Encouraging Task Creation Among Programming Teachers in Primary Schools

Jacqueline Staub(✉), Zaheer Chothia, Larissa Schrempp, and Pascal Wacker

Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland
{jacqueline.staub,zaheer.chothia}@inf.ethz.ch,
{larissas,pwacker}@student.ethz.ch

Abstract. Programming is being widely adopted as a classroom activity to promote computational literacy across the full spectrum of ages. As of now, however, there is a gap between curriculum designers and the teachers that work directly alongside pupils. Educators build their lessons around predefined curricula and programming environments with limited scope for customization. As a result, their involvement is limited to using teaching resources as black boxes and creating tasks that live external to the programming environment. This work presents a small extension to the XLogoOnline programming environment and demonstrates how non-technical users are empowered to define, share and evaluate their own programming tasks. Our proposed tool is targeted at navigation tasks on a two-dimensional grid. Different categories of tasks can be easily assembled in graphical form and submitted solutions are automatically verified. We report from practical experience over a time span of 18 months and give highlights from a collection of 1331 programming tasks. The tool offers value by allowing teachers to design handouts and exams and also encourages teamwork by allowing pupils to challenge their fellow classmates. Beyond their use in the classroom, the idea of collecting task sets is a useful foundation for self-guided learning, exams and even large-scale competitions – which we intend to pursue in future work.

Keywords: Programming education · Turtle graphics · Creating teaching materials · Automatic validation of submissions

1 Programming Has Entered Compulsory Schooling

Computer science is currently being established as a new school subject in many countries around the world, making programming an activity that hundreds of thousands of children and adolescents are now learning as part of their compulsory education. The details of how and when programming is first taught varies from country to country [17]. To take one example, the UK's National Curriculum contains a dedicated subject termed "*Computing*" that comprises activities such as programming that have deep-rooted origins in computer science. British

E. Barendsen and C. Chytas (Eds.): ISSEP 2021, LNCS 13057, pp. 135–146, 2021.
https://doi.org/10.1007/978-3-030-90228-5_11

children are exposed to those concepts from the very beginning of their schooling career [1]. In contrast, the Swiss Lehrplan 21, which is being implemented by all 21 German-speaking cantons in the country, does not contain a dedicated subject for these activities prior to fifth grade. The curriculum does, however, require that teachers foster algorithmic thinking in an interdisciplinary way from kindergarten on.

Teacher Participation is Key to Enacting Change

The success of introducing a new teaching reform stands and falls with the participation of teachers. Only when teachers actively help and support the ideas of a reform can new ideas take root in the school system. Historically, it is well-known that the implementation of a reform depends largely on whether and how teachers take responsibility for the implementation of new ideas [8].

At the same time, however, there is evidence that teachers often have a sketchy understanding of content when developing their own learning materials and that targeted support is quite useful [5]. With regard to the current introduction of computer science as a new school subject, this is hardly different – teachers are the pillars of this reform and, without their active assistance, meaningful change is not possible.

We argue that most current programming environments do not provide enough opportunity for teachers to participate in the design of teaching materials and teachers are thus forced into the unhelpful role as users of prefabricated materials. Without the ability to respond appropriately to individual circumstances, to generate curricular content themselves, and to integrate it directly into students' learning environments, there is a risk that teachers may not want to support the introduction of a new school subject.

Merits of Integrating Learning Environments with Curriculum

Many different programming languages and environments are used in schools [17]; some of which follow a *free-form* approach while others are *task-based*. These two categories can be distinguished as follows:

- **Free-form environments:** We consider a programming environment to be free-form if it is set up as generalist platform – i.e. it allows for numerous applications while not providing any concrete embedded programming challenges. This category includes several well-known environments such as Scratch and ScratchJr [12], AppInventor [19], TigerJython [9,18], or AgentCubes [6].
- **Task-based environments:** If a learning platform is designed around the idea of a built-in collection of programming challenges that programmers can attempt to solve, we consider the platform to be part of the family of task-based learning environments. Examples in this category include the programming environments Code.org, Lightbot, and Emil [7], as well as the programming competition platforms Pisek [11] and Algorea [10].

Regardless of whether students are taught in a free-form environment or on a task-based platform, the ultimate goal of programming education is typically

to make students explore the most important programming concepts, become proficient in them by creating tangible outputs, and learn how to solve problems in an efficient and elegant way. Most programming curricula used in K–6 revolve around the concepts of sequences, repetition, functions, parameterization, branching, and variables [1]. All of these concepts can be taught using an external textbook or via tasks that are directly integrated into the programming environment.

We argue that neither of these approaches alone adequately addresses the needs of teachers at the moment. Whilst free-form programming environments allow for an infinite variety of different programming activities, decoupling tasks from their associated programming platform has the disadvantage that it is not possible to give pupils automated individual feedback as they progress. Moreover, it is not guaranteed that learners actually use programming concepts in a meaningful way [13]. Task-based learning platforms, on the other hand, integrate a curriculum directly into a digital learning platform which allows for automated feedback. The downside of most platforms belonging to this category, however, consists in their limited number of tasks and lack of customization.

We have addressed these shortcomings by adding a task collection mode to the free-form programming environment XLogoOnline, effectively making it a hybrid task-based and free-form programming platform. The proposed tool allows teachers and students to create their own custom tasks which subsequently can be solved and shared with others. Different categories of tasks can be easily assembled in a graphical user interface and submitted solutions are automatically verified by the programming environment. In the following sections, we will present the technical basis of the tool (Sect. 2) as well as a small selection of tasks that have been created by our user base in the past 18 months (Sect. 3).

2 An Extensible Collection of Programming Tasks

XLogoOnline [14] is a Logo learning environment currently used by roughly 70 000 users every year. Most of these users are primary school students or teachers originating from any of the four language regions of Switzerland. There is, however, also a growing population of XLogoOnline users in various other parts of the world such as Germany, Spain, or Lithuania [15]. The platform was originally designed as a free-form programming environment that is used alongside a programming textbook. The textbooks we usually use [2–4] constitute a spiral curriculum from kindergarten to the end of high school [16].

This work is oriented towards K–2 but nevertheless ties into a larger framework that spans K–6 (children aged 6 to 12 years). Throughout the quoted age range, the environment comprises three stages that guide students from their first timid steps in the world of block-based programming with only a handful of unparameterized instructions to full-featured text-based programming with extended features and the possibility to add any number of abstraction layers thanks to user-defined procedures. Due to their linguistic differences (i. e., parameterized vs. non-parameterized basic commands), the application domain

used with the youngest age group is only remotely reminiscent of classical turtle graphics. Instead of drawing geometric shapes using arbitrary angles and lines, novices ages 6 to 8 years learn to steer the turtle through a two-dimensional grid and solve navigation tasks (see Fig. 1). As we will show, whilst a pared-down command set and grid-based task setting heavily reduce the problem domain for the youngest pupils, with minor tweaks this basic concept still permits a wide assortment of stimulating tasks.

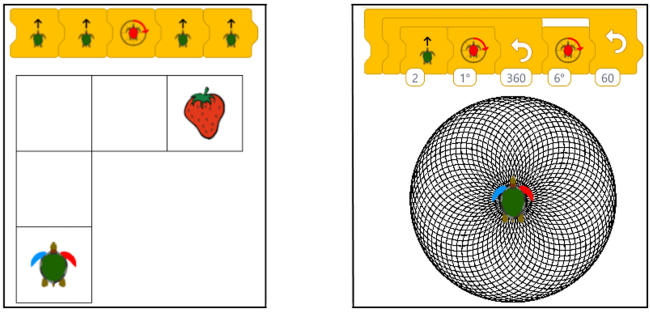


Fig. 1. XLogoOnline contains two different application domains. Students aged 9 years or older are acquainted with traditional turtle graphics where they draw geometric shapes (as shown on the right). Children aged 6 to 8 years, on the other hand, use a simplified vocabulary that does not provide the possibility to draw arbitrary angles. Instead, they learn programming by solving simple navigation tasks as shown on the left.

Programming is not only a challenge for students, but also for their teachers. Children are notorious for only skimming over assignments [20]. As a result, they struggle to verify the correctness of their own solutions and often rely on external guidance to point out mistakes. For teachers with limited experience in programming and holding the responsibility of devoting attention to an entire class, this creates a significant hurdle.

We created a task collection made up of almost 100 predefined tasks[1] which are embedded directly into the XLogoOnline programming environment. All the tasks require students to navigate a turtle on a two-dimensional grid using just the four basic commands `forward`, `back`, `left`, and `right`. To start with, the initial assignments are straightforward and ask the student to find a path to a given object in the grid. Once basic movements have become familiar, additional requirements are introduced such as obstacles that cannot be traversed, multiple objects that need to be collected, or finding a solution with constrained vocabulary. Latter tasks rely on students identifying patterns and writing more concise solutions by identifying symmetry and making use of the command `repeat`.

All of these tasks can be clustered in three rough categories: (i) tasks that contain exactly one target cell, (ii) tasks that contain several target cells which all need to be visited in a specific order or randomly, and (iii) tasks that

[1] https://xlogo.inf.ethz.ch/release/latest/mini.

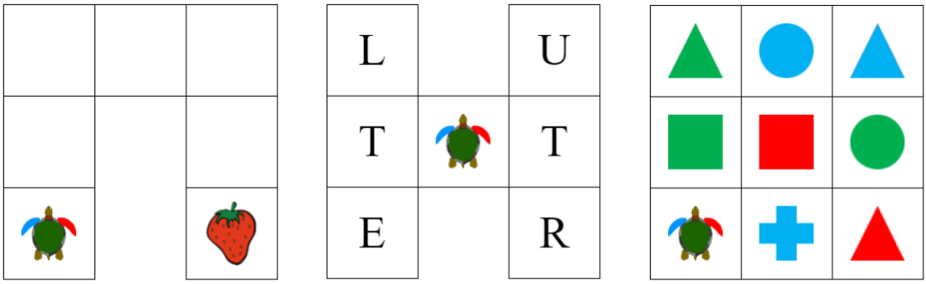


Fig. 2. The three tasks "find the strawberry", "collect the letters in the order implied by the word TURTLE", "find a green triangle without standing on a square" represent instances for each of the task categories (single target, multi-target, added restrictions). (Color figure online)

contain additional restrictions such as forbidden cells or a constrained vocabulary. Figure 2 shows some instances of tasks that cover all three of the mentioned categories.

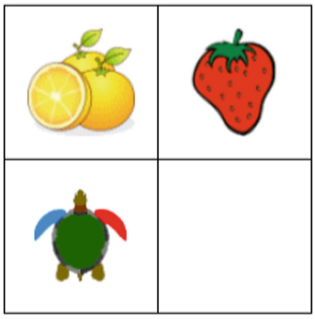
2.1 Finding Common Ground: Task Data Structure

Despite their differing formulation, all the previously-presented tasks share the same fundamental structure. That is, each task consists of three main elements that must be specified individually:

1. **Tiles:** Each task consists of a grid with one or more grid cells that must be arranged contiguously, but not necessarily in a perfect rectangular configuration. Each cell in the grid is uniquely identified by its x- and y-coordinate specifying its absolute location. The coordinate at the top left always corresponds to the location $(0, 0)$. All other cells are identified by their horizontal and vertical distance from this cell.
2. **Turtle:** Each task must contain exactly one turtle whose initial location on the grid must be specified in terms of an x- and y-coordinate. The turtle's initial orientation can be expressed using one of four possible states: 0 (facing North), 90 (facing East), 180 (facing South), or 270 (facing West).
3. **Objects:** As depicted in Fig. 2, there are several kinds of objects that can be used as part of a task assignment. Four categories of objects are provided: (i) strawberry and lemon patches with up to 4 fruit per field, (ii) eight colorful circles, (iii) nine colorful shapes including triangles, squares, and crosses, (iv) Unicode characters such as letters, digits, and emoticons.

Within the programming environment, tasks are represented using an internal data structure that contains detailed information for each of the task elements listed above. Figure 3 shows a side-by-side comparison between a visual task and its corresponding internal representation.

The presented data structure is suitable as a universal representation as it allows millions of different tasks to be described by the same underlying structure. XLogoOnline offers an embedded graphical user interface allowing both



```
"tasks": [
  "tiles": [
   {"x": 0, "y": 0},
   {"x": 1, "y": 0},
   {"x": 0, "y": 1},
   {"x": 1, "y": 1},
  ],
  "turtleData": {
     "x": 0, "y": 1, "dir": 0
  },
  "specialObjects": [
   {"id": "lemon", "x": 0,  "y": 0},
   {"id": "berry", "x": 1,  "y": 0}
  ],
]
```

Fig. 3. A simple task containing four grid cells, a turtle and two special objects. The code shows how the visual task on the left is represented internally.

teachers and students to intuitively develop and share their own exercises without having to look at the structure inside. The interface is able to interpret and visualize any task written in this format.

Without the possibility to check the correctness of student solutions, the tool permits sharing of ideas but serves only half of its educational purpose. The next logical extension is to explain how the data structure can be extended to automatically verify student submissions and – in the case of faulty solutions – to provide targeted feedback, which we will now elaborate on.

2.2 Solution Verification

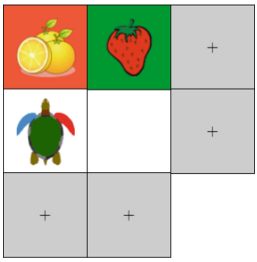
In this section, we explain how arbitrary solutions for the task categories presented earlier can be automatically verified. We discuss what different kinds of tasks exist and how each of these task classes can be verified.

So far, we have managed to represent all navigation tasks with one single unified data structure. Therefore, the question arises: Does this conceptual symmetry also apply to the aspect of solution validation? The answer, sadly, is 'no'. Despite superficial similarities, the correction of seemingly-related tasks often differs substantially and several differing verification strategies are needed to fully cover the spectrum of possible tasks.

Broadly speaking, the realm of different tasks can be clustered into three categories: (i) tasks with one target, (ii) tasks with multiple targets, (iii) tasks with additional restrictions. However, these categories need to be broken down further to actually differentiate all relevant classes from the verification point of view. Specifically, we identify a total of six different task types, namely:

- A: Tasks with one target field.
- B: Tasks with multiple targets.

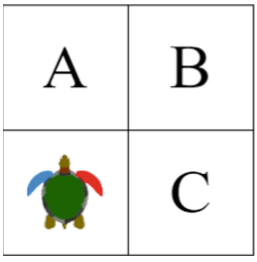
  - B1: Items can be collected in any order.
  - B2: Items must be collected in a specific order.
- C: Tasks with additional restrictions.
  - C1: Tasks with obstacles, i.e., forbidden fields.
  - C2: Tasks with walls, i.e., forbidden passage between fields.
  - C3: Tasks with restricted command set.



```
"tasks": [
  "tiles": [
   {"x": 0, "y": 0, "forbidden": true},
   {"x": 1, "y": 0, "success": true},
   {"x": 0, "y": 1},
   {"x": 1, "y": 1},
  ],
  "turtleData": { ... }
  "specialObjects": [ ... ]
]
```

Fig. 4. A simple task containing one success field and one forbidden field.

The constraints and requirements of categories A, C1, and C2 all relate to properties of the grid structure and cells therein. Accordingly, the "tiles" attribute can be annotated with information that describes the specific task type. Any cell in the grid can be labeled as the designated target by setting the "success" flag in it. Likewise, grid cells can be marked as inaccessible terrain by applying a "forbidden" attribute (as in Fig. 4). In addition, the "walls" attribute is used to shield fields from one side only.

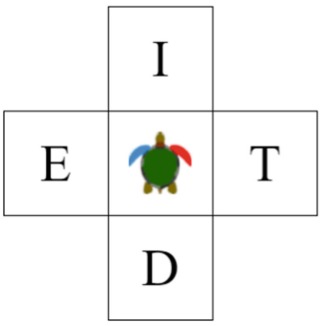



```
"tasks": [{
  "tiles": [ ... ],
  "turtleData": { ... },
  "specialObjects": [
   {"x": 0, "y": 0, "value": "A"},
   {"x": 1, "y": 0, "value": "B"},
   {"x": 1, "y": 1, "value": "C"}
  ],
  "solution": { "1": "ABC" }
 }
]
```

Fig. 5. A task that requires three fields to be visited in a specific order to form the character sequence 'ABC'.

As soon as a task requires more than one target (as in task types B1 and B2), the question arises whether the individual targets are independent of each other.

If this is the case, it is sufficient to annotate several cells with a corresponding "success" flag. If, however, the cells are to be visited in a specific order (as in Fig. 5), an adapted representation must be used. In this case, the task contains Unicode characters which are accessible via the "value" attribute of a special object. Using these values and a simple concatenation of all visited fields, we are able to distinguish the correct solution "ABC" from an incorrect one like "CBA".

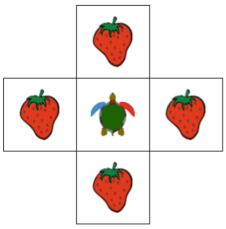



```
"tasks": [{
  "tiles": [ ... ],
  "turtleData": { ... },
  "specialObjects": [
   {"x": 1, "y": 0, "value": "I"},
   {"x": 0, "y": 1, "value": "E"},
   {"x": 2, "y": 1, "value": "T"},
   {"x": 1, "y": 2, "value": "D"}
  ],
  "solution": {
    "1": "DIET", "2": "EDIT",
    "3": "TIDE", "4": "TIED",
  }
}]
```

Fig. 6. The task asks for a meaningful concatenation of the four letters 'D', 'I', 'E', and 'T'. There are, however, multiple correct solutions to this question. We need to note them all as valid solution candidates.

Some tasks may inherently permit more than just one correct solution and there is no technical barrier to prevent users from creating such tasks. Figure 6 depicts a scenario where this is the case: using the four given letters 'D', 'I', 'E' and 'T', we are required to form an English word. The obvious solution "DIET" could be substituted by any of the following alternative words: "EDIT", "TIDE", "TIED". All four solutions are valid and should be recognized as acceptable by the automatic verification mechanism. In order to reach this goal, we need to extend the "solution" entry with all possible solutions to the task such that the algorithm is able to validate that the student's submitted solution comes from this permitted set.

The last category of tasks C2 imposes constraints on the linguistic level (i.e., tasks may restrict the vocabulary that can be used or the number of commands allowed in a solution). This idea is especially intriguing when working with the `repeat` statement which often goes hand-in-hand with a drastic reduction of program length. In order to restrict solutions linguistically, we state which commands are allowed to be used, how many such commands are permitted, or what total length the final program is allowed to have. Figure 7 shows an example of a task with a restricted command set.



```
"tasks": [{
  "turtleData": { ... },
  "tiles": [ ... ],
  "specialObjects": [ ... ]
  "constraints": [
   {"total": {"type": "<=", "amount": 4}}
  ],
}]
```

Fig. 7. All 4 berries must be collected with at most 4 commands.

In this section we presented the underlying representation form and mechanisms used to support custom tasks within XLogoOnline's learning environment. All the previously-presented features are fully integrated and accessible from a graphical frontend. In this way, teachers have the possibility to create custom content for their class and adapt tasks to the linguistic, cognitive, and cultural background of their learners. In the next chapter we dive deeper into this collaborative aspect and present new and imaginative tasks that were created by our user base.

3 Case Study: User Activity During the Past 18 Months

Our tool also offers a graphical frontend from which users can easily design, test and publish custom content directly within the XLogoOnline programming environment. This task creation mode has been publicly available since January 2020 and, from the very beginning, it has been used by both teachers and students alike. Over the past 1.5 years, community contributions have grown to number well over 1331 tasks. To give a glimpse of this valuable dataset, we will now present a few exemplary tasks of particular interest.

One noticeable observation within the data was the presence of numerous misspellings and linguistic errors in a non-negligible proportion of all tasks. These errors are characteristic of our target audience (children aged 6 through 8) and we consequently assume that some teachers actively use the tool with their pupils in class. To help pick apart these differing audiences, we divided the data into two categories using the presence of orthographic errors as a rough indicator of whether the task author was presumably a child or an adult. We acknowledge that this criteria isn't foolproof and defer a more thorough analysis to future work.

Among the tasks we attribute to adults, we see a broad variety of task types and, in particular, several imaginative examples that rely on transfer from other fields (e.g. spatial reasoning, arithmetic) which were not previously included in our task collection in this form. Figure 8 shows two such examples: (i) The first task requires students to connect digits and mathematical operators in a grid in such a way that the resulting text is both a valid and numerically-correct mathematical statement; (ii) The second task falls into the category of tasks with

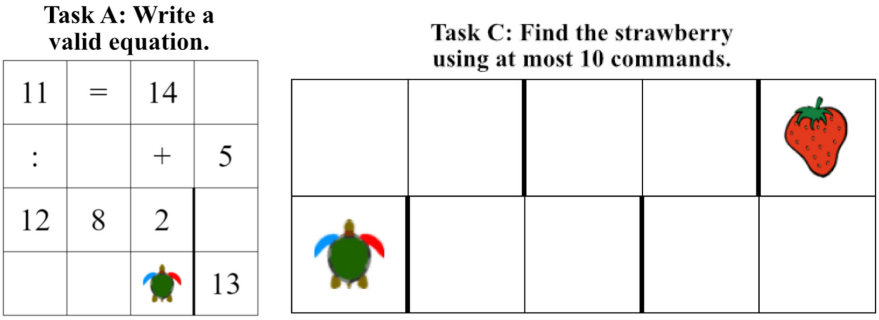


Fig. 8. Two examples of teacher tasks. The task on the left requires pupils to first identify a mathematically valid statement and then make the turtle discover the equation. The task on the right is a natural use case for the repeat statement.

restrictions. In this latter example, the turtle is expected to find the strawberry at the end of a slalom using the shortest possible representation. In order to solve the task, students need to make use the repeat statement. We consider both tasks as a testimony that teachers – when given the opportunity – do indeed come up with highly creative and cognitively-appealing tasks.

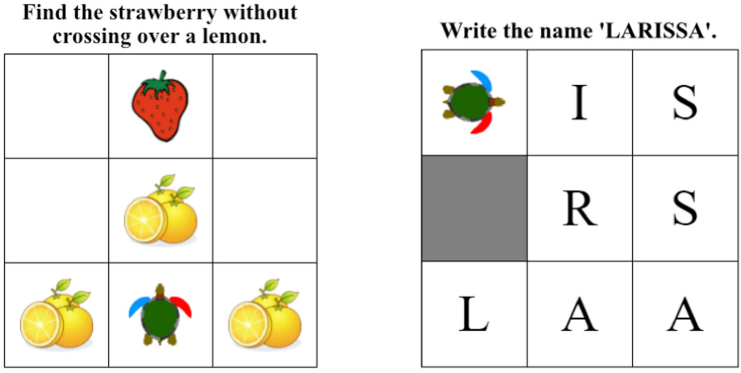


Fig. 9. Two examples of student-created tasks. Both cannot be solved correctly.

Ultimately, tasks are intended to support pupils in their learning. If we instead look at tasks created by children themselves, these are more playful and exploratory in nature. There are numerous interesting examples that show how students explore the limits of the provided rule system or otherwise try to test the feasibility of tasks. Whether intentional or not, there are several instances of tasks that have a clear formulation yet are unsolvable as given and in Fig. 9 we see two such examples. Though the attempt didn't succeed in these cases, this shift to the pupil's point of view is valuable as it gives the teacher a window

into how the pupil operates and what they think. In any case, the tool obviously provides a learning ground for pupils to explore the system they work in and expand their understanding.

4 Conclusion

Computer science has established itself as a vibrant and growing discipline in research, however, is still as-yet unformalized and unfamiliar to the broader population of educators. With many countries having recently introduced Computer Science as a new subject across all age ranges, teachers are being forced to reorient themselves and are confronted with the daunting task of creating new teaching materials, exercises, differentiation tasks, tests and more. Naturally, there are a range of books, curricula and learning environments that exist, however, these touch on a specific set of topics, present closed facts and overall leave little room for tweaks and adaptation. Precisely this adaptation is vital as it is well-known that educational reforms must be supported and directly driven by teachers to be successful. Formulated differently, this implies that teachers need the possibility to create individual learning opportunities for their students.

In this work, we presented a collection of navigation tasks where pupils direct a turtle around a 2D grid and collect various objects. Aside from demonstrating how the same basic structure permits a range of task difficulties for different ages, we formalized a technological solution that is tightly integrated within the XLogoOnline programming environment. Using the new task collection mode, teachers – or even pupils themselves – are able to playfully design, test, and share their own tasks without any background knowledge. As an added benefit, the environment includes an automatic verification mechanism that provides students with individualized feedback, allows them to progress at their own pace and reduces the burden on teachers. Because of these attributes, the tool is well suited as a platform to host in-class exams or even competitions. We have begun early exploration in this direction and plan to further develop this aspect and eventually create a full-featured programming competition platform that allows pupils to practice in an authentic context, supports the workflow of designing new problem sets, and facilitates distribution and grading of tasks.

References

1. Berry, M.: Computing in the national curriculum: a guide for primary teachers (2013)
2. Gebauer, H., Hromkovič, J., Keller, L., Kosírová, I., Serafini, G., Steffen, B.: Programmieren mit LOGO. ABZ, Ausbildungs-und Beratungszentrum für Informatikunterricht (2015)
3. Hromkovic, J.: Einführung in die Programmierung mit LOGO, vol. 206. Springer, Heidelberg (2010)
4. Hromkovič, J.: Einfach Informatik 5/6: Programmieren. Primarstufe. Begleitband. Einfach Informatik (2019)

5. Huizinga, T., Handelzalts, A., Nieveen, N., Voogt, J.M.: Teacher involvement in curriculum design: need for support to enhance teachers' design expertise. J. Curric. Stud. **46**(1), 33–57 (2014). https://doi.org/10.1080/00220272.2013.834077
6. Ioannidou, A., Repenning, A., Webb, D.C.: AgentCubes: incremental 3D end-user development. J. Vis. Lang. Comput. **20**(4), 236–251 (2009)
7. Kalas, I., Blaho, A., Moravcik, M.: Exploring control in early computing education. In: Pozdniakov, S.N., Dagienė, V. (eds.) ISSEP 2018. LNCS, vol. 11169, pp. 3–16. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02750-6_1
8. Kirk, D., MacDonald, D.: Teacher voice and ownership of curriculum change. J. Curriculum Stud. **33**(5), 551–567 (2001). https://doi.org/10.1080/00220270010016874
9. Kohn, T.: Teaching python programming to novices: addressing misconceptions and creating a development environment. Ph.D. thesis, ETH Zurich, Zürich (2017). https://doi.org/10.3929/ethz-a-010871088
10. Léonard, M.: Score et chronomètre sur l'interface: quels effets sur les résultats et la perception d'un concours en ligne? Actes des huitièmes rencontres jeunes chercheur-es en EIAH, p. 56 (2020)
11. Lokar, M.: Pišek - programming with blocks competition a new Slovenian programming competition. In: Kori, K., Laanpere, M. (eds.) Proceedings of the International Conference on Informatics in School: Situation, Evaluation and Perspectives, Tallinn, Estonia, 16–18 November 2020. CEUR Workshop Proceedings, vol. 2755, pp. 1–12. CEUR-WS.org (2020). http://ceur-ws.org/Vol-2755/paper1.pdf
12. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The Scratch programming language and environment. ACM Trans. Comput. Educ. **10**(4) (2010). https://doi.org/10.1145/1868358.1868363
13. Meerbaum-Salant, O., Armoni, M., Ben-Ari, M.: Habits of programming in Scratch. In: Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, ITiCSE 2011, pp. 168–172. Association for Computing Machinery, New York (2011). https://doi.org/10.1145/1999747.1999796
14. Staub, J.: xLogo online - a web-based programming IDE for Logo. Master's thesis, ETH Zurich, Zürich (2016). https://doi.org/10.3929/ethz-a-010725653, masterarbeit. ETH Zurich
15. Staub, J.: Programming in K-6: understanding errors and supporting autonomous learning. Ph.D. thesis, ETH Zurich, Zurich (2021). https://doi.org/10.3929/ethz-b-000491971
16. Staub, J., Barnett, M., Trachsler, N.: Programmierunterricht von Kindergarten bis zur Matura in einem Spiralcurriculum. Informatik Spektrum **42**(2), 102–111 (2019). https://doi.org/10.1007/s00287-019-01161-6
17. Szabo, C., Sheard, J., Luxton-Reilly, A., Becker, B.A., Ott, L.: Fifteen years of introductory programming in schools: a global overview of K-12 initiatives. In: Proceedings of the 19th Koli Calling International Conference on Computing Education Research, pp. 1–9 (2019)
18. Trachsler, N.: WebTigerJython - a browser-based programming IDE for education. Master's thesis, ETH Zurich, Zurich (2018). https://doi.org/10.3929/ethz-b-000338593
19. Wolber, D., Abelson, H., Spertus, E., Looney, L.: App Inventor. O'Reilly Media, Inc. (2011)
20. Wolf, M.: Reader, Come Home: The Reading Brain in a Digital World. Harper, New York (2018)