Olympiads in Informatics, 2022, Vol. 16, 55–73
© 2022 *IOI, Vilnius University*
DOI: 10.15388/ioi.2022.06

Error Handling in XLogoOnline

Jacqueline STAUB
Department of Computer Science Trier University
email: staub@uni-trier.de

Abstract. This article presents an approach to error handling with Logo novices from first to sixth grade. While structural programming errors can be mostly prevented using visual programming interfaces, logical errors must be dealt with from the very beginning. We have developed a turtle graphic task collection with an integrated solution verification to determine logical correctness of student solutions. Once learners transition from block- to text-based programming, a sizeable field of possible structural errors opens up. Thanks to static program analysis most structural programming errors can be detected while the programmer is still typing. Using a debugger, finally, programmers of all ages have the possibility to observe their program's behavior while stepping through the code. We summarize the error handling tools provided as part of the XLogoOnline programming environment and explain how students can use such aids to attain a constructive attitude towards errors.

Keywords: educational programming, K--6, turtle graphics, Logo, error diagnosis.

1. Introduction

More and more countries have recently started to include computer science (and thus also programming) in their public curricula. With this, even elementary school students now have a chance to learn how to program in various countries around the world. This political change opens up many opportunities, but also raises unresolved questions: What should programming instruction look like in kindergarten when children are not yet literate? How can a teacher provide individualized support to students when programming is known to be an error-prone activity and every child in a class is most likely struggling with errors? We want to clarify these questions by presenting a spiral curriculum for programming in K-6 as well as an approach to error handling. This article summarizes the core ideas presented in XLogoOnline (Staub, 2021; Menta *et al.*, 2019; Forster *et al.*, 2018; Staub *et al.*, 2021).

The recent introduction of computer science in public schools provides children with the opportunity to explore the exciting world of algorithms by learning to program. In order to teach basic programming concepts in an age-appropriate way, both teaching materials (Komm *et al.*, 2020; Hromkovič and Kohn, 2018; Hromkovic, 2010) and learning environments (Trachsler, 2018; Maloney *et al.*, 2010; Cooper *et al.*, 2000; Hromkovič *et al.*, 2017b; Kohn and Manaris, 2020; Repenning and Ioannidou, 2006) must be readily available. Around the world, there are millions of students that are expected to develop the ability to solve problems algorithmically by the time they enter lower secondary school. Researchers proposed generic frameworks for fostering algorithmic thinking (Dagienė *et al.*, 2021) in computer science as well as concepts with a focus on programming (Hromkovič *et al.*, 2017a; Hromkovič *et al.*, 2016).

Programming is a creative but error prone (Fitzgerald *et al.*, 2008; Gugerty and Olson, 1986) form of learning that enables teachers and students to explore and develop algorithms for a wide range of different problem classes. In essence, programming is a form of communication with a computer; for this a language is required that the computer "understands". Programming languages, much like natural languages, have a vocabulary and a grammar. In contrast to natural languages, however, programming languages are more precise and computers lack the ability to interpret ambiguous statements. As a result, programmers must take special care to express their thoughts accurately. Errors are inevitable and learning to resolve them is a core competence any programmer needs to establish.

The spectrum of programming errors ranges from simple structural errors (e.g., incorrect punctuation, missing or incorrect arguments, and unbalanced parentheses) to more complex logical errors (e.g., reversed loop conditions, forgotten invariants). While both of these error classes are the bread and butter of any programmer regardless of age and experience, there are some error classes that are specific to children. Programming is possible from as young as six or seven years; at that age novices are able to understand basic programming concepts and anticipate program logic (Ettinger, 2012), but they have difficulty expressing themselves in a written language (Solomon, 1993). In order to prevent structural issues due to typing, block-based programming interfaces have been developed and are used in various environments nowadays (Weintrop, 2019).

It is our firm belief that error handling is one of the most essential skills a young programmer needs to acquire by the end of their programming education. Programming environments therefore need to be equipped with error diagnosis tools that are dedicated to the use in programming classes to handle logical errors from the very beginning and later on structural errors in addition. In this work, we summarize the tools we employed in our programming environment XLogoOnline. The environment allows students to begin programming without literacy skills, and provides useful tools for finding, analyzing, and fixing programming errors throughout programming instruction from kindergarten to sixth grade.

Section 2 provides a overview on the linguistic features of the famous programming language Logo and its decade-old traditions yet without diving into the application domain turtle graphics. More detail on our curriculum, the turtle philosophy, and the concrete implementation of these ideas in XLogoOnline follow in Section 3. Finally, Section 4 and Section 5 discuss the concrete details of how error handling is managed in XLogoOnline before concluding.

2. The Logo Programming Language

More than 50 years ago, there first emerged the idea of creating dedicated programming languages that could be used in an educational context with children and adolescents. Although computers were anything but a commodity at that time and the few available models were mostly reserved for universities, thanks to the initiative of Seymour Papert and his team (Papert, 1980; Solomon *et al.*, 2020), school kids as young as secondary or even primary school were able to enjoy the pleasure of programming. Papert and colleagues built the foundation and revolutionized the field of programming education by designing a programming language that specifically targeted to the needs of novices. The resulting language, Logo, is distinguished by its exceptionally minimal and elegant syntax, which (despite its age) still stands out today due to three unique attributes:

- **Whitespace as statement delimiter:** Instead of classical statement delimiters like semicolons or line breaks (as known from Java or Python respectively), Logo allows any number of statements to be placed side by side without an additional delimiter other than a bare space. Three procedure calls `foo`, `bar` and `baz` can thus be simply concatenated without requiring any additional characters in between: `foo bar baz`
- **Whitespace as argument delimiter and no brackets:** Like many other programming languages, Logo supports parameters for procedures. How many parameters a procedure can take depends on its specification; from the linguistic point of view any number of arguments are possible. Moreover, while arguments in other programming languages usually are surrounded by parentheses and require to be separated by a dedicated syntactic character, say a comma, Logo allows a simple and elegant alternative – no parentheses required and arguments are separated by a single white space: `mod 4 2`
- **Deliberate reduction to the minimum:** Cognitive load in programming is reflected (among others) by the number and complexity of the programming concepts used (Hromkovič *et al.*, 2017b). Instead of overburdening students with a multitude of different programming constructs in one go, the Logo philosophy proposes to construct their own more complex language elements. In a truly constructivist manner, the programming language "grows" together with the programmer's proficiency level.

In addition to these purely linguistic aspects, Logo is also famous for its world-renowned application domain Turtle Graphics. The concept of the Turtle as well as a corresponding curriculum for K–6 are presented in the following section.

3. A Spiracl Curriculum for Programming Classes in K-6

Turtle Graphics has stood the test of time and proved a valuable way of introducing beginners to programming. The principle is based on the visualization of the program execution by means of a virtual or physical computing agent, i.e., the "turtle". This turtle is, in essence, an object whose position and orientation in space can be changed programmatically. Students understand the turtle as a tangible representation of the abstract executions mechanism used in a computer. In order to control its behavior, students need to learn the turtle's "mother tongue" Logo which initially consists of four simple commands:

- **`Forward`:** The turtle moves straight ahead [by a given number of pixels].
- **`Back`:** The turtle moves backwards [by a given number of pixels].
- **`Right`:** The turtle turns to the right [a given angle].
- **`Left`:** The turtle turns to the left [a given angle].

With these four basic commands it is possible to solve simple navigation tasks of the form "guide the turtle from A to B without visiting C along the way" (as illustrated in Fig. 1) to more complex geometric tasks (as shown in Fig. 2). All of these tasks could be posed both in a block-based and text-based interface. One aspect that distinguishes navigation tasks from simple geometry tasks is that for pure navigation in a grid no parameters are required (i.e., unit distances and angles can be used) whereas geometry tasks profit from parameterized basic commands.

Our approach proposes a spiral approach for programming instruction from kindergarten to grade six. Notable milestones in the intended learning progress can be summarized in three stages:

1. **Stage 1 (kindergarten to 2nd grade):** In the youngest age group, children work in a block-based interface with basic commands that do not include parameters (i.e., the `forward` and `back` movement commands cause motion at unit distances while the `right` and `left` rotation commands only perform 90 degree turns). In this framework, learners immerse themselves in the following task types: (i) building sequences of basic commands, (ii) creat-

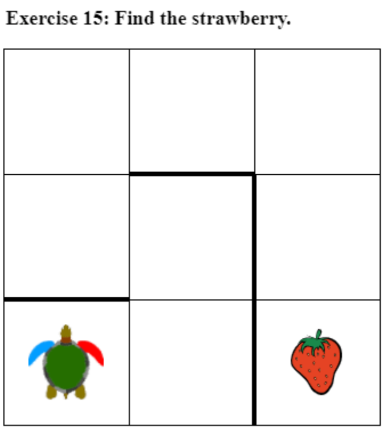


Fig. 1. Navigation task.

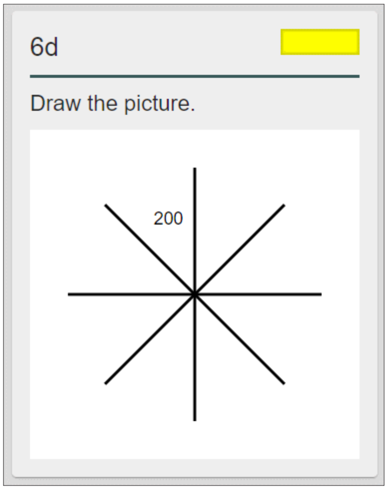


Fig. 2. Geometry task.

ing programs under constraints, (iii) working with colors, (iv) covering longer distances with `repeat`, (v) shortening repetitive program sequences with `re-peat`.

2. **Stage 2 (3rd and 4th grade):** After the first stage, students transition from the previous navigation-based tasks to the more traditional geometry tasks. For this purpose, the basic commands `forward`, `back`, `right` and `left` are extended with parameters (i.e. the two movement commands allow to draw lines of arbitrary length and the rotation commands can cause rotations of arbitrary angles), while the interface stays block-based. In terms of content, this second stage focuses on the following concepts: (i) building sequences of basic commands, (ii) creating programs under restrictions, (iii) working with colors, (iv) shortening repetitive program sequences using `repeat`, (v) sequences of `repeat`, (vi) nested `repeat`.
3. **Stage 3 (5th and 6th grade):** Finally, the transition from block-based to text-based programming takes place. While the Turtle Graphics application area remains the same, this step mainly changes the input form and the depth of the concepts covered in the curriculum. Students engage in the following types of tasks: (i) they form longer sequences of basic instructions, (ii) they shorten repetitive program sequences using `repeat`, (iii) they define their own procedures, (iv) and use these procedures as subroutines, (v) they parameterize their own procedures, (vi) they define and use their own parameterized subroutines.

More information about the currciulum and its specific contents are provided in REF (Hromkovic, 2010).

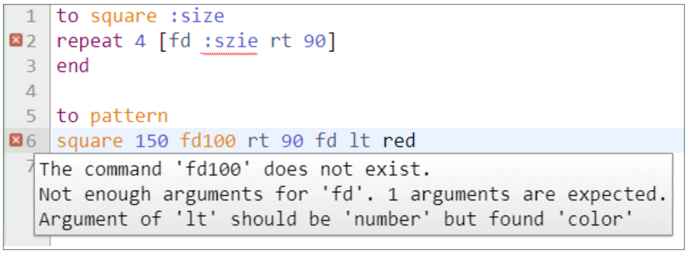
The following section presents the programming environment XLogoOnline and its approach to error handling dedicated to the above curriculum.

4. Error Handling in XLogoOnline

The XLogoOnline programming environment aims at students' autonomous error recovery by providing assistance in three domains: (i) the environment proactively diagnoses structural errors in text-based Logo programs, (ii) it automatically detects logical errors thanks to a task specification system with integrated solution verification, and (iii) it provides a debugger for students to investigate logical errors on their own.

4.1. *Reporting Structural Errors*

We refer to structural errors as any error that causes the execution pipeline to terminate unexpectedly; be it syntactic errors (which are already apparent during the construction of the parse tree), semantic errors (such as naming errors, missing or redundant arguments. Note that these programs parse legitimately but then fail during interpretation), or more general runtime errors (e.g., type errors, index errors, or other problems that are detected at runtime).



```
to square :size
repeat 4 [fd :szie rt 90]
end

to pattern
square 150 fd100 rt 90 fd lt red
```

```
The command 'fd100' does not exist.
Not enough arguments for 'fd'. 1 arguments are expected.
Argument of 'lt' should be 'number' but found 'color'
```

Fig. 3. An example of how XLogoOnline visualizes errors in the environment. All of these cases can be detected statically.

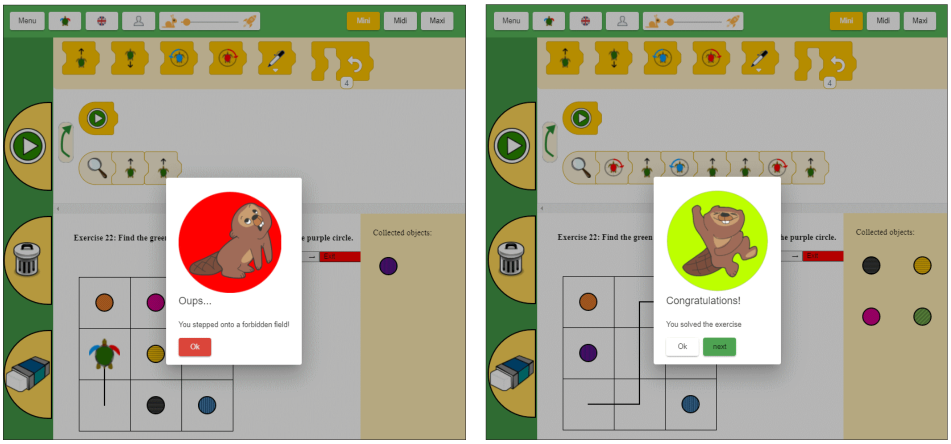
XLogoOnline follows the philosophy of reporting structural errors proactively, that is as early on as possible. For this purpose, the program text is continuously parsed on every keystroke in order to immediately detect syntactical errors in the parse tree. Moreover, the environment tries to turn as many runtime errors as possible into static errors that can be detected before the program is executed. This can be achieved using static program analysis and combined with the check for syntactic errors.

All detected errors are localized in the source code (i.e., the respective token stream) in order to find the exact position in the program text, see Fig. 3. Afterwards, the corresponding text is visually highlighted in the editor and a corresponding error message is added. In the formulation of error messages we make sure that the language is understandable (that is, we use few words that are written in the language learners are acquainted with from the curriculum) but we also take care of formulating error messages consistent throughout all cases.

4.2. *Detecting Logical Errors in Turtle Graphics*

Logical errors, unlike structural errors, do not cause the execution pipeline to fail, but rather produce unexpected results. Such errors can arise in any contexts and must be handled differently than structural programming errors. In fact, what may look like an error in one context may be intentional in another. That is, without insight into the specific objectives of a given program, it is not possible to discern correct from incorrect solutions.

In order to still automatically detect logical errors, XLogoOnline provides predefined tasks with exact specifications of permissible solutions (Staub *et al.*, 2021) (user manual provided in the Appendix). Several grid cells can be connected in a navigation task in which one or more *target cells* are to be visited in any or a predefined order. Additionally, certain cells can be forbidden and also the commands available in a solution can be restricted. Moreover, even the available command set can be restricted by a task. A formal specification allows to discern correct solutions from incorrect ones (see Fig. 4).



Exercise 22: Find the green … the purple circle.

Oups...

You stepped onto a forbidden field!

Congratulations!

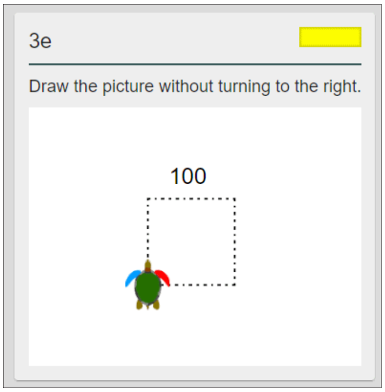
You solved the exercise

Fig. 4. Two examples illustrating how XLogoOnline automatically detects correct and incorrect solutions in grid-based navigation tasks.

4.3. *Resolving Logical Errors in Logo*

Although it is possible to detect logical errors automatically if the objective and the specification of an admissible solution are known, automatically locating logical errors is neither easy nor didactically desired. The problems used in our curriculum deliberately allow more than one solution. For example, Fig. 5 shows a problem in which a given picture is to be drawn without making right turns. In our experience, this task elicits multiple different solution strategies (e.g., Fig. 6), that all adhere to the given condition and solve the problem. This flexibility should be maintained in order to encourage the exchange of ideas within the class and the comparison of different strategies.

Due to the numerous possible correct solutions and individual ways of thinking, we do not intend to solve the localization of logical errors automatically. By making



Draw the picture without turning to the right.

Fig. 5. Sample task.

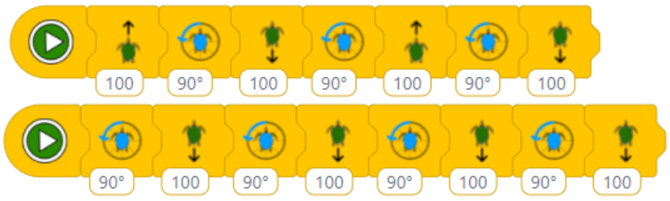


Fig. 6. Two solutions that are equivalently valid.

students solve logical errors independently, they can gain insights that would otherwise be denied to them. In this sense the last tool presented here pursues a different objective than the previous two we present a debugger that enables programmers to fix logical errors on their own.

XLogoOnline provides a reverse debugger, which allows to manually analyze erroneous programs one step after another. At the push of a button, an instruction is executed, allowing the user to compare his or her mental image of program execution with reality. Based on this experience, novices can draw conclusions about the location and the nature of an underlying logical error. Using a simple stack, the program state can be stored in each step allowing previous stages to be reached easily and enabling the course of an error to be replayed as often as desired.

5. Conclusion

For several decades, our community has been using block-based learning environments to provide beginners with a smooth start into programming. There are various reasons why block-based environments are useful: some (like ourselves) see a potential to reach young children who would otherwise struggle with writing. Others, meanwhile, consider structural programming errors a threat for all programmers, independent of their age and experience. Consequentially, opinions also diverge on the question when to switch from block- to text-based programming. Some suggest that blocks should be used well into tertiary education, while we argue that there is no need to stick with block-based environments for so long. We showed an approach of error handling that is employed in the XLogoOnline programming environment and which encourages the autonomous handling of programming errors; both logical and structural ones.

Various text-based learning environments for novices have a *reactive* approach to handle errors. That is, they report errors only during runtime. This decision causes a long and oftentimes frustrating process to start once execution begins: for each fixed structural error, programmers need to re-execute their code, possibly receiving yet another red flag which needs to be fixed before starting all over again. We argue that a proactive approach to error handling can help students skip over this tedious phase more quickly. Our approach allows structural errors to be located and reported at compile time which may lead to a majority of all structural programming errors to be detected and potentially resolved before execution even starts.

References

Cooper, S., Dann, W., and Pausch, R. (2000). Alice: a 3-d tool for introductory programming concepts. *Journal of computing sciences in colleges*, 15(5), 107–116.

Dagienė, V., Hromkovic, J., and Lacher, R. (2021). Designing informatics curriculum for k-12 education: From concepts to implementations. *Informatics in Education*, 20(3), 333–360.

Ettinger, A.B. (2012). Programming robots in kindergarten to express identity. *Industrial Engineering*, 2012.

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., and Zander, C. (2008). Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, 18(2), 93–116.

Forster, M., Hauser, U., Serafini, G., and Staub, J. (2018). Autonomous recovery from programming errors made by primary school children. In: Sergei N. Pozdniakov and Valentina Dagienė, editors, *Informatics in Schools. Fundamentals of Computer Science and Software Engineering*, volume 11169, pages 17–29, S.l. Springer. 11th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2018; Conference Location: St. Petersburg, Russia; Conference Date: October 10–12, 2018.

Gugerty, L. and Olson G. (apr 1986). Debugging by skilled and novice programmers. *SIGCHI Bull.*, 17(4), 171–174.

Hromkovic, J. (2010). *Einführung in die Programmierung mit LOGO*, volume 206. Springer.

Hromkovič, J. and Kohn, T. (2018). Einfach informatik 7–9: Programmieren. sekundarstufe i. begleitband. *Einfach Informatik*.

Hromkovič, J., Kohn, T., Komm, D., and Serafini, G. (2016). Examples of algorithmic thinking in programming education. *Olympiads in Informatics*, 10(1–2), 111–124.

Hromkovič, J., Kohn, T., Komm, D., Serafini, G., *et al*. (2017a). Algorithmic thinking from the start. *Bulletin of EATCS*, 1(121).

Hromkovič, J., Serafini, G., and Staub, J. (2017b). Xlogoonline: a single-page, browser-based programming environment for schools aiming at reducing cognitive load on pupils. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, pages 219–231. Springer.

Kohn, T. and Manaris, B. (2020). Tell me what's wrong: a python ide with error messages. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 1054–1060.

Komm, D., Hauser, U., Matter, B., Staub, J., and Trachsler, N. (2020). Computational thinking in small packages. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, pages 170–181. Springer.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4), 1–15.

Menta, R., Pedrocchi, S., Staub, J., and Dominic Weibel, D. (2019). Implementing a reverse debugger for logo. In: Sergei N. Pozdniakov and Valentina Dagiene,˙editors, *Informatics in Schools. New Ideas in School Informatics*, volume 11913, pages 107–119, Cham, 2019. Springer. 12th International Conference on Informatics in Schools: Situation, Evolution and Perspectives (ISSEP 2019); Conference Location: Larnaca, Cyprus; Conference Date: November 18–20.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., USA.

Repenning, A. and Ioannidou, A. (2006). Agentcubes: Raising the ceiling of end-user development in education through incremental 3d. In: *Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE, p.p. 27–34.

Solomon, C., Harvey, B., Kahn, K., Lieberman, H., Miller, M.L., Minsky, M., Papert, A., and Silverman, B. (2020). History of logo. *Proceedings of the ACM on Programming Languages*, 4(HOPL), 1–66.

Solomon, P. (1993). Children's information retrieval behavior: A case analysis of an opac. *J. Am. Soc. Inf. Sci.*, 44, 245–264.

Staub, J. (2021). *Programming in K–6: Understanding Errors and Supporting Autonomous Learning*. PhD thesis, ETH Zurich, Zurich.

Staub, J., Chothia, Z., Schrempp, L., and Wacker, P. (2021). Encouraging task creation among programming teachers in primary schools. In: *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, pages 135–146. Springer.

Trachsler, N. (2018). Webtigerjython-a browser-based programming ide for education. Master's thesis, ETH Zurich.

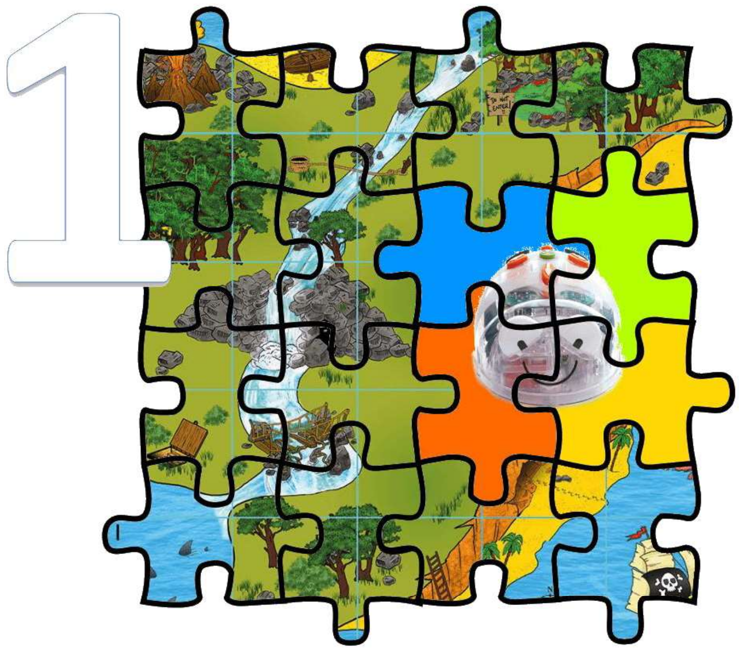
Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, 62(8), 22–25.

J. Staub holds the chair of Computer Science and its Didactics at the University of Trier. She studied computer science at ETH Zurich and completed the teaching degree for high school level at the same university. Starting in 2016, she conceived the programming environment XLogoOnline as part of her doctoral studies, which is designed as a tool to explore errors in programming education with novices and to enable the teaching of computer science concepts in a spiral curriculum from kindergarten to high school. XLogoOnline is a learning platform that is currently offered in seven different languages and is now used in schools around the world. Prior to her appointment at the University of Trier, she was a postdoctoral researcher at the Ausbildungs- und Beratungszentrum für Informatikunterricht at ETH Zurich and worked as a lecturer at the University of Teacher Education Graubünden (PHGR). As part of her work at the University of Trier, she promotes teacher education in computer science in the Rhineland-Palatinate area, continues the development of the programming environment XLogoOnline, and uses it to study error handling among programming novices.

Appendix

Exercise Creation in XLogoOnline & LogoOlympia

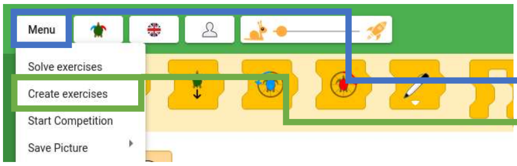
User Manual



This Document serves as a reference guid for educators using XLogoOnline Mini or the built-in competition mode LogoOlympia

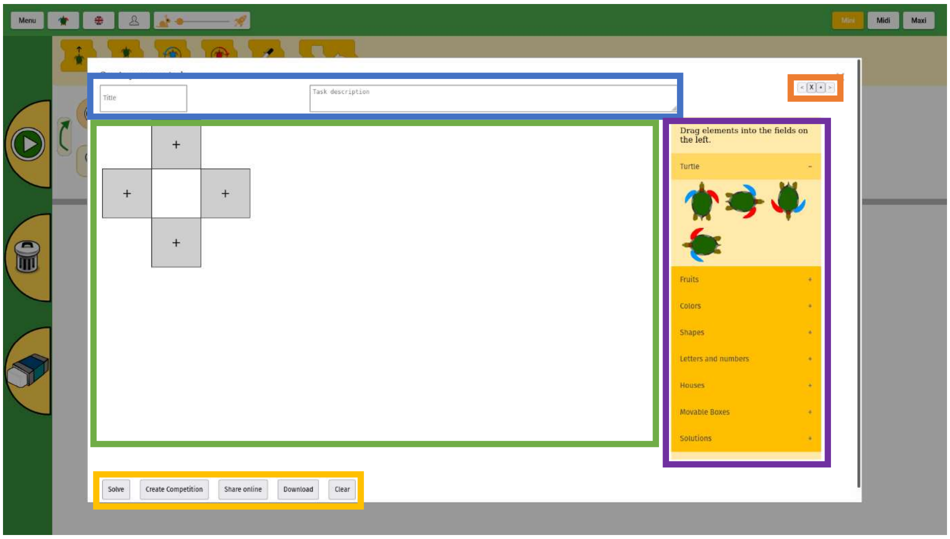
Contents

1. Create Exercises



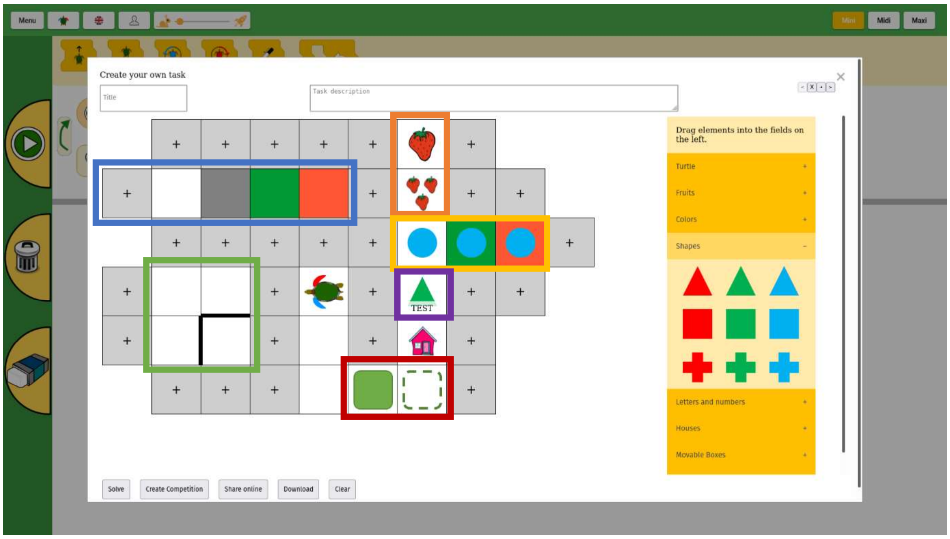
To create a new exercise (or edit a previously loaded one) you first need to open the Menu and then click the "Create exercises" button

This brings up the "Teacher-Tool Dialog".



- The 🟦-area holds the exercise title and description.
- The 🟩-area represents the grid, on which exercises can be solved.
- The 🟧-area allows to add and remove exercises as well as to switch between different exercises.
- The 🟨-area presents the options on how to access or share an exercise. A detailed explanation can be found in chapter 3.
- The 🟪-area holds the turtle as well as other objects that can be placed on the grid. Furthermore, validations can be configured here, we will explain them in more detail in chapter 2.

2. Validate Exercises



- The tiles in the 🟦-area from left to right are:
  1. new tile [+].
  2. default tile [white].
  3. forbidden tile (visible to the student) [grey].
  4. target tile [green].
  5. forbidden tile (not visible to the student) [red].

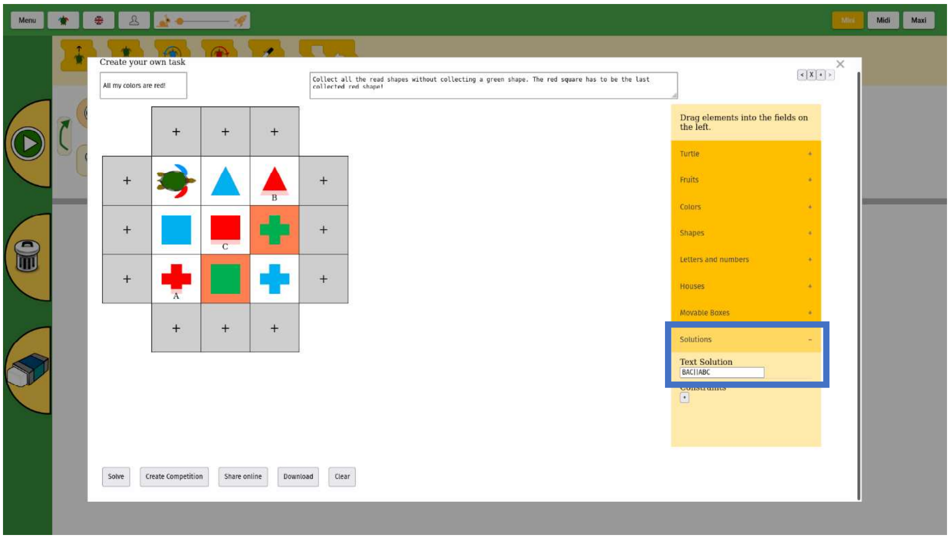
  Stepping on a forbidden tile, results in a failure state, while stepping on a target tile, results in a success state.
  By clicking on a tile, you can cycle through the five states a tile can be in.

- In the 🟩-area two "walls" are set up on the top and left of the bottom right tile. If the turtle walks through a wall (in any direction).
- The 🟨-area holds 3 times the "blue color" object. From left to right, they were placed on a default tile (2), the second one on a target tile (4) and the last on a forbidden tile (5).
- The object in the 🟪-area has a value assigned to it. By default, every object has a value of 1, by clicking on a tile, you can assign a specific value to it. As soon as any object has a value assigned (that is not a number) all other objects will have the default value "" (empty string).
- The 🟧-area holds multiple instances of the strawberry object. The strawberry object is special, as it has a default value equal to the number of strawberries depicted on

the object. The strawberries range from one to four strawberries and thus a default value between one and four.

- The 🟥-area holds a moveable box (the filled-out object on the left) and a target box (the right box with the dashed outline). The moveable box can be pushed by the turtle and the goal is to push a moveable box on every target box. Once this is achieved, a success state is reached and the constraint get checked.

2.1. Solution



All my colors are red!

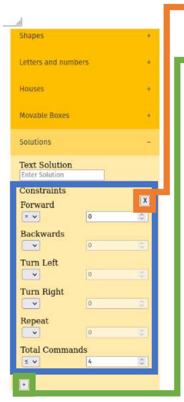
Collect all the read shapes without collecting a green shape. The red square has to be the last collected red shape!

BAC||ABC

In the “Solution” tab, in the “Text Solution” input field, a number can be entered, which then will work as a sum. The values of all the collected objects, will be summed up and compared to the target number. If the sum of all collected objects matches the solution, a success state is reached and the constraints are checked.

If instead of a number, a string is entered into the text solution field, the values of the collected objects will be concatenated and compared to the given solution. If they match a success state is reached and the constraints are checked. In this mode, multiple possible solutions can be separated by “||”. In the given example above, both “BAC” as well as “ABC” would be accepted as solutions. Another option would be to make the solution “AAC” and give both the red cross and the red triangle the value “A”.

2.2. Constraints



Text Solution
Enter Solution
Constraints
Forward = 0
Backwards 0
Turn Left 0
Turn Right 0
Repeat 0
Total Commands ≤ 4

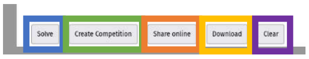
The [X] button can be used to remove a constraint set.

With the [+] button a new constraint set can be added. If a given program passes any constraint set, it will be considered valid by the system.

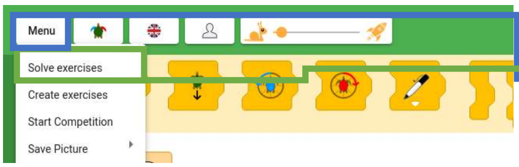
Constraints can be used to limit or ban the use of certain command or all of them. The options for a single constraint are “less than”, “less or equal than”, “equal”, “not equal”, “more or equal than” or “more than”. To ban the use of a command, you can set the constraint type to “equal” and the amount to zero, as in the example has been done to the “Forward” command. To limit the use of commands altogether, a constraint can be put on “Total Commands”. In the example in total up to 4 commands can be used, but none of them can be the forward command.

3. Store and Load Exercises

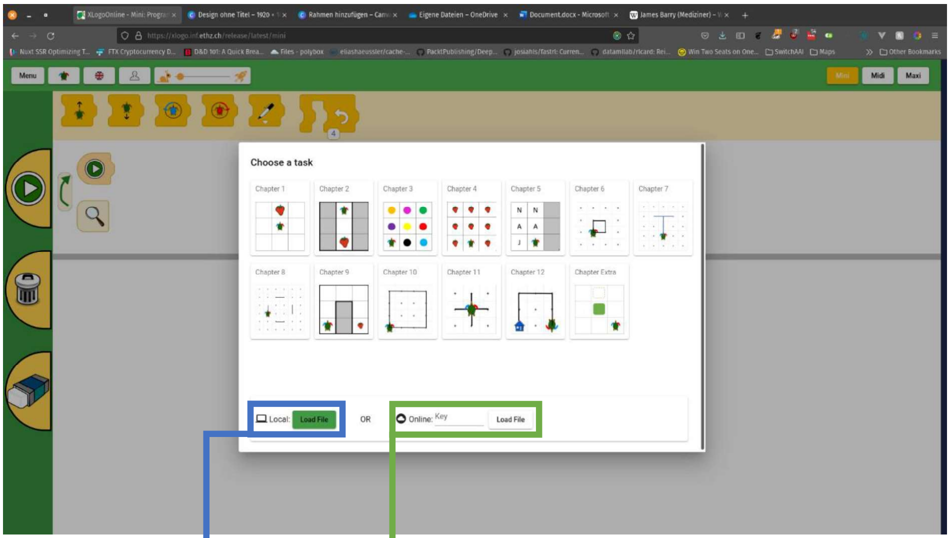
Storing exercises can be done from the teacher tool, with the buttons in the bottom left.



- The 🟦-button loads the current exercises in the exercise mode to be solved in the browser. By opening the teacher tool again, the exercises can be modified further and/or saved.
- The 🟩-button creates a competition of all exercises and presents you with a competition- and storage-key. The storage key is needed to load the exercises in the future for further modification, while the competition key is needed by participants to enter a competition.
- The 🟧-button will save the exercises to our server and present you with a storage key, you can use in the future to retrieve the exercises again.
- The 🟨-button can be used to download the current exercises to your computer. They can be uploaded to XLogoOnline on a later date.
- The 🟪-button lets you reset the current exercise if you have created multiple exercises the others will not be affected.

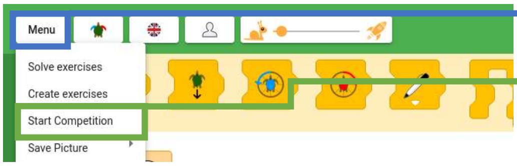


To load an exercise you have to open the "Solve exercises" dialog from the menu

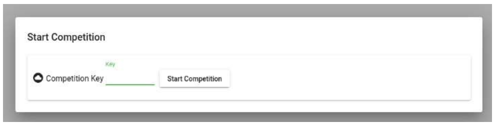


You can either upload one or saved files or load them from the server by specifying one or more storage keys separated by commas.

4. Competition



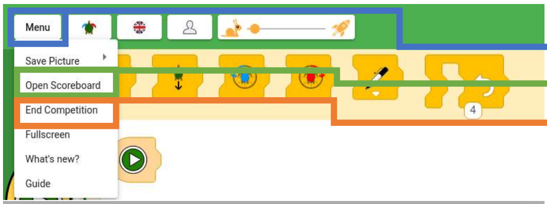
To participate in a competition you need to open the menu and select "Start Competition".


Start Competition

Competition Key — Key

Next you need to enter the competition key.

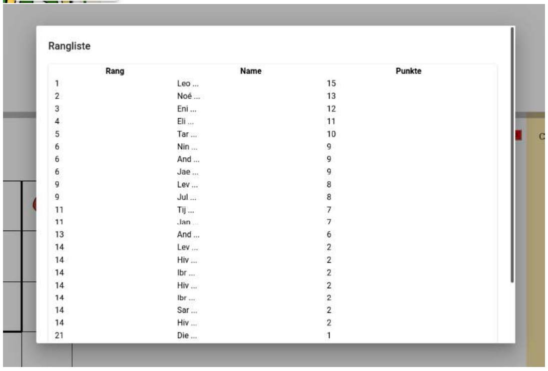
4.1. Score Board and End Competition



When you are entered in a competition the menu contains an "Open Scoreboard" as well as "End Competition" option.

The "End Competition" button will end the competition for the participant.

The "Open Scoreboard" button will pull up the current competitions score board


Rangliste

Rang	Name	Punkte
1	Leo ...	15
2	Noé ...	13
3	Eni ...	12
4	Eli ...	11
5	Tar ...	10
6	Nin ...	9
6	And ...	9
6	Jae ...	9
9	Lev ...	8
9	Jul ...	8
11	Tij ...	7
11	Jan ...	7
13	And ...	6
14	Lev ...	2
14	Hiv ...	2
14	Ibr ...	2
14	Hiv ...	2
14	Ibr ...	2
14	Sar ...	2
14	Hiv ...	2
21	Die ...	1