FicWebBoard: A Playful and Collaborative Learning Platform Built for All People and All Programming Languages

Eddy Caron, Nicolas Chappe

FicWebBoard: A Playful and Collaborative Learning Platform Built for All People and All Programming Languages.

Eddy Caron
Univ Lyon, ENS Lyon, UCBL, CNRS, Inria, LIP,
F-69342, LYON Cedex 07, France.

Nicolas Chappe
Univ Lyon, ENS Lyon, UCBL, CNRS, Inria, LIP,
F-69342, LYON Cedex 07, France.

***Abstract*—In this paper we introduce *FicWebBoard*, a learning concept and accompanying online platform that allows teaching programming languages to highly heterogeneous groups of students. Students are given great freedom, in particular in their choice of programming languages, but not at the cost of the quality of supervision as they are still under a certain hidden control. Gamification and the opportunity to participate in a challenge (an optional competition), also help to make the concept attractive to students. This original approach targets students with basic knowledge about programming, and allows them to earn deeper or wider programming knowledge. To validate the approach, we designed a survey to collect feedback from past students after using this approach for seven years.**

I. Introduction

The genesis of this project came from the observation that teaching programming to a very heterogeneous public did not meet the expectations of the students, nor the teachers. Usually, the learning process involves practical implementations of projects in one or two given languages. We propose a completely different approach to programming learning. For that we plan to answer a set of questions. Which language to use to teach programming to students? How to train our students to learn tomorrow's languages (sometimes better, sometimes just more fashionable)? How to manage the heterogeneity between our students? How to make learning fun, and why not challenging?

A. The main concept

Students are provided a dependency graph of exercise sheets for various programming languages, and can answer them at their pace, depending on their interests and their needs. The concept of this course is step-by-step (or notion-by-notion) learning through practical and directed exercises instead of a formal lecture. Each exercise sheet introduces a new concept and includes links to online tutorials about the concept, and students who struggle with an exercise can ask teaching assistants for help. During the teaching process, teachers validate or reject the students' submitted answers, and can help them if needed. Another aim of the project is to build a collaborative platform: we invite teachers and even students to join the contribution team as soon as possible if they have strong knowledge in a given language. They can contribute new exercise sheets that the pedagogical team validates and includes in the platform for the other students. At the end of the semester, students are graded on a project in the language of their choice, and a prize is given to the student who has answered the most exercise sheets. It's important to note than the evaluation (the student grade) is based only on the quality of the final project, and the quantity of exercise sheets is just for the fun of the challenge, and for the prize.

B. The platform

The concept introduced previously is supported by a platform that takes the form of a website, on which students can answer exercise sheets and get feedback from teachers. The platform also enforces some pedagogical constraints. We have been using this platform in the context of the Master's program degree for several years and our course is well-received by our students. The departure from the classical model of giving lectures on a specific language allows beginners and experts alike to gain new programming knowledge.

After an overview of the related work (Section II), we detail in Section III the approach through the prism of objectives and constraints. In Section IV we introduce the platform. Building on the previous sections we relate our observations based on seven years of experience and feedback from students in Section V.

II. Related work

Our work is at the intersection of several different objectives and concepts.

a) Teaching programming: In the active research topic of programming education, many initiatives are focused on introductory programming courses [7]. By contrast, our students already have basic knowledge of programming and can already implement algorithms with e.g. functions and loops.

b) Competitive programming: Some programming courses make use of competitive programming. For instance, in [10], students submit programs in C++ or Java to answer a problem, and it is accepted if it passes a set of secret tests. It also cites some of the many non-academic competitive programming platforms. As for [11], it proposes a platform with automatically graded programming exercises in the spirit of competitive programming, and some gamified elements. Exercises are separated in different categories, and students can tackle the ones they want, in any order, to further their understanding of a C programming course.

While the classical competitive programming setting can give students a lot of freedom in their choice of programming language, automation has a cost: unit tests should be carefully designed to cover all cases and ensure that the desired concepts and data structures were used in the student's answer.

c) Gamification: Gamification is increasingly used to increase engagement in engineering courses and particularly computer engineering courses [8]. Many online platforms have been developed for that purpose, many of which focus on teaching the fundamentals to beginners, often through visual programming [13]. Some other existing platforms are focused on intermediate or advanced programming courses for higher education, with some gamified elements. [12] adds badges and a leaderboard to an advanced computer engineering course and observes that badges were well-received, but the leaderboard was seen less positively by students. [1] proposes an online platform to complement a programming course, with small programming challenges that give experience points. Evaluation is mostly automated, except for some free text questions that are peer-reviewed. [6] proposes a platform for programming education with quizzes, short textual questions, and code to be completed, the latter being manually evaluated. Gamification takes the form of points and badges.

Several studies report positive effects of gamification in software engineering courses. [2] experimented with a combination of individual quizzes and group quests that both contribute experience points, with mostly positive effects on students. [5] tells about points-based and story-based gamification experiments for an advanced software engineering course, and reports an increase in motivation from students.

Thus, some gamification can be desirable to make a course more engaging. However, these platforms do not necessarily give more freedom to students, or they do it by relying more on automation. But in the context of competitive programming or gamification alike, the lack of human feedback from teachers can be discouraging to some students, especially lower-performing students [3]. Gamification, coaching and heterogeneity are three key aspects of *FicWebBoard* that will be further detailed in the next section.

III. Objectives and constraints

The goal of this pedagogical project was to propose an all-in-one solution to answer point by point a set of objectives and constraints.

Learning: How do you learn a new programming language? Following a couple of slides from a teacher in an auditorium? Getting the appropriate book? Finding a good tutorial through the Internet and trying it? Seeking help from a friend or from a community of developers? All these methods are valid and we don't want our students to be restricted to a specific one. In our platform, everything is a question: an exercise sheet with a couple of questions allows to introduce a new knowledge. The idea is that to provide a correct answer to all the questions, a student has to reasonably understand the relevant concepts. The student is probably not yet an expert on the new concept/information but they know that it exists and they are able to use it to answer basic questions. They will probably be able to reuse it later. There are no lectures and the exercise sheets do not include courses, but it starts with links to online resources about the relevant concepts. Thus, students are free to learn in their own way (or to fast forward, if they already know the concept).

Gamification: The platform should be attractive. The initial inspiration came from video games. Why do we get hooked on a video game? Because it is simple to understand and easy to win, at least at the beginning. Then the difficulty increases as the player becomes an expert gamer. We have to find the right balance between motivating challenges and their feasibility. Similarly to a video game we have different challenges. First, each exercise sheet can be seen as a video game level and after a series of exercise sheets the project is the final boss. On their way, the students also have to collect some ⋆ (see paragraph Evaluation). The second challenge is not mandatory but students can participate in the "Hall of Fame" and each week the top five is given. See the Contest paragraph for more information.

Simplicity: Directly linked with the attractiveness, simplicity is a key to success. We hence have chosen to split exercise sheets following a basic rule: one concept corresponds to one exercise sheet. Sometimes it is difficult to do so without impacting the exercises that can be proposed. For example in C, one exercise sheet introduces loops and one exercise sheet introduces arrays. As we can imagine, in any order, the first exercise sheet suffers from the lack of notion to provide many exercises. We still stick to this approach, and we will give more complex exercises later when both concepts have been introduced.

Diversity: Another problem has motivated the development of *FicWebBoard*. The usual way to teach programming is to focus on one or two languages. Which language is the best? Which kind of language? Imperative, structured, procedural, object, typed, untyped, functional, logic, etc.? Everybody has their personal answer but no way to find a consensus. The right answer is "it depends". It depends on your needs, it depends on what you plan to do and the context. This is why we decided to provide a large range of opportunities, with, as of now, about 20 different languages. This is an ever-growing count, as contributions from Teaching Assistants and students are welcome (refer to the Upgradability paragraph for more information). Thus students are free to build their own learning

path. Obviously, this freedom is under a certain control from the coach (see the next paragraph).

Coaching: Our aim was not to build a fully automated platform that replaces human teachers, as automated feedback cannot fully replace human feedback, which is especially valued by lower-performing students [3]. Rather, we wanted to provide a simple framework to automate some organizational matters: keeping track of a student's progress, communicating exercise sheets to students, etc. This means we kept the interactivity and the link between student and teacher. Teaching assistants act like coaches: each of them follows a set of students and they have a good understanding of the real efforts made by the students (useful for the evaluation as discussed in the Evaluation paragraph). We want to have a way to provide advice as often as needed, to prevent students taking bad programming habits. Thus each submitted exercise sheet must be validated by the student's coach (see Section IV).

Heterogeneity: Another problem we dealt with was our highly heterogeneous groups of students that have vastly different backgrounds and training. Our course was originally built for the Computer Science department, but some students from other departments (Mathematics, Physics, Biology, Economy) are occasionally joining it, reinforcing this heterogeneity. The simplicity of the exercise sheets allows to absorb part of this problem (see paragraph Simplicity). The beginners discover gradually a language. The intermediate students check and complete their knowledge. Even expert students must respect the order of the exercise sheets and go through basic exercises first. But from our experience this is not a problem for them because they solve the exercise sheets at a high speed. For a given concept/sheet the amount of questions is low enough that they don't feel like they are losing too much time. Moreover, this kind of student usually participates in the contest race (see the paragraph Contest).

Upgradability: As introduced previously it's very important to provide a high quality and diversity of resources. We decided to build the platform under the collaborative paradigm. Over the years, the Teaching Assistants who coach the students are invited to create new exercise sheets or to improve existing ones. Moreover the students are also invited to compete for the title of contributor of the year. They can therefore submit new exercise sheets which are then validated by the teaching staff before being added in the platform. It is often very rewarding for the students of the current year to see their classmates solving the problems of the exercise sheet they have created. This key form of motivation encourages creativity from students, sometimes giving rise to atypical exercises such as a thorough set of six exercise sheets for the Brainfuck language. Another way to improve the quality of the exercises is the feedback of the students, that they can give thanks to a mandatory question in each exercise sheet:

"According to you, in the context of this exercise sheet, which notion(s) would be interesting to address through an exercise? If you do not have anything specific to say, simply indicate "CLEAR"."

Then the teaching staff can take into account the comment or the suggestion and update the exercise sheet for the next students (even in the current class).

Evaluation: It is important to know that not all of the answers given for each exercise sheet are part of the evaluation. They are simply milestones for learning and a way for the teacher to follow the student's progress and efforts. At the end of the series of exercise sheets for a language, a project or a set of projects are proposed to the student and they are evaluated on the quality of the project. The score is based on a grade out of 20. Nevertheless a couple of rules are added to drive the evaluation.

- If the student fails to get four ⋆ they have a 5-point penalty. A ⋆ is a milestone that usually corresponds to a "beginner" level in a programming language, it is very easy to get them.
- If the student fails to send the project at the end they get a 5-point penalty and they are evaluated based solely on the exercise sheets they submitted. This usually results in a non-validation of the course.

Fig. 1. The trophy given to the "Best Coder" of the year

Contest: Beyond the evaluation we organize a contest named "Just for Fun" to increase motivation, fun and competitive spirit. The ranking is only based on the quantity of solved exercise sheets. Each week before the session a teacher provides the "Hall of Fame" by e-mail, in which a ranking with the Top 5 programmers is given. It is out of the question to give the complete ranking so as not to deprecate those who are beginners or simply not interested in the contest. Indeed, it has been reported that leaderboards can have a negative impact on students [4], and be perceived negatively by them [12]. In addition to "the best programmer of the year" we added other categories to the contest: "the best team of the year" (all the student under the supervision of the same coach), "the best contributor of the year" (based on the amount of new exercise sheets written by a contributor). We even formalized this competition with a prize-giving ceremony and a pizza party sponsored by the Computer Science department. The best programmer of the year receives a trophy (see Fig. 1). The ceremony for year n takes place at the beginning of year

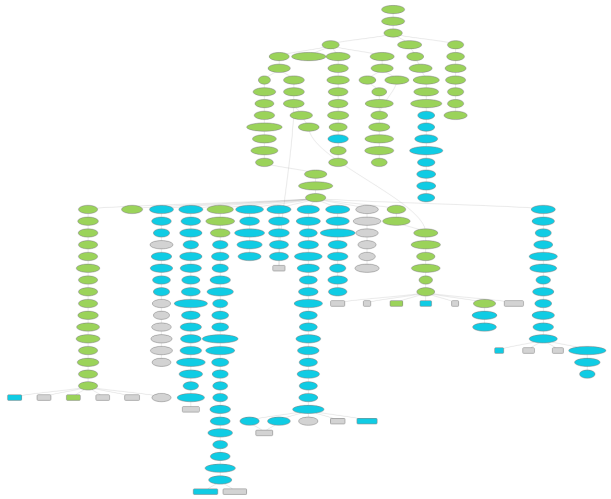

Fig. 2. The full dependency graph, zoomed out.

$n+1$, so that the new students can attend the ceremony of the previous year and get motivated for the contest.

IV. The platform implementation

The application of this learning concept is eased by a supporting website based on the custom software *FicWebBoard*.

It allows to access the various exercise sheets (more than 200 as of 2023), represented as a dependency graph, and implements various features for students and teachers. The (zoomed-out) dependency graph is represented on Fig. 2.

A. The dependency graph

An exercise sheet is made of two kind of questions: *text* questions, that can be answered as Markdown, and *code* questions, that can be answered in the relevant programming language. In both cases, the text field is syntax highlighted. The sheets also contain metadata on their difficulty and recommended readings before answering it.

At first, students can only access the exercise sheets for Markdown, as it is how text answers are formatted on the website. They can then access introductory exercises to the use of terminal emulators and shells, before reaching more varied exercise sheets.

Exercise sheets for a given language are usually laid out as a linear graph, with the last sheet giving access to different options of end-of-semester projects for the language. A simplified dependency graph between the various languages is depicted on Fig. 3.

Of important note is the dependencies of most programming languages to Shell and C, making them basically mandatory. This is due to the fact that our students generally have little programming background, but basic understanding of systems programming is essential for the continuation of their Computer Science studies. As the only programming course of the Master's program of our school, this is the most appropriate time to introduce our students to the use of the Shell and to C.

The Shell introduction includes 3 exercise sheets, covering basic use of a terminal emulator, and various classical POSIX commands, partly relying on pre-existing teaching material [9]. As for the introduction to C, it includes 13 exercise sheets, covering basic features of the language and, for the three last ones, pointers. The very last exercise sheet on pointers is more thorough than regular sheets, and demands to write a few hundred lines of code, with non-trivial operations on linked lists. Because of the higher complexity of these exercises, this exercise sheet is probably the most time-consuming one for students (excluding final projects). This is also very time-consuming to correct for Teaching Assistants, so we recently introduced a script to automatically run students' code on a number of tests. This does not replace manual evaluation however: as the last mandatory exercise sheet on C, precise feedback to students is essential.

B. Features for students

Students can explore any branch of the graph they have unlocked, with a few rules to respect:

- To download an exercise sheet, they should have already fully answered all its dependencies and submitted them on the website (the exercise sheets do not need to have been validated by a Teaching Assistant however). We do not allow any exception to this rule, meaning that students who already know a language still have to do the basic exercises. There are two reasons for that. First, students do not necessarily have a clear understanding of their level of expertise and of the best practices. Second, the vast majority of exercise sheets are short, asking students to write a few dozen lines of code at most, so a knowledgeable student will not lose too much time answering the basic exercises.
- If the submission of an exercise sheet has been rejected, this locks the progress of the student in the graph and they have to submit a new version and wait until their coach validates it before submitting answers to the subsequent sheets. However there is no limit to how many times a rejected exercise sheet can be re-submitted.
- Students can only submit five levels deep of exercise sheets, they cannot access the subsequent exercises until the previous ones have been validated by their teaching assistant. For beginners, this encourages them to take their time to really understand the programming concepts introduced in each exercise sheet, and this prevents them from persevering too far in a wrong direction if they misunderstand a concept or follow bad practices. For more knowledgeable students, this encourages them to explore the graph more widely instead of rushing through their favorite language.

Students can see in a sidebar how many of their exercise sheets are currently waiting for feedback, or have been validated or rejected. They can also see their progress on the requirements for the validation of the course (see Section III) thanks to a counter in the sidebar. Sheets worth a ⋆ and

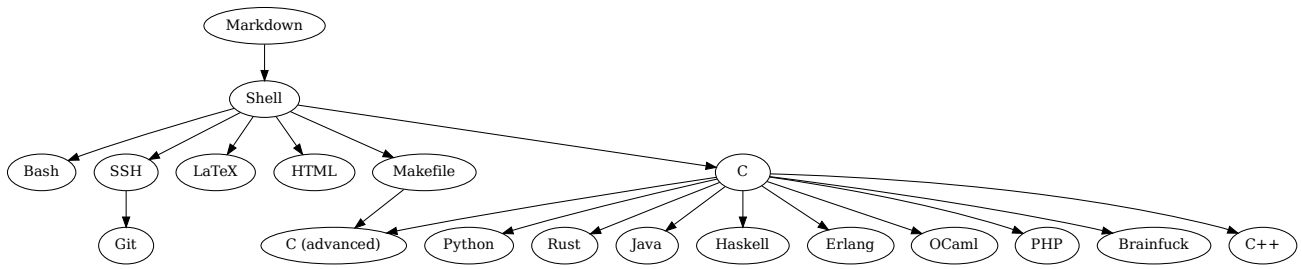

Fig. 3. Simplified dependency graph. The dependency of most languages to C is not strictly needed, it is a way to force our students to learn C because it is important for other courses.

projects sheets are visually distinct from the other ones in the graph.

C. Features for Teaching Assistants

Each student is assigned to a teacher that follows their progress during the semester. As of 2023, each teacher supervises a group of 6 to 12 students.

To validate (or reject) students' work, Teaching Assistants can use a graph view that shows in different colors exercise sheets that have submissions in need of attention. Alternatively, there is also a sortable and filterable list view of all the submissions of the group. Similar list views allow to manage students and to manage exercise sheets.

To evaluate submissions, Teaching Assistants can look at text and code answers online, or download them as an archive. Most exercise sheets come with a fully answered version to help Teaching Assistants in their evaluation. Accompanying scripts partly automatize extraction and compilation of submitted code.

When validating or rejecting a submission, teachers can write a comment that is sent by e-mail to the student. Similarly, Teaching Assistants are notified by e-mail when a student needs particular attention: when a previously rejected exercise sheet has been submitted again, and when the buffer of five exercise sheets submitted on the same branch becomes full.

Teaching Assistants can also check some statistics about their students, notably their ranking amongst all the students following the course. This allows to identify students lagging behind who may be in need of help.

V. Experience Feedback

The available exercise sheets and the dependency graph have evolved over the years. The C exercise sheets have seen major updates, based on feedback from students, and many new exercise sheets for other languages have been written by teaching assistants and students.

A. Statistics

The frequent additions of new exercise sheets seems to have a positive effect on students, as they tend to submit more and more exercise sheets over the years. Figures 5 and 6 respectively show the average and the highest amount of submitted exercise sheets per student each year. The highest number of submitted exercise sheets at a given year is usually close to the total number of available exercise sheets at the time, excluding final projects.

These figures have been steadily increasing since the first edition in 2015, with a drop in 2020 during the Covid-19 pandemic and its forced remote teaching. As an online platform, *FicWebBoard* did not need any adaptation for remote teaching, but we of course had to suspend in-person classes with coaches. This experience highlights the crucial role of the in-person interactions between student and coach, which contrasts with fully-automated learning platforms.

B. Heterogeneity

In our experience, *FicWebBoard* is indeed effective for addressing highly heterogeneous groups of students. We identified a few typical student profiles:

- Some students with moderate interest and little experience in programming do the mandatory (Markdown, Shell, basic C) exercise sheets, and move on to Python, which they already know a bit. To satisfy the constraint of four ⋆, they also submit a couple of exercise sheets in another branch of the graph. They usually do not completely explore the branch but stop after getting the star.
- Students with more interest in programming tend to try a few languages and explore at least one in depth. They may settle for their favorite language and further their understanding of it, or decide to learn a new language.
- Each year, a few students get hooked on the game, or want to compete for the title, and submit nearly all available exercise sheets (about 200 at the time of writing), learning a lot of new concepts.

We believe this shows that *FicWebBoard* perfectly fits for an academic context of great heterogeneity: beginners learn C and deepen their knowledge of Python, intermediate students learn in depth a language, and advanced students can get a wide picture of programming paradigms and tooling (Makefile, Git, etc.).

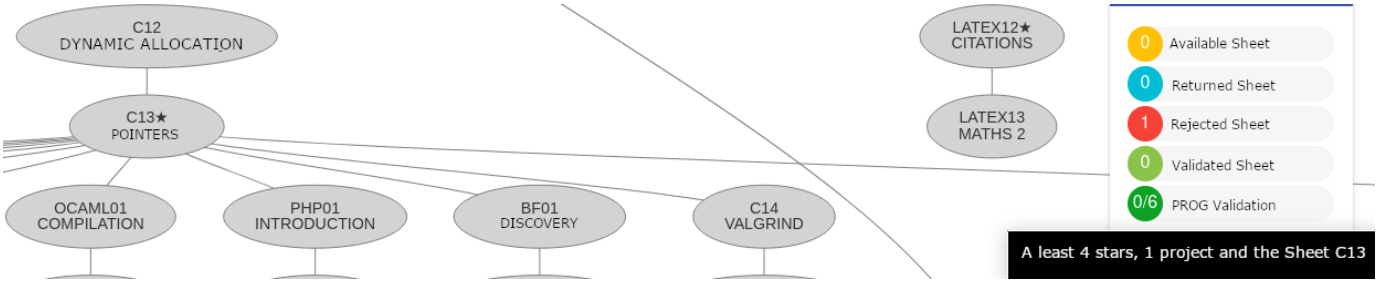

Fig. 4. Part of the user interface for students. Each node represents an exercise sheet. The sidebar shows the counts of unlocked/submitted/rejected/validated exercise sheets, and the progress on the course requirements.

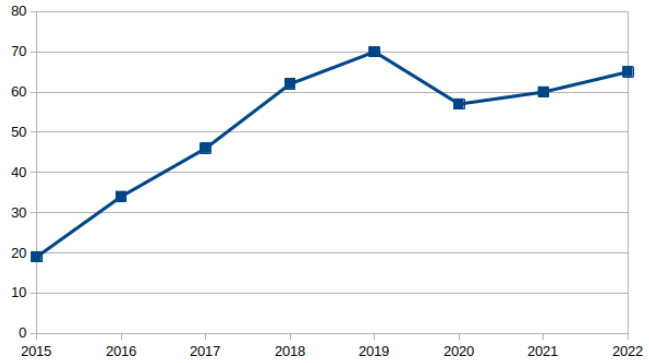

Fig. 5. Evolution of the average number of submitted exercise sheets per student. The low figure in 2015 is in part because students were working in groups of two that year.

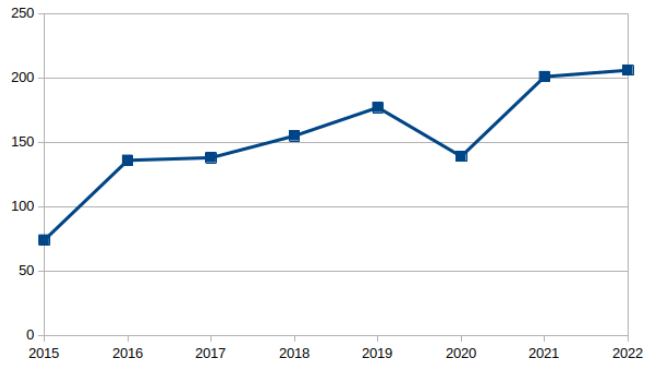

Fig. 6. Evolution of the year's highest number of submitted exercise sheets.

C. Institutional context

The concept and platform this paper introduces has been developed at and used for seven years at École Normale Supérieure de Lyon (ENS de Lyon) in France, to teach programming notions to first year students (*L3*, third year of higher education) of the highly selective master's program of the computer science department. The course is mandatory and takes place during the first semester.

The vast majority of our students come from "*classes préparatoires MP*" (Mathematics and Physics), and have some experience implementing algorithms in Python and OCaml. Most of them are interested in theoretical computer science and plan to pursue an academic career, with a minority that have a strong interest in programming or software engineering.

In this particular context, there was a need for a platform that allows teaching programming to a heterogeneous group of students, that led to the development of *FicWebBoard*. However we believe this teaching concept is applicable to other contexts in other institutions.

Following recent changes in the French higher education system (the introduction of "*classes préparatoires MPI*" that focus more on computer science) we expect that about half our students next year will already have learned C programming, causing further heterogeneity. Thanks to the high adaptability of *FicWebBoard*, we will not need to significantly adapt the programming course to these students.

For use of *FicWebBoard* in another institution, we can foresee two possible adaptations that would be needed. First, we expect that adaptations of the course constraints (the number of stars to get, mandatory languages, etc.) would be needed. On the software side, the shape of the graph can be changed, but there is no simple way to change the course constraints for now. Then, there is also a language barrier as the user interface, and part of the source code, are in French. Overall, these are minor drawbacks, and we believe the *FicWebBoard* concept is applicable in many other institutional contexts. And beyond this point, we could study how we can reuse the concept and the platform to teach other topics (even if some parts are designed for learning programming languages, *e.g.,* the capability to enter an answer in Markdown with a preformed programming language).

D. Feedback from students

We created an anonymous survey for computer science students who attended the course over the 7 previous years. 45 full answers were registered. Table I details the distribution of year of study of respondents. Most students from 2019-2020 or before are no longer enrolled in our institution and were thus harder to reach.

Students reported having learned an average of 3.47 new programming languages, and 53% of students reported having made progress in the languages they already knew. However, these figures are self-reported and students may have different ideas of what "learning a new language" means.

Year	Respondents
2016-2017	1
2017-2018	3
2018-2019	1
2019-2020	1
2020-2021	8
2021-2022	11
2022-2023	20
Total	45

TABLE I
RESPONDENTS PER YEAR

On the concept: Then, a series of questions asked the students to evaluate different aspects of their learning experience, on a scale from 0 to 6, values towards 0 indicating a strong preference for a more conventional course with lectures, practicals, and a final project, and values towards 6 indicating a strong preference for the *FicWebBoard* approach. Fig. 7 sums up the answers collected on the 6 criteria (for a description of the criteria, refer to Section III). Students from 2016 to 2020 are only counted in the general average due to a low number of responses.

On the contest: Of the 47% of respondents who took part in the competition, two thirds had fun. Only 7% of respondents found the weekly hall of fame announcement "stressful".

On the user experience: Students were asked to rate their experience with the *FicWebBoard* online platform on a scale from 0 to 20. The average observed rating is 14.5/20. The satisfaction seems to increase with the evolution of the platform: the average for the year 2021-2022 is 14.7, and the average for the year 2022-2023 is 15.1.

On the evaluation: 44% of students expressed a positive opinion of the grading system, while 18% expressed a negative opinion. A question on the $\star$ constraint elicited similar answers: 49% of students expressed a positive opinion on this requirement, while 13% expressed a negative one.

Discussion: On average, respondents showed a slight dislike for the coaching aspect of *FicWebBoard*. Especially for this criterion, there are large variations between the years. The gaps in the ratings between 2020-2021 and 2021-2022 may be due in part to the Covid-19 pandemic. Indeed, during the 2020-2021 academic year students attended the course remotely and could not directly interact with their teaching assistants. Another possible cause is the different teaching styles of teaching assistants, as they come and go over the years. In any case, we acknowledge the slight dissatisfaction with the coaching aspect and we will try to improve on it in the next years. By contrast, the user experience, the diversity of programming languages and the gamification were particularly well-received. Finally, on average the students did not rate the approach as particularly effective or ineffective to address heterogeneity or to improve learning, nor did they have strong opinions on simplicity. However the coaching, diversity and heterogeneity criteria were polarizing, as the standard deviation for these is approximately 1.9, while it ranges from 1.4 to 1.6 for the other criteria.

Based on this feedback we are proud to see that the diversity is a strong point and the gamification is clear for students. The overall user experience evaluation (14.5/20) is more than we expected and encouraging for the next version. We were surprised to discover that the evaluation was well accepted. The optional contest has been well assimilated by students for what it is: just a fun game.

VI. CONCLUSION

In this paper we propose a platform dedicated to programming language learning for students with high heterogeneous background from beginners to experts. This project is a collaborative project that keeps improving in quality and quantity of programming languages. It allows a regular follow-up of the students too. And the optional competition works like a game, which is appreciated by students either as participants or watchers. After seven years of usage with Master's students and the success of the approach we get a concrete proof of the efficiency of the concept. Now we're thinking about how best to open up the platform to a wider audience and share the code as an open source project.

ACKNOWLEDGEMENTS

A special thanks to Romain Liautaud who contributed the first prototype of *FicWebBoard*. So many thanks to all contributors for the platform and the exercise sheets (the list is too long): colleagues, Teaching Assistants and students. A thanks to Eric Thierry from the Computer Science Department who believed that this crazy project could be possible. And last but not least, thanks to each and every student, one by one, for being the most important part of this adventure.

REFERENCES

[1] Markus Fuchs and Christian Wolff. Improving programming education through gameful, formative feedback. In *2016 IEEE Global Engineering Education Conference (EDUCON)*, pages 860–867, 2016.

[2] Patrícia Gomes Fernandes Matsubara and Caroline Lima Corrêa Da Silva. Game elements in a software engineering study group: A case study. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, pages 160–169, 2017.

[3] Beate Grawemeyer, John Halloran, Matthew England, and David Croft. Feedback and engagement on an introductory programming module. In *Proceedings of 6th Conference on Computing Education Practice*, CEP '22, page 17–20, New York, NY, USA, 2022. Association for Computing Machinery.

[4] Michael D. Hanus and Jesse Fox. Assessing the effects of gamification in the classroom. *Comput. Educ.*, 80(C):152–161, jan 2015.

[5] Isabel John and Tobias Fertig. Gamification for software engineering students - an experience report. In *2022 IEEE Global Engineering Education Conference (EDUCON)*, pages 1942–1947, 2022.

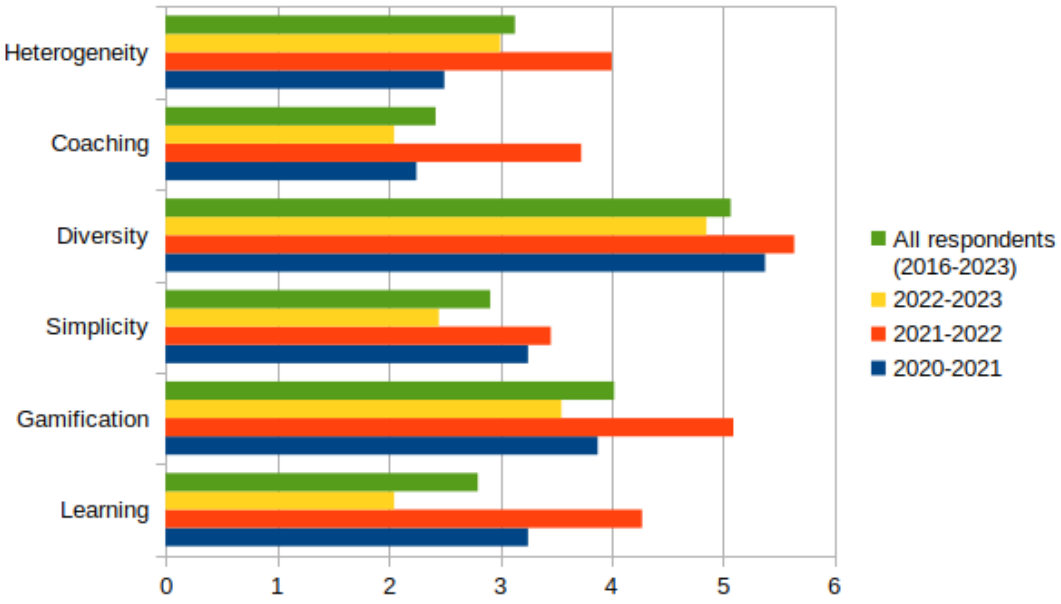

Fig. 7. Ratings (0-6) from students who answered the survey, grouped by year

[6] Balraj Kumar and Kanika Sharma. A gamified approach to achieve excellence in programming. In *2018 4th International Conference on Computing Sciences (ICCS)*, pages 107–114, 2018.

[7] Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcão. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, 62(2):77–90, 2019.

[8] Marek Milosz and Elzbieta Milosz. Gamification in engineering education – a preliminary literature review. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 1975–1979, 2020.

[9] Matthieu Moy. Efficient and playful tools to teach unix to new students. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, page 93–97, New York, NY, USA, 2011. Association for Computing Machinery.

[10] Andrés F. Pineda-Corcho and Julián Moreno-Cadavid. Proposal of a gamified virtual learning environment for computer programming courses. In *2017 IEEE Global Engineering Education Conference (EDUCON)*, pages 1671–1675, 2017.

[11] Giuseppina Polito and Marco Temperini. A gamified web based system for computer programming learning. *Computers and Education: Artificial Intelligence*, 2:100029, 07 2021.

[12] Mauricio Ronny de Almeida Souza, Kattiana Fernandes Constantino, Lucas Furtini Veado, and Eduardo Magno Lages Figueiredo. Gamification in software engineering education: An empirical study. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, pages 276–284, 2017.

[13] Adilson Vahldick, Antonio José Mendes, and Maria José Marcelino. A review of games designed to improve introductory computer programming competencies. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–7, 2014.