Informatics in Education, 2024, Vol. 23, No. 4, 767–781
© 2024 *Vilnius University*
DOI: https://doi.org/10.15388/infedu.2024.26

Spark: The First Choice for Novices

Judith GAL-EZER[1], Smadar SZEKELY[2]
[1] *The Open University of Israel, Israel*
[2] *Weizmann Institute of Science, Rehovot, Israel*
e-mail: galezer@openu.ac.il, smadarsz@gmail.com

This paper is written in tribute to Niklaus Wirth.

Abstract. Spark, one of the products offered by MyQ (formerly Plethora), is a game-based platform meticulously designed to introduce students to the foundational concepts of computer science. By navigating through logical challenges, users delve into topics like abstraction, loops, and graph patterns. Setting itself apart from its counterparts, Spark boasts an innovative formal language and a rich set of features. Unlike traditional platforms, Spark emphasizes computational problem solving over programming syntax, making it accessible to learners of all levels. With progressively challenging levels and an intuitive graphical interface, students engage in problem solving, content creation, and collaboration within the MyQ community. Using Spark makes it less probable for students to utilize generative AI (GAI) to solve challenges, thereby sparing teachers the struggle of assessing tasks that might have been accomplished using GAI.

In this paper, we provide an examination of Spark, its functionalities, the challenges it tackles, its merits, limitations, and prospective trajectories.

Key words: computer science education, programming, first choice language, algorithmic thinking, computational problem solving.

1. Introduction

At the beginning of the 90s of the previous century, a new high school computer science curriculum was about to be designed in Israel. It was very clear that this curriculum should include the foundations of the discipline, long-standing concepts, and that theoretical and implementation issues should be interwoven like a zipper (Gal-Ezer *et al.*, 1995). The curriculum should not be dependent on fast-evolving technology. Solving algorithmic problems was one of the subjects included in the curriculum. Algorithms must be implemented by a programming language; one of the big challenges was what language to choose. The decision was to opt for *Pascal*, the brainchild of Niklaus Wirth, which became the preferred language for CS1 courses in numerous academic institutions.

The discussion of what should be the mother tongue of programming languages was seen all over, in conferences, professional journals, and among faculties. Groups of educators and researchers believed in their choice of language, and the debates took on the character of a religious war. Pascal follows an imperative paradigm and is procedural programming language. In that time other procedural programming languages and other paradigms were out there, for example, the logic paradigm (PROLOG). But it seemed that Pascal was the language designed especially for teaching and learning the very first steps of programming.

Since then, many other paradigms were designed and used, the object-oriented paradigm was one adopted in many curricula worldwide. But there was also the recognition that the language is not the main issue, algorithmic problem solving is the issue, and curricula designers let go, and let the schools, universities, educators, and teachers choose what they felt comfortable with.

While it was initially difficult to convince policymakers, school masters, and others that it is crucial to integrate computer science into the school system, the need to do so has finally been recognized worldwide. And not only in high school but on all school levels. Many reports have been written on this, too numerous to mention here, but here are some examples: CSTA K–12 Standards (CSTA, 2017), CECE report (CECE, 2017), or the Informatics Reference Framework for School (Caspersen *et al.*, 2022).

Visual environments such as *Alice* (Cooper *et al.*, 1995), block-based paradigms such as *Scratch* (Maloney *et al.*, 2008), and various other paradigms and platforms became popular for novices (Fincher and Petre, 2004). One of the game-based platforms is *Spark*, designed by MyQ in collaboration with computer scientists from the Weizmann Institute of Science, and pedagogical experts from the Israeli Center for Educational Technology. In this paper, we will dwell on the features of this platform, its uniqueness, capabilities, advantages, and disadvantages.

One comment is in place here, the motivation to integrate computer science into the school system must be accompanied by recruiting teachers to teach the material. Teachers certified to teach computer science, similarly to teachers certified to teach Mathematics or Physics, should have the knowledge and pedagogy skills required to be able to teach. The teachers' issue has also been discussed worldwide, in conferences and professional journals (Hazzan *et al.*, 2010; Gal-Ezer and Stephenson, 2010). Spark, as we will see, also assists teachers in class, especially for lower levels (elementary school) where usually they are not special subject teachers, but rather are educators and general teachers.

2. Spark

As mentioned above, Spark is a game-based platform designed in collaboration with computer scientists from the Weizmann Institute of Science, and pedagogical experts from the Israeli Center of Educational Technology.

It was designed to serve students starting in fourth grade and beyond, helping them develop computational problem solving abilities without focusing on programming (Armoni *et al.*, 2021, 2024). Students are not expected to code or even read conventional programs. One of the most interesting underlying principles of Spark is to help understand complex systems by changing a given system's rules of behavior and seeing the effect of this by viewing the resulting system in action.

Fundamental concepts of computer science are easily introduced with Spark, from easy-to-grasp concepts, such as cause and effect, user interactions, and sequentiality, to more advanced topics, such as abstraction, graph patterns, real-time, recursion, and even parallelism, which is not only explicitly explained, but is also an integral part of the platform.

Spark consists of multiple, increasingly difficult game challenges, and an editor with which players can create their own challenges and share them with the MyQ community. The language used is visual and designed to be intuitive. Yet, it is accompanied with text in the learners' native language, making it easy to read and understand. The supported languages include: Hebrew, English, Arabic, Italian, French, Spanish, Chinese, and more. Finally, the Spark package includes a dashboard for teachers and provides lesson plans with guidance for educators, as well as unplugged activities and real-life examples.

2.1. *Spark and Scenario-based Programming*

Spark is built upon the relatively new programming paradigm of scenario-based programming, which was launched with the advent of the language of *Live Sequence Charts (LSC)* (Damm and Harel, 2001; Harel and Marelly, 2003; Harel *et al.*, 2010). In the scenario-based approach, a program consists of a set of multi-modal scenarios. The execution mechanism follows all scenarios simultaneously, adhering to them all so that any run of the program is legal with respect to the entire set of scenarios. Most scenarios are mandatory ones, whereby if a certain sequence of one or more events occurs, then another sequence must follow it.

The LSC language was developed for intuitively specifying and simulating complex systems' behavior without the need for actually coding the system. This approach and its supporting tools introduced an extensive and unique way of specifying requirements, analogous to a human's way of thinking when defining scenarios of a system's behavior. After more than a decade of experience with the LSC language and tools, Spark was developed with the idea of utilizing a simple and intuitive, yet expressive, visual language, adapted for young students, to allow them practice computational problem solving skills, by describing the behavior of abstract systems based on geometric shapes (Gal-Ezer *et al.*, 2020).

2.2. *The Spark Experience*

Spark is composed of several components:

- *Subjects Screen:* This screen serves as a gateway to the challenges and displays the set of topics covered by Spark (see Figure 1). Through this screen, students gain explicit awareness of the concepts they engage with and learn.
- *The Main Game Screen:* This is the main area where the activity takes place. Here the students see and solve the challenges. We elaborate on the challenges later in this section.
- *Spark Studio:* With Spark Studio, the students invent their own challenges and can share them with friends. They learn that creating a software product is an iterative process composed of defining an initial state, goal, and system behavior, and eventually testing and validating that the system does what they intended it to do.

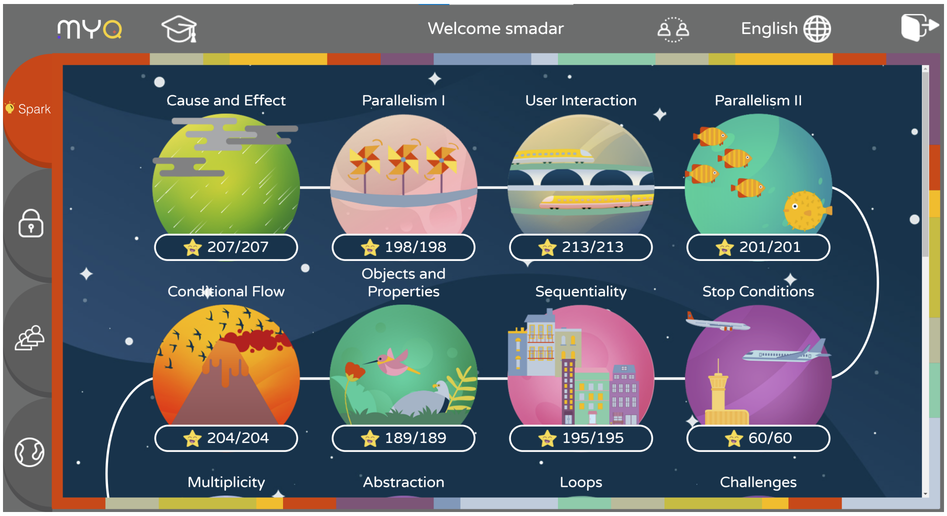


Fig. 1. Spark subjects screen

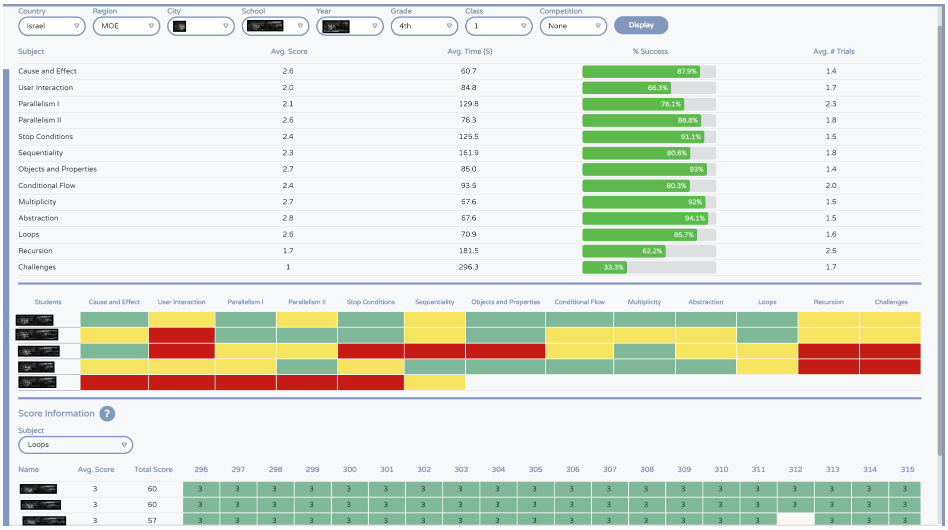
- *Community:* Students can create and share challenges as well as browse and solve challenges defined and shared by other students in the MyQ community.
- *Dashboard for Teachers:* The Spark platform includes a dashboard for teachers to help them track and assess their students' progress.

The dashboard, as can be seen in Figure 2, is divided into 2 main parts: at the top part of the screen information about the progress of the class per subject is shown, and at the bottom part of the screen detailed information per student. Moreover, the dashboard is capable of consolidating and displaying data across all classes within a specific grade level or within the entire educational institution. This is beneficial specifically for academic coordinators and school administrators.

With Spark, students engage in solving challenges and creating new ones themselves. These challenges are organized by topics representing algorithmic concepts and include levels of varying difficulty.

Each challenge is a small system composed of an initial state, a goal state, a set of partially specified activation rules, and a halting rule. Students are tasked with completing these partially specified rules so that, when executed, they cause the system to behave correctly and achieve the goal state. The rules themselves are depicted visually using simple and intuitive icons, independent of conventional programming language syntax, and are complemented by text in the student's native language. This design choice facilitates an understanding of computational problem solving concepts without the need to engage directly with traditional programming.

In Figure 3 we present a simple-to-solve challenge from the advanced subject parallelism. This challenge starts with one yellow triangle and one purple star on the game screen, as can be seen in the game screen and as declared on the top left part of the screen. The mission that the student has to accomplish is completing the missing cards in the rule, so



Country	Region	City	School	Year	Grade	Class	Competition
Israel	MOE	[redacted]	[redacted]	[redacted]	4th	1	None

Display

Subject	Avg. Score	Avg. Time (S)	% Success	Avg. # Trials
Cause and Effect	2.6	60.7	87.9%	1.4
User Interaction	2.0	84.8	66.3%	1.7
Parallelism I	2.1	129.8	76.1%	2.3
Parallelism II	2.6	78.3	88.8%	1.8
Stop Conditions	2.4	125.5	91.1%	1.5
Sequentiality	2.3	161.9	80.6%	1.8
Objects and Properties	2.7	85.0	93%	1.4
Conditional Flow	2.4	93.5	80.3%	2.0
Multiplicity	2.7	67.6	92%	1.5
Abstraction	2.8	67.6	94.1%	1.5
Loops	2.6	70.9	85.7%	1.6
Recursion	1.7	181.5	62.2%	2.5
Challenges	1	296.3	33.3%	1.7

Students	Cause and Effect	User Interaction	Parallelism I	Parallelism II	Stop Conditions	Sequentiality	Objects and Properties	Conditional Flow	Multiplicity	Abstraction	Loops	Recursion	Challenges
[redacted]	🟩	🟨	🟩	🟨	🟩	🟨	🟩	🟩	🟩	🟩	🟩	🟨	🟨
[redacted]	🟨	🟥	🟩	🟩	🟩	🟨	🟩	🟨	🟨	🟨	🟩	🟨	🟨
[redacted]	🟩	🟥	🟨	🟨	🟥	🟥	🟥	🟨	🟩	🟨	🟨	🟥	🟥
[redacted]	🟨	🟨	🟨	🟩	🟨	🟩	🟩	🟩	🟩	🟩	🟨	🟥	🟥
[redacted]	🟥	🟥	🟥	🟥	🟥	🟨							

Score Information ?

Subject: Loops

Name	Avg. Score	Total Score	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315
[redacted]	3	60	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
[redacted]	3	60	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
[redacted]	3	57	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3		3	3	3

Fig. 2. Spark dashboard for teachers

that there are three yellow triangles and three purple stars on the game screen, as specified on the top right part of the screen.

The students complete the rules by dragging the cards from the cards' repository, placed on the bottom left part of the screen. Whether the completion of rules is correct or not, the rules will become active and are executed by the game engine. The results will be played on the game screen.

If the rules are completed correctly and the mission is accomplished, the students receive a 'Well Done' notification and can move on to the next challenge. In the case that the rules were completed in a way that does not achieve the goal, they still execute, letting the student observe the results. After a timeout, the students receive a notification suggesting that they should try a different solution.

Figure 4 shows the same mission, this time solved – a yellow triangle card was dragged to the missing place in the first rule, a screen card was dragged to the first missing place in the second rule, and a purple star card to the second missing place in the second rule.[1]

The next challenge presented is taken from the subject 'Bar charts' (see Figure 5). This challenge starts with one green triangle and one yellow square, as can be seen on the game screen, and as written on the top left part of the screen. The challenge that the student has to accomplish is completing the missing cards in the rules, so that at the end of the game the number of shapes meets the bar chart as defined in the top right part of the screen.[2]

The last example we bring here is taken from the subject 'Graph patterns' – students need to complete the rules so that during runtime the trend of the number of shapes matches the graph pattern. Once they complete the rules, the game starts behaving accordingly, and

[1] https://youtu.be/D_qjpY4-npI
[2] https://youtu.be/DWyvas8A3Gw

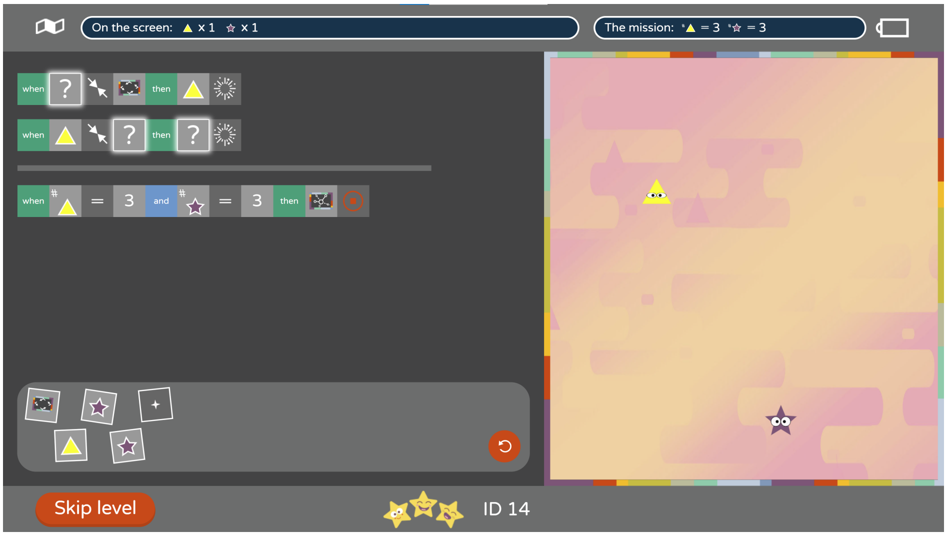


Fig. 3. Parallelism – a challenge

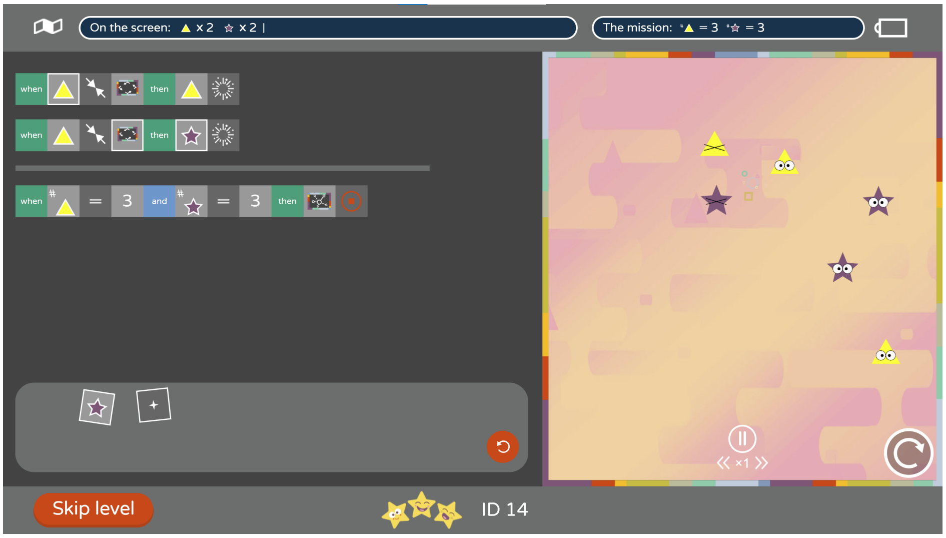


Fig. 4. Parallelism – a solved challenge

so red circles are created – the students' challenge is to make the shapes created at the right time to match the graph line trend.[3]

[3] https://youtu.be/Ky6CNKjUs0g

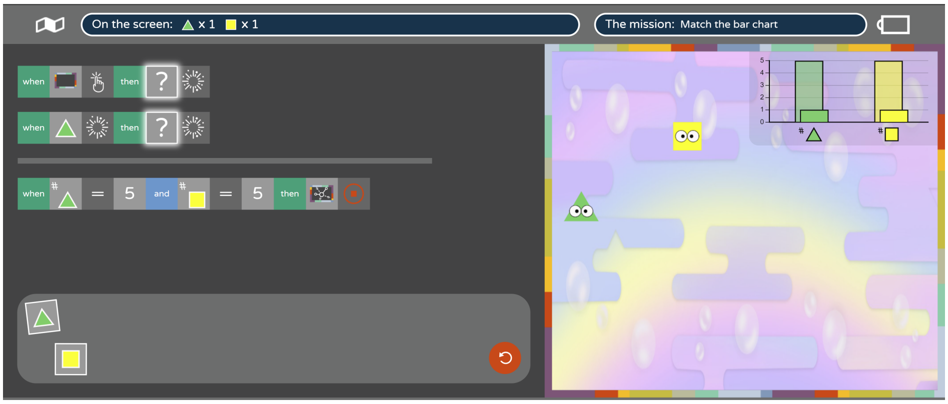


Fig. 5. ‘Bar charts’ – a challenge

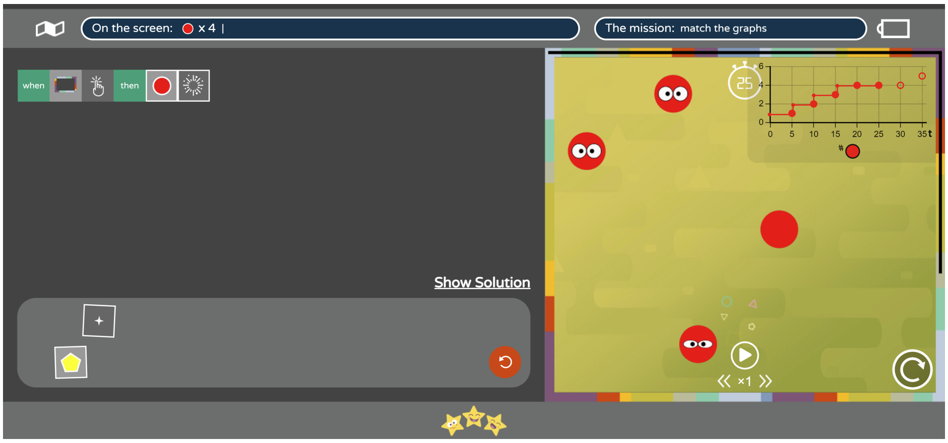


Fig. 6. ‘Graph patterns’ – a solved challenge

As learners progress in each subject, the challenges get more difficult and include more concepts learned so far. Figure 7 shows an example of a more advanced challenge which includes user interactions, multiplicity, conditional flow, and abstraction.

Finally, as mentioned earlier, students can create their own new challenges, using Spark Studio, and share them with their classmates or with the entire MyQ community.[4]

2.3. *Spark – Advantages and Disadvantages*

Spark offers numerous benefits for beginners. It is intuitive, uses the students’ native language, and isn’t tied to a specific programming language that may change over time. Spark covers a wide range of computer science topics, touches concepts explicitly – ensuring

[4] https://youtu.be/DpWF90pZH94

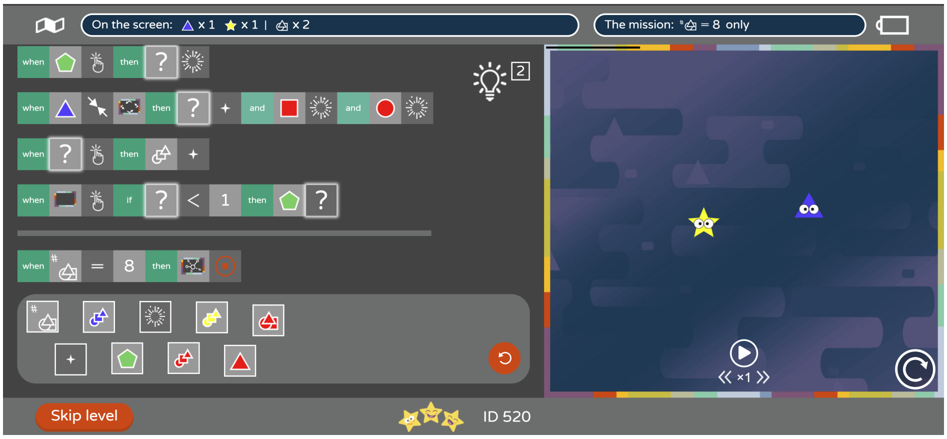


Fig. 7. An advanced challenge

students know exactly what they are learning. Each concept starts with simple levels and progresses to advanced levels, and as new concepts are learned, previous concepts are integrated into the challenges, so that as lessons progress challenges include more and more concepts.

Spark also has disadvantages. It doesn't teach a popular programming language, creativity is limited to the structure and shapes in the platform, and the first levels may not make more advanced students passionate enough to continue and play.

In the following section we elaborate on the advantages and disadvantages of Spark, especially when referring to novices.

2.4. *Spark – Advantages for Novices*

Intuitive learning environment. Spark's game-based platform facilitates an engaging and intuitive learning environment and a programming language that combines visual representation and text in students' mother language, making it inclusive and accessible to students without prior programming experience.

Emphasis on computational thinking. By focusing on the concepts and skills of computational problem solving through graphically attractive logical challenges, Spark prioritizes computational thinking over the syntax of conventional programming languages.

Foundation in scenario-based programming and LSCs. Spark is grounded in extensive research on scenario-based programming and Live Sequence Charts (LSCs), ensuring that the platform is based on solid scientific foundations that enhance the learning experience (Damm and Harel, 2001; Harel and Marelly, 2003; Harel *et al.*, 2010).

Scenario-based programming paradigm. Unlike traditional programming paradigms that require detailed instructions on how tasks are to be performed, Spark allows students to conceptualize solutions in terms of behaviors, fostering a deeper understanding of computational logic and problem solving strategies.

Comprehensive coverage of computer science concepts. The platform covers a wide set of computer science concepts. Each concept is introduced explicitly, making students aware of what they are learning. As new concepts are introduced, previous ones are integrated into challenges, ensuring a cumulative learning experience where challenges become increasingly complex, encompassing multiple concepts.

Lesson plans. The Spark team prepared detailed lesson plans that are provided with the platform. They guide teachers through the intricacy of computer science concepts and the various elements of the game. This is an additional advantage that can assist beginners and in-service teachers who are not familiar yet with the platform or teaching computer science.

2.5. *Spark – Disadvantages as a First Programming Language*

Limited exposure to programming syntax. While the emphasis on computational thinking is beneficial, students might not gain sufficient exposure to the syntax and structures of conventional programming languages, potentially hindering their ability to transition to more advanced programming tasks.

Restricted creative expression. Although Spark offers a rich environment for learning computational concepts, the platform's reliance on predefined structures and shapes may limit students' creative expression, confining their problem solving approach within the bounds of the platform's design.

Narrow scope of application. The specific focus on scenario-based programming and modeling may limit students' exposure to a broader range of programming paradigms and applications, potentially impacting their versatility in the field of computer science.

Dependence on visual and game-based learning. The heavy reliance on visual and game-based learning might not cater to all learning styles, possibly disadvantageous to students who would benefit from more traditional or text-based instructional approaches.

2.6. *Spark in the Age of AI*

In the domain of computing education, there is a vibrant tradition of innovating pedagogical methods to enhance students' understanding in foundational courses and to support educators in their daily instructional activities and assessments of students' tasks. Recent advancements in Artificial Intelligence (AI), particularly the use of LLMs to successfully generate executable code from problem descriptions in natural language, have introduced new challenges. The ease of use and widespread availability of these models, such as Chat-GPT, notably influence computing education, especially introductory courses. A critical issue arising from this development is the assessment of students' assignments that might have been completed using one of the AI tools.

In contrast, learning basic algorithmic concepts through Spark, which does not involve traditional programming languages used for algorithm implementation, significantly simplifies the assessment challenge at the foundational level. The Spark experience for learning the core principles occurs in a distinctly different manner, as coding is not required. Moreover, Spark's challenges are designed such that students must complete visual rules and run simulations that include user interactions. These tasks are beyond the current capabilities of AI tools, thereby maintaining the integrity of student assessments and learning outcomes.

3. Teachers

In the ever-evolving landscape of education, particularly in the still relatively young discipline of computer science, the demand for certified teachers has become critical. From elementary school to high school, recruiting qualified and certified computer science teachers poses a significant challenge (Gal-Ezer and Stephenson, 2010; Brandes and Armoni, 2019).

At the heart of the issue is the rapid integration of technology into every aspect of life, and the recognition that computer science, the discipline underlying the technology, should be introduced into the school system. Elementary schools face a unique dilemma in addressing the shortage of computer science teachers. Unlike their counterparts in secondary or higher education, elementary school teachers often lack specialized training in a discipline. Many educators at this level are generalists, responsible for teaching multiple subjects. Consequently, integrating computer science into the curriculum becomes challenging without dedicated expertise and resources.

In this context, the importance of detailed lesson plans cannot be overestimated. Comprehensive lesson plans serve as roadmaps, guiding teachers through the intricacies of the discipline's concepts and ensuring that students receive a structured and coherent learning experience. For elementary school teachers without a background in computer science, detailed lesson plans provide invaluable support, offering step-by-step instructions, activities, and assessments tailored to different learning styles and abilities.

Moreover, detailed lesson plans help to standardize education across schools, ensuring consistency in learning outcomes and facilitating the sharing of best practices. They empower educators to deliver high-quality instruction, even in the absence of specialized training, thereby mitigating the impact of teacher shortage on student learning.

The Spark team believes that the most effective learning strategy is combining the independent work of the students with explanations and discussions led by the teachers. Thus Spark is accompanied by detailed lesson plans.

We strongly believe in teachers as mediators of the subject and learning process. At the same time, as explained above, we acknowledge the need for lesson plans. The lesson plans provided with Spark cover the algorithmic concepts in depth, providing explanations and real-life examples especially from the students' daily life. These plans can also be used by teachers not yet experts of the platform or not confident enough in teaching the subject.

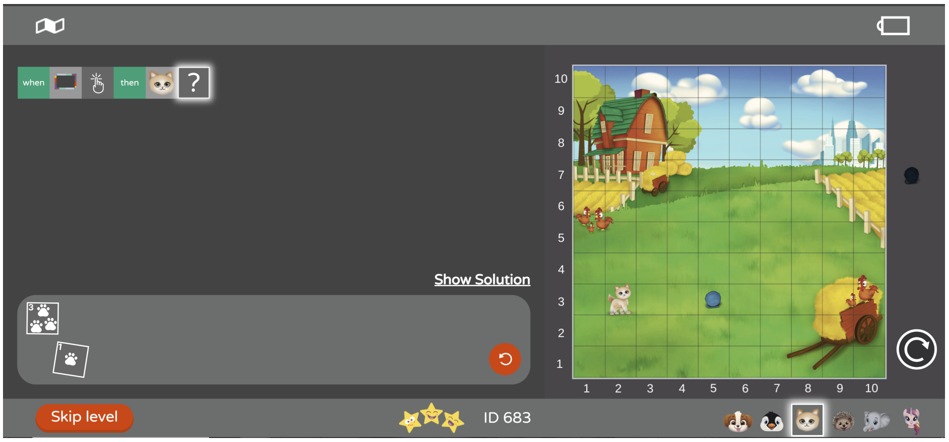


Fig. 8. User Interaction – An easy-to-solve challenge

In conclusion, the advantages of Spark mentioned above are relevant also for teachers, especially beginners or in-service teachers who do not have a lot of experience yet.

4. MyQ Pixel for Primary Education

MyQ offers a variant of Spark, named *Pixel*, which has children in 1st to 4th grade as target audience. Pixel is different from Spark in a few aspects which make it more suitable and engaging for young students.

First, it is designed to have cute animals each associated with a collectable item, a sound, and a typical background. As the students progress with the subjects, they receive more animals and can select with which animal to play.

For young children, Pixel, similar to Spark, eliminates the need to read by using a visual language. Unlike Spark, Pixel inherently defines a single and intuitive target for all missions – the animal, for example a cat, has a mission to collect all the collectable items associated with it, for example, balls of string.

This makes the interface simpler and the learning process more focused on a single type of mission, leaving more cognitive space for the subjects learned.

Similar to Spark, each subject in Pixel starts with simple challenges and progresses to more advanced ones, some of which including various subjects learned up to that point. Figures 8 to 10 show examples for increasing challenges in Pixel.[5]

5. MyQ Cosmos

Following the two tier strategy published by *Informatics for All* (Caspersen *et al.*, 2018) and aligned with the guidance in the *K–12 Science Education Framework* (NRC, 2012), which

[5]https://youtu.be/6eIr2rsf8aA

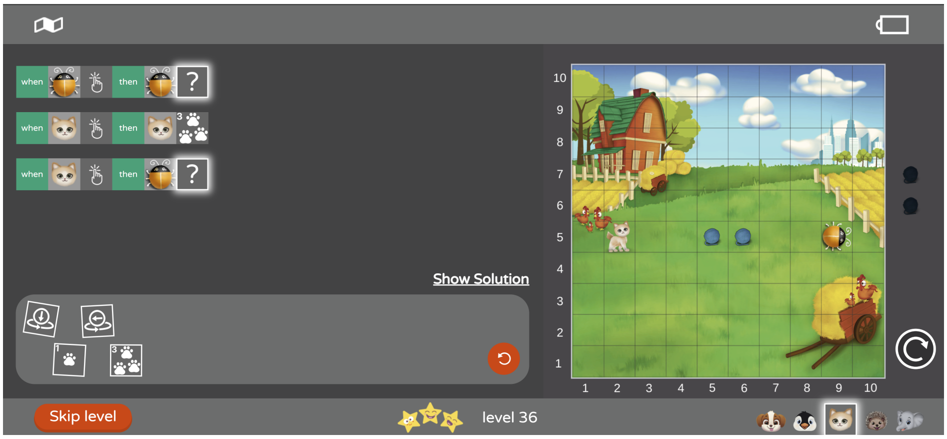


Fig. 9. Parallelism – A more advanced challenge

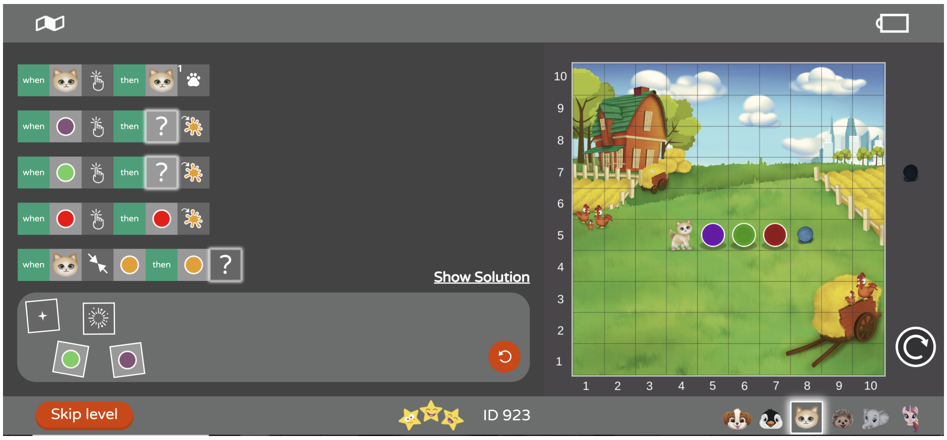


Fig. 10. Challenges gradually get more and more difficult

emphasizes the acquisition and application of computational thinking skills to formulate, analyze, and interpret computational models of scientific phenomena, MyQ introduces *Cosmos*. It enables students to construct scientific models through the advanced language capabilities of Spark. Cosmos bridges computational problem solving skills and STEM education.

When two disciplines are taught concurrently with the goal of problem solving or elucidation of phenomena, a significant challenge is the reciprocal enrichment of these disciplines (Carter, 2014; Nissani, 1997). Within the realms of computer science and the natural sciences, computational problem solving and mechanistic reasoning stand out as intertwined methodologies. Mechanistic reasoning, a pivotal scientific method, sheds light on phenomena by delineating interactions among entities. This reasoning style, mapping inputs (causes) to outputs (effects) through functional and interactional steps, bears resemblance to computational problem solving. Therefore, employing both methodologies to explain and simulate natural processes offers a profound opportunity for in-depth interdisciplinary education. This approach not only bridges critical crosscutting concepts from the K–12 frameworks, such as “Using Mathematics and Computational Thinking” and “Developing and Using Models,” but also integrates them with “Cause and Effect: Mechanism and Explanation,” alongside foundational ideas from the natural sciences, for instance, ecosystems or motion and stability (NRC, 2012; Saleh *et al.*, 2019).

Cosmos exemplifies such interdisciplinary educational practices. It lets students model and elucidate natural phenomena via simulations. Initially, students explore these phenomena using predefined simulations (e.g., an ecological system). Subsequently, they are presented with various pieces of evidence (e.g., a decrease in the number of mice correlates with a decrease in fox population), which they use to complete or refine rules within their model. After implementing these rules, students can execute the simulation to verify the stability and accuracy of their model against the provided evidence. Once confirmed, they utilize the model to predict changes under different conditions, such as the influence of building a neighborhood around a forest on the forest populations. This interdisciplinary platform thereby fosters concurrent development of computational problem solving and mechanistic reasoning, enhancing STEM education through model building and evidence-based reasoning within the context of natural phenomena.

Figure 10 shows a mission from the *Ecology* lesson in Cosmos aimed at 8th-grade students. On the left side are the rules, completed by the students and driving the simulation. On the right side is the simulation, in this case foxes, mice, and hawks in a forest with grass and trees. The simulation behaves according to what the student defines in the rules. For example, one of the rules in this mission says that when a fox meets a mouse it eats the mouse and gains energy. Accordingly, when the students observe the simulation, they will see that happening.

If the students had established a rule where a meeting between a fox and an eagle results in the eagle preying on the fox, the simulation would have illustrated this ecological interaction accordingly. Yet, as the simulation progresses, subsequent missions might present evidence that contradicts this rule, compelling the students to revise their assumptions and align the simulation rules with the new data.

6. Future Trends

As part of Spark’s disadvantages, we have mentioned its lack of creativity and its closed, though abstract, domain. MyQ addresses this with its new product for designing and creating games, the *Gamelab*. By leveraging the scenario-based, graphical, and intuitive language that has become the hallmark of MyQ, Gamelab offers a seamless transition for students already familiar with Pixel, Spark, and Cosmos. Yet, it significantly expands the horizon of possibilities. Gamelab will provide a rich graphical library, enabling the visual realization of students’ imaginative ideas.

Recognizing that the cultivation of creativity alongside analytical skills is paramount in preparing students for the complexities of the modern world, Gamelab is planned to

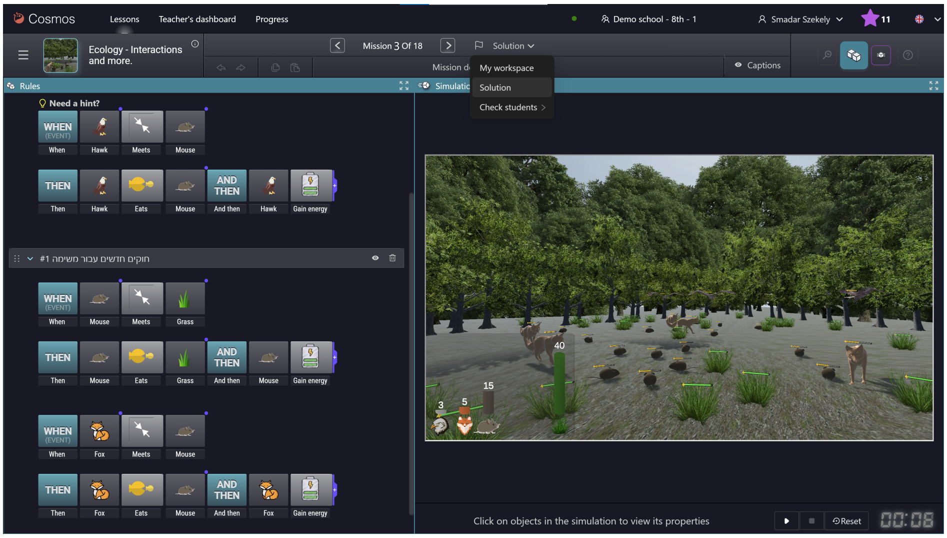


Fig. 11. A mission from Ecology lesson

demonstrate a holistic approach to education that combines creativity, computational, and analytical skills with fun. Thus, after experimenting with Spark, students can move one step further and apply the computational thinking skills acquired in a more open-ended context with Gamelab.

In summary, Gamelab, along with Spark and Pixel, focusses on blending creativity and problem solving skills. This has the potential of ensuring that students, starting from elementary school, are prepared with the necessary skills to thrive in the future.

Acknowledgements

We thank ChatGPT for its assistance in editing the language of this paper.

References

Armoni, M., Gal-Ezer, J., Haskel-Ittah, M., Marelly, R., Szekely, S. (2021). Computational Problem Solving with Plethora. In: *Local Proceedings of the 14th International Conference on Informatics in Schools (ISSEP 2021)*.

Armoni, M., Gal-Ezer, J., Harel, D., Marelly, R., Szekely, S. (2024). Plethora of Skills: A Game-Based Platform for Introducing and Practicing Computational Problem Solving. In: Abelson, H., Siu-Cheung, C. (Eds.), *Computational Thinking Curricula in K-12: International Implementations*. MIT Press.

Brandes, O., Armoni, M. (2019). Using Action Research to Distill Research-Based Segments of Pedagogical Content Knowledge of K-12 Computer Science Teachers. In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2019)*. Association for Computing Machinery, New York, NY, USA, pp. 485–491.

Carter, L. (2014). Interdisciplinary Computing Classes: Worth the Effort. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE 2014)*. Association for Computing Machinery, New York, NY, USA, pp. 445–450.

Caspersen, M.E., Diethelm, I., Gal-Ezer, J., McGettrick, A., Nardelli, E., Passey, D., Rovan, B., Webb, M. (2022). The Informatics Reference Framework for School. https://www.informaticsforall.org/the-informatics-reference-framework-for-school.

Caspersen, M., Gal-Ezer, J., Hall, W., McGettrick, A., Nardelli, E., Schnabel, J. (2018). Informatics for All: The Strategy. https://www.informaticsforall.org/informatics-for-all-the-strategy.

CECE (2017). Informatics Education in Europe: Are We All in the Same Boat? Technical report, Informatics Europe and ACM Europe. https://dl.acm.org/doi/book/10.1145/3106077.

Cooper, S., Pausch, R., Shaw, C. (1995). Alice: A Rapid Prototyping System for Virtual Reality. In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 8–11.

CSTA (2017). K-12 Computer Science Standards. Revised. https://csteachers.org/k12standards.

Damm, W., Harel, D. (2001). LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, *19*(1).

Fincher, S., Petre, M. (2004). *Programming Environments for Novices*. PB.

Gal-Ezer, J., Stephenson, C. (2010). Computer Science Teacher Preparation is Critical. *ACM Inroads Magazine*, 1(1), 61–66.

Gal-Ezer, J., Marelly, R., Szekely, S. (2020). Plethora of Skills – Play-Learn-Practice-Invent-Share. In: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2020)*. Association for Computing Machinery, New York, NY, USA, pp. 541–542.

Gal-Ezer, J., Beeri, C., Harel, D., Yehudai, A. (1995). A High-School Program in Computer Science. *IEEE Computer*, *28*(10).

Harel, D., Marelly, R. (2003). *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag.

Harel, D., Maoz, S., Szekely, S., Barkan, D. (2010). PlayGo: Towards a Comprehensive Tool for Scenario Based Programming. In: *Proceedings of the IEEE/ACM 25th International Conference on Automated Software Engineering (ASE)*.

Hazzan, O., Gal-Ezer, J., Ragonis, N. (2010). How to Establish a Computer Science Teacher Preparation Program at your University-The ECSTPP Workshop. *ACM Inroads Magazine*, 1(1), 35–39.

Maloney, J.H., Peppler, K.A., Kafai, Y.B., Mitchel Resnick, N.R. (2008). Programming by Choice: Urban Youth Learning Programming with Scratch. In: *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2008)*. Association for Computing Machinery, New York, NY, USA, pp. 367–371.

Nissani, M. (1997). Ten Cheers for Interdisciplinarity: The Case for Interdisciplinary Knowledge and Research. *The Social Science Journal*, *34*(2).

NRC (2012). *A Framework for K–12 Science Education: Practices, Crosscutting Concepts, and Core Ideas*. The National Academies Press, Washington, DC.

Saleh, A., Hmelo-Silver, C.E., Glazewski, K.D., Mott, B., Chen, Y., Rowe, J.P., Lester, J.C. (2019). Collaborative Inquiry Play: A Design Case to Frame Integration of Collaborative Problem Solving with Story-Centric Games. *Information and Learning Sciences*, 120(9/10), 547–566.