Plethora of Skills

A Game-Based Platform for Introducing and Practicing Computational Problem Solving

Michal Armoni[1], Judith Gal-Ezer[2], David Harel[3], Rami Marelly[4], Smadar Szekely[5]

Abstract

Many educational endeavors address the challenge of developing approaches, tools and platforms for strengthening analytical and structural ways of thinking, deemed essential to effective problem solving in various disciplines. One important approach along these lines, *computational problem solving* (CPS), often called computational thinking, encompasses a set of techniques and strategies that involve expressing problems and solving them in ways that a computer could execute. Plethora addresses this challenge, concentrating on CPS. It is a game-based platform, invented by computer scientists from the Weizmann Institute of Science, collaborating with pedagogical experts from The Center for Educational Technology. Plethora teaches the concepts and skills of CPS via graphically attractive logical challenges that are easily and concisely described in natural language. This contrasts other platforms aimed at teaching CPS using standard programming environments. Plethora is implemented in many schools across Israel, and was selected as the leading tool for school-level cyber competitions in Israel. Research we conducted recently with 4th grade students to evaluate the effectiveness of teaching CPS, exhibited positive results concerning students who experienced Plethora relative to those who did not.

1. Introduction

For over twenty years, Israel's high school system has been following a rather novel program in Computer Science (CS). The program, which is an elective, like Biology, Chemistry and Physics, was updated periodically according to the development of the discipline. Some of its units have been thoroughly revised, and some replaced by new ones. The rationale of this program is to introduce CS as a scientific subject that deals with solving computational problems. The program and its implementation have been described in, e.g., [Gal-Ezer et. al., 1995, Gal-Ezer & Harel 1999, Gal-Ezer & Zur 2004].

[1] Dept. of Science Teaching, Weizmann Institute of Science, michal.armoni@weizmann.ac.il

[2] Dept. of Mathematics and Computer Science, The Open University of Israel, galezer@cs.openu.ac.il

[3] Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, dharel@weizmann.ac.il

[4] Plethora Technologies Ltd. rami@iamplethora.com

[5] Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, smadar.szekely@weizmann.ac.il

More recently, corresponding programs for elementary school (grades 4-6, ages 10-12) and middle school (grades 7-9, ages 12-14) have also been designed and implemented, although in terms of usage some are still in the pilot stage. Such programs should strive to introduce new pedagogical platforms and tools to help teach the discipline's basis, which is new to most of the teachers and learners.

In this paper, we discuss a new approach for teaching young students the basic concepts of CS and endowing them with the relevant problem-solving skills, using the innovative, game-based Plethora platform. We also discuss the implementation of the approach in Israel and report on results of its usage with 4th graders.

Plethora is very different from other platforms with similar goals. First, it builds upon a relatively new programming paradigm, *scenario-based programming*, which was launched with the advent of the language of *Live Sequence Charts* (LSC) [Damm & Harel, 2001, Harel & Marelly, 2003]. 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 wrt 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 second difference is that Plethora develops computational problem solving skills without focusing on programming. Students are not expected to code or to even read conventional programs. One of the most interesting of Plethora's underlying principles is to help understand complex systems by changing a given system's *rules of behavior* (i.e., the mandatory scenarios) and seeing the effect of this by viewing the resulting system in action.

Plethora consists of multiple, increasingly difficult, game levels, and a level editor, with which players can create their own challenges and share them with the Plethora community. The intuitive visual rules are presented in a way that reflects the learners' "mother-tongue". Finally, the Plethora package provides lesson plans with guidance for teachers, as well as unplugged activities and real-life examples.

The Israeli Ministry of Education adopted Plethora as a platform in its country-wide Cyber competitions, and encourages schools to adopt it as an environment for introducing computational problem solving in elementary school. We have conducted a study, where 4th graders were introduced to Plethora. It shows that Plethora helped the students learn basic concepts of CS, and that this experience had the strongest impact when preceded by a short lesson introducing it.

In Section 2, we discuss computational problem solving and its relationship with computational thinking and algorithmic thinking. Section 3 describes the Plethora platform, and in Section 4 we discuss teaching and learning computational problem solving with Plethora. Section 5 discusses the implementation and usage of Plethora in the Israeli school system, and our research with the 4th graders. Chapter 6 discusses some thoughts about the future.

2. Computational Problem Solving

Computer scientists and computer science practitioners deal with and study computational problem solving. They solve algorithmic problems by devising appropriate algorithms that are as efficient as possible and establishing their correctness. For problems that cannot be solved algorithmically at all, they try to prove that fact, and for the solvable ones they attempt to determine inherent levels of complexity. Identifying connections between problems and classifying them by their solvability and complexity are central to their work.

This work is characterized by a set of ideas, thinking patterns, strategies and skills (e.g., abstraction, non-determinism, and decomposition), which, over the years, has been given different names. The first was "algorithmic thinking" [e.g., Knuth, 1985]. More recently "computational thinking", originally coined by Papert [1980] and used by Jeannette Wing in her influential article [Wing, 2006], has become especially popular. We prefer to emphasize the context in which this set of concepts and skills is used, and hence refer to it as *computational problem solving*, or CPS for short.

The ingredients of this set are not unique to CS. Most of them are useful in other disciplines too, although they may be expressed in different ways, which was the rationale for the call to teach it broadly [Wing, 2006]. However, as the discipline of CS has developed and matured, the basics of CPS have been enhanced, deepened, expanded, becoming more powerful and more effective. For example, skills related to data analysis and organization are valuable in many contexts outside CS, and were exploited therein long before CS emerged as a discipline. Still, computer scientists have become experts in developing optimal methods of organizing data together with the appropriate actions and manipulations that are to be performed on them. Therefore, teaching CPS from the perspective of CS can provide learners with effective additions to their toolbox, provided they are taught in a generalizable and transferable manner.

Many initiatives for teaching CPS to a young audience utilize designated programming environments, and assume that by programming one is exposed to CPS at a level sufficient to enable its use in other contexts. Working in such environments indeed exploits some concepts that are beyond programming per se (e.g., abstraction). However, these are not always explicitly acknowledged and discussed, limiting the usual terminology to the programming act itself and often only to its low-level and technical facet, i.e., coding. With such learning, one can hardly expect the students to recognize higher-level ideas and thinking patterns, which is necessary for acquiring them as part and parcel of their toolbox, and allowing their transfer to other contexts.

In addition, although many of these environments are based on non-imperative programming (e.g., Scratch, based on the event-driven paradigm, and Alice, based on the object-oriented one), the corresponding educational programs often emphasize the imperative approach. Although some concepts central to these other paradigms are taught (e.g., message passing and events in Scratch),

the greater emphasis is on sequential instructions and simple control structures (sometimes also procedures). This inherently limits exposure to CPS to a restricted collection of ideas and habits thereof.

Moreover, in line with Bruner's view on teaching fundamental ideas [Bruner, 1960], by failing to explicitly acknowledge paradigm-related issues, one also misses the opportunity to relate to manifestations of the general ideas that are unique to the paradigm at hand, rendering generalization and transfer to other contexts even harder. The classic example of this is the notion of abstraction, essential to CS, which has different manifestations in each of the problem-solving paradigms. An explicit treatment of abstraction, while recognizing similarities and differences among its different flavors, has the potential of promoting the acquisition of general abstraction skills as a valuable and transferable tool.

Plethora offers a fresh approach. It is based on a paradigm that is very different from the imperative and procedural paradigms, and lends itself naturally to the ability to introduce to young students many of the fundamental concepts of CPS, while emphasizing the phase of devising solutions rather than that of programming them.

3. Plethora

The Plethora Game

Plethora is an online game-based educational platform. It is organized by topics representing algorithmic concepts, each of which includes numerous levels of varying difficulty.

Figure 1 Part of the topics screen.

A *level* is a small system with a set of possible behaviors, and is composed of four parts:

1. An *initial state* – a definition of the set of the shapes present on the game screen when the level starts.

2. A *goal* – a definition of the set of shapes required to be on the game screen at the end of the level.

3. A set of partially-specified *activation rules* – when completed, these are to correctly define the behavior of the system represented by this level.

4. A *halting rule* – specifies the condition that causes the level to end.

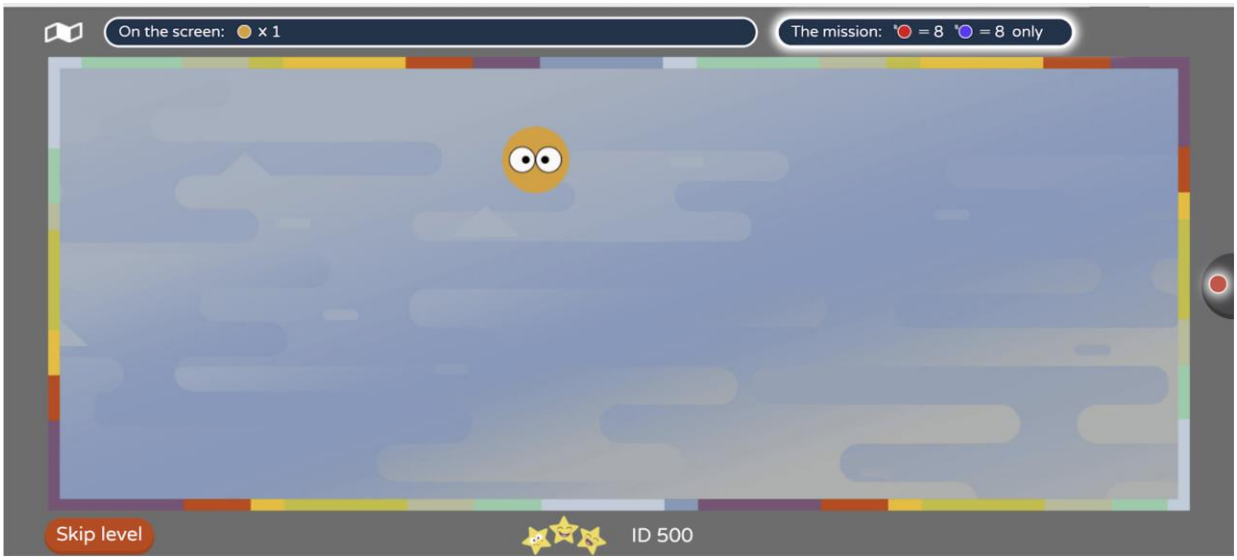

Figure 2 A simple initial state.

The challenge is to complete the partially-specified rules so that, when run, they will cause the system to behave correctly: if started in the initial state it will eventually halt with the goal state holding. The difficulty of the challenge increases as one progresses up levels.

The rules themselves are depicted visually, using simple icons with intuitively clear meanings, and are independent of conventional programming language syntax. The graphic visualization is accompanied with text in the students' native language. After completing the missing parts, the students watch the full completed system running/executing on the game screen.

The user can also experiment with completing and activating only some of the rules, observing how these affect others, and determining the overall effect on the system's execution.

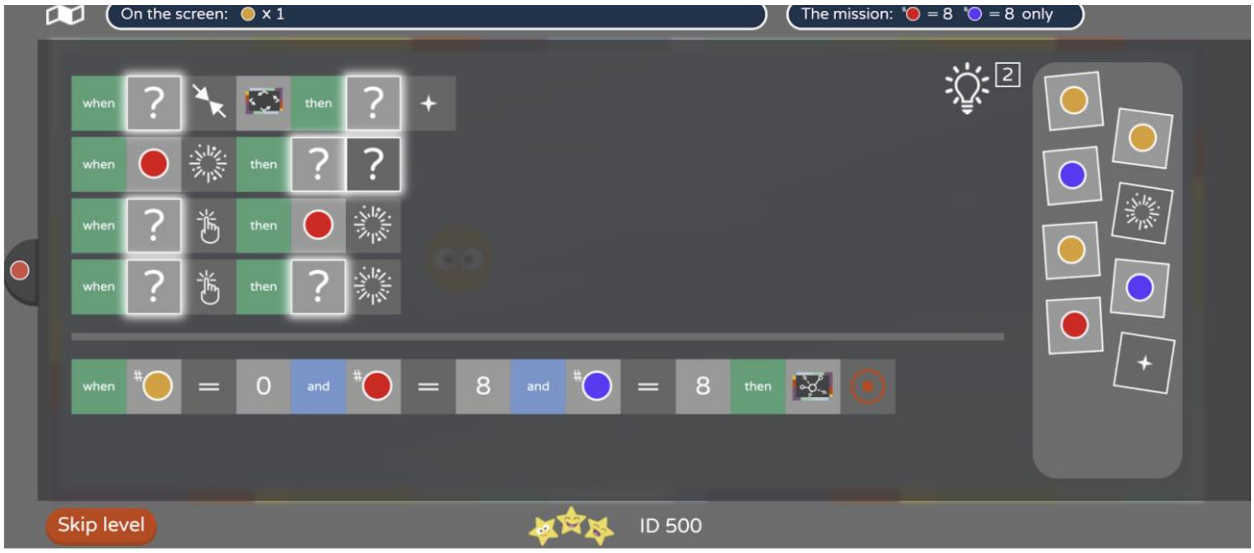

Figure 3 A level's challenge, calling for filling the empty slots in the rules on the left (denoted by "?") with icons from the "icon tray" on the right, in a way that causes the goal to be achieved.

If the completed set of rules causes the system to reach the goal state, the level ends successfully and the next level unlocks. Otherwise, after a certain fixed running time (currently about 40 seconds), the system will halt on its own accord , and will ask the student to either continue with the current level's execution or go back to the activation rules and try to solve the challenge in a different way. This iterative process continues until the goal is achieved.

Figure 4 An execution of the completed set of rules, where one can watch the system in action, perform user actions if and when required, and check whether it runs as expected.

Plethora Studio

Addressing the challenges Plethora offers, especially the harder ones, requires elaborate computational problem-solving skills, and of various cognitive kinds. However, a totally new dimension of creativity is required when asked to create a new challenge from scratch. This calls for assimilating the learnt concepts sufficiently well to be able to utilize them in wider contexts, and it is here that Plethora Studio comes in – a level editor, with which learners can create their own new challenges and share them with the Plethora community.

Plethora Studio takes students through the structural process of creating a new level, a process that is not unlike actual system design. The initial state and the goal state have to be defined, accompanied by the set of rules that are to lead from the former to the latter.

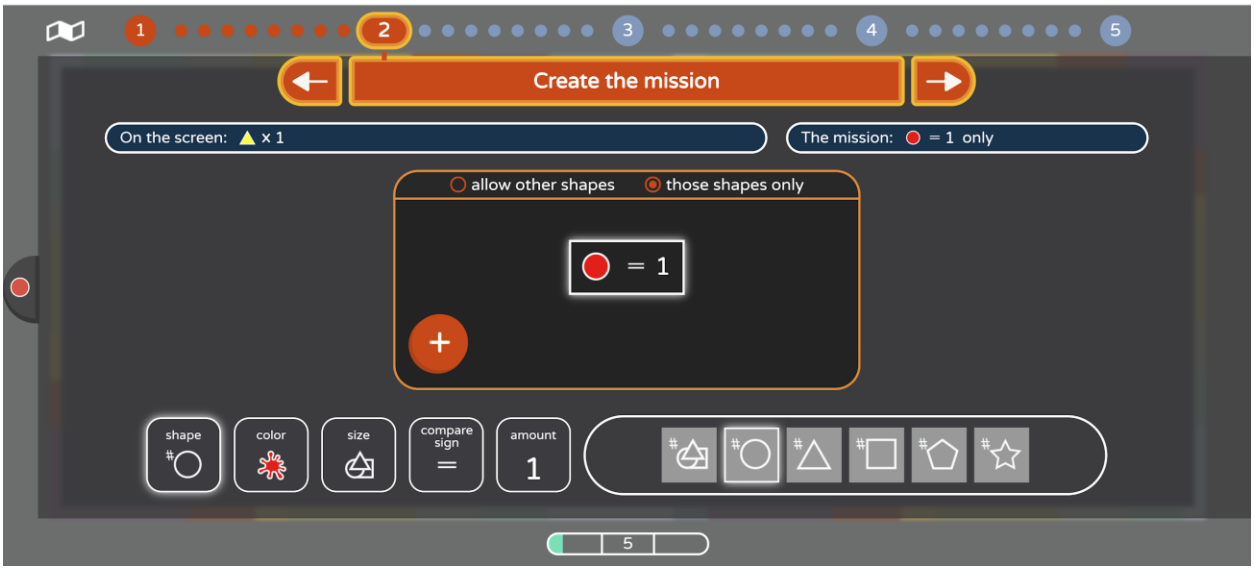

Figure 5 Working with Plethora Studio: defining the new level goal state.

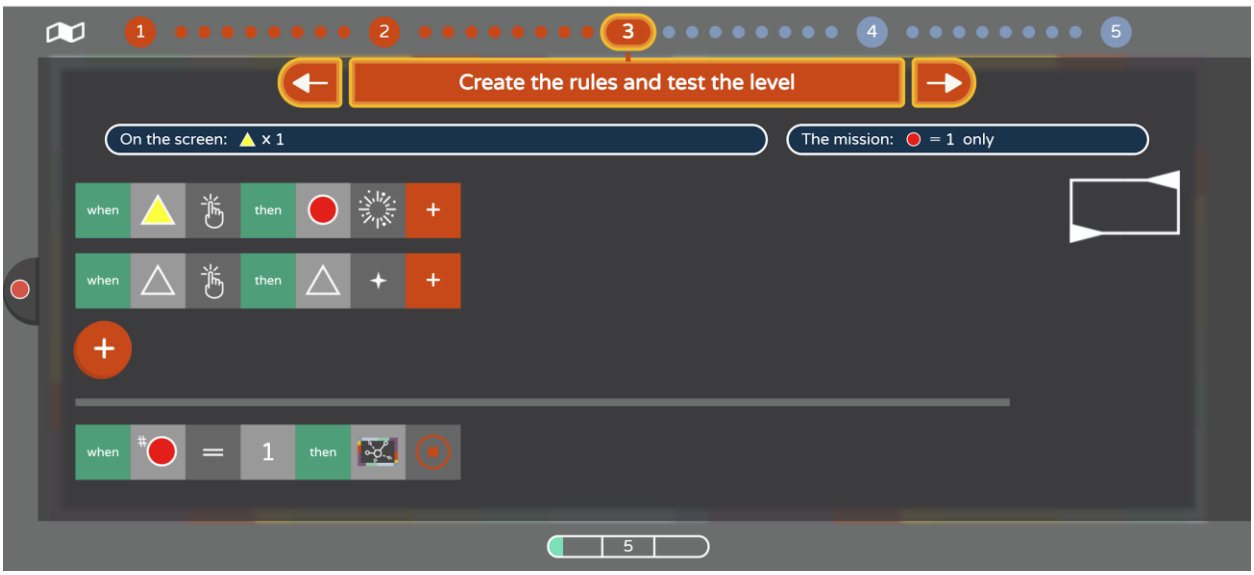

Figure 6 Working with Plethora Studio: defining the set of rules representing the system's behavior.

Once the new level is defined, the user turns it into a challenge by "un-completing" some of the rules, replacing parts of them with the "?" symbol. The challenge can then be shared with schoolmates or with the entire Plethora community.

The combination of a closed approach, where given challenges are to be solved, and an open one, where users create challenges of their own, is of great value. It addresses the two classical ideas of learning through riddles and learning by doing (creating), which are known to be motivating instructional approaches [Schank, Berman, & Macpherson, 1999; Delal & Oner, 2020].

Plethora and the Scenario-Based Paradigm

Plethora is based on the scenario-based programming paradigm (SBP), originating with the language of *Live Sequence Charts* (LSC) [Harel & Damm, 1999], the play-out execution method [Harel & Marelly, 2003] and the Play.Go tool [Harel, Maoz, Szekely & Barkan, 2010]. It was intended for software and systems engineers, enabling them to intuitively specify and execute complex system behavior without the need for conventional programming. SBP introduces a unique way of specifying requirements, which, rather than being centered on prescribing the system's state-to-state progress, is based on defining scenarios of a systems' behavior, including mandatory, possible and forbidden ones. It is thus very close to a human's way of thinking about system behavior.

Another important aspect of SBP is that it focuses on the *inter-object* interactions between the system's objects rather than the *intra-object* behavior of each single object. As a result, the focus is on *what* is done in the system rather than on *how* it is accomplished.

Due to the intuitive nature of the SBP and its close alignment with how people express requirements in natural language, we found it to be highly suitable for exposing students to the basics of CPS. For example, consider the following rule, which describes a desired abstract behavior.

Figure 7 A Plethora rule.

The rule states that when a yellow star is created, all red circles change their size to be large and all green shapes become blue. However, it does not specify **how** this is achieved (e.g., how do all shapes know that a yellow star was created, what is the mechanism to notify all red circles that they should change their size, etc). Hence, a set of rules in Plethora does not look like a set of internal instructions for specific objects, but rather as a high level description of system requirements, and therefore does not follow the traditional pattern of a program. Indeed, our experience shows that students do not perceive their work with Plethora as programming but as engaging in a challenging game. This makes the CPS class much more attractive.

Plethora covers a variety of subjects, which are also considered important by PISA in the new mathematical framework [PISA, 2021]. One of the most important of these is *decomposition*, which is actually an expression of algorithmic abstraction, a crucial facet of CPS. Because it follows that scenario-based approach, Plethora's rules represent a natural decomposition of the system's behavior, not its structure. Students are encouraged by the rules to decompose their solution into smaller behavioral fragments that together comprise the overall desired behavior. Hence, we actually have behavioral abstraction.

Plethora supports many concepts inherent to CPS (cause and effect, conditional flow, iterative actions, compound conditions, algorithm halting, multiplicity, etc.), some of which are considered hard to teach in younger grades, and are made easier with Plethora. In the next section we discuss some of these, but, due to space limits, not all of them.

Some video clips demonstrating Plethora in action can be found at:
https://drive.google.com/open?id=1CEDOIfwxqZe9c5Sh5tLY1dUmrrd2SlJM.

4. Teaching CPS with Plethora

Our experience shows that playing and practicing alone is not enough for achieving deep understanding of fundamental concepts. One should also receive carefully constructed and guided explanations and examples thereof. Accordingly, in this section, we elaborate on how we recommend teaching with Plethora.

Plethora's unique approach promotes teaching the foundations of CPS to extend naturally into other aspects of the students' life. This is supported by the use of simple, domain-neutral, abstract geometric shapes, but also, significantly, by an accompanying set of lesson plans, with which teachers can better explain CPS and show how to use it in day-to-day domain-specific problems. Each of the lesson plans includes an introduction to and explanation of the learnt subject, examples from Plethora and from real-life situations, hands-on practice with Plethora and a summarizing discussion about the students' experience.

Iterative Action Lesson Structure

Activity	Time
Part 1 – Unplugged activity	10 min
Part 2 – Introduction to iterative action	10 min
Part 3 – Independent activity with Plethora	20 min
Part 4 – Discussion	5 min
	Total: 45 min

Figure 8 The structure of a lesson, taken from Plethora's Iterative Action lesson plan.

The learning process of a computational topic begins with social unplugged classroom activities, exposing students to it in a direct, informal manner. These include games, where, e.g., an action performed by one student triggers an action of another (cause and effect) or where several students carry out activities simultaneously (parallelism), and fun quizzes, where students have to guess an object by asking questions about its characterizing properties (objects and properties).

The next step is to introduce the topic in the context of Plethora. The meanings of the icons supporting it are first described, according to the semantics established in class in the previous activities, thus further strengthening links between the game and the surrounding reality. The students then start practicing Plethora on their own, or in pairs, solving new challenges as time permits. Some levels are left for the students to continue at home.

During this learning phase, interesting dynamics usually emerge. Some competition between the students obviously exists, but we also witnessed high motivation of students to help those who are experiencing difficulties. Thus, interestingly, competition becomes collaboration and the teacher becomes a learning facilitator rather than the sole source of knowledge and wisdom.
After practicing on their own, the class ends with a discussion and summary, with students showing how they solved some of the more challenging levels.

This process serves two goals. First, students experience first-hand the fact that the same problem, even limited in size and complexity, may have multiple valid solutions. The discussion can thus

address the pros and cons of the solutions proposed (complexity, clarity, etc.). Second, students are required to explain their solutions to the others, so they must be able to articulate the steps taken to build the solution and justify the reasoning behind it. All this is far more demanding than merely solving a Plethora challenge, and helps the students learn the principles of CPS, and develop the corresponding skills, on a far deeper level.

After the students are able to solve non-trivial challenges for a certain topic, the learning process takes another step, where they are required to creatively invent their own challenges using Plethora Studio. Here they are exposed to system design issues – planning, adhering to requirements, etc. They can be asked to create "freestyle" challenges, where they determine the initial state, the final goal and the difficulty of the level, or to create a challenge that meets certain criteria; for example, being given the initial state and goal state and constraints regarding the number of rules used, the icons that are allowed to appear, etc. Creating such a challenge, making sure it adheres to the constraints, and then turning it into a problem for their friends, is both challenging and fun.

Here now are some of the most interesting CPS topics that can be taught with Plethora.

Parallelism: Dealing with systems that exhibit not-necessarily-sequential behavior, where different parts of the behavior can occur simultaneously and thus affect each other in hard-to-predict ways. Parallelism is known to be difficult to teach [Ben-David Kolikant, 2004], since it requires the student to consider the system as a whole and take into account the occurrence of simultaneous actions. Nevertheless, being a central facet of real-world complex systems, parallelism is also an important part of CPS.

Being inherent to the scenario-based approach, parallelism is thus a natural part of Plethora, where it shows up in the three forms: (i) the rules are all continuously checked and monitored in parallel; (ii) a single trigger can activate multiple rules, causing them to execute in parallel, as shown in Figure 9; and (iii) using the "and" operator, a single rule can specify that two or more events are to occur simultaneously.

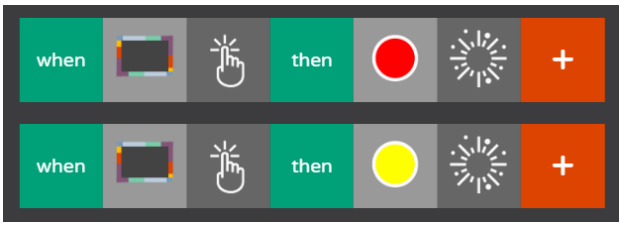

Figure 9 Example of parallel rules.

Plethora distinguishes between "and" (indicating parallel execution) and "and then" (indicating sequentiality). The difference between the two is also emphasized by the execution mechanism of

Plethora that uses virtual clock ticks to synchronize the execution of parallel events and delay intervals to enforce sequentiality.

Randomness: The unpredictability in parallelism is enhanced by the randomness inherent to Plethora, since the shapes move within the screen in random directions. By experimenting with Plethora, the students become familiar with the effects of randomness and learn to take them into consideration when solving challenges and constructing new ones.

Data abstraction: Plethora rules can refer to abstract shapes by ignoring the values of some of their attributes, thus allowing one to specify the behavior of different objects using a single abstract rule. For example, a rule can say that when any green shape meets any big circle all the red squares change their shape to a star. This way, students understand the concept of multi-dimensional abstraction without having to explicitly introduce inheritance and polymorphism.

Time: Plethora introduces the concept of real-time and timing constraints using a timer object, which can be reset and queried for its value. Thus, students have to deal with tasks required to complete within a given time, and with actions that can be executed only within an allowed time window.

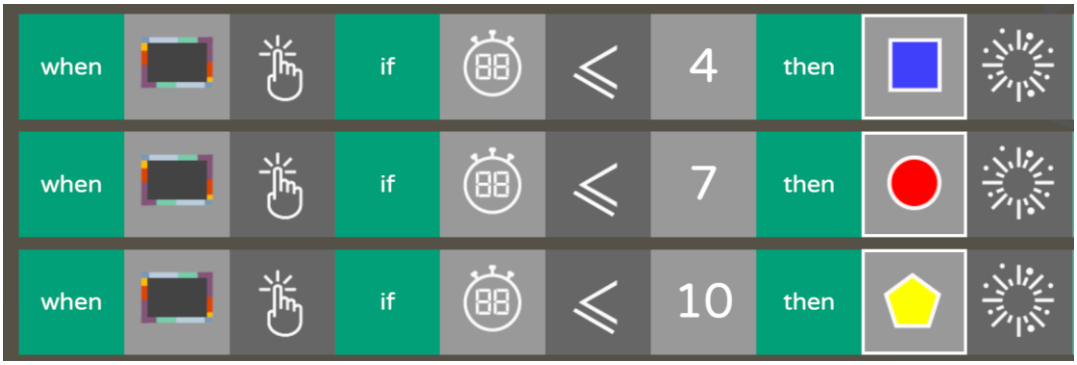

Figure 10 Rules that involve a timer. The first specifies that a click causes a blue square to be created only if no more than 4 seconds elapsed since the beginning of the execution.

Recursion: Considered to be one of the most elusive and difficult to teach, recursion occurs when a thing, or a behavior, is defined in terms of itself or of its type. The most common kind is procedural recursion, where a procedure can invoke itself. Procedural recursion is based on procedural abstraction, and specifically on the concept of a black box, known to be difficult in its own right [Armoni, Gal-Ezer & Hazzan, 2006]. To understand or design a recursive procedure, one has to accept that the procedure does what it is supposed to do before its details are fully known. This difficulty is augmented by the inherent unbounded nature of an entity that includes itself, so to speak. To deal with this, instructors often use the copies model [Kahney, 1989], which helps students realize that the invoking and invoked entity are not really the same entity but merely two copies thereof.

Plethora has a simple kind of recursion, also taken from the scenario-based paradigm: a rule can cause the occurrence of the event that triggers it (see Figure 11). In this way, the baffling self-referential nature of classical recursion is eliminated: there is no explicit (visual or textual) inclusion of the name of a rule inside the rule itself. Furthermore, it is clear that the recurring occurrences of the event are not the same. We thus get the copies model for free. Mutual recursion is also easily represented in Plethora, as depicted in Figure 12.

Figure 11 Recursion: The rule triggers the very same rule, by the appearance of one orange pentagram causing the appearance of another.

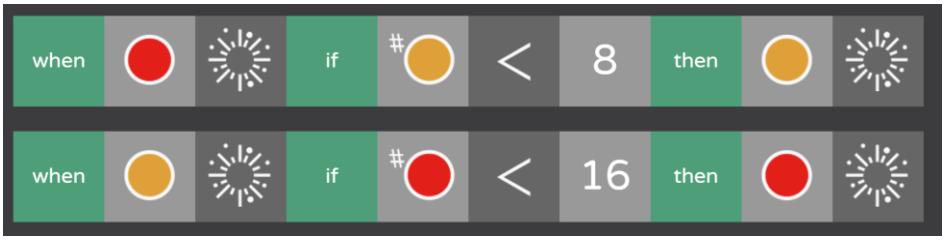

Figure 12 Mutual recursion.

Students can experience the power of recursive behaviour by executing recursive rules and watching their effect on the game's screen. Figure 13, taken from Plethora's lesson plan, illustrates a rule's recursive behavior by showing how each instance is activated by the effect of a previous instance of the same rule.

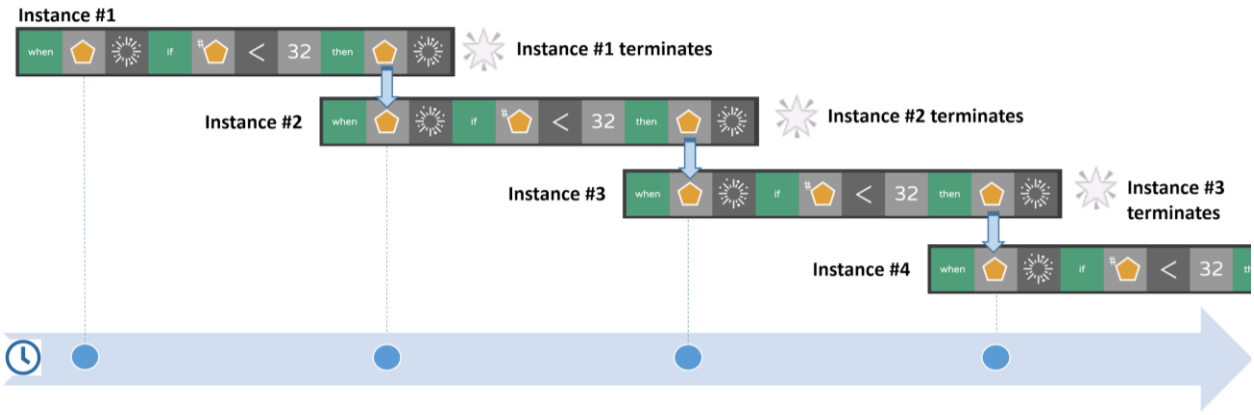

Figure 13 Multiple instances of a recursive rule, with time flowing from left to right. Upon the creation of an orange pentagon, a new rule instance is created and activated. This recursive execution will continue until the halting condition is met.

5. Plethora in Israel

Israeli Cyber Competition

To encourage students to learn and practice CPS, Israel's Ministry of Education (MoE) organizes an annual event, called the Israeli Cyber Competition (ICC). Students from grades 1 to 12 are exposed for a month to various platforms, according to their age, from which they can choose where to play. The competition is conducted between schools, with each school receiving the points accumulated by its students, normalized by the number of its students.

Plethora was chosen by the MoE as one of the platforms for the 2019 ICC, thus exposing the game to over 55,000 students in grades 5 to 9, country-wide. Following its success, in 2020 the MoE selected Plethora for students in grades 4 to 9, exposing it to an additional 140,000 students.

During the competition, the following topics were introduced via Plethora: cause and effect, user interaction, sequentiality and parallelism, termination conditions, objects and attributes, logical operators, generalization through multiplicity and abstraction, iteration and recursion. Here are our main insights.

A surprising fact was registered at the junior-high level, where the number of players increased by a factor of six (from 3,000 to 18,000) in comparison to pre-Plethora years, suggesting the engagement that Plethora was able to create. We believe that there are two main reasons for this. First, working with Plethora is not perceived as programming, a fact that lowers the level of fear and resistance for some of the students. Second, Plethora was designed from the start to encourage determination and collaboration, rather than competition, and to encourage accepting that mistakes are a normal part of the learning process.

The 5th and 6th graders played 150 levels in an average time of 10.5 hours, and the 7th to 9th graders played 200, more challenging, levels in an average time of 20.5 hours. During the two 3-week competitions, students practiced a total of almost 12M levels, of which 8M levels were completed successfully, and the rest incorrectly.

During the competition's finals, students were exposed to Plethora Studio for the first time. After a short introductory video clip, they managed to create their own levels, which were then handed over to the other competitors to solve. We were very impressed by the enthusiasm of the students in creating their own levels, and their excitement in solving levels created by other students.

Finally, one of the most interesting phenomena was gender diversity: 49% girls vs. 51% boys, a balance that CS educators strive to achieve.

Figure 14 The Israeli Cyber Competition.

In-Class Study

We conducted a preliminary in-class study to examine the effects of Plethora on understanding and internalizing the principles of CPS. In particular, we were interested in cause and effect, parallelism, logical conditions, abstraction, and algorithm design. We include here only a very brief description of the research and part of its findings.

The research population included 257 4th graders from five elementary schools in Israel. They were divided into five groups, each following a different path and then responding to an identical questionnaire.

Group	**No. of Students**	**Path**
Q	51	No prior exposure to Plethora → **Q**uestionnaire
PQ	82	Experienced **P**lethora with no guidance → **Q**uestionnaire
LPQ	68	Attended a Plethora **L**esson → practiced **P**lethora → **Q**uestionnaire
PSQ	24	Practiced **P**lethora → attended a **S**ummary lesson → **Q**uestionnaire
XPQ	32	Previous e**X**posure to **P**lethora → **Q**uestionnaire

Table 1: The five groups participating in the research.

Since the five groups were not of the same size, we chose to examine the pool of *all* answers (the number of questions multiplied by the number of students), and compute the success or failure rate of each group by dividing the number of right or wrong answers of the group by the number of the entire pool of answers, respectively.

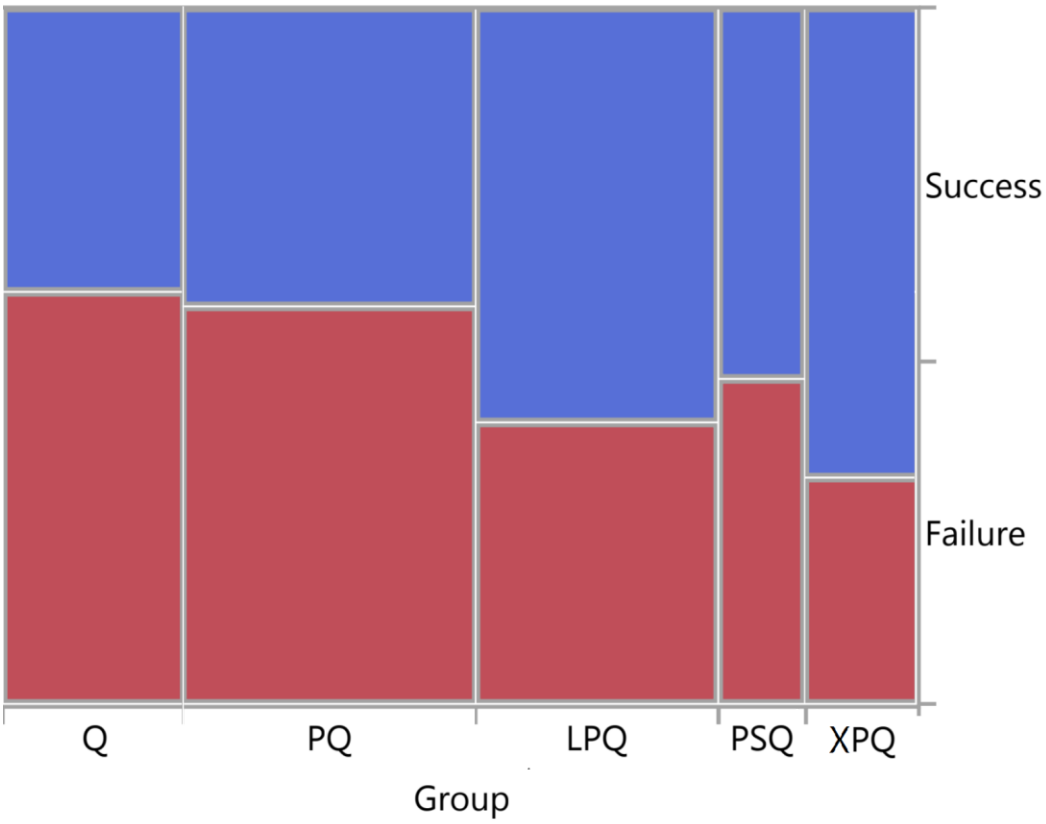

Figure 15 The success and failure rates for all the groups.

Figure15 shows the findings for all groups. The students in Q (no experience with Plethora) exhibited significantly lower results compared to those in LPQ and PSQ, whereas no significant differences were found between LPQ and PSQ or between PQ and Q. The differences between LPQ and Q, and between PSQ and Q are significant. These results are in line with what we expected: The combination of Plethora with in-class learning contributes to the students' CPS abilities, supporting the learning process described in Section 4.

It is noteworthy that students who experienced Plethora prior to the study (XPQ) performed significantly better than all other groups, though this should be taken with a grain of salt since we did not have sufficient information regarding their previous background.

The study provided some additional interesting findings, the details of which are beyond the scope of this paper. For example, some of the results may suggest that a richer experience with Plethora increased students' self efficacy regarding their CPS abilities.

All in all, the findings show that Plethora indeed makes a difference. Even a relatively modest exposure to Plethora, enhanced by a short in-class lesson, is significant. This is in line with the known importance of teacher involvement when introducing a new subject or tool.

6. Conclusions and Future Directions

Plethora is a new educational platform based on the novel scenario-based programming approach. Utilizing the strengths of this approach, combined with gamification, students design, plan and understand complex systems and algorithms in an intuitive, fun, and engaging way. The platform, with its associated lesson plans, serves as a powerful tool for developing CPS skills. We envision additional ways to further exploit the potential of Plethora for CPS and other domains.

One of the required skills for the workforce of the future is teamwork, which must be accompanied by the ability to collaborate with peers. Accordingly, our in-class study also examined some aspects of working in pairs. There are, in fact, numerous reports that discuss the advantages and drawbacks of working in pairs [e.g., Tsan et al., 2020; Denner et al., 2014]. In the context of Plethora, we found that pairs were able to complete more levels in less time than when working individually. However, we also saw that working in pairs had two drawbacks. One is that it can be done properly only with physical presence, i.e., the pair sit side by side. The second is that a stronger student can be dominant, resulting in the weaker student not really participating [Braught, Wahls, & Eby, 2011].

To allow a more constructive way to collaborate, which can be effective also in periods of social distancing and lockdowns, Plethora has recently introduced a teamwork mode, where students connect remotely and solve challenges together. Each student receives only some of the icons and is responsible for completing only some of the rules. Hence, to succeed, they must work together and figure out how each can contribute to the overall process. We plan to investigate the impact of this collaborative mode on the learning experience and its effectiveness.

To further explore the impact of working with Plethora on the understanding and internalization of CPS, we have joined forces with IsraAid organization and the Ministry of Education of Dominica to conduct an extensive longitudinal study in the 2021 school year.

As we claimed earlier, Plethora's mission is to teach the foundations of CPS, so that the students can use them in other aspects of their life. Being built around the scenario-based paradigm, and

thus being able to describe the behavior of complex systems in a rather intuitive way, we believe Plethora is suitable for teaching fundamental thinking processes relevant to science in general. Recently, the US National Academies of Sciences, Engineering and Medicine (via NRC, the National Research Council) published a Framework for K-12 Science Education. It introduces the notions of *practices*, *crosscutting concepts* and *core ideas*, emphasizing the need to focus on skills and methods rather than on specific domain content.

This framework is also aligned with the so-called *mechanistic reasoning*, which occurs when students develop explanations that are plausibly aligned with concepts in a particular domain, even if they contain inaccuracies (Southard et al., 2017). This encourages students to draw on their own ideas of explaining phenomena rather than to focus on memorizing the "correct answers" given by their teachers.

There are several similarities between the NRC framework's mechanistic reasoning approach and the process students go through when facing challenges in Plethora. We plan to use the visual infrastructure of Plethora and its scenario-based paradigm and execution mechanism to model various scientific phenomena. The learning process will include challenges to be solved, mainly to get familiar with the scientific terms and basic behaviors, followed by a phase where students build models of the phenomena and try to explain different facts by executing their models.

Introducing the CPS approach for teaching science is a challenge but we believe that with Plethora and the scenario-based approach, we can achieve the goals of the NRC Framework for K-12 Science Education, while at the same time developing students' CPS abilities. Recently, the Israeli Innovation Authority acknowledged this challenge and has decided to support our research.

Acknowledgement

We thank Livnat Ben-Hamo, whose Master's thesis, supervised by three of the authors, forms the basis of the in-class study subsection.

References

Armoni, M., Gal-Ezer, J., & Hazzan, O. (2006). Reductive thinking in computer science. *Computer Science Education, 16*(4), 281-301.

Ben-David Kolikant, Y. (2004). Learning concurrency: Evolution of students' understanding of synchronization. *International Journal of Human Computers Studies, 60*(2), 259–284.

Braught, G., Wahls, T., & Eby, L. M. (2011). The case for pair programming in the computer science classroom. *ACM Transactions on Computing Education, 11*(1), 2:1-21.

Bruner, J. S. (1960). *The Process of Education*. Boston, MA: Harvard University Press.

Damm, W. & Harel, D. (2001). "LSCs: Breathing Life into Message Sequence Charts", Formal Methods in System Design 19:1, 45-80. (Preliminary version in *FMOODS*'99.)

Delal, H., & Oner, D. (2020). Developing middle school students' computational thinking skills using unplugged computing activities. *Informatics in Education*, *19*(1), 1-13.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education*, *46*(3), 277-296.

Gal-Ezer, J., Beeri, C., Harel, D., & Yehudai, A. (1995). A high-school program in computer science. *Computer*, *28*(10), 73-80.

Gal-Ezer, J. & Harel, D. (1999). Curriculum and course syllabi for high-school computer science program. *Computer Science Education*, *9*(2), 114-147

Gal-Ezer, J., & Zur, E. (2004). The efficiency of algorithms – misconceptions. *Computers and Education*, *42*(3), 215-226.

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 Int. Conf. on Automated Software Engineering (ASE), Antwerp, Belgium, pp. 359-360.

Kahney, K. (1989). What do novice programmers know about recursion? In E. Soloway, & J. Spohrer (Eds.), *Studying the Novice Programmer* (pp. 315–323). Hillsdale, NJ: L.Erlbaum.

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly*, *92*(3), 170-181.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. NY, New York: Basic Books, Inc. Publishing.

PISA (2021). Mathematics Framework. Retrieved December 17, 2020 from http://pisa.e e-wd.org. Schank, R.C., Berman, T. R., & Macpherson, K. A. (1999). Learning by doing. In C. M. Reigeluth (Ed.) *Instructional-Design Theories and Models: A new Paradigm of Instructional Technology*, vol. II (pp. 161-182). Mahwah, NJ: Lawrence Erlbaum Associates.

Tsan, J., Vandenberg, J., Zakaria, Z., Wiggins, J. B., Webber, A. R., Bradbury, A., Lynch, C., Wiebe, E., & Boyer, K. E. (2020). A comparison of two pair programming configurations for upper elementary students. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 346-352.

Wing, J. (2006). Computational thinking, *CACM*, 33-35.