



TÉCNICO
LISBOA

GameCourse - The Next Level

Catarina Passos Correia Quintino Gonçalves

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

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

Examination Committee

Chairperson: Prof. Nuno Miguel Carvalho dos Santos
Supervisor: Prof. Daniel Jorge Viegas Gonçalves
Member of the Committee: Prof. Hugo Miguel Aleixo Albuquerque Nicolau

November 2022

Acknowledgments

I would like to thank my parents for their unconditional support and encouragement over all these years. I would also like to thank my brother, sister-in-law and my nephew for always making me feel confident in my abilities. I cannot put into words how lucky I am to have such a caring and present family.

I would also like to thank my boyfriend Diogo for believing in me, and always being there for me. All of your ceaseless support and help continues to motivate me to be better, and achieve greater things. I am truly grateful to have you by my side, and I hope to continue celebrating these milestones with you.

I would like to thank my friends for their friendship and support. Never in a million years did I think that attending university would connect me with such amazing people. Sofia, Miguel, Francisco, and Paulo, I am forever grateful for your words of encouragement and help throughout all these years. Also, Murteira, and Benzinho, thank you for your friendship, and cheerful presence that continues to put a smile on my face. I am certain I could not have made it without you.

I would also like to thank Joana, my GameCourse partner. It was a pleasure working with someone so devoted. Without a doubt, working with you made this journey more enjoyable. Also, a big thank you to everyone that made this journey possible - all of my friends and colleagues from Instituto Superior Técnico as well as my professors.

Last but not least, this endeavor would not have been possible without the support my dissertation supervisors Prof. Daniel Gonçalves and Prof. Amir Nabizadeh gave me. Thank you for giving me the opportunity to work on this project. It was an honor and I truly enjoyed every second of it.

To each and every one of you – Thank you.

Abstract

Gamification can be defined as the use of game elements and game design elements in non-game contexts. Throughout the years, this concept became increasingly popular due to its benefits when it comes to motivating students to be active in a course. The Multimedia Content Production (MCP) course at Instituto Superior Técnico uses GameCourse, a platform that aims to motivate students through gamification. The current system has a variety of game-like components that make the experience more engaging, encouraging students to learn the course content. To award these game elements, the system relies on rules that specify what are the requirements to give a certain award. However, the system had some performance issues related to the rules that needed to be dealt with and new game elements that could be integrated. The purpose of this thesis is to improve the system's performance, and increase game element diversity in the system. Three new modules were created: Streaks, Teams and Virtual Currency. This also included creating a new type of leaderboard, suitable for teams. From the results obtained from user testing, we concluded that we were successful in implementing and integrating the new modules. Additionally, we were able to find relevant bugs and collect feedback from the students enrolled in MPC 2021/2022, which rated these new game elements to be positive for their learning experience. Finally, the results of the performance tests made show that we improved the the system's performance when it comes to the rules.

Keywords

Gamification; Education; Learning; Tailored Gamification; Rule System.

Resumo

A gamificação pode ser definida como a utilização de elementos de jogo em contextos não relacionados com jogos. Ao longo dos anos, este conceito tornou-se cada vez mais popular graças aos seus benefícios quando se trata de motivar os estudantes a serem activos num curso. O curso de Produção de Conteúdos Multimédia (PCM) do Instituto Superior Técnico utiliza o GameCourse, uma plataforma que visa motivar os estudantes através da gamificação. O sistema contém uma variedade de componentes de jogo que tornam a experiência mais envolvente, o que motiva os estudantes a aprender o conteúdo do curso. De forma a atribuir estes componentes, o sistema baseia-se em regras que especificam quais são os requisitos necessários. No entanto, existiam alguns problemas de desempenho que precisavam de ser tratados e a diversidade de elementos de jogo podia ser maior. O objectivo deste projeto é melhorar o desempenho do sistema, e aumentar a diversidade de elementos de jogo. Foram criados três novos elementos de jogo, um dos quais permitiu a existência de equipas, e, como consequência, um novo *Leaderboard* para exibir as equipas do curso. A partir dos resultados obtidos nos testes aos utilizadores, pode concluir-se que fomos bem sucedidos na implementação e integração dos novos elementos no sistema. Além disso, obtivemos feedback dos alunos inscritos em PCM 2021/2022 e conseguimos encontrar pequenas falhas na implementação que foram depois resolvidas. Finalmente, os resultados dos testes de desempenho efectuados mostram que melhorámos com sucesso a performance do sistema no que respeita as regras.

Palavras Chave

Gamificação; Educação; Aprendizagem; Gamificação à Medida; Sistema de Regras.

Contents

1	Introduction	1
1.1	Objetives	4
1.2	Document Structure	5
2	Related Work	7
2.1	Gamification	9
2.2	Tailored Gamification	10
2.3	Player Profiling	11
2.3.1	Personality Traits	12
2.3.2	Player Types	13
2.4	Discussion	16
3	GameCourse	19
3.1	GameCourse and its Components	21
3.2	Rule System	26
4	Development	31
4.1	External Data Sources	33
4.2	New Game Elements	35
4.2.1	Virtual Currency	35
4.2.2	Streaks	39
4.2.3	Teams	46
4.3	Rule System Improvements	50
4.3.1	Preliminary Work	50
4.3.2	Performance Improvements	51
4.3.3	Automatically Generated Rules	58
5	Evaluation	63
5.1	MCP 2021/2022	65
5.2	User Tests	68
5.2.1	Procedure	70

5.2.2	Participants	70
5.2.3	Result Analysis	70
5.2.3.A	Individual Task Analysis	71
5.2.3.B	NASA Task Load Index	73
5.2.3.C	System Usability Scale	74
5.3	Rule System Performance Testing	74
5.4	Discussion	76
6	Conclusion	77
6.1	Conclusions	79
6.2	System Limitations and Future Work	80
	Bibliography	83
A	GameCourse Partial Database Schema	89
B	User Tests	91
B.1	Tasks	91
B.2	Initial Questionnaire	92
B.3	SUS Questionnaire	92
B.4	NASA TLX Questionnaire	93
B.5	Final Questions	93

List of Figures

2.1	Bartle's extended player types graph [1].	14
3.1	GameCourse modules.	22
3.2	View creation.	23
3.3	Choosing the roles on the View settings.	23
3.4	GameCourse's Plugin module architecture.	24
3.5	Skill Tree example.	25
3.6	Profiling module configuration page.	26
3.7	Example of a rule that uses a metadata variable.	27
3.8	Suggestion given to the users while they type.	28
4.1	Plugin module.	33
4.2	GameCourse's data sources new architecture.	34
4.3	GoogleSheets configuration before.	34
4.4	GoogleSheets configuration now.	35
4.5	Modules page.	36
4.6	Total amount of tokens.	37
4.7	Currency in Skill Tree.	37
4.8	Virtual Currency Prototype	37
4.9	Multimedia Content Production (MCP) 2021/2022 Virtual Currency configuration page. . .	38
4.10	Current Virtual Currency configuration page.	39
4.11	Streaks template.	41
4.12	Streaks within the Profile page.	41
4.13	Creation of a streak.	42
4.14	Examples of consecutive Participations.	44
4.15	Example of the Participations needed to award Constant Gardener Streak.	45
4.16	Common information between a <i>graded post</i> and <i>peerforum add post</i> Participations. . . .	45

4.17 Team Leaderboard prototype.	47
4.18 Team Leaderboard.	48
4.19 Teams import modal.	48
4.20 Teams configuration page.	49
4.21 Teams editing page.	49
4.22 Example of an inactive rule.	51
4.23 Log file sections and separator.	57
4.24 reTrailer Rule after changing the skill dependencies.	60
4.25 Template rule for periodic streaks.	61
5.1 Teams select modal.	73
5.2 Rule System execution times by number of students.	75
6.1 Template for Rule Editor's possible highlight functionality.	81
A.1 Partial GameCourse's database schema.	90

List of Tables

2.1	Game elements used in different gamified systems.	17
4.1	Time complexity in Python data structures.	54
4.2	Execution times of firing certain rules for a student before and after improvements.	56
5.1	Mean, median and standard deviation values of the rating given by students, from 1 to 5, when asked about how engaging and fun were the game elements.	68
5.2	Success Rate for each task.	71
5.3	Mean, median and standard deviation values of the information collected from each task.	72

Acronyms

CSV	Comma-Separated Values
CSS	Cascading Style Sheets
EM	Expectation-Maximization
FFM	Five Factor Model
MCP	Multimedia Content Production
SDT	Self-Determination Theory
SUS	System Usability Scale
TCP	Transmission Control Protocol
TIPI	Ten Item Personality Measure
XP	Experience Points
MCP	Multimedia Content Production
MUD	Multi-User Dungeon
TCP	Transport Control Protocol
UI	User Interface

1

Introduction

Contents

1.1 Objectives	4
1.2 Document Structure	5

Education can have an enormous and powerful impact in one's life. However, as important as it may be, students' lack of interest and motivation is still a big and ongoing concern. To improve students' engagement and performance and, consequently, overcome this struggle, teachers have been trying to adapt their teaching methods. With this need, the concept of Gamification emerged and became increasingly popular throughout the years. Due to its positive impacts, it has been applied in several contexts, education being one of them.

Gamification can be defined as "the use of game elements and game design elements in non-game contexts" [2]. Studies have shown that, in comparison to non-gamified systems, students become more engaged and motivated in courses that use gamified systems [3]. There are several examples of successful applications of gamification in learning environments. One example is the Multimedia Content Production (MCP) course at Instituto Superior Técnico, University of Lisbon. This gamified MSc course rewards students with Experience Points (XP) for the accomplishment of certain tasks, such as skills concluded or badges earned. Each student can gather up to 20000 XP, and at the end of the course, the amount of XP accumulated is converted into a 0 to 20 score. Moreover, MCP has been the main focus of several studies and theses since it uses GameCourse, a platform that has been developed and improved throughout its years in use.

GameCourse is the system that makes MCP a gamified course. The current platform contains several game elements, like Skills, Badges and Leaderboard, that contribute to a more engaging experience which makes students more motivated in learning the content of the course and encourages them to achieve higher grades. Furthermore, to escape the common "one-size-fits-all" approach [4] and be able to take into consideration each user's preferences and needs, a way of profiling users was implemented and incorporated into GameCourse with the intent of assigning them to a specific player profile. This new feature allows the existence of an adaptive gamified system where we can adapt the way each student profile views certain parts of the system. Consequently, the system now holds two different leaderboards, the typical infinite leaderboard that allows you to see every student ranking and a relative one that only displays the student viewing the leaderboard and the nearest students to him/her (rank wise), so that different student profiles only view the one that will, in theory, motivate them. Additionally, the system relies on rules that students need to meet to be awarded a game element, prize or grade. The rules and their functions are handled by the Rule System, a crucial part of the system.

Even though GameCourse has come a long way, there still is some room for improvement. The system had some weaknesses that needed fixing, and the rule system had a poor performance, taking a long time to run or crash trying. Furthermore, several game elements can be implemented and incorporated into the system to increase the diversity of elements as well as further motivate and encourage students to be active in the enrolled courses.

1.1 Objectives

The main goal of this thesis is to take GameCourse to the next level. To do so, we plan on **polishing the system by solving bugs and performance issues the system may have as well as creating and incorporating new game elements that can improve the gamified experience.**

GameCourse was already a well-accomplished system with several functionalities that allowed users to freely interact with it. However, when it comes to the organization of the system's modules, they were alphabetically organized without any sections to separate them according to their purpose, for instance, modules that enable game elements or that enable external data sources into the system. Therefore, there was the need to separate this particular module into new ones, one for each external data source, as well as to be able to differentiate between different types of modules, system-wise and visually. To do so, we added a new attribute *type* to the modules that needs to be specified upon a module creation. Additionally, to guarantee a better organization, a new section for each existing module type was created.

Moreover, Professors were able to create and configure a diverse set of game elements, like badges and skills as well as create pages containing whatever course-related information they desired. These pages could also be made to be viewed in different ways by different types of students. However, despite the existent diversity and functionality, other game elements that could be added to the system. To accomplish this, the state-of-the-art was reviewed so that we could study the game elements that have been explored in similar contexts, focusing on the ones that shown to be the most promising. Additionally, we analyzed and reviewed the data gathered in the previous MCP course to decide what game elements to implement. We ended up incorporating in GameCourse Teams, Virtual Currency as well as Streaks.

Additionally, the system relies on rules that specify what students must do within the course to be awarded a game element or grade. As such, each rule contains *when* and *then* clauses that specify the conditions that must be met for a certain action to be performed. All the existing rules are managed by the Rule System, a crucial part of GameCourse that runs in the background. Professors are able to create and edit the rules responsible for awarding these elements once the student meets all the requirements. However, GameCourse did not provide any functionality that allowed for the automatic generation of rules based on the game elements created. Furthermore, after implementing the new game elements and analyzing how the system behaved during MCP 2021/2022 course, we discovered some performance issues in the Rule System. By the end of the course, we found that the rule system was taking a lot of time to run or would crash trying. This revealed some limitations in how the errors were dealt with and how the rule system functions were implemented. We addressed these issues by dealing with the accesses to the database, refactoring some functions to make them more efficient, creating new ones, and changing existing rules.

1.2 Document Structure

This thesis is organized as follows: the Introduction is followed by Chapter 2 where we review the current state of the art regarding gamification and how it is used to improve students' motivation.

In Chapter 3 we explain the GameCourse system, highlighting some of its features which allows us to give some context about the system in which we developed our work.

In Chapter 4 we describe the work developed and integrated into the system, explaining how it was accomplished and the challenges we encountered.

Then, Chapter 5 is dedicated to explaining how we evaluated our work. We start by describing what problems and bugs were found during the Multimedia Content Production course. This is followed by an explanation of how we performed the user and performance tests. We conclude this section by describing the obtained results.

Finally, in Chapter 6, we present our conclusions and final remarks about this work. We finish by enumerating the future work to be implemented in GameCourse.

2

Related Work

Contents

2.1 Gamification	9
2.2 Tailored Gamification	10
2.3 Player Profiling	11
2.4 Discussion	16

This section discusses the state of art of gamification, including its definition and applications. As such, we will present a review of the existing literature and its findings including the means to create a gamified system that is able to adapt to its users. Thus, a focal point of this section is also the player and user typologies that have been developed as well as its benefits and disadvantages.

2.1 Gamification

Gamification, defined as “the use of game elements and game design elements in non-game contexts” [2], has become increasingly popular throughout the years due to its positive impacts and potential to improve motivation, engagement, and social influence. The use of game elements and game thinking has been around for years, and it is used in several ways. It may pass unnoticed, but teachers rewarding their students with stickers after completing a certain task or even companies or supermarkets giving out stamps for their clients to collect to achieve a certain prize, counts as the use of gamification.

The exponential increase in the interest in gamification, particularly in the fields of education, wellness and business, has been the catalyzer to the making of several studies regarding this topic. In all contexts, the use of gamification has shown to be an advantage in increasing user engagement [3], improving behavior [5] and knowledge retention [6]. When it comes to education and learning, the goal is to motivate students to engage and be active in a course to maximize their success. This is usually done by implementing a system of rewards or providing feedback by indicating their level of performance [7].

Nearly all gamified platforms are built on the use of rewards as its main dynamic by awarding game elements like badges, trophies, and points to their users after they achieve a certain goal or finish a certain task. These achievement-related features [8] are visual displays of the users’ progress which gives them immediate feedback [9, 10], helping them assess their performance and accomplishments. In addition, to allow users to keep track of their own progress as well as their peers, gamified platforms usually incorporate levels and leaderboards, the latter considered one of the most engaging game elements [11] since it positively impacts social comparison between users [9]. Furthermore, to promote cooperation, collective game elements are also implemented. Teams and, subsequently, team leaderboards, have shown to have many positive benefits, promoting social connection amongst users, making them engage in strong social connectivity via competition and comparison of points and scores [12].

There are several examples of well-succeeded educational gamified applications, such as ClassDojo¹, Duolingo², Kahoot!³, Khan Academy⁴ and SoloLearn⁵. Each one of these systems uses different game elements that further motivate its users to be active and stay engaged. Duolingo, for instance,

¹<https://www.classdojo.com>

²<https://www.duolingo.com>

³<https://kahoot.it>

⁴<http://khanacademy.org>

⁵<http://sololearn.com>

was created with the intent of providing the best universally available online education so that its users can learn a new language while having fun. To keep users motivated and engaged with the courses, Duolingo takes advantage of the use of game elements such as leaderboards, levels, points, progress bars, streaks and lingots (in-game currency). With a staggering total of 500 million users and around 40 million monthly active users, this approach can be, without a doubt, considered extremely successful. Other implementations concerning gamification follow a similar design, focusing on a single and generalized approach in which the game elements used are the same for all users.

Taking into consideration that individual needs are a major and crucial factor in gamification, they are, at the same time, one of the main causes for its applications to fail. Gamification is only effective if it enhances users' motivation and improves their engagement and performance [3]. Thus, the general use of game elements is not adequate since, although appropriate game elements can lead to higher levels of user motivation, inappropriate game elements can seriously demotivate users. For instance, game elements that allow social comparison to stimulate a competitive environment are not appealing to users that do not enjoy competition. Furthermore, the rewards given when acquiring game elements must captivate the users by bringing additional utility: students that received badges without having a concrete reward, such as an impact on their grades, disliked the use of gamification [10, 13].

Even though there are several works showcasing the benefits of gamification and its successful applications, there still is room for improvement. Identifying user needs and preferences and discovering the best ways to use this information as an advantage in the implementation of gamified systems could help create a more inclusive version of gamification.

2.2 Tailored Gamification

The current gamified applications are designed based on a "one-size-fits-all" idea. However, studies have indicated that treating individuals as a homogeneous group is not an optimal design approach since everyone has its own preferences and needs. Researchers concluded that using personalized gamified learning experiences helps raise students' performance, achievements, and gamefulness experiences [5]. As such, personalization seemed to be better at engaging students behaviorally and emotionally [14].

Tailored gamification emerged as a means to improve the effectiveness of gamification [15]. It can be described as the personalization of gameful design elements, the interaction mechanics, the tasks, or the game rules for each user, according to their preferences [14]. This personalization can be achieved by customization, allowing the user to select the elements to be used, or by adaptation, where the system selects the elements for each user. When it comes to customization, the goal is to improve user experience by allowing them to choose the elements they desire. Tondello et. al [14] conducted their

own studies where they compared a customizable gamified system to a generic one. Participants in the customization group had the freedom to choose as many game elements as they wished from eight available options and the ones in the control group had all elements automatically enabled for them. It was concluded that personalization or customization of gameful application leads to higher performance than generic gamification.

Considering how user motivation is affected by personality traits or user types, implementing a new and user-focused version of gamification seems like a promising approach to several researchers [16]. Understanding the relationship between player types and personality types and traits in relation to game elements and mechanics can lead to more appropriate and meaningful choices for gamified systems. Therefore, researchers became interested in implementing models that have the potential to improve the gamified experience, by automatically tailoring it to each individual. For that effect several models have been developed and tested.

Some of the conducted studies tested several models and found that, for each model, certain game elements are better suited for a particular trait or type of user. This means that all personalized gamified experiences depend on player profiling focusing either on player type models such as Hexad or Bartle's or on personality-based models such as the Five Factor Model (FFM), also known as the "Big Five" Model. In certain studies, the gender of the participants was also examined but never studied in depth since it wasn't shown to be suitable and accurate due to its many limitations. Nonetheless, many of the developed studies provide clear and valid results that can help guide the implementation of personalized gamification.

2.3 Player Profiling

To create a gamified system that is able to adjust to its users' motivations and preferred types of interactions and, thus, improve their experience as well as performance, player profiling should be implemented. Users must be classified into groups, i.e., player types, that best fit their characteristics as players. This helps in tailoring the system and discarding the "one-size-fits-all" approach.

There are various studies that investigated if player profiling truly works while simultaneously trying to find the best and most effective methods to implement. In addition, studies also tried to find the correlation between player types and game elements. Some authors focused on personality traits, believing that categorizing players based on these traits is the best approach. Others focused on player type models. In this case, there are several available models that can be implemented, each categorizing users into a different set of player types.

However, some approaches to player profiling have shown better and more accurate results than others. Nevertheless, these findings are important and relevant to reach a conclusion on what the best

method is.

2.3.1 Personality Traits

Personality can be defined as “a stable core of emotions, dispositions, attitudes, and behaviors that uniquely characterize a person at a specific point in time and shape development across the lifespan” [6]. In other words, the way we feel, think, act, and behave at a certain moment in our lives helps to build our foundations, as individuals, influencing the development of ourselves as we grow [17]. On the other hand, personality traits refer “to an inner tendency or predisposition for a person to act in a certain way” [6] which means that these characteristics, psychological in nature, are stable over time [18]. Thus, they can act as a type of measuring stick, assessing to what extent a person exhibits or possesses a particular trait.

The importance of personality traits has been widely acknowledged since it has a significant impact in daily tasks like working, learning [19] and, consequently, on academic achievement. So, the interest in identifying which traits each user possesses and using them to improve personalized gamification comes as no surprise. The FFM, the most used model when it comes to personality, characterizes individuals based on their personality traits, identifying five dimensions: Neuroticism, Extraversion, Openness, Agreeableness, and Conscientiousness. This is a complete model that provides a coherent taxonomy [18], being extremely helpful with player profiling. Moreover, personality is a relatively stable measure to explain behavior so authors that have used this model consider it to have advantages over using models based on perception. Due to its popularity, newer and revised versions of the FFM have been developed, for instance, the Ten Item Personality Measure (TIPI). This version was created to provide faster results since the FFM requires time to be correctly employed and, subsequently, give accurate and trustworthy results. However, since TIPI is a briefer measure, it is considered less reliable.

Results from studies that focused on personality based tailored gamification and used the FFM showed that gamification was considered more beneficial to introverts participants than extroverts [20]. Concerning conscientiousness, results showed that, for highly conscientious individuals, gamification is negatively perceived [18] which, in theory, matches this personality trait since it is associated with planned, thorough, careful, and precise behavior. When it comes to the interaction with game elements, introverted participants had a higher number of points, medals, and logins than extroverted ones [20] while the use of customization, leaderboards and levels were suggested to the latter [21]. This finding is consistent with other studies that have shown that introverts behave in a more extroverted way in environments that require less human interaction [19]. Moreover, individuals with low neuroticism [14] or low agreeableness [19] seemed to enjoy progress feedback game elements while individuals with high openness enjoyed levels the most [14].

However, the results regarding personality traits were not coherent, having different or weak results

concerning the same personality traits. This might be due to some limitations that were acknowledged. Most studies had limited time to survey their users and some had a small sample of participants, making the researchers use the median for classification [20]. Furthermore, in many of the studies, game elements preferences were measured using questionnaires and, as it has been proven, self-reported items are known to be problematic for analysis due to the many biases in the responses [19]. In addition, when it comes to the existing research concerning this type of player profiling, it has been demonstrated that personality traits provide only a partial explanation of differing motivations and playing style in games [16]. Therefore, other models have been studied to implement a more stable and trustworthy personalized gamified system.

2.3.2 Player Types

Another way of profiling users is by categorizing them into player types. Player types have been described as being the same synthetization as personality types, with the difference being the context in which they are applied [17]. There are several works regarding player types, most relying on player or user typologies like BrainHex, Hexad and Bartle's player type model, one of the oldest and most frequently used.

Bartle's player typology was created based on a specific game type: Multi-User Dungeon (MUD). It originally divided users into four player types based on two axes that express the player's desire to interact with or act on the virtual world or on other players [16]. Players could be Achievers, that act on the world, Explorers, that interact with the world, Socializers, that interact with other players, and, finally, Killers, that act on other players. Due to some limitations of this model and with the intent of enhancing player type diversity, a third dimension, based on implicit (done automatically without the intervention of the conscious mind) and explicit actions (planned actions, usually to achieve a certain goal) [22], was later added to this model (Fig. 2.1). With it, each of the already existing types was divided into 2 sub-types, resulting in a total of 8 player types.

When it comes to the relation between player types and game elements, some works suggested challenges and levels for Achievers, collections for Explorers, badges, leaderboards and levels for Killers and guilds for Socializers [16]. However, this model provides a way to specifically classify MUDs' players which can have some application limitations since it may not work in other contexts. In addition, there was not a consensus in the existing literature regarding the relations between Bartle's player types and game elements which can lead to an inaccurate tailored gamified system.

Another well-known player model is BrainHex, a top-down approach based on neurobiological insights [23]. This player typology was developed to improve individual gamified experience and takes inspiration from literature on game emotion, neurobiological player satisfaction research as well other typologies like the Demographic Game Design 1 and Demographic Game Design 2. It presents 7 dif-

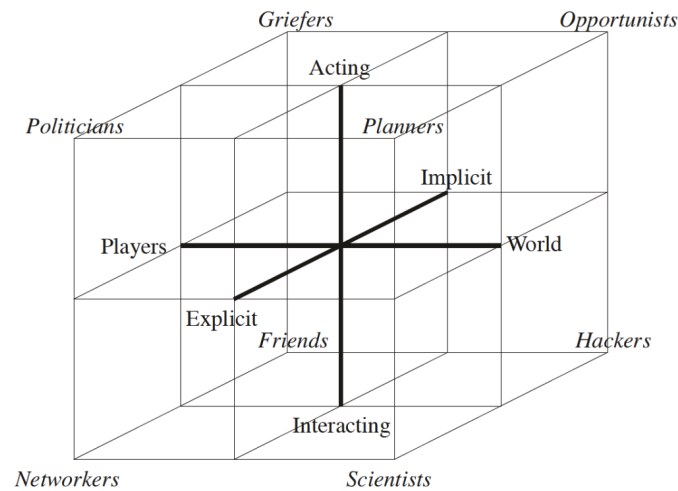


Figure 2.1: Bartle's extended player types graph [1].

ferent player types: Achiever, Conqueror, Daredevil, Mastermind, Seeker, Socializer, and Survivor, each with distinct motivations. Several studies employed this typology and some suggested the use of levels for Achievers and leaderboards for Conquerors while others proposed time pressure (having a reduce period of time to perform a certain task) for both Achievers and Daredevils and signposting, which serves to guide the player in a certain direction without a concrete instruction, for Socializers.

Different authors also proposed other player typologies. Barata et al. [24] studied user performance and gaming preferences. To classify users into different profiles, clustering algorithms, such as the (Expectation-Maximization (EM)) algorithm, were used. Four student types were proposed: Achievers, Regular Students, Halfhearted Students and Underachievers [21]. After employing this typology and analyzing the results, challenges, points and badges were suggested to attract Achievers, Regular and Halfhearted users. Moreover, Ferro et al. [17] studied various models of personality traits and types as well as player types to relate them with game elements and mechanics. Their work, purely theoretical, proposed a player type model that identifies 5 different player types: Creative, Dominant, Humanist, Inquisitive and Objectivist.

Marczewski [25] introduced the Hexad typology, specifically developed for gamification. This model was initially based on the Self-Determination Theory (SDT) since the proposed user types differ based on how they can be motivated by either intrinsic (which refers to “[...] doing something because it is inherently interesting or enjoyable [...]” [26]) or extrinsic motivation (which refers to “[...] doing something because it leads to a separable outcome” [26] like receiving rewards). In fact, SDT proposes that tasks are more likely to be intrinsically enjoyable if they support competence, autonomy, and relatedness, three basic human psychological needs. Thus, the player types proposed in the Hexad model differ based on these needs as well as intrinsic and extrinsic preferences. Furthermore, meaning (purpose)

showed to be an important factor and, as such, it is also considered relevant to this model. Tondello et al. [27] extended the Hexad model by developing a scale for scoring an individual, assigning each one to a certain user type. Users must answer a self-report questionnaire that presents the results as a collection of 6 scores [16], i.e, categorizing users into 6 different players:

- **Philanthropist:** motivated by purpose and meaning, these are the altruistic players that are willing to give without expecting a reward.
- **Achievers:** motivated by competence and mastery, they wish to learn new things and excel in the activities they are involved in.
- **Disruptors:** motivated by change and, as such, have a tendency to disrupt and test the limitations of the system.
- **Free Spirits:** motivated by autonomy and self-expression, they value freedom and enjoy exploring without restrictions.
- **Socializers:** motivated by relatedness, these players want to interact and connect with others.
- **Players:** motivated by extrinsic rewards, they are willing to do what it takes to gain rewards within a system.

In the existing literature concerning this typology, there is strong evidence that establishes relations between game elements and player types. Achievers enjoy challenges and levels, Free spirits prefer unlockables and customization, Players like leaderboards and virtual economy and Socializers appreciate networks, social status and competition. Results also suggested customization to Disruptors and gifting to Philanthropists [21].

Furthermore, Lopez and Tucker [28] asked participants to rank the game elements based on the degree to which they perceived them as fun, useful, preferable, motivating and frustrating. Using a sensor to measure performance in real-time, their results showed that Achiever participants improved their performance more than others but participants that scored high in the dimensions of Philanthropist or Free Spirit on average performed better than other participants. However, Free Spirit, Philanthropist, or Player participants tended to perform worse in the gamified application than in the non-gamified application. This can be explained since the effects of extrinsic rewards, such as the use of game elements like points and badges, on an individual's motivation vary according to their perception of these elements as controlling or informational.

Regarding the relation between player typologies and game elements, several different approaches have been taken. Most studies focused on interaction analysis and used surveys and questionnaires to evaluate user experience as well as understand how players perceived different game elements. Aldemir

et al. [29] used interviews, in-class observations and documents, such as e-mail logs, comments, and online activities, as their main methods for data collection. Interviews were considered the most fruitful method when it comes to providing information concerning the participants' perceptions and reactions to the game elements.

On the other hand, Knutas et al. [30] conducted a case study using interaction analysis and K-means clustering with Pearson's correlation coefficient to create gamification preference profiles. Using the two-axed Bartle player typology, they presented an evidence-based method for deciding which game elements are the most appropriate for each player type and how to apply them. Lavoue et al. [31] proposed an adaptive gamification model based on a linear model to estimate the adequacy of gaming features for the BrainHex player types. Participants were asked to answer the BrainHex player type questionnaire. Then, relying on a matrix factorization, learners' preferences for each feature was computed as $\mathbf{R} = \mathbf{B} \mathbf{A}$, the product of the preference of each BrainHex player type for each feature (\mathbf{A}) by players' answers to the BrainHex questionnaire (\mathbf{B}). Even though this study relied on a specific player typology, the model proposed can also be used for other player profiling techniques, such as personality traits. Moreover, Tondello et al. [27] calculated the correlation between each pair of Hexad types with the corresponding set of game design elements using Kendall's rank correlation (τ).

2.4 Discussion

The existing literature shows how promising gamification is as well as the benefits its use brings in the field of education. Overall, it was concluded that participants who interacted with gamified applications performed better than those who interacted with non-gamified applications.

Results also concluded that certain game elements, such as points, badges, and leaderboards, had an overall positive impact on gamified experiences. Moreover, it was concluded that gamification is more effective when the interests and needs of its users are put into consideration. Results with Hexad were the most consistent with the definitions of its user types, and that its types had more influence on the perceived user motivation than those from previously discussed models [32]. It was also concluded that tailored gamification leads to higher performance than generic gamification. Thus, tailoring the experience to each type of player is an advantage when it comes to engaging users and improving their performance.

Concerning game elements, studies found that players that are motivated by extrinsic rewards enjoy leaderboards as well as virtual currency while players that wish to excel in the gamified system prefer levels, challenges and time pressure. Moreover, signposting, competition and guilds were suggested for players that enjoy interacting with others [16, 21]. However, most gamified systems and studies made use the same game elements leaving others unexplored. As can be seen in Table 2.1, some of the most

Table 2.1: Game elements used in different gamified systems.

Gamified Systems	Badges	Levels	Leaderboard	Points	Streaks	Virtual Currency	Teams
ClassDojo	✓			✓			
Duolingo	✓	✓	✓	✓	✓	✓	
Kahoot!		✓	✓		✓		✓
Khan Academy	✓	✓		✓			
SoloLearn	✓	✓	✓	✓			

popular and known gamification platforms use badges, levels, points and leaderboards, and only two of them explore Streaks, Currency or Teams. In addition, there are several works that focused only on the most popular and common gamified components, such as badges, levels, leaderboards and points. Thus, there are strong conclusions regarding these elements whilst there is little evidence for others, like teams, signposting or easter eggs [21], that could be particularly beneficial to use in gamification.

When it comes to leaderboards, for instance, results showed that most students enjoyed the competitive environment it creates. However, a small group of participants did not enjoy them, which can suggest that the implementation of different types of leaderboards, including one for teams, could be an advantage [29]. On the other hand, Streaks, a not so commonly used game element in the existing literature, have shown promising results for Duolingo, Kahoot! and other gamified systems [33]. Kahoot!, for example, introduced a streak that rewards users for consecutive correct answers with the intent of preventing them from randomly clicking on answers just to get the next question. They found out that users cared more about their streak than overall score in points [33].

Since different player types have different preferences and needs, some users in gamified experiences do not enjoy certain game elements as much as other players. This can be explained by the fact that the effects of extrinsic rewards on an individual's motivation vary according to their perception of these rewards. This can lead to less active users and motivating them as shown to be a common concern. Roosta et al. [34] suggested that students could choose the game elements they enjoyed the most. This approach can help understand if the users' chosen elements match their respective player types when player profiling is done. However, to choose the right elements one must be acquainted with them, which is usually not the case since users are not necessarily gamers. Thus, users may not be familiar with game design vocabulary and can end up choosing elements they perceive as motivating that end up having the opposite effect.

Moreover, even though the mentioned contributions shown positive impacts when it comes to the use of gamification, there are some limitations that should be dealt with. Several studies relied on self-reported items, such as questionnaires, for analysis. These methods have shown to be problematic since they are prone to bias [19]. Participants are not always aware about how they will behave when faced with a competitive or a collaborative environment. Small or unbalanced groups of participants should

also be avoided, and studies should not have limited time since time restrictions can end up providing unclear conclusions. Furthermore, Tondello et al. [14] found no relation between participants' player type and their most influential or motivational game elements. This suggests that using only a single player or user typology to adapt a gamified system to individuals needs is not an optimal approach. Therefore, gameful designers should use player type models as one of the factors for adaptive gamification but not the only one.

3

GameCourse

Contents

3.1 GameCourse and its Components	21
3.2 Rule System	26

GameCourse, initially created for the Multimedia Content Production Course from the Information Systems and Computer Engineering MSc, is a framework that courses can use to provide a gamified environment to their students. It uses several game elements such as, experience points, badges and levels, to motivate students to be more engaged and active in their course. However, the system has been the subject of several thesis, and as such, went through numerous improvements and changes. In this section, we will describe GameCourse as it was when the development of this thesis started.

3.1 GameCourse and its Components

Throughout the years, GameCourse has been the focus of several theses and studies. Thus, these contributions helped bring GameCourse to its current state. In 2016, the Smartboards system was introduced by André Baltazar [35] which, although functional, relied on manually run scripts. However, as a result of Diana Lopes' GameCourseBeyond [36], Patrícia Silva's GameUI [37], and Inês Paiva's AutoGame [38], this system was improved, introducing GameCourse as we know it. Finally, in 2021, Ana Nogueira's DynaGame [39], Ana Gonçalves' GameCoursePersonal [40], and Mariana Saraiva's GameCourseUI [41] added the Rule Editor and Profiling module in the system, both crucial functionalities that we explain more in-depth later in this section, as well as improvements to the system's User Interface (UI). Moreover, students that were enrolled in the Multimedia Content Production (MCP) course also provided feedback regarding their experience and what could have been done to improve it.

GameCourse's architecture is composed of several components that support its features, such as Course, CourseUser, Role, Modules and Views. A Course can have many CourseUsers, each representing a GameCourseUser within a specific Course. Each CourseUser can have different Roles within a Course. The default Roles are "Teacher", "Student" and "Watcher" but custom ones can be created. Moreover, GameCourseUsers can authenticate through Fénix, Google, Facebook or LinkedIn. A GameCourseUser can have Admin permissions which allows them to create new Courses, activate or deactivate users, and change Admin permissions. They can also create new roles, assign them to CourseUsers, and choose which permissions that role has. By default, the first account created in GameCourse is always an Admin and other accounts always start without Admin permissions.

The system also offers a repository with several Modules, shown in Figure 3.1. Each module requires its own set of functions and variables to be properly integrated in the system, which are saved in the module's own library and stored in a global **Dictionary**. All modules use the same vocabulary, that we refer to as **Expression Language (EL)**, that enables semantic operations within GameCourse. Each module is responsible for enabling a new functionality into the system. The existing modules include game elements, like Skills and Badges, as well as other important tools to either profile students into a role that best suits their behavior or to retrieve and insert data from external data sources. Each module

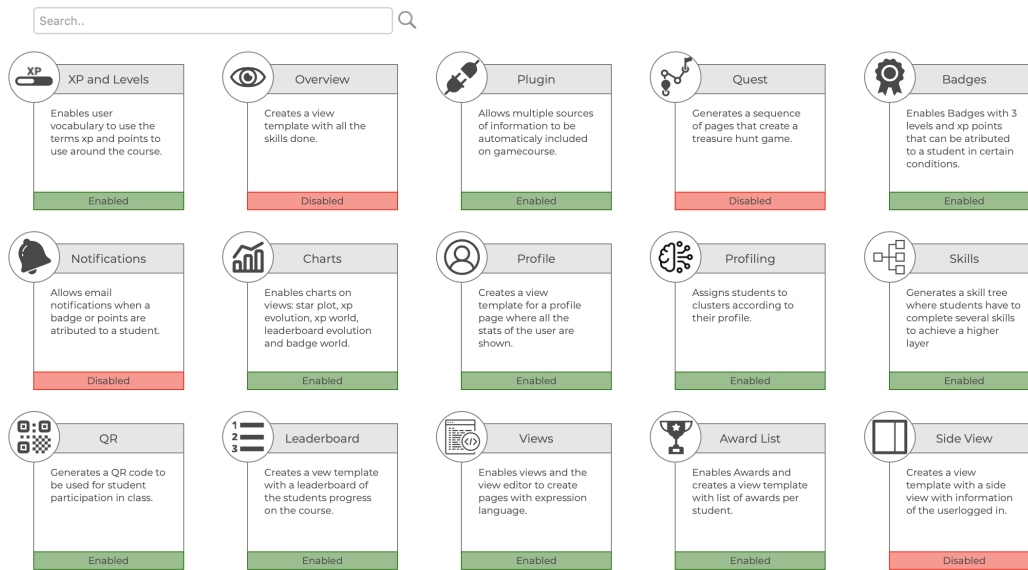


Figure 3.1: GameCourse modules.

can be enabled or disabled within a specific Course following some dependency rules that may exist between modules. For instance, if we wish to enable the Skills module, we need to enable the **Views** and **XP and Levels** modules first. It is important to note, however, that every module in GameCourse depends on the **Views** module.

Views are the contents displayed to users. Each view is composed of other views which allows the creation of more complex ones. A view is composed by multiple elements: **Content**, defining what to display, **Data**, a query to fetch the data from the database to be used in the view, **Events**, the actions to perform when an event is triggered, **Visibility**, the visibility of a view, **Label**, a string of characters to identify the view, **Style**, Cascading Style Sheets (CSS) code, and **Variables**, a list of GameCourse variables that can be used in the context of the view and its sub-views.

The most basic views are images or text elements that can be inside tables or blocks. There are also headers, values and templates, one of the most complex view types. Templates, as the name suggests, allow any view to be saved in the View Library as a template. A user can later use it to create a view, either by copy or by reference. On the other hand, creating a view from scratch (Fig. 3.2) is done with the help of a view editor and it can be customized using the system's Expression Language, which is a built-in functionality that provides a vocabulary specific to the system. Once a view is created, the user can choose to show it as a Page assigning a certain name to it, for instance, the Leaderboard page.

In addition, GameCourse supports rendering of views in distinct ways. This is achieved through Aspects that allow the customization of views for different roles or users. There are two types of view aspects: **Role-Single** and **Role-Interaction**. Role-Single shows a different aspect according to the

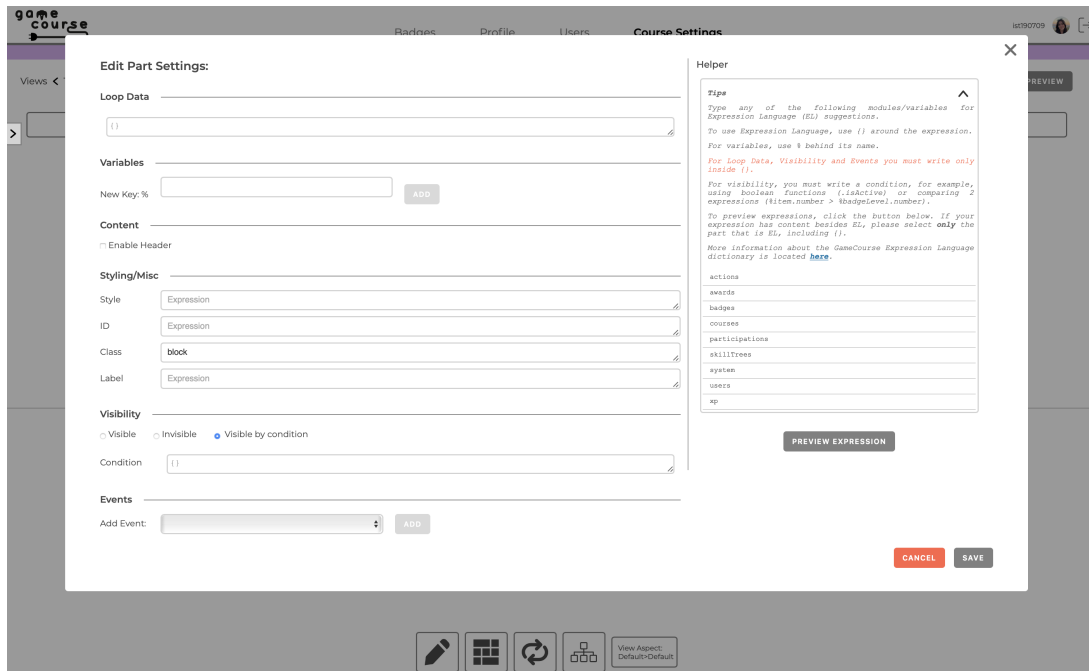


Figure 3.2: View creation.

viewer's role while Role-Interaction also takes into consideration the role of the user associated with the page. Each view defines a pair (user role, viewer role) with "user role" being the user role that a page belongs to and "viewer role" being the role of the page viewer. By default, this pair is (Default, Default), except for Pages that belong to a specific "user role", such as the Profile Pages. This pair can be defined in the specialization section as shown in Figure 3.3. When a CourseUser loads a page, GameCourse picks the most appropriate aspect based on the pair values. This possibility of displaying a page according to the roles of the users viewing the page or the user associated with it opens the door for new adaptive capabilities in the system.

Another important module is the **Plugin** Module that enables into the system the functionality needed

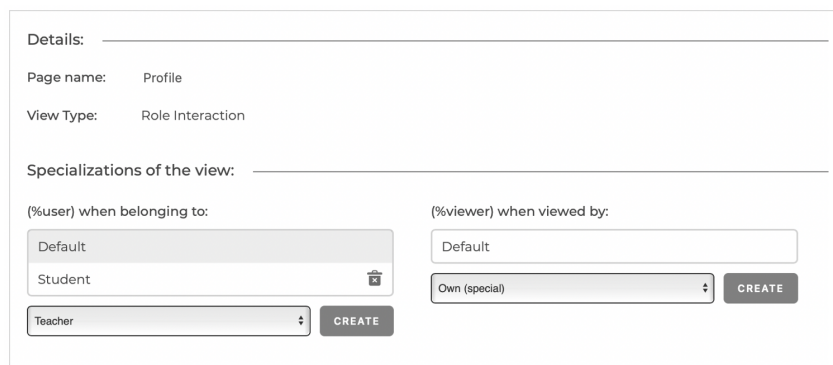


Figure 3.3: Choosing the roles on the View settings.

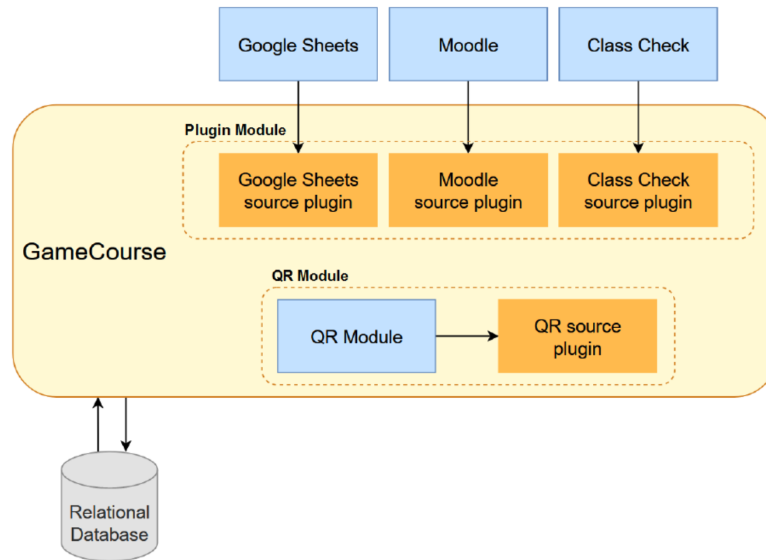


Figure 3.4: GameCourse's Plugin module architecture.

to retrieve data from external data sources, like, Moodle ¹ and GoogleSheets ². GameCourse needs to have access to external data from different sources and as such, there is a script for each data source that are run in the background, scheduled and repeated at an established frequency through Cron Jobs, a job-scheduling tool for Linux/UNIX. These scripts are responsible for accessing and parsing the data that is going to be inserted into the **participation** table in the database or updated if that data already exists (Figure 3.4). All the data logs within this table are referred to as Participations. There are a total of 43 different types of Participations that differ according to its source that allow us to identify what a student has done. For instance, Moodle inserts Participations of type *"peeforum add post"* and *"graded post"* while GoogleSheets inserts Participations like *"lab grade"* and *"attended lab"*. Additionally, the information each Participation holds also varies according to the data source it was retrieved from. They do, however, all possess information regarding the user, course, type of participation, and the timestamp from when it was stored in the database.

Moreover, several modules enable a different type of game element onto the system. The **Badges** module enables badges whose main goal is to reward students for certain behaviors. For example, the "Course Emperor" badge in the MCP course is given to the student with the highest course grade. Additionally, the **Skills** module enables the existence of skills within the system. Skills are assignments that can be completed at any time by a student and later graded from one to five by a professor of the course. All the available skills have a thread within Moodle where students can submit their work, and professors can grade them. Both the students' attempts and the grades received, are inserted as

¹<https://moodle.com/>

²<https://www.google.com/sheets/about/>

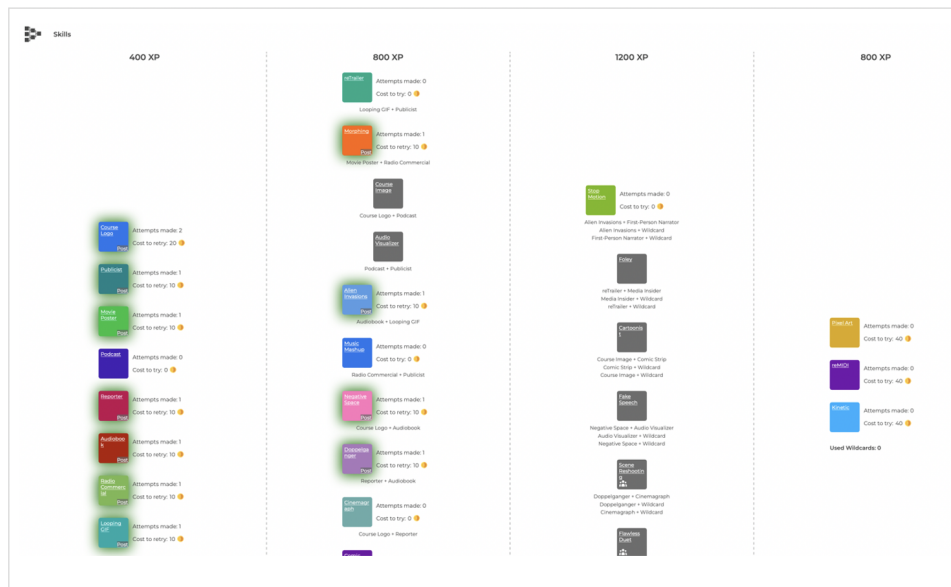


Figure 3.5: Skill Tree example.

Participations into the database when the Plugin Module is enabled. The several skills the students can complete are grouped by the amount of work needed and organized into tiers. Furthermore, most skills have dependencies that require a pair of skills to be completed beforehand. If those requirements are met, the skill is unlocked for the user to perform. All this information is displayed in the Skill Tree, a view part inside the Profile Page where each rounded square represents a skill and, below it, the dependencies are displayed. If a skill is unlocked, its square is filled with that skill's color. Additionally, once a skill submission is graded with a rating equal or higher than three, the skill is considered complete, and a green shadow is displayed beneath it, as it can be seen in Figure 3.5.

Additionally, supporting individual differences through adaptive learning mechanisms has become one of the main purposes of the system. In one of the most recent theses done with GameCourse, GameCoursePersonal by A. Gonçalves [40], the Profiling Module (Fig. 3.6) was created. This module provides a predictor to determine the ideal number of student profiles and a profiling functionality, responsible for assigning each student to their respective profile, each representing a player type according to Barata et. al [24] player typology. To assign a user to a certain student profile, the profiler makes its decision based on three types of data: the students' total XP per day, badges earned and skills completed per day by student and, finally, all the actions performed per day per student. Consequently, GameCourse also offers two leaderboards types: an infinite leaderboard, that shows all users and their scores, as well as a relative one, where the student views the three nearest students to him/her (rank wise). The former is more suitable for students in the top positions and the latter for students in the lower positions. The reasoning behind this is that, for students in higher positions, it can be motivating to see how close they are to the top and how many students are below them. On the other hand, being in the

lowest positions means scrolling past every single student and seeing how many students are above them which can be quite demotivating. To be able to attribute the right view to each student according to their profile, professors are now able to assign a view to a specific cluster of students.



Figure 3.6: Profiling module configuration page.

3.2 Rule System

GameCourse requires rules that establish what are the conditions students need to meet to win a game element or prize. The Rule System, also referred to as AutoGame, is the component responsible for the rules and it deals with everything related to retrieving the data that needs to be validated for the rules to have the desired effects.

GameRules is a component of this system that allows the creation and usage of expressive rules since it provides the back-end functionality that is responsible for attributing the existent game elements or removing them if needed. Its essential elements are rules, facts, targets, scope, and outputs. **Rules**, a text-based rule written in Python with *when* and *then* clauses, where the preconditions to met and actions to perform are established; **Facts**, the data that the rule system can operate over, which can consist of objects, logs of previously fired rules and external data; **Targets**, a set of objects for which a rule can be fired; **Scope**, that establishes what can be expressed by the rules, i.e., which functions, variables, and modules are available; and, finally, **Outputs**, that are the effects of firing a rule. In MCP,

```
rule: Initial Bonus
# Attributed to everyone in the beginning of course

when:
    bonus = METADATA["initial_bonus"]
then:
    award_prize(target, "Initial Bonus", bonus)
```

Figure 3.7: Example of a rule that uses a metadata variable.

the Rule System's targets are the students, the facts are the participations students have done, the output are the awards given to the users, and, finally, the Rule System's scope contains the metadata variables. These are a set of variables that are relevant to the whole Rule System and, as such, are global variables that can be used in all rules of a given course.

The rules can be created to perform any action the user might need and they may vary depending on the course. Consequently, they are stored in different text files according to their context and course. In MCP, for instance, there is a file for each game element for which rules are needed, one for badge-related and another for skill-related rules, and a separate file for all rules that are not game element related. Due to the rules-related dependability some modules may have, the Rule System also provides the automatic generation of rules based on a given template. However, this is only available for the Skills Module and only happens when a user creates a skill. As such, if a skill is edited, no change will happen to its rule.

In order for the system to interpret the contents of a rule, the **rule parser** module within GameRules was created. This module provides the functionality for parsing the rules files and its contents. As such, this allows the system to identify the functions to be used. The rule parser also allows the use of functions from a module's library. This is done through the **gc** function that opens a client socket to communicate with GameCourse. To apply this in a rule, the **GC** prefix must be written, so that the rule parser knows it has to call the **gc** function, followed by the name of the library to access and the function to run, as can be seen in Figure 3.8.

In addition, within GameRules, there is also a Python code module called **connector** that is responsible for dealing with all the necessary database connections performed by the Rule System. With it, we can retrieve the targets that have performed recent participations on the system, retrieve the respective logs from the database, return the desired effects that are usually inserted or updated in the **award** table, and manage the execution of the Rule System.

Another important mechanism is the logging mechanism. It is responsible for writing into individual files, one for each of the existing courses in GameCourse, the dates at which the rule system started and finished running as well as any errors that might have occurred while the Rule System was running.

Considering that the system can crash or return incomplete results due to any errors that can appear, this logging mechanism has proven to be extremely helpful since it allows us to understand and locate the root of an error that occurred while the Rule System was running.

Since Rules are quite important and can be a bit complex, GameCourse now has a Rule Editor that allows users to perform multiple actions related to the rules, as the result of Ana Nogueira's DynaGame [39]. In the Rules page, the users views a list of the existent rules within the course, where they can choose to create new ones or import, export, view and edit those rules. In the editing page (Figure 3.8), the user can establish the name and description of the rule and then write it. The page also contains a mechanism that provides tailored suggestions to the user. This is done in three different ways: by automatically suggesting a function while the user types, which is an autocomplete functionality that displays all the suggestions that match the syntax typed, by displaying those same suggestions on the function suggestion box to the right of the page and by listing the available metadata variables.

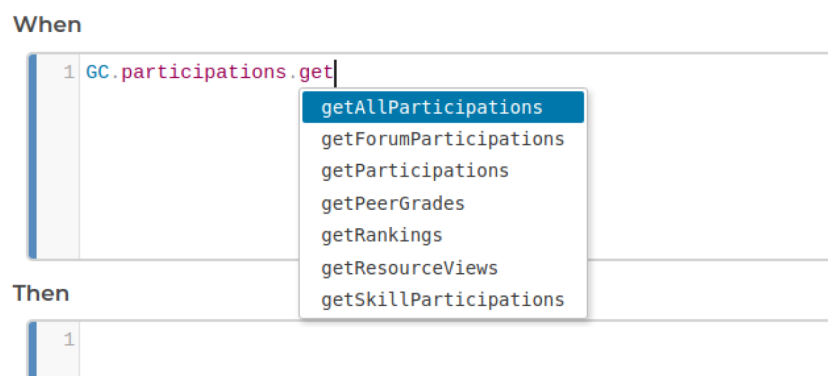


Figure 3.8: Suggestion given to the users while they type.

In addition, a user can also preview each function to check if it is correctly written and returning the desired output. This can also be done for the rules so that the user can test it to determine its validity. If it turns out to be a valid rule, the preview will display the changes that will be inserted in the GameCourse database once that the rule is fired. Otherwise, the preview will return an error message that will help the user with correcting the rule.

To fire and parse the rules, the Rule System makes a call to GameRules. This call is done every time new data, retrieved from the data sources, is inserted or updated. To establish communication between the system and AutoGame, GameRules creates a Transport Control Protocol (TCP) socket in PHP, that acts as the server, and a Python socket that acts as the client, and connects to the server when it is required. Then, the rules are retrieved from the system and they will only be run for the students' that have new Participations or the previously inserted ones have been updated. After all rules have been run for each target, the total XP and the current level of each of the target are calculated. Then, the system sets itself as non-running and updates the start and finish dates for the execution that are stored

in the **autogame** table, also writing these dates in the respective log file.

A partial database schema of GameCourse when we started this project is shown in Appendix A. The schema does not include all the existing tables since each module can create other tables when it is enabled. However, all the tables previously mentioned, and other relevant ones, are displayed to allow a better understating of what was explained in this chapter.

4

Development

Contents

4.1 External Data Sources	33
4.2 New Game Elements	35
4.3 Rule System Improvements	50

In this chapter, we explain all the work done throughout the development of this project. We begin by describing the changes made that improved the system. Then, we explain the new game elements that were integrated into GameCourse, including all the work done related to their implementation. Finally, a detailed description of the changes made that allowed the improvement of the Rule System is given.

4.1 External Data Sources

The **Plugin module** allowed multiple sources of information, such as Moodle and GoogleSheets, to be automatically included on GameCourse. However, as can be seen in Figure 4.1, there were no indicators of what were those sources of information since the description of this module is vague. To find out, one would have to visit the module's configuration page which contained a section for each one of the external data sources. Subsequently, this implementation did not allow users to activate and deactivate them separately since they were all contained in the same module. As such, enabling the Plugin module would include all those sources in the system, even if the user did not require them all.

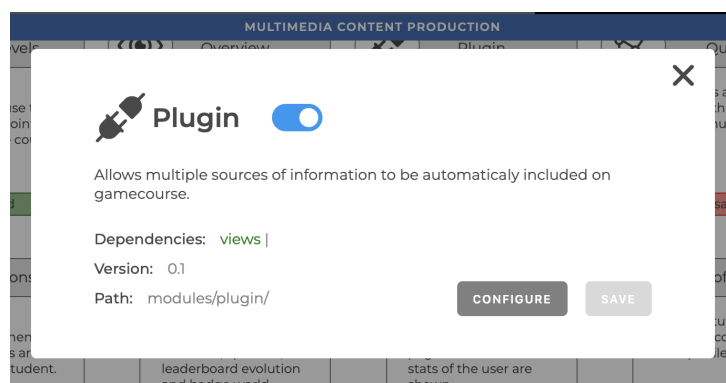


Figure 4.1: Plugin module.

Thus, this module was divided into four new ones, each corresponding to a different data source. Code-wise, each data source was already being handled individually, having a script responsible for retrieving the data in question. However, the functions responsible for storing and editing the variables of each data source were all within the same file, which we had to compartmentalize. In addition, the configuration page of the Plugin module had a lot of styling and understandability problems that we fixed when creating this page for each one of the new modules. The GoogleSheets before and after is a great example of these improvements, as can be seen in Figures 4.3 and 4.4. There was no alignment between the components and adding a new sheet was not an easy task since there were no placeholders on the text inputs. As such, a user would have a hard time understanding what those inputs should be. On top of that, the user did not have the option to remove a sheet, having to delete the content of the text inputs to achieve a similar result. After our improvements, the page had a cleaner

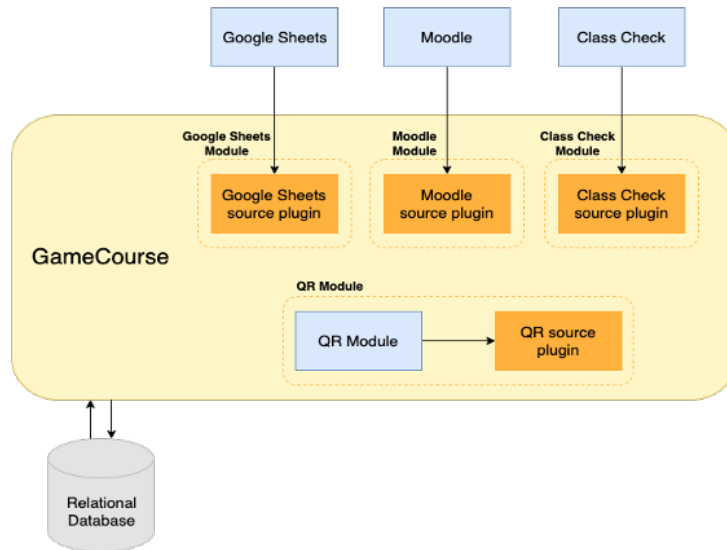


Figure 4.2: GameCourse's data sources new architecture.

Figure 4.3: GoogleSheets configuration before.

look, the text inputs were identified with a label and each added sheet could be removed by clicking on the icon on the right. Furthermore, as a result of my colleague's, Joana Seginando, work regarding the GameCourse UI, these parameters now have an informative component right next to them that, when hovered, displays an informative description (Figure 4.4).

Also, to separate game element modules from data source modules, two sections were created in the Modules page. In order for the system to know which section each module belongs to, changes in the modules' creation were made. Now, when creating a new module, its type needs to be specified: "DataSource" or "GameElement". Later on, we decided to add a new type called "Tools" since modules like Fenix, Profiling or Notifications, did not really fit in any of the already existing types. Now, the Modules Page within GameCourse has three different sections, as seen in Figure 4.5, which culminated

Figure 4.4: GoogleSheets configuration now.

into a better organization of the modules.

4.2 New Game Elements

In this section, we describe the development that allowed us to incorporate new game elements into GameCourse. A total of three new modules were added and integrated in the system: **Virtual Currency** and **Streaks**, both used in MCP 2021/2022, and **Teams**. Just like any other module in GameCourse, all of them can be enabled or disabled.

When it comes to their actual implementation, the following requirements must be met:

- **Configuration page:** a configuration page must exist and support the functionalities related to the module in question. For instance, in the Virtual Currency configuration page it should be possible for the user to establish the name of the currency, choose which Participations are worth awarding tokens for and how many as well as where they can be spent.
- **New tables in the database:** new tables need to be created to store the all the data related to the module.
- **Rules:** several implementations within the Rule System need to be done so that the system supports these new modules and is able to operate accordingly. For example, functions to perform the preconditions needed that allow Streaks and Virtual Currency to be awarded to a student. Additionally, for Teams, functions to give XP and tokens to each team need to be created. Finally, existing rules will need to be changed and new ones need to be created.

4.2.1 Virtual Currency

The idea behind this new module was to create a concept similar to shopping where students could earn tokens after completing a task. These tokens would then be saved on their wallet and could later

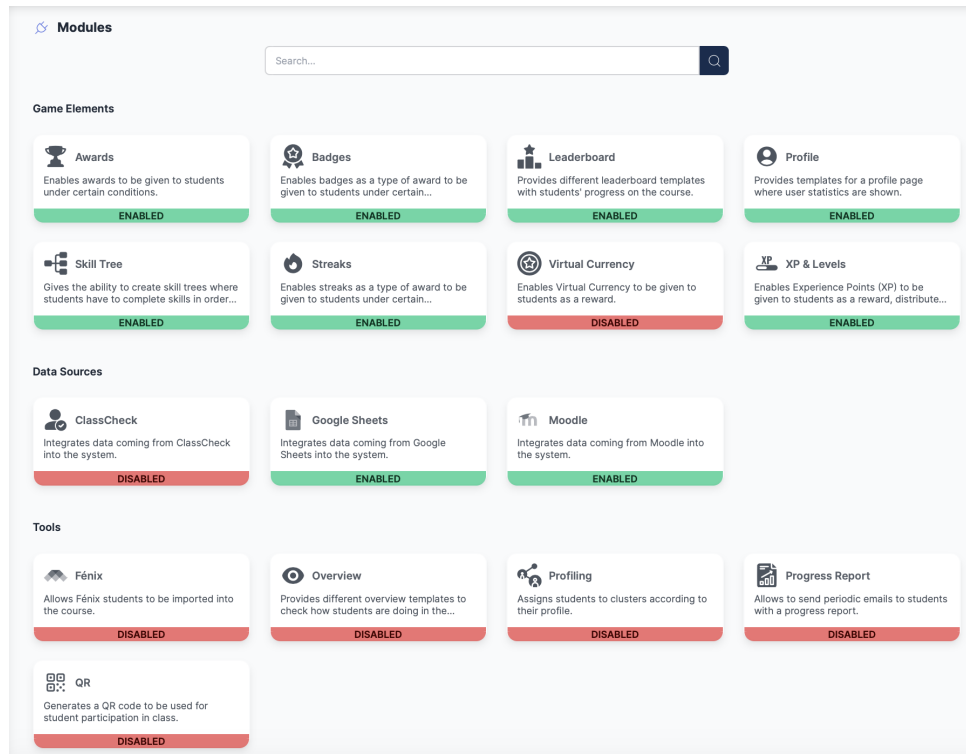


Figure 4.5: Modules page.

be spent to perform certain tasks like, for instance, retrying a skill. One of the first approaches regarding Virtual Currency was understanding how the MCP professors intended on using this new module and how the students would earn and spend their tokens. It was decided that students would earn them by peer-grading their colleagues as well as completing Streaks, a new game element that we discuss later in Section 4.2.2. Furthermore, they could only spend their tokens on retrying a skill or unlocking a wildcard, a type of skill that is not bound to dependencies or levels. In previous iterations of the course, the wildcards would become available at random times and only for a short period of time. Now, they would be available all the time but would cost tokens to complete. Additionally, the cost should increment, according to a specific formula, each time a student retried at a skill. Finally, this module should have a functionality that allows students to exchange their tokens for XP at the end of the course.

The next step was deciding on how we wanted the Virtual Currency to be displayed in the system. Since we wanted this module to be perceived as digital wallet, we decided to display the user's current number of tokens at the top bar right next to the user's name as well as in user profile page. Moreover, since the skills would have a cost, we decided that the amount of tokens needed to be spent on a certain skill should be displayed right next to the it, in the Skill Tree. As such, we designed some prototypes to showcase how we wanted this information to be displayed, as seen in Figure 4.8. Design-wise, we ended up following to these prototypes.

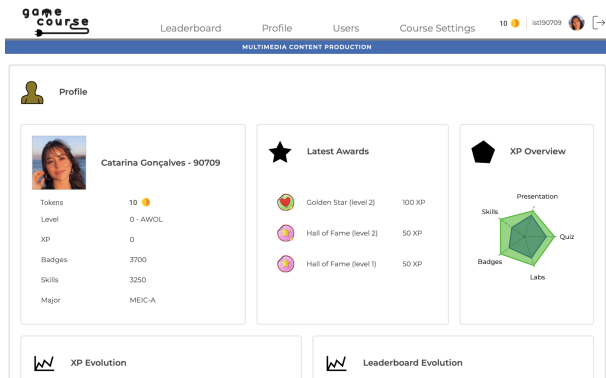


Figure 4.6: Total amount of tokens.

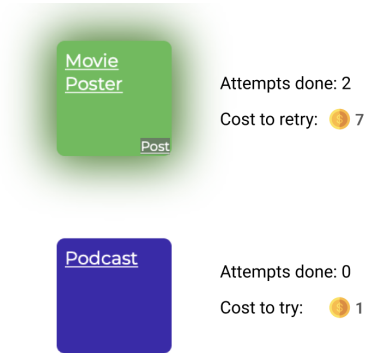


Figure 4.7: Currency in Skill Tree.

Figure 4.8: Virtual Currency Prototype

Once everything was determined and we had an idea of what this module should support, we began implementing it. Initially, since we only had a few weeks to implement this module and knew what was exactly needed for MCP 2021/2022, only three tables were added to the database: the **virtual_currency_config** table, that holds the information regarding all meaningful variables related to the virtual currency, the **removed_tokens_participation** table, that saves the Participations performed by a student that have a cost, and, finally, the **user_wallet** table, that stores the total number of tokens each student has.

We then implemented all the functions that were essential to ensure a well integrated and working module, which included creating a new library that was stored in GameCourse's Dictionary. This was followed by the creation of the module's configuration page. At this stage, a user could only perform four actions (Figure 4.9), like change the currency name, the wildcard initial cost, the minimum rating an attempt at concluding a skill had to have for it to count as a Participation that costs tokens, and, finally, the established increment cost, i.e., the increment that should be added to the skill's previous attempt cost.

Afterwards, we created functions that were going to be used in the rules related to these new modules. Now, to award a skill, the system checks if the student in question has the necessary tokens in their wallet. If not, the skill will not be awarded until the required amount of tokens is available. Otherwise, the award is given and the wallet of the student is updated. As such, we created functions that calculate the new total of tokens a student has in case the skill is completed, and one to update a student's wallet with that value. In addition, this activity and corresponding cost are stored in the **removed_tokens_participation** table so that we can keep track of where the students are spending their tokens. We created the **award_tokens** function responsible for solely awarding a number of tokens to a target, both given as argument. Additionally, a more flexible function that updates a student's wallet according to a specific game element was created. This function receives as arguments a student's id,

game course

Leaderboard Users Course Settings

ist190709 [→]

MULTIMEDIA CONTENT PRODUCTION

This Course

Roles

Modules

Rules

Views

Virtual Currency Configuration

Allows Virtual Currency to be automatically included on GameCourse.

General Attributes SAVE CHANGES

Name Min. Rating for Attempt

Wildcard Initial Cost Increment Cost

Figure 4.9: MCP 2021/2022 Virtual Currency configuration page.

the type of the game element (skill, streak, badge, and so on), that element's name, and the number of tokens to award. Both these functions insert this activity in the **award** table so that we can keep a record of the tokens awarded for each student. Lastly, we created new rules and changes existing ones to include these currency-related functions. Using the **award.tokens** function, we created a rule responsible for giving out initial tokens to the students when they access GameCourse for the first time so that they would not start with an empty wallet. Then, we added the first mentioned functions to the skill-related rules.

Finally, we implemented the functionality that allowed students to exchange their tokens for XP. We accomplished this by creating a function that is responsible for retrieving each student's total amount of tokens and multiplies it by the Tokens to XP ratio, a variable later added and stored in the **virtual_currency_config** table. Subsequently, since this exchange was optional and up for each student to decide, we created a page with a view containing a button that, when pressed, would make the exchange for the student. This activity was also stored as an award in the **award** table so that we could keep a record of all users that exchanged their tokens.

Once the course ended, we began refactoring what was already implemented to make this module more flexible as it was way too tailored towards MCP 2021/2022. The first step was to add two new tables to the database: the **virtual_currency_to_award** and **virtual_currency_to_remove** tables. These tables would allow a user to store the type of a Participations that should be awarded or penalized with tokens, for instance, if the professors of a course want to award the students each time they perform a

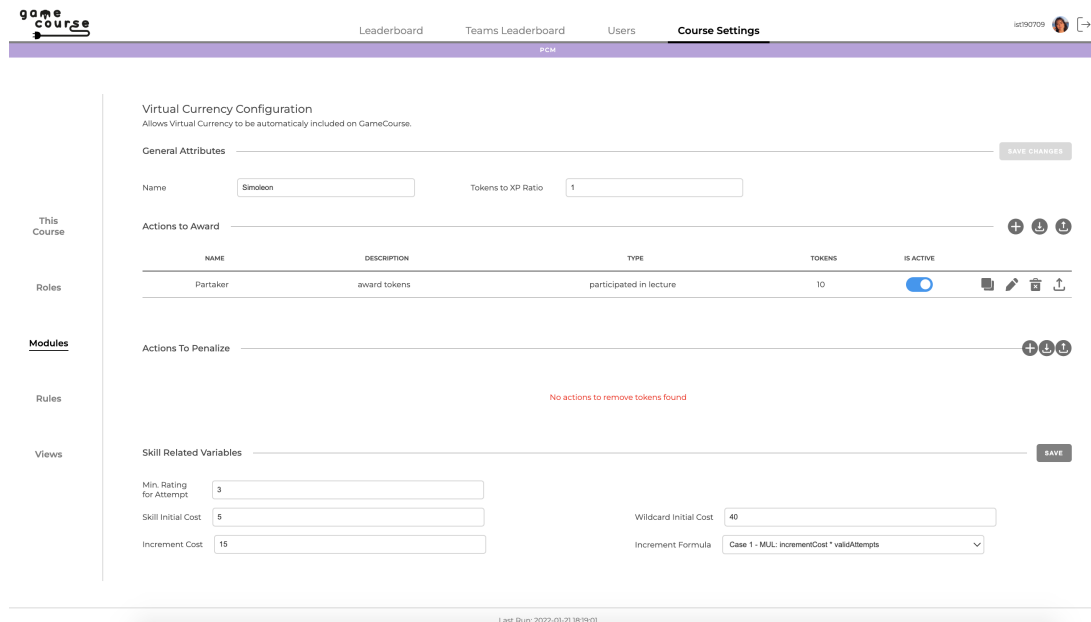


Figure 4.10: Current Virtual Currency configuration page.

peer-grade assessment or participate in a lecture. We did this so that the system is the one responsible for doing these updates on the students' wallets without a user having to create a rule for each case. We also changed the **virtual_currency_config** table by adding a two new columns: one where the increment formula to be used when calculating the cost of a retry at a skill is stored, and another to store the initial cost of a normal skill. Both of these variables were hard-coded for MCP 2021/2022 but, due to their importance, they needed to be variables that a user could freely change for a given course. Now, a user is able to do so, having three possible increment formulas to choose from. Then, since we made a considerable amount of modifications, a few changes were made in the configuration page which culminated into organizing the page into sections, as can be seen in Figure 4.10.

Now, there is a section exclusively for the general attributes, two other sections displaying the types of Participations to award and to penalize, respectively. In those sections a user can perform a set of actions, like create, edit, duplicate and import or export the existing cases. Finally, at the bottom of the page there is a section dedicated to all the variables related to the skills that is only displayed if the Skills module is enabled. To accomplish this, we access the **course_module** table to check whether that module is enabled for the course in question.

4.2.2 Streaks

In order to keep track of the tendency towards consecutive behaviors and awarding students for it, the Streaks module was implemented. Similar to the Virtual Currency module, discussed in Section 4.2.1, our work began with understanding how this new addition was going to be used for MCP 2021/2022

including what was required be displayed within the module's configuration page and the system.

To do so, we needed to grasp the concept of a streak and how it could be incorporated into the system. As such, the main question we had was what would count as a streak: was it solely going to be for consecutive behaviors like receiving the same grade on consecutive skill submissions, or should more parameters be taken into consideration, such as periodicity? In addition, since there were many possible cases for the implementation of a streak, the system needed to provide the essential functionalities to allow a flexible creation and editing process. When discussing with the MCP professors, some possible streaks were discussed which allowed us to better understand what needed to be implemented. For example, a possible streak was one with the duty to track the number of consecutive lectures a student attends. As such, assuming students had to attend three consecutive lectures, each time they attend a lecture it should count as a valid Participation, increasing the streak's participation counter for that student, referred to as streak progression, by one. Once the progression is equal to three (the established value to be achieved), the streak is considered done and should be awarded to the student. Additionally, some streaks could be awarded multiple times, being considered repeatable streaks. As such, with that in mind, we concluded that a single streak should hold the following information:

- **Identifiers:** each streak needs to have a name and description so that users can easily understand what is the streak's goal and, consequently, the behaviors that are going to be tracked.
- **Counter:** an accomplishment count, that can also be referred to as success threshold, must be established so the system knows when to award the streak.
- **Type:** a streak can either be completed by doing a specific activity consecutively or periodically. Simultaneously, a streak can be repeatable. As such, for the system to be able to do the necessary verifications and award it the correct number of times, this information must be stored.
- **Periodicity:** the periodicity that the established behaviors must respect to count as valid Participations.
- **Progression:** the number of valid Participations done. A streak starts at zero and increases by one for each valid Participation a student does. It resets to zero when a invalid Participation is identified. This allows each student to understand how close they are to completing a streak.

With that in mind, we had to make a decision regarding the integration of this module into the system, i.e., how the streaks and their progressions were going to be displayed for each student. Since all the other existing and active game elements were displayed within the Profile page, the same was going to be done for streaks. However, none of the existing game elements could be used as a reference, so a prototype was created to guide us during the implementation. As we can see in Figure 4.11, each streak would have its name, accomplishment count needed, represented by the fire icons, and the rewards

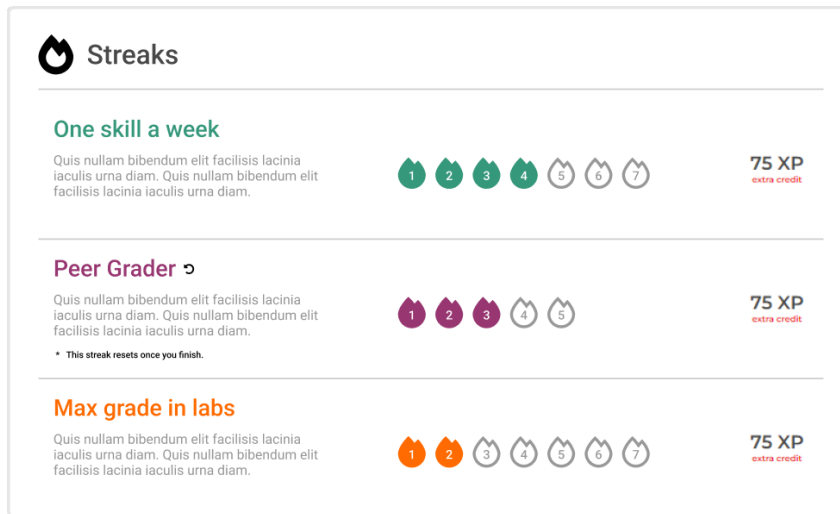


Figure 4.11: Streaks template.

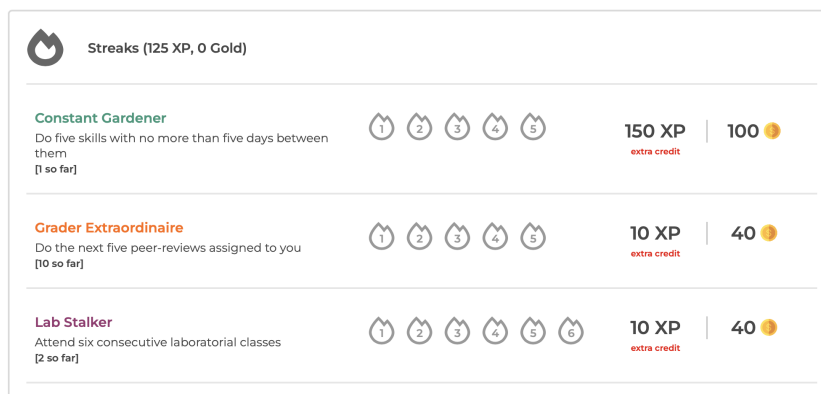


Figure 4.12: Streaks within the Profile page.

displayed. Additionally, a color would be assigned to each streak and, to display the progression, the necessary fire icons would be filled with it.

Design-wise we did not diverge much from the prototype, as can be seen in Figure 4.12. The few changes made were the addition of the tokens to be also given as a reward, positioned right next to the XP reward, and the addition of a counter, placed below the streak's description, that indicates of how many times the student has been awarded that streak. This replaced the revert icon (circular arrow going from right to left), positioned right next to the streak's name, that had the objective of representing a repeatable streak.

At this stage, the configuration page of the module was created, which followed as examples the ones from the Skills and Badges modules since they would be similar at their core. Consequently, there is a section for the general attributes and below that another containing a detailed list of all the existing streaks in the course. Here a user can perform a set of actions, such as creating a streak, editing,

Figure 4.13: Creation of a streak.

duplicating, or removing an existing one as well as importing and exporting the streaks. In the creating and editing streak dialog (Figure 4.13), a user can establish the name, description, color, type of streak, the amount of XP and tokens to be awarded, and, for periodic streaks, its periodicity. Once this was concluded, the functionality to support the previously mentioned actions was implemented.

We started by adding four new tables to the database: the **streaks_config** table, to store the general attributes of this module such as the maximum total XP each student could earn with the streaks, the **streak** table, where all the streaks and respective information are stored, the **streak_progression**, where a record of each streak's progression for each student is kept, and, finally, the **streak_participations** table, where the information related to the Participations made by each student to achieve a certain streak is stored. Then, the necessary functions that allowed all the previously mentioned actions the users could perform were created. Their main duty consisted in inserting new data in the tables, like new streaks, or updating the existing one.

This was followed by one of the most challenging parts of the development of this game element: its integration into the Rule System. There already were a considerable amount of functions responsible for retrieving from the database the data logs to be processed and validated, i.e., the Participations made by a student. As such, we only needed to create the functions that would analyze those logs and check their legitimacy to count as valid Participations. We also needed to take into consideration that these functions must support time verifications for minutes, hours, days, and weeks, which were the available options for the periodicity time that a user could choose from. In addition, the verifications to implement would have to cover all the possible combinations of enabled variables, as can be seen in Figure 4.13.

These could be by enabling only `isCount` or `isPeriodic`, enabling both `isPeriodic` and `isCount`, or both `isPeriodic` and `isAtMost`.

- **isCount**: only enabling this variable would result in the creation of a consecutive streak. As such, the system only needs to verify if the data logs are indeed consecutive. A good example of this is a streak where the student needs to *attend four consecutive lectures*.
- **isPeriodic**: in this case, a simple periodic streak is created and, consequently, the established periodicity is to be checked between each one of the data logs. An example of a `isPeriodic` streak is *doing a total of three submissions, one every five hours*, where the system has to go through all the data logs in question and check if the difference between their timestamps matches the established periodicity.
- **isPeriodic** and **isAtMost**: here the periodicity is dealt with as a sort of deadline a student must respect, like *doing a total of four submissions with no more than 6 days in between*. As such, the periodicity checks are not as strict as in the above case.
- **isPeriodic** and **isCount**: a good example of this combination in a streak would be *doing five submissions in a week*. The main difference from the examples above is that here the system only has to check if the five submissions were indeed made within a week's time. Therefore, only the timestamps between the first and last log need to be compared and the submissions counted.

All the functions created to enable the above verifications receive as arguments the data logs to be processed and validated, i.e., the Participations made. However, when creating these functions we had to take into consideration how the data logs are stored in the database. Consecutive Participations are not simple consecutive lines in the **participation** table for a student but rather consecutive Participations of a certain type and, for some cases, with a specific filter applied. For instance, logs related to lecture and lab attendance are external data inserted into the database when the GoogleSheets module is enabled. As can be seen in Figure 4.14, the only way to check whether these Participations are consecutive is by using the description since it holds the number of the lab/lecture (1, 2, 3, and so on). The same happens for the quiz grade Participations where we must also check the description using the number that comes after "Quiz " to verify if the quizzes are consecutive. As such, the data logs received must be sorted by ascending number in the description.

On the other hand, to support the time verifications mentioned, we used Python's **datetime**¹ module and the **timedelta** function, which allowed us to easily retrieve the duration between two dates or times. This way, the functions created only have to receive the streak's name and Participations made. The former allows us to retrieve the streak's information from the database so that the function is able to

¹<https://docs.python.org/3/library/datetime.html>

id	user	course	description	type	post	date	rating	evaluator
83	7	2	1	attended lecture	NULL	2022-06-10 18:51:31	NULL	6
84	7	2	2	attended lecture	NULL	2022-06-14 16:34:19	NULL	6
85	7	2	3	attended lecture (late)	NULL	2022-06-17 20:46:28	NULL	6
86	7	2	1	attended lab	NULL	2022-06-09 15:12:37	NULL	6
87	7	2	2	attended lab (late)	NULL	2022-06-13 18:29:49	NULL	6
88	7	2	3	attended lab	NULL	2022-06-16 21:08:10	NULL	6
89	7	2	Quiz 1	quiz grade	NULL	2022-06-10 13:31:47	1000	6
90	7	2	Quiz 2	quiz grade	NULL	2022-06-14 15:45:09	1000	6
91	7	2	Quiz 3	quiz grade	NULL	2022-10-22 19:21:41	475	6

Figure 4.14: Examples of consecutive Participations.

perform the periodicity checks. Moreover, the `isRepeatable` variable could be enabled and added to any of the above combinations, since its only purpose is to let the system know that students are allowed to receive the streak award repeatedly. As such, once a student is awarded a repeatable streak, its progression resets to zero. For this to be possible, the Participations made that made the award possible are stored in an already existing table in the database, the **award_participation** table, that serves solely for this purpose, as can be seen in Appendix A. This guarantees that only a single streak is awarded by a set of Participations, and those same Participations will never be used for another award of the same streak.

All of this development was followed by the implementation of other functions that would allow extra verifications to the already processed and validated logs. A good example of this would be *getting the maximum grade in four consecutive quizzes*. In this case, besides the logs having to be consecutive, they should have a specific rating. We decided that separating each type of verification into different functions was the best approach since this allowed the users to have the freedom to write the rules of the streaks as they pleased.

At this stage, we believed that all the necessary functions and verifications were created and being made, since the data logs contain all the information needed, such as their description, rating, and the timestamp from when the log was stored in the database. Nonetheless, an exception emerged due to the "Constant Gardener" Streak, created for and used in MCP. This streak is awarded when students *do five skills with no more than five days between each one*. However, the interpretation of what are the verifications needed may vary. For the MCP 2021/2022 professors, this should not be interpreted as simply submitting an attempt at completing a certain skill since it could result in students submitting below-average posts to complete the streak. As such, in this case, doing a skill meant completing it, i.e., getting a rating greater or equal to three (the minimum required) in the submission made. However, we could not use the graded logs for the periodicity checks since these logs are inserted in the database once the professors rate the student's post. As such, the timestamp used for verification would not be the one from when the students submitted their attempt at that skill.

To solve this issue, for each graded skill Participation (logs of *graded post* type), its corresponding

id	user	course	description	type	post	date	rating	evaluator
566	42	1	Re: Course Logo	peerforum add post	mod/peerforum/discuss.php?id=126&parent=6409	2022-10-25 12:47:00	NULL	NULL
567	42	1	Skill Tree, Re: Course Logo	graded post	mod/peerforum/discuss.php?id=126#p6409	2022-10-25 12:48:38	4	6

2 rows in set (0.001 sec)

Figure 4.15: Example of the Participations needed to award Constant Gardener Streak.

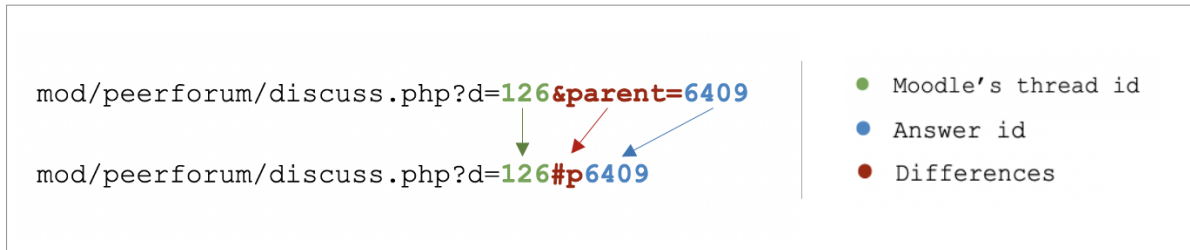


Figure 4.16: Common information between a *graded post* and *peerforum add post* Participations.

submission Participation (logs of *peerforum add post* type) needs to be retrieved so that the correct timestamp is used. Figure 4.15 shows an example of the Participations pairs needed to award this streak. As can be seen, the *description* and *post* columns are the only ones that can be used to guarantee that those Participations are associated with each other. The *description* column always contains the name of the Skill, with the only difference being that the *graded post* Participations associated with skills have a "Skill Tree," in the beginning. The *post* column always contains Moodle's thread id and the id of the answer given in, i.e., the id of the post submitted by a student. Both these columns allow us to correctly retrieve the respective *peerforum add post* log, as can be seen in Figure 4.16 to then perform the time verifications. As such, since we had already created a function that covered all the verifications needed for periodic streaks, we decided it was best to create a function solely responsible for this specific case where two types of logs are needed. This way, by separating the functions, the users are free to choose the function that best suits their interpretation of "doing a skill" when creating a rule.

Additionally, there was one more MCP 2021/2022 Streak that revealed the need for additional functions. The "Grader Extraordinaire" Streak required the students to *do the next five peer-reviews assigned to them*. Moodle offers peer-grading assessment where a set of students are assigned another student's submission, having a limited time to grade it and explain the criteria and reasoning behind that grade. This type of Participation is inserted in the **participation** table like any other Moodle data, when the module is enabled. However, this activity is not mandatory and only the peer-grade assessments made are stored in GameCourse's database, which would not allow us to check whether the students were performing consecutive peer-grade assessments. As a result, for this streak to work, we had to use Moodle's and GameCourse's database simultaneously. We had to analyze the tables within Moodle's database to check if the information needed for this streak was even stored. Two tables were found: the **mdl_user** and the **mdl_peerforum_time_assigned** tables. The former contains a column *id* where

the students' moodle id is stored and another named *username* containing the students' usernames, values that are also stored in GameCourse's *auth* table. This made it possible to identify the students as GameCourse users. The latter contained all the information we needed regarding each student's peer-grading activity. It had a column containing the students' Moodle id, one containing a timestamp of when the assessment was assigned, one named *ended* containing only boolean values whether a student had done that peer-grading and, finally, a column containing information regarding whether that peer-grading had expired or not.


At this stage, we had to implement the functions that would allow this Streak to work as expected. First, we created a function that received as argument the students GameCourse id and would return its username, so that we could later use it to find the student's Moodle id. Then, a different function responsible for retrieving the data from the **mdl_peerforum_time_assigned** table in Moodle's database was created. The query made only retrieves data from the columns mentioned above and orders it by ascending timestamp. Then all that remained was verifying if the peer-grading assessments made were consecutive. To do so, we created a function that receives as argument the data retrieved and verifies the *ended* column log by log to see if it has been done or not, which is sufficient since the logs are already order by date.

Finally, to conclude the development of this module, the rules needed to award each on of the streaks that were going to be used during MCP 2021/2022, were created. Depending on what the streak was (periodic, countable or both), the functions used in the rules would vary. However, all of them start by retrieving the logs from the database. If there are logs of that type for that student, those logs are given as arguments to the functions responsible for validating them according to the streak in question. Then, inside the then clause, the *award_streak* function is called.

4.2.3 Teams

Last, but not least, the Teams Module was created. Implementing a collaborative game element was bound to improve the diversity of elements in GameCourse since most of them are for individual practices like skills, streaks and badges. In addition, we also wanted this module to provide a leaderboard view specifically for teams. This leaderboard should display some of the team member's information as well as the team's number, XP, and the total number of awards given for a certain game element, like the total amount of badges or streaks earned. As such, we created a prototype of how we wanted this view to look like, as can be seen in Figure 4.17.

Design-wise, we ended up not diverging much from this prototype. As shown in Figure 4.18, the main difference from the prototype is the existence of a Members column instead of the four member-related columns. To accomplish this, we created the necessary functions that have the duty to retrieve the information that is going to be displayed. As such, the created functions retrieve all the teams of

 Team Leaderboard











#	Photo	Num	Name	Major	Experience	Badges
1		12345	Sofia Correia	MEIC-A	850 XP 	37 out of 66 
		12346	Ana Catarina	LEIC-T		
		12347	Miguel Silva	MEIC-A		
2		12348	Paulo Gonçalves	MEIC-T	500 XP 	44 out of 66 
		12349	Francisco Neves	MEIC-T		
		12350	Diogo Maria	MEIC-A		

Figure 4.17: Team Leaderboard prototype.

the course, their team members as well as the respective tokens and XP. Additionally, we also created functions that are responsible for retrieving the number of badges and streaks a team has earned since it could be of use in the future.

When it comes to the design of the page where the user can create or edit a team, we only need the necessary fields to write the team's name and number as well as a section to search course users and add or remove them as members of that team. Since the Users Page within a course contained a similar functionality, i.e., we can add a new user by searching the existing users in the system, we just needed to reuse the components in question and adapt them to the Teams Module. Thus, a prototype was not crucial and we did not produce one.

To integrate teams into the system, four new tables were added to the database. We store the teams' general and configuration variables in the **teams_config** table, all the teams are stored in the teams table while the respective team members are stored in the teams_members table. Finally, the XP and tokens of each team are stored in the **teams_xp** and **teams_wallet** tables, respectively.

To encapsulate the module's functionality, its configuration page was created. In this page, there is a section for general attributes where a user can establish the maximum number of elements in a team and also enable team names. In addition, if a user tries to change the maximum number of elements and there are teams with a superior total number of members, a warning message will be displayed and it will be impossible to save the changes. Moreover, there is another section that lists all the existing teams, where a user can create a new team, edit existing ones, and import or export them. To import teams, a user has to prepare a Comma-Separated Values (CSV) file that respects a specific format. As can be seen in the import modal in Figure 4.19, there can only be two columns, one with the student's username and the other with the team's number, and their separator must be a semicolon. This way, when the system is reading the imported file, it has all the information needed regarding the student and

game course

Leaderboard **Teams Leaderboard** Users Course Settings ist190709 [→]

PCM

Teams Leaderboard

#	Members			Experience	Level
	Photo	Major	Name		
1		MEIC-A	Catarina	3100 XP	3 - Aware of your surroundings 900 for L4 at 4000 XP
		MEIC-A	Diogo		
		MEIC-S	António		
2		MEIC-A	Ana Sofia	2500 XP	2 - Self-Aware 500 for L3 at 3000 XP
		MEIC-A	Paulo		
	Photo	Major	Name		

Last Run: 2022-01-21 18:19:01

Figure 4.18: Team Leaderboard.

the team it should belong to. There are two ways a user can choose to import teams: by updating already inserted data or by ignoring duplicates. The former allows the system to update the team members of any team to a new one while the latter ignores any changes and simply imports new teams.

On the create or editing page of a team, there is a search bar that displays the name of the selected users, where a user can also type the name of a student. Below that, there is a section that lists all the course users that are not yet inserted in a team and, if anything is typed by a user, that list will be filtered accordingly. Here a user can also establish the team number and name, a field that is only displayed if the user has enabled team names in the configuration page. Since there is a maximum number of

✕

Please select a .csv file to be imported
 The header must be: username;group
 The separator must be semicolon
 The encoding must be UTF-8

Choose file No file chosen

**IMPORT TEAMS
(UPDATE IF NEEDED)**

**IMPORT TEAMS
(IGNORE CHANGES)**

Figure 4.19: Teams import modal.

elements per team, once that maximum is reached, a warning will appear on the create/edit team modal while the button to add an element will disappear (Figure 4.21). This way the user will not be able to add any other elements to the team.

Teams Configuration
Creates a view template with a leaderboard of the students teams progress on the course.

General Attributes SAVE CHANGES

Nr of team members Allow Team Names

#	TEAM NUMBER	MEMBERS	
1	6	Nesuko	MEIC-A 20030
		Tanjiro	MEIC-A 20031
		Zenitsu	MEIC-A 20032
2	8	Catarina	MEIC-A 23460
		Diogo	MEIC-A 45124
		António	MEIC-S 22228
3	22	Ana Sofia	MEIC-A 23456
		Paulo	MEIC-A 23458

Figure 4.20: Teams configuration page.

PCM

Creates a view template with a leaderboard of the students teams progress on the course

Edit Team: X

Select Members:

Team is complete! You'll need to remove existent members to add new ones.

ANA SOFIA PAULO MIGUEL Search..

- António - 22228
- Catarina - 23460
- Diogo - 45124
- Miguel - 23457
- Francisco - 23459

Save Team SAVE

Figure 4.21: Teams editing page.

To finalize the integration of this new module in the system, we implemented rule system functions that could be needed in the future and wrote template rules to be used as references. One of them takes into consideration MCP and its group presentation, awarding the XP earned to each team. Nevertheless, new rules can be easily written due to the suggestion mechanisms of the Rule Editor or they can also be written based on the existing rules that award prizes, grades or game elements to users individually.

As such, students can now be assigned to a team and perform evaluations as one. Using MCP as an example, a course that supports group evaluation, there is the group presentation, as mentioned above, but there are also certain laboratory classes where the evaluation needs to be done by the whole group. However, this grade is given individually to each user of the group. With this new module, this grade can be awarded to the team the users belong to which allows the system to calculate the team's XP evolution and display it in the Team's Leaderboard.

4.3 Rule System Improvements

In this section, we describe the development made within the Rule System. We start by explaining the preliminary work accomplished as well as all the implementations and changes done to improve the system's performance, followed by a description of the work done to allow automatically generated rules.

4.3.1 Preliminary Work

As previously mentioned, Rules are a text-based rule written in Python that has preconditions to be met before executing the defined action. The actions are the effects the rule should have, usually rewards stored in the **award** table. The preconditions can include anything a user wants that rule to check, usually verifications regarding the Participations the student has done, all stored in the **participations** table, that are required for the awards to be given. Additionally, some preconditions check whether a certain action has been performed, i.e., if a certain award has been given.

With that being said, our work within the Rule System began while the MCP 2021/2022 course was being taught. Once the students started to do skills that had dependencies, the Rule System would not award these skills even if all the preconditions of the rule were met or it would crash trying. For instance, to complete the "reTrailer" Skill, a student must first unlock both the "Looping GIF" and "Publicist" Skills. Using this example, to award a dependant skill like "reTrailer", its respective rule checks, through the **rule_unlocked** function, if the rules responsible for awarding the "Looping GIF" and "Publicist" Skills have been fired since this would mean that the awards for those skills were given. However, even though this function had worked in the previous iteration of the course, it no longer did. As such, the system would not award any skill with dependencies which was a big concern considering there were 18 different dependent skills in the Skill Tree, as can be seen, as can be seen in Figure 3.5 .

It should be noted that the course was taught during a seven-week period and, due to the complexity of the Rule System, understanding how it worked to then be able to properly locate the root of the problem, would take more time than we had to spend on solving this issue. As such, we decided that, for the course to properly work in the system, our best approach was to implement a new function to

```
rule: Presentation King
INACTIVE
# Present your thing, be the best
#   lvl.1: Have the highest grade in the presentations

when:
  logs = GC.participations.getAllParticipations(target, "presentation king")
  nlogs = len(logs)

  # Compute the level of the badge that the student deserves
  lvl = compute_lvl(nlogs, 1)

then:
  award_badge(target, "Presentation King", lvl, logs)
```

Figure 4.22: Example of an inactive rule.

replace the existing one. The quickest way to solve this issue was by creating a function that would access the **award** table to check if the skills in question had been awarded to that user.

Then, while exploring the system to check if there were more bugs related to the Rule System, we discovered that deactivating a game element, like a badge or a skill, would not deactivate the rule associated to it and responsible for awarding it. This meant that the rules of the deactivated game elements would still be run by the system, which should not happen. A rule is not active when it contains the word **INACTIVE** right below its name, as can be seen in Figure 4.22. Therefore, when a user clicks the toggle button responsible for activating or deactivating the game element, a call to the function responsible for adding or removing the **INACTIVE** tag from the rule is now made.

As previously explained, the rules are grouped into separate files, each for a specific game element, while there is also a single file to store rules that do not belong in the other files since they award grades or prizes instead of a game element. Consequently, the function receives as arguments the type of game element (streak, skill, badge, and so on), the current state of the element in question (active or inactive) as well as its name, which is usually the same as the rule. The former allows the system to know which of the files to open and read while the latter helps the system find the rule within the file. Then, once the rule is obtained, the **INACTIVE** tag is either removed or added which depends on the status received as an argument. As such, when a game element is now activated or deactivated, the same is applied to its rule.

4.3.2 Performance Improvements

A problem that appeared throughout the MCP 2021/2022 course was the Rule System's performance, more specifically, the time it took to run. With more data being inserted into the database every time it ran, this problem would aggravate, and as we got closer to the end of the course, the system would take at least an hour to finish running or would crash trying. We started by deactivating the rules that were no longer needed to decrease the number of verifications done each iteration, thus, decreasing the time

it took. At the time, this was a sufficient approach but once the course ended we began exploring the problem. Also considering the behavior of the Rule System throughout MCP, the main issues we found were:

- **AutoGame would not restart on its own:** if an error occurs during the execution of the system, the script does not reach the part where it sets itself as not running in the **autogame** table, as explained in Section 3.2. As such, if the system ran after an error had occurred, it would assume that it was still running and would be stuck in a loop of unsuccessfully trying to run. The only way we could solve this was by manually changing, in the **autogame** table, the respective value that gives this information to 0.
- **Connections to the database:** each function implemented in the Rule System and used in the rules would create and close a connection to the database. This meant that each time the system ran, several connections would be made which introduced a considerable overhead.
- **Repeated MySQL queries:** throughout the code, the same exact queries would be executed several times. The results from several of these queries remain unchanged during the execution which means that the same query same query done multiple times has always the same outcome. This culminates in several unnecessary repeated accesses to the database.

Nevertheless, without making a deeper analysis, we could not say that these issues were the main causes of the poor performance of the system. To grasp the underlying problem and find out where the system was wasting the most time, profiling was in order. We first populated the database with data that would activate most rules and so, specific logs were inserted in the **participation** table that would fire them. Then, we just needed to obtain the time each one of the preconditions and actions of the fired rules took to execute. To do so, Python's time and logging modules were used. As mentioned before in Section 3.2, the Rule System has a logging mechanism that writes, for each course and into separate files, all the information regarding the system's execution. As such, the time obtained was written on the logs file of the course in question. This way all the desired information, like rule name, precondition or action and respective time, was orderly stored in the same file for later analysis. Then, to try and replicate as close as possible the scenario of running the Rule System when a course is active, we ran it on the virtual machine that hosts GameCourse.

We discovered that any function stored in the Dictionary responsible for retrieving data was taking the longest to execute such as, **getAllParticipations** from the participations library or **wildcardAvailable** from the skillTrees library, with some of them taking almost five seconds. In addition, the functions responsible for awarding a particular game element, grade, or prize were also very time consuming, with the **award_skill** and **award_streak** taking the longest. These functions are of extreme importance to the system since most rules within a course rely on them to gather all the specific data logs for a

certain target and produce the desired and expected outcome like, for instance, awarding a streak after a target has met all the established preconditions. As a consequence, rules that were dependent on these functions ended up taking almost eight seconds in total to execute which, for a course with around 100 users, would culminate in an excessive amount of time to run the Rule System, something that we had already experienced in MCP 2021/2022.

To fix the Rule System's performance, we started by replacing all the multiple connections made to the same database with a single connection that is made once the system starts running. This was accomplished by creating a class called **Database** that creates the connection in case there was not one already and deals with all the actions related to the database. In addition, on the script responsible for running the Rule System, we added a **finally** block to the already existing *Try-Except* statements where the connection is closed since, by the time the script reaches this part, the connection is no longer needed.

At this stage, another important change made to the Rule System was creating what could be called a **data broker**, responsible for storing the queries executed and the respective results. Most queries done in the Rule System's functions, especially the ones that would retrieve information regarding a certain game element, should be executed only once since their results would remain unchanged while the system was running. As such, with this addition to the system, if a certain query had already been executed, its result could just be obtained by accessing the data broker, sparing the system from unnecessary query executions. Since we wanted our system to run as fast as possible, time complexity was the decisive factor when choosing the data structure where we would store this information.

Just by looking at Table 4.1, we can see that almost all operations have the same time complexity for Python's Lists and Dictionaries², except for the **x in s** operation (lookups), one of the operations that we would use the most since it allows us to check whether a specific query has already been executed and stored or not. The lookups for dictionaries have a constant time complexity, $\mathcal{O}(1)$, while lists have a linear time complexity, $\mathcal{O}(n)$, which makes dictionaries faster. Python Dictionaries are collections of key-value pairs implemented using hash tables, which means that each key has to be an immutable object, like a number or string, that can be assigned a hash value³. Consequently, we decided to implement our data broker using a dictionary where the query itself is the key and the value is the result of executing that query. We created two dictionaries so we could distinguish between queries made for particular students and those made for specific courses. Additionally, both of these dictionaries were nested so that we could group the data by user or course. In this manner, the course or game course user id would serve as the keys of each dictionary and the query-result pairs would serve as their values.

The implementation of the data broker was followed by the creation of the necessary functions that allow this new addition to properly work in the system. We created a function that receives as an argu-

²<https://www.geeksforgeeks.org/complexity-cheat-sheet-for-python-operations/>

³<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>

Table 4.1: Time complexity in Python data structures.

Data Structure	Operation	Average Case	Amortized Worst Case
List	Append	$\mathcal{O}(1)$	$\mathcal{O}(1)$
	Get Item x in s	$\mathcal{O}(1)$ $\mathcal{O}(n)$	$\mathcal{O}(1)$
	Iteration	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Dictionary	Insert	$\mathcal{O}(1)$	$\mathcal{O}(n)$
	Get Item x in s	$\mathcal{O}(1)$ $\mathcal{O}(1)$	$\mathcal{O}(n)$ $\mathcal{O}(n)$
	Iteration	$\mathcal{O}(n)$	$\mathcal{O}(n)$

ment the id, either of a student or a course, and the query to be executed. As such, it is responsible for accessing the desired dictionary and checking whether that query has already been stored for that id. If it has, it will return the respective result. Otherwise, the query is executed and a call is made to another newly created function that has the duty to store that query and its respective result in the correct dictionary. Initially, when creating both functions, the lookup operation was used within a conditional statement (*if-else*). However, these conditional statements can be replaced by *Try-Except* blocks, usually used to handle Exceptions. Additionally, depending on the exception to be handled and how many times it will be thrown, these blocks are comparatively faster than the explicit *if-else* statements⁴. Consequently, since the only exception that could be thrown in our function would only be thrown once when that key did not exist within the dictionary, we decided to replace the conditional statements with a *Try-Except* block where we try to directly add the query-result pairs to a certain id. If an exception is thrown, we first add the key to the dictionary and then the query-result pairs.

Afterwards, we began replacing the existing code that executes the queries with the data broker. Whilst doing so, we had to be careful to only store queries that we are certain the results remain unchanged throughout the system's execution. For instance, queries that retrieve a student's total amount of tokens should not be stored since this value can be changed depending on the rules that are fired. We then proceeded to check if these changes had any impact on the system's performance. A reduction in the execution time did happen although the functions from the Dictionary were still very time-consuming. Therefore, we decided to tackle this problem further and analyze how they were implemented.

The functions in question were mainly used to retrieve specific Participations from the database or check if a certain award had been given. Consequently, the queries to execute are written based on the received arguments, that are the course id, the student's GameCourse id, type of participation to fetch, and any other variables that can be used to further filter the results. As such, these functions had a considerable amount of conditional statements to check what arguments were given so the queries could be correctly written. Only then, would the query be executed, which could have an impact on the execution time. Additionally, the Rule System has to access the respective library in the GameCourse

⁴<https://www.geeksforgeeks.org/try-except-vs-if-in-python/>

Dictionary to use and run these functions. As such, both of these combined could be the root of the problem we were trying to solve. Once we reached this conclusion, we decided to replace the Dictionary functions with new Rule System ones that access the database directly, without having to access the Dictionary. To see what were the consequences of this change, the following functions were created to replace the existing ones:

- **get_logs**: responsible for retrieving all logs. As arguments, it receives the target and type of the participation.
- **get_graded_logs**: has the duty to retrieve graded logs. As argument, it receives the target, the ratings the logs should have and a boolean that is responsible for including graded logs only related to skills.
- **get_graded_skill_logs**: is a function solely focused on graded skill logs. As arguments, it receives the target, the desired ratings and, optionally, the skill name in case the user only wants the logs of a certain skill.

In addition, since the award functions were also taking a long time to run, especially the function responsible for awarding streaks, we decided to analyze them to check whether a refactor of the code was in order. To do so, we must take into consideration how the rules work: all the preconditions in the *when* clause must be met for the actions in the *then* clause to be performed. All the award functions are an action that is triggered if the preconditions are met, as such, they should simply award the streak with a few verifications being needed. However, we realized that this was not the case. In the **award_streak** function, some verifications regarding the streak's progression were being made, which should not happen. Additionally, the number of streaks to award was also calculated inside the function, which was unnecessary. Since streaks with the `isRepeatable` variable enabled can be awarded several time, this verification checks whether the already processed and validated data logs have already been used to award the streak once. As such, the system checks if they are already stored in the **award_participation** table, where all the participations associated with an award are stored. We created functions responsible for performing those verifications, removing them from **award_streak**, and added the new functions as preconditions in the streaks' rules. Consequently, the refactored **award_streak** function also receives as an argument the total number of awards to give, and the Participations that those awards possible. Now, the function only needs to perform a single verification, which is to check whether the streak is repeatable, to then award it multiple times or not.

Furthermore, refactoring was also done to the already existing functions that dealt with the processing and validation of the Participations required for the streak. Initially, a single function was used to perform all of the verifications needed. However, this required a substantial amount of conditional statements which, as previously stated before, could have a negative impact on the performance of the

Table 4.2: Execution times of firing certain rules for a student before and after improvements.

Rule	Description	Execution Times Before (s)	Execution Times After (s)
Publicist	Awards Publicist skill if preconditions are met.	1.0914247	0.4644244
reTrailer	Awards reTrailer skill if preconditions are met.	0.8382614	0.6235328
Foley	Awards Foley skill if preconditions are met.	0.9258363	0.6006000
Practitioner	Awards Practitioner streak if preconditions are met.	2.5239008	0.4984679
Sage	Awards Sage streak if preconditions are met.	2.5239008	0.6807893
Stalker	Awards 2 Stalker streaks if preconditions are met.	7.6509702	1.2063188

system. Therefore, the function was split into three new ones, each with a specific purpose: two for different types of consecutive checks and one for periodicity checks. The **check_consecutive_logs** function checks based on the type of the logs received whether they are indeed consecutive and the **check_periodic_logs** function checks if the logs received respect the periodicity of the respective streak, whose name is also received as an argument since it allows use to retrieve all its information from the database. Finally, the **check_consecutive_peergrading_logs** function is only to be used when dealing with peer-grading logs from Moodle's database. Since these logs are different from the ones stored in GameCourse's database so are the verifications needed, as such, a separate function was in order. This was followed by editing the rules and replacing the functions with one of the ones mentioned above.

Once all these changes were implemented and working as expected, we had to run the Rule System again to check if they had any impact on the execution times. Since the database was already populated, we only had to remove the previously awarded elements to then run the script. As it can be seen in Table 4.2, there was a significant reduction in the execution times of the rules. For instance, running the "Practitioner" rule took, approximately, one-fifth of the time it did before, which is a substantial speed-up. Similar results happened as well for the other rules which proves that these changes did indeed have a positive impact in improving the Rule System's performance.

Finally, there was only one issue left to analyze: AutoGame not restarting on its own. To solve this problem, we started by replicating the conditions of the errors we had previously identified and testing how the system would respond when we forced it to crash. This mainly consisted in manually running the script and abruptly killing the respective process before it finished running. Then, we would run the script once more to check the AutoGame status and, as expected, it would get stuck and would not run


```
=====
[2022/04/29 16:00:13] : AutoGame started running.
=====

=====
[2022/04/29 16:03:05] : AutoGame finished running.

Autogame ran for the following targets:
Unordered:      [130, 209, 368]
Ordered:       [130, 209, 368]
=====
```

Figure 4.23: Log file sections and separator.

until we manually changed the **autogame** table.

To solve this, we took advantage of the Rule System’s logging mechanism mentioned in section 3.2. The log file has an established separator for a user to be able to visually differentiate between the information written. As can be seen in Figure 4.23, a separator is written before and immediately after each block of information. However, if an error had occurred, the script would not reach the part where the last separator is written and, as a consequence, the file would not end with one. As such, to know if an error occurred or not, the system now analyzes the respective log file to check whether it ends with a separator or not. To achieve this, we created a function, **checkAutoGameStuck**, that opens and reads the last line of the file to then compare it with the established separator. It returns a boolean value depending on the result. Then, we added a conditional statement within the function responsible for executing the AutoGame script, where a call to **checkAutoGameStuck** is made. This way the system resets AutoGame if needed and only then is the script executed, which guarantees that it will not be stuck in a loop of unsuccessfully trying to run due to an error that had previously occurred.

Additionally, we noticed that the code responsible for starting the server socket, mentioned in Section 3.2, was incomplete. The blocks of code that handled the Exceptions thrown would just write the error in the log file, without closing the connection to the socket that was made at the beginning of the script. At this stage, we decided it was best to check the dimension of this problem by accessing the system’s virtual machine and checking how many connections to the sockets had not been closed yet, and found that there were 69 sockets in TIME WAIT. The incompleteness of the code was most likely the explanation for why there were so many sockets to the same server in this state. This could have an impact on the system’s performance as well. Consequently, we closed the connection to the server socket within all the **except** blocks of the Try-Except statements. This way, if an exception is thrown, the connection to the server socket is closed.

4.3.3 Automatically Generated Rules

Another important implementation made and integrated in the system was the automatic generation of rules. For this functionality to properly work, rule-templates for each game element must be created and stored inside a folder within each module. Additionally, there can be more than one template, each containing the functions to be used as preconditions and actions to perform that cover that case. Since they only serve as templates, there are placeholders for the system to change according to the game element created. This way, a complete rule is automatically created based on that element's attributes and is ready to be used. However, as we will discuss later on this section, there are some cases where the system does not have all the information to replace all the placeholders, like retrieving Participations of a specific from the database. Consequently, this requires a user to manually replace the unchanged placeholders.

This functionality, although already implemented for the Skills, as discussed in section 3.2, was incomplete and faulty. If a user edited any skill and did not remember to access the rule editor or the corresponding rules text file to manually change the rule according to the modifications made, it would remain unchanged. As a consequence, the results of firing this rule would most likely be wrong and produce, for instance, unwanted awards. As such, **our work aimed to improve this functionality by fixing what was already implemented, add the missing functionalities and apply it to other modules.**

We started out by refactoring some functions that were not working as expected, for instance, the function **changeRuleStatus**, that has the responsibility of activating or deactivating a rule, was not properly working and, in some cases, would not activate an inactive rule. Within the function, there was a bug that would not allow the system to acknowledge the existence of the INACTIVE word to then remove it. After we fixed this bug, we began implementing the functionality that edits the rule according to the changes made by a user in the editing page of a game element.

The editing of a rule could either be done by replacing it as a whole with the one created based on the updated data or the system could edit the rule to only rewrite the parts that needed to be changed. The first option only made sense if the user changed all or most of the game element's fields, which, would be almost the same thing as creating a new element from scratch. In addition, we must also take into consideration that only a few fields actually have an effect on the rule, like the name or dependencies of a skill. As such, the latter was the most adequate approach to be implemented. The data of each game element is stored in a specific table or tables within the database. For instance, each badge is stored within the **badge** table while each skill is stored within the **skill** table and its dependencies are stored in the **dependency** and the **skill_dependencies** tables. Once a user saves the changes made, the system makes a call to a function responsible for updating the necessary tables. As such, the system has to check which of the stored fields do not match the ones in the incoming updated data so that only the

necessary parts of the rule are changed. To accomplish this, we decided to retrieve the already stored data and save it in different variables before it was updated, so that the system is able to compare each field's previous value with its current one.

As previously mentioned, **only a few fields have an effect on the rules**. When it comes to the skills, the rules would only be affected if the skill name was changed or if the dependencies were modified. As such, at the end of the function that is called when a user saves the changes made, being responsible for editing that skill, two verifications were added: one for the name and the other for the dependencies. The checks for the name were done using string comparison, however, for the dependencies, we first check whether the number of dependencies has changed. If it has not, we then check dependency by dependency. Since these verifications are done separately, if any differences are found, the system calls the functions responsible for making the necessary changes to the rule, i.e., if only the dependencies were changed, only the function responsible for replacing them within the rule is called. Additionally, to keep a record of what was changed, we decided to comment the lines to be changed and then add the new updated lines below, as shown in Figure 4.24. This way, a user can review the previous contents of that rule before modifications were made.

To achieve this, we created a function for each case, both receiving as argument the rule. When the name of the element is changed, its old name is also given as argument so that the system is able to easily find all of its occurrences. Finally, so that the user knows what the rule's previous name was, a commented line with that information is added right below the current name. On the other hand, when the dependencies are changed, we have to go through all the lines to identify the ones to comment. As can be seen in Figure 4.24, the lines related to the dependency checks all start with a specific word, either *combo*, *wildcard*, *skill_based* and *use_wildcard*. Consequently, for each line, the function checks whether it starts with one of those words and, if it does, proceeds to add a Python comment (#) at the beginning so that, if the rule is later fired, the line is ignored.

Once all of this was accomplished for the Skills Module, **we replicated it and applied it to the remaining ones that also required this functionality**. As explained in the beginning of this section, for the rules to be automatically generated, templates need to be created where the fields to be changed are identified by placeholders. These templates are stored in a folder within each module that requires the generation of rules. As such, to all the modules where this functionality was going to be added, a new folder named *rules* was created. Then, for each module, the rule templates had to be created, taking into consideration what the rules might have in common. As such, we had to check the already existing ones for each module to try to find a pattern or determine what was going to be common to all rules. Additionally, there could be more than a single template, so we had to determine how many were needed and what they should include. When it comes to skills, only two types of rule templates are needed: one for normal skills and the other for skills that can be unlocked using a wildcard. The templates are very

```

rule: reTrailer

  when:
    #CHANGED:
    #combo1 = rule_unlocked("Course Logo", target) and rule_unlocked("Movie Poster", target)
    #combo2 = rule_unlocked("Album Cover", target) and rule_unlocked("Publicist", target)
    #combo1 or combo2

    wildcard = GC.skillTrees.wildcardAvailable("<skill-name>", "<tier-name>", target)
    combo1 = rule_unlocked("Publicist", target) and rule_unlocked("Album Cover", target)
    combo2 = wildcard and rule_unlocked("Course Logo", target)
    combo1 or combo2

    skill_based = combo1
    use_wildcard = False if skill_based else True

    logs = GC.participations.getSkillParticipations(target, "reTrailer")
    rating = get_rating(logs)
    rating >= 3

    # NOTE:
    # Virtual Currency Enabled ? Uncomment code below : Delete

    # valid_attempts = get_valid_attempts(target, "reTrailer")
    # valid_attempts > 0
    # (new_total, removed) = get_new_total(target, valid_attempts, rating)
    # new_total >= 0

  then:
    #CHANGED:
    #award_skill(target, "reTrailer", rating, logs)

    award_skill(target, "reTrailer", rating, logs, use_wildcard, "Wildcard")
    # update_wallet(target, new_total, removed, logs)

```

Figure 4.24: reTrailer Rule after changing the skill dependencies.

similar with the only difference being the existence of verifications related to the use of a wildcard. The same line of thought was applied for streaks and badges. For the streaks, we created a more flexible template while the other was to be specifically used for periodic streaks since the preconditions used are different. Then, for badges, a more general template was also created that covered most of the existing badges, and another to be used for badges with a specific variable enabled (the isPost variable), since it requires the use of a specific function to retrieve Participations from the database.

Then, to incorporate this functionality, we created the functions responsible for retrieving the templates and generating a rule when a new element is created. Then, all that remained to implement were the conditional statements that check what needs to be modified in the rule if the respective game element suffers any changes. For badges and streaks, the only modification made that would indeed affect the rule was changing the element's name and there was already a function responsible for doing these changes, as we mentioned above. As such, no new functions had to be created.

However, while skills' generated rules are ready to be used once they are created, the same could be done for badges and streaks. The template rules for skills only have two parameters that need to be changed, the skill name and the tier name, and both templates use the exact same function to retrieve the Participations since, to award the skill, the system only needs to retrieve graded skill submissions. On the other hand, the rules for badges and streaks require the user to specify the type of the Participation

```

rule: <streak-name>
# Replace all <..> with the respective value.

when:
# NOTE:
# get_logs() is the default function.
# CHANGE IT if that's not the correct function to use in this rule:
#   - Get all graded logs -> get_graded_logs(target, minRating, include_skills = False)
#   - Get all graded skill logs with the desired ratings -> get_graded_skill_logs(target, ratings)

logs = get_logs(target, "<participation-type>")

check_periodicity(target, "<streak-name>", logs)
to_award, participations = awards_to_give(target, "<streak-name>")
to_award > 0

then:
award_streak(target, "<streak-name>", to_award, participations, "<participation-type>")
# Virtual currency enabled?
# award_tokens_type(target, "streak", "<streak-name>", to_award)

```

Figure 4.25: Template rule for periodic streaks.

to be fetched. Take the **"Talkative"** Badge for instance, it needs all the data logs of type *"participated in lecture"* and *"participated in invited lecture"* from the **participation** table. As such, we decided that, for these particular game elements, some parameters should not be automatically filled by the system and remain unchanged with a placeholder, like *<participation-type>* (Figure 4.25). As a consequence, all the automatically generated rules for streaks and badges elements are not ready to be used once that element is created which requires a user to access the Rule Editor to replace the unchanged parameters.

To ensure that the user does indeed complete these changes, once a rule is automatically generated, the user is redirected from the current page to the editing page in the Rule Editor, displaying the rule that needs to be finished. This was achieved by integrating my work with the work of my GameCourse colleague, Joana Seginando. We thought about searching for the rule in the respective text file, like what is done when a rule is automatically edited. However, this approach is prone to errors, all it takes is for the rule name not match the game element. As such, we decided to store the rules in the database, just like what is done for the majority of data, which allows us to guarantee that the correct rule is always retrieve. This way each rule could have an unique id that would allow us to easily retrieve the rule and simplify the redirect process. Joana Seginando added three new tables to the database: **rule_section**, **rule_tag** and **rule**, where the system stores information like id, name, description, *when* and *then* clauses as well as the rule's position in its respective text file.

The last change we implemented was adding comments to the template that could further help a user perform any required modifications. As mentioned above, for the badges and streaks, a user has to check the rule and change the default function or its arguments according to desired type of participation. As such, we decided it would be helpful to leave a note within the template explaining that the function used is the default one, i.e., the most commonly used or flexible function, and what replacement options are available. For instance, in Figures 4.24 and 4.25 we can see that the template contains several

commented lines. The first ones aim to explain to the user that the default function to retrieve logs can be replaced, showing them the other functions that can be used. With this we intended to provide more specific suggestions, in addition to the ones the Rule System provides that we mention in Section 3.2. Additionally, the templates also contain commented currency related functions that are implemented in a way that does not require any change to be made. As such, if the Virtual Currency module was enabled, the rules already have the necessary functions that allow that element to have an effect on the student's tokens, only requiring a user to remove the comments. We did this since we cannot guarantee that a user would want these functions applied to all game elements. As a consequence, we decided to keep them commented within each template to allow users to freely decide whether to include them or not.

5

Evaluation

Contents

5.1	MCP 2021/2022	65
5.2	User Tests	68
5.3	Rule System Performance Testing	74
5.4	Discussion	76

In this chapter we will present the evaluation performed over the new game elements and improved Rule System to understand whether we were successful in developing and integrating these changes in GameCourse. We start by analyzing the results from MCP 2021/2022, including the answers given to the final questionnaire students were asked to perform. Then, we explain what tests were done to check whether our changes were well integrated and working as expected. We provide a detailed description of how the user tests were conducted, the rationale for each task and by characterizing the users that participated in the evaluation. This is followed by an explanation regarding the performance tests made in the Rule System. We conclude by providing an in-depth analysis of our results and discuss what we observed.

5.1 MCP 2021/2022

The Streaks and Virtual Currency modules were developed and integrated within GameCourse in time for them to be used during the Multimedia Content Production 2021/2022 course. Before the course began, we created a GameCourse test environment where we could test the modules beforehand to guarantee they were properly working. This was done by testing if the changes made within the configuration pages were having the proper effect in the database as well as verifying whether the new rules were correctly implemented and awarding streaks and tokens.

When it comes to the Streaks, we first had to check whether a new streak was being inserted in the streak table. To do so, we created a streak in the configuration page of the Streaks Module and then verified if the new element had been inserted in the database. Additionally, we did the same thing for editing, deleting, duplicating, and deactivating a streak. At this stage, all the tested functionalities were working as expected, so we created all the seven streaks that were going to be used for MCP, which covered almost all possible cases for a streak with the exception being a simple `isPeriodic` streak. Then, for each streak, we populated the participation table with the Participations required for awarding that streak. Once this was done, we ran the Rule System to check whether the streak had been awarded. Additionally, since we wanted to test all the scenarios that we could imagine in which a streak should not be awarded, we inserted Participations that did not correspond to what a streak required. For instance, for a streak where a student has to have three maximum grades in a quiz, we tested out all the scenarios that would not result in awarding this streak. As such, we added a Participation where one of the grades was not equal to the maximum, which would break the consecutiveness of the logs. While testing this, some errors occurred in the time verifications made for periodic streaks where the comparison between dates needs to be equal to the established periodicity. This happened because, when comparing for days or hours, we were using the full timestamp and, as such, the periodicity would not be verified. We fixed this by using the `timedelta` function from Python's `datetime` module that can be used for calculating

differences in dates as well as date manipulations in Python ¹. With it, the time verifications for the periodic streaks were now working as expected which fixed the forementioned problem.

Then, once the streaks were tested, we moved onto the Virtual Currency Module. Firstly, we tested the configuration page by changing the variables, creating cases where Participations must be awarded or penalized and edited them as well. Luckily, all these changes were being correctly inserted or updated within the respective tables in the database. Afterwards, we had to check if the Rule System functions we had created were working. As such, we firstly tested if awarding a streak was correctly updating the student's wallet. To do so, we deleted, from the database, one of the awards previously given when testing out the streaks, added the functions responsible for updating the students wallet as an action in the streak's respective rule and ran the Rule System to see if the streak and respective tokens were awarded. We verified that the functions were successfully working by checking if an award of type tokens, with its description being the name of the streak, had been inserted in the **award** table for that student and if their wallet had indeed been updated in the **user_wallet** table.

Since the streak-related rules are complex and of extreme importance, it was clear that manually populating the database and running the Rule System to test them was not an optimal approach. As such, **we created a PHP script that the users can simply run to check if the streaks are properly working**. This script covers eight different streaks: the seven used for MCP 2021/2022 and a isPeriodic streak, that was not created nor used for the course. So that the user does not have to worry about the database, the script is responsible for both populating the database to set up for the tests as well as cleaning up afterward, which guarantees that it starts with a clean environment every time it is run. After setting up the environment, the script makes a call to the function that runs the Rule system so that the necessary rules are fired. Afterward, it checks if the correct awards were given.

This was followed by testing the skill-related rules that we had already changed to include the currency-related functions. Now, since most skills had a cost to retry or be unlocked, students had to have a specific amount of tokens for that skill to be awarded. As such, a few preconditions regarding the currency were added to these rules. We had to test two distinct scenarios: one where the students had the necessary tokens to do the skill and another where they did not. To do so, we had to populate the database with Participations to simulate that a student submitted a post as an attempt to complete a skill and a professor had graded with a rating equal or greater than three, which is the minimum rating required for a skill to be completed. Additionally, we updated the tokens in the student's wallet to an amount that was not sufficient to complete the skill, for example, if that skill had a cost of 40 tokens the updated amount had to be less than that.

Then, we replicated this for a different student so that we could cover both scenarios, the difference being that this student had the tokens needed to unlock the skill. Then, all we had to do was run the

¹<https://www.geeksforgeeks.org/python-datetime-timedelta-function/>

Rule System and check whether the skill had been awarded to the student that had the necessary tokens, if those tokens had been removed and if the Participation that fired this rule that resulted if the removal of tokens was inserted in the **remove_tokens_participation** table. Additionally, within the Virtual Currency Module, the user can establish the minimum rating a submission at a skill needs to have for it be recognized by the system as an attempt that cost tokens. The default value for this skill-related variable is three. As such, a third scenario where the student does not receive a rating equal or greater than three was also tested. To do so, we simply removed the award that resulted from the tests mentioned above, updated the rating received to a two and ran the Rule System again. All the scenarios tested had the desired and expected effects which proved the modules were properly working.

By the time the course started, all the functionalities were tested. During the seven-week period in which the course was taught, we were able to gather feedback and improve the developed functionalities by fixing any relevant bugs that were found. Even though we tried to cover all the possible error scenarios, a streak-related bug within the Rule System functions appeared and was pointed out to us by the students. This bug was related to the streaks' progression: **if a student only had one valid Participation for a streak, it was not being counted as a Participation for the progression**. This was due to the fact that, within the code, the system would iterate through the received logs in pairs. As such, if there was only one Participation made, there was no verification possible to be made until a new one was made. Therefore, in the beginning of the functions responsible of the validation of the Participations, we **added a verification regarding the number of Participations received as argument**. If there was only one, we would add it to the progression of the streak since the first Participation is the one used for the first comparison and, as such, is always valid.

Another bug that was caught during MCP was that **the rules were being fired for professors that had previously been students in a different course** within the system. In MCP 2021/2022, there were two professors that were enrolled as students in MCP 2020/2021, a course that, although inactive, still existed in GameCourse and, subsequently, in the database. When we were checking the award table to find any bugs, we detected that the Professors had been awarded with tokens. This meant that the rule responsible for awarding the initial tokens had been fired for those two professors. The root of this problem was that the queries responsible for retrieving the Rule System targets did not take into consideration the existence of several courses within the system. As such, it would just retrieve users with the Student role without specifying the course. To fix this, we had to filter the results by course, adding this verification to the when clause of the query.

Once the course ended, the students were asked to answer a final questionnaire where they could express their thoughts regarding GameCourse and its game elements. The questionnaire contained questions where the students had to rate, in a scale of 1 to 5, each one of the course features on how it made them feel engaged with the course, and how fun they were. As can be seen in Table 5.1, XP, Skill

Table 5.1: Mean, median and standard deviation values of the rating given by students, from 1 to 5, when asked about how engaging and fun were the game elements.

Game Element	Perceived Engagement			Perceived Enjoyment		
	Mean	Median	Std. Dev	Mean	Median	Std. Dev
Badges	3.15	3	1.46	3.4	4	1.37
Leaderboard	2.5	2	1.73	2.98	3.5	1.60
Levels	3.35	4	1.37	3.3	3.5	1.3
Skill Tree	3.97	4	1.21	3.98	4	1.22
Streaks	3.15	3	1.17	3.2	3.5	1.34
Tokens	2.85	3	1.23	2.98	3	1.27
XP	4.1	5	1.19	3.83	4	1.26

Tree, and Levels were considered moderately engaging with an average score of 4.1, 3.97, and 3.35, respectively. Followed by Badges, and Streaks, both with an average score of 3.15. This means that Streaks were considered somewhat engaging which can mean that they are indeed having a positive impact in engaging the students but there is also room for improvement. Tokens, on the other hand, had an average score of 2.85 leaving it between slightly and somewhat engaging which can be explained by the fact that tokens are only earned by doing streaks (which also awards XP at the same time), which can only be spent to retry or unlock skills and exchanged them for XP at the end of the course.

In addition, when asked about the game elements that they had enjoyed the most, the most common answers were badges and streaks related to Skill Tree, attendance, and peer-grading assessment. Some students even mentioned that these elements had motivated them to behave in a certain way to earn the game element in question. They proceeded to explain that the streaks motivated them to try and do their best regularly, some even mentioning that thanks to the "Constant Gardener" Streak, they were able to create that pace of work to complete skills.

5.2 User Tests

To evaluate the success of our implementation regarding the new game elements, we resorted to user testing since we could observe real users attempt to complete a set of tasks. This way any areas of confusion could be found as well as possible opportunities to improve the system. As such, we came up with the following sets of tasks, one for each game element, based on the functionalities we intended to test:

Virtual Currency Module:

1. Set currency name to 'Simoleon' and Tokens to XP Ratio to 0.5.

2. Create action to award with name “Partaker”, description as “award tokens”, select type “participated in lecture” and set tokens to 10.
3. Change the initial skill cost to 5, the increment cost to 15 and increment formula to case 2.

Streaks Module:

4. Set the Streaks’ maximum reward to 500 XP.
5. Add a new countable Streak (isCount) with name “Countable”, description as “countable streak”, accomplishment count with value 20, XP reward with value 10 and tokens with value 4.
6. Add a new isPeriodic and isAtMost Streak called “Regular”, description as “periodic streak”, periodicity with value 5 and select days as the periodicity time.
7. Edit the Streak with name “Practitioner” by changing its XP reward to 20, tokens to 0 and setting it as repeatable.
8. Delete Streak called ‘Ackerman’.
9. Duplicate Streak called ‘Sage’.
10. Disable Streak called ‘Conqueror’.

Teams Module:

11. Set maximum number of elements in a team to 4.
12. Add new team with team number equal to 22, and add Ana Sofia and Paulo as team members.
13. Edit team with number equal to ‘5’, add António as member and change the team’s number to 8.
14. Import teams (updating if needed) using the file “import_teams_example.csv”.

To evaluate the performance of a participant in each of these tasks, we collected the following information: the **number of errors** and **clicks made**, the **time spent** on each task and the **success** in completing them. The number of errors made and the success in completing a task allows use to calculate the effectiveness of the participants, i.e., the accuracy and completeness with which participants accomplish the established tasks. Moreover, the time spent to complete a task allows us to understand how efficient the participant was. Most of the user tests were performed in person where the list of tasks was shown on a different device so they could reread them at need without wasting unnecessary time changing screens. The same logic to prevent delays was applied for the remote tests but, in this case, we split our screen so that a part of it would display the tasks to perform. We were able to perform these

tests remotely using the Zoom ² platform, that gathered all the features needed such as, voice and video calls, screen-sharing, and remote-control.

5.2.1 Procedure

We started each test by asking the participants to answer a short questionnaire that would give us some information regarding their age, and knowledge of GameCourse. This was followed by a brief presentation about the system. For participants that had no previous contact with the MCP course, we gave an explanation regarding how gamification is being used and incorporated in education as well as a more in-depth demonstration of the system. For participants that were already acquainted with the system, we only focused on giving them context regarding the modules they were going to interact with.

Each participant was allowed to explore each module, in order for them to get acquainted with it before the tasks began. Next, we randomized the order of the tasks listed above for each participant, to assure that there was no influence caused by the learning curve, and gave them to the participant. Once they were ready to start a task, the time counter would start. After the task was complete, the users were asked to rate that task in a scale of **1 (Very Difficult)** to **7 (Very Easy)** and also had to answer a questionnaire whose answers would allow us to calculate the NASA Task Load Index. Finally, once the all the tasks were done, participants had to answer a final questionnaire where they were asked about their overall experience and if they had any suggestions to improve the system.

5.2.2 Participants

We conducted tests with 21 participants, which is one participant more than the minimum number necessary for a summative analysis. The most common age range of the participants was 21 to 25 years old. However, there were some participants' were between the ages of 16 to 20, 36 to 40, and 56 to 60. Six participants had previously used GameCourse as students while the remaining fifteen was not acquainted with the system.

5.2.3 Result Analysis

In this subsection we analyze the data collected during the User Tests, performing a more general analysis and then a more detailed and specific analysis. This analysis was accomplished by calculating usability metrics, such as, the success rate of each task, as well as calculating the mean, median and standard deviation values of each one of the measurements collected during the tests for each task.

As it can be seen in Table 5.2, participants did not have difficulty in performing 12 out of 14 of the defined tasks, since they present a success rate higher than 90%. On the other hand, tasks number

²<https://zoom.us>

Table 5.2: Success Rate for each task.

Task Number	Success Rate
1	90.48%
2	100%
3	100%
4	100%
5	76.19%
6	47.62%
7	100%
8	100%
9	100%
10	100%
11	100%
12	100%
13	95.24%
14	100%

5 and **6** had a success rate of 76,19% and 47,62%, respectively. Additionally, by looking at Table 5.3, we can observe that the largest number of errors occurred in tasks **5**, **6**, **12**, **13** and **14**, two of them having the lowest success rate. These tasks either required the creation of two distinct streaks, each with different properties, or required the creation, modification or import of a teams into the system. Additionally, when it comes to the time each participant spent on a task, we can conclude that tasks 5 and 6, the ones with the lowest success rate, were the most time consuming.

5.2.3.A Individual Task Analysis

The tasks the participants had to perform were not all similar in difficulty. Tasks **1**, **4**, **8**, **9**, **10** and **11** were similar in terms of difficulty and were, in theory, not as demanding as the others since it only required participants to change specific variables or recognize and click certain buttons within the UI to successfully complete the task. The remaining tasks, on the other hand, were a bit more difficult. However, all the tasks performed by the participants allowed us to better comprehend and detect the existing flaws and problems with the UI.

As mentioned in the Procedure, subsection 5.2.1, after completing a task, the participants were asked to rate it in a scale of **1 (Very Difficult)** to **7 (Very Easy)**. The participants considered the majority of the tasks easy to complete, rating them, in average, with values higher than 6 on the difficulty scale, with the exceptions being tasks **5**, **6** and **7**. However, even though the participants did not find those tasks as difficult as the other three just mentioned, we were able to gather important insights on how the participants interact with the UI and observe some weaknesses within it.

In tasks **7** and **9** we observed that several participants hovered over the icons for its tooltip to be displayed so that they could be certain they were clicking on the correct icon. Moreover, in tasks **12** and

Table 5.3: Mean, median and standard deviation values of the information collected from each task.

Task	Time (s)			Number of Clicks			Number of Errors		
	Mean	Median	Std. Dev	Mean	Median	Std. Dev	Mean	Median	Std. Dev
1	20.0	19.9	6.5s	3	3	0.45	0.10	0	0.30
2	31.4	32.6	16.5	6.86	7	0.57	0	0	0
3	27.6	27.9	11.0	5.24	5	0.44	0	0	0
4	6.4	5.6	5.5	2	2	0	0	0	0
5	50.4	49.4	15.7	8.76	9	0.94	0.29	0	0.56
6	42.9	49.9	20.9	9.86	10	1.62	0.71	1	0.85
7	23.6	24.5	11.9	4.81	5	0.75	0	0	0
8	7.8	4.7	10.7	2.14	2	0.65	0.05	0	0.22
9	4.2	2.9	3.5	1.38	1	0.80	0	0	0
10	6.5	4.1	6.5	1.19	1	0.60	0	0	0
11	11.4	9.5	8.2	2	2	0	0	0	0
12	25.3	22.6	9.3	6.33	6	0.66	0.76	1	0.46
13	19.6	18.4	10.8	5.10	5	0.70	0.17	0	0.51
14	21.7	21.3	4.6	5.05	5	1.02	0.33	0	0.48

13 participants had to create or edit a team and, in both tasks, they had to add team members, having to interact with the select modal that consists of a search bar and a listing of all the course users that have yet to be added to a team. As you can see in Figure 5.1, each line below the search represents a course user, and at its right side there is a button represented by a *plus* (+) so that users can add that user to a team. However, most participants tried to click on the line instead of the actual button, so it might be advantageous to add this functionality in this modal. Moreover, some participants used the search bar to find the students to add to the team. However, after adding that student, what the participants had wrote was not being removed, and they had to manually erase it. As such, several participants suggested that the system should clean the search bar automatically after adding a student to a team.

In addition, another mistake that participants made was adding the wrong team member, having to remove them. When this happened in task **12** no errors occurred, however, when editing an existing team in task **13**, an error occurred, revealing that the system was not able to retrieve that course user's id which made it impossible to remove users when editing the team. This bug, that we later fixed, occurred since the list of users in that team was not being correctly obtained, only retrieving the students' name and number when their id was essential.

Additionally, in task **14** participants had to import teams using a csv file and then clicking the button that allowed to update the existing teams in case any change was included in the file. However, seven

Select Members:

Search..	
António - 22228	+
Catarina - 23460	+
Diogo - 45124	+
Miguel - 23457	+
Francisco - 23459	+

Figure 5.1: Teams select modal.

participants tried to click the button before selecting the file to import. Only after doing this and, in some cases, repeatedly clicking the button, did they successfully complete the task. Perhaps, it would also be helpful to display a warning when a user tries to import teams without having selected a file beforehand.

Finally, as stated in the beginning of this subsection, the tasks the participants found the hardest were tasks **5**, **6** and **7**, the first two having the lowest success rates. In these tasks the participants had to either create a streak or edit an existing one. All the participants that were not able to successfully complete these tasks forgot to enable a certain attribute of the streak. For task **5** the most common error that culminated in failing the task was not enabling the `isCount` variable. A similar situation happened for task **6** where participants either did not enable the `isAtMost` or `isPeriod` variables or, in some cases, both. During the tests, we observed that most students had a hard time navigating through the create/edit modal. One of the main problems was the positioning of the toggle buttons that were placed at the right of each variable, being closer to the next variable than to the one it was referring to. Consequently, some participants suggested the implementation of an informative icon next to each variable to give them extra information that may be crucial to correctly create streaks. One participant also suggested that GameCourse should become a multilingual system where users can chose which idiom they want the system to be in. Both the positioning as well as this informative icon are, at the time of writing this thesis, already implemented in GameCourse.

5.2.3.B NASA Task Load Index

To better grasp the perceived workload of each task, students were asked to answer the NASA Task Load Index (NASA-TLX) questionnaire after completing a task. This way we could better understand the amount of mental and physical effort the participants had to apply to perform each task. To perform this summative analysis, we used the Raw NASA-TLX variant of this assessment tool, which instead of using weighted scales, only sums the scores of each questions in the test. As such, to each score given

we subtract 1 and multiply by 5. The average score obtained was **18.04**, which is a relatively low score considering it can go from 0 to 100. Consequently, this reflects that the participants' perceived mental workload when performing the tasks was low.

5.2.3.C System Usability Scale

At the end of the questionnaire given to the participants, all the 10 questions of the System Usability Scale (SUS) were asked so that we could calculate the system's overall SUS score. This tool allows us to comprehend what were the participants thoughts regarding the usability of the modules tested by analyzing the overall SUS score of the system. The participants had to answer the questions by selecting a value in a scale of **1 (Strongly disagree)** to **5 (Strongly Agree)** for us to calculate this score. The SUS score is considered above average when its value is higher than 68 while an excellent score is closer to 80 [42].

The odd-numbered questions are related to the the positive aspects of the system while the even-numbered focus on the negative aspects. As such, to calculate the SUS, we had to firstly subtract 1 from the score of the odd-numbered questions and, for even numbered questions, the score was subtracted by 5 to invert it. Then, we summed this scores and multiplied it by 2.5 [42]. These steps were repeated to all the 21 responses we had and, after calculating the sum of all the individual scores, we obtained a **89.88** average SUS score, which is considered an excellent result.

5.3 Rule System Performance Testing

Finally, we had to test if the changes made regarding the Rule System, described in Section 4.3.2, had allowed us to meet our goal of improving its performance. During the development, we had already tested the impact the changes made had on individual functions and rules when fired for a single student so that we knew we were on the right track to fix the problem. However, we decided that we should test these improvements when there are more students and, consequently, more data in the database, to replicate, on a smaller scale, what happens during a course. As such, we measured the time this new version took to fire the rules and award the respective game elements to a different number of students so that we could compare it to the older version. All of these tests were done in a test environment (https://pcm.rnl.tecnico.ulisboa.pt/gamecourse_test_v2/) within the virtual machine that hosts GameCourse.

Before performing any tests, we needed to set up the system so that we could start with a clean test environment. First, we deleted all the users with the role Student in the course as well as the data stored in the tables that we were going to use (the participation and award tables). We performed tests for 20, 40, 60, 80, and 100 students. As such, since we would need to populate the test database with

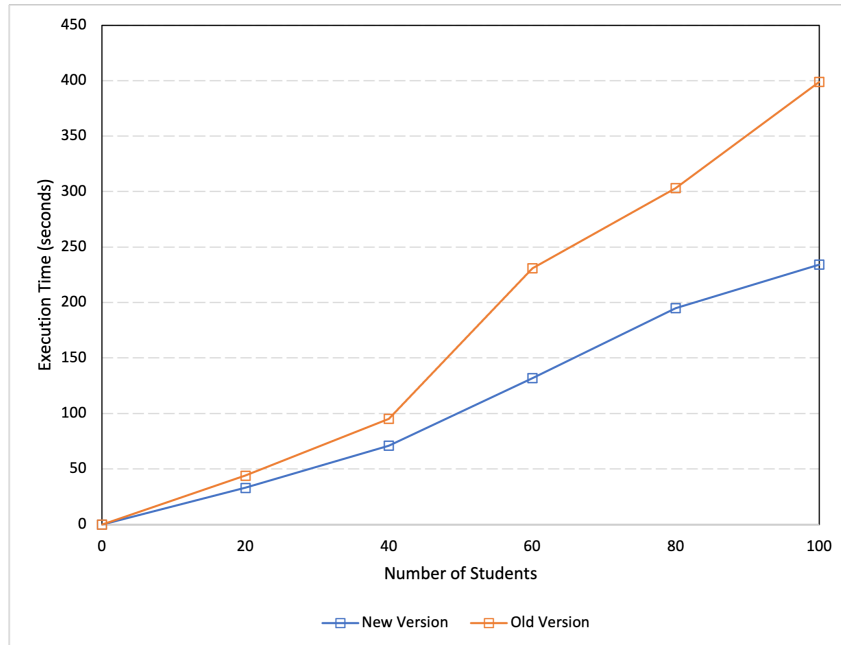


Figure 5.2: Rule System execution times by number of students.

a considerable amount of data, we created a CSV file containing up to 100 users to be imported into the system. Then, we created a script that would output a CSV file containing the Participations to be inserted in the **participation** table required to fire the rules for each student. However, we did not want to insert the students and their Participations all at once since this would not allow us to measure the time it took to fire the rules for that group of students. Additionally, since the rules responsible for awarding streaks and skills had proven to be the most time-consuming and were the main focus of the changes made in the Rule System, the Participations inserted in the database would only fire these specific rules. As such, we divided the CSV file into 5, each containing the Participations of 20 different users. Using the DB Beaver³ tool, we imported one file at a time, which allowed us to easily populate the database.

We can observe, in Figure 5.2, the execution times of the oldest and newest versions of the Rule System took to fire the same rules for a different number of students. As can be seen, for smaller groups of students the differences between the execution times are smaller. Nonetheless, the new version is indeed faster than the old one, and, for groups with more than 40 students, it takes less than half the time the old version took, which is a significant improvement. While performing these tests, we noticed that the execution times would vary even if we were firing the same rules and awarding game elements for the second time. Even though these fluctuations were small (fewer or extra milliseconds in comparison), we needed to understand why this was happening in case these few milliseconds became extra seconds of execution time. These changes would happen at random hours of the day, making it hard to find a pattern: sometimes the Rule System was faster during the day and slower during the

³<https://dbeaver.io>

night, and other times it was the other way around. The culprit of this problem was the virtual machine that hosts GameCourse. We did not explore this in-depth since the changes were minor but doing so in the future could be advantageous.

5.4 Discussion

Setting up the new game elements from the modules we had implemented and using them for the Multimedia Content Production allowed us to discover any weaknesses our implementations had. This was extremely helpful since we were able to improve the developed functionalities by fixing any relevant bugs that appeared throughout the seven-week period in which the course was taught. Additionally, some bugs that were observed and fixed allowed us to further improve how the system behaves when there are several courses.

Then, the user tests allowed us to comprehend if the new modules were easily understood by the users and, if not, what the sources of confusion were. We obtained satisfactory results and, overall, the organization of each one of the new modules was well received and participants did not have a hard time navigating through them. Many users resorted to the icons' tooltips to check if the icon they were hovering was the one they needed to complete the task. Additionally, one of the participants suggested that GameCourse should support different languages allowing the users to select which idiom they would like the contents of the system to be. Finally, since many participants tried to add team members by clicking on their name or respective line, this functionality could be created in the future.

Moreover, the only major sources of confusion that we observed were related to the creation of the streaks. To fix this, some users suggested the addition of an informative tooltip or icon next to each property that would give them the necessary information about it. This is already implemented by the time we are writing this thesis, which possibly would improve the obtained results.

Finally, performance testing allowed us to comprehend if the changes we had made in the Rule System had indeed improved its performance. We had already tested out the time each function and rule took to individually run for a single student, however, we had to apply it to several students firing a rule for each one, which resulted in awards being given. The results from these tests were positive since the execution times for different users and awards given had significantly decreased, which matched our expectations.

6

Conclusion

Contents

6.1 Conclusions	79
6.2 System Limitations and Future Work	80

6.1 Conclusions

Even though GameCourse offers several functionalities that contribute to a better gamified experience, our work aimed to improve the existing ones as well as implement and integrate new modules to increase game element diversity. Additionally, we also intended to improve the Rule System's performance, a crucial part of GameCourse, that was taking a long time to execute.

With those goals in mind, our work, GameCourse - The Next Level, added three new game elements, Streaks, Teams and Virtual Currency. The use of the Streaks and Virtual Currency Modules in MCP 2021/2022 revealed some vulnerabilities in their implementations that we did not foresee, but were able to quickly fix. Additionally, the feedback obtained from the students at the end of the course, allowed us to conclude that, between these two new game elements, Streaks were considered more engaging. Nevertheless, this can be explained due to the fact that the Virtual Currency relies on other game elements, like Streaks and Skills, to be earned or spent. Students also mentioned that streaks had a very positive impact in the way they behaved throughout the course, as they felt the need to do their best regularly. Although there is still room for improvement, we can conclude that these new game elements were valuable additions to GameCourse.

Additionally, to understand if the new modules were correctly integrated, and were of easy understandability for a user, we conducted user tests with 21 participants. The tests gave us the needed insights regarding any areas of confusion and weaknesses of our work. When it comes to Streaks, participants struggled to correctly create streaks as they were stated in the tasks given. These results can be explained since Streaks are a complex game element that needs to hold specific information to correctly work. To tackle this, informative icons were added next to each one of the fields, holding its description. On the other hand, when participants were performing tasks related to Teams, some bugs were found as well as new possible UI improvements. Overall, the user tests allowed us to further improve the modules' functionalities and usability.

Lastly, we improved the Rule System's performance by tackling the four main issues identified. This mainly consisted in improving the already existing functions, including the accesses to the database. To evaluate what were the effects of our work in the system's performance, we conducted performance tests. As such, we measured the time each one of the rules that had proven to be the most time-consuming took to fire on the new and old versions. The results of the tests performed showed that the new version takes less than half the time the old version took, which is a significant improvement. This allowed us to confirm that we did indeed improved the poor performance of the Rule System, achieving our goals.

6.2 System Limitations and Future Work

Our work has increased the diversity of game elements as well as improved the existing system. In addition, one of the new additions to the system, allowed the creation of a new type of leaderboard that can be potentially beneficial when it comes to motivating students. Nevertheless, there still is room for improvement.

When it comes to the Teams module, a way of automatically generating teams and randomly assigning students to them should be explored and implemented since it may be advantageous for courses where professors do not want to import or manually create teams. In addition, the system could be further improved to allow a more complete integration of the teams since the whole system is extremely tailored to individual Participations. This means replicating how the users data is dealt with and applying it to teams.

In addition, regarding the Rule System, there are a few changes that can be made. The connector module could be separated into files, each containing the functions used in the same context, which would culminate in a better organization of this part of the system. Additionally, the automatic generation of the rules should be a global functionality that each module can simply use. As of now, if a user wants to have this functionality within a new module, it needs to replicate the existing code to then apply it to the module. Moreover, the Rule Editor could have a highlight functionality that could, for instance, give a specific color to a certain part of the rules like what is done in text editors, as it can be seen in the prototype shown in Figure 6.1. This would be extremely advantageous for users to be able to visually differentiate from the actual parts of the rule that are going to be run and the ones that are commented.


```

rule: reTrailer

  when:
    #CHANGED:
    #combo1 = rule_unlocked("Course Logo", target) and rule_unlocked("Movie Poster", target)
    #combo2 = rule_unlocked("Album Cover", target) and rule_unlocked("Publicist", target)
    #combo1 or combo2

    wildcard = GC.skillTrees.wildcardAvailable("<skill-name>", "<tier-name>", target)
    combo1 = rule_unlocked("Publicist", target) and rule_unlocked("Album Cover", target)
    combo2 = wildcard and rule_unlocked("Course Logo", target)
    combo1 or combo2

    skill_based = combo1
    use_wildcard = False if skill_based else True
    |
    logs = GC.participations.getSkillParticipations(target, "reTrailer")
    rating = get_rating(logs)
    rating >= 3

    # NOTE:
    # Virtual Currency Enabled ? Uncomment code below : Delete

    # valid_attempts = get_valid_attempts(target, "reTrailer")
    # valid_attempts > 0
    # (new_total, removed) = get_new_total(target, valid_attempts, rating)
    # new_total >= 0

  then:
    #CHANGED:
    #award_skill(target, "reTrailer", rating, logs)

    award_skill(target, "reTrailer", rating, logs, use_wildcard, "Wildcard")

    # NOTE:
    # Virtual Currency Enabled ? Uncomment code below : Delete

    # update_wallet(target, new_total, removed, logs)

```

Figure 6.1: Template for Rule Editor's possible highlight functionality.

Bibliography

- [1] R. Bartle, "Virtual worlds: Why people play," *Massively multiplayer game development*, vol. 2, no. 1, pp. 3–18, 2005.
- [2] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: defining "gamification"," in *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*, 2011, pp. 9–15.
- [3] Z. Zainuddin, S. K. W. Chu, M. Shujahat, and C. J. Perera, "The impact of gamification on learning and instruction: A systematic review of empirical evidence," *Educational Research Review*, vol. 30, p. 100326, 2020.
- [4] D. Pratt, "Good teaching: One size fits all?" *New Directions for Adult and Continuing Education*, vol. 2002, pp. 5 – 16, 03 2002.
- [5] A. H. S. Metwally, L. E. Nacke, M. Chang, Y. Wang, and A. M. F. Yousef, "Revealing the hotspots of educational gamification: An umbrella review," *International Journal of Educational Research*, vol. 109, p. 101832, 2021.
- [6] L.-M. Putz, F. Hofbauer, and H. Treiblmaier, "Can gamification help to improve education? findings from a longitudinal study," *Computers in Human Behavior*, vol. 110, p. 106392, 2020.
- [7] I. Furdu, C. Tomozei, and U. Kose, "Pros and cons gamification and gaming in classroom," *arXiv preprint arXiv:1708.09337*, 2017.
- [8] N. Xi and J. Hamari, "Does gamification satisfy needs? a study on the relationship between gamification features and intrinsic need satisfaction," *International Journal of Information Management*, vol. 46, pp. 210–221, 2019.
- [9] J. Krath, L. Schürmann, and H. F. von Korflesch, "Revealing the theoretical basis of gamification: A systematic review and analysis of theory in research on gamification, serious games and game-based learning," *Computers in Human Behavior*, vol. 125, p. 106963, 2021.

- [10] E. Kyewski and N. C. Krämer, "To gamify or not to gamify? an experimental field study of the influence of badges on motivation, activity, and performance in an online learning course," *Computers & Education*, vol. 118, pp. 25–37, 2018.
- [11] P. Garone and S. Nesteriuk, "Gamification and learning: A comparative study of design frameworks," in *International Conference on Human-Computer Interaction*. Springer, 2019, pp. 473–487.
- [12] J.-W. Chang and H.-Y. Wei, "Exploring engaging gamification mechanics in massive online open courses," *Journal of Educational Technology & Society*, vol. 19, no. 2, pp. 177–203, 2016.
- [13] S. Bai, K. F. Hew, and B. Huang, "Does gamification improve student learning outcome? evidence from a meta-analysis and synthesis of qualitative data in educational contexts," *Educational Research Review*, vol. 30, p. 100322, 2020.
- [14] G. F. Tondello and L. E. Nacke, "Validation of user preferences and effects of personalized gamification on task performance," *Frontiers in Computer Science*, vol. 2, p. 29, 2020.
- [15] L. Rodrigues, A. M. Toda, P. T. Palomino, W. Oliveira, and S. Isotani, "Personalized gamification: A literature review of outcomes, experiments, and approaches," in *Eighth International Conference on Technological Ecosystems for Enhancing Multiculturality*, 2020, pp. 699–706.
- [16] G. F. Tondello, A. Mora, A. Marczewski, and L. E. Nacke, "Empirical validation of the gamification user types hexad scale in english and spanish," *International Journal of Human-Computer Studies*, vol. 127, pp. 95–111, 2019.
- [17] 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*, 2013, pp. 1–6.
- [18] 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.
- [19] D. Codish, I. Beer-Sheba, and G. Ravid, "Personality based gamification: How different personalities percive gamification research in progress," *No. Davis*, vol. 2012, 1989.
- [20] R. Smiderle, S. J. Rigo, L. B. Marques, J. A. P. de Miranda Coelho, and P. A. Jaques, "The impact of gamification on students' learning, engagement and behavior based on their personality traits," *Smart Learning Environments*, vol. 7, no. 1, pp. 1–11, 2020.
- [21] A. C. T. Klock, I. Gasparini, M. S. Pimenta, and J. Hamari, "Tailored gamification: A review of literature," *International Journal of Human-Computer Studies*, vol. 144, p. 102495, 2020.

- [22] L. Suárez, C. Thio, and S. Singh, "Why people play massively multiplayer online games?" *International Journal of e-Education, e-Business, e-Management and e-Learning*, 01 2013.
- [23] L. E. Nacke, C. Bateman, and R. L. Mandryk, "Brainhex: A neurobiological gamer typology survey," *Entertainment computing*, vol. 5, no. 1, pp. 55–62, 2014.
- [24] 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.
- [25] A. Marczewski, "Even ninja monkeys like to play," *London: Blurb Inc*, 2015.
- [26] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: Classic definitions and new directions," *Contemporary educational psychology*, vol. 25, no. 1, pp. 54–67, 2000.
- [27] G. Tondello, R. Wehbe, L. Diamond, M. Busch, A. Marczewski, and L. Nacke, "The gamification user types hexad scale," 10 2016.
- [28] C. E. Lopez and C. S. Tucker, "The effects of player type on performance: A gamification case study," *Computers in Human Behavior*, vol. 91, pp. 333–345, 2019.
- [29] T. Aldemir, B. Celik, and G. Kaplan, "A qualitative investigation of student perceptions of game elements in a gamified course," *Computers in Human Behavior*, vol. 78, 10 2017.
- [30] 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, pp. 47–62, 07 2016.
- [31] [U+FFFD] Lavoué, B. Monerrat, M. Desmarais, and S. George, "Adaptive gamification for learning environments," *IEEE Transactions on Learning Technologies*, vol. 12, no. 1, pp. 16–28, 2019.
- [32] S. Hallifax, A. Serna, J.-C. Marty, G. Lavoué, and E. Lavoué, "Factors to consider for tailored gamification," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, 2019, pp. 559–572.
- [33] T. Jiang, "Research: In-game streaks," Jul 2018. [Online]. Available: <https://blog.prototypr.io/research-in-game-streaks-92bfb229e776>
- [34] 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)*, 2016, pp. 637–642.
- [35] A. Baltazar, "Smartboards," M.Sc. dissertation, Instituto Superior Técnico, 2016.
- [36] D. Lopes, "Gamecoursebeyond," M.Sc. dissertation, Instituto Superior Técnico, 2020.

- [37] P. Silva, "Gameui," M.Sc. dissertation, Instituto Superior Técnico, 2020.
- [38] I. Paiva, "Autogame," M.Sc. dissertation, Instituto Superior Técnico, 2020.
- [39] A. Nogueira, "Dynagame," M.Sc. dissertation, Instituto Superior Técnico, 2021.
- [40] A. Gonçalves, "Gamecoursepersonal," M.Sc. dissertation, Instituto Superior Técnico, 2021.
- [41] M. Saraiva, "Gamecourseui," M.Sc. dissertation, Instituto Superior Técnico, 2021.
- [42] K. Betteridge, "What every client should know about sus scores — bentley university." [Online]. Available: <https://www.bentley.edu/centers/user-experience-center/what-every-client-should-know-about-sus-scores>



GameCourse Partial Database Schema

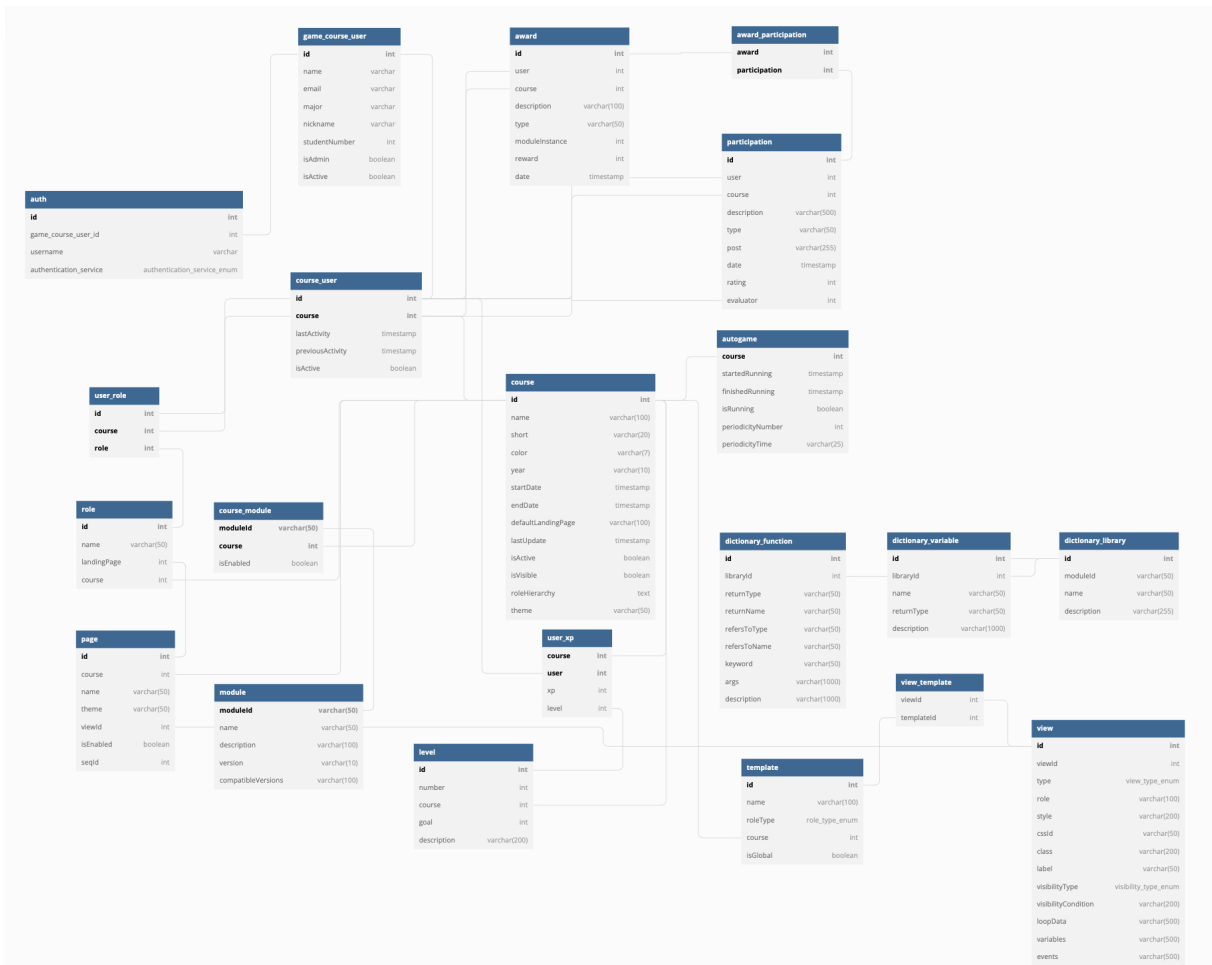


Figure A.1: Partial GameCourse's database schema.

B

User Tests

B.1 Tasks

1. Set currency name to 'Simoleon' and Tokens to XP Ratio to 0.5.
2. Create action to award with name "Partaker", description as "award tokens", select type "participated in lecture" and set tokens to 10.
3. Change the initial skill cost to 5, the increment cost to 15 and increment formula to case 2.
4. Set the Streaks' maximum reward to 500 XP.
5. Add a new countable Streak (isCount) with name "Countable", description as "countable streak", accomplishment count with value 20, XP reward with value 10 and tokens with value 4.
6. Add a new isPeriodic and isAtMost Streak called "Regular", description as "periodic streak", periodicity with value 5 and select days as the periodicity time.
7. Edit the Streak with name "Practitioner" by changing its XP reward to 20, tokens to 0 and setting it as repeatable.

8. Delete Streak called 'Ackerman'.
9. Duplicate Streak called 'Sage'.
10. Disable Streak called 'Conqueror'.
11. Set maximum number of elements in a team to 4.
12. Add new team with team number equal to 22, and add Ana Sofia and Paulo as team members.
13. Edit team with number equal to '5', add António as member and change the team's number to 8.
14. Import teams (updating if needed) using the file "import_teams_example.csv".

B.2 Initial Questionnaire

1. How old are you?
 - (a) 16-20
 - (b) 21-25
 - (c) 26-30
 - (d) 31-35
 - (e) 36-40
 - (f) 41-45
 - (g) 46-50
 - (h) 51-55
 - (i) 56-60
2. Have you ever used GameCourse?
 - (a) Never
 - (b) Used it as a Student
 - (c) Used it as an Administrator

B.3 SUS Questionnaire

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.

3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I think that I would need the support of a technical person to be able to use this system.
6. I found the various functions in this system were well integrated.
7. I thought there was too much inconsistency in this system.
8. I would imagine that most people would learn to use this system very quickly.
9. I found the system very cumbersome to use.
10. I felt very confident using the system.
11. I needed to learn a lot of things before I could get going with this system.

B.4 NASA TLX Questionnaire

1. How mentally demanding was the task?
2. How physically demanding was the task?
3. How hurried or rushed was the pace of the task?
4. How successful were you in accomplishing what you were asked to do?
5. How much did you have to work to accomplish your level of performance?
6. How insecure, discouraged, irritated, stressed and annoyed were you?

B.5 Final Questions

1. Were all icons used in the system easily understandable? If the answer is "No", please indicate the ones that were not.
2. Would you change anything in the system? If yes, what would it be?
3. Any other suggestions?

