

GameCoursePersonal

Ana Rita Alves Gonçalves

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Daniel Jorge Viegas Gonçalves
Amir Hossein Nabizadeh Rafsanjani

Examination Committee

Chairperson: Prof. João António Madeiras Pereira
Supervisor: Prof. Daniel Jorge Viegas Gonçalves
Member of the Committee: Prof. António Fernando Coelho

November 2021

Acknowledgments

I would like to thank my parents for their encouragement and caring over all these years. Without you I wouldn't be where I'm at today. I would also like to thank my sister for all the support she gave me and for her positive energy. Thank you to my grandparents and the rest of my family as well for accompanying me through this wild journey. Pipa, Manel, aunt, I cannot thank you enough for all that you've done for me.

I would like to thank my boyfriend for his unconditional support, for always believing in me, even when I didn't myself. Words cannot describe how lucky I am to have someone like you in my life. I'm sure you'll do great things in life, and I hope I get to celebrate them with you.

I would like to thank Ana, my friend and partner on the GameCourse project for all her support. It's always a fun time working with you, but I'm even luckier for getting to have you as a friend. No doubt in my mind you'll go on to achieve great things.

Last, but not least, I would like to thank my advisors Daniel Gonçalves and Amir Nabizadeh for the immense support they gave me thought the entire project. It was an honor working with you. I've learned so much from you, and I can only hope you enjoyed working on this project as much as I did.

Thanks to everyone who in some way helped me in my journey through Instituto Superior Técnico—professors, friends, and colleagues. A big thank you to the professors of Usability and Information Systems for their collaboration.

This work was supported in part by the National Funds through the Fundação para a Ciência e a Tecnologia (FCT) under Project UIDB/50021/2020, and in part by the Project GameCourse, Portugal, under Grant PTDC/CCI-CIF/30754/2017.

To each and every one of you – Thank you.

Abstract

Gamification is the application of game design elements in non-gaming contexts. Previous experiences with Gamification in education have reported an increase in students' motivation and engagement when compared to non-gamified experiences. Still, even using such techniques, an underlying problem persists: education often operates under a "one-size-fits-all" model. GameCourse is the system used in the Multimedia Content Production course at Instituto Superior Técnico to make it a gamified course. However, there was a need to improve the previous version of the system, as well as to make GameCourse work correctly while being used for multiple courses. The objective of this thesis is to improve the system by making it more suitable for a larger array of students and courses, and by improving its performance and correctness. To better address the students' individual needs, we implemented a profiler that divides students into clusters, so that the system can adapt to each student cluster. This also included creating a new type of leaderboard, more suitable for a certain type of student. Our results from benchmark tests show that we improved the efficiency of the new version of the system, and the results from unit and integration tests show its correctness. To show that GameCourse can now be used for a larger variety of courses, we prepared for a semester of Usability and Information Systems using the system to make it a gamified course.

Keywords

Gamification; Education; Adaptive Gamification; Player Profiling

Resumo

Gamificação consiste na utilização de elementos de *design* de jogos em contextos que não jogos. Experiências anteriores com Gamificação na educação revelaram um aumento da motivação e participação dos alunos, quando comparado com experiências sem Gamificação. Contudo, mesmo usando esta inovação, continua a existir um problema: a educação funciona segundo um modelo unitário, não se adaptando aos diferentes tipos de aluno. O GameCourse é o sistema usado na cadeira de Produção de Conteúdo Multimédia no Instituto Superior Técnico para a tornar numa cadeira gamificada. Contudo, era preciso melhorar o sistema e fazer com que funcionasse corretamente ao ser usado para várias cadeiras em simultâneo. Este projeto teve como um dos objetivos melhorar o GameCourse para que possa ser usado por uma maior variedade de cadeiras. Trabalhámos também em assegurar o correto funcionamento do GameCourse e em melhorar a sua eficiência. Para melhor satisfazer as necessidades individuais dos alunos, implementámos uma funcionalidade que divide os alunos em grupos de acordo com as suas semelhanças para que o sistema consiga adaptar-se a cada um desses grupos. Adicionalmente, criámos uma nova versão do *leaderboard* que é mais adequada para um certo tipo de alunos. A eficiência e o correto funcionamento desta nova versão do GameCourse foram avaliados por testes de *benchmark*, e por testes unitários e de integração, respetivamente. Os resultados dos testes demonstram a melhoria em ambas as vertentes. Para demonstrar que o GameCourse é agora indicado para uma maior variedade de cadeiras, preparamos um semestre de Usabilidade e Sistemas de Informação para fazer dessa cadeira uma cadeira gamificada.

Palavras Chave

Gamificação; Educação; Gamificação Adaptativa; Perfis de Jogadores

Contents

1	Introduction	1
1.1	Objectives	4
1.2	Document Structure	5
2	Related Work	7
2.1	Gamification	9
2.2	Player Profiling	10
2.2.1	Learning Styles	11
2.2.2	Personality Traits	11
2.2.3	Player Types	12
2.3	Adaptive Gamification	14
2.4	Discussion	18
3	GameCourse	21
3.1	GameCourse as it was	23
3.2	Previous Research	26
4	Development	29
4.1	GameCourse in the Virtual Machine	31
4.2	Code Cleanup	31
4.3	Importing and Exporting Data	32
4.3.1	Users	33
4.3.2	Badges	33
4.3.3	Courses	34
4.4	External Data Sources	35
4.5	QR Module	38
4.5.1	QR Plugin Periodicity	38
4.5.2	Bugs and Security Issues	38
4.5.3	Configuration Page	40
4.6	Levels and Badge Levels	41

4.7	Performance Improvements	43
4.8	Course Data and Database Editor	44
4.9	Wildcards	46
5	From GameCourse to GameCoursePersonal	51
5.1	Profiler	53
5.2	Predictor	55
5.3	Clusters	56
5.4	Profiling Module	57
5.4.1	Configuration Page	57
5.4.2	Running the Profiler	59
5.4.3	Running the Predictor	60
5.5	Towards differentiated leaderboards	61
5.5.1	Leagues	62
5.5.2	Teams	62
5.5.3	No-disincentive Leaderboards	62
5.5.4	Micro Leaderboards	65
6	Evaluation	67
6.1	Usability and Information Systems (UIS) Case Study	69
6.2	Benchmark Testing	72
6.3	Automated Unit and Integration Testing	75
6.3.1	Former Testing Script	75
6.3.2	New Unit and Integration Test Suites	77
6.4	Discussion	77
7	Conclusion	81
7.1	Future Work	83
	Bibliography	85
A	Appendix A	93

List of Figures

2.1	Bartle's extended player types graph [1].	12
2.2	An example of linear model $R = B A$ [2].	16
2.3	Statements and illustrations used in Cloud Clicker [3].	18
3.1	GameCourse class diagram.	24
3.2	GameCourse modules.	24
3.3	View settings.	25
3.4	GameCourse's Plugin module architecture [4].	26
3.5	Final leaderboard position of different clusters (top position on the left) [5].	27
3.6	Radar chart comparison of participation measures by cluster [5].	27
4.1	Diagram with example of old file structure.	32
4.2	Diagram with example of new file structure.	33
4.3	Badge import modal.	34
4.4	Partial interface of the courses' list page.	34
4.5	Partial interface of the courses' list page.	35
4.6	Scheduled Cron Jobs.	36
4.7	Result of wrongful deletion.	36
4.8	Google Sheets configuration page before changes.	37
4.9	Google Sheets configuration page with new input box.	37
4.10	Form for redeeming a QR code.	39
4.11	Previous QR module page.	40
4.12	QR module's new configuration page.	41
4.13	Original database schema regarding badges and levels.	42
4.14	New database schema regarding badges and levels.	43
4.15	Previous version of the "My Course" page.	45
4.16	New information in "My Course" page.	45

4.17	Options to open the table data editor.	45
4.18	Table data viewer and editor.	46
4.19	Search by two individual columns.	46
4.20	Editing a table entry.	47
4.21	Example of skill dependant on wildcards.	47
4.22	Example of skill with multiple dependencies.	49
4.23	Skill tree.	50
5.1	Role hierarchy diagram.	57
5.2	Module Profiling's configuration page.	58
5.3	Interaction with Sankey diagram.	58
5.4	Partial interface of the Profiling module's configuration page.	59
5.5	Status label when the profiler is running.	60
5.6	Set of buttons when the profiler is running.	60
5.7	History table with results from the profiler.	60
5.8	Cluster options.	60
5.9	Predictor method selection.	61
5.10	Refresh button when the predictor is running.	61
5.11	Predictor results modal.	61
5.12	GameCourse's infinite leaderboard.	63
5.13	Prototype of the relative leaderboard from the perspective of user "Tiago Lopes".	64
5.14	New no-disincentive leaderboard.	64
6.1	Time spent importing levels.	72
6.2	Profiler execution time by number of students.	74
6.3	Profiler execution time by number of participations.	75
6.4	Profiler execution time by number of attributes.	75
6.5	Old test script test results.	76
6.6	Result of running every test in the "tests" folder.	77
A.1	Partial database schema in the beginning of the project.	94

List of Tables

2.1	Dimensions of Learning Styles. [6]	11
6.1	Comparison between Multimedia Content Production (MCP)'s and UIS's configuration.	69
6.2	Comparison between MCP's and UIS's required Moodle data.	70
6.3	Leaderboard load time on different versions of GameCourse.	73

Listings

4.1	Line in CronTab with scheduling for the Moodle plugin script to run.	35
4.2	Request to redeem a QR code.	39

Acronyms

AGQ-R	Achievement Goal Questionnaire-Revised
AI	Artificial Intelligence
API	Application Program Interface
CSV	Comma-Separated Values
DGD1	Demographic Game Design 1
DGD2	Demographic Game Design 2
EL	Expression Language
FFM	Five Factor Model
ILS	Index of Learning Styles
ISO	International Organization for Standardization
IST	Instituto Superior Técnico
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbors
MCP	Multimedia Content Production
OS	Operating System
QR	Quick Response
SIMS	Situational Motivation Scale
SMOTE	Synthetic Minority Oversampling Technique
TIPI	Ten Item Personality Measure

UAb	Universidade Aberta
UI	User Interface
UIS	Usability and Information Systems
URL	Uniform Resource Locator
XP	Experience Points

1

Introduction

Contents

1.1 Objectives	4
1.2 Document Structure	5

Education is one of the most important investments someone can make in their life. Throughout the years, the biggest struggle that teachers have been facing is students' lack of interest and motivation. Teaching techniques have been continuously evolving as a response.

Computers can support and facilitate these new ways of teaching. Besides allowing for creative alternatives to traditional teaching methods, they also facilitate the delivery of educational opportunities to students of widely differing backgrounds, as well as to students dispersed geographically [7]. However, computers have been used in schools for over three decades now and there are still improvements to be made. A promising concept that has been introduced and explored over the previous years is the concept of Gamification. Although Gamification can and has been applied without computers [8], they can form a powerful combination [9].

Gamification can be defined as “the integration of game-based mechanics, aesthetics and game thinking to engage people, motivate action, promote learning, and solve problems, in non-game contexts” [10]. With its growing popularity, gamification has been applied in several different contexts, such as healthcare [11] and fitness [12], as well as education [13]. Previous experiences with Gamification in the field of education have reported an increase in students' motivation and engagement when compared to non-gamified experiences. There are many cases of Gamification being successfully applied in learning environments. One of those cases is the Multimedia Content Production (MCP), which is a MSc course for the Information Systems and Computer Engineering field at Instituto Superior Técnico (IST). This course uses GameCourse, a framework that incorporates game elements such as badges and a leaderboard to keep students engaged and motivated to learn the course materials and reach higher levels in the game, which ultimately translates to their final grade.

GameCourse as it was at the beginning of this project could already operate without the need of running scripts manually, and many previously hard-coded components had been replaced by customizable ones. However, that version of the system still had problems that needed to be fixed, and it had yet to be used by actual students and professors during a semester of MCP, which meant that the system needed in-depth analysis and testing to find and solve any latent problems. Furthermore, even though one of the goals of GameCourse is to be suitable for a large variety of courses, the framework had never been used for a course other than MCP.

Even using novel educational techniques such as gamification, an underlying problem persists in systems such as the GameCourse: education often operates under a “one-size-fits-all” model where the learning process is the same for every student, disregarding their singular needs and preferences both as learners and, in the particular case of gamification, as players. This can be a source for the lack of students' motivation that teachers have been struggling with. GameCourse has already proven to increase students' motivation and participation [14, 15], but since students have different needs and preferences, they show different levels of engagement and overall enjoyment. This means that a new

type of Gamification, geared towards the adaptation to each student's profile, could be even more effective in increasing participation and overall motivation. To this effect, Adaptive Gamification started being studied. It is a form of gamification that includes processes for adapting the system based on a player model. With this adaptiveness, the learning process could potentially be more inclusive and effective.

1.1 Objectives

This thesis aims to **make overall system improvements, and to implement supporting functionality so that the existing GameCourse system can seamlessly provide students with different profiles an adapted gamified learning environment.**

GameCourse was already a standalone system that allowed professors to create custom game elements such as badges, skills, levels, and rules. Professors were also able to create the pages that students would see when they entered the system. When it came to multiple courses co-existing in a single instance of GameCourse, the system lacked proper separation of the courses' content, especially when they shared students. Furthermore, some elements weren't as customizable as they appeared to be, which was revealed when we prepared for a semester of Usability and Information Systems (UIS) with the intent to use GameCourse. The system was too tailored towards MCP. We needed to make the system more suitable for a larger variety of courses and make them work correctly in the presence of other courses. To this effect, we had to fix all separation problems that we discovered, and we implemented new features that make the system more flexible and, thus, more suitable for a larger array of courses.

The latest version had yet to be used by actual students and professors, which was the ultimate test GameCourse could be subjected to. Doing so exposed problems that had gone unnoticed thus far. Besides correctness issues, there were performance issues that became noticeable when going to pages such as the leaderboard and users' profile pages. We fixed all bugs that we found either while interacting with the system or by running the test script, as well as the bugs that were reported by students and professors from MCP. We also addressed performance issues by refactoring some functions to make them more efficient, and by replacing some of the values calculated on demand for the views by previously calculated values that are then kept in the database to be available when needed. Another important component of our work was to create features that would help the professors. We implemented a way for professors to see and edit the awards and participations inside their course. Not only can this help professors to implement rules, since they get a better understanding of how participations are represented in the system, but it also provides an easy way to give, edit, and remove awards and participations manually, should it be needed, without requiring professors to access the database.

The big novelty that came from our work was the functionality to profile students. Once we knew we had a strong base to build upon, our big goal was to make GameCourse into an adaptive gamification environment. However, in order to adapt, the system first had to determine each student's profile to know what to adapt to. Because it would generate a large overhead to adapt the system to each individual student profile, we divide students into clusters which represent groups of similar students. To determine the clusters, we use a machine learning algorithm that we implemented into a script. Professors can easily get these results through the interface, without the need of manually running said script. This was important since the system is supposed to be easy to use for professors with different levels of computer knowledge.

Once we could successfully profile students, we started thinking about what to show to students from each cluster. In this work, we were able to take the first step towards adapting the system, with the implementation of a new leaderboard that is more suitable for some clusters. Professors can now choose which type of leaderboard to show to each student according to their cluster. Now that the base for achieving an adaptive learning environment has been built, more analysis can be made into the preferences of each cluster to create pages and elements more suitable for them.

1.2 Document Structure

Following the Introduction, we present Chapter 2 which contains a review of the state of the art in the fields of gamification and its adaption to better fit different users. In the same chapter, we overview player models that have been developed over the years, as well as their advantages and disadvantages.

In Chapter 3, we briefly describe the evolution of the GameCourse system over the years. We then describe the GameCourse system as it was at the beginning of this work, and we finish by exploring previous studies made on the GameCourse system regarding its users.

In Chapter 4, we present the problems that were uncovered in GameCourse's implementation, and the solutions we found to each one of them. We also explain the work that was done to improve the existing functionality.

Chapter 5 is dedicated to describing the new adaptive capabilities of GameCourse including the process of developing it and the solutions we found to overcome the hurdles that we encountered. We close that chapter by presenting the research done into different types of leaderboards, and the implementation of a new leaderboard more suitable for certain types of users.

Then, in Chapter 6, we evaluate our work based on the results of benchmark, unit, and integration testing. We also conduct an experiment that consisted of using GameCourse to provide a gamified experience in the UIS course.

Lastly, in Chapter 7, we present our conclusions and final observations about this work, and talk

about what the next steps are in terms of future work, now that GameCourse can adapt to its users.

2

Related Work

Contents

2.1 Gamification	9
2.2 Player Profiling	10
2.3 Adaptive Gamification	14
2.4 Discussion	18

This section presents a review of what has been done so far in the fields of gamification and its adaptation to better fit different users. Given the context of this study, this section also focuses on player types, including player models that have been developed over time and their advantages and disadvantages.

2.1 Gamification

Gamification can be defined as “the integration of game-based mechanics, aesthetics and game thinking to engage people, motivate action, promote learning, and solve problems, in non-game contexts” [10]. Previous research has showed positive results [16]. Gamification is particularly useful in producing desired behavior change through the formation of habits by reinforcing the reward and emotional response of the users, thus requiring fewer cognitive resources each time the desired activity is reproduced [17].

With its growing popularity, gamification has been applied in several different contexts, such as healthcare [11] and fitness [12]. Another context where gamification has been successfully applied is education [13]. According to Furdu et al., applying gamification in learning environments provides a better learning experience, by combining having fun with learning. It is also a way of providing instant feedback since gamification provides metrics that facilitate knowing how a participant is progressing. Furthermore, it makes for a better learning environment because students can evolve at their own pace, as well as choose the assignments at their own will to obtain enough points to pass the course [10].

In many cases where the application of gamification fails it is because the processes have been inappropriately gamified. This may be due to the lack of understanding of what gamification is, how it works and, more specifically, how to design gamification experiences that inspire player behavior changes and result in desirable outcomes [17]. To better guide the gamification design process, Robson et al. introduced the MDE framework, which “outlines the interdependent relationship of the gamification principles of mechanics, dynamics, and emotions”, and illustrates how these principles can be applied together to create the player experience. Mechanics are the foundational aspects of gamified experience. Dynamics are the types of player behavior that emerge as players partake in the experience. Gamification dynamics are difficult to predict and thus can lead to unwanted behaviors and outcomes. Emotions are the mental affective states and reactions triggered by the gamified experience. From the player standpoint, emotions are key. Another contribution to guide the design of gamified experiences came from Mora et al. [13] with the creation of the SPARC framework. The learning experience is structured according to three viewpoints: the rules (describe the basics of the activity mechanics), the metaphor (contextualizes the rules in the learning context, giving sense to the whole activity so it may not be considered arbitrary by the students), and the tool (used to implement the rules and the metaphor). The name of the framework stands for the dimensions that must be considered during the design process: Sense, Purpose, Autonomy, Relatedness, and Competency.

There are some practical examples of gamification being applied to learning environments. Some approaches are simple, introducing a single element to assess how students' behavior and motivation was affected by it [10], while others included a combination of elements such as points, rewards, levels, and leaderboards [16]. However, it is difficult to draw conclusions from studies that involve the combination of game design elements, as it makes it hard to assess how individual game elements are linked to behavioral and motivational outcomes [18].

One thing that all the approaches mentioned so far have in common is that the elements (or element) included are the same for every user, without taking into consideration individual needs. Gamification can only be successful if it can keep users motivated and engaged, otherwise they will move on to something more exciting (in terms of emotions) or more engaging (in terms of overall experience) [17]. Moreover, motivators like points, badges, leaderboards are not effective for students who are not competitive, and if these elements play a central role, the students will eventually lose their interest or get discouraged. This can be an explanation as to why previous work shows that the positive effects of gamification seem to lessen over time. Across different studies, the elements that seem to prompt the most divergent reactions are those that promote competition between students, like leaderboards, with some students seeing them as a source of motivation, and others being discouraged by them. To tackle the issue regarding leaderboards, for example, Marczewski has suggested the use of both absolute leaderboards (showing progress against all other users) and relative leaderboards (showing performance within a domain and / or a developmental stage) [19]. This way it could be possible to please both groups of students without having to remove the element entirely.

What we can conclude is that, although there are many successful cases of gamification being applied, there is still a need to better understand the users, what their individual expectations and needs are, and how to cater to those needs. This could be the solution for keeping users engaged and motivated. Instead of creating a static environment that must please everyone, it would be much more interesting if the environment could satisfy each user individually. In other words, there is a need to make these gamified systems adapt to their users.

2.2 Player Profiling

In order to implement a system that is capable of adapting to its users in a meaningful and effective manner, we must first know how to categorize them into "player types". These player types work as a synthesis of the users' exhibited behavior towards games. Ferro et al. describe player types as being the same synthetization as personality types, with the difference being the context in which they are applied [20].

While some authors believe that personality traits - which influence how someone experiences and

perceives the world - are more relevant than player types [20], others have also studied the impact of learning styles - how someone receives, interacts with, and integrates educational materials - on academic achievements and overall experience of gamification, in comparison to the impact of personality traits [21].

All previously mentioned authors, however, base their studies on previously constructed frameworks that help guide the categorization process and ultimately make it possible to compare results and draw conclusions.

2.2.1 Learning Styles

Regarding learning styles, Felder’s Index of Learning Styles (ILS) categorizes students according to their preferences on four dimensions: Sensing/Intuitive (what type of information does the student perceive more easily), Visual/Verbal (which sensory channel is the most effective to perceive external information), Active/Reflective (how does the student prefer to process information), and Sequential/Global (what is the student’s progress like towards understanding the materials) [6] (Table 2.1). This model was first introduced in 1988 [22], but it was later updated, with the suppression of a previously existing fifth dimension [6]. It has been validated and widely cited over the years.

Dimension	Learning Style
Perception	Sensing Intuitive
Input	Visual Verbal
Processing	Active Reflective
Understanding	Sequential Global

Table 2.1: Dimensions of Learning Styles. [6]

2.2.2 Personality Traits

When it comes to personality traits, the most used model is the Five Factor Model (FFM), also known as the OCEAN model or the Big Five model. According to McCrae et al., this model can be described as “a hierarchical organization of personality traits” [23]. This model identifies five basic dimensions (traits): Openness to Experience, Conscientiousness, Extraversion, Agreeableness, and Neuroticism. Authors that have previously used this model in their research claim that using a stable measure such as personality to explain behavior has advantages over using models based on perception [24]. As complete as this model is, one of the problems that researches often come across is having a limited time to survey participants. For this reason, briefer measures of the FFM have been developed. Not only

do they provide faster results, but they can also avoid any disruptions that may come from participants getting fatigued or distracted [21]. One of those measures is the Ten Item Personality Measure (TIPI), a 10-item measure of the FFM [25]. Although TIPI has been validated, it is a very short measure and thus has some limitations. One very important limitation to take into account is that, compared with standard multi-item measures such as FFM, TIPI is less reliable [25].

2.2.3 Player Types

Another category of models is those that focus on player types. The most well known player typology model is Bartle's player type classification [26]. The first version of this model divided players into four categories (Killers, Achievers, Socializers, Explorers), across two axes: whether the player prefers to act or interact, and whether the player prefers to interact with other players or with the world itself [26]. Reasons to choose this model include its simplicity and easy testability [27]. However, this original model had some flaws. First, Bartle observed that players often switch between player types. The concept of player types is very much built upon personality traits, which are dynamic and change frequently depending on context and environment [20]. The issue is that the first model does not elaborate on how and why this happens. The second problem is that it only allows for the attribution of a single player type to each player. This means that the model does not predict sub-types, resulting in a relatively low diversity of player types. To overcome these limitations, Bartle [28] introduced a new axis, as can be seen in Figure 2.1, that relates to the nature of actions: implicit actions (done automatically without the intervention of the conscious mind) as opposed to explicit actions (planned actions, often with a goal in mind) [1]. Despite these improvements, Bartle's model is bound to a specific game genre and may not work in other contexts such as gamification [29].

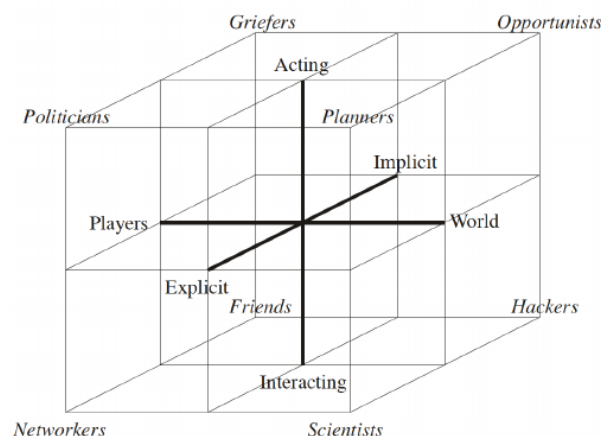


Figure 2.1: Bartle's extended player types graph [1].

Another player model is the Demographic Game Design 1 (DGD1) which allows for players to be

classified with combined playing styles. Then came one of the most recent contributions in this area: BrainHex. This model was based on insights from neurobiological findings as well as the results from earlier demographic game design models (DGD1 and Demographic Game Design 2 (DGD2)) [30] and besides not being related to a specific game genre, it allows players to have both a main type and sub-type, and it includes a list of oppositions to each player type. Additionally, there is an online questionnaire for this model. The model presents seven different archetypes of players:

- **Achiever:** goal-oriented and motivated by long-term achievements.
- **Conqueror:** enjoy defeating impossibly difficult foes, struggling until they achieve victory, and beating other players.
- **Daredevil:** focused on thrill seeking, excitement and risk taking.
- **Mastermind:** enjoy solving puzzles and devising strategies, as well as focusing on making the most efficient decisions.
- **Seeker:** curious about the game world and enjoys moments of wonder and richly interpretable patterns.
- **Socializer:** people are their primary source of enjoyment.
- **Survivor:** enjoy the intensity associated with terror experiences.

To investigate the correlation between player types, and personality types and traits, Ferro et al. [20] created a table with five columns: player types (from other models such as Bartle's), personality traits and types, game elements, game mechanics, and classification. By creating these clusters of similar player types, they ended up with five categories of players: **Dominant** users exhibit a strong need to be visible, whether through sociability, assertiveness, aggressiveness and so forth; **Objectivists** seek to build upon their knowledge by demonstrating their skills and intelligence; **Humanists** like to involve themselves in tasks that rely on social engagement; **Inquisitive** users like to explore and investigate new things; **Creative** individuals like to create and develop things through utilizing skills that they obtain through experimentation.

Not convinced that video game focused player types were a good fit for gamification, Marczewski [31] developed the User Type Hexad. The idea that "people cannot be broken down into simple individual categories" [31] led him to create a model where the three user types - Intrinsic User, Player User, and Disruptor User - can each be divided into four sub-types. This division is similar to that done in the original Bartle's model [26], with an axis dividing users that prefer to act from those that prefer to interact (or if they have good or bad intentions in the case of Disruptors), and another axis that divides users that would rather interact with other users, from those that would rather interact with the world.

By knowing how to identify different player profiles, what is left for us to explore is how can that information be used to adapt the system to each of the different players.

2.3 Adaptive Gamification

Personalized gamification can be described as the customization of the game elements, the interaction mechanics, the tasks, or the game rules for each user, according to their preferences [32]. Škuta et al. state that “any learning environment can be considered adaptive if it is able to monitor and interpret all user activities and derive user preferences based on interpreted activities” [27].

Some work developed so far offer recommendations by identifying game elements that would better suit different students. Knutas et al. [33] explored an evidence-based method for deciding which game elements are the most appropriate for each Bartle player type and how to apply them. Students from a software engineering course were interviewed and their actions were recorded during a project. The list of analyzed interactions was studied, resulting in a frequency distribution of interactions by type and by interaction target for each person. Also using Bartle's player types, Škuta et al. [27] studied players' preferences but in this case they connected them with the most appropriate game principles, which are higher level categories of game elements (for example, the game principle Visible Status can be implemented using the elements: levels, badges, and leaderboards). This culminated in a decision matrix that could be a basis for the decision-making processes of an adaptive system. Barata et al. [34] conducted another study on adaptive gamification applied to a college course. This study analyzed how students performed over the term and used clustering algorithms to identify what different types of strategies were adopted by students. The clustering was made based on data regarding students' performance as well as their self-reported gaming habits and preferences, including the result they obtained in the BrainHex questionnaire.

All the studies mentioned so far rely on profiling by player types. A study that, on the other hand, focused on personality traits using FFM also provides insight on what different users may prefer. Codish et al. [24] applied gamification to an academic course using a pen and paper information system. This system included feedback mechanics - point, rewards, and badges -, and presentation mechanics - leaderboard and progress bars. The preferences for these mechanics were measured using a questionnaire and perceived playfulness was measured. This is one of the limitations of this study since responses to self-reported items may be bias, which means that the research evaluated the theoretical behavior and not the actual behavior.

While the classification of Ferro et al. [20] was still a work in progress, Monterrat et al. [35] released a series of recommendations for developing adaptive gamification systems to be plugged into learning environments. These systems are both generic (since they use game elements as epiphytic functional-

ities, which does not affect the host environment when integrated with the user interface) and adaptive (because they use a player model that defines the player type matching best with the user). In this case, player profiling was made using on Ferro's classifications [20], for the convenience of providing a direct link between player types and game design elements. The values for each student's motivational factors are initialized based on user data (age, gender). The values will then be updated based on usage data (i.e. interactions with the system) using human-written rules. The adaptation should be done in four steps:

1. Tracing data from the learning environment and the game elements.
2. Evaluating the engagement level of the user.
3. Updating the player model, based on adaptation rules, using basic data about the user, data from the use of the environment, and data describing the learning context.
4. Integrating within the user interface the epiphyte matching best with the player model.

This study provides an insight on what an adaptive engine could look like, but since it was never implemented, not much can be concluded.

There are also some examples of adaptive gamification systems being implemented and tested. According to Todello et al. [32], personalization can be implemented in two ways: as a customization (also referred to as user-initiated personalization), where the user selects the elements that they wish to use, or as an automatic adaptation (also referred to as system-initiated personalization), where the system takes the initiative to select the elements for each user - with or without some user input in the process.

Because most studies thus far relied on self-reported data, which we have already established is not optimal, Tondello et al. [32] conducted their own study in which they compared a generic gamified system to a customizable gamified system without resorting to self-reported data sources such as questionnaires. Students were divided into two groups: participants in the control group were presented the list of game design elements for information only and all elements were automatically enabled for them. Player profiling was made using Hexad user types and each of the player profiles was updated as students interacted with the system. Participants in the experimental group were asked to select as many game elements they wanted from the eight available options. The goal was to allow the users to improve their experience by removing the elements they did not want from the interface.

Hexad user styles have been used in other studies as well. Mora et al. [13] first surveyed participants to assign them to the group that would better fit them, according to their Hexad user type. The engine included an assignment function that would link each student to the most fitting group: alpha, beta, delta, or gamma. They explored the route of taking into consideration more than one player type

for each player. This means that the adaptation can be made either based on the highest scored user type or based on a pair of user types if the user got a high score for more than one Hexad user type. Each of the four groups had a different Trello dashboard to sign in to, which means that students belonging to the same group would have the same view of the system. The gamified platform created by Roosta et al. [36] consisted in implementing game design elements and embedding them into an existing learning environment called DoosMooc. What differentiates this approach from the previous two is that player profiling was based on users' motivation styles. To start, students were asked to answer Elliot Achievement Goal Questionnaire-Revised (AGQ-R). The answers were analyzed, and the motivation styles of learners were identified. Then, students were randomly distributed in two groups (control and experiment). In the experiment group, gamification elements were assigned to the learners based on the categorization (i.e. learners' motivation styles) while the control group received random gamification elements. The elements used were the most widely used in the gamified environments: progress, leaderboard, feedback, and badge.

An alternative approach is to calculate preferences using the product of two matrices. Monterrat et al. [2] and Lavoué et al. [29] both tested this approach in a very similar way. To determine a student's preference for each feature (R), they calculate the product between the preference of each BrainHex player type for each feature (i.e. game design element) (A) and the users' score for each BrainHex player type (B). To better illustrate this method, in Figure 2.2, u1 to u4 each represent a user, f1 to f3 each represent a feature and C and S could stand for Conqueror and Socializer respectively, since the player model is BrainHex, but could also represent personality traits like competitor and social if the profiling was based on personality traits, for example. This means that this procedure could be used with different profiling methods.

$$\begin{array}{ccc|c|cc}
 & \underline{f1} & \underline{f2} & \underline{f3} & & \underline{C} & \underline{S} & & \underline{f1} & \underline{f2} & \underline{f3} \\
 \mathbf{u1} & 10 & 00 & 05 & = & \mathbf{u1} & 10 & 00 & \mathbf{C} & 1 & 0 & \frac{1}{2} \\
 \mathbf{u2} & 00 & 06 & 12 & & \mathbf{u2} & 00 & 12 & \mathbf{S} & 0 & \frac{1}{2} & 1 \\
 \mathbf{u3} & 06 & 03 & 09 & & \mathbf{u3} & 06 & 06 & & & & \\
 \mathbf{u4} & -08 & 03 & 02 & & \mathbf{u4} & -08 & 06 & & & &
 \end{array}
 \times$$

Figure 2.2: An example of linear model $R = B A$ [2].

Users' score for each BrainHex player type is easily determined through a questionnaire, but two approaches have been tested by Monterrat et al. [2] to determine the preferences of each BrainHex player type for each feature. One way of determining said values is through human expertise. Another way is by empirically deriving those values, which is more objective and potentially more accurate but requires sufficient data. The first approach, although straightforward, is prone to bias. Although the version of the method that relies on empirical data could not be validated in this experiment because this type of model requires a large amount of users, the experiment using the experts' approach showed

that a group of students using a system that provided adapted features spent 39% more time on the Learning Environment than a group of students using a system that provided counter-adapted features. Continuing the study of this approach, Lavoué et al. [29] integrated five gaming features in an existing web-based learning environment named Project Voltair in such a way that they can be toggled on and off. The features were: stars, leaderboard (which only shows the user's neighbors rather than the best ones), tips (that learners can give to each other), a hiker (showing progress), and a timer. To initialize players' profiles, users filled the BrainHex questionnaire. Learners' motivation to use the learning environment was evaluated using the Situational Motivation Scale (SIMS). The system also collected data on learning activities and outcomes, as well as the total time spent on the system. Once again, students were divided into two groups. For members of the group with adapted conditions, the two gaming features with the highest score in R were activated. For the group with counter adapted conditions, the two gaming features with the lowest score in R were activated.

Machine learning algorithms can also be useful for classifying students based on their level of participation. Mbabu et al. [37] developed a platform that should use Artificial Intelligence (AI) techniques to adapt to students' learning behavior. Due to the small dataset in this study, K-means was used for clustering students and K-Nearest Neighbors (KNN) for classifying students adaptively based on how student engage in Moodle platform. The adaptive tool is linked to the Moodle backend where individual user data is retrieved from the logs and then evaluated. This evaluation is passed to the classification algorithm which classifies the user to one of the identified clusters, and its value is recorded, which, along with student evaluation, is used to provide the appropriate gamification elements. The system initially does not know which cluster each student belongs to. As the user interacts with the system, the adaptive engine evaluates traits and behavior and passes this data to the classifier. A pre-study was done from logs of an e-learning platform to be the training data for the machine learning algorithms used and four main clusters were identified. Each main cluster was given their own set of game elements that would better fit them.

Preparing the experience is very important as well since it is the first contact that users will have with the system. Most of the experiments mentioned so far start with students filling out questionnaires to start building their profile, either based on personality traits, player types, user types (in the case of Hexad model, for example), among other factors. On the subject of Hexad questionnaires, in practice they can be less suitable for gamification applications since filling out a questionnaire can potentially affect users' experience and immersion negatively [3]. Altmeyer et al. [3] transferred the 7-point Linkert scales in Hexad questionnaires into binary choice questions in the application. The result was Cloud Clicker, that presents users two statements in each of the fifteen questions. As illustrated in Figure 2.3, each statement corresponds to an Hexad user type and users must decide which one suits them better. Although this is not a form of adaptive gamification, it can certainly be used to set up a personalized

experience of that type.



Figure 2.3: Statements and illustrations used in Cloud Clicker [3].

2.4 Discussion

The literature shows that, although Gamification has shown to be effective in education, there is still room for improvement. Combining the strengths of Gamification with the advantage of tailoring the experience to each user can be the key to achieve longer-lasting user motivation and engagement. Not only that but also to appeal to a larger spectrum of users at the same time. The mentioned contributions have been discovering the path towards a more effective form of gamification. Although the overall results have been positive regarding the adaptation of gamified experiences to better suit individual users, some details still need to be reconsidered and explored. The first issue that should be addressed is the evaluation of the solutions. Small sample sizes as well as limited experiment time make it hard to draw any major conclusions from studies [33, 34, 36]. Moreover, although using questionnaires or other sources of self-reported data to capture students' opinion and reaction to environments is convenient, self-reported items are always problematic for analysis since there are many biases in the responses [24].

Regarding the actual adaptation engine, it is important to decide what type of data is required. One of the major details pointed out by Tondello et al. [32] and Roosta et al. [36] is that the adaptation should not be based solely on dominant player types. Tondello et al. [32] found that the students' user type scores were not related to their preferred, most influential, or most motivational game elements. This may point to other factors being needed for a more precise determination of each user's preferences. Lavoué et al. [29] concluded that adaptation of gaming features only increases participation for the most active users. If the system does not captivate the users from the start, motivation may drop and users may become less active. At this point, adaptation mechanisms may not be enough to increase

the participation of these less active users. This implies that other mechanisms or more complex user profiles would have to be determined to captivate users from the start of use onward.

Another interesting possibility is to allow students to choose the elements they like the most, like Roosta et al. [36] suggest. Monerrat et al. [35] also highlights the need to compare systems that provide automatic adaptation to systems where the user can manually choose the new functionalities. Giving the users the freedom to add or remove elements is particularly useful when the system does not have any information about a student when they first start using it, as this provides important feedback and an insight on users' preferences. This is known as a cold-start problem, when no prior events are known for certain users [38]. Nevertheless, it appears that learners may not be aware of the gaming features that motivate them the most to use the environment. So, it indicates that adaptation should not be based on explicit learners' choices, but rather on indirect measurements through, for example, interaction traces [29].

3

GameCourse

Contents

3.1 GameCourse as it was	23
3.2 Previous Research	26

GameCourse is a framework that incorporates game elements such as badges and levels to keep students engaged and motivated to learn the course materials and reach higher levels in the game, which ultimately translates to their final grade. In this section, we start by describing the GameCourse system as it was at the beginning of this work, and we then explore previous studies made on the GameCourse system regarding its users.

3.1 GameCourse as it was

Over the years, GameCourse has been the subject of several theses. In 2016, André Baltazar [39] developed Smartboards, an improved system mainly visually but also functionally, yet still reliant on manually run scripts and hardcoded values. Then, Alice Dourado [40] made some major improvements to the system that led to a more modularized architecture and allowed users to create and edit pages and templates using a newly created expression language. In 2020, Patricia Silva [41] and Diana Lopes [4], were responsible for upgrading SmartBoards into its successor – **GameCourse** -, through extensive work done on improving both the frontend and the backend of the system. GameCourse would then become a truly standalone system with the addition of a rule system and rule editor developed by Inês Paiva [42] and Ana Nogueira [43].

Figure 3.1 shows a simplified class diagram of the GameCourse system. GameCourseUsers can authenticate either through Fénix, Google, Facebook or LinkedIn. A GameCourseUser with Admin privileges is the only user who can create courses, activate and deactivate users, and change Admin permissions. By default, the first account created in GameCourse is always an Admin and every account created afterwards starts without Admin permissions. A CourseUser represents a GameCourseUser inside a specific Course. CourseUsers can have different Roles in a specific Course. GameCourseUsers with Admin permissions can create new roles, assign them to CourseUsers, and choose which permissions each role has. The default roles are “Student” and “Teacher”. Teachers can change Course details, export GameCourse elements, configure Course elements, add and import Students, change Roles, and update Course Data.

Once a course has been created, GameCourse provides a set of modules that can be disabled or enabled at any time for that course in the Course Settings page shown in Figure 3.2. These modules are the core of the gamified experience. Each module brings with it new entities and concepts relevant for the gamified experience like the already-mentioned badges, skills, and Experience Points (XP), among others that will be discussed throughout this chapter. All this is reflected in module-specific functions and variables that know how to handle those entities. All modules, when enabled, register those functions and variables in a global Dictionary. The resulting vocabulary, referred to as the **Expression Language (EL)**, is used in system-wide autocomplete suggestions for expressions when designing views

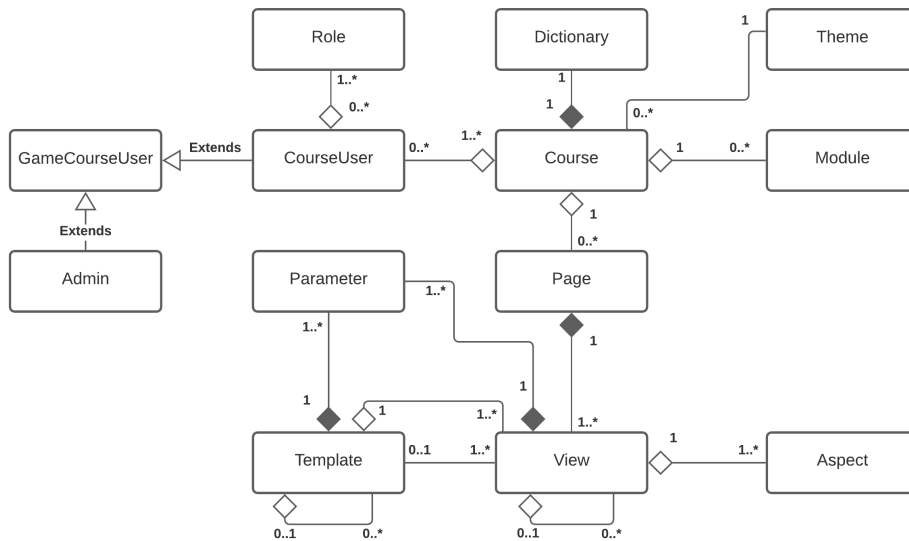


Figure 3.1: GameCourse class diagram.

and writing rules. Some modules have dependencies, which means that, in order to enable them, you first need to enable the module(s) on which they depend on. In the current version of the GameCourse, all modules depend on the “Views” module.

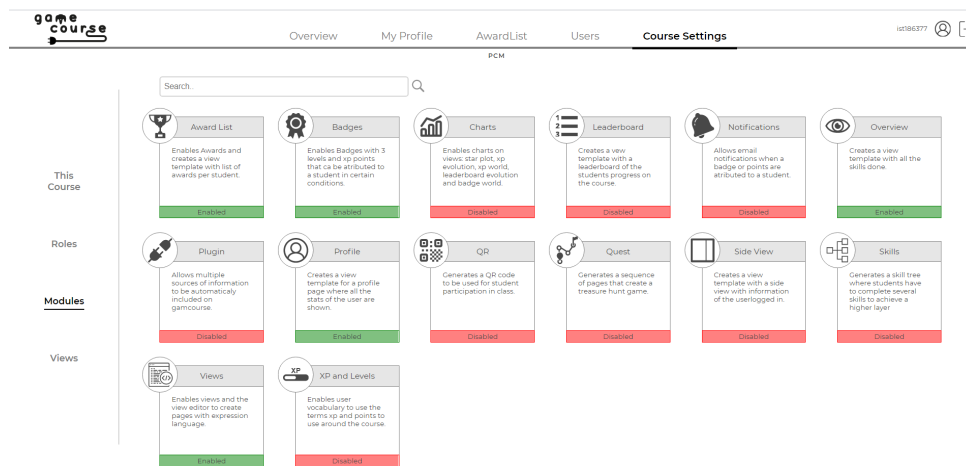


Figure 3.2: GameCourse modules.

By enabling the Views module, Teachers can create custom views. These views can then be used to create pages accessible by other CourseUsers. A View is composed by other views, which may be called view parts. There are different types of view parts, with the most basic being images or text elements that can be inside tables or blocks. By nesting views, we can create more complex views (e.g. Leaderboard).

Currently, the views and game elements that are accessible by each student are the same, although

it is already possible to present distinct views to different users. This is achieved through Aspects. There are two types of view aspects:

- **Role-Single:** view that can show a different aspect according to the viewer’s role.
- **Role-Interaction:** view that can show a different aspect according to the roles of the viewer and of the user associated with the page. As an example, consider two students, Jon and Jane. What Jon sees when he goes to Jane’s profile page may be different from what a professor sees in Jane’s profile page.

As shown in Figure 3.3, each view defines a pair (user role, viewer role). “User role” refers to the user role that a page belongs to. “Viewer role” refers to the role of the page viewer. The default value for this pair is (Default, Default), except for the views of pages that belong to a specific “user role”, like the Profile pages. Professors can edit a view to add and delete (user role, viewer role) pairs. When a user loads a page, the system shows the most appropriate aspect of a view based on the pair values.

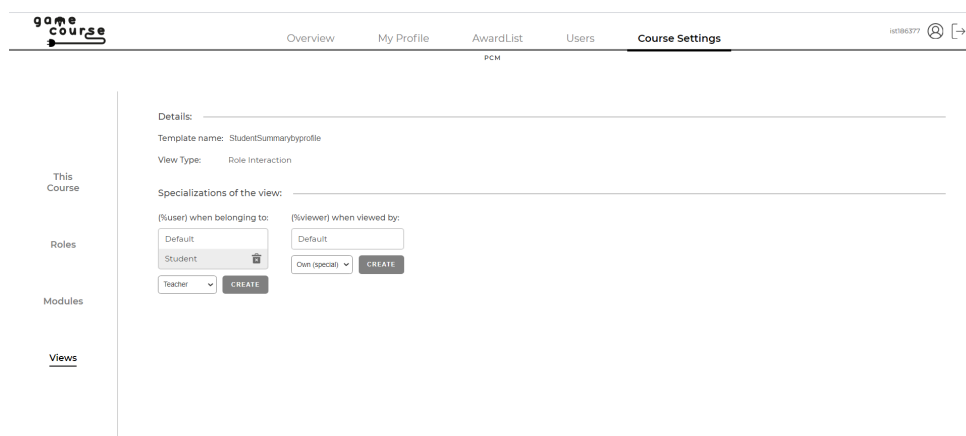


Figure 3.3: View settings.

Students have many ways of working towards their grade and this is represented by Participations, which can then lead to students being awarded XP and Badges. All actions that count towards earning XP take place outside of GameCourse. When a student participates in class, professors can give the student a Quick Response (QR) code that represents an event and will possibly count towards a Badge. If, for example, students wish to participate on the Skill Tree, they can make their submissions on Moodle. This led to the implementation of the Plugin module (Figure 3.4) that works as an abstraction layer to GameCourse so that every external source of information can be treated in a similar way [4]. There is a specialized script for each data source that knows how to access and parse the external data to then be inserted into the GameCourse database as a Participation. These scripts run in the background and are scheduled through Cron Jobs. Cron is a job-scheduling tool for Linux/UNIX environments that allows you to schedule scripts or commands to run at a specified time and repeat at a designated frequency [44].

We can use the CronTab command to list the Cron Jobs that are scheduled to be executed by the Operating System (OS) at a specified time.

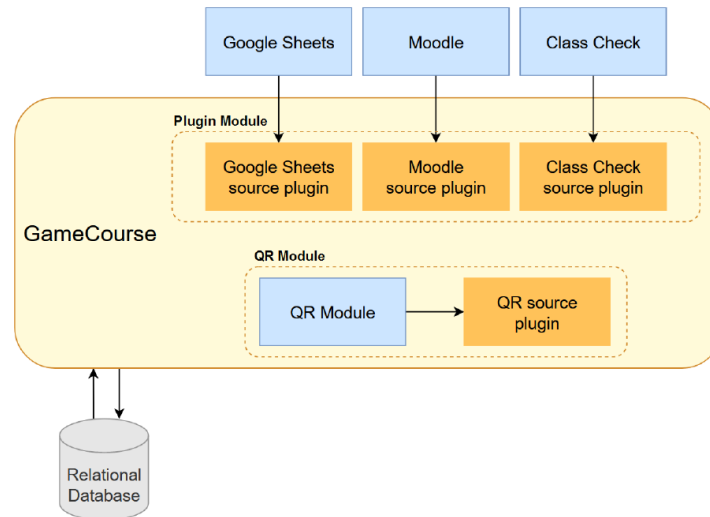


Figure 3.4: GameCourse's Plugin module architecture [4].

After getting students' Participations into the system, GameCourse then transforms these Participations into Awards. Like any regular game, GameCourse needs rules that dictate what conditions must be met in order for the student to get a certain reward. The component responsible for parsing and firing these rules inside GameCourse is **GameRules**. Every time data comes in from one of the data sources and, consequently, participations are created or updated, a call is made to the GameRules rule system so that awards can be given (or taken away) according to the current data. Using the Skills module as an example, the grade that a student got for participating in a Skill (from one to five) is only a Participation until GameRules runs its rules against this Participation, checks if the grade is at least three out of five, and, **when** such condition is true, **then** XP is awarded for completing the Skill.

Appendix A represents the database schema of the GameCourse system when we first started this project. It does not show all tables that can exist in the database, as some of the modules may create additional tables when enabled. Still, some tables that should only be created by enabling modules (i.e., level) were being created in the beginning along with the base tables needed to use the system. These flaws will be addressed in Chapter 4.

3.2 Previous Research

Besides the work made on building and improving the GameCourse system, there has also been an interest in studying the users of the system, more specifically the students. With GameCourse being used in multiple iterations of the MCP course, it was possible to study student behavior and evolution,

and compare the results of each year. Barata et al. [5] collected student data from three consecutive terms of MCP. Between each iteration, improvements were made to address students' needs. Cluster analysis was performed based on student behavior and performance measures which included, for example, online participation, lecture attendance, and evaluation results.

A total of six different clusters were identified, but only four of them were observed in the third iteration of the course. These four clusters were: Achievers, Regular students, Halfhearted students, and Underachievers. Performance-wise, both Achievers and Regular students performed the best, with Halfhearted students performing below average and Underachievers having the worst performance. Participation-wise, the ranking is the same but there is a bigger gap between Achievers and Regular students, with Achievers participating significantly more. As a result of these differences, Achievers and Regular students tend to occupy the top positions of the leaderboard, followed by Halfhearted students and, lastly, the Underachievers (Figure 3.6). The clusters also differ in which activities they chose to partake in the most (Figure 3.5).



Figure 3.5: Final leaderboard position of different clusters (top position on the left) [5].



Figure 3.6: Radar chart comparison of participation measures by cluster [5].

The next step, according to the authors, would be to use this information along with machine learning

techniques to create an adaptive gamified learning environment, capable of detecting a student's profile and adapting accordingly [5].

The latest effort in understanding how different students behave in and interact with gamified environments, with analysis made in the context of MCP while using GameCourse, comes from Nabizadeh et al. [45]. Over a ten-year period, student data was collected and analyzed. Taking into account student performance, they used the elbow technique [46] to identify the optimal number of clusters and determined that number to be **four**, consistent with previous results. To determine the cluster that each student belonged to, the amount of accumulated XP per day was used as attributes for the cluster analysis later performed using the K-means algorithm [47]. More detail about this methodology will be provided in Section 5.1. It is important to note that one of the conclusions they came to was that there wasn't any correlation between students' clusters and their BrainHex player types [30].

4

Development

Contents

4.1	GameCourse in the Virtual Machine	31
4.2	Code Cleanup	31
4.3	Importing and Exporting Data	32
4.4	External Data Sources	35
4.5	QR Module	38
4.6	Levels and Badge Levels	41
4.7	Performance Improvements	43
4.8	Course Data and Database Editor	44
4.9	Wildcards	46

In this chapter, we present the problems that were found and how we fixed each of them, as well as the work that was done to improve the existing functionality.

4.1 GameCourse in the Virtual Machine

At the beginning of this work, Moodle and the production version of GameCourse already existed in a virtual machine in pcm.rnl.tecnico.ulisboa.pt [4]. This time around, it was necessary to update Moodle from version 3.8.1+ to the most recent version which, at the time, was version 3.10.1. This upgrade was performed and it didn't require any changes on the GameCourse system's side.

Up until then, there was only one instance of GameCourse on the virtual machine which was the production environment (<https://pcm.rnl.tecnico.ulisboa.pt/gamecourse/>). Because during this work we would need to test our code in the virtual machine before moving any fixes/improvements into production, we created a test environment called GameCourse test (https://pcm.rnl.tecnico.ulisboa.pt/gamecourse_test/). We also created a database to be exclusively used by this test environment. Any changes or extensions to the code are first put in this environment, and only after assuring everything is working as expected should they be added to GameCourse.

Once we had two instances running in the same virtual machine, we discovered a problem where, if a user was authenticated in one instance and went to the link of the other instance, then the user would appear as logged in as another user. We fixed this situation by identifying the user that is logged in by the username indicated in the current session variables (that is used to log in, and, for this reason, is unique) instead of by its ID in the system.

4.2 Code Cleanup

GameCourse has been an ongoing project for quite some time now and, over the years, various colleagues have made their contributions. For someone who starts working on the system and is familiarizing oneself with the code, it can be overwhelming due to the amount of functionality that GameCourse already has, especially if code that is outdated and no longer used doesn't get deleted. We decided it was time to organize the code, place functions in the right classes/files, and delete legacy code that was no longer needed.

This cleanup also included tackling a big organizational problem that we had in GameCourse. There was a folder originally called "legacy_data" due to data being gathered from that folder by a script used to set up a course in SmartBoards (GameCourse's predecessor) called loadLegacy.php. In the previous version of GameCourse, when the loadLegacy.php script was no longer used, this folder stored various resources (e.g, images, videos, html files) needed for game elements such as skills (in this case, for

the pages explaining the goal of each skill). Although the previous version of GameCourse was already programed to create an individual folder for each course that got added to the system, the data inside the course folder was not well organized. Plus, some resources like badge images and rules were still stored outside of these folders (Figure 4.1), meaning that courses would have to share these resources, limiting how customizable courses were.

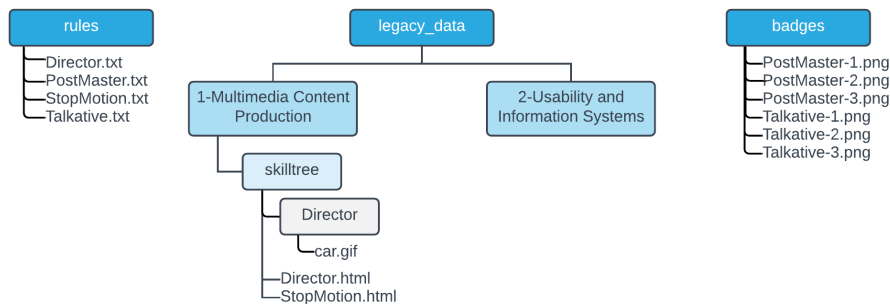


Figure 4.1: Diagram with example of old file structure.

Because the name “legacy_data” no longer reflected the content of the folder, we found that the name “course_data” would be much more suitable. Plus, each module is now responsible for saving their resources inside the correct course folder, and inside the folder designated for that module. The folders designated for module resources are created when certain modules are enabled, and the name will be the name of the module. Not all modules require the creation of data folders. Ana Nogueira [43] also designed the rule editor to save the rules in the correct folder. Figure 4.2 shows the new organization for the resources shown in Figure 4.1.

To conclude, still in the topic of cleaning up the code, we removed a field from the participation table, “moduleInstance”, since it no longer served a purpose. The queries and all code affected by this change were adapted as well.

4.3 Importing and Exporting Data

An important feature that was introduced in the previous version of GameCourse was to allow users to import and export data [4]. In that version you could import and export users, course users, courses, levels, and skills. Although this already covered most of data that it made sense to allow users to import and export, we had to fix a bug associated with exporting **users**, we were still missing the ability to import and export **badges**, and the process of importing, exporting, and duplicating **courses** needed to be upgraded.

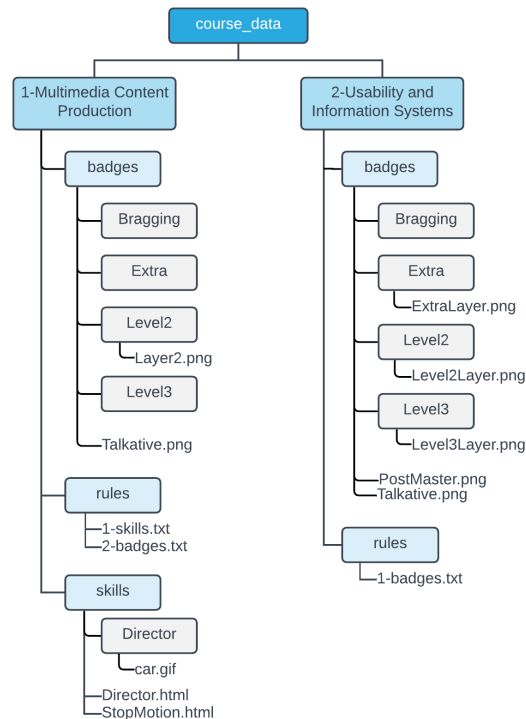


Figure 4.2: Diagram with example of new file structure.

4.3.1 Users

Our first intervention related to this topic was to fix the function that exports users. There had been a change made to the **game_course_user** table, GameCourse’s database table that stores users, where field “major” got added. At the time, the function that imports users was correctly updated, but not the export function, which was missing the users’ major. Our solution was to include the major of each user, so that the exported data could successfully be imported if needed.

4.3.2 Badges

Following the logic that was used to import and export the other types of data, we implemented the functions to import and export badges. The output of exporting the badges belonging to a course is a Comma-Separated Values (CSV) file containing the data needed to represent all of that course’s badges, including all the badge levels. The only items that are not included are the images for the badges.

When it comes to importing badges, similarly to what happens in the already implemented importing functions, we let users decide how to deal with repeated data. They can choose to either ignore duplicate badges (i.e., badges with the same name) or replace them (Figure 4.3). This feature accepts CSV files and text files, provided they follow the format of the file that comes from exporting badges.

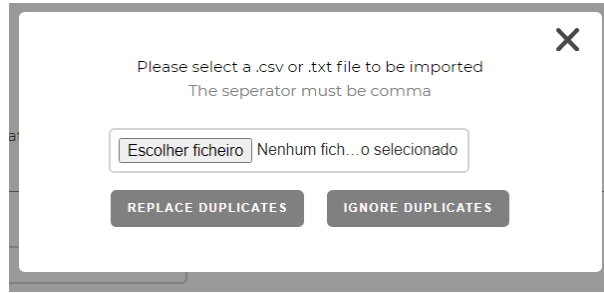


Figure 4.3: Badge import modal.

4.3.3 Courses

As we mentioned, it was already possible to import, export, and duplicate courses. Starting with the duplicate feature, the improvements described in Section 4.2 to the organization of the data folders prompted us to review the duplication of courses. This feature is mainly intended for courses whose content doesn't vary a lot from year to year. To facilitate setting up such courses, duplicating a course will create a new course that contains the same data, excluding everything user-related (e.g., users, awards, participations). In other words, it copies the shell of a course. However, this didn't include copying the module resources stored in the course's data folder as described in Section 4.2. We considered of the utmost importance to add the process of copying the content of a course's data folder when duplicating it, because otherwise it would still require professors to add these resources manually. On the topic of professors, we now add the user duplicating the course as a "Teacher" in the resulting copy of the course. That user can be removed from the course if that behavior is not intended.

Regarding the export feature, one of the limitations that this feature had was that it was only possible to export all courses together. This wasn't ideal for use cases such as backing up a course. For that reason, the first improvement we made was to give users the possibility of exporting courses individually, while still maintaining the option to export all courses in bulk. To export a single course, users must click on the export button (the rightmost button) on that course's row, featured in Figure 4.4.

NAME	SHORT	# STUDENTS	YEAR	ACTIVE	VISIBLE				
Multimedia Content Production	MCP	80	2020-2021	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
Usability and Information Systems	USI	50	2021-2022	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				

Figure 4.4: Partial interface of the courses' list page.

Clicking one of those buttons opens up a modal to select what data the user wants to export. There is a constraint that obligates course users to be included if the awards and participations are being exported, hence why we disable the selection if the "Awards and Participations" option is selected, as shown in Figure 4.5. The output is a ZIP file containing a JavaScript Object Notation (JSON) file with the

selected data and may contain a copy of the course’s data folder if the “Modules” option is selected.

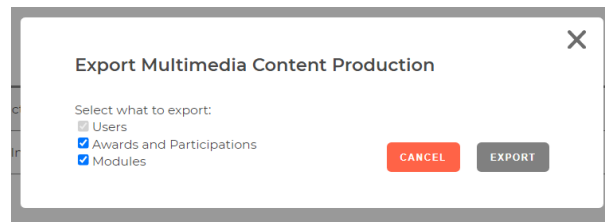


Figure 4.5: Partial interface of the courses’ list page.

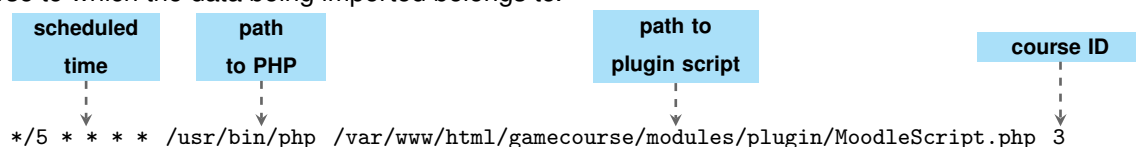
Regarding the bulk export, the output of exporting all course used to be a JSON file containing only the courses’ users and modules’ data that was stored in the database. Awards, participations, and resources in the courses’ data folders were not included. We felt this was important data to include, to be able to create an exact copy of the exported courses, so we changed the output to a ZIP file containing the same JSON file, but now including awards and participations, and the data folders of each course.

Because we changed the output of exporting courses, we had to change the logic behind importing courses. This function now accepts ZIP files, deals with the insertion of awards and participations, and knows how to copy the courses’ data folders’ content into the data folders of the resulting courses.

4.4 External Data Sources

As mentioned in chapter 3, GameCourse uses data that comes from external data sources (e.g. Moodle, Google Sheets). The way GameCourse retrieves this data is by using the Plugin module, formerly implemented by Diana Lopes [4]. This module contains a specialized script for each data source that knows how to access and parse the external data to then be inserted into the GameCourse database as a participation. These scripts run in the background and are scheduled through Cron Jobs. Still, by using the Plugin module in the real-life context of running a course, we discovered some critical problems as well as edges that needed to be polished.

The first critical problem to surface had to do with deleting and editing the schedule for these scripts to run, in a context where multiple courses coexisted in GameCourse. Every script in the Plugin module requires one argument to be supplied in order to run: the course ID (Listing 4.1). This indicates the course to which the data being imported belongs to.



Listing 4.1: Line in CronTab with scheduling for the Moodle plugin script to run.

The problem was that deleting/editing the schedule for a certain plugin script to run in a certain course would delete the schedules for that plugin to run in every course. If we took, for example, the scheduled Cron Jobs in Figure 4.6 and decided to delete the schedule for MoodleScript.php to run in course 1, the scheduling for MoodleScript.php to run in both course 1 and 2 would be deleted (Figure 4.7). The only action that was correctly implemented was creating a schedule, as no unwanted schedules were created by that action. This was very problematic because professors from other courses would stop getting new data despite the database indicating that the Cron Job was active. Every time a change was made, all Cron Jobs were checked to see if they matched the job being deleted, edited, or created. This comparison, however, was being made based solely on the path to the script and not taking into account the course ID argument. Thus, the solution for this problem was to check both the path and the course ID, and only act upon jobs that matched both values.

```
*/10 * * * * /usr/bin/php /var/www/html/gamecourse/modules/plugin/MoodleScript.php 1
*/5 * * * * /usr/bin/php /var/www/html/gamecourse/modules/plugin/GoogleSheetsScript.php 1
*/20 * * * * /usr/bin/php /var/www/html/gamecourse/modules/plugin/MoodleScript.php 2
* */10 * * * /usr/bin/php /var/www/html/gamecourse/modules/plugin/GoogleSheetsScript.php 2
```

Figure 4.6: Scheduled Cron Jobs.

```
*/5 * * * * /usr/bin/php /var/www/html/gamecourse/modules/plugin/GoogleSheetsScript.php 1
* */10 * * * /usr/bin/php /var/www/html/gamecourse/modules/plugin/GoogleSheetsScript.php 2
```

Figure 4.7: Result of wrongful deletion.

The second problem had to do with the script that deals with data from Moodle, more specifically with retrieving professors' ratings on forum posts. A special feature that already existed in the Plugin module is that records are fetched iteratively based on time, meaning that when the script runs it will only retrieve data from after the last time the script had ran. The key to make this work is to make the right time comparisons, and, unfortunately, this was not working correctly for the grades in Moodle forum posts. The script was checking the time in which the post had been created, instead of the time in which it had been graded. This means that if the post was not created and graded in the same interval where the script was not running, the grade would never be registered in GameCourse. Ultimately, the solution was to check the time in which the grade was given against the last time last time that data was fetched from Moodle.

In addition to these fixes, some much needed improvements were made to the Plugin module. The improvements contribute to a more customizable system, that may be fit for a larger variety of courses. One thing that made professors' jobs harder was the sheet naming requirements that the plugin for Google Sheets had. Participations have an optional field called "evaluator", for situations where it is possible to know who the evaluator was (e.g., who evaluated the skill, who evaluated the lab assignment). If each professor had a page in the same Google Sheets sheet in which only they gave input regarding

students, it was possible for Game Course to link those inputs to the evaluator. But there was a catch: the name of the page had to be of the format “[username]_”. Soon, it became tiresome to have to check the names of the pages since the usernames contained numbers, making them hard to distinguish at first glance. Plus, the users were not made aware of this naming requirement and only experienced users would know to be cautious when setting up their sheet. Since the user already had to insert the name of the pages (Figure 4.8), we took advantage of that and created a new input box for inserting the page owner’s username as well (Figure 4.9). This way, the link between a page and a professor is kept, while still allowing professors to freely choose how to name their pages. Furthermore, a single professor can now own multiple pages at the same time (as long as they have different names), whereas before that was impossible to achieve since Google Sheets doesn’t allow for two pages to have the same name.



Figure 4.8: Google Sheets configuration page before changes.

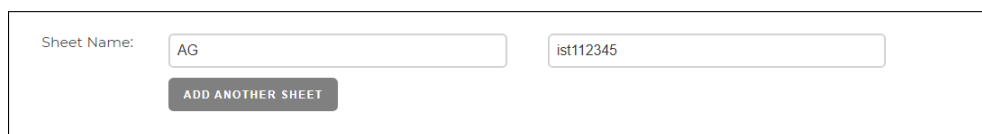


Figure 4.9: Google Sheets configuration page with new input box.

The set of improvements we are going to talk about next come from the need to include more types of grading. Every course is different, and since one of the biggest goals in GameCourse is to accommodate different courses, the system must be as inclusive as possible, especially when it comes to evaluation. Gamification should facilitate knowing how a student is progressing [10], and this implies that, for GameCourse to be suitable for a course, it must provide a way to access every grading element in that course for the total XP to be determined.

We started by adapting the script to be able to fetch and parse gradings that students gave to each other’s work (as it was a new form of evaluation in MCP), as well as the professors’ grades on those same posts, which would ultimately be the official grade on that post. These two types of grades had to be fetched separately because the data for them was stored in different tables within Moodle’s database. In the case where the grade was given by a student, we decided to insert that student as the evaluator (while before the evaluator could only be a professor), because this allows for the monitorization of how many times a student has graded their peers’ work.

Regarding Moodle, we could now get the grades from quizzes, forums, peerforums (where students can grade each other’s work), and some activity logs, but this was still not enough. Professors often

choose to use Moodle assignments to collect and evaluate student work. When a professor creates an assignment, students can then submit their work, and, afterwards, the professor can grade each submission. The problem is that GameCourse's plugin for Moodle wasn't programmed to retrieve these grades, and, with interest being shown to use GameCourse in a course that relies on assignments, we decided it was time to make this upgrade. Thus, we programmed a way to access the tables in Moodle's database which contained the data we wanted, as well as to parse that data into entries in GameCourse's participation table. Much like the rest of the script, we make the appropriate time comparisons for this data to be retrieved incrementally, and we carefully check for updates (e.g., a grade that was altered) should there be any.

4.5 QR Module

As mentioned in Chapter 3, professors can reward students for participating in class by giving them QR codes that represent said participation and can possibly count towards a badge. These QR codes are generated inside the GameCourse system, by enabling and using the **QR module**. Although this module had already been implemented, we took the opportunity to fix the bugs we found and make the changes we identified as necessary.

4.5.1 QR Plugin Periodicity

Initially, the QR module worked similarly to the Plugin module: a Cron Job had to be scheduled for a script to read the source's data and parse it into participations to be inserted in the participation table. There was, however, a noteworthy difference. Unlike the other data sources, the data from this source was already stored in a table inside GameCourse's database. Because the QR module is fully integrated with the GameCourse system (Figure 3.4), we figured we were able to simply insert the participations directly to the participation table when a QR code is redeemed by a student, eliminating the need for the scheduling and usage of the QR plugin script.

4.5.2 Bugs and Security Issues

There are two main activities related to the usage of QR codes in GameCourse: generating codes and using a code to register the participation. Professors can generate codes through the User Interface (UI). They can then give the codes to users who participate in class, but the process doesn't end here. To register the event as a participation, students must scan the code and fill out the information on the form depicted in Figure 4.10.

IST - DEI - CGM

Multimedia Content Production - Spring 2020-2021

Bonus XP - Lecture Active Participation

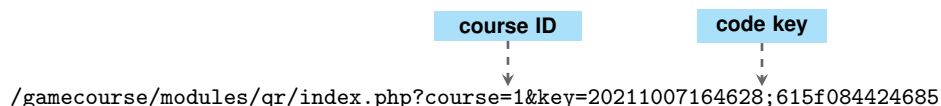
Your IST Student Number:
Type of Class:
Lecture Number:
All fields are required.

Figure 4.10: Form for redeeming a QR code.

The reason why we first started working on this module was because an error message appeared every time this form was submitted, saying that the student wasn't enrolled in the course even if they were. We discovered that the form data wasn't being handled correctly. The query used to verify if the student was enrolled was searching for the student number in a table that doesn't store this value, explaining why the assertion failed every time. This was promptly fixed, and students can, once again, get their participation registered.

While fixing this bug, we took the opportunity to analyze the rest of the code pertaining to the security checks made when a student redeems a code. The conclusion we came to was that, besides checking if the user is enrolled in the course, two additional checks should be made:

- Checking if the code exists in the database: every time a QR code is generated, an insertion is made to a table in the database called **qr_code**. That entry contains a key that uniquely identifies the QR code, and the course ID. These two arguments are provided in the request to redeem a QR code (Listing 4.2).



Listing 4.2: Request to redeem a QR code.

If students scan a QR code provided by a professor, the key will always be an existing key. The only way to provide an inexistent key would be to try to enter one manually. If provided with an inexistent key, the execution does not fail, nor does it allow students to cheat by redeeming an inexistent code. However, by not checking if the code exists, we are also not registering failed attempts due to a code not existing. For security reasons, after verifying that the student belongs to the course, we now verify if the key exists. If it does not exist, we now register the failed attempt as an error in the **qr_error** table. Professors have access to this data and are able to see if any exploitations are being made to find the code keys.

- Checking if the code was already redeemed: the most important verification that was lacking was to check if a QR code had already been redeemed. As discussed earlier, when a code is generated, an insertion is made to the qr_code table containing the course ID and the key. Still, this table has other fields, one of them being the “studentNumber”, containing, as the name suggests, student numbers. When a code is generated, this field is null. Only when a student redeems the code does the entry containing that code’s key get updated to contain that student’s student number in the “studentNumber” field. The problem here was that, before doing this update, no verification was being done to check if the code had already been redeemed, which means that if a second student redeemed the same code, their student number would overwrite the first student’s student number, who would essentially have the code stolen from them. The solution was to add the verification to see if the code has already been redeemed, and, if it has, an error message appears, and the failed attempt is registered as an error in the qr_error table.

4.5.3 Configuration Page

The QR module was the only module that came with a designated page, automatically created by enabling the module (Figure 4.11). The rest of the modules that professors can configure or interact with all have a designated configuration page accessible through the “Modules” menu. To make the system more cohesive, we removed the page that was originally created for the QR module and created a configuration page instead.

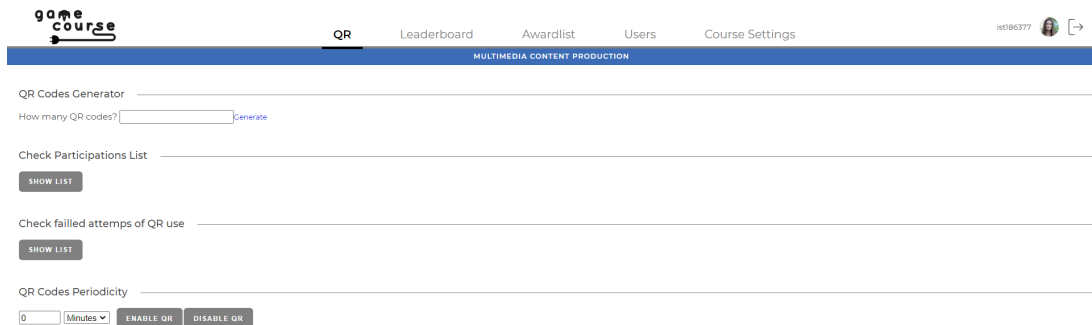


Figure 4.11: Previous QR module page.

We kept a similar layout, because our main goal was to replace the page by a configuration page. The overall appearance of the page should be improved in the future. Nevertheless, we made some changes we feel improved the page.

Besides removing the bottom section regarding periodicity, since it no longer applies, we created a table for failed attempts to appear directly on the page instead of opening a new page to show them

(Figure 4.12). The reason why we didn't do the same thing for the participations' list was because, by the end of the semester, this table tends to have a lot of records. For example, in MCP, by the end of the semester, over six hundred QR codes had been generated.

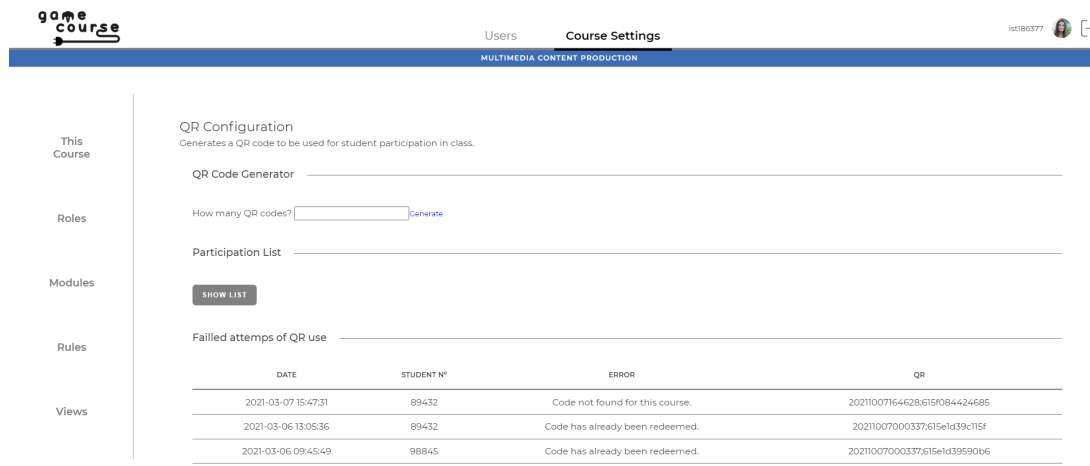


Figure 4.12: QR module's new configuration page.

4.6 Levels and Badge Levels

As mentioned in chapter Chapter 3, there is a set of database tables (Appendix A) that is created when GameCourse is set up for the first time. However, additional tables are created when certain modules are enabled for the first time. Even though it would be simpler to create every possible table during the first set up, we draw two main benefits from keeping this behavior:

- **Separation of concerns:** keeps the code tidier, which is important to ensure **modifiability** – the “degree of ease at which changes can be made to a system, and the flexibility with which the system adapts to such changes” [48]. Developers know which tables are specifically used by each module. Modules shouldn't make changes to tables that belong to other modules.
- **Flexibility:** if a module is developed outside of GameCourse long after the first set up, the developer simply has to import that module and the tables will be created when this new module is enabled. Otherwise, the tables would have to be created manually in GameCourse's database.

When we first started working, this was being respected for the most part except for the **level** table, which was being created during the first set up even though it belongs to the **XP and Levels** module. This module unlocks the possibility to have levels in terms of XP (reaching 1000 XP puts you in level 1, reaching 2000 XP upgrades you to level 2, and so on), and provides expression language functions to refer to levels and XP. We fixed this situation so that the level table is only created when the XP and

Levels module is enabled for the first time, as well as deleted when there's no longer a course with this module enabled.

While making this change, we checked for any dependencies that there might be that could explain why this table was being created during set up. In fact, the separation between modules was not being respected: module **Badges** was using the level table to store its own levels. As the name indicates, module Badges enables the creation of badges. Each badge has at least one level, representing the bare minimum a student must do to earn that badge. An example of a badge with multiple levels is the “Post Master” badge that was used in MCP, where the goal for the first level was to make twenty posts, the goal for the second level was to make thirty posts, and the goal for the third level was to make fifty posts. Although it was just one badge, each level awarded students a certain amount of XP. There is a database table – table badge – that stores data regarding each badge, but the data regarding each badge level was being stored in the level table, mixed with the XP levels we talked about previously. The association between a badge and its levels was made through a third table called badge_has_level (Figure 4.13).



Figure 4.13: Original database schema regarding badges and levels.

Even though this architecture was working as expected, using this table to store two distinct types of levels had its downsides. It required extra care to separate XP levels from badge levels every time a query was made to the level table, as to not get the wrong type of level by mistake. This had a negative impact on the system's performance and the modifiability of the code. The solution we came up with was to get rid of the badge_has_level table and create a new table dedicated exclusively to badge levels: the **badge.level** table (Figure 4.14). We have the same number of tables as before, but accessing badge levels now involves, at most, joining two tables, while before it would require joining three tables. Besides, there is no longer the need to check if a level belongs to a badge or not while querying for XP levels and vice-versa, nor the risk of making mistakes if such check is not done properly. Like the other tables belonging to module Badges, the badge.level table is only created when the module is enabled

for the first time and deleted when it is no longer enabled in any course.

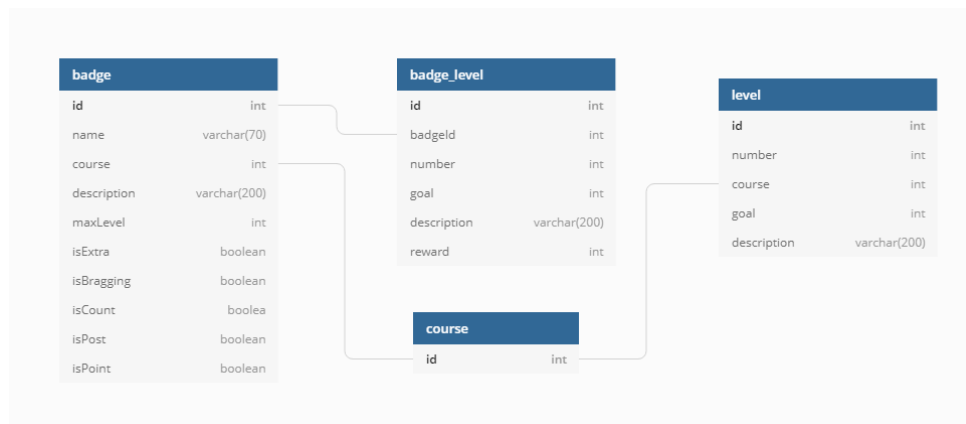


Figure 4.14: New database schema regarding badges and levels.

4.7 Performance Improvements

A problem that we had in the GameCourse system was its **performance**. Some pages took a long time to load, which prompted us to investigate the cause of that inefficiency. The biggest problem that we seem to have in GameCourse is the logic being how views are loaded when we access a page that contains views. Because this was outside of this work's scope, we focused on making other improvements that would speed up the backend side of GameCourse.

The first major change we made had the intent of reducing the load time of the Leaderboard and Profile pages. These pages have some elements in common, one of them being the **total XP**: the profile page shows the total XP for a specific student and the leaderboard shows the total XP for each student in the course. While analyzing the code, we realized that every time one of these pages was loaded, the total XP was being calculated on demand. It may not seem like a big problem for the profile page, but for courses like MCP with close to ninety students and nearly four thousand awards by the end of the semester, calculating the total XP on demand becomes very time consuming. Moreover, the total XP is something that only changes when users receive new awards, i.e., when the rule system runs. So, we came up with a plan to have these values in the database, one query away. In collaboration with Ana Nogueira [43], who worked on the rule system, we created the **user_xp** table which keeps, for each student, their total XP and their current XP level (to avoid having to calculate that as well). This table is created when module **XP and Levels** is enabled for the first time, following the logic discussed in Section 4.6. Since all changes in XP are a consequence of a GameRules' execution, Ana Nogueira [43] created the means so that GameRules recalculates the XP and XP level of each user who participated since the last time the rule system was called. We then refactored Expression Language functions so

that they read the users' XP level and total XP from the database rather than calculating it at runtime.

The second move we made in trying to mitigate this problem was to refactor multiple functions throughout the system, to get rid of unnecessary complexity. One of the functions whose time complexity we were able to improve was the function used to import levels in the **XP and Levels** module. For each new level, a call was made to another function that would then make a single insertion to the database. We replaced this behavior by waiting to insert all new levels in a single query, which is considerably faster than using separate single-row INSERT statements [49].

Still trying to eliminate any unnecessary complexity, we discovered that one of the most used functions throughout the entire system, function `getCourse`, was being misused. This function receives two arguments: the course ID and a flag signalling if modules should be initialized, with the default value being true, meaning that, by default, if no argument is provided, modules will be initialized. Initializing a module includes checking if its data tables and data folder (if the module requires so) have already been created, creating them if they don't yet exist, and registering its EL functions in the dictionary.

The problem was that in most of the places where this function was called, no argument was being passed for the flag, which means that the modules were being initialized by default. So, we went case by case, identifying where it wasn't necessary to initialize the modules and passing that flag as false. We came up with forty-three different functions that were, needlessly, requesting this extra computation of initializing the modules. Ordinary tasks like adding or editing a user, importing and exporting items, and even various Application Program Interface (API) functions, among others, were positively impacted by this change in terms of time.

4.8 Course Data and Database Editor

While navigating a course in GameCourse, professors have exclusive access to a page containing information about the course Figure 4.15.

As you can see, this page was rather empty, only informing its viewers about the name and ID of the course. We decided we wanted to make this page more useful for professors, so we added three more indicators: the number of active users, the number of awards, and the number of participations (Figure 4.16).

As you may notice in Figure 4.16, we also added a button to that page. When you click on that button, you get to choose one out of two tables: table award or table participation (Figure 4.17). Choosing one of the options then takes you to one of the new features in GameCourse: a database content viewer and editor.

With the goal of making GameCourse as accessible to its users as possible in mind, we realized, during this semester of MCP, that the professors should be able to access certain data on GameCourse's

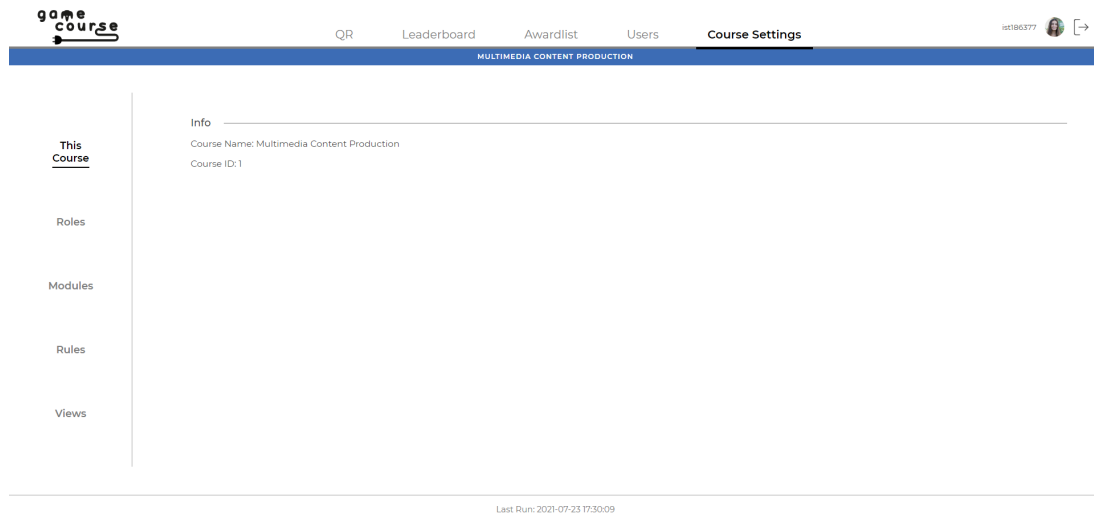


Figure 4.15: Previous version of the “My Course” page.

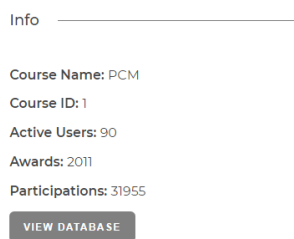


Figure 4.16: New information in “My Course” page.

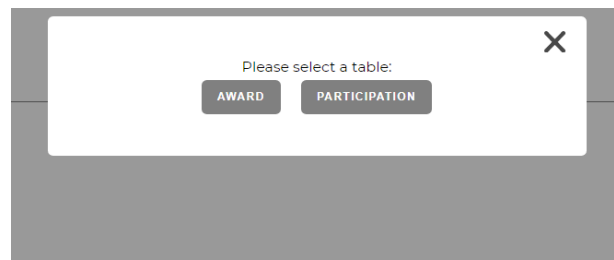


Figure 4.17: Options to open the table data editor.

database through the UI. It was especially important to see what is in the two tables mentioned above, the award table (which stores all the awards that were given) and the participation table (which stores all participations). One of the reasons why we found this to be an important feature was for correction checking purposes, as professors can now easily find out if any information is missing or incorrect. The second reason comes down to how easy it now is to analyze students’ participation in the course, and how fruitful their participation is being in terms of awards.

In the new page, depicted in Figure 4.18, the entries are displayed in a table, separated into pages to facilitate viewing since both the award and participation table tend to have a lot of entries. The entries can be sorted by each of the columns (ascending or descending), each entry can be edited or deleted, and new entries can be added.

Figure 4.18 also shows the global search bar, which filters the results by matching the search text to any column, and the individual search bars in each column, which only search for matches within the corresponding column. As a use case, let's imagine we want to see all quiz grades for student “Daniel Gonçalves”. By typing the name on the search bar for column “name” and “quiz” in the search bar for column “type”, we get the results we wanted.

award

Show 10 entries

ID	NAME	STUDENTNUMBER	DESCRIPTION	TYPE	MODULEINSTANCE	REWARD	DATE
1	Gonçalo Santos	77915	Initial Bonus	bonus		500	2021-03-01 16:10:31
2	Carlos Antunes	81525	Initial Bonus	bonus		500	2021-03-01 16:10:33
3	Joana Sesinando	81583	Initial Bonus	bonus		500	2021-03-01 16:10:35
4	Miguel Ramalho	81948	Initial Bonus	bonus		500	2021-03-01 16:10:36
5	Afonso Luis	83419	Initial Bonus	bonus		500	2021-03-01 16:10:37
6	Pedro Campos	83951	Initial Bonus	bonus		500	2021-03-01 16:10:39
7	Francisco Sousa	84051	Initial Bonus	bonus		500	2021-03-01 16:10:40
8	João Vaz	84103	Initial Bonus	bonus		500	2021-03-01 16:10:41
9	Sarah Leão	84489	Initial Bonus	bonus		500	2021-03-01 16:10:43
10	Alexandre Manso	84694	Initial Bonus	bonus		500	2021-03-01 16:10:44

Showing 1 to 10 of 2,011 entries

Figure 4.18: Table data viewer and editor.

award

Show 10 entries

ID	NAME	STUDENTNUMBER	DESCRIPTION	TYPE	MODULEINSTANCE	REWARD	DATE
260	Daniel Gonçalves	98845	Quiz Grade	quiz	1	450	2021-03-08 18:48:57
581	Daniel Gonçalves	98845	Quiz Grade	quiz	2	600	2021-03-15 17:02:46
947	Daniel Gonçalves	98845	Quiz Grade	quiz	3	450	2021-03-23 00:02:06
1164	Daniel Gonçalves	98845	Quiz Grade	quiz	4	600	2021-04-05 17:03:25
1594	Daniel Gonçalves	98845	Quiz Grade	quiz	5	150	2021-04-12 17:39:28
1865	Daniel Gonçalves	98845	Quiz Grade	quiz	6	150	2021-04-19 17:40:59

Showing 1 to 6 of 6 entries (filtered from 2,011 total entries)

Figure 4.19: Search by two individual columns.

The columns that are shown don't correspond exactly to the fields in each of the tables. To identify the user, both the award and participation table simply keep the ID of that user in the system. Since showing the ID of the user would make it harder to search for, edit, and add entries, before showing the results we get, for each entry, the corresponding name (first and last name) and student number. When adding or editing entries, the user is identified by its student number because it is unique, as opposed to names (Figure 4.20). Furthermore, not all fields can be edited. One of the fields that didn't make sense to be editable was the entries' ID field, as it is an auto-increment field and must be unique.

4.9 Wildcards

As a game element, the skill tree already allowed student to choose from a broad number of paths in terms of practical work. However, with the beginning of a new semester of MCP came a new family of skills: **wildcards**. Before talking about this novelty, we must first talk about two terms that we will use throughout this section: a skill being "unlocked", and a skill being "completed". A skill is unlocked if the

Figure 4.20: Editing a table entry.

student has completed the pre-requisites for that skill, i.e., the skills that the skill depends on. A student completes a skill once they receive an award for that skill. For that award to be given, the skill must be unlocked.

Up until now, skills could only depend on other skills, but, this time around, skills can also depend on a tier. For example, looking at Figure 4.21, the Kinetic skill depends on the Course Logo skill and on tier Wildcard which has three skills (Pixel Art, Morphing, and ReTrailer). This means that, in order to unlock the Kinetic skill, besides completing the Course Logo skill, a student would have to complete one of those three skills on tier Wildcard. To keep the example simple, we named the wildcard tier “Wildcard”, but it could have any other name and still be used as a wildcard tier.

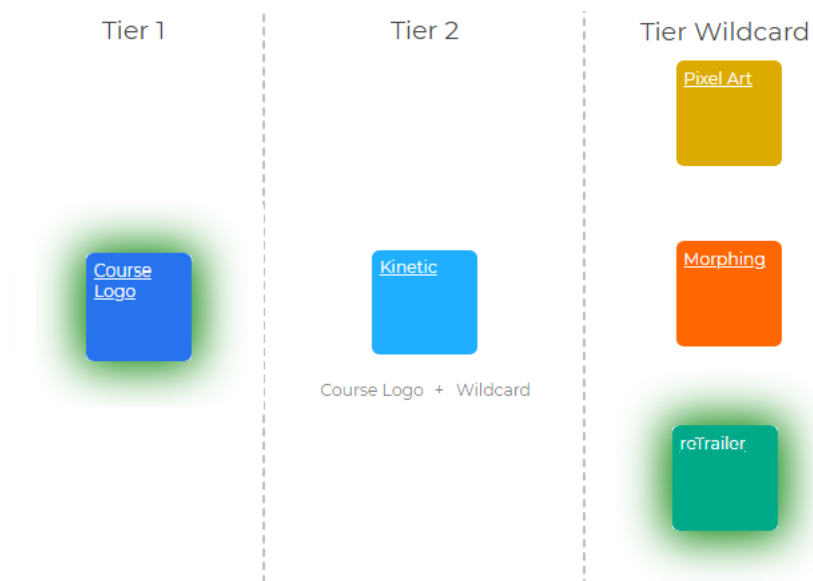


Figure 4.21: Example of skill dependant on wildcards.

Wildcards broaden the number of different paths a student can take to earn XP from the skill tree and thus students have more freedom to choose how they want to move forward in the course. There is, however, a rule when using wildcards: they can only be used once, thus students must think strategically. If this rule didn't exist, the game would possibly become too easy. To make this possible, we had to find a way to:

- Allow skills to depend on a whole tier instead of a specific skill.
- Keep track of used wildcards, including the context in which they were used.

Regarding the first goal – allowing skills to depend on a tier –, it would be possible to simply add a dependency on each of the skills on that tier. However simple, this was not a solution for our problem. If the tier that we wanted our skill to depend on had a large number of skills, it would not only be laborious to add each dependency one by one, but most of all it would be extremely prone to errors. Our solution was to change the way dependencies are saved to maintain whether the dependency is on a skill or on a tier, and the ID of the skill (if the dependency is on a skill) or the tier (if the dependency is on an entire tier). With this solution, the user only has to insert one dependency which prevents errors and makes the process much more efficient.

The second goal was trickier to achieve. Ultimately, there had to be a table to keep the record of used wildcards. When a student completes any skill, a record is inserted into the award table. However, when a student uses a wildcard to complete a skill, besides the record on the award table, a record is inserted in the **award_wildcard** table that comes from enabling module Skills.

The rules for skills that depend on wildcards only need to know how many wildcards a student has used up, and how many skills from the wildcard tier that student has completed, so we implemented functions in the Skills module that provide this information. Ana Nogueira [43] adapted the rule system to be able to deal with the peculiarities that come with wildcards, namely the insertions into the award_wildcard table, and deciding whether or not a wildcard will be spent. In cases like one depicted in Figure 4.21, it's mandatory to use a wildcard, but because skills can have multiple dependencies, it is possible, in other cases, to save the wildcards for later. Looking at the example in Figure 4.22, it is possible to unlock the Stop Motion skill without using a wildcard. In this case, if the student has completed both Alien Invasions and Doppelganger, that dependency takes precedence over the dependencies containing wildcards and no wildcard will be spent.

In the context of this new feature, we added more expressiveness to the EL, for it to be used in both the rules and the views. An EL function that was created with the intent of being used in views to show the names of the dependencies, as depicted in Figure 4.22, was the **dependencies** function, which receives a dependency ID and returns the names of the skills and/or tiers that make up that dependency.



Figure 4.22: Example of skill with multiple dependencies.

We also found it essential to be able to show students how the number of wildcards they have already spent. So, we created the **usedWildcards** EL function, which receives a tier name and a user ID, and returns the number of wildcards from that tier that the user has spent. Because the skill tree view (Figure 4.23) is created using a loop that iterates over each tier, we needed to know whether each of them was a wildcard tier, to only show the number of used wildcards on those tiers. To make this conditional visibility work, we created an EL function - **hasWildcards** - that receives a tier name and returns either true or false according to whether the tier is a wildcard tier or not.

The most important function, due to it being used to write the rules for skills that depend on wildcards, was the **wildcardAvailable** function. It was also the most laborious to implement since it had to be changed after the first implementation. The purpose of this function was to simply return true or false according to whether a user still had wildcards from a certain wildcard tier to spend. This was then used in the rules to indicate if a user could use a wildcard to complete a skill. However, every time the rule system runs, every rule runs for each user, even the rules for skills that the user has already completed. While conducting tests, we realized that wildcards were being spent on the same skills more than once. To avoid spending multiple wildcards on the same skill, we changed the **wildcardAvailable** function to return true or false according to whether a user can spend a wildcard on a certain skill. The difference is that the function now receives the name of the skill the user is trying to complete, as well as the tier name and the user ID like before. In terms of behavior, the output follows the same logic as before, but we also check if the user has already completed the skill, and, if so, the output will be false to signal the rule system that no wildcard should be spent for that skill.

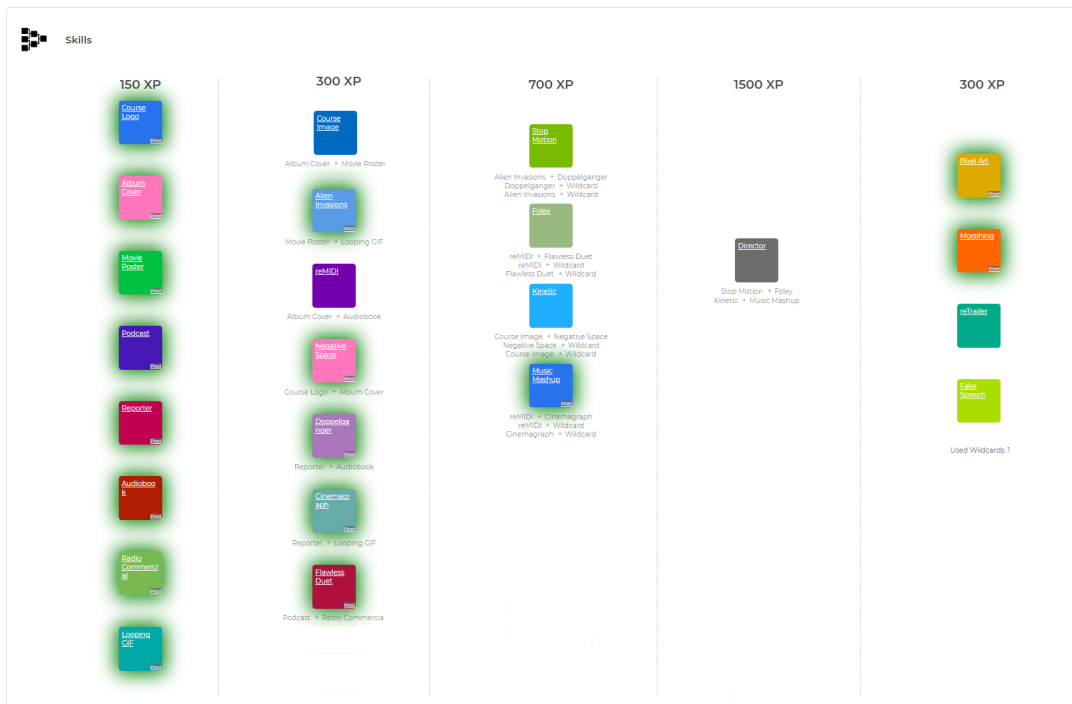


Figure 4.23: Skill tree.

5

From GameCourse to GameCoursePersonal

Contents

5.1 Profiler	53
5.2 Predictor	55
5.3 Clusters	56
5.4 Profiling Module	57
5.5 Towards differentiated leaderboards	61

Once we had solved the existing problems and made the changes described in Chapter 4, we decided that we finally had a solid base to work on and take the first steps towards converting GameCourse into an adaptive gamified environment. In this chapter, we describe the development of a student **profiler** mechanism and a **predictor** for the ideal number of clusters. The work done towards achieving system adaptation included the creation of a new module - the **Profiling module** - which encapsulates and provides these new functionalities. We also explore different types of leaderboards, as a way of satisfying the needs of different types of students.

5.1 Profiler

The key component when building an adaptive system is the **profiler**. To implement a system that can adapt to its users in a meaningful and effective manner, it must first know how to categorize them. As presented in Section 2.3, there are many ways of approaching this problem.

For our work, we decided that the best solution would be to use the same machine learning approach as Nabizadeh et al. [45], developed specifically for MCP. This approach consists in a nine-step procedure that includes data preprocessing. We start by calculating the XP that each student accumulated thus far. We then generate a large dataset, which we will call Main data, that includes three types of data:

- Badges earned and skills completed, on each day, by each student.
- Students' total XP, on each day.
- Actions performed, on each day, by each student (e.g., viewing a specific item, postin on a Moodle forum).

Each row represents a student, and each column has the name of a badge, skill, action, or “xp” for the accumulated XP. The columns represent the features used later on in the procedure. Each column also has a row for the date since we must sort these columns by date. We then drop the constant features from the resulting dataset and proceed to normalizing it. The next step is to use the **K-Means** algorithm [47] to generate four clusters based on the XP calculated on the first step. If a cluster comes up as having less than four students, we generate three clusters instead of four. Knowing the number of students in each cluster, we then use **Synthetic Minority Oversampling Technique (SMOTE)** to balance the Main data. The goal of this step is to add dummy students to the smaller clusters so that they have a similar size to the highest populated cluster. This is done in preparation for the next step, which is to apply feature selection on the balanced data to detect the most important features. The algorithm used for this step is the **Boruta** algorithm. After selecting the most important features, we use the **Random Forest** classification algorithm to generate the prediction model.

Because we wanted to automatize the process, we implemented the procedure into a Python script. Our two main options as far as programming language is concerned were R and Python, since both have libraries and packages that are excellent for machine learning, but we ultimately chose Python to facilitate future work being done on this script, as it is more widely known than R. Furthermore, since other parts of the system are also implemented in Python (e.g., the rule system), by choosing Python over R we avoid increasing the number of programming languages used in the project.

We refer to the algorithm for determining the cluster each student belongs to as the **profiler**. The profiler requires three arguments to be given: the **course ID**, the **number of clusters**, and the **minimum cluster size**. This script is the implementation of the algorithm developed by Nabizadeh et al. [45], explained above. Both the accumulated XP from the first step and the data that later composes the Main data are represented using dictionaries, and we iterate over the awards and participations from the course identified by the course ID, to collect the necessary data. We had to be careful in calculating the XP because there is a maximum amount of XP that students can earn from skills and badges. When a student reaches that threshold, they can still earn badges and complete skills, so we must include those actions in the Main data, even if XP earned from those actions must not count towards the XP calculation. Regarding the dates, even though we can get the exact time, down to the seconds, in which a participation occurred, or an award was given, we only take into account the date in the International Organization for Standardization (ISO) date format [50]: YYYY-MM-DD. We then sort the keys in the dictionary containing the data for Main data, which correspond to days. Afterwards, we iterate over each day to transform the data in the dictionary into a Dataframe (from the *pandas* library [51]), that will represent the Main data. The next step is to drop the constant features, using the “VarianceThreshold” feature selector from the *scikit-learn* library [52]. Throughout the implementation, this library was used in every situation where it could provide a solution, as it is one of the most powerful and widely used Python libraries for machine learning. To normalize the resulting Dataframe, we use the “MinMaxScaler” from the same library.

At this point in the execution, the data preprocessing phase has been concluded. The step that follows that phase is to generate the number of clusters (n) that the profiler was given as an input, using the K-Means algorithm, for which we used “KMeans” from the *scikit-learn* library [52]. If any of the clusters doesn't reach the minimum cluster size (given as input), we generate $n - 1$ clusters instead. To balance the Main data, we use the “SMOTE” class methods from the *imblearn* library [53], which allow us to perform oversampling using SMOTE. It is then time to detect the most important features in the balanced Main data with the Boruta algorithm. We chose to take advantage of the “BorutaPy” class from the *boruta* library [54] for this step, with a maximum of 2000 iterations. Finally, for the last step, we use the “RandomForestRegressor” class from the *scikit-learn* library, with a maximum depth of 10.

The configuration values mentioned, like the maximum depth for the Random Forest, were chosen

based on the algorithm developed by Nabizadeh et al. [45]. Most of the classes used came from the *scikit-learn* library [52]. Still, we had to use other libraries for the methods for which *scikit-learn* doesn't yet provide an implementation. Those libraries were chosen due to being the most commonly used for each of the algorithms and techniques that were lacking in *scikit-learn*.

Once the results are ready, they are written into a file called “[course ID] – results.txt”. The output written to this file consists in the cluster centre for each cluster and the cluster that each student belongs to.

5.2 Predictor

As explained in the previous section, one of the arguments passed to the profiler is the **number of clusters**, which, for MCP, had already been determined as being four [45]. However, we wanted to make this feature as appealing to all courses as possible, and it would be hard to determine the ideal number of clusters for other courses. We anticipated professors wouldn't know which number to choose or even choose a number that would lead to unsuccessful results, so we decided to create a way of advising professors on how many clusters to create. Because there are multiple methods for determining the optimal number of clusters for K-Means [47], we implemented two options based on the two most well-known methods for predicting the best number of clusters for K-Means [47], but other methods can easily be integrated into the system in the future. Those two methods are the **Elbow method** [46] (also used by Nabizadeh et al. [45]), and the **Silhouette method** [55].

Similar to the approach taken for the profiler, we implemented a Python script we'll refer to as the **predictor**. The predictor requires two arguments to be given: the course ID, and a character representing the method we wish to use: “e” for the elbow method and “s” for the silhouette method. The prediction is made based on the XP that each student accumulated thus far, since this is the data that is fed into K-Means in the profiler, stored, in this case, in an array instead of a dictionary since it facilitates the next step. We then start the actual prediction. Before explaining how we implemented each of the methods, it is important to explain how these methods work. To determine the ideal number of clusters, it is necessary to calculate a score for each number of clusters, and then decide the best number of clusters based on those scores. This means that we had to choose which numbers we wanted to test for. We decided it didn't make sense to test for less than two clusters, as having one cluster would be equivalent to not having any (i.e., the students would remain indistinguishable). We also decided that, for the contexts in which GameCourse is currently used, having more than ten clusters was not reasonable, not to mention that the bigger the array of numbers of clusters we test for, the more overhead we add to the execution. The procedure then becomes similar for both methods: for each number of clusters, we apply the k-means algorithm to the data (using “KMeans” from the *scikit-learn* library [52]) as in the

profiler), and we then calculate the scores. Each score is stored in a list.

In the case of the elbow method, each score is the sum of squared distances of samples to their closest cluster center and is given by an attribute - **inertia_** - from the “KMeans” class. We then create an instance of the “KneeLocator” class from the *kneed* library [56], that receives the list of scores. This class has an attribute – **elbow** – which indicates the elbow point that corresponds to the number of clusters which has achieved the best score.

For the silhouette method, each score is given by the silhouette coefficient, which we calculate using “silhouette_score” from the *scikit-learn* library [52]. The recommended number of clusters is the one that gets the highest score.

Similarly to the profiler, the predictor writes its output (i.e., the recommended number of clusters) to a file called “[course ID] – prediction.txt”.

5.3 Clusters

So far, we’ve covered how to find the best number of clusters and how to determine the cluster each student belongs to, so now it’s time to discuss how to represent those clusters in the GameCourse system.

In Chapter 3, we mentioned that users can have multiple roles inside a course, the default ones being “Teacher” and “Student”. This is how we individualize a group of users, so that we can give each group/role the proper permissions and show them the correct views. Since those roles can be assigned or removed at any time, our solution was to represent the different clusters as roles. When a student is assigned to a cluster, besides keeping the “Student” role, they will be assigned the role that represents that cluster. If the student is moved to another cluster, that previous role is removed and a new role is assigned. Because we need to be able to differentiate the “cluster roles” from the other roles, we create them as children of a role called “Profiling” (Figure 5.1), the latter being a child of the “Student” role. As an example, a student belonging to the Regular cluster would have two roles: role “Student” and role “Regular”. The views can then be generated with the aspects that match the most specific role possible.

It is important to note that, although the “Achiever”, “Regular”, “Halfhearted”, and “Underachiever” roles are at the same level of the hierarchy, the way we represented them in Figure 5.1 was intentional, since their order as children of the “Profiling Role” is important. This order will determine the name of the clusters, based on the cluster centers – the average of all points (elements) belonging to each cluster [57]. In this case, the cluster with the highest center will be represented by role “Achiever”, the runner up by role “Regular”, and so on.

Because the names and order of the clusters is determined by the children of the “Profiling” role, professors can choose the names and create as many clusters (roles) as they wish. If these roles are

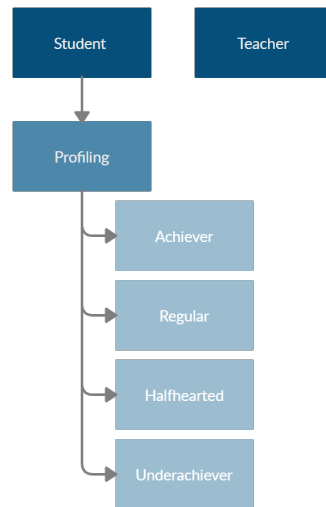


Figure 5.1: Role hierarchy diagram.

not created prior to running the profiler, the names and order shown in Figure 5.1 are used by default. If these names aren't enough, we give the remaining clusters default names: "Cluster1", "Cluster2", and so on.

5.4 Profiling Module

In this section, we describe the development of the **Profiling module**, responsible for processing user data and assigning students to clusters as a result. Like any other module in GameCourse, it can be enabled or disabled.

5.4.1 Configuration Page

Because this is a brand-new module, we had to create a configuration page where the new profiling features can be used. When designing this page, our goal was to provide data regarding students' profiles but, more importantly, to show that data in a meaningful and easily interpretable way. The page is divided into two sections:

- **Overview:** once students have been assigned to clusters, it is possible to build a Sankey diagram showing the flow between clusters and how the number of students in each cluster evolved over time (Figure 5.2). Because professors can run the profiler multiple times throughout the semester, this diagram is a visual representation of how students are performing in the course at specific points in time, which provides useful feedback for professors. This diagram was implemented using the Highcharts library [58] for Javascript.

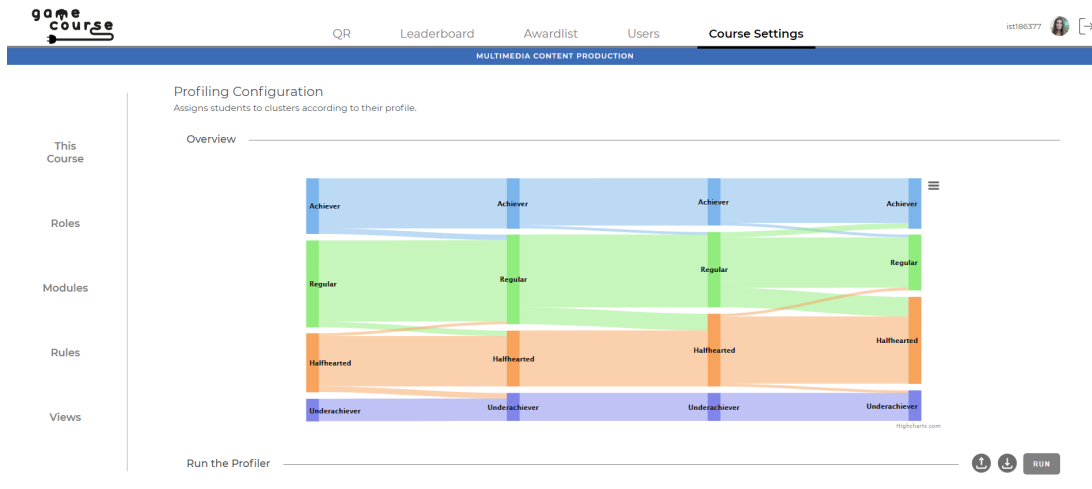
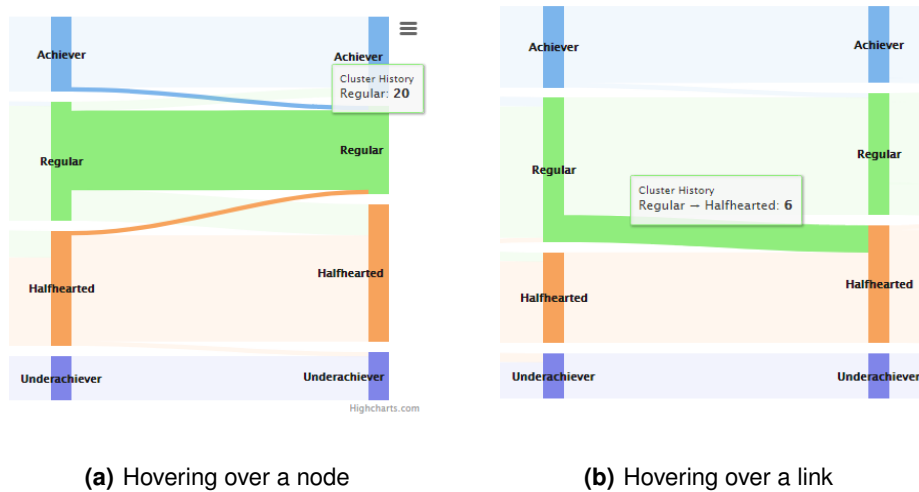


Figure 5.2: Module Profiling's configuration page.

When hovering over a node, the exact number of students in the cluster is shown and the origin of those students is highlighted (Figure 5.3(a)). If you hover over a link between two nodes, the UI will show the number of students that transitioned from the one node/cluster to the other (Figure 5.3(b)). This interaction provides professors important information regarding student's evolution since they can get a better understanding of exactly how many students improved or worsened their performance.



(a) Hovering over a node

(b) Hovering over a link

Figure 5.3: Interaction with Sankey diagram.

- **Tools and Results:** contains the tools and settings to run the profiler (Section 5.4.2) and the predictor (Section 5.4.3), as well as a table showing the cluster history for each active user, which can be sorted by each of the columns. These elements, along with the buttons for importing and

exporting cluster data, can be seen in Figure 5.4.

Run the Profiler ⬇ ⬇ RUN

Number of clusters: PREDICT Minimum cluster size:

Last run: 2021-05-14 18:31:42
Status: not running
Profiling Results:

↑ ↓ STUDENT	↑ ↓ 2021-05-04 15:10:08	↑ ↓ 2021-05-14 18:31:42
Afonso Luis	Regular	Regular
Alexandre Chicharo	Regular	Regular
Alexandre Manso	Halfhearted	Halfhearted
Ana Silva	Underachiever	Underachiever
Andreia Matos	Halfhearted	Halfhearted
André Araújo	Underachiever	Underachiever

Figure 5.4: Partial interface of the Profiling module's configuration page.

5.4.2 Running the Profiler

Recalling the implementation of the profiling script discussed in Section 5.1, besides the course ID it receives, as input, the number of clusters and the minimum cluster size. Originally, we had made both of these arguments hard-coded values, with both being equal to four. However, although four is the recommended number of clusters for MCP, the profiler may be used, in the future, for other courses in which the optimal number is not four, or professors may decide they want to create another number of clusters. For that reason, we turned the number of clusters and the minimum cluster size into passable arguments for the script. The professor can choose the number of clusters and the minimum cluster size in the designated input spaces shown in Figure 5.4. The default values that appear on the UI for both of these fields is **four**, because these were the values determined by Nabizadeh et al. for MCP [45].

The time complexity of the machine learning methods, together with the amount of data required to get accurate results made the execution time for the profiler be too long for the API call to the profiler to be synchronous (i.e., to block code execution until the API call returns). For this reason, the API call to run the profiler makes the profiler start executing in the background and immediately returns so that the UI doesn't freeze waiting for a response. The status will indicate that the profiler is running (Figure 5.5), and the "Run" button is replaced by a "Refresh" button to check if results are ready for viewing (Figure 5.6). Professors can exit the page and even exit the system safely because, once the results are ready, they are saved into a file identified by the course ID, as explained in Section 5.1. For this reason, it is possible to have the profiler running for multiple courses at the same time (since they execute as a background process) without results getting mixed-up, as the profiling module will only read the results from the file identified by the right course ID.

Once results are ready, a new column is added to the history table, as shown in Figure 5.7. The user

Last run: 2021-05-14 18:31:42

Status: running



Figure 5.5: Status label when the profiler is running. Figure 5.6: Set of buttons when the profiler is running.

can then choose, for each student, between assigning the cluster determined by the profiler or assigning any of the other available clusters (Figure 5.8). Users can cancel (deletes the results), submit (assigns students to the chosen clusters), and save (saves chosen clusters to an auxiliary database table without submitting) these results. Until the user either cancels or submits, results aren't deleted, and it is not possible to re-run the profiler.

Run the Profiler CANCEL SAVE COMMIT

Number of clusters: PREDICT Minimum cluster size:

Last run: 2021-05-20 09:21:15
Status: not running
Profiling Results:

STUDENT	2021-05-04 15:10:08	2021-05-14 18:31:42	AFTER
Afonso Luis	Regular	Regular	→ Regular
Alexandre Chicharo	Regular	Regular	→ Achiever
Alexandre Manso	Halfhearted	Halfhearted	→ Regular
Ana Silva	Underachiever	Underachiever	→ Halfhearted
Andreia Matos	Halfhearted	Halfhearted	→ Regular

Figure 5.7: History table with results from the profiler.

2021-05-14 18:31:42 AFTER

Regular	→	Regular
Regular	→	Achiever
Halfhearted	→	Regular

Figure 5.8: Cluster options.

5.4.3 Running the Predictor

Because there are multiple methods for determining the optimal number of clusters for K-Means, we implemented the two most well-known, as explained in Section 5.2: the elbow method and the silhouette method. Since these two approaches are available, we give professors the option to choose the method they want to use (Figure 5.9).

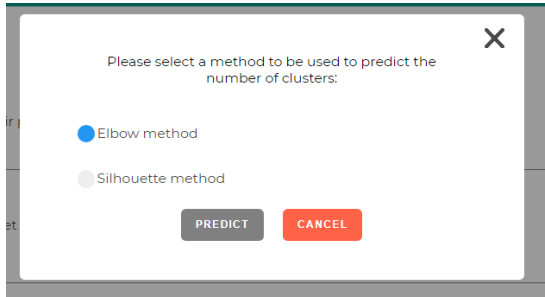


Figure 5.9: Predictor method selection.



Figure 5.10: Refresh button when the predictor is running.

Like the approach we took for the profiler, the predictor runs as a background process so that, in the meantime, the user can freely exit or interact with the system. Likewise, the predictor can be running for multiple courses at the same time. The output is saved in a file once it's ready, but, while the prediction is underway, the user is not able to re-run the predictor and can only use the "Refresh" button to check if the results are ready (Figure 5.10).

Once the best number of clusters is determined, the user is informed of the result and has to decide if the advised number of clusters should be used for the next time the profiler is ran (Figure 5.11). If so, the value on the input box for the number of clusters gets replaced.

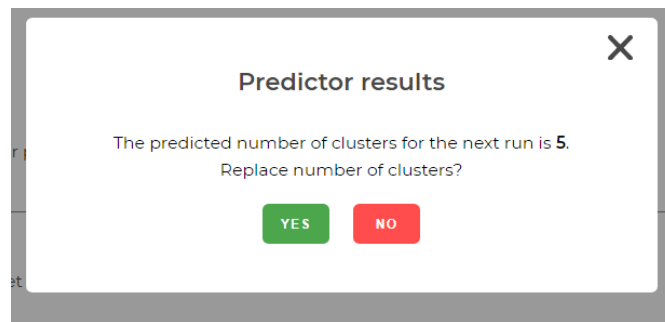


Figure 5.11: Predictor results modal.

5.5 Towards differentiated leaderboards

As part of our work, we wanted to explore what should be shown differently to different types of students, and, more specifically what to show to each type. For the mixed feelings it may cause on students, we decided to explore **leaderboards**. As a game design element, a leaderboard consists of "a visual display that ranks players according to their accomplishments" [59]. With leaderboards being one of the most used game design elements in Gamification, they have been the subject of numerous studies over the years. Mixed results on how effective leaderboards are at motivating people have led to trying to understand how different people are affected by this game design element, and how different types of

leaderboards may appeal to different people.

5.5.1 Leagues

One of the ideas that has been explored is the idea of leagues/tournaments. These are usually made up of people from the same geographical location as the user (e.g. same city, same country). Since GameCourse is, at the moment, only used for courses in which the majority of the students are from Portugal, the idea of leagues could be adapted to create a leaderboard showing only the peers from the same major. The students are often more familiar with other students from their major and may get more enjoyment and motivation from competing against them, with studies supporting that's the effect of competing against people with whom they were familiar, such as friends or close colleagues [60].

5.5.2 Teams

The idea of having a team-based leaderboards has been explored and has shown good results in motivating users when compared to the effect of player-based (individual) leaderboards [61]. This concept could also be incorporated into GameCourse, although it may not be applicable to every course. In the future, a new module (e.g., module "Teams") could be created to allow professors to create teams of students, to then provide students with a team-based leaderboard. To motivate students even further, team-based rewards, like a badge for the best team, could be created.

It is important to note, however, that team-based leaderboards shouldn't be used by themselves, but rather as one of the available leaderboards, as team-based leaderboards may take away the opportunity to receive value from others on own actions, which may "neglect some users' need for status and recognition" [61].

5.5.3 No-disincentive Leaderboards

It has been concluded that users get different levels of satisfaction depending on the position in which they are placed on the leaderbeard, with optimal positions being the second, fourth, and just above the bottom-three positions (e.g., seventh position if ten users are presented) [62]. This is what originally prompted us to explore different leaderboard designs. As explained in Section 3.2, students belonging to different clusters tend to occupy certain leaderboard positions, like the fact that Achievers usually place in the top positions and Underachievers are typically in the bottom positions.

There are two main types of leaderboards. The leaderboard used in GameCourse is considered an **infinite (or absolute) leaderboard**, since it displays all users and their scores [63]. Figure 5.12 shows the current leaderboard design, although seeing the positions below sixth requires scrolling down. The

second type of leaderboard is the **no-disincentive (or relative) leaderboard**, where “users only see their rank as compared with the users ranked below and above them” [63].

#	Photo	Num	Name	Major	Experience	Level	Badges
1		10758	Vasco Pinho	MEIC-T	20035 XP	20 - Science God	37 out of 66
2		10765	Carlos Soares	MEIC-A	19605 XP	19 - Professor 395 for L20 at 20000 XP	42 out of 66
3		10754	Miguel Gonçalves	MEIC-A	19460 XP	19 - Professor 540 for L20 at 20000 XP	36 out of 66
4		10767	Catarina Rodrigues	LEIC-T	19410 XP	19 - Professor 590 for L20 at 20000 XP	35 out of 66
5		10763	Inês Alves	LEIC-A	19240 XP	19 - Professor 760 for L20 at 20000 XP	32 out of 66
6		10762	Daniel Santana	MEIC-A	18765 XP	18 - Savior of Mankind 235 for L19 at 19000 XP	32 out of 66

Figure 5.12: GameCourse's infinite leaderboard.

At first, the concept of showing things differently to different people made us try to come up with a different leaderboard design for each cluster we had in MCP. However, we reached the conclusion that, for individual leaderboards, these two designs would suffice. The no-disincentive would be much more suitable for students on the bottom positions (Halfhearted students and Underachievers), and the infinite leaderboard is already suitable for the students in the top positions (Achievers). The reason why we don't advise showing a no-disincentive leaderboard to every student is because students in the top positions may draw more motivation from seeing exactly how many students are behind them and how close they are to the top.

Once we decided to provide the option to have no-disincentive leaderboards in GameCourse, we needed to decide the best way to do this. The student viewing the leaderboard must appear right in the middle of the leaderboard [64], but we had to decide how many students to show above and below them. We based our decision on the conclusions drawn by Sun et al. [62]. Since the levels of satisfaction are higher for users placing second, fourth, or just above the bottom-three positions, we decided that the best option would be to show the **three students above**, and **three students below** the viewing student. With this approach, the user is visually always in the fourth position, as well as in the position just above the bottom-three positions. Figure 5.13 shows the prototype we made using Whimsical [65], the same software used by Patrícia Silva [41] for the prototypes that led to the current UI.

When it came to implementing the EL functions to make this possible, we decided the best approach would be to provide a template for this new leaderboard (showing three users above and three users below) when the **Leaderboard** module is enabled, while still keeping the number of neighbors customizable by the user, if they wish to change it. For that, we implemented an EL function - **getKNeighbors** - ,

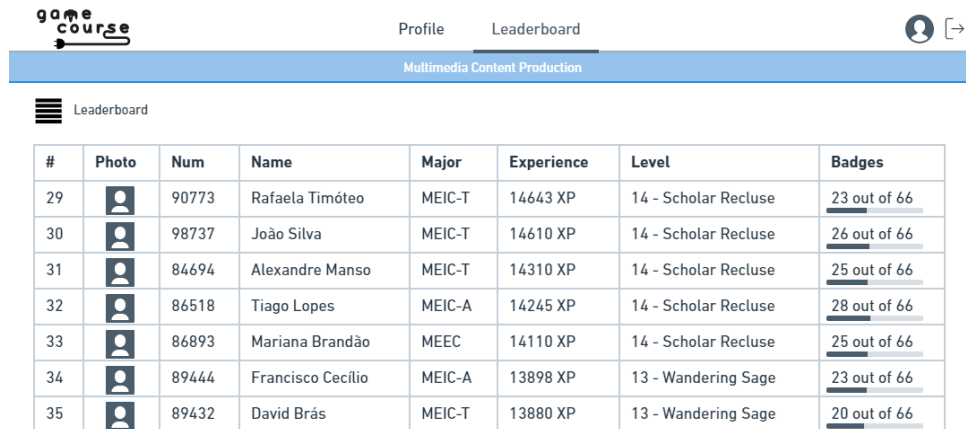


Figure 5.13: Prototype of the relative leaderboard from the perspective of user “Tiago Lopes”.

made available by enabling the Views module, that receives a collection of items, a target, and k as the number of neighbors, and returns a collection containing, in this order, the k items before the target, the target, and the k items after the target. The function was implemented in such a way that it can be used with any collection of items, not just users, to keep it versatile. To each item in the resulting collection, we add the original position, since we don't want to mislead the students into thinking that they are in a position they aren't in, hence why in Figure 5.13 the positions start at twenty-nine. Even though we show the student in fourth, they must know their actual position. The resulting leaderboard is shown in Figure 5.14.

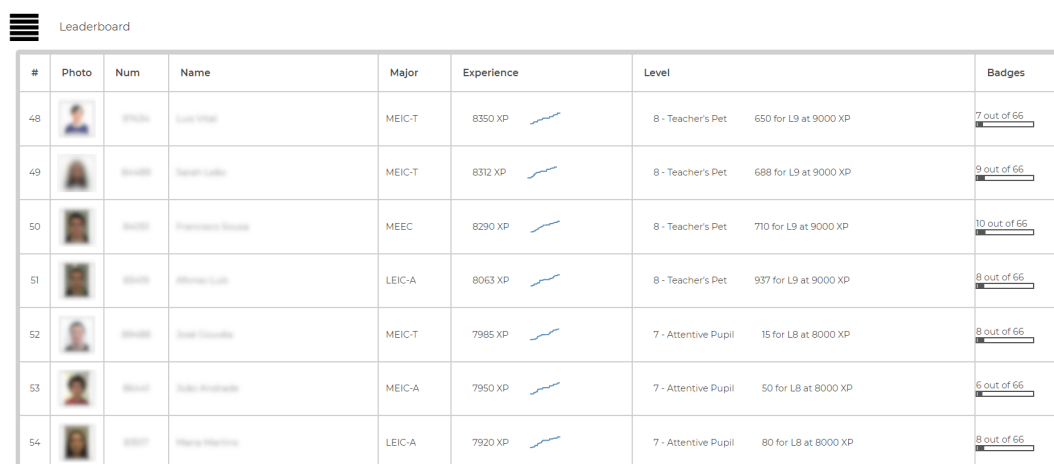


Figure 5.14: New no-disincentive leaderboard.

5.5.4 Micro Leaderboards

When it comes to leaderboard content, we can distinguish between **macro leaderboards** and **micro leaderboards** [66]. Macro leaderboards contain multiple (if not all) the elements used to measure student's performance, while micro leaderboards contain a small subsection, or even just one of those elements. As an example, GameCourse's original leaderboard (Figure 5.12) would be considered a macro leaderboard, while a micro leaderboard could be, for example, one dedicated exclusively to the number of badges.

But if we already have a macro leaderboard in GameCourse, what would be the benefits of having micro leaderboards? We've already established that students' position on the leaderboar influence their level of satisfaction. By having multiple micro leaderboards, it is more likely that a student will be among the upper positions in at least one of them [66]. Park et al. [66] made some recommendations for designing macro and micro leaderboards. First, macro and micro leaderboards should be used together, so, if micro leaderboards were to be implemented in GameCourse, the macro leaderboard should still be available. Second, all the elements used to measure students' performance should be incorporated into a micro leaderboard. Currently, the EL already allows creating micro leaderboards for badges, skills, quizzes, labs, assignments, and presentation grade, which covers all elements for both MCP and UIS.

6

Evaluation

Contents

6.1	UIS Case Study	69
6.2	Benchmark Testing	72
6.3	Automated Unit and Integration Testing	75
6.4	Discussion	77

In this chapter, we present the evaluation performed over the GameCourse system, more specifically its flexibility to adapt to a course other than MCP, its performance, and its correctness. We start by describing a case study of using GameCourse to gamify the UIS course. Then, we describe the load, unit, and integration tests performed, and we close the chapter with a thorough analysis and discussion of our results.

6.1 UIS Case Study

The GameCourse system has been used in the course of MCP over the years, showing promising results. Over those years, the course has suffered changes according to students' feedback. However, the changes were never too drastic, so the system, which was already prepared to be used in the course, was never tested on how far it could be used for courses that differ significantly from MCP in terms of evaluation and game elements. We decided to challenge the system by configuring a new course in GameCourse to be used during a semester of Usability and Information Systems (UIS), a course from the master's in Information and Enterprise Systems which is a joint degree between Instituto Superior Técnico (IST) and Universidade Aberta (UAb). The course lasts for seven weeks, and each week there's **a challenge, a quiz, a self-evaluation quiz** (which doesn't count towards the final grade), **a video**, and **an assignment**. During the semester students can ask questions in a questions forum, and other students can answer them to help their colleagues. UIS is an online course in which all submissions and interactions are performed in Moodle. We started by meeting with the professors to gather the requirements for the course and to know which game elements they wished to use. The elements that would better fit this course were the leaderboard, profile pages for students, and badges. Other rules could be created to reward students for their participation in Moodle. The differences between MCP and UIS are compiled in Table 6.1.

	MCP	UIS
Leaderboard	Yes	Yes
Profile	Yes	Yes
Badges	Yes	Yes
Skills	Yes	No
QR	Yes	No
Data Sources	Google Sheets, Moodle, Class Check	Moodle
User Login	Fénix	Fénix, Google

Table 6.1: Comparison between MCP's and UIS's configuration.

We found our first hurdle when we started discussing user login. GameCourse supports four types of login: Fénix, Google, LinkedIn, and Facebook. Fénix is an official repository for IST's academic information. Both the professors that teach at IST and the students enrolled in this university are provided

with accounts to enter the Fénix system. Due to the way the plugin to get data from Moodle was implemented, users must use the same login in Moodle that they use in GameCourse, in order to match the Moodle user to the GameCourse user. Because this is a joint course between UAb and IST and, thus, not all students have Fénix accounts, the professors decided they would ask students to log into both platforms using a Google account, while the professors would log in with their Fénix accounts.

When it came to the Moodle records that were relevant to the course (Table 6.2), we realized that we were missing the ability to fetch assignment grades. This had never been implemented because there was never a need for it, but now that there was, we implemented this feature as described in Section 4.4. The professors also intended to have a badge for students who watched the course’s videos, so we had to be able to fetch the logs for video visualizations. Since these videos were posted on Youtube and the Uniform Resource Locators (URLs) were embedded in Moodle forums, the action of visualizing these videos was represented in Moodle’s database as accessing an URL. For that reason, we asked the professors to name the videos in Moodle starting with a tag (e.g., “[Video]”) to be able to distinguish video URLs from other URLs. GameCourse’s Moodle plugin was already able to get these records from Moodle, but we had to fix it because the name of the URL wasn’t being registered in the participations. This detail was crucial to create the aforementioned badge and avoid rewarding a student twice for watching the same video. These improvements mean that, from now on, professors can implement innovative badges and rules involving URLs and assignments, for any course in GameCourse.

	MCP	UIS
Quiz Grades	Yes	Yes
Forum Posts	Yes	Yes
PowerPoint Visualizations	Yes	Yes
Peer Grades	Yes	No
Assignment Grades	No	Yes
Video Visualizations	No	Yes

Table 6.2: Comparison between MCP’s and UIS’s required Moodle data.

When it came to badges, the professors looked at the ones we had for MCP for ideas. They decided to create some similar ones for UIS and had no problem in creating them. Additionally, they were able to create a couple of new badges that had never been created for a course before. The second significant hurdle appeared when creating the rules for the course. The rules for the badges that were similar to the ones from MCP could be adapted from the rules for those badges in MCP, only requiring slight changes to the rules to reflect, for example, UIS’s Moodle forum names. However, the professors had some rewards in mind that had never been used for MCP, and we also realized that the logic behind awarding rewards was too tailored towards MCP. There was a rule system function called **award_grade** that was used to award quiz, presentation, and lab grades. This function wasn’t flexible enough to be used to award all the grades in UIS. Firstly, it didn’t work for assignment grades. Secondly, it didn’t allow

to convert grades into XP, which meant that the grade that came from Moodle would be the amount of XP that the student would receive. Because we didn't want to make an already-complex function even more complex, we introduced three new functions.

The **award_assignment_grade** function receives a student ID, a list of assignment participations, the amount of XP that each assignment is worth, and the maximum grade possible for the assignments. This function then iterates over the assignment participations and calculates and awards students the XP they deserve. It is also able to deal with grade updates.

Even though the **award_grade** function could already be used to award quiz grades, it was not flexible enough to define the equivalence between a grade and the XP that should be awarded. For that reason, we created the **award_quiz_grades** function, which, likewise, receives a student ID, that student's quiz grades from Moodle, the amount of XP that each quiz is worth, and the maximum grade possible. Optionally, a string can be provided containing the name of the quizzes to ignore. This was necessary because in both MCP and UIS there were training quizzes whose grade should not be awarded in GameCourse.

The last function we implemented in the rule system was **award_post_grades**. This function was implemented because there wasn't a way to award the grade from a post. It receives a student ID, that student's post grades from Moodle, the amount of XP that each post is worth, the maximum grade possible, and, optionally, the name of a specific forum where students can make posts to earn XP. It then calculates and awards students XP based on the grades given to the student's posts in Moodle.

In the future, we should identify the similarities between these functions and try to implement a function that can deal with any kind of award, without making it too complex and without reducing the flexibility that we now offer with these new functions.

In addition to these functions, we added three EL functions to be used in rules and views. Function **getXPByType** receives a student ID and a string indicating a type of award (e.g., "quiz", "lab"), and returns the total XP that the student has received from that type of award. There were already functions to get the total XP received from specific types of awards, like quizzes and badges, but there was a need for a function that could get the total XP for any type of award. To implement the rule for the video visualization badge, we had to implement an EL function, **getVideoViews** that returns a collection of unique video URLs that a certain user has accessed. After implementing this function, we took the opportunity to make this function more flexible so that it can be used for any type of URL. The function is now called **getLinkViews**. It receives, as input, a user ID and a string that the URL must contain (in the case of UIS, to get video visualizations, the string is "[Video]"). It now returns a collection of unique URLs containing the string provided that a certain user has accessed. Finally, the professors wanted to award students XP for uploading a profile picture in GameCourse, so, in order to write the rule for that reward, we implemented function **hasPicture**, which receives a student ID and returns true or false

whether the user has a profile picture in the system or not.

While setting up the course together with the professors, we were able to collect their feedback on the system, as well as their suggestions on how we could improve the current version. These remarks will be addressed in Section 6.4.

6.2 Benchmark Testing

To assess if our goal of improving the system's performance as described in Section 4.7 had been accomplished, we measured the time that it takes our version to import different numbers of levels and how long it now takes to load the leaderboard page.

Starting with the time complexity of importing XP levels, we measured the time it took for both the old and the new version to import different numbers of XP levels. Twenty measurements were made for each version when importing twenty, forty, sixty, eighty, and one-hundred XP levels, and we then calculated the mean time to be able to visualize how the number of levels impacted the execution time of each version (Figure 6.1).

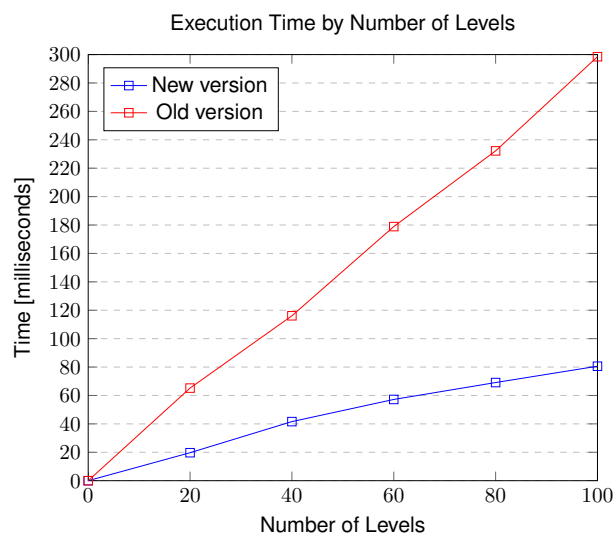


Figure 6.1: Time spent importing levels.

Figure 6.1 compares the results of each version. We can observe that the new version is consistently faster than the old version, which appears to have a linear time since the time appears to grow in direct proportion to the size of the input data. In this case, we tested importing up to one-hundred levels, even though MCP and UIS only use twenty levels, because we wanted to make sure that our solution would be better for more than twenty levels. Although the time difference gets higher as the number of levels increases, our solution is already significantly better for twenty levels, taking less than half the time that the old version took. We managed to reduce the execution time by replacing single-row insertions of

each new level by a single query that inserts all of the new levels, as explained in Section 4.7.

To evaluate if we were successful at reducing the time it takes to load the leaderboard page, we used the data from the end of the semester of MCP and created a leaderboard page. We then made measurements of how long it took each version, running on the virtual machine that hosts GameCourse, to load the same leaderboard page with the same data. For each version, we collected twenty samples using the hard reload and another twenty samples using the “normal” reload. We then used those samples to calculate the mean, median, standard deviation, and determined the percentage difference between the old version and the new version. The results are shown in Table 6.3.

	Old Version		New Version			
	Hard Reload	Normal Reload	Hard Reload	Diff	Normal Reload	Diff
Mean	13,5081s	12,7367s	9,6204s	-29%	9,1471s	-28%
Median	13,55s	12,66s	9,55s	-30%	9,36s	-26%
Std. Deviation	0,7517	1,0589	0,7639		0,5523	

Table 6.3: Leaderboard load time on different versions of GameCourse.

After collecting the samples, we tested each set of samples for normality using the **Shapiro-Wilk test** [67]. From the results of those tests, we concluded that only the samples from the old version followed a normal distribution, thus we performed the **Wilcoxon Signed-Rank test** [68] to compare the two versions. The tests comparing hard reload times revealed that there is a significant difference in load time between the two versions ($p=0.000029 < 0.05$). The result for the test comparing the “normal” reload times revealed a significant difference in load time between the two versions ($p=0.000037 < 0.05$) as well. We can, therefore, conclude that the new version of GameCourse has a faster load time for the leaderboard page than the older version of the system.

Given the scope of our work, it was important to test the profiler script that was described in Section 5.1. We decided to test the performance of the profiler script to find out if we had introduced any unnecessary complexity. The approach that was taken consisted in collecting time samples of how long the profiler took to finish executing and provide an output. Those time samples were collected for different numbers of students, for different numbers of attributes, and for different numbers of participations, since we wanted to assess how each of these three variables impacted the execution time. For each of these three tests, we ran the profiler twenty times for each value of the variable under test, with both the number of clusters and the minimum cluster size set to four. The values presented below are the average time of each set of samples.

Figure 6.2 shows how the execution time varied according to the number of students. We were careful to maintain the number of attributes, since it was inevitable to increase the number of participations and awards due to the profiler ignoring students who don't have participations and awards. Our goal was to simulate the data at the end of the semester for different numbers of students. We can conclude, as expected, that increasing the number of students does increase the execution time.

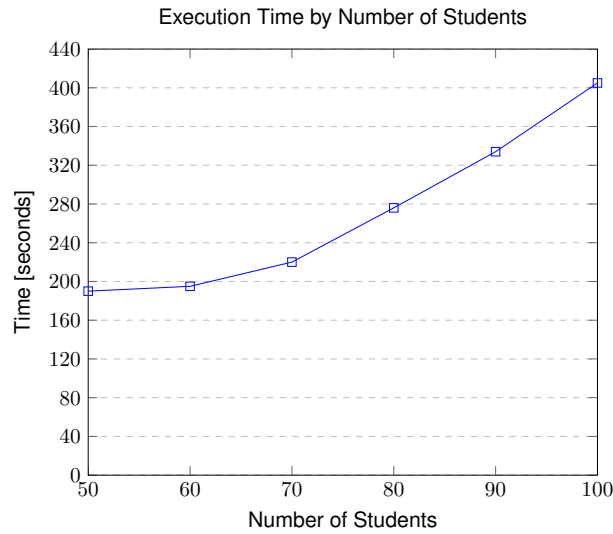


Figure 6.2: Profiler execution time by number of students.

According to the documentation for the *scikit-learn* library that was used, k-means' average complexity is given by $\mathcal{O}(k \cdot n \cdot T)$, where k is the number of clusters, n is the number of samples, and T is the number of iterations. However, according to David et al. [69], the worst-case complexity is given by $\mathcal{O}(n^{(k+2/p)})$, where k is the number of clusters, n is the number of samples, and p is the number of features. In our case, the number of features used in k-means is always one (i.e, the accumulated XP), the number of samples corresponds to the number of students, and the number of clusters was kept at four. With this information, we can conclude that the values correspond to what would be expected, and no additional complexity has been introduced when it comes to the manipulation of students, since the time complexity never goes beyond the worst-case complexity (i.e., $\mathcal{O}(n^6)$).

After collecting the time samples for different numbers of students, we set out to test how the number of participations influenced the execution time of the profiler. Figure 6.3 shows the results of this experiment, and we can observe that the time increases in a seemingly linear way, as the number of participations increases. Because participations aren't the only attributes taken into account, as explained in Section 5.1, we were interested in exploring the relation between the number of attributes and the execution time. Because the number of attributes is determined by the number of participations, awards, and days, we varied these values while maintaining the values as realistic as possible. This was achieved by using data from MCP at the end of the semester and deleting the data from entire days until we got the desired number of attributes to conduct each round of tests. The results, presented in Figure 6.4, show that an increase in the number of attributes does lead to a longer execution time. There seems to be a higher increase in time when we go from having 1000 attributes to having 1100 attributes.

Both the number of participations and the number attributes appear to have a similar impact on the execution time, which was to be expected since the participations are one of the components that

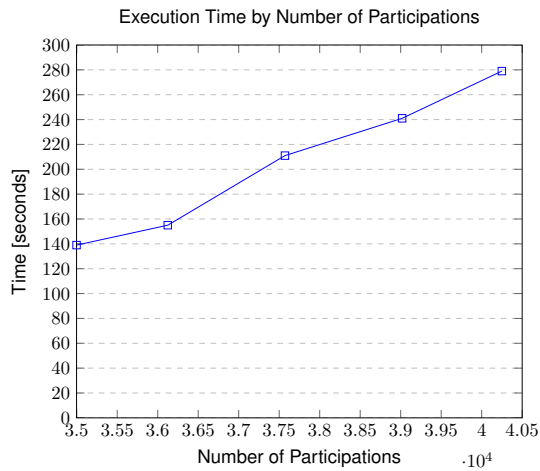


Figure 6.3: Profiler execution time by number of participations.

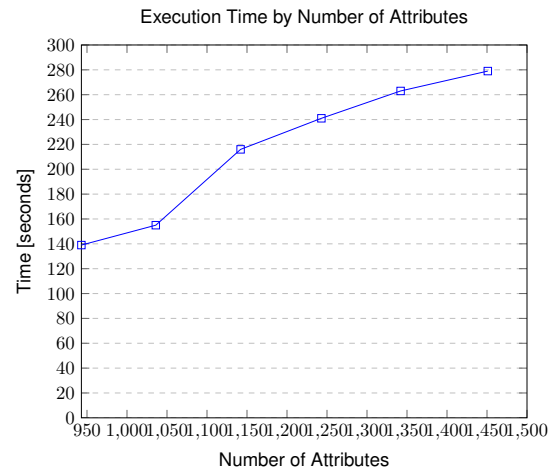


Figure 6.4: Profiler execution time by number of attributes.

make up the attributes, together with the awards and the XP accumulated on each day since the first award or participation. The time complexity for the Boruta algorithm $\mathcal{O}(P \cdot N)$, where P and N are, respectively, the number of attributes and objects (in this case, students) [70]. Solé et al. [71] state that the computational complexity for a Random Forest of size T and maximum depth D is $\mathcal{O}(T \cdot D)$. Because the maximum depth is always the same, and we maintained the number of students, we expected the time complexity to be linear. The slope resulting from only varying the number of participations is closest to the slope of a linear time complexity than the slope resulting from varying the number of attributes. Nevertheless, the slope resulting from varying the number of attributes approximates more to that of a linear time complexity, or even a logarithmic time complexity, than that of a super-linear time complexity (e.g., $\mathcal{O}(n^2)$). We can conclude that the observed time complexities are within our expectations.

6.3 Automated Unit and Integration Testing

Part of our evaluation consisted in conducting unit and integration testing. We maintained and adapted the existing testing script as the code evolved, but we ultimately decided that we needed to create new unit and integration test suites for a more thorough evaluation.

6.3.1 Former Testing Script

A test script had previously been implemented for GameCourse [4]. This script covered twenty-six tasks, including login picture creation, plugin configuration, and importing and exporting users, course users, and courses. Whenever we made changes to any function covered by the tests, we updated the script and made sure that all tests passed (Figure 6.5).

Automated Test Script

Group	Test	Score	Coverage
Login Picture	Success: Photo was created.	100% (1/1)	100% (1/1)
Fénix Import	Success: Users uploaded.	100% (4/4)	100% (4/4)
	Success: Course Users uploaded.		
	Success: Users' authentication uploaded.		
	Success: Users updated.		
Plugins	Success: Moodle variables were set.	100% (3/3)	100% (3/3)
	Success: Class Check variables were set.		
	Success: Google Sheets variables were set.		
Dictionary	Success: Library created.	100% (9/9)	100% (9/9)
	Success: Library updated.		
	Success: Library deleted.		
	Success: Function created.		
	Success: Function created.		
	Success: Function created.		
	Success: Variable created.		
	Success: Variable updated.		
Import/Export Users	Success: Users exported.	100% (3/3)	100% (3/3)
	Success: Users imported - created and updated (not replaced).		
	Success: Users imported - updated (and replaced).		
Import/Export Course Users	Success: Course Users exported.	100% (3/3)	100% (3/3)
	Success: Course Users imported - created and updated (not replaced).		
	Success: Course Users imported - updated (and replaced).		
Import/Export Courses	Success: Courses exported.	100% (3/3)	100% (3/3)
	Success: Courses imported - updated (and replaced).		
	Success: Courses imported - updated (and replaced).		
Total		100% (26/26)	100% (26/26)

Figure 6.5: Old test script test results.

It is important to note that these tests required the user to first set up a course with specific characteristics. This means that if the user didn't setup the course correctly, tests would fail. Furthermore, test failures could cause other tests to fail because the database tables were not emptied between test cases. Also, each task was only tested once, only covering a success case for each task. This means that boundary values (e.g., values of -1, 0, and 1 for an argument that must be a positive integer), where bugs may have been hiding, were left untested, as well as situations where functions are expected to throw exceptions. Lastly, the tests involving exporting data weren't checking the output thoroughly enough. They only asserted that a file had been created and that that file wasn't empty, without checking the actual content of the exported file.

6.3.2 New Unit and Integration Test Suites

Because the existing tests were neither truly unit tests, nor automatic, and because they left some key functionalities completely untested, we came up with our own test suites. We created a “tests” folder where we put all of these new test scripts. For implementing the test suites, we used the most popular testing framework for PHP: **PHPUnit** [72]. Unlike the previous test suite, we start tests with completely empty database tables and the test classes have the responsibility of both setting up before tests and cleaning up afterwards so that every test starts with a clean environment. Additionally, these tests are completely automatic because no action is required from the user conducting tests. With one simple command in the terminal, it is possible to run a single test case, full test suites, or even the whole “tests” folder in one go (as shown in Figure 6.6).

```
PS C:\Users\anago> phpunit C:\xampp\htdocs\gamecourse\tests\  
PHPUnit 9.5.8 by Sebastian Bergmann and contributors.  
  
..... 63 / 463 ( 13%)  
..... 126 / 463 ( 27%)  
..... 189 / 463 ( 40%)  
..... 252 / 463 ( 54%)  
..... 315 / 463 ( 68%)  
..... 378 / 463 ( 81%)  
..... 441 / 463 ( 95%)  
..... 463 / 463 (100%)  
  
Time: 00:18.034, Memory: 30.00 MB  
OK (463 tests, 1302 assertions)
```

Figure 6.6: Result of running every test in the “tests” folder.

Because we created these test suites from scratch and the GameCourse system already had considerable complexity, we unfortunately couldn’t implement the tests for every single function in the system, but we did cover a lot of ground and found many bugs as a result. Every function created in the scope of our work is fully tested, as well as the functions we altered. This amounts to a total of 463 tests that run in under 19 seconds. These tests cover both unit testing and integration testing. For integration testing we use a bottom-up integration approach, meaning that we begin by testing the components having the fewest dependencies and follow that order throughout.

6.4 Discussion

The UIS case study was useful for understanding the weaknesses of our system when it came to harboring courses other than MCP. All the difficulties were successfully overcome, some requiring the implementation of new features. This shows us that, although more studies need to be done with different courses to assess the necessities that GameCourse doesn’t yet cover, the system isn’t that ill prepared for receiving different courses, and the outcome was positive. The features that were implemented can

now be used by any course in GameCourse, meaning that the problems that led to their implementation will no longer arise. Therefore, we can say we've made the system more flexible, especially in the rule system side of the system, since none of the solutions found are hard coded for the new course. One of the most important outcomes of this experiment was the feedback that we received from the UIS professors. Because there was the inconvenience of asking students to log in with their Google accounts, requiring students to have a Google account, the professors suggested exploring the implementation of logging in into GameCourse through Moodle. They also suggested that it would be useful to make GameCourse into a multilingual platform, where users would be able to select their preferred language and the content of the pages would adapt to this selection. When it came to the process of setting up the course, the professors recommended that the system should allow them to export only a selection of elements such as badges and rules, since they were only interested in using a subset of MCP's badges and rules in their course and saw themselves having to export all badges/rules and manually deleting the unwanted elements before importing them into UIS.

We aimed at improving the system's performance as described in section Section 4.7. To that effect, we changed the function that imports levels to do batch inserts of new levels instead of performing a single insert for each new level. As presented in Section 6.2, the new version is significantly faster than the older version, proving that our goal for these functions was achieved. The older version appears to have a linear time complexity, which can be explained by the number of insertions growing proportionally to the number of levels being imported. In contrast, new version appears to have a sub-linear time complexity, and although it is faster than the older version for any of the tested inputs, the gap between the two versions increases as the number of levels increases. The results were already to be expected, since the documentation for the database states that INSERT statements that insert several rows at a time are considerably faster than using separate single-row INSERT statements [49]. We can conclude that this replacement made the current system more efficient.

The second inefficiency problem we tried to tackle was related to the time it took to load pages. As the test results show, the changes we made have positively affected the time it takes to load the leaderboard, making it faster than the older version. We can confidently say we've achieved our goal of reducing the time it took to load the leaderboard page, but there is still work to be done. It still takes a long time to load pages in general, and by profiling the execution we've identified two main problems: the logic behind building the navbar, and the logic behind rendering the views. The second problem is much more substantial than the first one since the navbar is only built when you first enter a course. The views that make up pages must be rendered every time a user accesses them, making the process take an excessive amount of time. In the future, it could be beneficial to review the logic behind how views are built.

When it came to the performance of our profiler, we were already expecting long execution times,

since this was the factor that forced us to make the script run as a background process. These benchmark tests, however, had a positive outcome since the way that the number of students, participations, and attributes each influenced the execution time were within our expectations.

The unit and integration tests performed showed that our system works correctly, and the new test suite now brings more confidence to the correctness of the system since it achieves a much higher coverage than the previous test suite. Since GameCourse is already an extent and complex system, it is crucial to maintain these test scripts whenever there is a change in implementation, and to write tests for new features during their development so that testing can be done concurrently with the implementation.

7

Conclusion

Contents

7.1 Future Work	83
-----------------------	----

We stated out this project by improving the latest version of GameCourse so that we could proceed to reaching our goal of transforming it into a system with adaptive capabilities. By fixing the problems that were identified in the GameCourse system, we wanted not only to achieve a system that's more flexible and reliable, but most of all we intended to make sure that we had a strong base to build the adaptive capabilities on top of.

With those goals in mind, we worked on improving the system's performance and correctness, as well as its adaptability to other courses. We also came up with solutions for implementation details that, although working correctly, made the system more complex and less modifiable. An example of a problematic implementation detail was having different modules sharing the same database table. Once we overcame these initial hurdles, we were able to focus on taking GameCourse to the next level: becoming an adaptive gamification environment. The key to achieving this was creating a profiling script – the profiler –, so that the system could make an informed and effective adaptation to its users. We also focused on providing support to the professors by advising them on the ideal number of clusters to divide students into – the predictor. This culminated in the creation of the Profiling module. Once we could profile the students, by dividing them into clusters, the next step was to start creating new versions of the existing game elements, better-fitting for the types of students who the former elements weren't ideal for. We took the first step by creating a no-disincentive leaderboard that can be shown to designated users instead of the infinite leaderboard that existed in GameCourse.

To assess whether we had achieved our goals, we performed a case study to evaluate the extent to which GameCourse could be used on courses that differ from MCP, as well as benchmark tests to evaluate the performance of the system and unit and integration tests to evaluate the correctness of the resulting implementation.

In the end, the test results allow us to conclude that we did manage to improve the system's performance and correctness. They also show that it is possible to use GameCourse for courses that differ from MCP. Most importantly, we can confidently say that we've achieved our goal of providing GameCourse the capabilities of profiling and adapting to its users.

7.1 Future Work

Our work has opened the doors to finding creative ways of designing gamified environments tailored to meet students' needs, using GameCourse. We have already taken the first step by creating a new type of leaderboard, but further investigation should be done on what to show to different students based on their cluster. Those differences should be focused on the visual game elements, such as the leaderboard and the profile, and not on game elements that directly influence the final grade such as the badges, skills, and rules, as this could lead to students finding the game unfair. We already know, from previous

studies [5, 45], some characteristics of each cluster and what sets them apart from the others, but we have yet to know what they would like to see, for example, on their profile page.

The algorithm we based our profiler on was developed in the context on MCP, but further studies should be conducted to assess how accurate it is for other courses. Other algorithms could be created and integrated into GameCourse, so that there are multiple types of profilers that professors can choose from, like they can already do for the predictor.

As far as the Profiling module configuration page is concerned, it would be helpful, in the future, to show users the progress of the profiler and the predictor when either one of them are running.

Lastly, there should be more communication between the system and the students. There already exists a module – “Notifications” module – that was no longer used nor functional in the start of our work. It was used to notify students of the awards that they had won since the last time they had been active in GameCourse. This could be the starting point to develop notifications that are targeted at different clusters of students, offering suggestions on, for example, what skill to do next, or words of encouragement, like showing how close a student is to getting a badge or placing higher on the leaderboard.

Bibliography

- [1] R. Bartle, "Virtual worlds: Why people play," *Massively Multiplayer Game Development*, vol. 2, no. 1, pp. 3–18, 01 2005.
- [2] B. Monterrat, M. Desmarais, E. Lavoué, and S. George, "A player model for adaptive gamification in learning environments," in *International conference on artificial intelligence in education*. Springer, 06 2015, pp. 297–306. [Online]. Available: https://doi.org/10.1007/978-3-319-19773-9_30
- [3] M. Altmeyer, G. F. Tondello, A. Krüger, and L. E. Nacke, "Hexarcade: Predicting hexad user types by using gameful applications," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 219–230. [Online]. Available: <https://doi.org/10.1145/3410404.3414232>
- [4] D. Lopes, "Gamecoursebeyond," Master's thesis, Instituto Superior Técnico, Lisbon, 2021.
- [5] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, "Studying student differentiation in gamified education: A long-term study," *Computers in Human Behavior*, vol. 71, pp. 550–585, 06 2017. [Online]. Available: <https://doi.org/10.1016/j.chb.2016.08.049>
- [6] R. Felder and J. Spurlin, "Applications, reliability and validity of the index of learning styles," *International Journal of Engineering Education*, vol. 21, no. 1, pp. 103–112, 01 2005.
- [7] S. Milner and A. Wildberger, "Determining appropriate uses of computers in education," *Computers & Education*, vol. 1, no. 2, pp. 117–123, 1977. [Online]. Available: [https://doi.org/10.1016/0360-1315\(77\)90006-9](https://doi.org/10.1016/0360-1315(77)90006-9)
- [8] Z. Zainuddin and C. M. Keumala, "Gamification concept without digital platforms: A strategy for parents on motivating children study at home during covid-19 pandemic," *PEDAGOGIK: Jurnal Pendidikan*, vol. 8, no. 1, pp. 156–193, 2021. [Online]. Available: <https://doi.org/10.33650/pjp.v8i1.2174>
- [9] S. Coutinho, "Is gamifying your curriculum a win or a loss?" *Technology and the Curriculum: Summer 2019*, 2019.

- [10] I. Furdu, C. Tomozei, and U. Köse, “Pros and cons gamification and gaming in classroom,” *Broad Research in Artificial Intelligence and Neuroscience*, vol. 8, pp. 56–62, 08 2017.
- [11] A. Wilson and J. McDonagh, “A gamification model to encourage positive healthcare behaviours in young people with long term conditions,” *EAI Endorsed Transactions on Serious Games*, vol. 14, p. E3, 05 2014. [Online]. Available: <https://doi.org/10.4108/sg.1.2.e3>
- [12] Z. Zhao, S. Ali Etemad, and A. Arya, “Gamification of exercise and fitness using wearable activity trackers,” in *Proceedings of the 10th International Symposium on Computer Science in Sports (ISCSS)*, P. Chung, A. Soltoggio, C. W. Dawson, Q. Meng, and M. Pain, Eds., Springer. Cham: Springer International Publishing, 11 2016, pp. 233–240. [Online]. Available: https://doi.org/10.1007/978-3-319-24560-7_30
- [13] A. Mora, E. Planas, and J. Arnedo-Moreno, “Designing game-like activities to engage adult learners in higher education,” in *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*, ser. TEEM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 755–762. [Online]. Available: <https://doi.org/10.1145/3012430.3012603>
- [14] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, “Engaging engineering students with gamification,” in *2013 5th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*. IEEE, 09 2013, pp. 24–31. [Online]. Available: <https://doi.org/10.1109/VS-GAMES.2013.6624228>
- [15] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, “Improving participation and learning with gamification,” in *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, ser. Gamification '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 10–17. [Online]. Available: <https://doi.org/10.1145/2583008.2583010>
- [16] A. Gawel, *The Influence of Gamification Activities on Learning Outcomes in Higher Education*. Bogucki Scientific Publishers, 01 2019, pp. 135–152. [Online]. Available: <https://doi.org/10.12657/9788379862801-10>
- [17] K. Robson, K. Plangger, J. Kietzmann, I. McCarthy, and L. Pitt, “Is it all a game? understanding the principles of gamification,” *Business Horizons*, vol. 58, no. 4, pp. 411–420, 04 2015. [Online]. Available: <https://doi.org/10.1016/j.bushor.2015.03.006>
- [18] E. D. Mekler, F. Brühlmann, K. Opwis, and A. N. Tuch, “Disassembling gamification: The effects of points and meaning on user motivation and performance,” in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1137–1142. [Online]. Available: <https://doi.org/10.1145/2468356.2468559>

- [19] A. Marczewski, "Gamification a little on leaderboards," Jan 2013, accessed: 2020-09-26. [Online]. Available: <https://www.gamified.uk/2013/01/21/gamification-a-little-on-leaderboards/>
- [20] L. S. Ferro, S. P. Walz, and S. Greuter, "Towards personalised, gamified systems: An investigation into game design, personality and player typologies," in *Proceedings of The 9th Australasian Conference on Interactive Entertainment: Matters of Life and Death*, ser. IE '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: <https://doi.org/10.1145/2513002.2513024>
- [21] P. Buckley and E. Doyle, "Individualising gamification: An investigation of the impact of learning styles and personality traits on the efficacy of gamification using a prediction market," *Computers & Education*, vol. 106, pp. 43–55, 2017. [Online]. Available: <https://doi.org/10.1016/j.compedu.2016.11.009>
- [22] R. Felder and L. Silverman, "Learning and teaching styles in engineering education," *Journal of Engineering Education*, vol. 78, no. 7, pp. 674–681, 01 1988.
- [23] R. McCrae and O. John, "An introduction to the five-factor model and its applications." *Journal of personality*, vol. 60, no. 2, pp. 175–215, 1992. [Online]. Available: <https://doi.org/10.1111/j.1467-6494.1992.tb00970.x>
- [24] D. Codish and G. Ravid, "Personality based gamification: How different personalities perceive gamification," *ECIS 2014 Proceedings - 22nd European Conference on Information Systems*, 01 2014.
- [25] S. D. Gosling, P. J. Rentfrow, and W. B. Swann Jr, "A very brief measure of the big-five personality domains," *Journal of Research in personality*, vol. 37, no. 6, pp. 504–528, 2003. [Online]. Available: [https://doi.org/10.1016/S0092-6566\(03\)00046-1](https://doi.org/10.1016/S0092-6566(03)00046-1)
- [26] R. Bartle, "Hearts, clubs, diamonds, spades: Players who suit muds," *Journal of MUD research*, vol. 1, no. 1, p. 19, 1996.
- [27] P. Škuta and K. Kostolányová, "Adaptive approach to the gamification in education," *DIVAI 2018*, 2018.
- [28] R. Bartle, *Designing Virtual Worlds*. New Riders, 07 2003.
- [29] E. Lavoué, B. Monterrat, M. Desmarais, and S. George, "Adaptive gamification for learning environments," *IEEE Transactions on Learning Technologies*, vol. 12, no. 1, pp. 16–28, 04 2018. [Online]. Available: <https://doi.org/10.1109/TLT.2018.2823710>
- [30] L. E. Nacke, C. Bateman, and R. L. Mandryk, "Brainhex: Preliminary results from a neurobiological gamer typology survey," in *Entertainment Computing – ICEC 2011*, J. C. Anacleto, S. Fels,

- N. Graham, B. Kapralos, M. Saif El-Nasr, and K. Stanley, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 288–293.
- [31] A. Marczewski, *User Types HEXAD*, 1st ed. CreateSpace Independent Publishing Platform, 10 2015, pp. 65–80.
- [32] G. Tondello and L. Nacke, “Validation of user preferences and effects of personalized gamification on task performance,” *Frontiers in Computer Science*, vol. 2, p. 29, 2020. [Online]. Available: <https://doi.org/10.3389/fcomp.2020.00029>
- [33] A. Knutas, J. Ikonen, D. Maggiorini, L. Ripamonti, and J. Porras, “Creating student interaction profiles for adaptive collaboration gamification design,” *International Journal of Human Capital and Information Technology Professionals*, vol. 7, no. 3, pp. 47–62, 07 2016. [Online]. Available: <https://doi.org/10.4018/IJHCITP.2016070104>
- [34] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, “Identifying student types in a gamified learning experience,” *International Journal of Game-Based Learning*, vol. 4, pp. 19–36, 10 2014. [Online]. Available: <https://doi.org/10.4018/ijgbl.2014100102>
- [35] B. Monrerrat, E. Lavoué, and S. George, “Motivation for Learning: Adaptive Gamification for Web-based Learning Environments,” in *6th International Conference on Computer Supported Education (CSEDU 2014)*, Barcelona, Spain, 04 2014, pp. 117–125. [Online]. Available: <https://hal.archives-ouvertes.fr/hal-01130791>
- [36] F. Roosta, F. Taghiyareh, and M. Mosharraf, “Personalization of gamification-elements in an e-learning environment based on learners’ motivation,” in *2016 8th International Symposium on Telecommunications (IST)*. IEEE, 09 2016, pp. 637–642. [Online]. Available: <https://doi.org/10.1007/s10639-021-10456-9>
- [37] K. Mbabu, R. Oboko, and S. Kamunya, “An adaptive gamification tool for e-learning platform,” *Open Journal for Information Technology*, vol. 2, no. 2, pp. 41–52, 12 2019. [Online]. Available: <https://doi.org/10.32591/coas.ojit.0202.03041m>
- [38] Z. Gantner, L. Drumond, C. Freudenthaler, S. Rendle, and L. Schmidt-Thieme, “Learning attribute-to-feature mappings for cold-start recommendations,” in *2010 IEEE International Conference on Data Mining*, 2010, pp. 176–185. [Online]. Available: <https://doi.org/10.1109/ICDM.2010.129>
- [39] A. Baltazar, “Smartboards,” Master’s thesis, Instituto Superior Técnico, Lisbon, 2016.
- [40] A. Dourado, “Gamecoursenext,” Master’s thesis, Instituto Superior Técnico, Lisbon, 2019.

- [41] P. Silva, “Implementing a new graphical user interface for gamecourse,” Master’s thesis, Instituto Superior Técnico, Lisbon, 2020.
- [42] I. Paiva, “Autogame,” Master’s thesis, Instituto Superior Técnico, Lisbon, 2020.
- [43] A. Nogueira, “Dynagame,” Master’s thesis, Instituto Superior Técnico, Lisbon, 2021.
- [44] D. Kuhn, C. Kim, and B. Lopuz, *Chapter 10: Automating Jobs with cron*. Berkeley, CA: Apress, 2015, pp. 267–283. [Online]. Available: https://doi.org/10.1007/978-1-4842-1254-7_10
- [45] A. H. Nabizadeh, J. Jorge, S. Gama, and D. Gonçalves, “How do students behave in a gamified course?—a ten-year study,” *IEEE Access*, vol. 9, pp. 81 008–81 031, 5 2021. [Online]. Available: <https://doi.org/10.1109/ACCESS.2021.3083238>
- [46] R. Thorndike, “Who belongs in the family?” *Psychometrika*, vol. 18, no. 4, pp. 267–276, 1953. [Online]. Available: <https://doi.org/10.1007/BF02289263>
- [47] J. Hartigan and M. Wong, “A k-means clustering algorithm,” *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979. [Online]. Available: <https://doi.org/10.2307/2346830>
- [48] A. Pillai, *Software Architecture with Python*. Packt Publishing, 04 2017, p. 16.
- [49] “Optimizing insert statements,” accessed: 2021-10-02. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/insert-optimization.html>
- [50] “Iso 8601 — date and time format,” accessed: 2021-10-23. [Online]. Available: <https://www.iso.org/iso-8601-date-and-time-format.html>
- [51] “pandas - python data analysis library,” accessed: 2021-10-23. [Online]. Available: <https://pandas.pydata.org/>
- [52] “scikit-learn: Machine learning in python,” accessed: 2021-10-04. [Online]. Available: <https://scikit-learn.org/stable/>
- [53] “imbalanced-learn documentation,” accessed: 2021-10-08. [Online]. Available: <https://imbalanced-learn.org/stable/index.html>
- [54] “Boruta 0.3,” accessed: 2021-10-08. [Online]. Available: <https://pypi.org/project/Boruta/>
- [55] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 09 2009, vol. 344. [Online]. Available: <https://doi.org/10.1002/9780470316801>
- [56] “kneed 0.7.0,” accessed: 2021-10-04. [Online]. Available: <https://pypi.org/project/kneed/#description>

- [57] J. VanderPlas, "In depth: k-means clustering," accessed: 2021-10-04. [Online]. Available: <https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html>
- [58] "Highcharts," accessed: 2021-10-02. [Online]. Available: <https://www.highcharts.com>
- [59] K. R. Christy and J. Fox, "Leaderboards in a virtual classroom: A test of stereotype threat and social comparison explanations for women's math performance," *Computers & Education*, vol. 78, pp. 66–77, 2014. [Online]. Available: <https://doi.org/10.1016/j.compedu.2014.05.005>
- [60] Y. Jia, Y. Liu, X. Yu, and S. Voids, "Designing leaderboards for gamification: Perceived differences based on user ranking, application domain, and personality traits," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1949–1960. [Online]. Available: <https://doi.org/10.1145/3025453.3025826>
- [61] C. Höllig, A. Tumasjan, and I. Welp, "The interaction of trait competitiveness and leaderboard design - an experimental analysis of effects on perceptions and usage intention," in *HICSS*, 01 2018. [Online]. Available: <https://doi.org/10.24251/HICSS.2018.146>
- [62] E. Sun, B. Jones, S. Traca, and M. W. Bos, "Leaderboard position psychology: Counterfactual thinking," in *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1217–1222. [Online]. Available: <https://doi.org/10.1145/2702613.2732732>
- [63] M. Ortiz-Rojas, K. Chiluiza, and M. Valcke, "Gamification through leaderboards: An empirical study in engineering education," *Computer Applications in Engineering Education*, vol. 27, no. 4, pp. 777–788, 2019. [Online]. Available: <https://doi.org/10.1002/cae.12116>
- [64] G. Zichermann and C. Cunningham, *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*, 1st ed. O'Reilly Media, Inc., 2011, pp. 50–51.
- [65] "Whimsical," accessed: 2021-10-12. [Online]. Available: <https://whimsical.com/>
- [66] S. Park and S. Kim, "Leaderboard design principles to enhance learning and motivation in a gamified educational environment: Development study," *JMIR Serious Games*, vol. 9, no. 2, p. e14746, 04 2021. [Online]. Available: <https://doi.org/10.2196/14746>
- [67] A. Ghasemi and S. Zahediasl, "Normality tests for statistical analysis: A guide for non-statisticians," *International Journal of Endocrinology and Metabolism*, vol. 10, no. 2, pp. 486–489, 12 2012. [Online]. Available: <https://doi.org/10.5812/ijem.3505>

- [68] D. H. Kim and Y. C. Kim, "Wilcoxon signed rank test using ranked-set sample," *Korean Journal of Computational & Applied Mathematics*, vol. 3, no. 2, pp. 235–243, 06 1996. [Online]. Available: <https://doi.org/10.1007/BF03008904>
- [69] D. Arthur and S. Vassilvitskii, "How slow is the k-means method?" in *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, ser. SCG '06. New York, NY, USA: Association for Computing Machinery, 06 2006, p. 144–153. [Online]. Available: <https://doi.org/10.1145/1137856.1137880>
- [70] M. Kursa and W. Rudnicki, "Feature selection with the boruta package," *Journal of Statistical Software*, vol. 36, no. 11, p. 1–13, 09 2010. [Online]. Available: <https://doi.org/10.18637/jss.v036.i11>
- [71] X. Solé, A. Ramisa, and C. Torras, *Evaluation of random forests on large-scale classification problems using a bag-of-visual-words representation*. IOS Press, 01 2014, vol. 269, p. 273–276. [Online]. Available: <https://doi.org/10.3233/978-1-61499-452-7-273>
- [72] "Phpunit - the php testing framework," accessed: 2021-10-08. [Online]. Available: <https://phpunit.de/>



Appendix A



Figure A.1: Partial database schema in the beginning of the project.

