

Impact of Adaptive Feedback on Learning Programming with a Serious Game in High Schools' Classes

Matthieu Branthôme
matthieu.branthome@univ-rennes.fr
Univ Rennes, IRISA, CNRS
Lannion, France

Sébastien Lallé
sebastien.lalle@lip6.fr
Sorbonne University, LIP6, CNRS
Paris, France

Abstract

This study evaluates the impact of an adaptive feedback system in Pyrates, a programming serious game designed to ease the transition from block-based to text-based programming in high school classes. The adaptive feedback system was implemented to support student learning and lessen teachers' intervention workload in the classroom. To assess its effectiveness, a field user study was conducted with 190 high school students across two institutions. Results show that students progressed significantly further in the game when using the adaptive feedback system, as compared to playing without feedback, although it did not affect learning gains. We discuss the implications of these results for the design of adaptive feedback in programming serious games.

CCS Concepts

• **Applied computing** → **Interactive learning environments**; • **Computing methodologies** → *Supervised learning by classification*; • **Human-centered computing** → *Field studies*.

Keywords

Adaptive feedback system, Serious game, Learning programming, Machine learning, Pedagogical Agent, Field study, K12, Pyrates

ACM Reference Format:

Matthieu Branthôme and Sébastien Lallé. 2025. Impact of Adaptive Feedback on Learning Programming with a Serious Game in High Schools' Classes. In *33rd ACM Conference on User Modeling, Adaptation and Personalization (UMAP '25)*, June 16–19, 2025, New York City, NY, USA. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3699682.3728326>

1 Introduction

Block-based programming languages, such as Scratch [28], have become a popular medium for introducing coding to young children [4], with research highlighting its advantages over traditional text-based approaches [37, 50]. However, in higher education, text-based programming remains the primary choice for advanced computer science training, and this is even more pronounced in the industry, where languages like Python and Javascript dominate. As a result, students who begin with block programming during

their early education often face the challenge of transitioning to text-based coding. Supporting this transition remains a key open question in the field of computer science education [27, 49]. The Pyrates online application was developed with this objective in mind [7, 8]. Designed as a serious game, it aims to introduce high school students with prior experience with block programming (Scratch) to the Python textual language. It offers learning materials and a series of assignments that facilitate the acquisition of fundamental programming in Python. Additionally, Pyrates serves as an editing environment, drawing inspiration from block-based programming editors to leverage their benefits.

Research has shown that serious games like Pyrates can effectively promote the learning of programming concepts in an engaging way [17, 20, 24, 44]. However, despite their engaging nature, some students still struggle with programming difficulties and misconceptions, and the open-ended design of serious games can sometimes be disorienting [9, 13, 14, 25]. High school teachers often face additional challenges, such as limited time and resources to review student code and provide personalized feedback in crowded classrooms [8, 12, 44, 45]. To address these issues, a promising research direction involves providing automatic feedback tailored to students' difficulties, to support students' learning while easing the burden on teachers in the classroom [16, 29, 38].

While there has been extensive research on designing adaptive feedback for programming, e.g., [16, 29, 38], only a few studies formally evaluated the impact of an adaptive feedback system in programming serious games [18, 46], none of them targeting the transition from block-based to text-based programming. In this work, we aim to investigate whether such adaptive feedback integrated in Pyrates can be effective in supporting this transition. To do so, we build on our previous work where we designed machine-learning-based feedback selection models for Pyrates based on in-game students' data [7]. Specifically, we report on how we integrated and deployed the machine learning (ML) models into an adaptive feedback system for Pyrates, and conduct a formal user study in several 10-grade classes in two high schools to evaluate it. The study took place in actual Python lab sessions with CS teachers to maintain high ecological validity, where students learned either with Pyrates augmented with our adaptive feedback system, or without feedback. Based on the data we collected, we explore five research questions to assess the value of the adaptive feedback system in Pyrates:

- (RQ1) To what extent students use the adaptive feedback system, and what feedback do they get?
- (RQ2) Does the adaptive feedback system in Pyrates increase learners' progression in the game, compared to playing without feedback?

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

UMAP '25, June 16–19, 2025, New York City, NY, USA

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-1313-2/2025/06
<https://doi.org/10.1145/3699682.3728326>

- (RQ3) Does the adaptive feedback system in *Pyrates* increase learners' learning gains in key programming concepts in Python, compared to playing without feedback?
- (RQ4) Does the adaptive feedback system in *Pyrates* reduce how often students seek help from their teacher, compared to playing without feedback?
- (RQ5) Do students perceive the adaptive feedback as useful for learning?

In the rest of the paper, we first review existing studies on adaptive feedback in programming learning environments (Section 2). Next, we introduce *Pyrates* and the ML models we leverage in our feedback system (Section 3). In Section 4, we describe our design of the feedback system, its deployment, and pilot tests. In Section 5, we describe the user study we conducted to evaluate the adaptive feedback system, then report the study results in Section 6. Finally, we discuss the implications of our findings in Section 7.

2 Related Work

Pyrates is the only serious game we are aware of that is designed to assist the transition from block to text programming. Beyond *Pyrates*, two main types of learning environments support this transition: translation-based and hybrid ones [27]. Translation environments typically consist of two views: a block-based view for editing programs and a textual view where the blocks are automatically translated into a target textual language. In some environments, such as Patch [39] and Edublocks [41], the textual view is read-only, allowing students to view or execute the translated code but not modify it directly. Other environments, like Pencil-Code [1] and BlockPy [3], allow direct editing in the textual view, with changes automatically synchronized to update the block-based representation in the other view. In hybrid environments such as Pytch [43], Strype [48] and CodeStruct [19], blocks and text are being combined to create a single view. For instance, drag-and-drop and point-and-click can be used to insert high-level block structures (e.g., loops and conditions), while traditional text editing augmented with auto-completion is employed to introduce the expression-level code. *Pyrates* [8] mimics both translation-based and hybrid environments, offering a programming memo with matching Scratch and Python code snippets (general templates and practical examples) along with copy buttons to support code integration, and a beginner-friendly syntax analyzer to reduce programming errors. To the best of our knowledge, there is no automatic feedback system available in any environment specializing in block-to-text transitions.

There has been extensive research on how to design adaptive support in programming education, mostly by providing feedback in the form of automatic hints (see overview in [2, 22]). However, only a few serious games include such adaptive feedback system to foster either block-based or text-based programming. In particular, Nanodoc, a serious games for learning block-based programming with Scratch, infers from students' data their skill levels, so as to automatically provide them with predefined worked examples [46]. A user study with 102 elementary school students showed that they progress further in Nanodoc with the automatic worked examples, as opposed to playing without. They, however, did not investigate the students' learning gains to ascertain skill acquisition, as we do

in our work via a pre- and post-test. We also extend their work by providing different well-established types of feedback, as opposed to solely one (worked examples in this case), to better adapt to the diverse students' needs. In *AutoThinking*, a labyrinth game that fosters simplified block-based programming, an expert Bayesian network is used to recommend automatic hints and feedback in each level [18]. A user study with 78 high school students showed that students achieved significantly higher learning gain when playing with *AutoThinking*, as opposed to learning with a traditional lesson with PowerPoint. However, they did not include a group of students playing with *AutoThinking* without the feedback system, meaning that they could not formally distinguish whether and to what extent the positive results are actually due to the feedback system as we do. There are a few more related work on adaptive feedback in programming serious games, but focused on designing the feedback system, without any formal evaluation of its impact on students' learning. For instance, automatic hint generation based on analysis of students' block-based code was proposed (but not evaluated on students) in BOTS [17], ENGAGE [31] and Hour of Code [36]. In contrast, our work is focused on how to integrate a feedback system in *Pyrates* and formally evaluate it in the field, as compared with playing with no feedback.

Beyond serious games, adaptive feedback has been evaluated with end-users in other types of learning environments for programming. For instance, automatic feedback in iSnap, which fosters block-based programming via geometry problem solving, was found to increase students' engagement, albeit not on students' learning and performance, as compared to learning with iSnap without automatic feedback [30]. In APT, a tutoring system for Lisp, adaptive feedback had a positive impact on students' post-test score as opposed to not receiving feedback [10]. We contribute to this field by exploring adaptive feedback for a different type of learning activity (block-to-text transition in serious games).

3 *Pyrates*

We provide a brief description of *Pyrates* and outline a preliminary feedback selection model that was designed for it, sufficient for the purpose of this paper.

3.1 Description of *Pyrates*

*Pyrates*¹, which was created by Branthôme during his PhD [6, 8], consists of a platformer game allowing to control an avatar via a Python program. The pirate character must complete a variety of game goals. The different levels of this game were designed by implementing the constructivist learning paradigm [42], meaning that the fundamental programming concepts involved in each of the eight levels (e.g., conditions, loops, variables) are not explicit, but are made necessary by the game problems to be solved. The students can engage with the game material and components to learn on their own about the character control functions (e.g., walk, jump) and the programming concept (e.g., for loop) required to complete the game levels. Each game level introduces a new programming concept (see Table 1). A previous study has evaluated the usability and effectiveness of *Pyrates* with 240 high school students [8].

¹Game URL: <https://py-rates.org/> ; Pedagogical guide: <https://py-rates.org/guide/EN/>

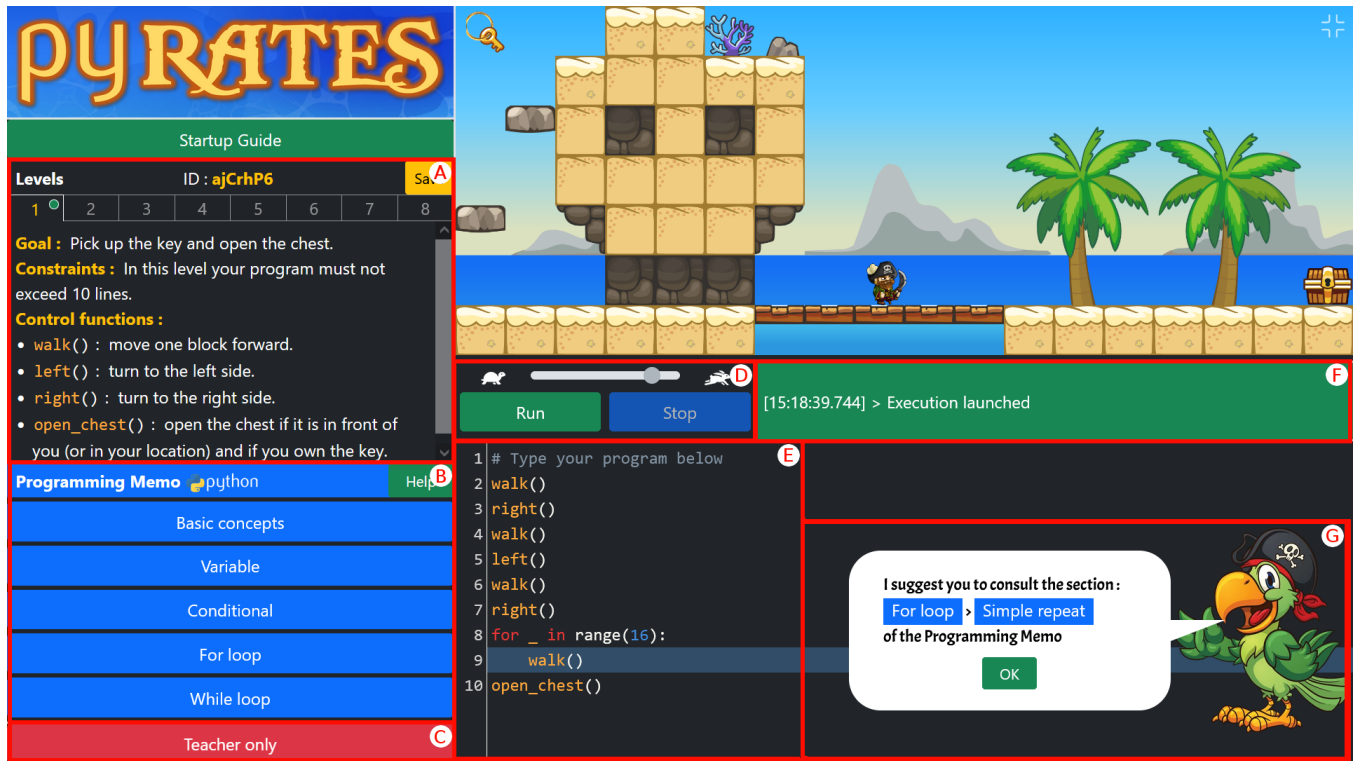


Figure 1: Different areas of Pyrates' graphical interface.

The Pyrates' graphical interface is shown in Figure 1. For each level of the game, the *Level* area (see Figure 1a) indicates the game goal to be achieved (e.g., pick up the key and open the chest), the level constraints (e.g., dot not exceed 10 lines), and the basic character control functions that can be used (e.g., walk, jump). A *programming memo* (see Figure 1b) provides students with learning materials and examples on the implementations of key programming concepts (for loops, variables, etc.) in Python and the corresponding Scratch blocks. A control panel (see Figure 1d) enables to manage the execution (run, stop, and speed slider) of the Python program written by the students in the code editor (see Figure 1e). When the *Run* button is pressed, the edited code is verified by a beginner-suitable syntax checker and then executed by the Python interpreter. Possible syntactic and semantic errors are displayed in the console when running the code, similarly to the standard Python editors (see Figure 1f).

Each level introduces one new programming concept, as summarized in Table 1, except Levels 2 and 7 which are meant to practice variations of the for loops previously introduced. This was meant so that students can focus on the use and Python implementation of one concept at a time.

3.2 Feedback in Pyrates

As part of his PhD thesis, Branthôme [7] built ML models meant to dynamically select relevant feedback to students based on their in-game data. However, these ML models were never actually integrated in Pyrates nor tested on actual students, which is the purpose

Table 1: Programming concepts per level.

Level	Programming concept
1	For loop without loop variable
2	<i>Idem</i>
3	Variable
4	Two-branches condition
5	Three-branches condition
6	For loop with loop variable
7	<i>Idem</i>
8	While loop

of this paper. We describe here the feedback selection models, and we will explained in the next section (Section 4) how we build upon this work to actually integrate, deploy and evaluate a fully fleshed-out adaptive feedback system in Pyrates.

The types of feedback selected by the ML models are derived from the Narciss' framework of tutoring feedback strategies [33], which is well-established in the educational field and was extensively used to design feedback in CS education [5, 15, 22, 26]. Specifically, the ML models are meant to select among four high-level types of feedback: Eliciting the goal of the level and the control functions available to do so (*CTRL*); Eliciting the programming concept required to solve the level (*CONC*); Eliciting how to implement the target programming concept in Python (*IMPL*); and Providing the correct complete solution (*SOLU*). This corresponds to the four

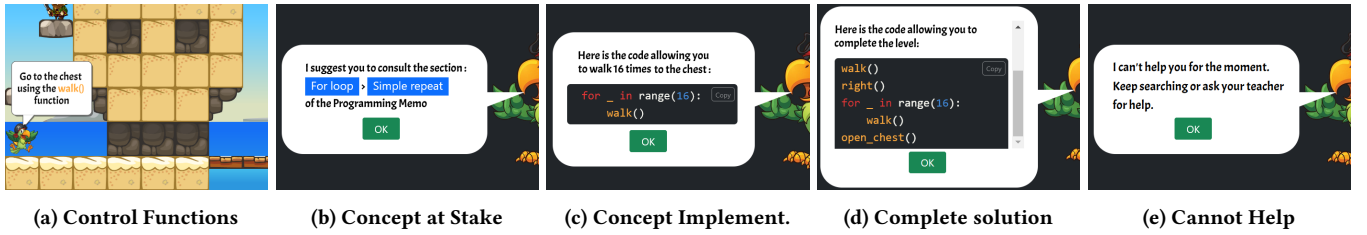


Figure 2: Examples of different feedback types.

feedback strategies from Narciss’ framework that they found to be the most used in classes with Pyrates in [7]². This feedback was implemented in Pyrates where it is delivered by a pedagogical agent in the form of a parrot (hereafter referred to as the *parrot tutor*) as shown in Figure 1g, and combines different modalities (text, images, animations) to increase engagement [40].

Consider, for instance, the first level involving the *for loop* concept, where the character must pick up a key to open a chest with a maximum of 10 lines of code. In the first feedback type (CTRL), the parrot flies over the game map to explain what the pirate character must do and with what control functions (see Figure 2a). Feedback 2 (CONC) reveals the programming concept at stake (*for loop*) with a reference to the programming memo (see Figure 2b), where students can see *for loop* snippets in Scratch and Python, as said above. Feedback 3 (IMPL) gives assistance on how to implement this concept in Python (see Figure 2c). Finally, feedback 4 (SOLU) provides a sample code for completing the current level (see Figure 2d). SOLU feedback is typically recommended to help the least successful learners with a concrete and directive form of feedback, so as to keep them engaged [40]. Such SOLU feedback is meant to be triggered as a last resort, when students show signs of frustration and are close to giving up. The content of the feedback for each level was created by a CS educational researcher and teachers, resulting in 4 (types of feedback) x 8 (levels) = 32 feedback instances.

In [7], the feedback selection models are based on supervised multi-class ML classifiers that aim to predict, for each level, a suitable feedback type whenever a student asks for help. To train the classifiers, Branthôme obtained about 1200 labels during a field study about what feedback types (CTRL, CONC, IMPL, or SOLU) CS teachers provided to their students during learning with Pyrates in 10th-grade CS lab classes. Specifically, teachers could trigger one of the four feedback types to be delivered by the parrot tutor in a Wizard-of-Oz fashion. To train the classifiers, a battery of features was engineered, based on the Python code written by students and their in-game actions so far, which were all logged at runtime. The results showed that a Random Forest classifier tuned with 100 trees outperformed all other models and baselines, reaching F1 scores ranging from about 0.6 to 0.7.

The aforementioned feedback types and ML models were meant to provide controlled guidance to the students while avoiding as much as possible to reveal the correct solution, following the Narciss’ framework. They were also intended to relieve the teacher of

some of the most common needs for help, allowing them to focus on the more severe misconceptions and at-risk students who need additional support.

4 Feedback System Integration in Pyrates

We report in this section how we integrated and deployed the feedback and ML models outlined in Section 3.2. To do so, we leverage the trained Random Forest classifiers, along with the training dataset³. Integrating adaptive feedback in Pyrates required to address three main challenges related to the lack of trained classifiers for certain levels, the prediction threshold, and the SOLU feedback type, as described below.

Lack of ML models in some levels. In their work [7], the authors trained the ML models for only half of the levels in the game (namely levels 1, 3, 4, and 5), which were the levels for which sufficient labels were collected to train the models. They reported that the lack of labels in level 2 is due to the fact that students needed little help in that level, as it does not introduce a new concept (see Table 1). As for Levels 6 to 8, insufficient labels were available because too few students made it that far. However, disabling the adaptive feedback system in these levels could disorient the students and the teachers, and violate UX/UI principles regarding consistency [21]. Instead, we implemented a *data-informed custom policy* for these levels. Namely, we analyzed the distributions of students’ completion times in these levels and computed the quartiles. We then developed the following custom policy: CTRL feedback is provided if the time spent in the level so far by the student falls in the first quartile; CONC feedback is provided in the second quartile; IMPL feedback is provided in the third quartile; and SOLU feedback is provided in the last quartile. We did so because these feedback types are to some extent incremental, from the more general (CTRL) to the most specific (SOLU) to solve a level. We acknowledge that this policy might be suboptimal, as unlike the ML models, it does not adapt to individual students’ code and behaviors. However, we deem it sufficient due to the nature of the target levels. Specifically, as mentioned in Section 3, levels 2 and 7 do not introduce new programming concepts. Level 6 only introduces a variation of the *for loop* with a variable, both of which were covered earlier. Level 8 is usually only reached by the best students who require less help on the first place. It is also the only level about the *while loop*, which we do not consider in our later analysis.

Decision threshold. Because the feedback type selection in each level is based on a four-class classifier, it outputs a probability

²Narciss calls these feedback types *knowledge about task constraint* (i.e., CTRL feedback in Pyrates), *knowledge about concepts* (CONC), *procedural knowledge* (IMPL), and *knowledge of the correct response* (SOLU).

³Both available in open-access at https://github.com/Matthieu-BRANTHOME/PyratesIA/tree/aied_modular.

vector corresponding to the four possible feedback types (to remember: CTRL, CONC, IMPL, and SOLU). By default, predictions can be performed by selecting the feedback type with the highest probability. However, in some instances, we observed that the ML models exhibit low confidence, resulting in probabilities that are very close for two or more classes. In the worst-case scenario to make a decision, all probabilities are close to 0.25. In practice, this means that feedback would be delivered regardless of the ML model's confidence. To address this, we opted to set a decision threshold. If the highest probability outputted by a ML model falls below that threshold, the parrot tutor would not provide feedback. Instead, it encourages students to seek help from their teacher (see Figure 2e). We refer to this type of intervention as a *No-Help* (NOHP) feedback.

To tune the decision threshold, we exported the outputted probabilities of the ML models on the test sets used to evaluate the classifiers. We then computed the smallest threshold for which the proportion of NOHP feedback would not exceed 30% of all feedback provisions, which ended up being 0.47. We deem 30% of NOHP feedback to be a good compromise between overwhelming the teachers with too many students' help requests and maintaining confidence in the feedback, as 0.47 is close to the common 0.5 threshold used in binary classification.

An additional benefit of using a decision threshold is that it leads to more accurate predictions whenever a feedback type is predicted with a confidence greater than the threshold. This is because more confident predictions tend to be more accurate. When applying the 0.47 threshold on the aforementioned test sets, it results in an increase in F1 scores of about 8% across the game levels.

SOLU feedback. As explained in Section 3.2, providing the correct solution (SOLU) is intended as a last resort to prevent students from giving up when they are stuck despite receiving prior feedback. However, providing unwarranted SOLU feedback would be very concerning, as it may hinder students' learning as well as the exploratory nature of Pyrates. Although incorrect predictions of the SOLU feedback were found to be rare in [7], they still amount to about 4% of all predictions. To further limit this issue, we added a meta-rule to prevent SOLU feedback from being delivered too early in a level. We defined "too early" as before half of the average completion time of each level. If a SOLU feedback instance is predicted too early, the parrot provides NOHP feedback instead.

Deployment. The ML models augmented with the aforementioned additions constitute our final adaptive feedback system (simply *feedback system* from now on)⁴. To deploy it, we first retrained the ML models on the entire dataset⁵, and deployed the trained models as *dill* files on a web application served by Flask⁶ (see Figure 3). Next, we modified the code of the parrot tutor so as to integrate the decision threshold and the SOLU meta-rule. Finally, we added a "Help" button in the game interface (see Figure 1b), allowing students to request feedback at will. When clicked, this button sends a HTTP query to the Flask server that includes the current level and all of the students' data accumulated so far in JSON format. The server then returns the selected feedback type along with the

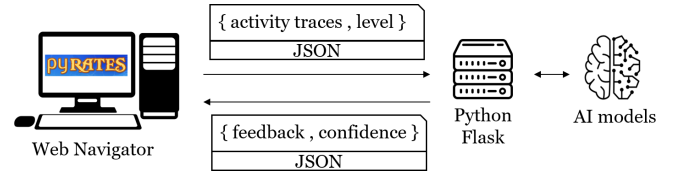


Figure 3: AI models integration in Pyrates application.

ML models' confidence, so that the parrot tutor can display in the browser the corresponding feedback instance for the current level, or NOHP, as explained above.

Pilot tests. To test this setup, we conducted a pilot test with four students matching the Pyrates' target audience, namely high school students with knowledge of Scratch and no prior Python experience. The pilot test revealed no bugs, and all students reported that the interaction with the parrot was intuitive and helpful. We also examined the proportion of NOHP feedback and confirmed that it did not exceed 30% of all predicted feedback. Based on these results, we decided to carry out a formal classroom evaluation of the feedback system, as reported in the next section.

5 User Study to Evaluate the Feedback System

To evaluate the feedback system described in the previous section, we conducted an in-classroom user study in a naturalistic setting, as reported in this section.

Participants. The user study took place in Fall 2024 at two different French high schools, located respectively in Lannion (a town in Brittany) and Cergy-Pontoise (in Paris' suburbs). It involved eight regular 10th grade CS classes⁷, representing 190 students. Each class was randomly split in two groups, one (C group) who played with the default Pyrates game without feedback (control condition from now on), and one (E group) who played Pyrates with the feedback system (experimental condition from now on). This was meant to control for the possible impact of the teachers on their students' learning, as each class was supervised by their regular CS teacher. The C and E groups are fully balanced (95 students in each), and three experienced teachers were involved in the study.

All students have been exposed to Scratch between the 6th grade and the 9th grade, per the national curriculum, but none of them have taken a course about Python yet, thus meeting the requirements for learning with Pyrates. Both the control and experimental conditions exhibit near gender parity.

Study protocol. The study was conducted over three 1-hour lab sessions. In the first session, the teacher informed the students about the study and introduced Pyrates, then administered a 12 minutes pre-test (described below) meant to assess the students' level in Scratch. Next, the students selected their pirate character in the game and were then free to explore the application and progress through the 8 game levels for the rest of the session, as well as during the second and third sessions. The sessions were spaced 2 weeks apart. At any time, students could seek help from their teacher (including following a NOHP feedback). In that case, the teacher would assess the student's difficulties and provide help as they normally do in their lab classes. Whenever teachers provided

⁴Watch a demo video of the feedback system for game level 1: <https://files.atypon.com/acm/d80c246de88c62123ce580bdcb712a5>.

⁵We did so because in [7], the ML models were trained in *k*-folds cross-validation, which produced *k* different models.

⁶<https://flask.palletsprojects.com/en/stable/>

⁷In the French system: *Sciences Numériques et Technologie* (SNT) classes.

help to a student, they asked the student to hit the "Teacher's help" red button (see Figure 1c), so as to log all teachers' interventions (see "material" below). At the end of the third and last session, the students took a Python post-test analogous to the Scratch pre-test, and filled a short usability survey. A Data Protection Officer assessed the study as compliant with the EU's General Data Protection Regulation. The study was approved by the head of the targeted high schools. In accordance with national regulations, IRB approval was not required for this study.

Material. The automatic feedback delivered by the system, along with the teacher interventions (i.e., clicks on the "Teacher's help" button), and all students' in-game actions and executed programs are recorded in a fully anonymous way in a database in xAPI format [23], for a total of 64,360 recorded statements.

The pre-test and post-test are based on the Second CS1 Assessment (SCS1) [34]. Specifically, we extracted for the SCS1 a series of 9 multiple-choices questions targeting the key programming concepts of Pyrates (for loop, variables, conditions). Namely, for each concept, there were three questions of increasing difficulty. The SCS1 is meant to be language agnostic, which enabled us to translate it in Scratch and Python. Importantly, we opted to do the pre-test in Scratch and not Python for several reasons. First, establishing the students prior knowledge with Scratch is required to assess whether Pyrates was actually successful in supporting the transition from Scratch to Python. Second, students have never taken a Python course as said above, meaning that a pre-test in Python would not be very helpful and possibly disorienting to students. In fact, this was the approach used in similar work, e.g., [24].

As for the usability survey, it includes two questions presented as sliders ranging from 0 (worst) to 100 (best). The first asked students to rate the usefulness of the feedback system ("Did you find the feedback useful for progressing in the game?"). The second asked about their intent to reuse the feedback system in the future ("Would you like to continue receiving feedback in the future?").

The tests and the survey are available in open-source at https://github.com/Matthieu-BRANTHOME/Pyrates_IA_V3/blob/main/tests/Pre-test_Post-test.pdf.

6 Results and Analysis

In this section, we describe the results of the experiment described above. We present findings related to each RQ outlined in the introduction. The data and code of the analysis are available at https://github.com/Matthieu-BRANTHOME/Pyrates_IA_V3/tree/main.

6.1 Tutor usage (RQ1)

This subsection deals only with students in the experimental group (E), that is, those who had access to the feedback system. During the course of the game, these students asked the tutor for help, i.e., clicked the "help" button in the Pyrates interface, an average of 22.31 times ($std.dev. = 14.81$). This represents 2,119 requests over the whole experiment. These figures demonstrate that the feedback system was widely adopted throughout the study.

Regarding the response of the feedback system, we can see in Figure 4, which indicates the distribution of received help by level, that most of this feedback came from levels that rely on a ML model (L1,L3,L4,L5). This represents 78.16% of the received feedback. This

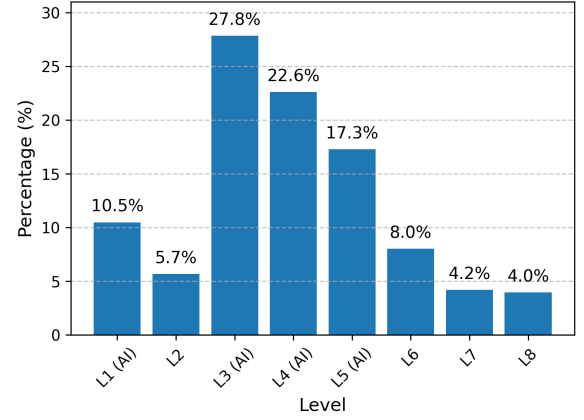


Figure 4: Distribution of received feedback by level.

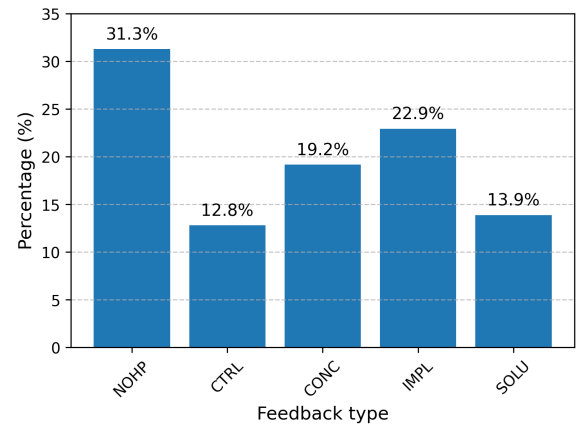


Figure 5: Distribution of different types of received feedback.

is likely because these levels are the most challenging as they introduce the concepts of loop, variable and condition [8]. The custom policy deployed in the other levels is less solicited by students (21.84% of feedback). We can thus claim that the levels requiring the most help are those featuring ML-driven feedback.

Then, as shown in Figure 4, which gives the distribution of the types of feedback provided by the tutor, the types are relatively balanced, even if the implementation of the concept at stake (IMPL) dominates. This is fairly consistent with the distribution of feedback provided by teachers during the data collection in [7] that allowed ML models to be trained (see Section 3.2). Note that feedback indicating the complete solution (SOLU) was not too frequent (only 13.9%), as we intended. Lastly, the tutor is unable to respond (NOHP) to about 30% of student requests. This is the same percentage used to tune the confidence threshold of the classifiers (see Section 4), which validates this choice.

In 74.7% of the cases of NOPH feedback, the decision threshold of the ML models was not reached. Next, around 21.1% of the NOPH feedback were due to a SOLU feedback occurring too early in the level. Finally, 4.2% of NOPH feedback were due to a technical

Table 2: Game progression comparison between groups

Metric	Group E	Group C
Average reached level	5.60	4.49
Standard Deviation (SD)	1.74	1.66
Difference E-C (absolute)	1.11	
Difference E-C (percentage)	+24.69%	
Mann-Whitney U test (p -value)	<0.001	

problem with the server supporting the ML models, which made it impossible to provide feedback.

6.2 Game progression (RQ2)

In this subsection, we evaluate the differences between the two groups in terms of progress in the game over the three lab sessions. Table 2 shows the average level reached by the two groups. We can see that students in the experimental group (E) reached significantly higher levels on average compared to control group (C). This difference is statistically significant⁸ ($p < .001$, see last row of Table 2), with group E outperforming group C by 1.11 levels on average (nearly 25% higher). These results indicate that the feedback system available in the experimental group had a positive impact on students' ability to progress in the game.

6.3 Learning gains (RQ3)

We focus here on student learning gains. Specifically, Table 3 shows the students' pre-test and post-test scores for both groups, along with their learning gain (defined as *post-test score* - *pre-test score*). The scores are normalized from 0 to 100 for the sake of clarity.

First of all, we can see that both groups demonstrated significant within-group improvement in scores, as shown by Wilcoxon tests (see "Within-group learning gains" row in Table 3). Thus, using the Pyrates game, both with and without the feedback system, results in a significant learning gain of around 14%.

Secondly, the learning gains between groups are very similar. A Mann-Whitney U test confirms that the difference in learning gains among the E and C groups is not significant (see "Comparison between groups" rows in Table 3).

Thus, these results suggest that, while Pyrates does foster learning, our feedback system did not have a significant impact (positive or negative) on learning gains as compared to the control group. We will discuss further this finding in Section 7.

6.4 Teachers' Interventions (RQ4)

This subsection examines the amount of teachers' interventions in their classes during the study. Note that, as described in Section 5, these students' requests to the teacher may be either the result of a NOHP feedback, or a spontaneous initiative.

Table 4 describes the average number of help requests made by students to the teacher in each of the two groups logged during the study. It should be noted that these data only concern 30% of the students of the study ($N = 55$), since for on-sites issues it was not possible to collect teachers' interventions in all classes. A post-hoc

⁸In this paper, we employ non-parametric tests (Mann-Whitney and Wilcoxon), because the data are not normally distributed, as per Shapiro-Wilk tests ($p < .05$).

Table 3: Learning comparison between groups

Metric	Group E	Group C
Pre- and post-test scores		
Average pre-test Score (%)	46.52	49.94
Standard Deviation (SD)	19.42	22.52
Average post-test Score (%)	53.12	56.81
Standard Deviation (SD)	20.61	22.89
Learning gain (absolute)	6.60	6.87
Learning gain (Standard Deviation)	15.05	17.21
Learning gain (percentage)	+14.19%	+13.76%
Within-group learning gains		
Wilcoxon Test (p -value)	<0.001	<0.001
Comparison between groups		
Learning gain difference E-C (absolute)	-0.27 (C > E)	
Mann-Whitney U test (p -value)	0.67	

Table 4: Teacher interventions comparison between groups for the 55 students for which this metric was collected.

Metric	Group E	Group C
Average nb. of teacher interventions	1.61	2.30
Standard Deviation (SD)	1.83	3.06
Difference E-C (absolute)	-0.69	
Difference E-C (percentage)	-30.01%	
Mann-Whitney U test (p -value)	0.75	

power analysis⁹ shows that we have enough statistical power ($> .80$) only to capture large effects, but we can still discuss the trend in the data.

Table 4 shows that students in the experimental group (E) required 30% less teacher interventions on average than the control group (C), albeit the difference is not statistically significant ($p = 0.75$). Therefore, we cannot formally conclude for RQ4. Although not significant (perhaps due to a lack of power), this difference is nonetheless substantial. It has been calculated that the average teachers' intervention time when helping a student in a Pyrates course is 3.2 minutes [7]. With an average difference of 0.69 intervention per student between the groups, this represents a gain of 53 minutes ($0.69 \times 24 \times 3.2$) for a teacher with an averaged class size of 24 students over the three sessions. This time can be used, for example, to give more attention to students who need more help to overcome their difficulties.

6.5 Students' perception of the feedback (RQ5)

In this subsection, we analyze the students' perceived usefulness of tutor assistance during the experiment. As explained in Section 5, students rated their experience on a scale from 0 to 100. The results indicated that the average score for the perceived usefulness of the tutor's help to progress in the game was 62.13%, reflecting a moderate level of effectiveness in supporting progress. Additionally, when asked whether they would like to receive similar assistance in the future, the average score was substantially higher at 72.64%, suggesting a strong preference for continued access to this type

⁹Performed in G-Power for the Mann-Whitney test with $\alpha = .05$, $1 - \beta = .8$, $N = 55$.

of support. These findings highlight a positive inclination towards tutor-facilitated learning interventions, while showing that there is an opportunity to improve their perceived effectiveness.

7 Discussion

Below, we outline key takeaways for each RQ and their implications.

RQ1. For RQ1 (extent of usage of the feedback system), our findings do show that students extensively used the feedback system, requesting feedback up to 22.3 times on average. This confirms that learning with such exploratory serious games remains challenging to some students, underscoring the need for some form of guidance, such as the one provided in *Pyrates*.

RQ2-RQ3. As for the impact of the feedback system on students' progression in the game (RQ2) and learning gains (RQ3), we found positive results for RQ2 but not RQ3. Specifically, the students progressed significantly further in the game with the automatic feedback, clearing 5.6 levels on average (out of 8 levels in total), as compared to 4.5 levels on average for students who played without feedback. This finding do show that the feedback system can effectively support the in-game progression, which is its main purpose. However, this did not result in higher learning gains from the pre-test to post-test, i.e., there is statistically no difference in learning gains among the two conditions. Yet, we did find that all students significantly improved from the pre-test to the post-test, showing that *Pyrates* does foster learning to some extent, regardless of whether the feedback system is enabled. While this highlights a limitation of our approach, it echoes research on adaptive support in education, which suggests that achieving a meaningful impact on learning beyond short-term measures of progression can be challenging [11, 35], especially in primary and high school settings [47]. While more work is required to fully examine this result, we can provide a few possible explanations for this finding and possible ways to tackle it. First, with the current study, the additional game progression enabled by the feedback system was of about 1 more level cleared on average, which might have been not sufficient to have a substantial impact on learning gains. However, since most learners did not finish the game, additional playtime could enhance the impact of the feedback system on progression, potentially increasing its effect on learning. Second, students could seek help from their regular CS teacher at any time, potentially reducing the impact of the feedback system on learning, i.e., teachers' interventions may have compensated for the absence of feedback in the control condition. This underscores the need to better understand how to design adaptive feedback that effectively enhances teacher facilitation when learning with *Pyrates* at school. Third, the pre-test and post-test were solely focused on usage of the three target programming concepts in isolation (for loop, condition, variable), whereas in *Pyrates*' levels students are typically expected to combine these concepts. Thus, incorporating additional test items to assess students' ability to combine these concepts in problem-solving could help determine whether the feedback system has an impact on their performance on these items. Interestingly, in related work on some programming serious games (i.e., *Nanodoc*, *AutoThinking*), previous studies did show that such games can support in-game progression [18, 46], but they did not formally compare the effect of their feedback system on learning gains, as

compared to playing with no feedback. Altogether, this shows that it is important to include formal measures of learning beyond game progression, and that future studies are needed to further assess the effectiveness of feedback systems in this context.

RQ4. Our findings for RQ4 (extent of help sought from teachers) are not significant, suggesting that students are still as likely to seek help from their teachers regardless of the presence of the feedback system. This can be due in part because we specifically designed the feedback system to invite students to ask their teacher for help whenever the confidence of the feedback selection models were too low. Nevertheless, we still noticed a trend for less teachers' interventions in the experimental condition (1.6 requests per student on average), as compared to the control condition (2.3 requests). This is a 30% reduction, which is quite substantial when summed over all students, and could allow teachers to focus more on at-risk students. As explained in Section 6.4, we only got data from a subset of the students (55) to investigate RQ4, so it is also possible that we lacked of statistical power to formally explore this trend. Beyond our study, we do not know of any other work on programming serious game that leveraged similar metrics on reliance to teachers' help. Thus, we encourage future work to further investigate whether and how automatic feedback can reduce the teachers' load in the classroom.

In addition, it would be interesting in future work to investigate the duration and nature of the help provided by the teachers (e.g., via audio recording synchronized with the game logs), so as to better understand whether and how the teachers' interventions differ across the two conditions. Furthermore, in the experimental condition, this would allow us to investigate whether the feedback system enables teachers to focus on deeper or more specific issues, as opposed to providing help on the programming concepts already fostered by the feedback system.

RQ5. Lastly, the students' perception of the feedback system was overall positive (RQ5), as a majority of the students found it useful (62%) and 73% of them expressed their intent to continue using it. This indicates that students are overall willing to use such system, providing a promising foundation for future research in this area.

8 Conclusion

Summary of findings. In this work, we formally evaluated the effectiveness of an AI-driven feedback system for the *Pyrates* serious games, as compared to playing the game without automatic feedback. *Pyrates* is meant to support the transition from block-based programming (Scratch) to text-based programming (Python), which is now part of many high school CS curricula. We did so by reusing existing machine learning models to select suitable feedback types in *Pyrates*, and report on how we adapted, integrated and deployed these models to form the feedback system. The user study we designed and conducted took place in naturalistic settings to ensure high ecological validity, involving 190 students across several 10th-grade lab classes of two high schools. To the best of our knowledge, this is the first evaluation of the value of automatic feedback in learning environments for the block-to-text transition.

Our findings indicate that the feedback system was well perceived and extensively used by the students, as well as effective in supporting their progression in the game, as compared to learning

without the feedback system. We did not find a significant impact of the feedback system on the students' learning gains nor reliance on the teachers, and we discussed several hypothesis for that in the discussion. Altogether, our work highlights the potential values and difficulties in deploying ML-driven formative feedback at scale for supporting the block-to-text transition in regular K12 courses.

Limitations and future work. There are limitations in our work that could be the target of future work. First, our findings are limited to one serious game, albeit *Pirates* is based on a commonly used game principle to teach programming, namely character control in a maze via a program written by the student. This game approach can be found for instance in *CodeCombat* [24], *BOTS* [17], and *SPY* [32], to cite a few, thus future work could be focused on ascertaining the generalisability of our feedback system in some of them. Second, we set the NOPH feedback rate to 30% when tuning the decision threshold as a compromise between maintaining high ML prediction confidence and avoiding overwhelming the teachers. However, further exploration of how different decision thresholds could impact our metrics would be valuable. Third, while we managed to involve three teachers in the study, it would be valuable to assess with a larger group whether and how the feedback system influences teachers' interventions, classroom management, and the adoption of *Pirates* in their courses. In addition, this would allow to deepen the analysis of students' reliance on teachers, not only in terms of the amount of help received but also its nature.

Beyond the above points, we plan to extend our study in order to explore the hypotheses outlined in the discussion section, in particular regarding learning gains, which may need to be measured along multiple dimensions rather than evaluating each programming concept in isolation. We also aim to extend our feedback system to include other types of feedback, such as worked examples as featured in *Nanodoc* [46]. Another important avenue for future work is in leveraging chatbots powered by LLMs in serious games like *Pirates*, and evaluate their impact on learning. For example, students could ask follow-up questions after receiving feedback. However, this would require extensive work to regulate the nature and amount of help provided by the chatbot while enabling it to 'perceive' the game levels and generate Scratch code suitable to this audience.

Acknowledgments

We are truly thankful to the high school principals for authorizing the experiment and the teachers (Alexandre Bernard, Gael Guélé, and Anne-Claire Gobard) for their dedicated involvement. This work was partially supported by the French National Research Agency (ANR) under Grant ANR-24-CE38-2478-01.

References

- [1] Emma Andrews, David Bau, and Jeremiah Blanchard. 2021. From Droplet to Lilypad: Present and Future of Dual-Modality Environments. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–2. <https://doi.org/10.1109/VL/HCC51201.2021.9576355>
- [2] Pedro Luis Saraiva Barbosa, Rafael Augusto Ferreira do Carmo, João PP Gomes, and Windson Viana. 2024. Adaptive learning in computer science education: A scoping review. *Education and Information Technologies* 29, 8 (2024), 9139–9188.
- [3] Austin Cory Bart, Javier Tibau, Eli Tilevich, Clifford A. Shaffer, and Dennis Kafura. 2017. *BlockPy: An Open Access Data-Science Environment for Introductory Programmers*. *Computer* 50, 5 (May 2017), 18–26. <https://doi.org/10.1109/MC.2017.132>
- [4] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. 2017. Learnable programming: blocks and beyond. *Commun. ACM* 60, 6 (May 2017), 72–80. <https://doi.org/10.1145/3015455>
- [5] Andrew Thomas Bimba, Norisma Idris, Ahmed Al-Hunaiyyan, Rohana Binti Mahmud, and Nor Liyana Bt Mohd Shuib. 2017. Adaptive feedback in computer-based learning environments: a review. *Adaptive Behavior* 25, 5 (2017), 217–234.
- [6] Matthieu Branthôme. 2022. *Pirates: A Serious Game Designed to Support the Transition from Block-Based to Text-Based Programming*. In *European Conference on Technology Enhanced Learning*. Springer, Cham, 31–44.
- [7] Matthieu Branthôme. 2023. *Apprentissage de la programmation informatique: analyses et ressources pour accompagner la transition collège-lycée*. Ph.D. Dissertation. Université de Bretagne occidentale-Brest.
- [8] Matthieu Branthôme. 2024. *Pirates: Design and Evaluation of a Serious Game Aimed at Introducing Python Programming and Easing the Transition from Blocks*. *ACM Trans. Comput. Educ.* (feb 2024). <https://doi.org/10.1145/3639061>
- [9] Cristina Conati and Sébastien Lallé. 2023. Student modeling in open-ended learning environments. *Handbook of Artificial Intelligence in Education* (2023), 170–183.
- [10] Albert T Corbett and John R Anderson. 2001. Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 245–252.
- [11] Eileen du Plooy, Daleen Casteleijn, and Denise Franzsen. 2024. Personalized adaptive learning in higher education: a scoping review of key characteristics and impact on academic performance and engagement. *Heliyon* (2024).
- [12] Osmane El Montaser, Sébastien Lallé, and Mathieu Muratet. 2025. Evaluating the Impact of Automated Hints in a 3D Educational Escape Game: A Comparative Study of Accessibility and Computer Science Versions. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*. ACM, 685–694.
- [13] Judith Gal-Ezer and Chris Stephenson. 2014. A tale of two countries: Successes and challenges in K-12 computer science education in Israel and the United States. *ACM Transactions on Computing Education (TOCE)* 14, 2 (2014), 1–18.
- [14] Tara Gallien and Jody Oomen-Early. 2008. Personalized versus collective instructor feedback in the online courseroom: Does type of feedback affect student satisfaction, academic performance and perceived connectedness with the instructor? *International Journal on E-learning* 7, 3 (2008), 463–476.
- [15] Luisa Greifenstein, Markus Brune, Tobias Fuchs, Ute Heuer, and Gordon Fraser. 2023. Impact of hint content on performance and learning: A study with primary school children in a scratch course. In *Proceedings of the 18th WiPSCE Conference on Primary and Secondary Computing Education Research*. 1–10.
- [16] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. 2017. Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis. In *Conference on Learning@Scale* (Cambridge, MA, USA). ACM, 89–98.
- [17] Andrew Hicks, Barry Peddycord, and Tiffany Barnes. 2014. Building games to learn from their players: Generating hints in a serious game. In *International Conference on Intelligent Tutoring Systems*. Springer, 312–317.
- [18] Danial Hooshyar, Liina Malva, Yeongwook Yang, Margus Pedaste, Minhong Wang, and Heuiseok Lim. 2021. An adaptive educational computer game: Effects on students' knowledge and learning attitude in computational thinking. *Computers in Human Behavior* 114 (2021), 106575.
- [19] Majeed Kazemitabaar, Viktor Chyhir, David Weintrop, and Tovi Grossman. 2022. *CodeStruct: Design and Evaluation of an Intermediary Programming Environment for Novices to Transition from Scratch to Python*. In *Proceedings of the 21st Annual ACM Interaction Design and Children Conference (Braga, Portugal) (IDC '22)*. ACM, New York, NY, USA, 261–273. <https://doi.org/10.1145/3501712.3529733>
- [20] Cagin Kazimoglu, Mary Kiernan, Liz Bacon, and Lachlan MacKinnon. 2012. Learning Programming at the Computational Thinking Level via Digital Game-Play. *Procedia Computer Science* 9 (01 Jan 2012), 522–531.
- [21] Wendy A Kellogg. 1987. Conceptual consistency in the user interface: Effects on user performance. In *Human-Computer Interaction-INTERACT'87*. Elsevier, 389–394.
- [22] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)* 19, 1 (2018), 1–43.
- [23] Jonathan M. Kevan and Paul R. Ryan. 2016. Experience API: Flexible, Decentralized and Activity-Centric Data Collection. *Technology, Knowledge and Learning* 21, 1 (01 Apr 2016), 143–149. <https://doi.org/10.1007/s10758-015-9260-x>
- [24] Chrysoula Kroustalli and Stelios Xinogalos. 2021. Studying the effects of teaching programming to lower secondary school students with a serious game: a case study with Python and CodeCombat. *Education and Information Technologies* 26, 5 (2021), 6069–6095.
- [25] Susan M Land and Michael J Hannafin. 1997. Patterns of understanding with open-ended learning environments: A qualitative study. *Educational Technology Research and Development* 45, 2 (1997), 47–73.
- [26] Nguyen-Thinh Le. 2016. A classification of adaptive feedback in educational systems for programming. *Systems* 4, 2 (2016), 22.

- [27] Yuhan Lin and David Weintrop. 2021. The landscape of Block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. *Journal of Computer Languages* 67 (2021), 101075. <https://doi.org/10.1016/j.cola.2021.101075>
- [28] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* 10, 4, Article 16 (Nov. 2010), 15 pages. <https://doi.org/10.1145/1868358.1868363>
- [29] Samiha Marwan, Bitak Akram, Tiffany Barnes, and Thomas W. Price. 2022. Adaptive Immediate Feedback for Block-Based Programming: Design and Evaluation. *IEEE Transactions on Learning Technologies* 15, 3 (2022), 406–420.
- [30] Samiha Marwan, Ge Gao, Susan Fisk, Thomas W Price, and Tiffany Barnes. 2020. Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science. In *Proceedings of the 2020 ACM conference on international computing education research*. 194–203.
- [31] Wookhee Min, Bradford Mott, and James Lester. 2014. Adaptive scaffolding in an intelligent game-based learning environment for computer science. In *Proceedings of the workshop on AI-supported education for computer science (AIEDCS) at the 12th international conference on intelligent tutoring systems*. 41–50.
- [32] Mathieu Muratet. 2023. How to Characterize and Analyze the Computational Thinking Skills of a Learning Game?. In *European Conference on Technology Enhanced Learning*. Springer, 263–277.
- [33] Susanne Narciss. 2013. Designing and evaluating tutoring feedback strategies for digital learning. *Digital Education Review* 23 (2013), 7–26.
- [34] Miranda C. Parker, Mark Guzdial, and Shelly Engleman. 2016. Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (Melbourne, VIC, Australia) (ICER '16)*. ACM, New York, NY, USA, 93–101. <https://doi.org/10.1145/2960310.2960316>
- [35] Radek Pelánek. 2024. Adaptive learning is hard: Challenges, nuances, and trade-offs in modeling. *International Journal of Artificial Intelligence in Education* (2024), 1–26.
- [36] Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas Guibas. 2015. Learning program embeddings to propagate feedback on student code. In *International conference on machine Learning*. PMLR, 1093–1102.
- [37] Thomas W. Price and Tiffany Barnes. 2015. Comparing Textual and Block Interfaces in a Novice Programming Environment. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (Omaha, Nebraska, USA) (ICER '15)*. ACM, New York, NY, USA, 91–99. <https://doi.org/10.1145/2787622.2787712>
- [38] Thomas W. Price, Yihuan Dong, and Dragan Lipovac. 2017. ISnap: Towards Intelligent Tutoring in Novice Programming Environments. In *SIGCSE Technical Symposium on Computer Science Education* (Seattle, WA, USA). ACM, 483–488.
- [39] William Robinson. 2016. From Scratch to Patch: Easing the Blocks-Text Transition. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education (Münster, Germany) (WiPSCe '16)*. ACM, New York, NY, USA, 96–99. <https://doi.org/10.1145/2978249.2978265>
- [40] Valerie J. Shute. 2008. Focus on Formative Feedback. *Review of Educational Research* 78, 1 (2008), 153–189.
- [41] Gavin Sim, Mark Lochrie, Misbahu S. Zubair, Oliver Kerr, and Matthew Bates. 2023. Moving Away from the Blocks: Evaluating the Usability of EduBlocks for Supporting Children to Transition from Block-Based Programming. In *Human-Computer Interaction – INTERACT 2023*. Springer Nature Switzerland, Cham, 317–336. https://doi.org/10.1007/978-3-031-42280-5_19
- [42] Svein Sjøberg. 2010. Constructivism and learning. *International encyclopedia of education* 5 (2010), 485–490.
- [43] Glenn Strong, Ben North, Brian Gillespie, Sara Fiori, Duncan Wallace, and Nina Bresnihan. 2024. Experiences Trialling a Novel Block-to-text Environment in a Summer School Context. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (Milan, Italy) (ITiCSE 2024)*. ACM, New York, NY, USA, 150–156. <https://doi.org/10.1145/3649217.3653585>
- [44] Lihui Sun, Zhen Guo, and Linlin Hu. 2023. Educational games promote the development of students' computational thinking: a meta-analytic review. *Interactive Learning Environments* 31, 6 (2023), 3476–3490.
- [45] Betül Tonbuloglu and İsmail Tonbuloglu. 2019. The effect of unplugged coding activities on computational thinking skills of middle school students. *Informatics in Education* 18, 2 (2019), 403–426.
- [46] Pavlos Toukiloglou and Stelios Xinogalos. 2023. Adaptive support with working examples in serious games about programming. *Journal of educational computing research* 61, 4 (2023), 766–789.
- [47] Fabienne M Van der Kleij, Remco CW Feskens, and Theo JHM Eggen. 2015. Effects of feedback in a computer-based learning environment on students' learning outcomes: A meta-analysis. *Review of educational research* 85, 4 (2015), 475–511.
- [48] Pierre Weill-Tessier, Charalampos Kyfionidis, Neil Brown, and Michael Kölling. 2022. Strype: Bridging from Blocks to Python, with Micro:bit Support. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 2 (Dublin, Ireland) (ITiCSE '22)*. ACM, New York, NY, USA, 585–586. <https://doi.org/10.1145/3502717.3532155>
- [49] David Weintrop. 2019. Block-based programming in computer science education. *Commun. ACM* 62, 8 (July 2019), 22–25. <https://doi.org/10.1145/3341221>
- [50] David Weintrop and Uri Wilensky. 2017. Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Trans. Comput. Educ.* 18, 1, Article 3 (Oct. 2017), 25 pages. <https://doi.org/10.1145/3089799>