solve them in efficient ways. It involves algorithmic thinking, evaluation, decomposition, abstraction and generalisation \cite{3}. To be able to understand computational
thinking, there are several approaches which improve engagement in learning programming. This study deals with the perception of the improved human-computer interfaces used in a game-based learning approach, which has two purposes - to be fun and to be educational [1].

For this study, we initiated, developed and researched the serious game sCool [2], a mobile learning tool for novice student programmers to help them with the basics in the world of coding and computational thinking. Previous research [9] on sCool showed that students were motivated while playing the game, but often got distracted by interface issues. The goal for this study is to improve the human-computer interface in such a way, that the learning process is not hindered by interaction problems. In this paper, we argue the importance of a concise and well-designed human-computer interface (HCI). We investigated how the user interface affects novice student programmers’ revision of learning programming as well as how experienced student programmers experience the improved interface. Students should be able to write code in a basic way and thus be able to concentrate on specific tasks, as well as get more helpful feedback and guidance from the system.

The remainder of the paper is organized as follows. In Sect. 2, we outline the existing tools and approaches in the area of game-based learning and which HCI aspects are important in this area. We then introduce sCool in detail and specify how it is used in contrast to other learning tools. We also discuss improvements that were made in order to obtain a concise and well functioning interface. The next section covers a detailed evaluation of the game’s new version, in which we tested the game and specifically its interface on novice and experienced student programmers. This includes a description of the setup of the experiment, the evaluation instruments, procedures and participant details, as well as a discussion of the results. We then summarize our findings and discuss future work with sCool.

2 Related Work

Mobile games have become increasingly popular over the last few years and are even an influential part of the learning behaviour of students. Such games are not only useful instruments for learning specific content, but they can also leave an impression on students [4]. In general, students are more motivated while playing games than when using other learning approaches and thus they can be totally immersed in the game, because of game’s complex graphics and sounds as well as instant responses to player actions [5].

Shabalina et al. [6] introduced a role playing game, developed for learning programming languages, which is mainly used in computer science classes at technical universities. Users can be further classified into beginner students and students who want to improve their skills. The authors proposed three basic concepts, on which the game is based on: 1) learners should get the course content through their interpretation of the game world, 2) they must see results of their programming in a game context and 3) the results should influence the
game. This approach is believed to improve students’ skills in programming and increase their motivation. Another learning tool is the game iPlayCode, proposed by Zhang et al. (2014) [10], which was developed for non-programmers to learn programming in C++ in a cycle-learning mode. Great value is placed on the game’s simple interface, which they state is a core component to improve the user experience. This tool includes a reward system that awards or deducts points, depending on the player’s answers. Besides functional testing and performance testing, the game’s usability was tested, divided into learnability, efficiency, memorability, errors and satisfaction. Results showed that users were able to manage the system’s interface easily at the first use.

The serious game sCool [2] is a learning tool, which was developed as a research collaboration between Graz University of Technology (Austria) and Westminster University (UK). This educational game is designed for novice programmers to learn computational thinking and programming in Python in a playful and exploring way. Based on an engaging story, it offers different game types to learn concepts as well as practical coding exercises. Additionally, educators are able to adapt the learning content using a web interface. With concept-learning tasks, players can explore maps and learn new programming concepts, which can then be applied in a practical mode. Users write actual Python code in order to develop a program that leads a robot to the destination on a squared chessboard playground. Therefore, basic commands are given in form of buttons (Move, Print, If, Var and For), but students are free to edit code blocks as well.

Previous research projects with sCool identified issues regarding its interface and controls and thus the game did not provide the best learning experience. The practical mode of the game had the most issues. To write code, a virtual in-game keyboard was initially implemented, which had some serious issues such as: overlapping other content, confusing layout and prohibiting the use of a native keyboard. Further issues included players not being able to write the first lines of code on their own, but rather had to edit a code block’s content. The drag and drop functionality also caused confusion as players often did not know how to use these elements properly. Players also identified issues with the game menu (missing descriptions of navigation and ambiguous named buttons) as well as tutorials and hints, which many users often skipped. As the design of hints did not stand out from the editor, players tended to gloss over them [9].

In more abstract observations, while using games in educational areas, the interaction between students and the system, the human-computer interaction (HCI), is an important aspect. Therefore, simple and well designed interfaces are essential for a high learning rate. To achieve this, HCI evaluations and usability testings are a significant part of contemporary applications [8].

According to Looi et al. (2010) [7], improvements on serious games platforms in order to obtain best results from students and high success rates can be achieved by HCI evaluations. Such evaluations can be used to overcome low engagement of students or repetitive nature of tasks, and to accomplish a successful educational game, which motivates students over a longer time period. In their work they stated that players’ engagement is based on how close the inter-
action between the players and the game appears. To evaluate this, they used the video-diary method, which investigates the gamers’ perspectives. Thereafter, questions about the student’s level of boredom during the game, whether they could sustain longer time periods and whether they prefer non-learning associated activities were evaluated.

3 sCool - Interface Improvements

Based on the issues revealed from previous sCool research projects [9], there was a significant need to adapt the game’s interface so that users on any device are able to write code in a simple way and do not get distracted by other reported interface problems. These issues concerned i) code editor, ii) controls/keyboard, and iii) menu/instructions/hints.

3.1 Requirements

**Code Editor.** There is a strong need to improve the code editor. For easier use of code blocks, users should be able to write the first line of code on their own, rather than using the drag and drop functionality. In general, they should be able to do this anytime. To show players that elements like code blocks can be dragged and dropped, a visible indicator (like an icon) should be displayed next to these elements.

**Controls and Keyboard.** To achieve a well-functioning multi-platform game, the initial virtual keyboard should be replaced by the device’s native keyboard, since it showed some earlier mentioned problems. Especially in the Windows version, players were frustrated as they had to click the keyboard’s buttons to write code.

**Menu, Instructions and Hints.** Once a player pressed the “Menu” button during a robot mission, s/he would be taken back to the menu right away, without confirming the action and all the written code would be gone. Thus, the data loss should be avoided, since it is an important aspect of system feedback. Another feature in that respect is the design of hints and descriptions during the game; they should be better emphasised so that they stand out more from the editor and background. Tutorials are shown when a player enters the practical mode and can also be displayed anytime. Some hints however appear only once and are never shown to the player again. The goal therefore is to ensure that players read and understand these messages, as they tend to gloss over them.
3.2 Implementation of Interface Optimizations

One of the most significant issues identified is regarding the controls and the virtual keyboard. A simple solution therefore is to remove the virtual keyboard completely and allow the use of native keyboards for all devices. The most effective way therefore is to use input field elements instead of ordinary text components as they automatically display the device’s native keyboard when they are in focus (see Fig. 1).

In order to enable a better coding experience, several components of the code editor were improved. Since there was no feedback from the system when a player clicked predefined drag and drop buttons, an alternative method was implemented which marks the selected button and enables an “Add” button. Drag and drop functionalities inside the code editor left out an important aspect of usability design, which is a visible indicator that a user can interact with an element. Users may not know that code lines are draggable and for this purpose, such an indicator in the form of an eight dotted icon was placed inside a draggable element. Since users are only able to edit existing code blocks, an empty code line including a placeholder was added to the editor. This line is placed at the end of the code at any time, so that the player can write additional lines of code one after another by pressing the “Enter” button.

To improve hints and tutorials, a simple approach to encourage players to pay more attention to in-game tutorials was implemented. Some examples are the descriptions of the different tab views (task description, code editor and output panel), which are important. These descriptions are only shown once to a player. To get the player focused on the tutorial and give the feeling that the game is paused momentarily, we added a darkened background canvas and an animation of the tutorial pop-up. Further, all of the code a player had written could be lost as a result of a single errant click, which is a grave interaction issue. This was resolved by adding a confirmation screen after clicking the “Menu” button. This functionality was added only in cases of data loss or major changes.

![Fig. 1. Improved user interface of the code editor.](image)
4 Evaluation

Based on the improved user interface (see previous section), an evaluation was performed to answer the following research questions:

**RQ1:** What are the effects of the improved human-computer interface on novices revising learning programming?

**RQ2:** How do improvements to the in-game code editor and communications (such as hints and tutorials) affect player’s game experience?

4.1 Setting and Instruments

Participation in this study was voluntary and was conducted

1. in an in-class activity in an Australian business information systems course for novice student programmers and
2. with students from different programs in computer science at a technical university in Austria addressing those with programming skills.

The instruments used involved the revised version of sCool, an online questionnaire and the usage data (collected on a server). The novice group experienced the game in a supervised setting while the experienced group used it in an unsupervised setting.

**Novice Student Programmers.** The study on novices was initially planned in classrooms, but due to the COVID-19 global pandemic restrictions, these classes were held online. There were 5 online classes in a Python introduction course at an Australian university, with around 25 students per class. The first half was a regular class held by the tutor. In the second half, the tutor left the classroom (due to the ethics approval) and an external supervisor joined to conduct the experiment. Due to occurring connection problems, we pre-recorded an introduction video showing the use of sCool. The students then had 30–40 min to play the game and were asked to answer an online questionnaire afterwards.

**Experienced Student Programmers.** We also had to conduct the study for experienced student programmers online. Computer science students from a technical university in Austria were asked to participate. As the experiment was not conducted in regular classes, we sent the participants an information document about the study, the introduction video and game download links. The participants were able to play the game anytime they preferred, so no supervisor was present. They were told to take 40–50 min for this experiment.

**Online Questionnaire.** Regarding usability testing and analysing, we included the Computer Emotion Scale [12] (CES) and the Game Experience Questionnaire [11] (GEQ) in our survey, which was hosted on the Qualtrics survey platform.
The GEQ holds 12 adjectives associated with the four basic emotions: happiness, sadness, anxiety and anger and allows the participants to mark how often they felt an emotion while playing the game by choosing from a range from “1 - none of the time” to “4 - all of the time”. The CES includes 19 statements, which describe the engagement of a user during the game. These can be associated with immersion, presence, flow and absorption; answers can be made using a Likert scale from “1 - strongly agree” to “5 - strongly disagree”. The mean and standard deviation from the CES and GEQ results were calculated using the open source programming language R.

We also asked the participants to choose from a Likert scale on how much they agree on eight statements about usability and interface, which are shown in Table 1. Besides two open-ended questions about HCI improvements, the participants were given seven game-related statements (involving the game’s language, the difficulty of levels and the encouragement of sCool), which they are required to rate as well on a Likert scale ranging from 1 to 5.

| ID | Statement                                      |
|----|-----------------------------------------------|
| S1 | The game’s control was easy to use            |
| S2 | The code editor was easy to use               |
| S3 | Writing code was easy                         |
| S4 | I liked to use the drag and drop function for coding |
| S5 | I liked to write code on my own               |
| S6 | Error messages were clear and understandable |
| S7 | The tutorials helped me to understand what to do |
| S8 | Hints helped me to complete levels            |

**Usage Data.** While playing the game, the system stores the player’s usage data (relevant information about player, player’s performance and log data) securely on a Microsoft Azure SQL database server. To observe how well students performed in the in-game course, the database provides information about all players’ attempts, unlocked skills and solved tasks. Analysis and visualization of the usage data, as well as the overall results from the online questionnaires, was done with Microsoft Excel.

4.2 Procedure

The experiments for novices and experienced student programmers were structured similarly, with the main difference that a supervisor was only present at the novices experiment. Both groups were introduced to sCool and the research project and given time to play the game. The in-game course, which was created explicitly for this experiment over the web platform included three parts:
Each part of the game included three concept-learning tasks and three practical exercises and students had to solve all tasks in order to unlock the next skill. This covered the Python introduction course’s content of the weeks before. After students revised their programming knowledge in the concept-learning section, they had to apply it in the practical part.

In the first part of the robot missions (practical part), students had to solve basic tasks related to fundamental programming concepts (commands and sequences) in Python which introduced them to the programming environment. The second part covered tasks related to data types and variables. Players learned this concepts using the so-called robot storage which serves as an internal memory where external data from the web application can be provided into the scope of the game. Also, participants were faced with arithmetic tasks. In the last part, students used loops and conditional statements in order to complete the missions. Table 2 shows an overview of the tasks given for the robot missions.

| Skill                  | Task                                                                 |
|------------------------|----------------------------------------------------------------------|
| **Introduction**       | 1. Use basic commands to move Rob                                    |
|                        | 2. Move Rob and avoid obstacles                                       |
|                        | 3. Print a phrase to the console                                      |
| **Data types**         | 1. Access the robot storage, add a value and print it                |
|                        | 2. Print the robot storage and (optionally) collect coins            |
|                        | 3. Use the robot storage and arithmetic operators for a calculation  |
|                        | and print it                                                         |
| **Constructs and logic**| 1. Use a loop to print the list in the robot storage                  |
|                        | 2. Use conditional statements to print right content of the robot    |
|                        | storage                                                               |
|                        | 3. Compute the Fibonacci Sequence and print the index of the list in |
|                        | the robot storage, where the lists differ                            |

### 4.3 Participants

As part of the study, 49 undergraduate students used sCool. Of these students, 39 gave their consent for sCool to store their in-game data to be used in the research and 11 completed the questionnaire. We also invited 20 experienced student programmers to the study, of which 9 participated and all of them completed the questionnaire. In total, 20 students fully answered the questionnaire.
Table 3 shows the details of the participants divided into the two groups: novice and experienced students. According to this table, the main difference between the two groups are the programming experience and age of students.

| Criteria                        | Possible responses | Novice student programmers | Experienced student programmers |
|---------------------------------|--------------------|-----------------------------|--------------------------------|
| Gender                          |                    | Male                        | 8                              | 8                              |
|                                 |                    | Female                      | 3                              | 1                              |
| Age                             | 18–24              | 10                          | 0                              |
|                                 | 25–34              | 1                           | 8                              |
|                                 | 35–44              | 0                           | 1                              |
| Played an educational game before| Yes                | 0                           | 1                              |
|                                 | No                 | 11                          | 8                              |
| Programming Experience          | Non existent       | 2                           | 0                              |
|                                 | Novice             | 6                           | 1                              |
|                                 | Intermediate       | 3                           | 1                              |
|                                 | Advanced           | 0                           | 5                              |
|                                 | Expert             | 0                           | 2                              |

4.4 Results and Discussion

To be able to answer RQ1, we analyzed the collected usage data of the novice student programmers. According to this data, of the 39 novice participants, 34 (87.18%) were able to solve all concept-learning tasks and 26 (66.67%) all practical tasks of the first skill and thus, unlocked the second skill. In the second part, 18 (46.15%) passed all of the concept-learning and 6 (15.38%) the practical tasks. This means that 6 (15.38%) novice student programmers were able to unlock the last skill, in which 2 (5.13%) solved all concept-learning and 1 (2.56%) all practical tasks. In contrast to previous experiments with the former sCool interface, in which no student reached the third part, this shows a slightly improved performance. This means, in respect to RQ1, that the improved interface indeed had a positive effect on the revision of novices as it improved their performance on a small scale.

On the other side, all 9 experienced programmers were able to reach the third part and thus solved 100% of the concept-learning and practical tasks of the first two skills. Regarding the third skill, 6 (66.67%) passed all concept-learning and 5 (55.56%) passed all practical tasks.
In general, both novice and experienced student programmers showed similar results according to the Game Engagement Questionnaire and Computer Emotion Scale. Results are shown in Figs. 2 and 3. Concerning the GEQ, both groups showed a high level of immersion ($M = 3.45$, $SD = 0.82$ for novices and $M = 3.78$, $SD = 0.83$ for experienced) and presence ($M = 3.09$, $SD = 0.88$ and $M = 3.22$, $SD = 1.12$). Since immersion can be described as a state of consciousness where the player is totally engaged in the game environment and presence the regular state of consciousness (Brockmyer et al. (2009) [11]), the novice players were aware of the virtual environment, but also engaged into it. Their level of flow ($M = 2.56$, $SD = 1.14$ and $M = 2.69$, $SD = 1.10$) indicates the balance degree between challenge and skill, which leads to a good learning experience [13].

The CES results show that most of the participants had positive feelings while playing the game. This can be observed from happiness levels ($M = 2.55$, $SD = 0.91$ for novices and $M = 2.69$, $SD = 0.68$ for experienced students). Levels of sadness, anxiety and anger were almost equal for both groups, however, experienced student programmers showed a slightly higher level of anger than novices ($M = 1.40$, $SD = 0.50$ and $M = 1.58$, $SD = 0.64$). Compared to evaluations from previous versions of sCool, the level of happiness increased while the level of anger decreased, which points to a positive effect regarding RQ2.

According to answers on sCool’s interface specific statements, the novice student programmers liked the possibility to write code on their own (S5) more than using the drag and drop functionality (S4). Their opinion on the usage of the code editor for writing code (S2 and S3) was positive. Thus, we can observe that the improvements had a reasonable impact on novices revision of programming (RQ1). Rather negative answers were collected about error messages and hints in the game (S6 and S8), which means that regarding to RQ2, communications between the system and the player did not improve for novices. Experienced student programmers also tend to like writing code (S5) more than using buttons (S4) and rated the simplicity of writing code (S3) very positively. However, they also stated overall positive feelings concerning the code editor (S2). Like the

![Fig. 2. Results of the game engagement questionnaire.](image-url)
other group, they disliked error messages and hints (S6 and S8), even more than novices. We can therefore state, that the improvements had a bigger impact on novices (RQ1) than they had on experienced students (RQ2).

Regarding open-ended questions, participants mostly had concerns and improvement suggestions for specific tasks and the “planet” exploration mode (“I couldn’t move from the starting point of the map to the discs and enemies”, “Change controls (A is for shooting AND going left)”). Answers categorized as “Interface” mostly concerned indentations and error messages (“Errors are a little hard to understand sometimes”, “Make hints easier to understand”, “Make adding indentations for desktop users more intuitive”). Also several suggestions regarding tutorials and hints were given. Overall, only a few answers were provided regarding the improvements implemented in the interface.

5 Conclusion

In the revised version of sCool, our goal was to implement an improved human-computer interface by adapting several issues identified with the interface. In two similarly-structured experiments, but with different levels of participants’ programming experience, our aim was to show how these improvements affect the game-play and whether they improved novices’ revision of learning programming.

Since participation and completion of the survey was completely voluntary, the evaluation was limited to 20 participants. The results of the evaluation showed that players did not have as many problems with the interface as before, but also that the improvements had a bigger impact on novices than experienced student programmers. Compared to iPlayCode [10] many of the novice but almost all of the experienced participants managed to interact easily with the interface. Observations of the GEQ and CES showed an increased level of happiness and a lower level of anger compared to previous research studies. On the other hand, we were not able to identify improvements on communications between system and player in the form of error messages and hints, but rather (like the approach from Moreno-Ger et al. [8]) respective communication issues.
Regarding the impact on novices’ revision of learning programming, the novice student programmers overall performed slightly better with the revised version of sCool.

For future work, the evaluation results indicated further improvements of the interface in the area of communications and feedback from the system such as error messages and hints. Our goal is to improve these communications such that a broad variety of players with different levels of programming experience are provided a more self-explanatory game-play experience and receive more understandable messages from the system.

Acknowledgment. We would like to gratefully acknowledge the support of Graz University of Technology for funding and RMIT University’s School of Business IT and Logistics for hosting Martin Sackl. We would also like to thank participants for volunteering in our research project.

The research with students from the Australian university was conducted with ethics approval from the RMIT Business College Human Ethics Advisory Network under register number 22675.

References

1. Bellotti, F., Kapralos, B., Lee, K., Moreno-Ger, P., Berta, R.: Assessment in and of serious games: an overview. Adv. Hum.-Comput. Interact. (2013). https://doi.org/10.1155/2013/136864
2. Kojic, A., Kojic, M., Pirker, J., Gütl, C., Mentzelopoulos, M., Economou, D.: sCool – a mobile flexible learning environment. In: Online Proceedings from 4th Immersive Learning Research Network Conference, Missoula, Montana (2018)
3. Wing, J.M.: Computational thinking. Commun. ACM 49(3), 33–35 (2006)
4. Gros, B.: Digital games in education: the design of games-based learning environments. J. Res. Technol. Educ. 40(1), 23–38 (2007). https://doi.org/10.1080/15391523.2007.10782494
5. Beck, J.C., Wade, M.: The Kids are Alright: How the Gamer Generation is Changing the Workplace. Harvard Business Review Press (2006). 9781422166499
6. Shabalina O., Vorobkalov P., Kataev A., Tarasenko A.: Educational Games for Learning Programming Languages. Institute of Information Theories and Applications FOI ITHEA, 1313-0455 (2008). hdl.handle.net/10525/1136
7. Looi, Q.E., See, S.L.: Effectively engaging students in educational games by deploying HCI evaluation methods. In: Proceedings of the World Congress on Engineering and Computer Science, vol. 1 (2010)
8. Moreno-Ger, P., Torrente, J., Hsieh, Y.G., Lester, W.T.: Usability testing for serious games: making informed design decisions with user data. Adv. Hum.-Comput. Interact. (2012). https://doi.org/10.1155/2012/369637
9. Steinmaurer, A.: Revising a game-based learning platform for computational skills in education. Master’s thesis, Graz University of Technology, Graz, December 2019. https://doi.org/10.13140/RG.2.2.35247.76967
10. Zhang, J., Lu, J.: Using mobile serious games for learning programming. In: The proceedings of The Fourth International Conference on Advanced Communications and Computation (INFOCOMP 2014), Paris, France, pp. 20–24 (2014)
11. Brockmyer, J.H., Fox, C.M., Curtiss, K.A., McBroom, E., Burkhart, K.M., Pidruzny, J.N.: The development of the game engagement questionnaire: a measure of engagement in video game-playing. J. Exp. Soc. Psychol. 45(4), 624–634 (2009). https://doi.org/10.1016/j.jesp.2009.02.016

12. Kay, R.H., Loverock, S.: Assessing emotions related to learning new software: the computer emotion scale. Comput. Hum. Behav. 24(4), 1605–1623 (2008). https://doi.org/10.1016/j.chb.2007.06.002

13. Csikszentmihalyi, M., Abuhamdeh, S., Nakamura, J.: Flow. In: Flow and the Foundations of Positive Psychology: The Collected Works of Mihaly Csikszentmihalyi, pp. 227–238. Springer, Dordrecht (2014). https://doi.org/10.1007/978-94-017-9088-8_15. ISBN 978-94-017-9088-8