Implementing GameCodes in CAFÉ 2.0

Auteur : Malcev, Lev
Promoteur(s) : Donnet, Benoît
Faculté : Faculté des Sciences appliquées
Diplôme : Master en sciences informatiques, à finalité spécialisée en "computer systems security"
Année académique : 2023-2024
URI/URL : http://hdl.handle.net/2268.2/20384

GAMECODES IN CAFÉ 2.0: IMPLEMENTATION AND ANALYSIS OF EARLY RESULTS

LEV MALCEV

Master's thesis completed in order to obtain the degree of
Master of Science in Computer Science

University of Liège
School of Engineering and Computer Science

Academic year 2023-2024

SUPERVISOR:
Prof. Benoit Donnet

Lev Malcev: *Gamecodes in CAFÉ 2.0: Implementation and Analysis of Early Results,* Master's thesis completed in order to obtain the degree of
Master of Science in Computer Science, © Academic year 2023-2024
SUPERVISORS:
Prof. Benoit Donnet

ABSTRACT

This master's thesis presents the implementation of an e-learning platform, called "GameCodes", integrated into the CAFÉ ecosystem at the University of Liège, an already existing e-learning project. The aim of this platform is to enhance student engagement and learning outcomes in computer science through gamification and detailed learning analytics (LA).

The work described herein includes the design and deployment of GameCodes in two first-year courses: INFO0946 (CS1), for computer science students, and INFO0061 (CA1), for civil engineering students. A total of 355 students were targeted, with 47 actively using the platform over a three-month period, providing initial data for analysis without any reported issues.

This work focuses on the data collection mechanisms, the challenges faced during the implementation, and the analysis of preliminary learning analytics to evaluate student participation and performance on GameCodes.

Key contributions of this thesis to the scientific community of computer science and computer science applied to pedagogy include the design and implementation of GameCodes, the deployment and use by students, the collection of detailed interaction data, and the public availability of the source code to encourage further research and development. Moreover, this thesis outlines how these analytics can be utilized in the future to help students at risk of failure by creating practical means and solutions, such as student profiling and improved educational methodologies.

Despite the limited dataset, we explain how GameCodes, when used in conjunction with other tools in the CAFÉ 2.0 ecosystem, have the potential to provide significant insights into student behavior and learning processes. Future work will focus on enhancing the gamification elements, improving the microservices architecture of the platform, and conducting further research to better understand the impact of this platform on student success.

In conclusion, this thesis lays the groundwork for future research into the application of gamification and learning analytics in higher education, aiming to support students more effectively and provide educators with valuable insights into their teaching practices.

ACKNOWLEDGEMENTS

First of all, I would like to warmly thank Prof. Benoit Donnet for his guidance, his support, his trust, and for believing in me. Thank you for giving me the opportunity to work with you for almost five years now, as well as for offering me such an interesting thesis subject.

I would also like to thank Géraldine Brieven, who in addition to being a wonderful colleague, has also become a very close friend. Thank you for your support, your presence, and your precious help in moments of doubt.

Then, of course, I would like to extend my warmest thanks to Alexandre Eymaël, my very close friend, who is now like a brother to me. Thank you for your friendship, your sincerity, and for being a great companion in my studies since your first year at our university. Your support has meant a great deal to me – more than you think. Thank you for everything.

I would also especially like to thank Camille Pradelli, who pushed me to try university in 2018. It was you who convinced me to at least give it a go, when I did not think I could. Thank you for reminding me to always aim for the moon because even if I fail, I will land among the stars.

I am also deeply grateful to my friends who have always been there for me since I started university, who have always believed in me, and without whom I would not be the person I am today. Thank you Clara, Brayan, Vanessa, Nawel, Marie-Sarah, Charlotte, Sophie, Damien, and Alyssia. You are the best friends anyone could ever dream of having.

Furthermore, I am very thankful to my student association, the CSS. The creation of the CSS is the project I have been most proud of since I started my studies, and I am very happy to have been able to contribute to the foundation of this incredible community. I am also very relieved to leave it in good hands – you rock, Roxane!

Lastly, I would like to acknowledge Prof. Bernard Boigelot for his help with this work, and for making me really enjoy computer science through all the conversations we have had over the past six years. And I would especially like to thank Thibault Gillis, and Simon Liénardy, for your support and help, and for everything you have helped me to learn, whether it was related to computer science or not.

I will dedicate myself during my doctoral thesis to proving that your support was a wise investment by doing everything in my power to contribute meaningfully to our field.

Lev Malcev

CONTENTS

Part I

INTRODUCTION

This part provides an introduction to the concept of GameCode, as well as a state-of-the-art presentation of similar work already existing in the scientific community. It is intended to provide the reader with the context in which this thesis was carried out and to provide insight into the choices of organization, design, implementation, and data collection that were made as part of this work. This part should also draw your attention to the fact that this master's thesis is a cross-disciplinary study, involving not only the development of IT tools, but also the application of computer science to the field of pedagogy, as well as data analysis on student learning outcomes.

1 CONTEXT

1.1 ORIGINS OF GAMECODES

During the COVID-19 pandemic, it became essential for university teachers to find a way to continue teaching their students virtually. An idea was therefore proposed in 2019 by Simon Liénardy, a former teaching assistant at the University of Liège, as part of Prof. Benoit Donnet's "Introduction to Programming" course, to enable students to continue working autonomously on their course, at home, while benefiting from static feedback and detailed explanations. This proposed idea is the first version of GameCodes.

As described in a paper written by Simon Liénardy and Prof. Benoit Donnet on the subject [17], a GameCode was initially a PDF document of around 50 pages containing a complex programming or algorithmic exercise, and based on the idea of Gamebooks. A *Gamebook*[1] is a work of printed fiction that allows the reader to participate in the story by making choices. For instance, the reader might find themselves in a scenario where they are facing a dragon, and the book would ask them to go to page 100 to fight the dragon, page 120 to get out of the room, or page 80 to open the treasure chest. This way, everyone can have a different conclusion to the story, and more importantly, a personalized journey.

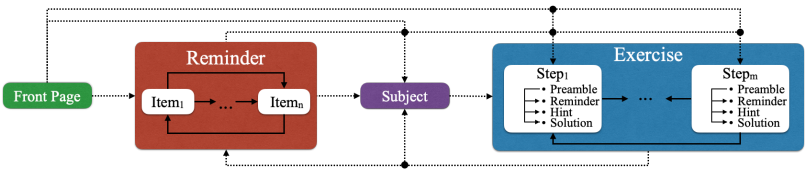


Figure 1: General architecture of any GameCode in its PDF format [17]

In the same way, the initial version of a Gamecode offers students a personalized course. It is divided into five stages (Figure 1): an introduction (front page), a number of global theoretical reminders to the exercise, a statement (subject), and the exercise, composed of resolution steps. Each resolution step (hereinafter referred to as "RS") is itself made up of a sub-statement, theoretical reminders, exercise(s) to solve, and a conclusion to the exercises. The PDF document is structured to contain hyperlinks (see the black dotted arrows on Figure 1) to its other sections, enabling students to read theoretical reminders, or hints if necessary, or to jump straight to the conclusion, or to solve

1 https://en.wikipedia.org/wiki/Gamebook

everything in order. Essentially, this allows them to have a personalized learning path, allowing students who need more time and explanation to take advantage of it, and more advanced students to skip steps if they wish. This idea was very well received by the course students, who expressed their interest in having more exercises in this format.

1.2 INITIAL LIMITATIONS

GameCodes in PDF format have several limitations:

- Everything is static. It is a document in which students can, naturally, navigate, but there is no mechanism for interaction, and therefore no opportunity for students to check their answers other than with the conclusions of each RS;
- No student participation data is collected. Again, given the static nature of this format, there was no mechanism for collecting data on answers, time spent in each section, errors, etc., nor any other learning analytics that might be of interest;
- Encoding new GameCodes is not efficient, let alone portable to other courses. Each GameCode is encoded as LaTeX files, and modularity was possible to some extent by including the same theoretical reminders files in different GameCodes, but when it comes to writing a new one, or porting one to another course, this quickly becomes very tedious, and takes the authors several weeks to write ;
- There is minimal ergonomics. Hyperlinks allow navigation, but students still have to scroll through a 50-page file on their screen, and open multiple browser tabs;
- Also related to ergonomics, paper GameCodes are very wordy and fail to implement an idea of *Gamification* (see Section 1.3), which was one of the author's objectives.

These various limitations prompted Prof. Benoit Donnet to consider a new version that would remedy all these limitations, which is the subject of this thesis.

1.3 E-LEARNING PLATFORM

The aim of this master's thesis is twofold. The first is to create a GameCodes web version, which would overcome the limitations of the PDF version. This web version would have a similar structure to the PDF one, and would allow students to participate in online programming

exercises, with an ergonomic interface, online compilation and a simplistic correction engine. As the correction engine for these exercises is not a priority for this project, a basic form of correction (in the form of a correction with an expected answer or output) will only be available. Students will be able to carry out a variety of online exercises, depending on the course to which a GameCode is linked, such as MCQs, fill-in-the-blanks, forms, binary-to-decimal conversions, and many others.

From the point of view of the teachers, the aim will be to enable them to encode GameCodes in an ergonomic way, while implementing a form of modularity that will enable them to reuse theoretical reminders from one GameCode to another, or even within the same one.

The second objective of this thesis revolves around the processing of data collected by the web interface when students perform exercises. Indeed, while students are working on a GameCode, we need to collect as much data as possible about their interaction with the platform. That is, the time they spend on each portion of text, the answers they provide to exercises, the mistakes they make, the times of day they work, their answer modification history, the items they click on, and other relevant statistics.

An exhaustive list of all these tasks from a more technical point of view, is presented in Chapter 3, Technical goals.

Another important element of this work is the concept of *Gamification*, defined as incorporating game elements in an environment that is not initially intended for this purpose, which is a promising approach in the domain of education [35]. Most of existing virtual teaching platforms (e. g., the ones mentioned in Chapter 2, State-of-the-Art) are fairly simple, and do not offer systems for collecting detailed data on student interactions, statistics of real interest to teachers, or complex tailored exercises, and often suffer from high dropout rates and low completion rates [14]. Gamification is commonly used independently of online platforms, but can also be integrated into them, as is the case with several existing ones (again, cfr. the ones mentioned in Chapter 2)

Offering "GameCodes" to students instead of "online exercises", or even worse, "homework", is not insignificant; it is a first effort to make online teaching more fun, to stimulate better engagement, and greater motivation. However, since this work is only a prototype, gamification efforts will not be paramount. The simple fact of cutting out exercises and allowing students to follow a personalized path is already a step closer to the concept of Gamebook, which is intrinsically fun and entertaining. Other gamification efforts are essentially focused on the presentation of the exercises (e.g., enthusiastic writing style, tutoring, ...), and on the appearance of the website. Other more

promising efforts will be explained for future work in Chapter 9, Future Improvements.

1.4 CONTRIBUTIONS TO THE SCIENTIFIC COMMUNITY

The project delivered by this master's thesis makes several contributions to the scientific community, which are described in this section. These contributions are formulated as a list rather than as answers to Research Questions (RQs), given that this work is essentially implementation work, and that the number of students who have used the platform is far too small to be able to draw conclusions and find correlations on the success of students who have used GameCodes, or on their impact on their June exams.

This work therefore delivers the following contributions:

- We implement the concept of GameCodes in the CAFÉ 2.0 ecosystem, and discuss our design choices;
- We deploy GameCodes in two different courses targeting first-year students in CS1 (INFO0946) and CA1 (INFO0061), allowing a total of 355 students to use the platform over three months without any bug reports;
- We collect and analyze data in the form of *learning analytics* by logging a quantity of interactions with the web platform;
- We provide a methodological approach to understanding the interaction between students and e-learning platforms. This could serve as a foundation for future research in educational data mining;
- GameCodes source code is publicly available to the research community, which promotes collaboration and innovation within the academic community. Other researchers and developers can build upon this work, leading to continuous improvements and new applications;
- We describe how the collected data could and will be used in the future to predict student success, increase students' chances to succeed, and to profile students;
- We discuss the limitations of the current implementation, and propose future improvements.

2

RELATED WORK

Even before the COVID-19 pandemic, numerous proposals for online learning platforms, whether applying the gamification concept or not, have been proposed and studied. The aim of this chapter is to carry out a short systematic study of some of these attempts, to highlight what they have in common with the GameCode concept, their limitations, and what this master's thesis work can remedy, as well as what this work is inspired by

The writing of this chapter is important in the context of this work, because it explains its purpose, mentions the sources from which it draws inspiration, and provides scientific justifications for certain design choices to avoid starting from scratch with this idea and reuse studies on the subject.

2.1 AUTOMATIC FEEDBACK AND PERSONALIZED EXERCISES

GameCodes are directly linked to CAFÉ [7, 18], a project that Prof. Benoit Donnet, his teaching assistants, and I have been working on for the past few years. CAFÉ is an e-learning platform implementing the online submission of graded assignments, called *Challenges*, with automatic correction and relevant feedback based on errors made by students. It also implements a tool for creating graphical loop invariants (GLIs) [6, 19], on which students in Prof. Benoit Donnet's *Introduction to Programming* course base their program construction methodology.

CAFÉ stand for "Correction Automatique et Feedback des Étudiants", in French

GameCodes are now part of the CAFÉ ecosystem, accessible to students from the same website, and will in future be connected to the same correction engine that CAFÉ uses to correct *challenges*. In particular, they will be used to populate CAFÉ's misconception library [6], which consists of a classification of student errors that is intended to be integrated into a dynamic exercise generation system, in the near future.

Beside CAFÉ, many automated systems providing programming exercises were already proposed (e. g., [3, 10, 16, 20, 24, 29]). Most of them apply test-based correction, i. e., student's code is corrected through unit testing. Kumar's Problets [16] enables step by step code execution as part of the feedback. Dodona [25] proposes programming assignments with automated feedback and harness the data to regulate the teaching materials. However, Dodona specializes in practicing coding only (considering different programming languages) in a collaborative environment by allowing students to ask questions on a forum. Overall, the scope of these studies is limited to a specific

focus, and does not include the collection of usage statistics, unlike the work carried out as part of GameCodes.

TARTARE [4] is a more recent study, on automated generation of C pointer statements with automated feedback. However, it could still be improved with an advanced usage of learning analytics, students clustering, better gamification, and personalized exercises.

Although the automatic correction of exercises is not central to this thesis, it was interesting to look at similar work in order to define in which direction CAFÉ's correction engine should be oriented in the future, and to already start thinking about how to structure the data that will be collected by GameCodes in the current state of implementation.

2.2 CLUSTERING BASED ON STUDENT PROFILES

The aim of this sub-section is to give relevance to the statistics gathered by GameCodes, by analyzing some work already done in the past, and exploiting e-learning platform usage of statistics, in a similar way to what GameCodes would allow us to do. This is also a set of ideas for future work on GameCodes, once they are no longer in the prototype stage. Indeed, as explained in Section 7.1, the main aim of gathering statistics in this context would be to enable the classification (or clustering) of students according to their level of understanding of a given course, and in fine, to help them progress from a cluster of academically "weak" students to a more qualitative cluster, in addition to the idea of tailored exercise generation. This task, however, is outside of the scope of this thesis, but is mentioned to motivate the data collected by the platform.

Numerous studies [8, 28, 30, 34] have already been carried out on clustering of students on the basis of their working methods and academic profile, often focusing on predicting their success or failure, rather than using clustering results for exercise generation and pedagogical support. Of particular interest in this context is Cristóbal R. et al.'s study [28], which focuses on predicting students' final results based on their participation in a discussion forum through *Moodle*. This study uses (Educational) Data Mining (DM and EDM) techniques, using traditional classification algorithms as well as clustering algorithms and feature selection methods for their model training and predictions. This work, which enabled them to obtain extremely interesting results in terms of success predictions, could be reused to carry out clustering, not based on forum discussions, but on the students' interaction with GameCodes.

The work of Hoffait A.-S. et al. [13] is also worth mentioning; while they also focus on predicting potential failure using data mining methods on student data at registration (by using random forest, logistic regression, and artificial neural network algorithms), similar

methods can be used in future the context of GameCodes to apply them to their behavior with the platform, rather than the socio-economic aspect of the students.

2.3 E-LEARNING PLATFORMS AND GAMIFICATION

The definition and usage of an e-learning platform varies from one scientific publication to another. In this case, we are going to look at platforms that are not content to simply download course content and offer simple automatic correction (e.g., character-by-character comparisons or MCQs), but that try to provide something new and captivating for students.

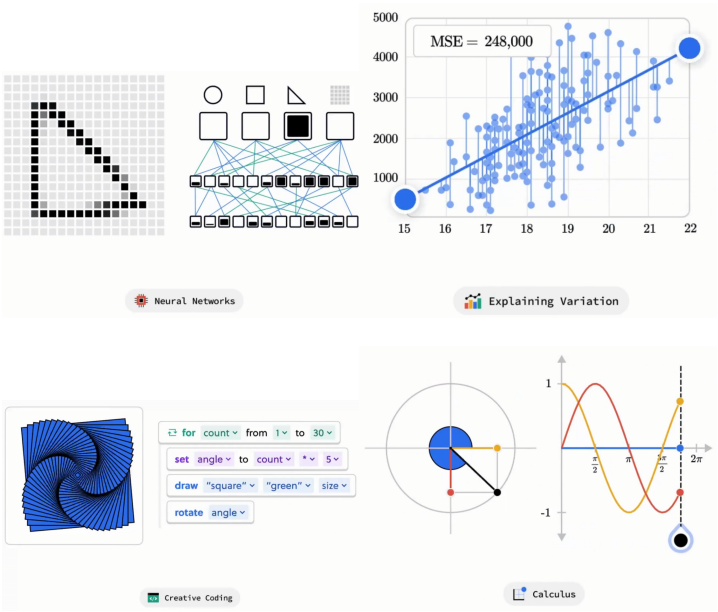


Figure 2: Examples of *Brilliant* exercises

The most notable example in this field is probably the *Brilliant.org* web platform. This e-learning platform stands out as a pioneering example of an e-learning platform that goes beyond traditional content delivery and assessment methods to provide engaging and interactive learning experiences. Founded in 2012, Brilliant has gained recognition for its innovative approach to education, particularly in STEM courses. A recent report shows that the usage of Brilliant.org in these fields have induced significant improvement in the academic performance of students who tried using the platform compared to control groups who have not [26].

Brilliant offers tailor-made exercises based on the subject point (e. g., Figure 2 shows the appearance of exercices related to neural networks, statistics, programming, and calculus). These exercises are certainly designed to be fun and highly ergonomic, but the develop-

ment time for a single exercise is considerable, and new ones can hardly be developed, as it is firstly a commercial product, and its development is only managed by its own developers, and is not open source. Also, most of the courses on offer, such as those on astrophysics, are aimed at the popularization of science, rather than an in-depth understanding of the subject matter.

Another problem is that its subscription-based model poses a financial barrier to entry for students and educators, and integrating this platform into existing courses can be challenging, as it lacks seamless integration and requires technical expertise. Furthermore, it primaly focuses on delivering dynamic course content rather than providing comprehensive learning management features, which limits its potential in a university setting.

However, since one of Brilliant's strong points is its ergonomics and attractive design, GameCodes take its inspiration from this platform.

Another prominent example is the *Khan Academy* platform, which is very similar in concept to Brilliant, and offers equally positive results for student learning [32] , but suffers from the same shortcomings as the former, and is also criticized for its ergonomics [33], and the way in which, as it is the case for Brilliant, exercises can hardly be integrated into it.

The references to the difficult integration of exercises for these two sub-mentioned platforms is an important point, given that this is a strong point of GameCodes, which will have customized, on-demand exercises based on the courses they cover. Given that this is a platform linked to university courses, as part of a research project, and not a commercial project, most of the limitations mentioned for these two platforms will not be relevant in our context of work.

Finally, on Figure 3 is a table that compares different e-learning platforms, including CAFÉ, Brilliant.org, and Khan Academy, as well as others such as Classcraft, Quizizz, Kahoot, Minecraft: Education Edition, Duolingo, and E-Campus. This table compares these different platforms according to the criteria that the GameCodes project seeks to address, initially in a fairly straightforward way as part of this master's thesis, and then in the longer term as part of a doctoral thesis. The criteria we focus on are :

- *Gamification*: whether the platform implements gamification or not ;
- *Personalized learning experience*: whether the platform offers a personalized experience based on the user's profile ;
- *Teacher feedback tool*: whether the platform offers real time feedback about students to a teacher (or administrator) ;
- *Simple course integration*: whether it is possible to easily integrate new exercises into a course (i. e., not having to code new modules from scratch for an exercise type) ;

- *Strong learning analytics*: whether the platform offers advanced analysis of user statistics ;
- *Free access*: whether the platform is free or fee-based.

The value of these criteria can be :

- *Yes*: the feature exists ;
- *Limited*: the feature is there in a way, but can be improved ;
- *No*: the feature does not exist.

Feature / Platform	CAFÉ	Brilliant.org	Khan Academy	Classcraft	Quizizz	Kahoot	Minecraft : Education Edition	Duolingo	E-Campus
Gamification	Limited	Yes	Limited	Yes	No	No	Yes	Yes	No
Personalized learning experience	No	Limited	Limited	No	No	No	Limited	Yes	No
Teacher feedback tool	Limited	No	No	Yes	Yes	Yes	No	No	Limited
Simple course integration	No	No	No	No	No	No	No	No	Limited
Strong learning analytics	Limited	No	No	Limited	No	No	Limited	No	No
Free access	Yes	Limited	No	Limited	Limited	Limited	Limited	Limited	Yes

Figure 3: Integration of the points covered by GameCodes, currently or in a future version, and their incorporation into leading e-learning platforms

From this analysis, we can observe that there is as yet no tool available that implements all these key points at once, and so a research project on the principle of GameCodes, as well as a master's thesis, will be of interest in the field of research into computer science applied to pedagogy.

Part II

IMPLEMENTATION

This part provides an overview of the technical choices made during the development of the GameCodes platform, as well as the organization of the work and the implementation of the various components of the platform. The aim of this part is to review the most important elements of the website, explain how they work, justify the choices made and demonstrate the work carried out.

3

TECHNICAL GOALS

The aim of this chapter is to explain the different objectives of this master's thesis from a more technical point of view, with a description of the tasks that were carried out, and a justification of the technology used. These objectives were briefly mentioned in Chapter 1, but will be discussed in more detail in this chapter. This chapter will also present how GameCodes were integrated in the already existing CAFÉ ecosystem.

3.1 AIMS OF THE PLATFORM

The GameCode e-learning platform is a website that allows students at the University of Liège to complete programming exercises online, with an automated correction system and online compilation. This platform contains reviews of course material that are always structured in the same five parts:

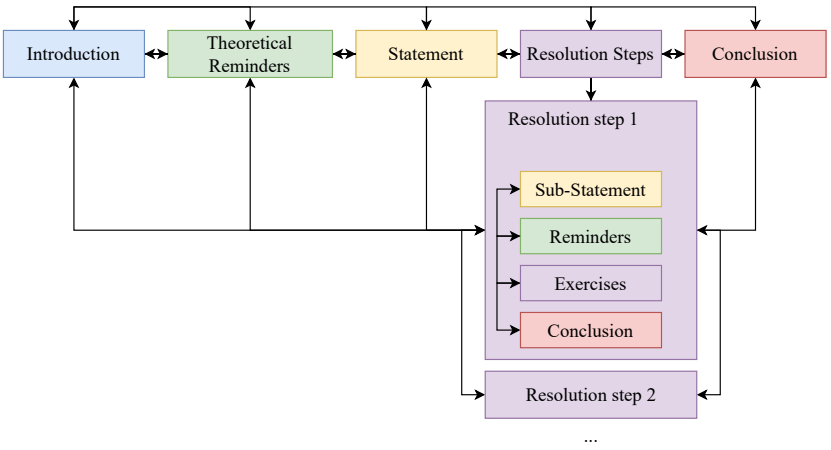


Figure 4: Diagram of a GameCode structure

Figure 4 is a visual representation of the structure of a GameCode. The black arrows represent the links between each part, i. e., it must be possible for a student to "choose their own path" by moving from any part of the GameCode to another one. For example, they could choose to move from the introduction to the resolution steps, or from the resolution steps to the theoretical reminders, etc. The same goes for the inside of a resolution step.

1. Introduction: a few introductory words on how a GameCode works and the content of the current chapter ;
2. Global theoretical reminders: general theoretical reminders about the whole GameCode, which can be found in other GameCodes as well, or within the resolution steps ;
3. Statement: a description of the tasks to be carried out within this GameCode;
4. Resolution steps: the steps to complete a GameCode, which are also broken down into several steps:
   - A sub-statement;
   - More targeted theoretical reminders;
   - Exercises to solve;
   - A conclusion, with a correction of the exercises (if applicable), and explanations on how to solve them.
5. Conclusion: a few concluding words on the material covered in this GameCode.

Students need to be able to navigate easily between the different parts of a GameCode (e. g., consult theoretical reminders easily while solving an exercise), and they need to be able to restart a GameCode whenever they want.

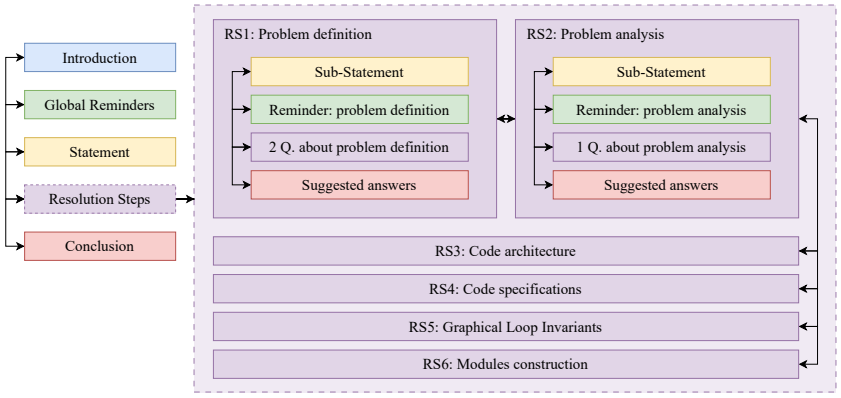


Figure 5: Structure of the GameCode about modular programming

Figure 5 is an example of such structure in practice, from the third GameCode of the INFO0946 course, which is about modular programming. The arrows represent the different paths students can choose to follow, between each part of the GameCode, and within resolution steps. These different chosen paths are part of the collected *learning analytics*, and treated in Section 8.2.3.

In addition, as many interactions with the platform as possible need to be logged. Every time a student changes page, or changes

their answer, submits an answer, makes an error, etc., it all has to be logged. An exhaustive list of the types of log considered is given in Chapter 7.

From a teacher's point of view, they must be able to encode a GameCode using an ergonomic administration interface that respects the imposed structure (the division into five parts), and that allows the same theoretical reminders to be reused within the same GameCode, or in different GameCodes. There is no dashboard for analysing usage statistics in the tasks to be carried out; all the analysis will be carried out a posteriori by independent modules that will take a database dump as input.

3.2 TECHNICAL DESCRIPTION OF THE TASKS

From a more technical point of view, this section describes the overall objectives for the development of the platform. The precise way in which these objectives were achieved is explained in Chapter 6; here, it is more a question of a general breakdown into sub-problems that was carried out before GameCodes were developed.

3.2.1 *From a student's point of view*

- Login using the University of Liège SSO;
- View and interact with the list of GameCodes by course;
- Be able to answer different types of exercise:
  - MCQ
  - Fill in the form
  - Code snippet to complete
  - Complex code snippets to be completed, broken down into different files (e. g., `main.c`, `header.h` ...)
  - Breakdown into sub-problems with inputs and outputs (i. e., what is expected of them in one of their code development methodology courses)
  - Converting binary numbers to decimal
  - Converting decimal numbers to binary
  - Perform a written binary calculation
- Progress through a GameCode by reading sections of text one after the other, and keep track of the progress ;
  One of the strengths of Brilliant.org being its ergonomic GUI[1], the way in which the GameCodes will present each of its parts

1 Graphical User Interface

will be in the form of small portions of text with wide margins between them, for better visibility and to avoid the problem of "too wordy" exercises in the original PDF version.

- Easily navigate between the different sections of a GameCode (introduction, reminders, statements, exercises, etc.);
- Obtain simple feedback for exercises that can be corrected using a character-by-character comparison mechanism;
- View a list of hints by exercise, if needed.

3.2.2 *From a teacher's point of view*

- Have additional permissions compared to students (a special rank);
- Create, edit and delete a GameCode and all its components;
- Write theoretical reminders, separately from GameCodes, and be able to include them independently;
- Adjust the visibility of a GameCode in the eyes of students and define its publication date.

3.2.3 *From the developer's point of view*

Generally, a developer's view of a task list of this type is not given, as these are more implementation detail tasks, and further explanations will be given later. However, here is a non-exhaustive list of the main challenges, which will be developed in greater detail later on.

- GameCodes must be accessible from CAFÉ;
- The website and database must be fully dockerised and portable;
- The virtual machine on which the application is hosted must be secure and communication with it must be encrypted using HTTPS;
- A user authentication system must be implemented, linking the university's SSO and the maintenance of this session in a browser;
- A back-up system must be implemented to minimise the loss of data that will be used for research in the event of an accident;
- The application must be GDPR-compliant;
- The GameCodes application must be able to communicate with CAFÉ's existing correction engine, to correct exercises for which code must be compiled and executed in a sandbox.

3.3 INTEGRATION INTO CAFÉ

As briefly described in Chapter 2, CAFÉ is a project supervised by Prof. Benoit Donnet for more or less 10 years now. When it started, it was called CAFÉ 1.0, and it was a set of Python scripts that were used to give an automated feedback to students' homework, used on Montefiore's *submit* platform, and has evolved into a web application, called CAFÉ 2.0, that is now used every year by first-year students in computer science in the context of their course *Introduction to programming*. The aim of this section is to describe the current ecosystem of CAFÉ, and how the GameCodes were integrated into it. However, the whole CAFÉ project is out of the scope of this thesis, and its description is presented only to provide the context in which GameCodes were integrated. All of the elements described in this section are available on CAFÉ's website, accessible by students and teachers at the University of Liège.

3.3.1 *Context of CAFÉ*

Figure 6 is a visual representation of the CAFÉ 2.0 ecosystem, before the integration of GameCodes. It used to be a web application with different components, which are the following:

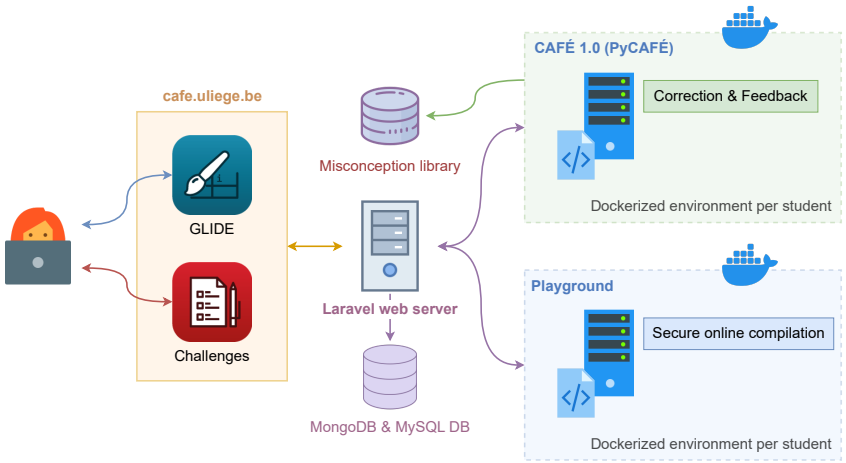


Figure 6: Diagram of the CAFÉ ecosystem, before GameCodes

- *Challenges*: online homework[2], called "challenges" in order to add an aspect of gamification to the platform, to be completed by students, with a deadline. There are five challenges per year that all count for a part of the final grade;

2 https://cafe.uliege.be/challenges

- *GLIDE*: which stande for Graphical Loop Invariant Drawing Editor[3], which allows students to draw graphical loop invariants (GLI), which is a work methodology used in the context of their course, in order to build code upon a loop invariant and a sub-problems decomposition;
- *PyCAFÉ*: a Python correction engine which was started around 2017, as CAFÉ 1.0, by Simon Liénardy, a former phD student, and which is now maintained and developed by Géraldine Brieven, a PhD student at the University of Liège. This engine is used to correct the challenges;
- *Misconception library*: a library of common mistakes made by students, which is used to give feedback to students when they make a mistake, and to collect learning analytics;
- *Playground*: a web page, within challenges, where students can test their code and see the output of their code thanks to a secure and dockerized environment;
- *Admin interface*: an administration interface for teachers to create, edit, and delete challenges, and to see aggregated results of the students, and to be able to see their code.

When these different components were more or less existing, work had to be done on the online integration of all these components into a single platform, which was done in 2021. The main goal was to have a single entry point for students, and to have a more coherent interface.

This work was supported by the CyberExcellence project funded by the Walloon Region, and our contributions were the following:

- Create a web interface to allow students to work on challenges that would be sent to be corrected by PyCAFÉ;
- Create a new version of GLIDE;
- Create an administration interface for teachers to create, edit, delete challenges, and manage users on the website;
- Allow students to log in using the University of Liège's SSO;
- Keep student's marks and results in a database;
- Log as much as possible of the interactions with the platform, in order to be able to analyse the usage of the platform and collect learning analytics;
- Work with Géraldine Brieven, which is in charge of the PyCAFÉ engine, to integrate it into the platform.

3 https://cafe.uliege.be/glide

This web platform required between 300 and 400 hours of work, and the number of lines of code (LoC number) resulting from our contribution amounts to around 20,000 lines (around 10k for the website infrastructure, models, controllers, ..., around 3k for the challenges JavaScript controllers, and around 7.5k for GLIDE), not counting SSO configuration and platform deployment, nor the code already present linked to the frameworks used.

Various technologies have been used for this work: the website is based on Laravel, which is a modern PHP framework, the SSO connection is done using the SAML protocol, using SimpleSAMLPHP, and MySQL and MongoDB have been used for the databases. A multitude of programming languages have also been used, such as PHP, JavaScript, HTML, Blade, CSS, C, shell scripting, Docker scripting, and Python, to name but a few.

It has now been used for two years by hundreds of students that count on it to do homework, train, use it for practical lessons (by using the tool GLIDE), and prepare for exams by revising the challenges. It has also enabled Prof. Benoit Donnet, phD student Géraldine Brieven, and I, to contribute to the scientific community by writing three scientific articles on the learning analytics collected by CAFÉ 2.0, and on the GLI abstraction skills methodology (i. e., [6, 7, 19]).

3.3.2 *Integrating Gamecodes*

In its previous version, as it is shown in Figure 6, CAFÉ 2.0 was composed of different servers running on a SEGI virtual machine, with no much dockerisation, except for the PyCAFÉ challenge corrector, and for the playground, which need a dockerised environment to run the students' code in a secure way. This was not ideal, since integrating new services (e. g., GameCodes) on the VM would require a lot of work, and would not be portable.

Therefore, with the integration of GameCodes, the CAFÉ ecosystem started to evolve into a microservices architecture, with the different services running in docker containers, and orchestrated by Docker Compose. A complete architecture diagram of the new CAFÉ ecosystem after the integration of GameCodes is shown on Figure 7, and a more detailed picture is also shown on Figure 51, in Section A.2.

Called CAFÉ 2.1 in the CAFÉ team.

Now, instead of having everything installed "à la 90s" on the VM, different services, which are all responsible for different tasks, and accessible independently[4] are running in different containers, and are orchestrated by Docker Compose. The different services are the following:

4 Micro services are accessible independently, but still require user authentication, and are not all exposed to the internet.

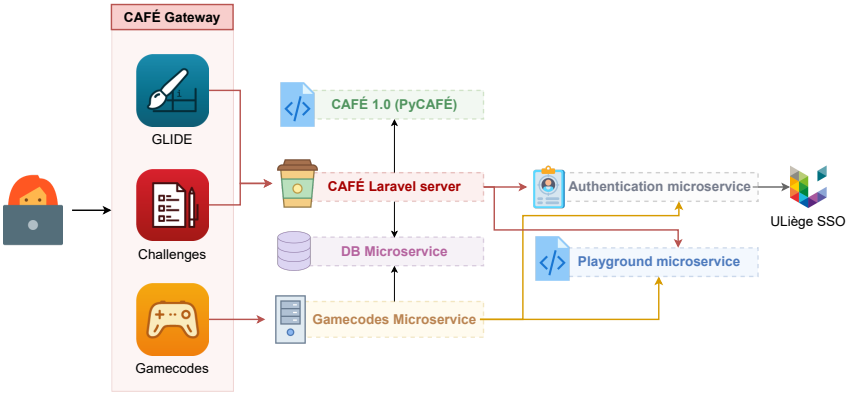


Figure 7: Diagram of the CAFÉ 2.1 ecosystem, before GameCodes

- *Database MS*: a MongoDB database, which is used to store GameCodes, users, logs, challenges, and GLIs.
- *Gamecodes MS*: the GameCodes microservice, which is the back and front end of the GameCodes, and which is accessible from the CAFÉ website.
- *Authentication MS*: a microservice that is used to authenticate users, and to maintain their session. This one uses the University of Liège's SSO.
- *Playground MS*: the playground microservice, which is used to run students' code in a secure way, and which is used by both challenges and GameCodes.

While some of these services are not dockerised yet, the goal is to have everything dockerised and orchestrated by a unique gateway, in order to have a more portable and scalable platform. However, this new structure, using a microservices pattern, allows an easy integration of new services, as it was the case with the GameCodes integration.

As it is shown on Figure 7, GameCodes are using the same database as other parts of CAFÉ, the same playground, and the same authentication microservices. They are accessible either via a "gateway" available on cafe.uliege.be, either via a direct link, which will use the authentication microservice shared with the CAFÉ Laravel webserver if the user is not already authenticated. Also, session is maintained between the different services, using JWT tokens (cfr. Chapter 6, *Components Overview*) and the logs are shared between the different services, in order to have a coherent view of the user's interactions with the platform.

4 TECHNOLOGY CHOICES

4.1 WEBSITE FRONT-END

From the description of the project, an interesting web development pattern to consider would be a *Single Page App* (SPA). An SPA is a type of website that interacts with its users by rewriting the current content of the page, using JavaScript, rather than reloading it in its entirety, and eliminates the waiting time that exists between page reloads. It is, in fact, an application where the general structure of a page does not vary much. Netflix, Facebook and Brilliant.org are good examples of SPAs.

A relevant choice of technology for the website's front-end would then be React. First of all, React has a component-based architecture, which allows developers to develop isolated components that can be easily reused throughout the application. Components can be altered dynamically, without the need to reload the page, depending on user interaction and requests or signals from a server. It is also a highly efficient technology that allows portions of components to be updated only when necessary, saving browser resources so that everything does not have to be rewritten each time.

Secondly, React offers an abstraction layer on top of JavaScript, and also allows Typescript to be implemented, which is, in short, an extension to JavaScript that corrects most of the flaws that are often blamed on this language.

In addition, the learning curve for React is relatively smooth and accessible, making it easy for someone taking over the application to continue working on it, as long as the work is well structured and documented. Also, this is an extremely well-documented, up-to-date library with an active community.

Finally, it is a library that we are used to, and which allows us to deliver quality work more easily than if we had to start again with another language.

Since it lends itself well to the project, and given all the benefits that React brings, it is the technology that has been chosen for the website's front-end.

4.2 WEBSITE BACK-END

A website developed in React is usually accompanied by a Node.js back-end. This is, therefore, the type of server with which the website's front-end will communicate. It is a technology that is widely

used in the development of web applications, and which is particularly well suited to the development of SPAs.

In addition, it is also a technology that is well documented and has a large community, which makes it easy to find solutions to problems that may arise during development. It also enables very simple connectivity with the NoSQL database. In fact, Node.js enables data to be manipulated to and from a NoSQL database via validation schemas, which ensure that the data is correctly manipulated, that there are no missing or superfluous attributes, and which also check the typing of these attributes.

Finally, the choice of Node.js is also motivated by the fact that it is a technology that is well suited to the development of APIs, which will be necessary for the website's front-end to communicate with the back-end.

4.3 DATABASE

The DBMS chosen for this project is MongoDB, a NoSQL database. The reason for this choice is, firstly, that all the data manipulated by the back-end and front-end is in the form of JSON documents. Using MongoDB therefore enables a direct bijection between the format of what is stored, i. e., JSON and BSON documents, and the format of the data manipulated by the website. This avoids the overhead, both in terms of resources consumed and development time, of having to convert, for example, SQL table entries into JSON objects.

Secondly, this is a project where a variable form of documents has been devised to store objects such as, for example, student answers (with a variable answer format depending on the type of question), or the questions objects, or even the various sections that make up a GameCode. Indeed, if we were to use a relational database, and if we take the example of student's answers, this would mean that a large number of tables, depending on the type of question, would be needed to store the answers properly, and it would require some rather tedious join queries to achieve this. This is not impossible, but for faster development and overall simplicity in terms of the format of the data manipulated, MongoDB seems to be a relevant choice.

Finally, if we assume that the platform will be deployed, in the long term, over several courses, with several other hundred students, and possibly in other universities, a great advantage of NoSQL databases is that they are easily scalable. Indeed, it is possible to add servers to increase data storage and processing capacity, without having to modify the database schema, which is not the case with relational databases.

5 DEMONSTRATION

Before going into the details of the implementation, it may be of interest to the reader to have a visual introduction to the GameCodes web platform and the various functionalities it offers. Please note that the platform will be available to all ULiège students and staff members during June 2024, so you can test it on your own if you wish. The purpose of this short section is to present the platform from a user's point of view, but everything described here is accessible by following the link: https://gamecodes.cafe.uliege.be.

5.1 WEBSITE OVERVIEW

When a user first arrives on the platform, they will be greeted by a login page, from which they can log in with their ULiège credentials. Once they are logged in, they will be redirected to the home page, with the list of all available GameCodes, as it is shown in Figure 8.

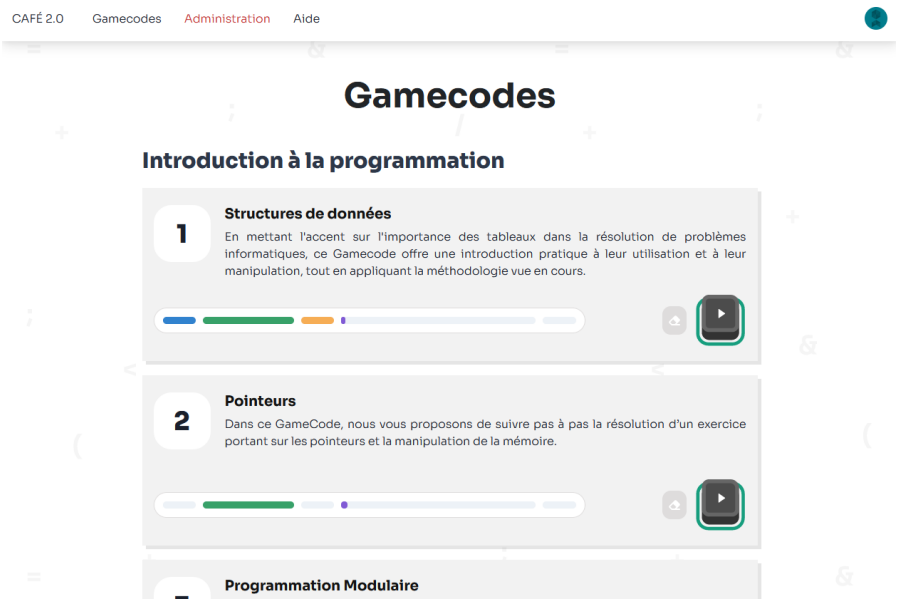


Figure 8: Home page of the platform

Each GameCode displays its title, description, and the current user's progress in the form of a progress bar. The user can click on a GameCode to access it and start working on it, or choose to start it from scratch.

Once they choose a GameCode, they will be redirected to the GameCode's page, where they will be able to see the different sec-

tions that make up the GameCode, as well as the progress they have made in each section.

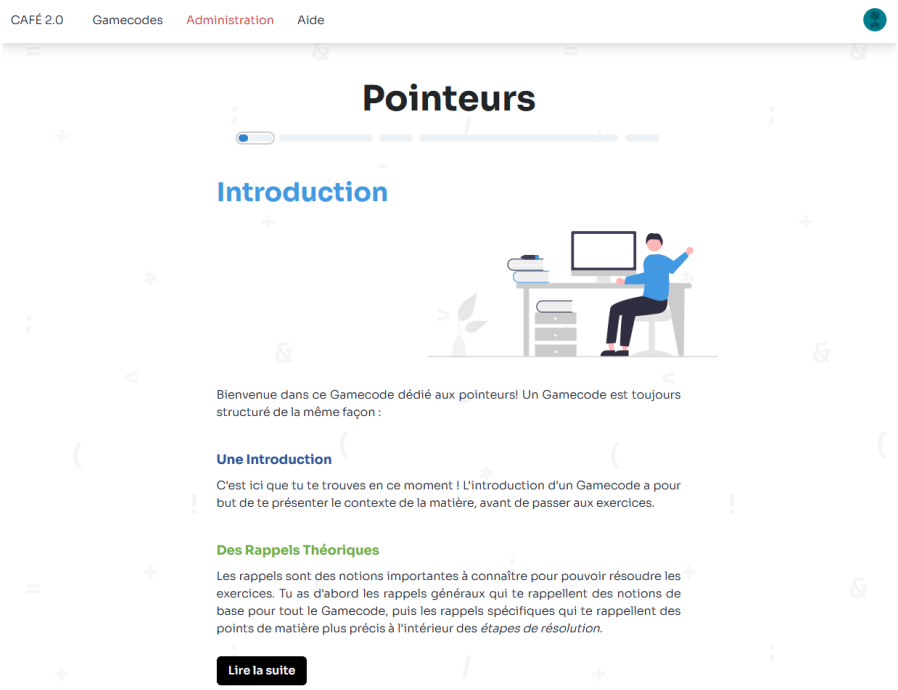


Figure 9: Introduction to a GameCode

For instance, on Figure 9, the user is at the beginning of the GameCode abount pointers in C language, and they are greeted by an introduction on the general structure of a GameCode. There is also a progress bar about all of the different sections of the gamecode on the top of the page. Also, as it was mentioned in Chapter 2 (State-of-the-art), the design of the website is inspired by Brilliant.org, especially when it comes to its very clean and modern design, and the way that a the user can display chunks of text one after the other by clicking on a button.

Figure 10: Global reminders of a GAMECODE

As the user progresses, they can choose to read the global theoretical reminders (see Figure 10) associated with the current GAMECODE, and also choose to add them into their favorite reminders folder. This folder is accessible at all time during the completion of a GAMECODE, thanks to a small book icon on the bottom right of the screen. This feature might be useful for users who would like to use the reminders during their exercises resolution.

Also, reminders are independent of a GAMECODE. They can be added into a GAMECODE or into resolution steps, and reused in other GAMECODES or exercises. This is a way to help teachers so that they do not have to rewrite or copy and paste the same reminders in other GAMECODES.

Once the user read the statement and the reminders they would like to read, they can choose to go to the resolution steps (see Figure 11) of the GAMECODE. Each resolution step is itself composed of:

- A statement;
- Theoretical reminders;
- A series of questions to answer;
- A conclusion, with a summary of what the right answer is and why.

Within a resolution step, the user can also choose how to move within the resolution however they would like. For example, they

Figure 11: Two first resolution steps of a GameCode about binary representation.

might choose to read some reminders first, or to read the conclusion of the GameCode, or to try answering first and then use the reminders if they are stuck at some point (see Figure 12).

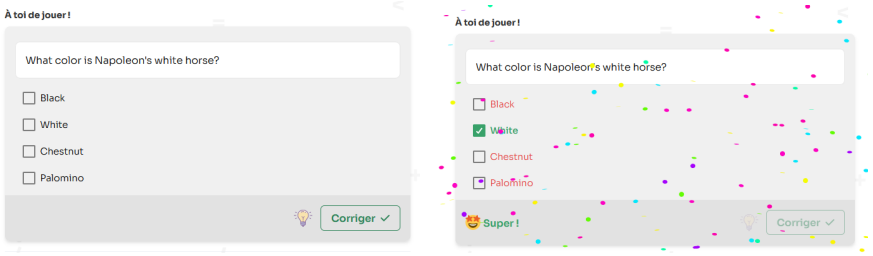
Figure 12: Resolution step of a GameCode

5.2 DYNAMIC COMPONENTS OVERVIEW

Within the resolution steps, in the "questions" section, a series of interactive components are presented to the user. The user can choose to answer them directly, after which an automatic correction is proposed. Automatic correction consists of a simple comparison with an expected output, encoded by the teacher; no advanced correction mechanism has yet been implemented.

In this section is an overview of some of the dynamic components of a GameCode. All of them will not be illustrated, because they are fairly visually similar, but here are some of the most interesting ones.

5.2.1 *MCQ*



(a) Example of MCQ component (b) Solved MCQ component

Figure 13: Screenshots of the MCQ component

The MCQ is a simple-looking component (see Figure 13). It provides one or more correct answers. Also, like all other components, it can include a list of hints that the user can choose to have displayed by clicking on the small light bulb at bottom right, next to the "Correct" button (see Figure 14).

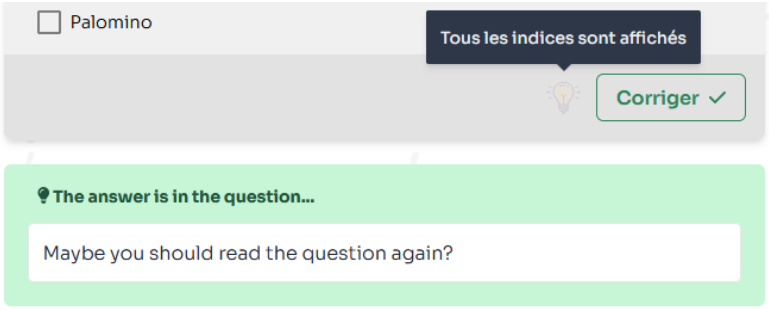


Figure 14: The user chose to display a hint

5.2.2 *Code snippet to complete*

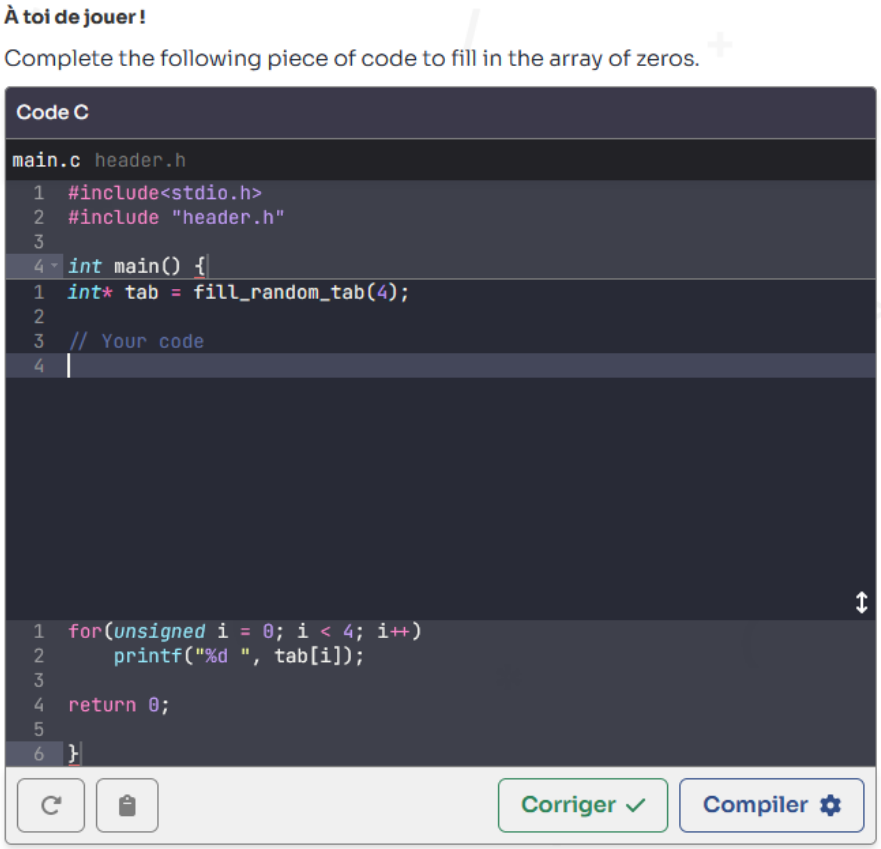


Figure 15: Code snippet to complete

With this component, the user can edit some sections of the C files that are proposed to them. They can also choose to compile the code to see the output of their solution, and they can ask for an automated correction that will compare the output of their program with the one encoded by the teacher.

The teacher can also choose to include an invisible file, that the student will not see, but that will be compiled along with the student's code. This is useful for the teacher to encode the expected output of the student's program.

Finally, the teacher can also choose to lock some parts of a code file. For instance, on Figure 15, the lighter parts of the code are locked, and the user cannot edit them. This is useful for the teacher to force the student to use a specific function, or to force them to use a specific variable name.

5.2.3 *Binary calculations*

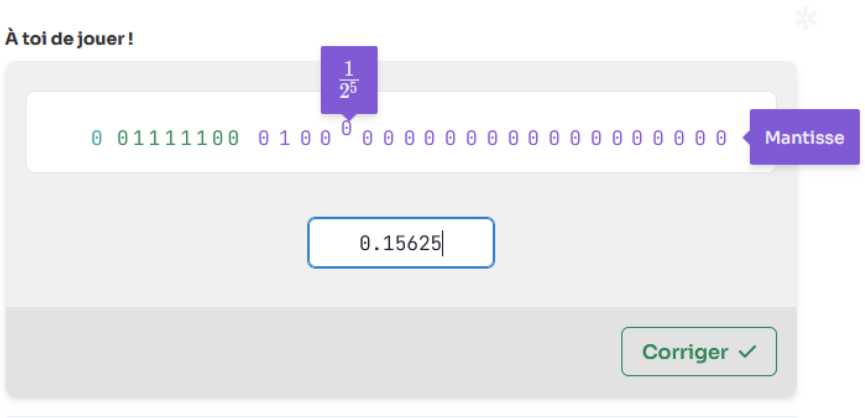


Figure 16: IEEE 754 binary conversion

This component is used to make the user practice binary encoding. The user can choose to convert a number from decimal to binary, or the other way around, and they can also practice with the IEEE 754 floating point representation (see Figure 16). If the exercise is about floating point representation, the user also gets visual help to help them understand how floating point numbers are stocked in a computer memory.

Another type of exercise around binary numbers is the binary addition (see Figure 17). The user can practice with the addition of two binary numbers, and they can also practice multiplication, and subtraction.

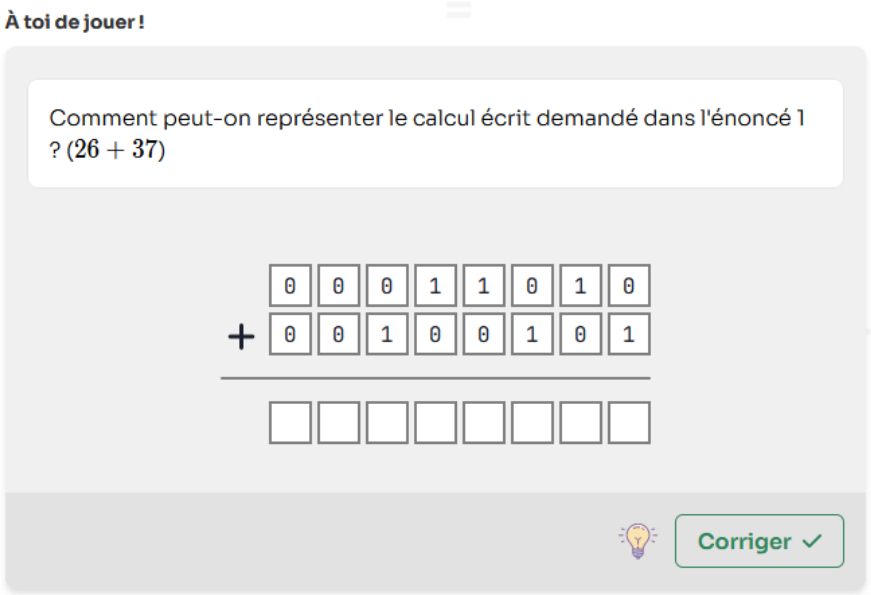


Figure 17: Binary number addition

5.2.4 *Breakdown into sub-problems*

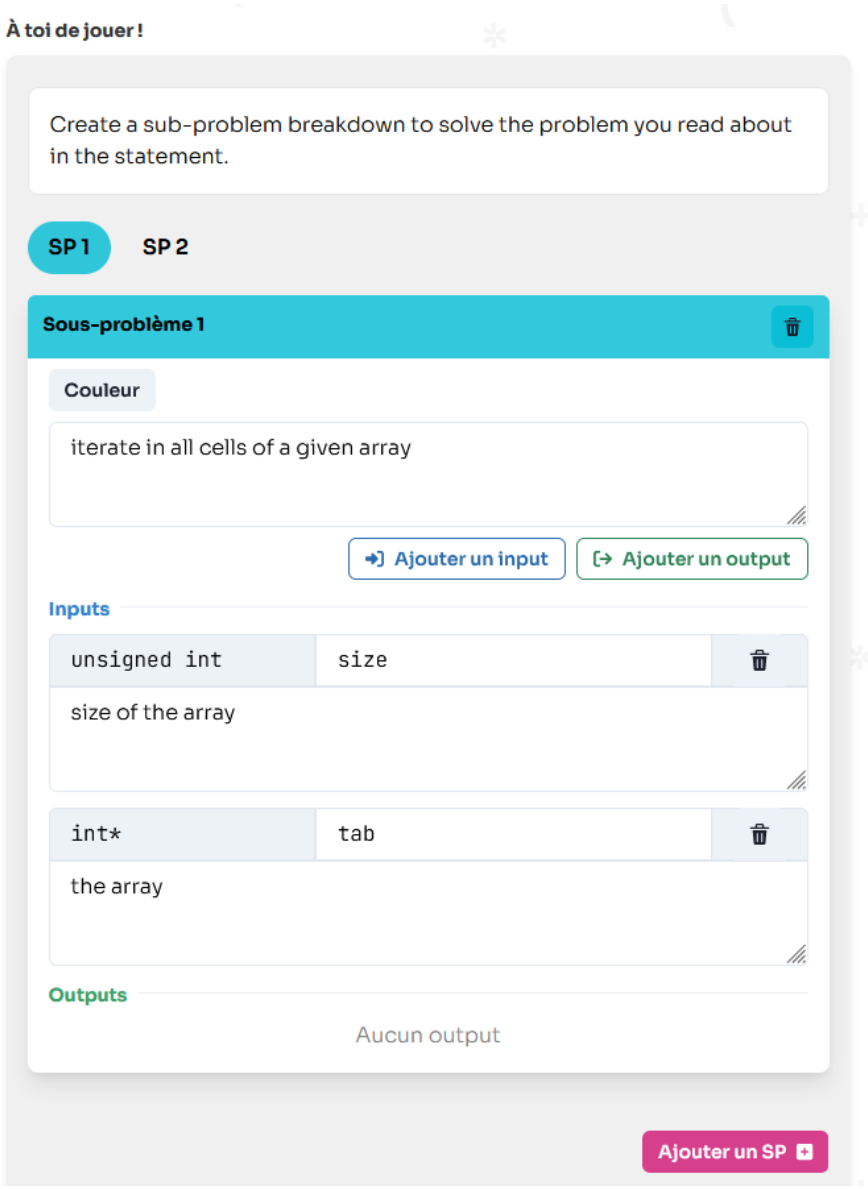


Figure 18: Breakdown into sub-problems

In the context of their first year course, *Introduction to programming*, students are often asked to solve a problem by breaking it down into smaller sub-problems. This component (see Figure 18) is used to make the user practice this skill. The user is given a problem to solve, and they have to break it down into smaller sub-problems, and then solve them one by one. This is a way to make the user think about the problem before starting to code, and to make them understand that a problem can be solved by breaking it down into smaller problems.

Afterwards, this breakdown can be injected into another component to help them to construct their solution to a programming problem by using the sub-problems they have identified (see Figure 19).

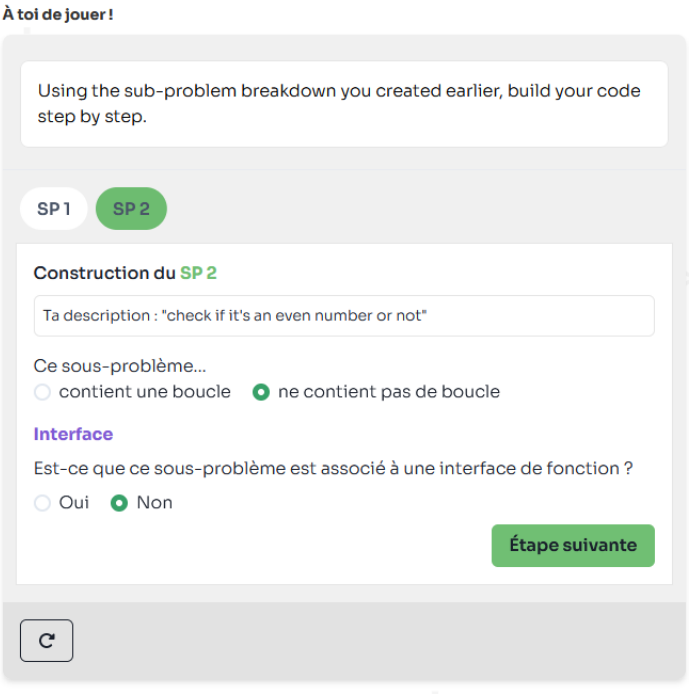


Figure 19: Construction of a solution by using sub-problems

5.3 ENCODING INTERFACE

The encoding interface is the part of the platform that is used by the teachers to encode the different GameCodes. It is accessible through the "Administration" button in the navigation bar, and is only visible to users that have an `admin` rank.

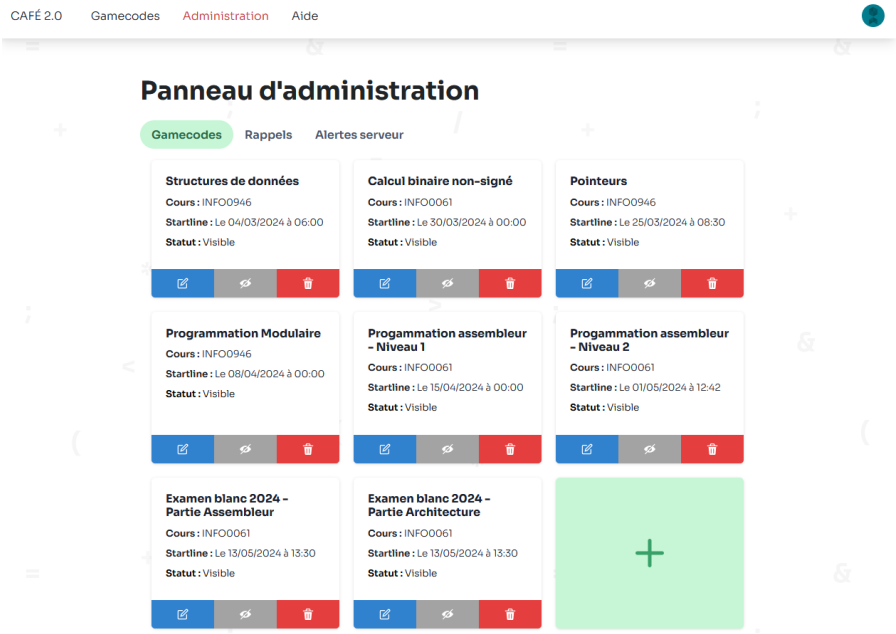


Figure 20: Administration interface

On this interface (see Figure 20), the teacher can create a new GameCode, and encode its different sections. They can also choose to edit an existing GameCode, hide it from students, or delete it. There is also a link to the list of all encoded reminders, so that they can be edited from there, and so that the teacher does not have to find the GameCode in which they are encoded.

Figure 21: Encoding a GameCode

Once the teacher starts encoding a specific GameCode, they can enter some basic information about it, such as its title, description, and to which course it is related to (see Figure 21). Then, they can start encoding each part of the GameCode individually (introduction, reminders, ...).

Again, an exhaustive list of all the components that can be encoded is not presented here, because they are fairly similar one to another. However, here is an example with the MCQ component. First, the teacher has to select the GameCode module in which they would like to encode the exercise, and then, choose from the dynamic components list (see Figure 22).

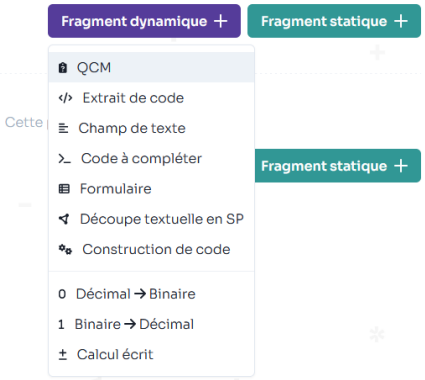


Figure 22: Selecting a dynamic component

Then, using a text editor, they can encode the statement of the exercise, the different answers, and select the correct one (see Figure 23).

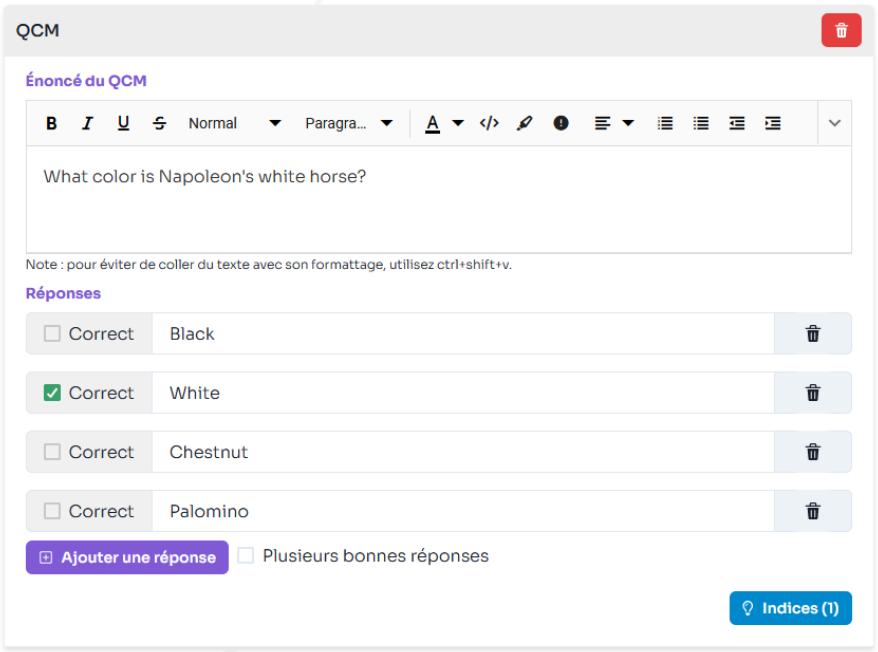


Figure 23: Encoding a MCQ

It is worth mentioning that we added a LaTeX feature into the text editor, so that teachers can specify some inline LaTeX code in the different components, between dollar signs $. It is then compiled and built on the fly when the component is loaded on a page.

Finally, the last component to be mentioned is the theoretical reminder editor. As already mentioned, these reminders can be written once, and then included in any of the other GameCodes, thanks to an import function (see Figure 24).

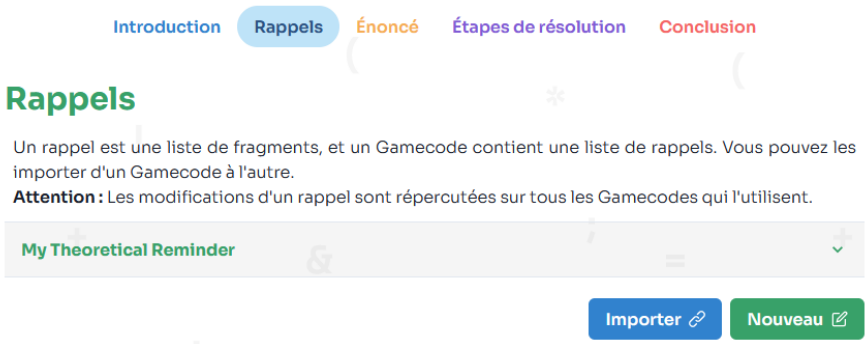


Figure 24: Encoding a theoretical reminder

6

COMPONENTS OVERVIEW

This chapter provides an overview of the platform components, with a detailed explanation of their implementation, in the order in which a user should call upon them when using to the platform. Two different points of view will be developed: firstly, that of a student, and secondly, that of a teacher, with regard to the encoding of a GameCode. Also, some of the parts of this overview are about components that are not directly implemented in the platform, but on CAFÉ's side.

The CAFÉ platform was already online prior to Gamecodes development, offering a homework submission system with personalized feedback to students.

Once all the components have been detailed, an overview of the platform's architecture will be given, explaining how the different components fit together, and how they are dockerized and deployed on the SEGI virtual machine.

6.1 AUTHENTICATION AND IDENTIFICATION

To access the platform, you must first be authenticated via the ULiège SSO. This authentication is a two-step process: the first step is to authenticate via the user's ULiège ID and password in order to open a session, and the second step is to redirect the user to the website, while retaining the user's data provided from the first step, in order to integrate it into requests made to the server in the form of a token attached to the browser's cookies.

6.1.1 *ULiège SSO*

Background on the concept of SSO

SSO, or Single Sign On, is an authentication method that allows a user to authenticate on a single platform, and use proof of identity on multiple websites to which this SSO can redirect the user.

For its SSO, the University of Liège uses SAML (Security Assertion Markup Language), an XML-based protocol specially designed to exchange information relating to a user's security.

Without going into too much detail about how SAML works, there are two important concepts to bear in mind: Service Provider (SP) and Identity Provider (IdP). When a user wants to authenticate on a third-party website, this third-party website will be called Service Provider (e. g., in our case, cafe.uliege.be), and the website on which the user enters their credentials is called IdP (e. g., my.uliege.be).

To be recognized as a valid SP, the third-party website must be added in a list of authorized SPs on the IdP's side. The SP must provide them with a signed certificate, which will contain the public key of the SP, as well as other information about the SP, its maintainer, and some configuration information about the domain on which the SP would like to use the SSO.

SimpleSAMLphp

In order to communicate with the University's SSO, gamecodes.uliege.be had to become a Service Provider. To achieve this, the PHP library *SimpleSAMLphp*[1] was used.

On the virtual machine on which GameCodes are hosted, a LAMP stack (i. e., Linux Apache MySQL and PHP) was configured to host this library. In concrete terms, this is a PHP web server that acts as an SP intermediary between the CAFÉ server and ULiège's SSO. Apache, which is an HTTP server, has been configured to redirect requests made via the `/simplesaml` route to this SimpleSAML server, which then handles user authentication.

Here is an explanation of the steps involved in user identification, as illustrated on Figure 25:

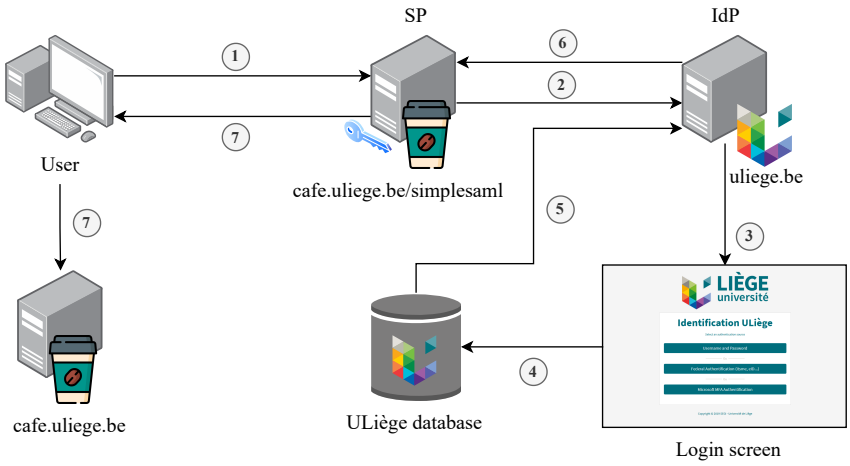


Figure 25: Illustration of SAML application with CAFÉ and ULiège.

1. User requests access to `cafe.uliege.be/simplesaml`.
2. Unauthenticated requests are redirected to the SAML IdP (uliege.be), with the SAML request.
3. The IdP displays a login page.
4. User's credentials are sent back in the IdP's user database for checking.

1 https://simplesamlphp.org/

5. The verification status is sent back to the IdP.
6. The IdP sends back a SAML response.
7. User is redirected with an authentication token, attached in its cookies, and is considered as logged in CAFÉ as long as it keeps its cookie.

Once the user is authenticated, here is an example of SAML data that the SSO will send back to the SP as a response, on Listing 1:

Listing 1: Example of a SAML response data

```
urn:oid:0.9.2342.19200300.100.1.1 : "s180162",
urn:oid:2.5.4.4 : "Malcev",
urn:oid:2.5.4.42 : "Lev",
urn:oid:1.3.6.1.4.1.5923.1.1.1.1 : "student",
urn:oid:0.9.2342.19200300.100.1.3 : "L.Malcev@student.uliege.be",
urn:oid:1.3.6.1.4.1.10383.2.1.56 : "Master sc. inform. fin. spec.
    comput. syst. secur.",
...
```

This data contains information about the user, such as their first name, last name, email, user ID, and the user's role in the university. This data is then used to create a user account in the CAFÉ database, if it does not already exist.

After the account is created, a JWT (JSON Web Token) is created and sent back to the user's browser, which will be used to authenticate the user in the future.

6.1.2 *JWT Authentication*

JWT, or JSON Web Token, is a standard for creating tokens that can be used to authenticate users. This token is a JSON object that is signed with a secret key, and can contain any information that the server wants to store in it.

It is typically encoded in base64, and is composed of three parts: the header, the payload, and the signature. The header contains information about the algorithm used to sign the token, the payload contains the data that the server wants to store in the token, and the signature is a hash of the header and the payload, signed with the server's secret key. Here is an example of a JWT token, using the SHA256 signing algorithm, with its header, payload, and signature parts:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJmaXJzdG5hbWUiOiLDiW1pbGllIiwibGFzdG5hbWUiOiJDaGFybGllciJ9
.QsQqXRHDWaRtbRo4vxMKtgw372tSVVPL6-Is-3lW8iA
```

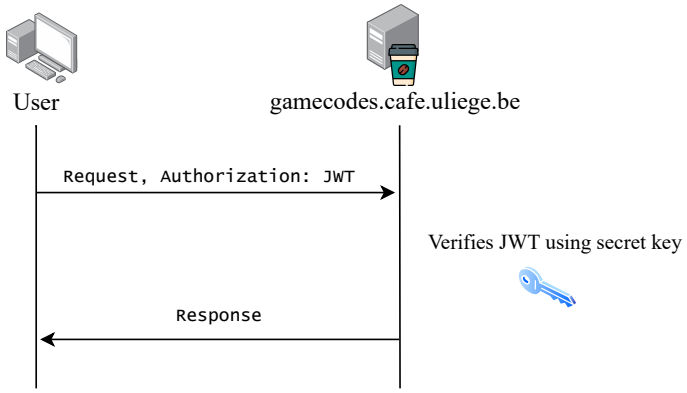


Figure 26: Illustration of Gamecodes JWT authentication.

In the case of CAFÉ, the JWT contains the user's ID (i. e., "sXXXXXX" or "uXXXXXX"), their rank, first name, last name, and the expiration date of the token. This token is sent back to the user's browser, and is attached to the browser's cookies.

When the user makes a request to the server (cfr. Figure 26), the JWT will be attached to the HTTP request in the `Authorization` header. The server will then check if the token is valid, and if it is, it will allow the user to access the requested resource. If the token is not valid, the server will return an error message, and the user will have to log in again. Reasons for the token to be invalid include the token being expired, the token being tampered with, or the token being signed with another key than the one stored on the server.

6.2 GAMECODE STRUCTURE

Once a user is logged in, they can see the list of available GameCodes to which they have access. As a reminder, a GameCode is always composed of the same five parts, as it is explained in Section 3.1: an introduction, global theoretical reminders, a statement, resolution steps, and a conclusion.

In this section, we will explain how they are structured in the database, and how they are displayed to the user.

6.2.1 *Fragments system*

Each part of a GameCode is made up of a list of HTML components, with which the student may or may not interact. These components can be, for example, chunks of text, images, exercises to solve, or even code to compile. To ensure a high degree of modularity and the ability to reuse these components in every part of a GameCode, a very specific data structure has been defined, named "*Fragment*".

A fragment can be either static, or dynamic:

- A *static* fragment is defined as a stateless HTML component, which allows a low degree of interactivity, such as text, images, or a static code chunks.
- A *dynamic* fragment, on the other hand, is a stateful component, which allows a high degree of interactivity, such as exercises, code editors, MCQs, written binary calculations, etc., and which can be solved by the student, played with, or modified, and it will be saved in the database.

An updated picture of the structure of a GAMECODE is given on Figure 27.

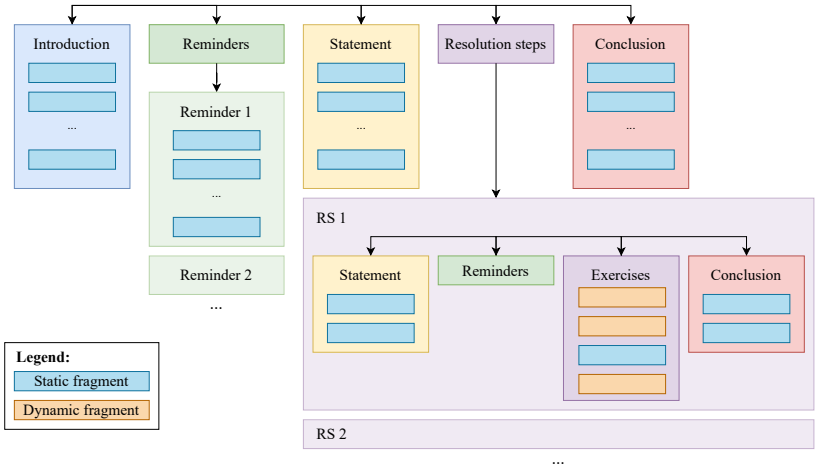


Figure 27: Illustration of a GAMECODE and its fragments.

For instance, Section 5.2 (Dynamic components overview) gives some visual examples of what a dynamic fragment might look like (e. g., MCQ, code snippet, ...). Static fragments are fairly similar, but without the interactive part.

Database modeling

One important thing worth mentioning about fragments is that, since there are two types of fragments, they are stored in two different collections in the database. The `static_fragments` collection contains all the static fragments, and the `dynamic_fragments` collection contains all the dynamic fragment. This becomes an issue when some part of a GAMECODE needs to use both types of fragments, for example, if a teacher would like to encode a series of questions and add some textual explanation in between dynamic fragments.

In order to solve this issue, a new collection was created, called `fragments`, which stores *fragment pointers*. These fragment pointers

are associated with either a static or dynamic fragment ID, and the type of the fragment (i. e., static or dynamic). In this way, a part of a GameCode can be composed of a list of fragment pointers, which can be either static or dynamic, and the server will know from which collection to fetch them, and send them to the user (see Figure 28). A more complete version of this model can be found in Section A.1.

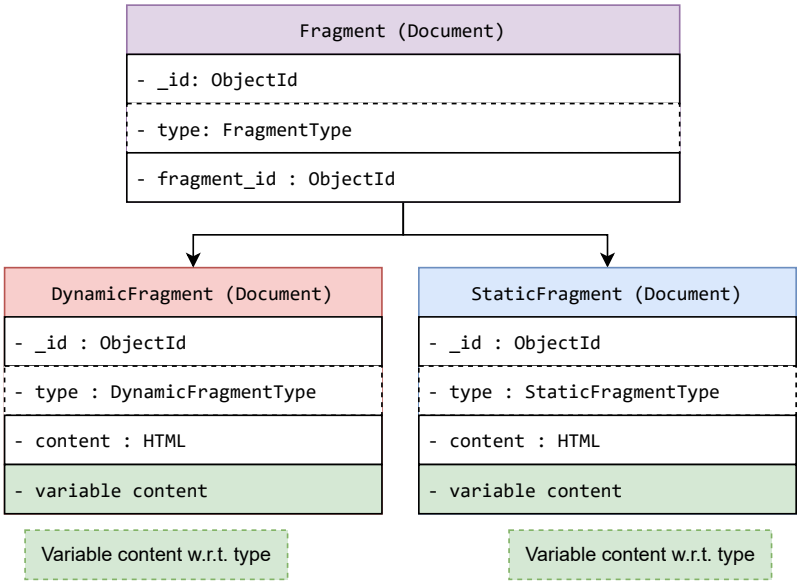


Figure 28: Illustration of the database model for fragments.

However, this technique adds a layer of complexity to the server, as it must now fetch the fragments from two different collections, and merge them into a single list of fragments to send back to the user. This is a small example of the trade-off between modularity and complexity, which is found to be acceptable int this context, since we are using hashes of documents (the `fragment_id` attribute of a `Fragment` document), it does not increase the fetching time of the fragments too much.

Fragment UI

All fragments have certain attributes in common: a statement, a similar design, hints, and a correct or compile button. But they are also very different: an MCQ-type fragment will not have the same question or answer structure as a fragment which purpose is to compile code. To represent them as well as possible, and to allow a certain modularity, I have chosen to apply the *factory* programming pattern.

The factory method (see Figure 29[2]) is a creational pattern that provides an interface for creating objects in a superclass, but allows

2 Image taken from the website *refactoring.guru (Design Patterns / Creational Patterns)*.

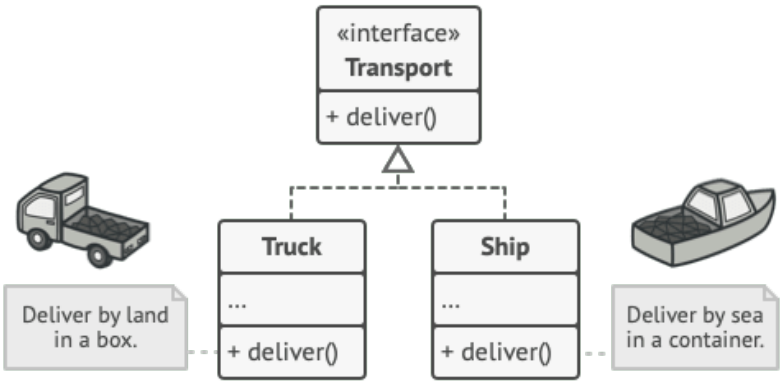


Figure 29: Illustration of the factory pattern.

subclasses to alter the type of objects that will be instanciated. In this case, the superclass is the `Fragment` class, and the subclasses are the different types of fragments that can be created.

In this context, this pattern takes the shape of a React JSX component, which will take a `fragment` object as a parameter, and will return the correct fragment component, based on the type of the fragment. This method also allows to add new types of fragments easily, by simply creating a new component, and adding it to the factory method.

6.3 AUTOMATED CORRECTION

Most of the dynamic fragments have a fairly simple automated correction, as the realization of an intelligent correction engine was not the focus of this master's thesis. This correction is based on a simple comparison with an answer previously encoded by the teacher. For example, an MCQ will be corrected by comparing the answers ticked by the student with an array of right or wrong answers. The same applies to the correction of a calculation written in binary: arrays of numbers encoded by the teacher are compared with those proposed by the student.

However, when it comes to correcting code snippets submitted by a student, a character-by-character correction becomes impossible, as the code would have to be compiled, executed and checked to see whether the answer obtained is correct or not. It is this correction mechanism that will be explained in this section.

6.3.1 *Online compilation*

The first step in correcting an answer to a code snippet question is to enable a student to compile their solution online. The main idea would therefore be to compile submitted files on the server

side, while the student would await the answer on their side, in the browser. And in fact, such a system was already partially present on the CAFÉ platform, which allows students to compile C code files in their browser as part of their homework, to practice before submitting an answer.

On the CAFÉ platform, this feature is called *playground*, and has been adapted so that it can also compile C files (or assembly files) from other websites (e. g., from the GameCodes platform), provided the request is authenticated. Here are the steps involved in the online compilation process (see Figure 30):

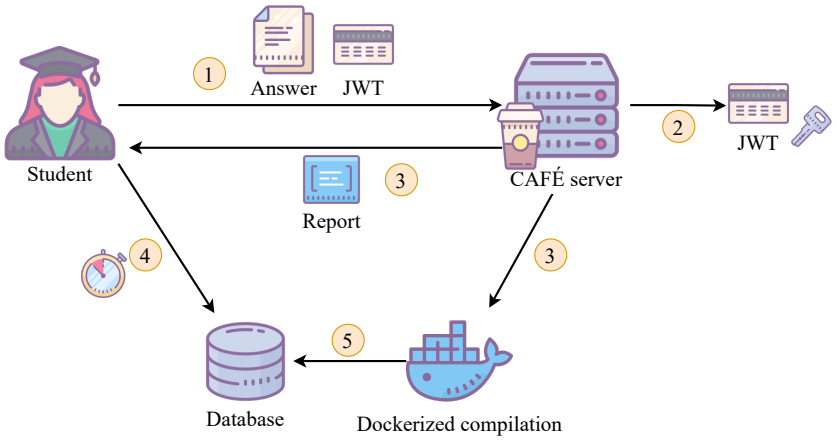


Figure 30: Illustration of the online compilation process.

1. *Submission*: The student submits their answer as an array of files, each defined by a filename and a content. This submission is sent to the CAFÉ backend, and not directly to the GameCodes back-end, along with a JWT token for their identity verification, and with the ID of the question they are answering.

2. *Verification*: CAFÉ's back-end verifies the JWT attached in the Authorization header of the request, with its private key, which is shared with the GameCodes platform, allowing both services to communicate with each other.

3. *Compilation*: once the student's identity is verified, the back-end of CAFÉ launches a dockerized compilation process. It also creates a new database entry, called a `cafe_report`, linked to the submitted answer ID, and sends back the ID of this (yet empty) `cafe_report` entry to the student's browser.

4. *Polling*: While the compilation runs, the student's browser polls the CAFÉ back-end every 500ms to check the compilation status.

5. *Completion*: Once the compilation is finished, and the program has been executed, compilation and the execution results are stored in the related `cafe_report` entry in the database, and the student's browser will fetch it and display the result to the student.

The CAFÉ back-end was not designed using React, but Laravel, a PHP framework. Routes have been configured for this purpose: `/gamecode/submit` and `/gamecode/compiler`. When a request arrives on the submit route, after sending back the empty `cafe_report` document to the student, the PHP server launches a new background process which will compile the submitted files.

This compilation process creates a new repertory using the student's ID and a timestamp, and copies the submitted files into this repertory. It then launches a Python script, called `playground_compiler.py`, which will, at its turn, run a secure compilation and execution of the student's program. This compilation process is composed of three steps:

1. Checking the files content: first, the script checks if the submitted files are ready for compilation. It will check if some functions considered as "dangerous" for the CAFÉ server are present in one of the files (e. g., `exec`, `popen`, `system`, ...), as well as if some functions that require user input are present (e. g., `scanf`, `gets`, ...). If so, the compilation is marked as failed, the next steps will not be started, and the error will be sent back to the student.

2. Compilation: then, a new process is launched to compile the files using `gcc`. A timeout is set to 10 seconds, and if the compilation takes longer than this, the process is killed, and the compilation is marked as failed. Whether the compilation succeeded or not, its output is stored in the `cafe_report` document.

3. Execution: if the compilation succeeded, the script will launch a new process to execute the compiled program. It will be executed in a docker container with a memory limit to 10 MB, a CPU usage limit, and a timeout of 10 seconds. The output of the execution, which is either the program's output, or an error message, is stored in the `cafe_report` document.

Once all of these steps are finished, and the `cafe_report` file is completed, the repertory that was created for the student is destroyed, and the results of the compilation and the execution of the program are available to the student.

6.3.2 *Correction*

Automating the correction of a computer program is a difficult task. It is even an undecidable one, if we refer to the *halting problem*. In this context, the problem addressed is not the verification of the correctness of a program for every possible input (otherwise, it would be about implementing a method to solve the halting problem, which would take quite a long time to implement[3]). Since the design of a correction engine is not the focus of this thesis, this problem was approached differently.

In this context, correcting a program is a simple matter of comparing its output with the output expected by the teacher. Obviously, this type of correction does not guarantee anything about the content of the program itself, since a student would only need to make a call to `printf` to display the content expected by the teacher. However, thanks to the hidden file system available to the teacher when encoding exercises, this becomes a fairly useful way of correcting a proposed solution.

Indeed, when encoding a question, the teacher can add *hidden files* to the student's workspace. These files are not visible to the student, but are accessible by the program. This allows the teacher to encode a solution to the question, and to compare the output of the student's program with the expected output.

Listing 2: Student file

```
void sort_tab(int* tab, const unsigned size) {
    // Your solution here
}
```

Listing 3: Hidden teacher file

```
#include <stdio.h>
#include "sort_tab.h"

int main() {
    int tab[] = {17, 0, 8, 42, 5, 9, -1};
    sort_tab(tab, 7);
    for(int i = 0; i < 7; i++)
        printf("%d, ", tab[i]);
}
```

For instance, if we take a simple array sorting question, the teacher could add a hidden file that displays the content of an array, sorted in ascending order. The student's program would then have to display the same content, in order to be considered correct. This technique is not perfect, but it is a good way to automate the correction of a

3 i. e., an infinite amount of time.

program, and to give the student immediate feedback on their solution. In this example, the student would have to write their answer in the student file (Listing 2), and the teacher would create a hidden file (Listing 3), and the corresponding expected output would then be: `-1, 0, 5, 8, 9, 17, 42 .`

6.4 DEPLOYMENT

The GameCodes platform is hosted on a virtual machine, provided by the SEGI. It is deployed on this virtual machine using docker-compose, which allows to run multi-container Docker applications.

It consists in three parts: the client, which is the React app, the Node.js server, and the MongoDB database. Once all applications have been launched via docker-compose, they are all listening to a defined port.

In order to make the application accessible to the public, the virtual machine is configured to redirect the domain name `gamecodes.cafe.uliege.be` to the IP address of the virtual machine. However, this is also the case for the domain name `cafe.uliege.be`, so a proxy server is used to redirect the requests to the correct application, based on the domain name. All requests made to `gamecodes.cafe.uliege.be` are forwarded to the GameCodes client, through a specific port number, and all requests made to `gamecodes.cafe.uliege.be/api` are forwarded to the back-end server.

On top of that, all requests are secured with HTTPS, using a certificate provided by *Let's Encrypt*, which is automatically renewed every year.

Finally, the virtual machine is also configured to redirect all requests made to the `/simplesaml` route to the SimpleSAMLphp server, which is used for the SSO authentication.

One final point that may be worth mentioning is that special measures were taken to calibrate the resources allocated to the virtual machine to accommodate a few hundred simultaneous connections to the machine. Indeed, the machine already had sufficient resources for CAFÉ, but it is very difficult to estimate the amount of memory needed for a worst-case scenario where 400 students connect at the same time (which did not happen). For this reason, a request was made to SEGI to monitor the machine's RAM and CPU consumption during a week when a large number of connections were expected, and it was concluded that there was no need to modify these parameters.

In the analysis sent by the SEGI, which was CPU and RAM consumption of the virtual machine during a week, and during a day of moderate use, CPU consumption never exceeded 20%, and RAM

consumption was negligible, which meant that the machine seemed to be well-dimensioned for the platform's use.

6.5 SUMMARY OF THE COMPONENTS OVERVIEW

It is not an easy task to find the right level of abstraction to provide enough information about the key elements of this project, without drowning it in trivial, low-level information. In this chapter, therefore, I have presented what I believe are the most important interesting, from a technical point of view, of the GameCodes platform.

In addition to all the implementation details described in this chapter, it may also be useful to note that the source code for GameCodes, as well as the whole CAFÉ environment (with the exception of the authentication microservice, for security reasons), is available on the CAFÉ Gitlab repository, which is accessible to the public. To give a further idea of the quantity of work that was GameCodes implementation, the total number of lines of code (LoC) is around 19,000 (around 4,000 for the server and 15,000 for the client), and the total LoC number for the learning analytics Python scripts, that were used for the content of Chapter 8, is around 2,500.

The implementation of this project lasted for more or less one year, from February 2023 to February 2024. Its first deployment was initially scheduled for September 2023, in order to collect data on first-year students from the start of the school year, but was postponed to March 2024, due to lack of time. Overall, the rollout went off without a hitch, and no bugs were reported. Since this is a prototype, there are a multitude of improvements planned between the end of 2024 and 2028, some of which will be developed in Chapter 9 (Future Improvements).

Part III

DATA ANALYSIS

This part provides an overview of the data collected during the deployment of the GameCodes platform, in the form of *learning analytics* (LA), as well as an analysis of this data. The aim of this part is to present the results of the data collection, to analyze what can potentially be analyzed, given the fact that very few students took part in the project, and to provide an overview of the strong potential for future work in this area.

7 DATA COLLECTION METHOD

As previously mentioned, a series of statistics on GameCodes usage was collected over several weeks. These data were collected by certain web mechanisms, saved in a database, pre-processed, while preserving the students' anonymity. The aim of this chapter is to present the data collection method used to gather the statistics on GameCodes usage.

7.1 DATA COLLECTION

Two types of data were collected via the platform: GameCode consultation time tracking data, and the quantity of student interaction with the website. In concrete terms, consultation time data is an approximation of the time that each fragment (dynamic or static) was visible on the student's screen. Interaction data consists of all student inputs to the platform, i. e., their history of answers to questions and their (lack of) consultations of reminders or hints.

7.1.1 *Time Tracking*

Each fragment visible on the screen has a heartbeat system. Every 15 seconds, each fragment sends a message to the server to indicate that it is displayed on the screen, and the server updates a document in the database, in the `fragment_sessions` collection, with its last heartbeat time. A fragment being "displayed on the screen" means that the fragment is :

- in the viewport of the browser, i. e., not scrolled out of view,
- and in the currently opened tab of the browser.

A fragment session document (see Figure 31 for an example) is created when a fragment is displayed for the first time, and is updated with each heartbeat. The document contains the following fields:

- *matricule* : the student's matricule,
- *fragment ID and type* : the ID and type of the fragment,
- *gamecode instance ID* : the ID of the current gamecode instance,
- *tab ID* : a random ID generated by the browser for the current tab,

- *first heartbeat* : the time at which the fragment was first displayed,
- *last heartbeat* : the time at which the last heartbeat was received.


```
_id: ObjectId('65edaf54bd9bc9f716fe54c3')
matricule: "s180162"
fragment_id: "65eb6877e1a92ead3d8f53fa"
fragment_type: "2"
gcinstance_id: "65eb62f8af56b76c23b279f7"
tab_id: "ltlivfyf2g5o36zwsr6"
first_heartbeat: 2024-03-10T13:02:12.269+00:00
last_heartbeat: 2024-03-10T13:03:52.489+00:00
```


Figure 31: Example of a fragment session document.

Thanks to this collection, it is possible to calculate the time spent on each fragment by each student. The time spent on a fragment is calculated by summing the time between each heartbeat.

7.1.2 *Interaction Data*

The second type of collected data is about interactions with the different components of a GameCode. Here is an exhaustive list of every type of logged interaction:

- *Answers history to each question*: each time an answer is changed or submitted, it is added into a history of answers for a given dynamic fragment;
- *Hints consultation*: when a hint to a question is consulted, the hint, the related question, and the consultation time are logged;
- *Compilation reports*: each code compilation with its output and the output of the program are logged;
- *Module change*: when a student changes module inside a GameCode (e. g., when they go from the introduction to the reminders), the action is logged.
- *The progress* within a given GameCode, and the progress within a given theoretical reminder: a list of GameCode instances and theoretical reminders instances for each student is logged, which allows to observe the progress of a student within a GameCode or a reminder.

7.2 PREPROCESSING

This section explains the various preprocessing measures that have been put in place concerning the GAMECODES themselves, participant discrimination, and time tracking.

7.2.1 *Gamecodes separation*

Six GAMECODES have been redacted for two courses: three for INFO0946 (computer science students) and three for INFO0061 (engineering and computer science students). The former has a very low participation rate, and the latter has a higher one. To avoid any bias in the analysis, the data from the two courses have been analyzed independently. Moreover, the analysis in the following chapter – Chapter 8 – will be done independently for each GAMECODE, since they are very different from each other, and the participation rate is very different from one GAMECODE to another.

Titles are kept in French for consistency with the content of the website.

The list of the analyzed GAMECODES, grouped by course, is the following:

- Group A – INFO0946:
  1. GCA0: *Structures de données*
     - *Statement*: Sort an array according to an imposed pivot value x: any value below x must be on the left of the array, and any value above must be on the right, using a single loop (i. e., the program must be O(N)).
     - *Global reminders*: one about arrays in C language.
     - *Resolution steps*: 7 resolution steps, following the course's work methodology.
  2. GCA1: *Pointeurs*
     - *Statement*: Given a memory state, evaluate the value of 10 C language expressions relating to pointers.
     - *Global reminders*: one about C expressions, and one about pointers.
     - *Resolution steps*: ten resolution steps, one for each expression.
  3. GCA2: *Programmation Modulaire*
     - *Statement*: Create a program divided into functions (applying the concept of modularity) that, given a number, evaluates whether the sum of its digits raised to the power of the number of digits of which the number is composed is equal to the number itself (e. g., is 153 equal to $1^3 + 5^3 + 3^3$?).
     - *Global reminders*: one about programming modularity.

  - *Resolution steps*: six resolution steps, following the course's work methodology.

- Group B – INFO0061:
  1. GCB0: *Calcul binaire non-signé*
     - *Statement*: Perform the following three calculations in unsigned binary format:$26 + 37$, $85 + 59$, 12×41
     - *Global reminders*: one about decimal encoding vs. binary encoding, and one about binary addition and multiplication.
     - *Resolution steps*: five resolution steps about the steps to take before the calculation, during the calculation, and after the calculation (checking step).
  2. GCB1: *Progammation assembleur - Niveau 1*
     - *Statement*: Create an assembly function that fills an array given as an argument, along with its size, with a given constant values.
     - *Global reminders*: four reminders – assembler calling convention, arrays manipulation, loops and conditions, and registers.
     - *Resolution steps*: two resolution steps about understanding the statement, and about the implementation.
  3. GCB2: *Progammation assembleur - Niveau 2*
     - *Statement*: Create an assembly function that, given an array and a constant value C, returns the number of times there is a number n in the array such that $2 \times n < C$.
     - *Global reminders*: four reminders – assembler calling convention, arrays manipulation, loops and conditions, and registers.
     - *Resolution steps*: four resolution steps, two about understanding the statement, and two about the implementation.

7.2.2 *Participants discrimination*

GameCodes have been deployed to a total of 355 potentially interested students. However, not all of them tried to participate to GameCodes, and the ones who participated potentially did not read the GameCodes in their entirety, or even simply did not try to answer any question at all and were just curious about the platform.

To discriminate students who were truly committed to GameCodes and those who were simply curious about them (i. e., to keep those who provided the most interaction with the platform), a new set of

students has been created: the ones called *active participants*. For a given GameCode, an active participant can be defined as a participant who answered to at least 50% of the questions. This threshold has been decided after comparing the number of participants per percentage of answered questions, and it has been chosen to keep a reasonable number of participants while excluding the ones who did not interact with the platform at all. The number of active participants for each GameCode and per defined threshold is presented in Figure 32. This figure shows, for each GameCode, the proportion of students over the total number of students who opened the GameCode, in function of the percentage of questions they answered. It never goes up to 100% students, because no GameCode has more than around 35% of participation, over all the students who opened it.

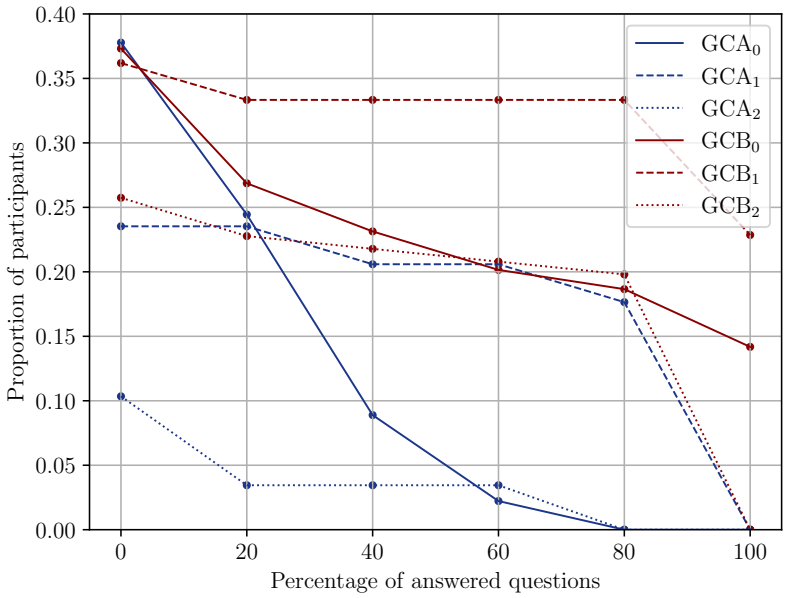


Figure 32: Cumulative distribution of the proportion of active participants per GameCode and per threshold.

There is a natural trade-off between the number of active participants and the proportion of answered questions: the more questions a student answers, the more collected data is available for analysis for each student. However, the less questions are answered, the higher the participation rate is, but the collected data quality is worse, since they did not participate "seriously".

Changing the threshold from approximately 20% to 80% does not drastically change the proportion of participants for Group B (in red, i. e., the group with the largest number of participants). Therefore, in the following, when we are referring to the *active participants* of a

GameCode, we are actually referring to students who answered to at least 50% of the questions.

7.2.3 *Time tracking*

Resolution path

As it is explained in Section 7.1.1 (Time Tracking), the fragment heartbeat system allowed us to collect an approximation of the time spent on each fragment. This is particularly useful when it comes to estimating the *resolution path* taken by a student, i. e., the order in which they solve a GameCode. For example, it could be interesting to know if they start answering questions before reading theoretical reminders, or vice versa.

However, this system is not perfect: heartbeats are sent every 15s, and when a `fragment_session` document is firtly created in database, the timestamp corresponding to the last heartbeat is set 15s in the future by default. This means that the time spent on a fragment is sometimes overestimated by 15s, and this becomes a problem when we wish to analyze the resolution paths, because each time they switch page, the time spent on a transitory page will always be of at least 15s, which is not true in reality, as they do not really read the content of a page.

In order to remedy this issue, a preprocessing step has been put in place: for each student, when their resolution path is analyzed, if they stay on a step for less than 16s, the step is simply removed from the resolution path. This way, the resolution path is more accurate and does not contain any transitory steps.

Figure 33 shows an example of a resolution path preprocessing. This is a scenario where a student spends 1min50 on the introduction, then transits towards the resolution of the GameCode through the reminders page, but does not actually read it, and the same goes for the last blocks, where they transit through the introduction to get back to the reminders.

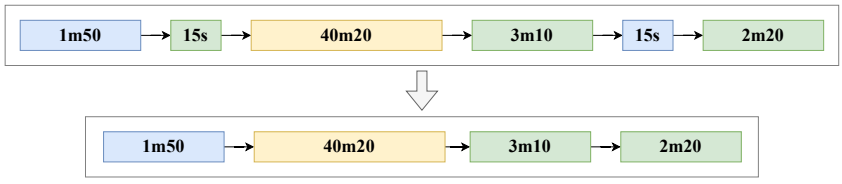


Figure 33: Example of a resolution path correction.

Overlapping fragments

Another issue with the heartbeat system is that every fragment on a GameCode page has its own heartbeat. This allows us to know

which fragment is displayed on the screen with greater granularity (e. g., if we seek information about which precise part of a theoretical reminder has more visibility), but becomes a problem when we would like to analyze the time spent on a page as a whole. Indeed, the time spent on a page is the sum of the time spent on each fragment, and if two fragments overlap, the time spent on the page is overestimated.

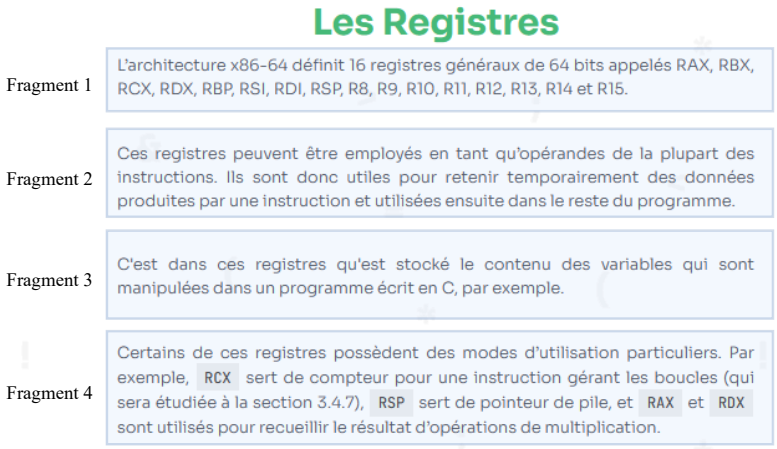


Figure 34: Example of overlapping fragments.

In the example on Figure 34, we observe 4 text fragments displayed in a reminder about CPU registers. The issue would be that, if we try to estimate the time spent on this page, doing a sum of all the time spent on each fragment of this page would not be correct, because if the user stays 20s on the page, the total wrong sum would be of 80 seconds.

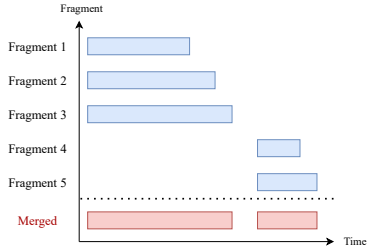


Figure 35: Ex. of merged fragments

A simple solution to this issue was to merge overlapping fragments from a same module (i. e., when we know they are on the same page) with each other, as it is illustrated on Figure 35. In blue are the fragments of a same module (e. g., the introduction), that might be visible all at the same time on the page. The merged equivalent of these fragments, which is an approximation of the time spent on the module, is in red.

8 ANALYSIS AND INTERPRETATION

In this chapter, we are going to present the analysis of the collected data. This analysis will be carried out in three stages:

- *Participation*: description of the subjects of the study (i. e., students in the dataset who participated, actively or not, in various GameCodes), as well as an analysis of which GameCodes seem to be the most attractive;
- *Performance*: description of the participation data collected thanks to the various trackers set up, as explained in Chapter 7;
- *Early analysis of learning analytics (LAs)*: interpretation of the results obtained, and discussion of their implications.

As mentioned in Section 7.2.1, GameCodes will be analyzed separately, and so will the contributions of the students, because each GameCode has a different focus, and appeals to different groups of students. Also, the only contributions that will be taken into account are the ones from *active participants*, i. e., students who answered to at least 50% of the questions of a given GameCode. The only analysis that will take GameCodes collectively into account is in the performance analysis, for information such as the time of the day students work on GameCodes, the time of the week, etc.

8.1 PARTICIPATION

GC	Total Active Part.	CS	Eng.	Other	Age Range	Med. Age
GCA0	4	4	0	0	19-21	19.5
GCA1	7	7	0	0	19-21	20
GCB0	29	7	21	1	18-23	19
GCB1	35	7	27	1	18-24	19
GCB2	22	3	18	1	18-24	19

Table 1: Participants demographics per GameCode

There are two groups of participants: group A (GCA0, GCA1, and GCA2), which are the GameCodes exclusively available to computer science students, and group B (GCB0, GCB1, GCB2), available to computer science students and engineering students. In the following, GCA2 will be excluded from the analysis because it was used by only one student.

Table 1 is a summary of the demographics of the participants in each GameCode. We observe that the median age is around 19 years old, and that GameCodes that were presented to all students (Group B) seem to be mostly used by engineering students.

In total, GameCodes were presented to:

- 100 first year CS students (Group A);
- 255 first year engineering students (Group B without A);
- a total of 355 enrolled students (Group A and B);

Overall, the participation rate is quite low, with a maximum of 35 active participants for GCB1, which is approximately 10% of the total number of students enrolled in the course. This is a limitation of this study, as it is difficult to draw any conclusion about the effectiveness of the GameCodes. However, we can still analyze the data collected and suggest improvements for future studies.

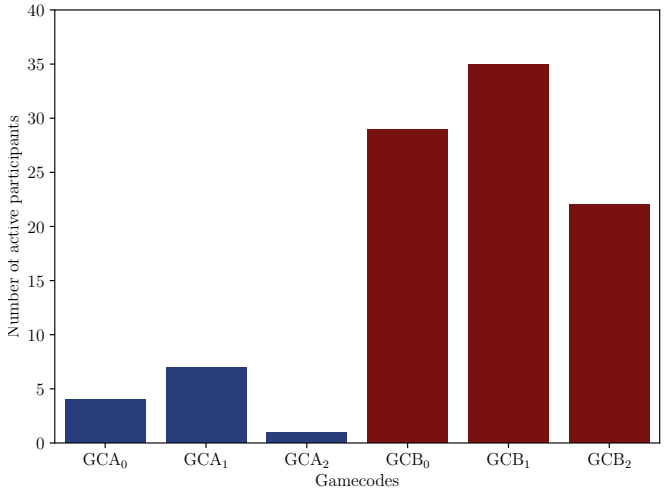


Figure 36: Active participation in the GameCodes

Figure 36 allows a more visual comparison between the participation rates per GameCode. The reasons for the low participation rate are multiple, but the main one is that the GameCodes were not mandatory, and students could choose to participate or not. Also, it is likely that all GameCodes will have a large participation increase a few days before the exam, as students will be more motivated to revise the course content.

Also, Group A consists of GameCodes that were exclusively available to first year CS students, i. e., 100 students, but *only in the context of remedial courses*, to which very few students participated (less than ten students came to class). On the other hand, Group B consists of GameCodes that were available to all first year CS students as well as engineering students, i. e., 355 students, and were often motivated by teaching assistants of the cours to use the platform.

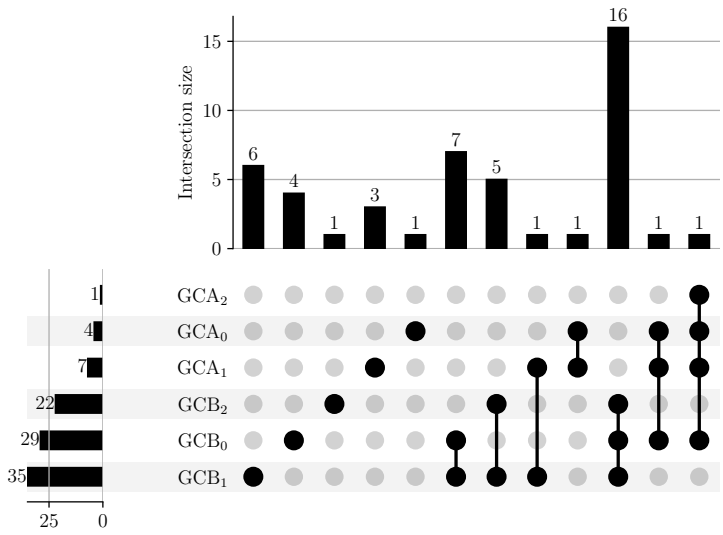


Figure 37: Upset plot of the active participants in the GameCodes

On Figure 37, the bars at the top represent the intersection size, showing how many participants are common across multiple GameCodes. The largest intersection (16 participants) is those who participated in GCB0, GCB1, and GCB2. Again, in view of the very low participation rate in Group A, there is no conclusion to be drawn on the intersection of Group A GameCode participation with Group B. Nevertheless, we can see from this plot that a significant proportion of Group B tended to participate in all their GameCodes.

In terms of attractiveness, we can see that the most attractive GameCode seems to be GCB1, which was the first GameCode about Assembly, followed by GCB0, which is about binary encoding, and finally, GCB2, which is the second GameCode about Assembly, and which has a few less participants than the other two, probably because it is the last one to be published.

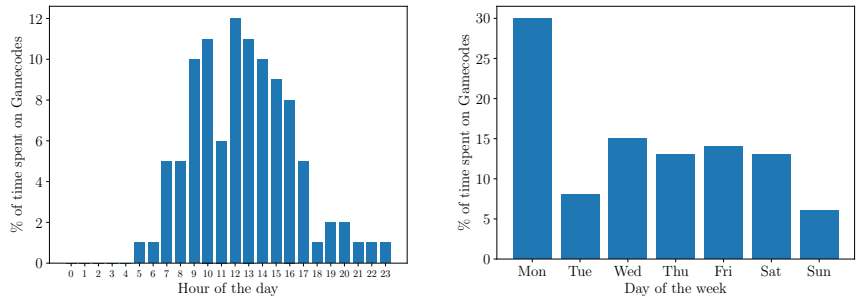
In summary, we observe that there is not much to analyze about Group A, because of the very low participation rate. On the other hand, Group B has a higher participation rate, and the students who participated in one GameCode tended to participate in all of them, which indicates that they were interested in the concept.

8.2 PERFORMANCE

In this section, a performance analysis is presented. This analysis is about the collected participation data, which consists of *how* students actually worked on GameCodes, in which order, at what time of the

day, and how much time they spent on them. These data was collected thanks to the various trackers set up, as explained in Chapter 7.

8.2.1 *Time distribution*



(a) Time of the day students worked on GameCodes

(b) Day of the week students worked on GameCodes

Figure 38: Time of the day and day of the week students worked on GameCodes

Thanks to the gathered information on the total time spent on each GameCode by students, we can see at which times of the day and at which times of the week this total time spent on the GameCodes was distributed. On Figure 38a, we can see that the majority of time spent on the GameCodes is between 9am and 4pm, with a peak at 12pm. On Figure 38b, we can see that the majority of time spent on GameCodes is on Mondays. These results are consistent with the fact that students seem to work on GameCodes on the same day they have the *Computer Organization* course, on Monday, and they might read them before or after their classes, to prepare for it or to make links with the subject matter. On the other hand, the low participation rate on Tuesdays is also coherent with the fact that students might not want to get back into GameCodes the day after they had a whole two hours of *Computer Organization*, followed by two hours of practice.

This analysis might seem trivial, but it is important to understand the behavior of students in order to improve GameCodes, and to know when students are more likely to work on them, in order to adapt the content to the time of the day or the day of the week. It offers insights for optimizing the platform to better align with student study habits, so that we will know about when would be the best time to send notifications to students, for instance, in coordination with supervisory staff.

8.2.2 *Time spent per Gamecode*

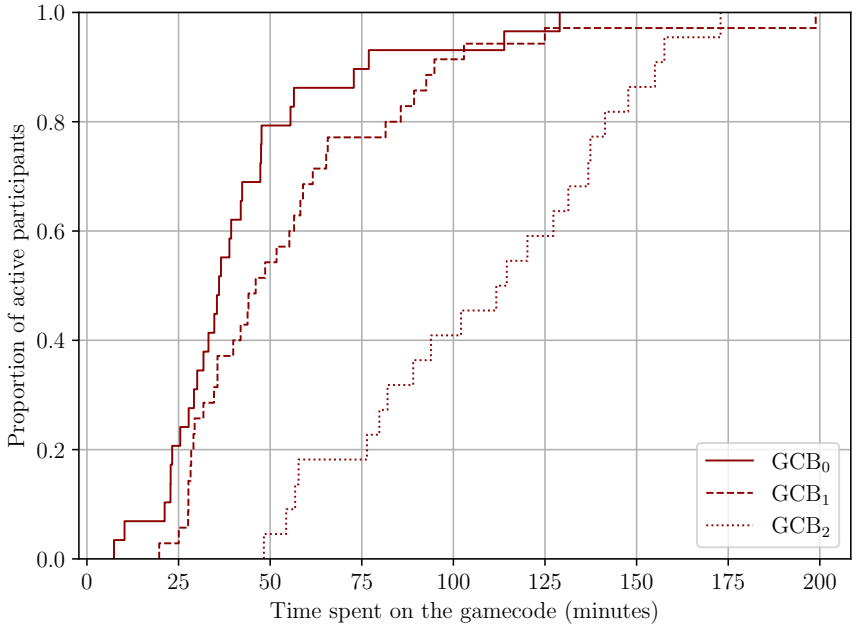


Figure 39: CDF of the time spent on each GameCode from Group B

Figure 39 is a CDF of the time spent on each GameCode by the students of Group B:

- GCB0: this GameCode is the easiest one, about binary encoding, and the time spent by participants is not very high, with more than 80% of the students spending less than one hour on it. This is consistent with the fact that binary encoding is simpler than exercises on Assembly, and students may be more familiar with the concept.
- GCB1: this GameCode being the first one about Assembly, we observe that students seem to spend a bit more time on it. However, the difference with GCB0 is not very significant, which could be explained by the fact that the exercise in this GameCode was actually also done in class.
- GCB2: this one was the second GameCode about Assembly, and we can see that the time spent on it is way higher than the other two, with more than 50% of the students spending nearly two hours on it. This could be explained by the fact that the exercise was more difficult, and has not been previously done in class. Also, it trains students on some key Assembly concepts (e. g., usage of the stack, function calls, etc.) that they were not as familiar with before.

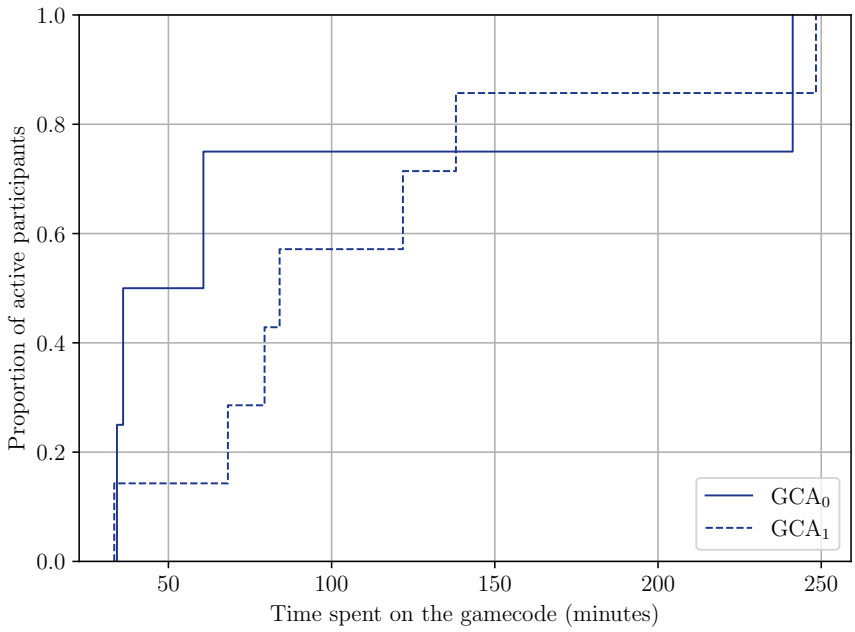


Figure 40: CDF of the time spent on each GameCode from Group A

Figure 40 is a CDF of the time spent on each GameCode by the students of Group A. It is difficult to draw any conclusion or comparison with Group B, because of the very low participation rate (4 students for GCA0 and 7 for GCA1).

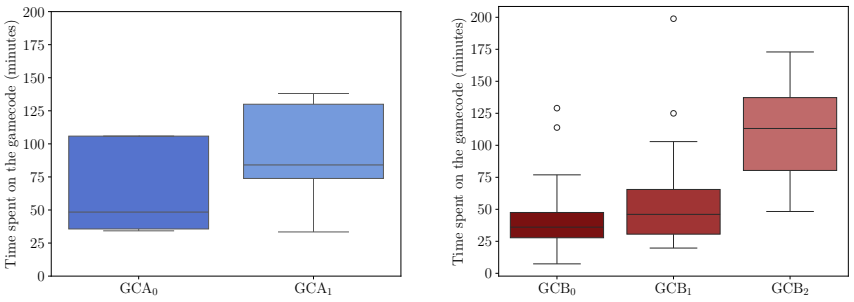
While it is difficult to make a comparison with Group B, some differences can be noted:

- The CDFs for Group A show more pronounced steps, indicating that the engagement times are more varied and possibly more influenced by individual differences in working pace or understanding.
- Group B's CDFs tend to show smoother curves, suggesting more consistent engagement patterns among a larger number of participants.

The boxplots in Figure 41 provide additional insights into the time spent on each GameCode. They show the median, interquartile range (IQR), and outliers for each GameCode, complementing the CDFs by highlighting central tendency and variability more clearly.

For Group A, GCA0 has a median time of around 50 minutes and GCA1 has a median time closer to 80 minutes. The IQR for GCA1 indicates considerable variability among participants.

For Group B, GCB0 has a median time around 50 minutes, GCB1 around 75 minutes, and GCB2 around 125 minutes. The larger IQR for GCB2 indicates the most variability in engagement time. Outliers



(a) Boxplots of the time spent on each GameCode from Group A

(b) Boxplots of the time spent on each GameCode from Group B

Figure 41: Boxplots of the time spent on each GameCode

in GCB0 and GCB1 show that some participants spent significantly more time than the typical range.

A high variability when it comes to the time spent on the GameCodes is expected, as each student has a different pace of work, and some may have more difficulties than others.

Again, this could be, in future work, an interesting comparison indicator to observe the differences in engagement between various GameCodes, and it could give us an idea of the difficulty of the exercises in each GameCode, from the point of view of the students. Observing the whiskers and outliers could also give us an idea of whether a GameCode is difficult to a majority of students, or only to a few.

8.2.3 *Learning paths*

In order to get an overall view of how students solve GameCodes, it may be useful to illustrate what we call their "learning path", in the form of an *Alluvial diagram*. An Alluvial diagram[1] is a type of flow diagram, in which the width of the arrows is proportional to the flow rate. In the context of this study, the flow rate is the number of students coming from one part of a GameCode to another, the parts being the introduction, general reminders, statement, resolution steps, and conclusion. It is composed of 10 "moves", which all represent a moment during their resolution of the GameCode, from the introduction to the conclusion, and where groups of students are at each moment.

This visualization can help us understand how students navigate through the GameCode, and whether they follow the intended learning path. It can also help us identify bottlenecks in the GameCode, i. e., parts where students tend to get stuck, or where they spend more time than expected.

1 https://en.wikipedia.org/wiki/Alluvial_diagram

GCB1 - Assembly level 1

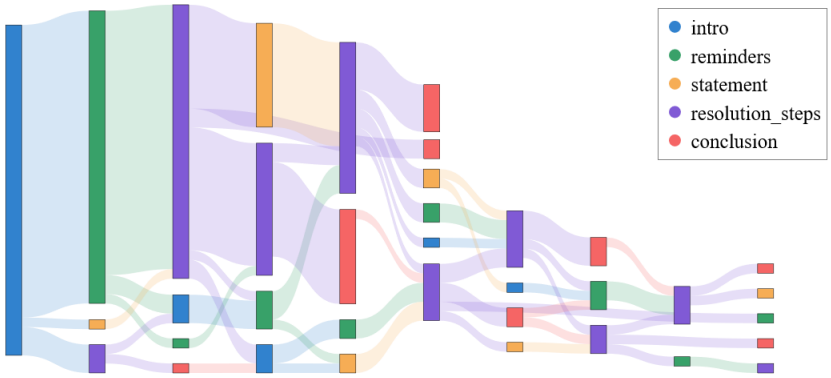


Figure 42: Alluvial diagram for GCB1 (Assembly (1))

To illustrate the importance of this analysis, the GameCode first from Group B, about Assembly language, is a good candidate, in view of its 35 participants and its non-trivial subject.

The initial idea behind GameCodes was to allow students to choose their own path and go back and forth between the different parts of the GameCode. And indeed, we can see on Figure 42 that the flow is not linear, and that students tend to go back and forth between the different parts of the GameCode. This is coherent with the initial idea, and it is a good sign that students are able to navigate through the GameCode as they wish.

Another interesting observation from Figure 42 is that the reminders seem to be useful for students, as they spend some time on them before going to the statement. This is a good sign, as it means that students are not skipping the reminders, and are actually reading them before starting the exercise. This is coherent with the fact that the reminders are there to help students remember some key concepts before starting the exercise, and even though they are not mandatory, students seem to find them useful.

This is all the more apparent when we look at Figure 43. This figure illustrates the way students solve the second resolution step of the GameCode, which is about Assembly code implementation. It illustrates in which part of the resolution step they are, over 10 moves, and even though a resolution step is composed of a statement, reminders, questions, and a conclusion, note that the statement is not present on the diagram because it is always visible on every page of a resolution step.

For all intents and purposes, here is the statement of this resolution step: "You are asked to write an assembler routine which takes

Figure 43: Alluvial diagram for the second Resolution Step of GCB1 (Assembly (1))

as input arguments: the address of an array of bytes, the size of this array, and a constant value. The routine you have to implement consists of filling the array with this constant value. For example, if we call the routine `fill_tab`, and we call it in a C program, an example of use would be: `fill_tab(tab, 10, 'x')` to fill the array `tab` of size `10` with the character `'x'`."

On Figure 43, we observe that most of the 35 students seem to read the reminders before starting to solve the exercise. Also, another interesting observation is that, at some point, maybe when they finished the exercise, there is a lot of back and forth between the question and the conclusion. This is an expected behavior, as the conclusion contains a suggested answer for the code they have to implement, they might want to check if their code is correct, and if it is not, they might want to go back to the question to see what they missed.

GCBo - Binary encoding

Another example of this type of *learning path* can be seen in the GameCode on binary encoding. This one differs from the first one on Assembly, particularly as regards the reading of theoretical reminders.

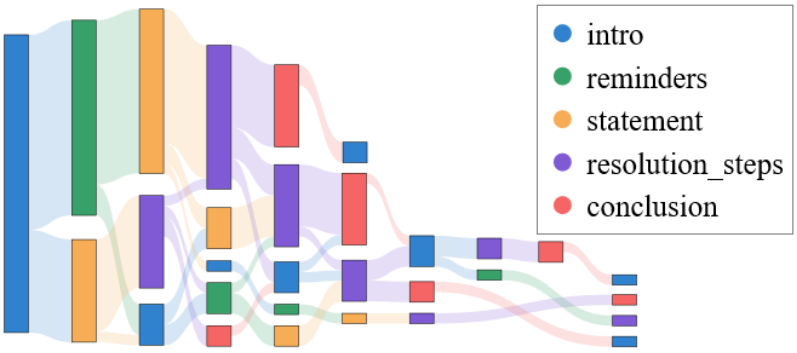


Figure 44: Alluvial diagram for GCBo (Binary encoding)

On Figure 44, we can see that students tend to skip the theoretical reminders more than they did with the GameCode about Assembly, and go directly to the statement. This is coherent with the fact that binary encoding is a simpler concept than Assembly, and students might not feel the need to read the reminders before starting the exercise.

It means that students are able to navigate through the GameCode as they wish, and that they are not forced to read the reminders if they do not want to. This is actually quite an important observation, as it shows that even though the second page after the introduction is about the reminders, students are not forced to read them, and can go directly to the statement if they feel confident enough. It was a doubt we had, given that the structure of a GameCode seems fairly linear in its design.

Therefore, this analysis could, with a larger dataset, and more GameCodes about different parts of a course, allow us to have an indicator of the level of confidence of students in the subject matter.

8.2.4 *Inferring from learning analytics*

It might be interesting to analyze relationships between these various learning analytics (LAs), in order to have a qualitative evaluation of

the effectiveness of GameCodes on students' success. Among them, the most interesting ones in the context of this thesis would be:

- The success rate of each GameCode for a given student;
- The success rate of an individual resolution step for a given student;
- Time spent on global theoretical reminders;
- Time spent on specific resolution steps reminders;
- Time spent on a GameCode;
- Number of compilation trials before achieving a right answer;
- Whether the student has read the reminders or not;
- Whether the student has read the reminders before answering the questions or not.

However, in the field of pedagogy and education, the impact of pedagogical tools on a student's success is extremely difficult to interpret due to a series of external factors, as has been frequently observed to date in various studies (e. g., [15, 22, 23, 27, 31]). These studies highlights that external factors such as course structure, pedagogical design, and the diversity of learners significantly influence learning outcomes and engagement. These factors can complicate the interpretation of the effectiveness of pedagogical tools, as individual learner characteristics and context play critical roles in their educational experiences [27].

For instance, if we would like to observe the impact of the reading of the reminders on the success rate of a given resolution step, if a correlation was to be found, it would be difficult to interpret it, as each student has different knowledge of the subject matter, and some might not need to read the reminders to succeed, while others might need to read them to understand the exercise. This is just one factor among many others; we could also mention the fact that some students are more focused than others, for instance.

Another difficulty to take into account when trying to infer from learning analytics is the fact that a qualitative dataset of students is needed to draw any conclusion, however small the effect might be. The population does not necessarily have to be a large one (however, it has to be large enough, as often stressed by several studies, e. g., [1, 2, 11, 14, 21]), but rather one that can be observed in a controlled environment, with a homogeneous background, and with a clear objective [9].

Taking these challenges into account, it may be interesting to look at a short analysis of the various learning analytics collected, and interpret them to give an idea of the kind of analysis that might be

possible with more data and a more controlled environment. The remainder of this section will be devoted to this, and no conclusions will be drawn, but rather some food for thought for future studies.

Review of learning analytics

A good candidate for a review of different learning analytics is GCB0, which is the GameCode about binary encoding, and which had 29 students participating in it. The following analysis will be based on the data collected for this GameCode, and for two of its resolution steps.

The first resolution step (RS1) consists in 14 MCQ type questions about the feasibility of the operation (i. e., whether the operation is possible or not), and the second resolution step (RS2) consists in 6 questions where students were asked to encode various binary numbers. These two resolution steps also contain two different theoretical reminders. This GameCode also contains two global theoretical reminders about a comparison of the decimal encoding system vs. the binary one, and another one about how to make a written calculation of binary numbers.

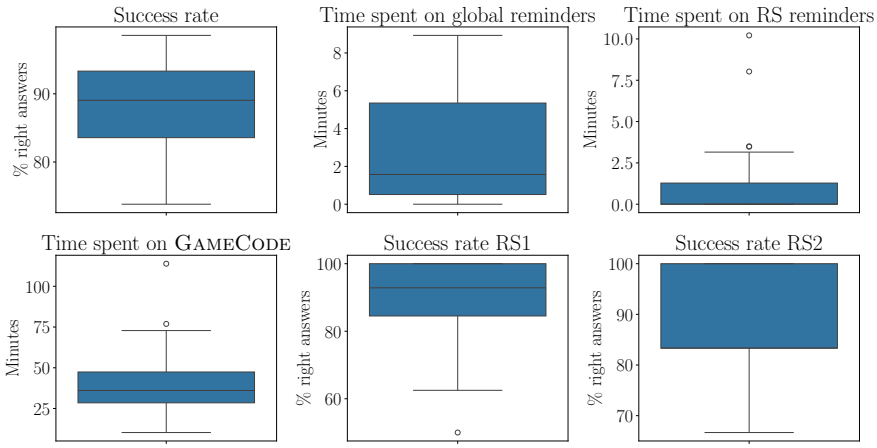


Figure 45: Boxplots of the LA for GCB0

From Figure 45, we observe the following:

- The success rate is generally high, with most students achieving between 80% and 90% of right answers.
- The median time spent on global reminders of the GameCode is around 4 minutes, with a range from 1 to 8 minutes.
- The median time spent is very low, around 0.5 minutes, with most students spending less than 2 minutes, which makes sense, given that the reminders are very short. There are also some outliers who spent more than 3 minutes on the reminders.

- The median time spent on the GameCode is around 25 minutes, with a range from 20 to 30 minutes.
- The success rate for RS1 is generally high, with most students achieving between 80% and 100% right answers.
- The success rate for RS2 is also high, with most students achieving between 90% and 100% right answers.

From these observations, we could infer that this group of students seems familiar with the concept of binary encoding, as the success rate is generally high. Also, the reminders of this GameCode did not attract much attention from students, as the time spent on them is very low, and the success rate is high.

Another question that could be asked is whether the students who read the reminders before answering the questions have a higher success rate than those who do not, i. e., whether the reminders appear to be useful to them or not, while bearing in mind that a multitude of external factors exist, and that we should not jump to any conclusions. For the sake of setting an example of what could possibly be done with a better dataset, let us demonstrate how this could be done.

The easiest way to do this would be to look for a correlation between the time spent on the reminders *before answering the questions*, and the success rate of the resolution steps. However, because of the reasons mentioned above, after trying to correlate the global success rate of the GameCode with the time spent on reminders before answering, the correlations were either very low, or not significant at all. This is not surprising, as the success rate of a resolution step is not only influenced by the reading of the reminders, but also by different reasons mentioned above.

For instance, the correlation between the reading of the global reminders of the GameCode with the global success rate is around -0.3348, and the p-value is 0.0879. This is not significant, and the correlation is very low. The same goes for the correlation between the reading of the reminders of RS1 and RS2 with the success rate of RS1 and RS2, respectively, which was not significant.

Another useful way to try to observe relationships between these variables is to generate a PCA biplot, which is a scatterplot that shows in a very visual way the relationships between the variables.

Figure 46 is a PCA biplot of the following variables:

The names have been shortened for better readability on the plot.

- T_GR: time on global reminders;
- T_RSR: time on resolution steps reminders;
- T_GC: time on GameCode;
- R_R1: whether the student read the reminders of RS1 or not;

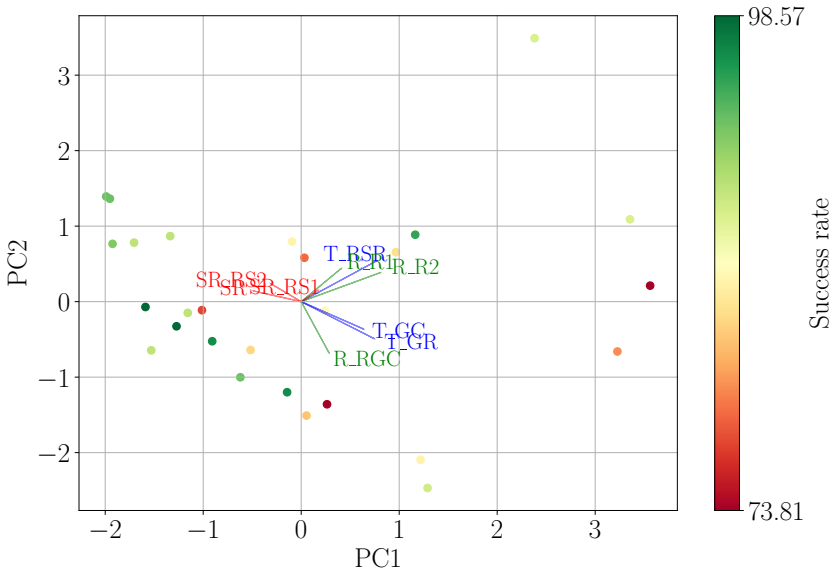


Figure 46: PCA biplot of the LA for GCBo

- R_R2: whether the student read the reminders of RS2 or not;
- R_RGC: whether the student read the global reminders of the GameCode or not;
- SR: global success rate;
- SR_RS1: success rate of RS1;
- SR_RS2: success rate of RS2.

A PCA biplot is a scatterplot produces by PCA, which is a technique used to reduce the dimensionality of a dataset, and to visualize the relationships between the variables. All of the scatter points are the students, and the arrows are the variables. The longer is an arrow, the more it influences the position of the students in the scatterplot. Also, this plot allows us to visualize the relationships between the variables, and to see which variables are somehow correlated with each other. If two arrows are in the same direction, it means that the two variables are positively correlated, if they are in opposite directions, it means that they are negatively correlated, and if they are perpendicular, it means that they are not correlated. Finally, the color of the points represents the success rate of the students, which varies from dark green (higher success rate) to dark red (lower success rate).

From this plot, we can observe some very obvious correlations:

- The global success rate and the resolution steps success rates are positively correlated, which is expected, as the success rate of the GameCode is the sum of the success rates of the resolution steps;

- The time spent on the GameCode and the time spent on its global reminders are also positively correlated, which is coherent, as the global reminders are part of the GameCode;
- The time spent on resolution steps reminders and the boolean variables indicating whether the students read the reminders are positively correlated, which is also coherent, as the time spent on reminders is influenced by the fact that the students read them or not.

We also observe the inverse correlation between R_RGC (i. e., whether the student read the global reminders or not) and the different success rates, as it was mentioned previously. This might seem as a surprising result, since, if we were to have a relevant correlation, this would mean that the more students tend to read reminders, the lower their success rate would be. However, this could be explained by the fact that students who are less familiar with the subject matter might tend to read the reminders more, and still have a lower success rate. This is yet another example of how difficult it is to interpret learning analytics, and is not a conclusion to be drawn, but rather an example for future studies.

The last element of this plot that has not been mentioned yet is the color of the points, i. e., the global success rate. Since every active student has a score between 70% and 100%, there is not much to analyze from this, and no distinct clusters can be observed.

8.3 PYGMALION EFFECT 2.0

Although this data is unfortunately not yet conclusive, it is expected to be in the very near future and to bear fruit from the start of the 2024-2025 academic year. To justify this hope, we refer to the study by Brieven et al. [5], which describes lessons learned from 6 years of remote programming challenges on CAFÉ 1.0.

Among these lessons, the authors looked at the relationships between the completion of challenges (i. e., graded online assignments with automatic feedback), and students' exam grades, over the last six years. They found that there were two very interesting correlations in this context: one with the number of hours before the first submission of a challenge (students are allowed three attempts), and their exam grades, and with the number of challenges completed during the year (they are all compulsory, with the exception of one, for which they can use a "joker"), and their exam grades.

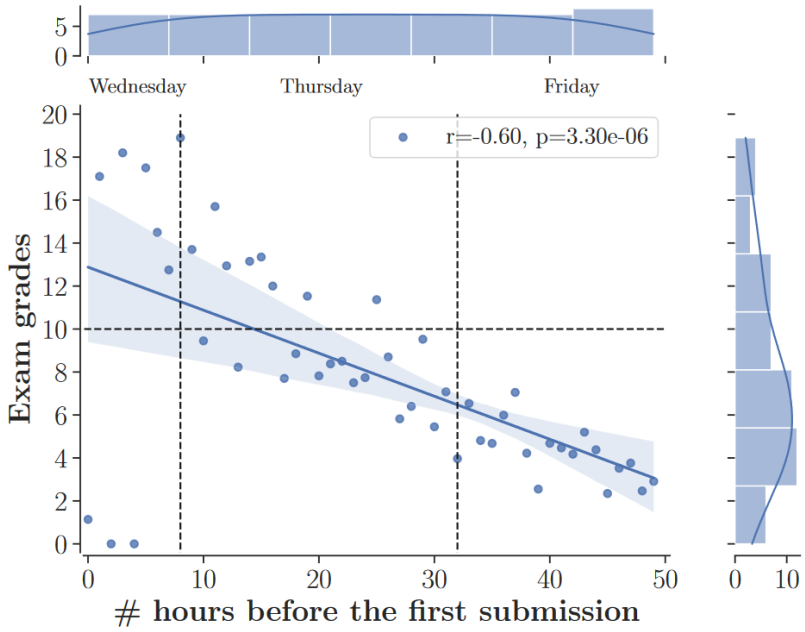


Figure 47: Correlation between "last minute" trade and learning rate (aggregation over the six years of interest). The histograms (top and right of the graph) gives the number of students per X or Y value. Grades range within [0; 20], 10 being the success/failure threshold (illustrated by the horizontal dashed line). [5]

The first one, on Figure 47, shows the relation between the number of hours before their first submission over all challenges, and their exam grades. It shows that the later the students submitted the first version of their solution, the lower the exam grades they obtained, eventually, and that claim appears reliable seeing the very low p-value behind that analysis [5]. The second one, on Figure 48, arrives

to a similar conclusion, but with the number of challenges completed during the year.

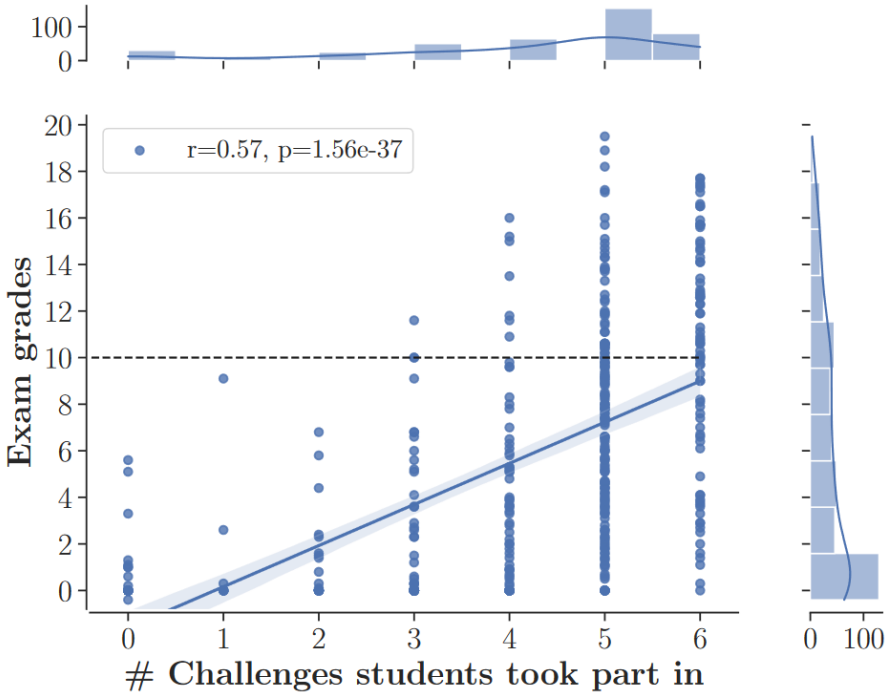


Figure 48: Correlation between the participation to the Challenges and the learning rate (aggregation over the six years of interest). The histograms (top and right of the graph) gives the number of students per X-Axis or Y-Axis value. Grades range within [0; 20], 10 being the success/failure threshold (illustrated by the horizontal dashed line). [5]

This part of the study not only highlights new means of using the LAs collected by the challenges to correlate them with exam results, but also enables future proactivity and prediction of student results. Indeed, from September 2024 onwards, work will be carried out to identify early on in the semester those students who are in a "grey area" in terms of projected exam points, i. e., students who are likely to score between 7 and 10 out of 20 on the exam, and it will be possible to target this group of students, thanks to these described methods, in order to provide them with additional resources and more personalized support to help them achieve greater success.

The reason why this "grey area" of students would be targeted is because there are a number of studies, including some by the *Education Endowment Foundation* (EEF), demonstrating promising results when targeted support in small groups, or one-to-one tuition, can be highly effective for pupils across different attainment levels. Also, interventions aimed at students with mid-range predictions (e. g., those scoring between 7 to 10) are likely to yield better returns in terms of academic improvement compared to those targeted at students

predicted to score very low (e. g., below 5) [12]. Providing targeted support to students who have *the potential* to improve their grades significantly (e. g., moving from a 7 to a 10 out of 20) can be more effective and motivating for both students and educators. These students are often more responsive to interventions because the gap to achieving higher performance is smaller and more attainable.

Thanks to Challenges, and now, in a second phase, thanks to GameCodes, we will have better means, by using a multitude of new LAs, of spotting patterns in student behavior that will enable us to better categorize their level of learning and gain a better insight into their work methods and organization, and this will offer us a lever we can pull to best support them towards success.

There is a clear parallel between GameCodes and Challenges, as the same kind of LAs can be collected, however, data collected by GameCodes allows for a better granularity, and more details about the way students organize their work, and the way they solve exercises. This is a very promising aspect of GameCodes, and it is likely that the same kind of correlations that were found with Challenges will be found with GameCodes, and that they might even be more accurate, and allow for more personalized support for students.

8.4 CONCLUSIONS ON LEARNING ANALYTICS

Through this analysis, we have reviewed some of the learning analytics that have been collected. The idea of this chapter was twofold: on the one hand, to detail the data collected and perform a surface analysis on them, on the participation and the performance of students, without drawing any conclusions about anything for the reasons submentioned, and on the other hand, to give an overview of the potential these data would represent if a larger number of participations were available.

In the context of this thesis, statistical analysis of these data is rather weak, and more for descriptive purposes than for inference. Indeed, it is important to remember that the online GameCodes project is only in its prototype stage, and that this first version gives us an overview of the data already available, without being able to exploit them fully.

Details of possible future analyses and improvements with respect to learning analytics are given in Chapter 9.

Part IV

FUTURE WORK AND CONCLUSIONS

This last part provides an overview of the potential improvements that can be made to the GameCodes platform, as well as a summary of the work carried out during this master's thesis. It also provides a conclusion to the work, as well as a summary of the results obtained, and a discussion of the potential for future work in this area.

9 FUTURE IMPROVEMENTS

This master's thesis has been completed in just under a year, at a steady pace, but still with a substantial workload alongside the completion of this project. It serves more as a cornerstone, or basis for future work, than a conclusive piece of work in its own right.

This chapter details possible improvements to the work carried out, as well as its current limitations. These improvements can be divided into two categories: implementation improvements and data analysis improvements.

9.1 IMPROVING THE IMPLEMENTATION

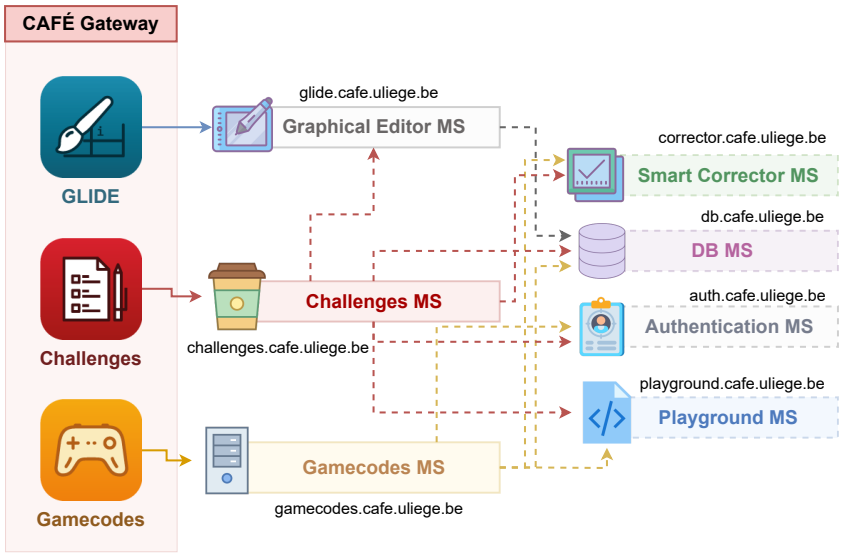


Figure 49: Future CAFÉ ecosystem with all microservices – Dotted arrows represent a MS being used by another part of CAFÉ

Microservices

As far as the implementation of GameCodes in the CAFÉ 2.0 ecosystem goes, the work is not yet completely finished. As explained in the section on implementation, we need to complete the breakdown of the ecosystem into microservices, as it is illustrated on Figure 49. For the time being, the GameCodes implementation is modular and easily deployable via Docker. However, the main `cafe.uliege.be` en-

gine is still installed directly on the machine, as is the authentication module. Also, the fact that CAFÉ's main server is written under Laravel, which is a PHP framework, instead of React, as GameCodes are, is a problem because there is a significant overhead in converting information structure between the JSON format, which is usually used by React, and the PHP language, where dictionaries and typing are sometimes "random", or in any case, ill-defined, due to the implicit modularity of MongoDB and JavaScript. Indeed, it is possible to declare object attributes "on the fly" in JavaScript, and in MongoDB documents, and there is a significant workload to maintain at PHP level to keep a consistent structure across all the different parts of the implementation. This issue will be resolved in the future when CAFÉ's main server is converted to React.

New heartbeat system

Secondly, the way in which the data is collected is not always optimal. A major limitation of the GameCode fragment heartbeat system is the lack of precision regarding the time spent by students on the fragments. This limitation results from the fact that the fragments send a heartbeat to the server, when they are visible on the user's screen, every 15 seconds, which implies that the time spent on each fragment is a minimum of 15 seconds, even if the student is only navigating one page to access another. One way of solving this problem would be to use web sockets instead of heartbeats. Indeed, if each client opens a persistent connection with the server for the duration of their visit to the site, it is possible to measure the time spent on each fragment with an accuracy of seconds, or even less, since each session will be closed when the connection between sockets ends. For added precision, it is also possible to combine the two systems by adding a heartbeat system, this time originating from the server and targeting the client socket, to check whether it is still present. This would enable us to better understand student behavior, and better adapt GameCodes to their needs.

Navigation

Another observation about the GameCodes website is that students do not have the freedom of navigation we desired them to have. They can freely navigate between GameCode modules, using a consistently visible navigation bar at the top of the screen, but they are not encouraged to do so, and will generally solve a GameCode in a linear fashion. One idea that was proposed during the implementation phase, but not realized due to lack of time, was to allow them to use a map that shows them where they are in a GameCode, which would allow them to move around any stage from a visually stimulating interface.

In the same place, the various resolution steps and their contents would be clearly displayed, along with the other general modules of a GameCode.

Database structure

Also, the database structure is still not ideal. Although a "factory" design pattern is applied, GameCode fragments quickly become tedious to analyze, as they contain too much information in the same document, depending on their type. Better modularity is to be expected in the future.

Gamification

Finally, another area for improvement is gamification. The fact that the platform offers "GameCodes " instead of "online exercises" is already a first step in this direction, but we need to make the platform much more attractive, and give students the impression that they are dealing with a video game rather than exercises that might seem boring to them. Various solutions can be applied to correct this, such as a more gamified graphical interface, experience point systems, earnable virtual currency, etc., and will be implemented in the years to come.

9.2 IMPROVING LEARNING ANALYTICS

The main problem encountered when analyzing the learning analytics was that very few students took part in the exercises. There are various reasons for this:

- GameCodes were not deployed quickly enough;
- Perhaps the GameCodes were not attractive enough;
- Deployment should have taken place earlier in the year, and not in the middle of the second quarter;
- There are no incentives for students to participate in GameCodes, such as bonus points or rewards, for example.

In future work, we will need to think about adding stimuli to encourage students to take part in GameCodes, such a bonus points, or to consider GameCodes as part of students final marks. It would also be interesting to deploy GameCodes in other courses, or other universities, to observe if the results are the same, and to have a higher number of participations. We will also need to have a series of GameCodes ready for use at the start of the academic year.

Gamecodes Perception

Another encountered problem during the analysis was that our organization did not allow us to ask for feedback on the *perception* of GameCodes. Indeed, in addition to student participation and performance, having qualitative feedback on the platform from students would have enabled us to get feedback from them on how to use it, and how best to improve it. Some students have contacted us to testify that they really appreciate the platform, and that they would like this type of exercise to be more popular at university. But these students are few in number, and while positive feedback is always appreciated, it is not interpretable in the context of this thesis.

Time frame

Then, the time frame for data collection is very short. It is difficult to draw conclusions based on data collected over a half of a single academic year, and it would be interesting to see if the results are the same over several academic years.

Predictive Analytics

Finally, as it was explained in Section 8.2.4 (Inferring from LA) and Section 8.3 (Pygmalion effect 2.0), stronger links must be established between the LA collected and the learning outcomes of the students. In particular, links between Challenges and GameCodes must be done, and the impact of GameCodes on the final marks of the students must be measured, in order to put in place a system that will allow us to help as best as possible students who are stuck in the "grey zone", between success and failure. The idea would be to create predictive models to identify students at risk of failure early enough in the semester by using historical data and flag students who may need additional support.

Real-time analysis

Currently, all analyses have been carried out a posteriori, via Python scripts. Another idea for improvement would be to implement real-time analyses, which would enable us to react more quickly to students' needs, either directly from the GameCodes platform, or from a microservice in its own right. This would give teachers a real-time overview at any time during the term, enabling them to see where students are in their courses. This implementation idea is already a work in progress, and goes by the name of CAFÉ's "progress tracker".

10 CONCLUSIONS

The aim of this master's thesis was to develop and implement a web version of GameCodes, and integrate it into the CAFÉ 2.0 ecosystem in order to enhance student engagement and learning outcomes in computer science. This objective was achieved through the creation of this interactive platform, which enabled us to collect an initial series of learning analytics on student interactions with the system, which have been processed and presented more in the form of an exploration of potential, creating links with the data already collected over the past six years through the programming Challenges, already existing in CAFÉ 2.0.

After more than a year in development, we have deployed GameCodes to 355 computer science and engineering students, without having to modify the platform after deployment, as no bugs were reported or observed. They have used it to practise for their exams, to review course material, and they allowed us to collect initial data on the use of the platform. Some of them have contacted us to express their satisfaction and gratitude with the platform.

Although the amount of data collected is not sufficient to research correlations and make predictions about their success, it is a first step in this direction. It is a step forward and an open door to rethinking the way university courses are taught, by incorporating a principle of gamification, monitoring, and analyzing learning analytics to help them learn better, while enabling professors to learn more about their students.

The implementation of this platform and the various microservices surrounding it is a cornerstone for future research into the application of computer science to support pedagogy, and will, in the near future be a very concrete means of helping a large number of students to improve their working and organisational techniques within the university.

Part V

APPENDIX

A

DATABASE MODELING

A.1 FRAGMENTS IN DATABASE

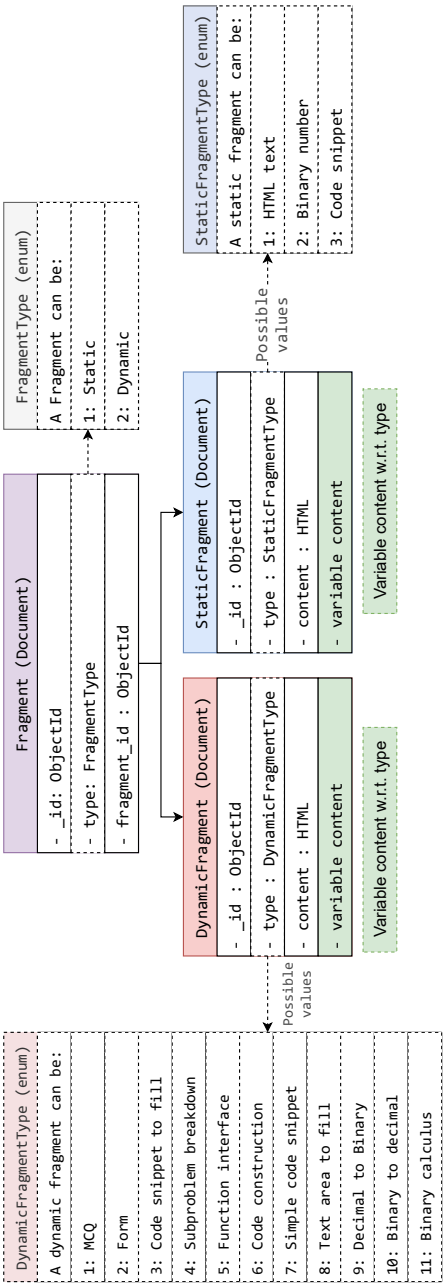


Figure 50: Illustration of the database model for fragments.

A.2 CAFÉ 2.1 ECOSYSTEM

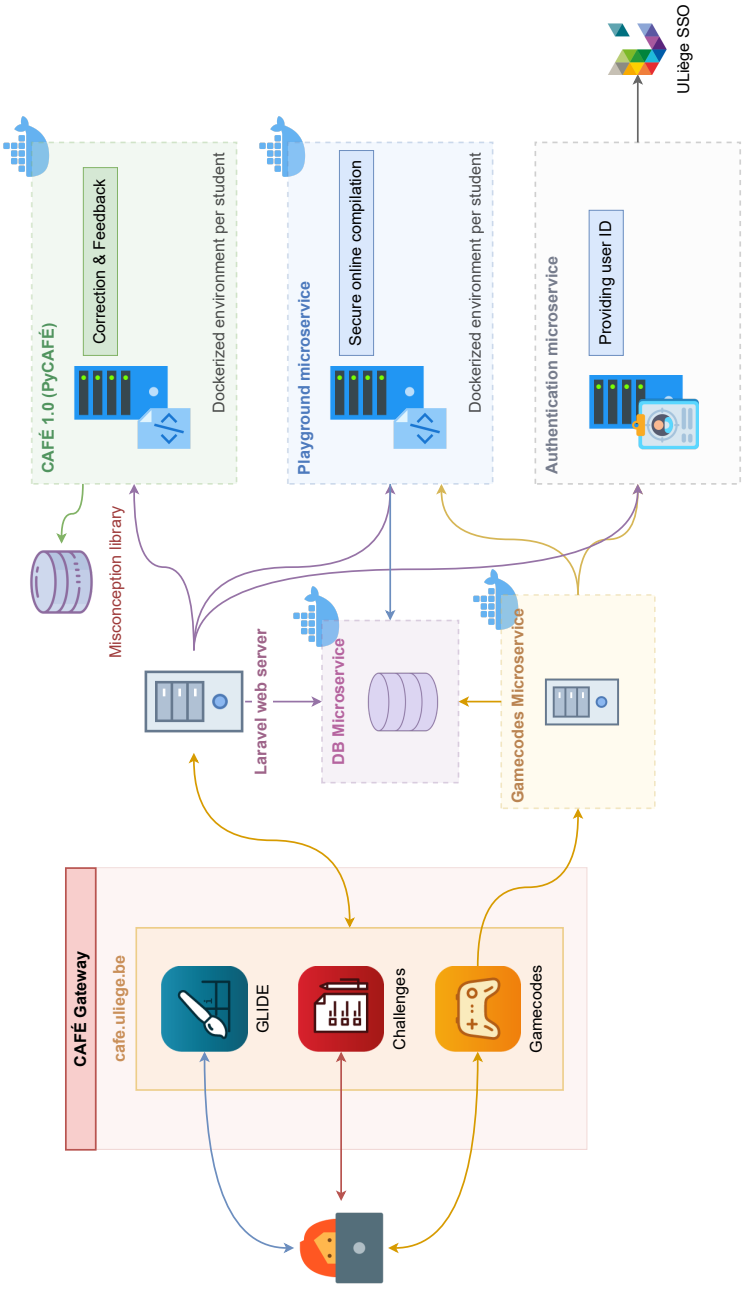


Figure 51: Section 3.3 – Diagram of the current CAFÉ 2.0 ecosystem, after adding GameCodes

BIBLIOGRAPHY

[1] S. Ahmad, A. S. Mohd Noor, A. A. Alwan, Y. Gulzar, W. Z. Khan, and F. A. Reegu. "eLearning Acceptance and Adoption Challenges in Higher Education." In: *Sustainability* 15 (2023), p. 6190. DOI: 10.3390/su15076190.

[2] L. Alzubaidi, J. Bai, and A. Al-Sabaawi. "A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications." In: *Journal of Big Data* 10 (2023), p. 46. DOI: 10.1186/s40537-023-00727-2.

[3] T. Beaubouef, R. Lucas, and J. Howatt. "The UNLOCK system: enhancing problem solving skills in CS-1 students." In: *SIGCSE Bull.* 33.2 (2001), pp. 43–46. ISSN: 0097-8418. DOI: 10.1145/571922.571953. URL: https://doi.org/10.1145/571922.571953.

[4] G. Brieven, V. Baum, and B. Donnet. "Tartare: Automatic Generation of C Pointer Statements and Feedback." English. In: *ACM 26th Australasian Computing Education Conference (ACE)*. Sydney, Australia: ACM, 2024. DOI: 10.1145/3636243.3636264.

[5] G. Brieven, S. Liénardy, and B. Donnet. "Lessons Learned from 6 Years of a Remote Programming Challenge Activity with Automatic Supervision." English. In: *European Distance and E-Learning Network (EDEN)*. Springer, 2022.

[6] G. Brieven, S. Liénardy, L. Malcev, and B. Donnet. "Graphical Loop Invariant Based Programming." English. In: *Proceedings of Formal Methods Teaching Workshop (FMTea)*. Springer, 2023.

[7] G. Brieven, L. Malcev, and B. Donnet. "Practicing Abstraction Skills Through Diagrammatic Reasoning Over CAFÉ 2.0." English. In: *IEEE Global Engineering Education Conference (EDUCON)*. Kos, Greece: IEEE, 2024.

[8] M. Clercq, B. Galand, and M. Frenay. "Transition from high school to university: a person-centered approach to academic achievement." In: *European Journal of Psychology of Education* 32 (Mar. 2016). DOI: 10.1007/s10212-016-0298-5.

[9] E. Dimitriadou and A. Lanitis. "A critical evaluation, challenges, and future perspectives of using artificial intelligence and emerging technologies in smart classrooms." In: *Smart Learning Environments* 10 (2023), p. 12. DOI: 10.1186/s40561-023-00231-3.

[10] S. Edwards and M. Pérez-Quiñones. "Web-CAT: automatically grading programming assignments." In: vol. 40. Aug. 2008, p. 328. DOI: 10.1145/1384271.1384371.

[11] X. Han. "Evaluating blended learning effectiveness: an empirical study from undergraduates' perspectives using structural equation modeling." In: *Frontiers in Psychology* 14 (2023). Original Research article. Sec. Educational Psychology. Front. Psychol., 18 May 2023, p. 1059282. URL: https://doi.org/10.3389/fpsyg.2023.1059282.

[12] J. M. Harackiewicz and S. J. Priniski. "Improving Student Outcomes in Higher Education: The Science of Targeted Intervention." In: *Annual Review of Psychology* 69 (2018), pp. 409–435. DOI: 10.1146/annurev-psych-122216-011725.

[13] A.-S. Hoffait and M. Schyns. "Early Detection of University Students with Potential Difficulties." In: *Decision Support Systems* 101 (May 2017). DOI: 10.1016/j.dss.2017.05.003.

[14] A. Khaldi, R. Bouzidi, and F. Nader. "Gamification of e-learning in higher education: a systematic literature review." In: *Smart Learning Environments* 10 (Jan. 2023). DOI: 10.1186/s40561-023-00227-z.

[15] M. Koretsky, J. Keeler, and J. Ivanovitch. "The role of pedagogical tools in active learning: a case for sense-making." In: *International Journal of STEM Education* 5 (2018), p. 18. DOI: 10.1186/s40594-018-0116-5.

[16] A. Kumar. "Using problets for problem-solving exercises in introductory (...)" In: Oct. 2013, pp. 9–10. DOI: 10.1109/FIE.2013.6684774.

[17] S. Liénardy and B. Donnet. "GameCode: Choose your Own Problem Solving Path." English. In: 2020.

[18] S. Liénardy, L. Leduc, D. Verpoorten, and B. Donnet. "CAFE: Automatic Correction and Feedback of Programming Challenges for a CS1 Course." English. In: *ACM 22nd Australasian Computing Education Conference (ACE)*. 2020. DOI: 10.1145/3373165.3373176.

[19] S. Liénardy, L. Malcev, and B. Donnet. "Graphical Loop Invariant Programming in CS1." English. In: Namur, Belgium, 2019.

[20] R. Lobb and J. Harlow. "Coderunner: a Tool for Assessing Computer Programming Skills." In: *ACM Inroads* 7 (Feb. 2016), pp. 47–51. DOI: 10.1145/2810041.

[21] K. Maisha and S. N. Shetu. "Influencing factors of e-learning adoption amongst students in a developing country: the post-pandemic scenario in Bangladesh." In: *Future Business Journal* 9 (2023), p. 37. DOI: 10.1186/s43093-023-00214-3.

[22] M. D. Marmet. "Bridging the power gap: the impact of pedagogical strategies and relationship-building on student success." In: *Journal of Research in Innovative Teaching and Learning* (2023). Open Access. Article publication date: 24 May 2023. Issue publication date: 4 September 2023. ISSN: 2397-7604.

[23] S. S. Oyelere, F. J. Agbo, and I. T. Sanusi. "Developing a pedagogical evaluation framework for computational thinking supporting technologies and tools." In: *Frontiers in Education* 7 (2022). ISSN: 2504-284X. DOI: 10.3389/feduc.2022.957739.

[24] N. Parlante. "CodingBat: Code Practice." In: URL: https://codingbat.com.

[25] C. Van Petegem, R. Maertens, N. Strijbol, J. Van Renterghem, F. Van der Jeugt, B. De Wever, P. Dawyndt, and B. Mesuere. *Dodona: learn to code with a virtual co-teacher that supports active learning*. 2022. arXiv: 2210.10719 [cs.CY].

[26] M. de la Puente and H. Perez. "Assessing the Impact of Brilliant.org on Enhancing Mathematics Academic Performance among High School Students in Colombia: A Quasi-Experimental Study." In: *MATHEMATICS TEACHING RESEARCH JOURNAL* 15.2 (2023), pp. 82–103.

[27] K. Regmi and L. Jones. "A systematic review of the factors - enablers and barriers - affecting e-learning in health sciences education." In: *BMC Medical Education* 20 (2020), p. 91. DOI: 10.1186/s12909-020-02007-6.

[28] C. Romero, M.-I. López, J.-M. Luna, and S. Ventura. "Predicting students' final performance from participation in on-line discussion forums." In: *Computers and Education* 68 (2013), pp. 458–472. ISSN: 0360-1315. DOI: 10.1016/j.compedu.2013.06.009.

[29] P. Van Roy, G. Derval, B. Frantzen, A. Gégo, and P. Reinbold. "Automatic grading of programming exercises in a MOOC using the INGInious platform." In: 2015. URL: https://api.semanticscholar.org/CorpusID:64841921.

[30] N. Talib, N. Majid, and S. Sahran. "Identification of Student Behavioral Patterns in Higher Education Using K-Means Clustering and Support Vector Machine." In: *Applied Sciences* 13 (Mar. 2023), p. 3267. DOI: 10.3390/app13053267.

[31] Y. Teng and X. Wang. "The effect of two educational technology tools on student engagement in Chinese EFL courses." In: *International Journal of Educational Technology in Higher Education* 18 (2021), p. 27. DOI: 10.1186/s41239-021-00263-0.

[32] B. Weeraratne and B. Chin. "Can Khan Academy e-learning video tutorials improve mathematics achievement in Sri Lanka?" In: *International Journal of Education and Development using Information and Communication Technology (IJEDICT)* 14.3 (2018), pp. 93–112.

[33] B. Weeraratne and B. Chin. "Enhancing Usability of E-Learning Platform: A Case Study of Khan Academy." In: *Sir Syed Journal of Education and Social Research (SJESR)* 4.2 (2021).

[34] D. Zakrzewska. "Cluster Analysis in Personalized E-Learning Systems." In: *Intelligent Systems for Knowledge Management*. Ed. by N. T. Nguyen and E. Szczerbicki. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 229–250. ISBN: 978-3-642-04170-9. DOI: 10.1007/978-3-642-04170-9_10.

[35] N. Zeybek and E. Sayg. "Gamification in Education: Why, Where, When, and How?—A Systematic Review." In: *Games and Culture* 19.2 (2024), pp. 237–264. DOI: 10.1177/15554120231158625.