Autogame

Inês Barral Paiva

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Daniel Jorge Viegas Gonçalves

Examination Committee

Chairperson: Prof. Luís Manuel Antunes Veiga
Supervisor: Prof. Daniel Jorge Viegas Gonçalves
Member of the Committee: Prof. Daniel Filipe Martins Tavares Mendes

January 2021

Acknowledgments

I would like to thank my parents for their constant love, support, and encouragement throughout the years. I love you and could not have done this without you.

To my supervisor, Professor Daniel Gonçalves, for his guidance and monitoring, which were essential to help this project reach its completion. Also, to Tomás Alves and Amir Nabizadeh, for the thorough reviews and tips in all of this project's stages.

To Ana, my companion in this adventure, you made this process much more comfortable, pleasant, and undoubtedly more fun.

To my high school friends, my second family, that were always there when things got rough.

To my teammates from IST's basketball team, which provided me with so many hours of good times that helped me maintain my sanity throughout this whole process. And mostly, Catarina, my biggest cheerleader, thank you for your never-ending help and support.

To my family, and mostly my grandparents, thank you. You showed me the kind of love that I carry with me in everything I do. Without you, I wouldn't be where I am today, this is for you.

This project was partly funded by FCT through project GameCourse PTDC/CCI-CIF/30754/2017.

Abstract

Education plays a crucial role in society, shaping young people, and giving them the necessary tools to accomplish their goals. However, education sometimes fails in giving students motivation and interest in learning. Gamification provides a solution to this problem. Using gamified elements allows students to learn in a more engaging and interactive way, motivating them through competition with their colleagues and themselves.

Multimedia Content Production (MCP) is a Msc. course at Instituto Superior Técnico that has been using gamification for several years, achieving excellent results and receiving positive feedback from students. However, the system currently used in this course to provide a gamification experience has some automation problems, which results in a lot of time wasted by the professors to keep it running correctly.

This thesis aims to develop AutoGame, a gamified rule-based system that uses rules to act on data and applies logic to calculate the course's gamified awards. Autogame was created in the context of MCP but can be adapted and used in other contexts.

Keywords

Autogame, Gamecourse, Gamerules, Gamification, Gamification in education, Rule-based system.

Resumo

A educação tem um papel basilar no progresso das sociedades humanas, uma vez que transmite aos jovens o saber fundamental acumulado por gerações, ajudando-os a concretizar os seus projetos de vida. No entanto, para que os jovens tirem proveito destes recursos educativos têm de estar motivados. A motivação é uma condição fundamental para que os jovens realmente aprendam, todavia os educadores nem sempre investem o suficiente neste fator. A gamificação cria uma solução para este problema, usando elementos de jogos, permite que os alunos aprendam de maneira mais interativa e cativante, desafiando-se a eles próprios e competindo com os colegas.

Produção de Conteúdos Multimédia é uma cadeira do Mestrado em Engenharia Informática e de Computadores do Instituto Superior Técnico que utiliza estratégias de gamificação há vários anos, conseguindo excelentes resultados e recebendo feedback positivo por parte dos alunos. No entanto, o sistema utilizado para providenciar uma experiência de gamificação tem alguns problemas de automatização que resulta no investimento de uma grande quantidade de tempo por parte dos professores.

O estudo desenvolvido no âmbito desta tese teve como objetivo o desenvolvimento do Autogame, um sistema de regras que atua usando lógica sobre um conjunto de dados: os introduzidos pelo professor e os gerados pela atuação dos alunos. Calculando assim os prémios desbloqueados pelos alunos de maneira automática, o que permite manter um fluxo constante de informação entre os intervenientes no processo educativo. Este sistema foi criado para PCM, mas poderá ser adaptado para ser utilizado noutros contextos.

Palavras Chave

Autogame, Gamecourse, Gamerules, Gamificação, Gamificação na educação, sistema baseado em regras.

Contents

List of Figures

List of Tables

Listings

Acronyms

MCP	Multimedia Content Production
IST	Instituto Superior Técnico
LMS	Learning Management System
SWRL	Semantic Web Rule Language
RDR	Ripple Down Rules
NLP	Natural Language Processing
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
XP	Experience Points
UI	User Interface
QR	Quick Response

1 Introduction

Contents

Education plays a crucial role in society, shaping young people, and giving them the necessary tools to help them achieve their goals. Being such an important part of human life, education has suffered and continues to suffer a considerable evolution, as we try to keep on improving it, finding new methods to make it as efficient as possible. One of the main flaws of the educational system nowadays is that it is often not motivational enough for students, making them uninterested in learning. Gamification is a strategy that utilizes gaming elements in non-gamified contexts, one of these contexts being education, and it provides a whole new learning experience. This method relies on giving experience points to students for completing specific tasks. It has been used in computer science courses and has proven to be a great way to motivate students to learn and to give their best, giving them friendly competition and rewarding them for their effort. Besides, it also helps the professors track their students better, providing more detailed information about what their students have been doing in the course.

Multimedia Content Production (MCP) is a course in the Information Systems and Computer Engineering Master in Instituto Superior Técnico (IST). This course has been using gamification as its primary learning method for several years and has been the subject of many studies that aim to understand the impact of gamification on students' motivation and behavior. For this purpose, it was created a platform called Gamecourse that works with other external sources: Moodle [1], GoogleSheets [2], *I am here* [2], a Quick Response (QR) module and Smartboards [3]. This platform allows students to interact with the course, submit their works, and check their progress. Everything is graded using Experience Points (XP), and there are online tasks (such as skill tree, and forums) and tasks that can only be done in class (such as quizzes, attendances, and answering questions). Also, students can decide which tasks they want to do and in which order they want to do them. As said above, there is a leader board in Smartboards [3], which displays each student's current progress, from the one with more XP to the one with less, allowing them to compare themselves with their peers. In the leader board, it is also possible to access the student's profiles and see where they got their points from, the badges they have gathered, and the skills from the skill tree they have completed.

Although this has shown excellent results throughout the years, there are still many things to improve to make this course even better, more automated, and more adaptable to the different kinds of students. Nowadays, to calculate the course's awards (badges, grades, and completed skills), a teacher must manually run a script and upload its result to the leader board where the students can access it. The automation of this platform will be the focus of this thesis, and for that purpose, it will be created a rule-based system to calculate the course's awards.

[1] `https://moodle.org/?lang=pt`
[2] `https://www.google.com/sheets/about/`

1.1 Objectives

In its current state, the platform used on MCP requires a lot of steps to assign awards to the students and flow the information between Moodle and the Leaderboard. A teacher must download data from the four different external sources, use it to run a script that calculates the awards, and upload its output to the leader board. Doing this requires a lot of time for the professors, and it's not an efficient process.

This thesis's goal is to create AutoGame, a rule-based system that retrieves data from Gamecourse, acts on it by calculating the students' awards, and uploads the results back to Gamecourse. Autogame is going to be used for MCP to make the process of calculating awards more automated, but can also be adapted to be used in other similar courses. The requirements for this system are the following:

- **Automatic**: automation is one of the most important requirements for the new system that will be developed. As previously stated, it requires many steps for the faculty of MCP to work with Gamecourse. The new system, AutoGame, should run automatically, collecting and processing all the data necessary.
- **Incremental**: in its current state, the system being used processes all the data every time the faculty runs the scripts, which wastes time and resources. The new system should have the capacity to distinguish data that has been processed from new data.
- **Efficient**: efficiency is an essential aspect of any technology system. The currently used system lacks efficiency because it takes a long time to update data and requires a lot of work to do so. With the use of a rule-based system, it will be possible to achieve better performance by applying rules only when changes have been made.
- **Grade retractions**: sometimes professors may want to change something in the system, such as a grade, which would change the results of a rule that had been applied previously. Autogame must allow users to make changes and must know how to deal with them.

1.2 Document Structure

This document will be divided in the following sections:

- **Section 2, Related Work**: brief explanation of the concept of gamification and its use in the context of education, specifically in MCP. In depth study of rule-based systems and the different approaches that can be used in the creation of rules, with an analysis of its usability and interfaces. Discussion of the research and how it will impact our solution.
- **Section 3, Gamecourse**: description of what Gamecourse is and how it works, specifically the parts that are most important for Autogame.

- **Section 4, Gamerules**: description of what Gamerules is and how we will use it as a basis for Autogame.
- **Section 5, Autogame**: detailed description about the thesis implementation and its functionalities and features. List of how each problem was solved and how the requirements proposed were met.
- **Section 6, Evaluation**: assessment of the system's outputs and performance. Description of the methods used to evaluate Autogame and its results.
- **Section 7, Conclusions**: Final remarks about the work and what could be done in the future to improve it.
- **Appendix A, Entitity-Relationship Model**: Complete Entity-Relationship Model for Gamecourse's database
- **Appendix B, MCP's rules**: List of the rules created for MCP.

2

Related work

Contents

To deal with the gamification features of the course, MCP uses Gamecourse. This platform interacts with other external sources to provide students with a good gamification experience. A part of that, there is a set of scripts that have to be run manually by the teachers to calculate the course's awards. So, to make the flow of information inside Gamecourse to be as automated as possible, a rule-based system must be created.

A rule-based system helps apply knowledge most efficiently and correctly, teaching systems how to act when certain conditions are met. These systems have shown results in very different areas such as medicine, helping diagnose multiple sclerosis [4] and tuberculosis [5], and in other areas, for example, in the integration of information in onboard devices [6].

This section aims to analyze the different kinds of rule-based systems and their advantages and disadvantages. It is also essential to understand how the system allows the user to create, edit, and delete rules and require expert knowledge to use. For this study, we divided rule-based systems into two different categories: visual and non-visual. These categories refer to how the rules are created within the system.

2.1 Non-visual rule systems

This section describes the non-visual ways of creating rule-based systems: propositional statements, audio, and Natural Language Processing (NLP). These methods are detailed in the following sub-sections, along with examples.

2.1.1 Propositional statements

A propositional statement is the most basic way of representing a rule, where the rule has a format like ”IF this condition occurs THEN this happens”. Using this method, it is possible to create rules with all kinds of complexity levels, from ones with one condition and one conclusion to ones with multiple conditions. Although this is a straightforward way of representing a rule, this can become very confusing with the increase of the rule's complexity.

Propositional statements have been used in the military to help control a weapon system of systems [7]. In this case, a rule-based system was created to help teach military commanders to make fair use of weapon systems. To create this, it was used a knowledge base and a rule base, which could be edited by the administrators, making this a scalable and flexible system. There are five categories of rules: target-matching, environment-matching, performance-constraint, equipment-combination, and evaluation-analysis. These rules are written using the following structure:

$$IF\ X_1\ AND\ X_2\ AND...\ AND\ X_n\ THEN\ Y_1\ OR\ Y_2$$

Where X1, X2, Xn, Y1, and Y2 are states of targets, equipment, and the environment. To help simplify the rules created, every rule that has an ”or” is decomposed into other more straightforward rules that only contain ”and” and implications (Fig.2.1).

Rule Counter	Specific Decomposed Rule
1	$x_1 \wedge x_2 \rightarrow Y_1$
2	$x_3 \rightarrow Y_1$
3	$x_1 \wedge x_2 \rightarrow Y_2$
4	$x_3 \rightarrow Y_2$

Figure 2.1: Decomposition of rule: if (X1 ∧ X2) ∨ X3 then Y1 ∨ Y2.

An example of a rule that could be created using this system is:

$$IF\ IsDay - time(DetectTime)\ AND\ IsTarget(PowerPlant)$$

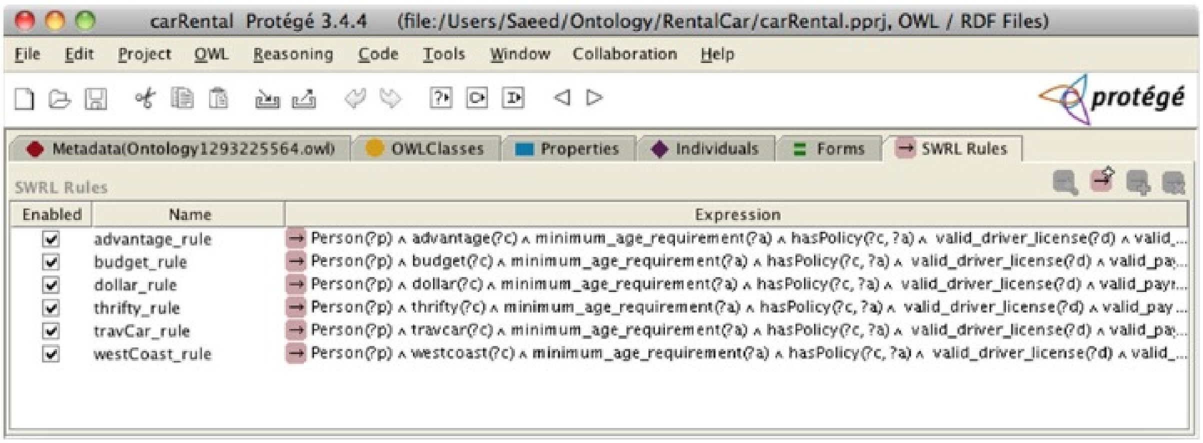
$$THENPowerPlantDetected(True)$$

This system has helped automate the weapon systems and has made its use more correct and efficient. However, as previously stated, it has a scalability problem, making it very difficult to create more complex rules. On the other hand, it is very straightforward and easy to understand because the rule almost describes itself.

2.1.2 Natural Language Processing

NLP uses linguistics and artificial intelligence to extract information from text [8]. This can also be applied to our case, allowing users to write the rules in their own language, using their vocabulary. Although it can be an excellent alternative to facilitate the creation of rules, NLP techniques are still in a research phase, meaning that they are not entirely reliable and are still limited [9]. Using text can be an easier way to make sure that all users can understand and interact with the system. Yet, at the same time, it can make it confusing and not usable in mobile devices, which would need something more compact than a whole text to represent a rule.

This method has been used in a framework that extracts rules from online text [10], which uses NLP and text mining methods to acquire knowledge from the internet and encode it as rules. This framework uses existing knowledge about a particular domain, including core concepts and deductive relationships to extract rules. The rules are created using Semantic Web Rule Language (SWRL), which is a language used to express rules and logic. Rules are composed of a head and a body, which are then formed by atoms that are predefined types of words. To find dependencies in the text the framework uses the Stanford Parser [1] to analyze grammatical relationships. After finding the dependencies, the system finds the correspondent classes, and finally, it can start assembling rules. Using the previously found relationships the framework produces a chain of atoms that represent the rule. In Fig. 2.2 we show you an example of a framework that uses NLP to extract rules from text.


carRental Protégé 3.4.4 (file:/Users/Saeed/Ontology/RentalCar/carRental.pprj, OWL / RDF Files)
File
Edit
Project
OWL
Reasoning
Code
Tools
Window
Collaboration
Help
protégé
Metadata(Ontology1293225564.owl)
OWLClasses
Properties
Individuals
Forms
SWRL Rules
SWRL Rules

Enabled	Name	Expression
✓	advantage_rule	Person(?p) ∧ advantage(?c) ∧ minimum_age_requirement(?a) ∧ hasPolicy(?c, ?a) ∧ valid_driver_license(?d) ∧ valid_...
✓	budget_rule	Person(?p) ∧ budget(?c) ∧ minimum_age_requirement(?a) ∧ hasPolicy(?c, ?a) ∧ valid_driver_license(?d) ∧ valid_pa...
✓	dollar_rule	Person(?p) ∧ dollar(?c) ∧ minimum_age_requirement(?a) ∧ hasPolicy(?c, ?a) ∧ valid_driver_license(?d) ∧ valid_payr...
✓	thrifty_rule	Person(?p) ∧ thrifty(?c) ∧ minimum_age_requirement(?a) ∧ hasPolicy(?c, ?a) ∧ valid_driver_license(?d) ∧ valid_pay ...
✓	travCar_rule	Person(?p) ∧ travcar(?c) ∧ minimum_age_requirement(?a) ∧ hasPolicy(?c, ?a) ∧ valid_driver_license(?d) ∧ valid_pa...
✓	westCoast_rule	Person(?p) ∧ westcoast(?c) ∧ minimum_age_requirement(?a) ∧ hasPolicy(?c, ?a) ∧ valid_driver_license(?d) ∧ valid_...

Figure 2.2: Framework with rules describing car rental requirements extracted from text.

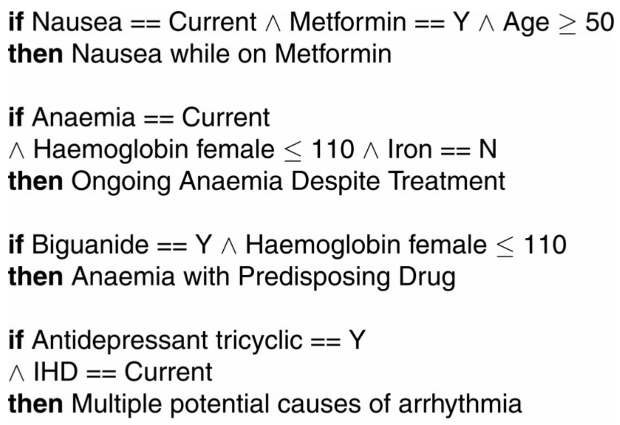
2.1.3 Audio

The use of audio interfaces is a subject that is starting to gain relevance in artificial intelligence, with the appearance of applications such as Siri and Alexa. D'este et al. have created a medication review robot [11] to monitor the medication and condition of aged patients using a rule-based system that uses audio. The robot receives information from the patient, takes sensor readings, uses the data gathered to make inferences using the expert knowledge system, and then provides the conclusions to the patient. The robot can consider the patient's changing condition, consult the medication review, and give recommendations like eating something when the patient has low blood sugar, reminding them to taker their medications or reduce medication. For all of this to be possible, an expert system called Ripple Down Rules (RDR) was implemented. RDR is a method of incrementally adding rules from batch and online learning to form a knowledge base. This is an excellent method to implement in a robot because it is simple enough to work with, and it is continuously updated using the robot's interactions with its environment.

To perform a medication review, the robot asks specific questions. It takes the patient's answers as input, which it analyses using context awareness and finding the keywords related to the question that was asked. The patient's responses are used to update their medical record, and if necessary new

[1] https://nlp.stanford.edu/software/lex-parser.shtml

rules are created. Once all the questions are answered and the robot has the necessary information, it attempts to match said information with the existing rules' conditions. Finally, the robot makes recommendations based on the rules in which the conditions were met. In fact, the rule system created by the robot uses basic propositional statements (Fig. 2.3), but since it is created using speech-to-text recognition, it made sense to separate it in a new section.



if Nausea == Current $\wedge$ Metformin == Y $\wedge$ Age $\geq$ 50
then Nausea while on Metformin

if Anaemia == Current
$\wedge$ Haemoglobin female $\leq$ 110 $\wedge$ Iron == N
then Ongoing Anaemia Despite Treatment

if Biguanide == Y $\wedge$ Haemoglobin female $\leq$ 110
then Anaemia with Predisposing Drug

if Antidepressant tricyclic == Y
$\wedge$ IHD == Current
then Multiple potential causes of arrhythmia

Figure 2.3: Example of rules from Medication Review Robot.

2.2 Visual rule systems

In this section, we explain visual approaches to create rules in a rule-based system and give an example for each of the approaches. We enumerate each method's advantages and disadvantages and use this information to decide which features are relevant to our project.

2.2.1 Flowcharts

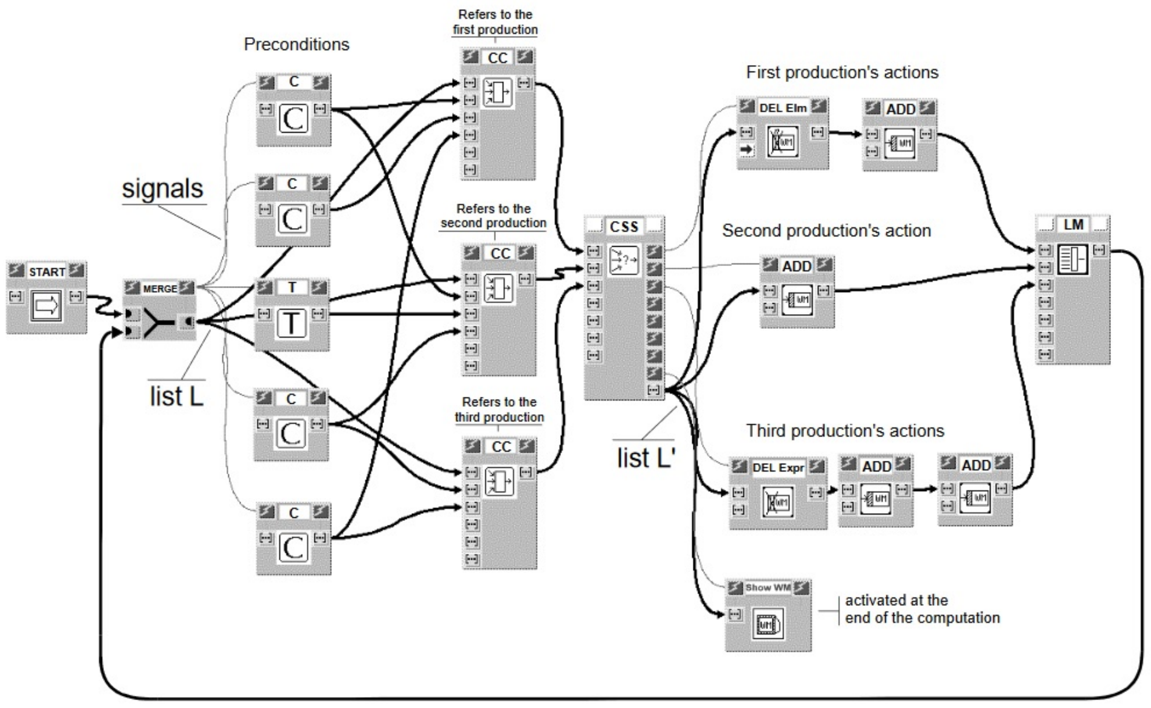
A flowchart is a type of diagram that describes an algorithm step-by-step, referring to every possibility and giving all of them a path that the system should follow if the corresponding condition is met. Given that this representation is so simple and provides such a clear path to follow in each situation that may appear, flowcharts are very popular in computer science. It has even been used in teaching programming [12].

The use of flowcharts to create rules provides a more visual representation. In the flowchart, each box represents a part of the rule that can be connected to other parts of the rule using a logic link. Each logic link comes from a condition and will be connected to a box with a different shape, representing the result of the condition being met. Using this method, the user can create rules with all kinds of complexity. Although this is a visual representation and is easy and natural to understand, it has the disadvantage of requiring a certain level of abstraction that undergraduate users do not have.

The use of flowcharts to create rule has been implemented [13] and analyzed. In this case, two methods to visually build rules have been tried.

The first one (Fig. 2.4) has condition blocks, test blocks, and three different productions. There is also a Condition Collector (CC) that analyses the inputs and determines if it is possible to apply the rule. List L is a container for information on past and present actions. There is also a Conflict Set Solver (CSS) block, which has various types and is used to choose which one of the three production exemplars should be activated. Add (ADD) and delete (DEL) blocks are used to add/delete a symbolic structure from memory. Finally, there are also merge blocks (MERGE and LM, list merge).

The second one does not use Condition Collector blocks because list L will be analyzed and modified by condition and test blocks throughout the scheme in a distributed manner. This will make necessary the repetition of conditions, but it will make it easier to identify and modify conditions.



Preconditions
Refers to the first production
Refers to the second production
Refers to the third production
signals
list L
list L'
START
MERGE
C
T
CC
CSS
First production's actions
DEL Elm
ADD
Second production's action
Third production's actions
DEL Expr
Show WM
activated at the end of the computation
LM

Figure 2.4: Scheme of data-flow visual language.

2.2.2 Matrices

Matrices are an alternative way to create a clear visual representation of a rule. There are two inputs, one represented in the rows and the other in the columns. The output for each pair of inputs is represented in the corresponding cell. This method has the upside of forcing the rule base's completeness because each cell should be filled, meaning that every scenario is covered. On the other hand, it has the downside of only allowing two inputs simultaneously, which makes it impossible for the user to create more complex rules that depend on more than two variables. FuzzyTECH software supports different kinds of rule editors, being one of them a matrix rule editor. The matrix rule editor window (Fig. 2.5) displays the two variables and the matrix representing the relation between them. Each white square of the matrix represents a rule, and the black ones represent rules that have not been defined yet. The user can define a new rule by double-clicking the corresponding white square and can delete an existing rule by double-clicking its black square. There are three lists with the IF and THEN statements' values at the bottom of the editor window. The selected rule is indicated by a red square in the matrix and a red highlight in the lists' respective lines. The user can also add a weight to each rule by changing the field "Degree Support", where a value of zero represents an implausible rule, and a value of 100 represents an entirely plausible rule.

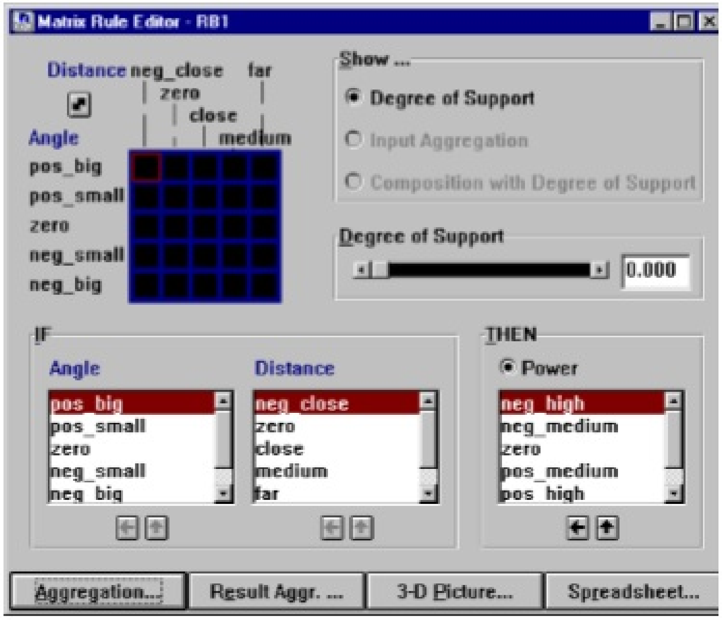
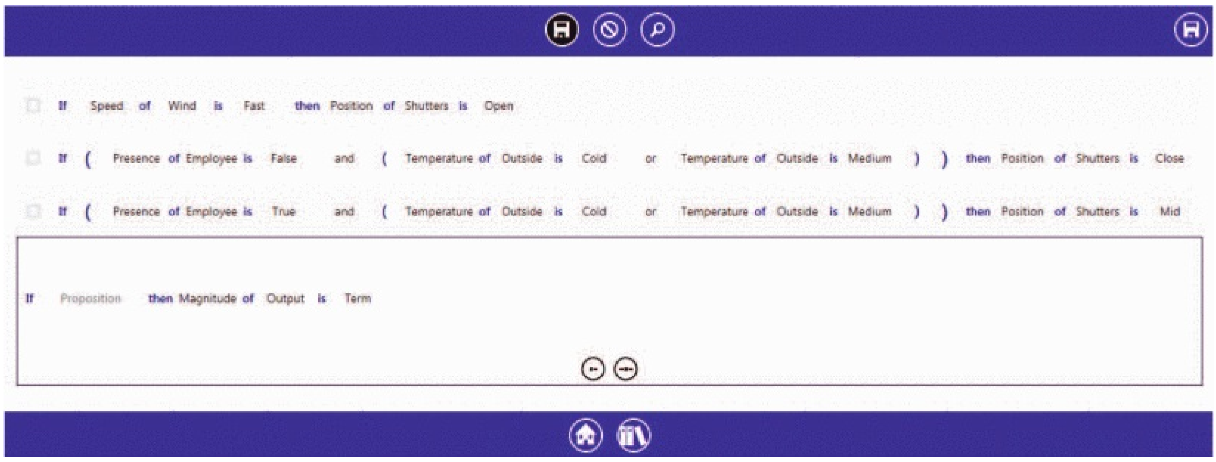


Figure 2.5: Matrix rule editor from FuzzyTECH.

2.2.3 Drag and drop

Drag and drop method [14] for creating rules uses a "filling in the blanks" design, where the possible inputs and outputs for the system have been previously created. The user produces rules by choosing its general form, and then it is possible to move the different parts of the rule or add operators, which increases the complexity of the rule. Then, the user must fill in blanks with propositions to complete the rule. To ensure that the user has some guidance during this procedure it was created a tool to show possible actions, called *feedforwards*, and a tool to help the user achieve a result based on his last action, called *feedbacks*. To add a rule, the user has to click the plus sign, and the rule being edited will be highlighted. Then, the user has to select the proposition blank which will make a pie menu appear to select the type of proposition which has two possible forms: magnitude OF input IS adjective; or input EQUALS value. The customizable terms of the rule (magnitude, input, adjective, and value) can be changed using one item from a list of options. Adding operators works in a similar way, the user clicks on the operator blank, and a pie menu will appear with the possible options.

This rule editor is as generic as possible, so it can be adapted and used in different domains and by different kinds of users, just by changing the basic inputs and outputs that the system works with.


If Speed of Wind is Fast then Position of Shutters is Open

If (Presence of Employee is False and (Temperature of Outside is Cold or Temperature of Outside is Medium)) then Position of Shutters is Close

If (Presence of Employee is True and (Temperature of Outside is Cold or Temperature of Outside is Medium)) then Position of Shutters is Mid

If Proposition then Magnitude of Output is Term

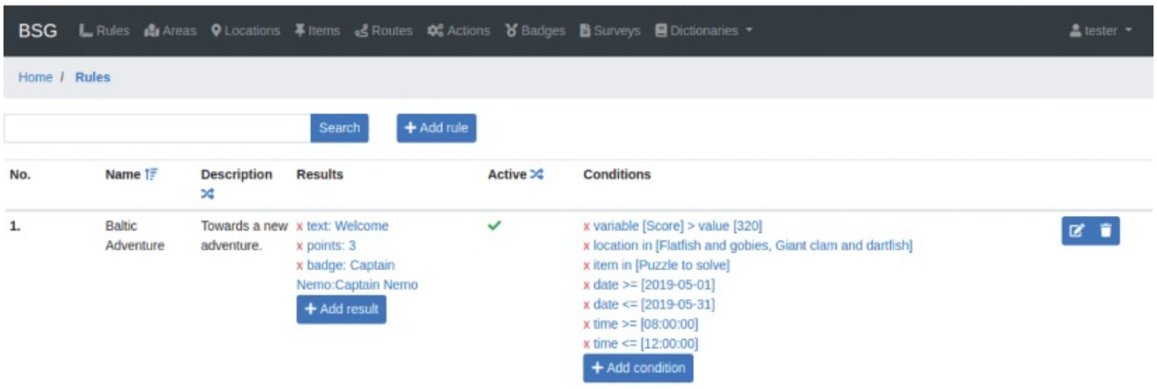
Figure 2.6: Example of rule creation using drag and drop.

2.2.4 Rule Editor

Another method studied is a rule editor that provides a more visual way to create rules. This has been used in tourism [15] by textitBalticMuseums: Love IT! international project [2] which has an e-guide gamification web service for tourists. The visual rule editor was created to help the addition, view, modification, and deletion of rules for this service, which tourists can use on their mobile devices. The editor (Fig. 2.7) is pretty simple and clearly separates the rules into three parts: name, conditions, and results. The conditions and results are specified next to the rule's name and there can be as many as the designer wants. There are buttons to edit a rule, add a new one, and to add conditions and results. When editing a condition, the designer has to specify the variable that will be used, the comparison operator, and the threshold for the condition. When editing a result, the designer has to specify the type of result that can be a badge, points, or even text, and then choose the badge, text, or number of points given to the user.

A rule editor has also been used in a project regarding the Internet of Things [16]. Here, the authors defend that objects have events, states that can be used as conditions and methods representing

[2] http://bmloveit.usz.edu.pl/



No.	Name	Description	Results	Active	Conditions
1.	Baltic Adventure	Towards a new adventure.	x text: Welcome x points: 3 x badge: Captain Nemo:Captain Nemo	✓	x variable [Score] > value [320] x location in [Flatfish and gobies, Giant clam and dartfish] x item in [Puzzle to solve] x date >= [2019-05-01] x date <= [2019-05-31] x time >= [08:00:00] x time <= [12:00:00]

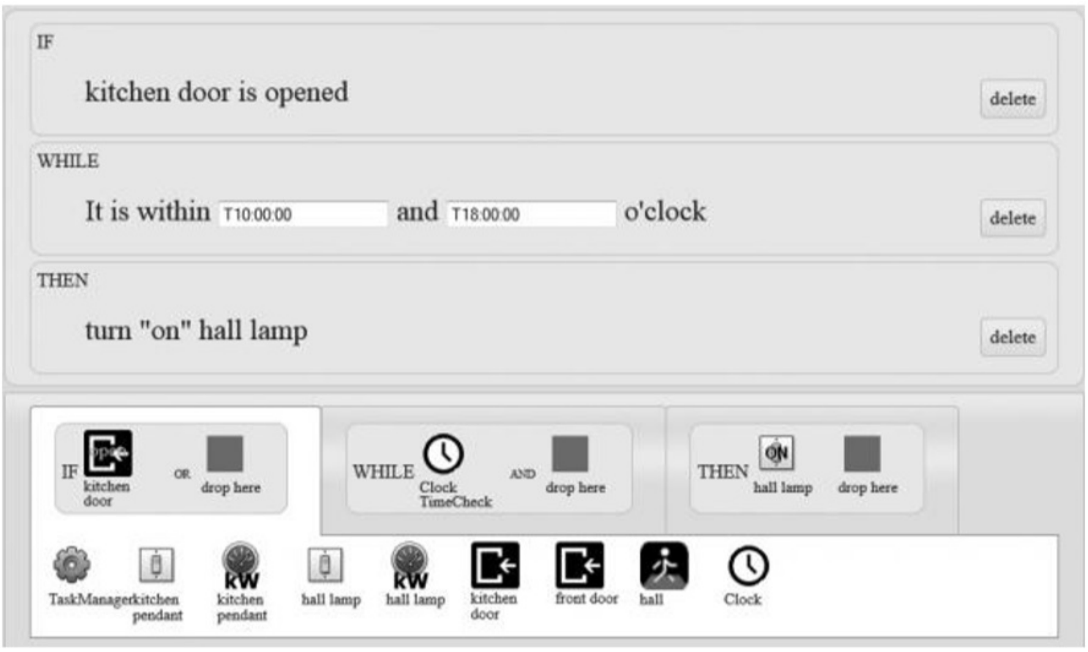
Figure 2.7: Visual rule editor used in tourism.

the results or actions. A workshop was organized with the project's stakeholders to decide the most appropriate way to represent this, and the results and feedback were documented.

First, the editor was created using paper, where the user had to create a rule by choosing the variables (in this case, objects) used in the IF, WHILE, and THEN area. Then, in the edit menu, the user had to specify each area's details, what was happening or should happen to the objects being analyzed. Finally, after defining a description, the rule could be saved in the list of existing rules.

After the tests with paper, it was created a rule editor implemented in Javascript [3]. On this rule editor, the rules can be aggregated into tasks, and each rule has one or more IF clauses, zero or more WHILE clauses, and one or more THEN clauses. The editor has a list of all the existing rules. Each rule has a symbol of the object associated with it, the name of the object's location, and a description of the rule's conditions and results. On the editing interface (Fig.2.8) there is a tab for object, condition, and result, and in each tab, the user has to drag and drop the available components, which are shown in each tab, to the grey slots available. When a component is dropped on a grey slot, the user has to choose a template and add the clause's missing parameters.

[3]https://developer.mozilla.org/pt-PT/docs/Web/JavaScript


IF

kitchen door is opened

WHILE

It is within T10:00:00 and T18:00:00 o'clock

THEN

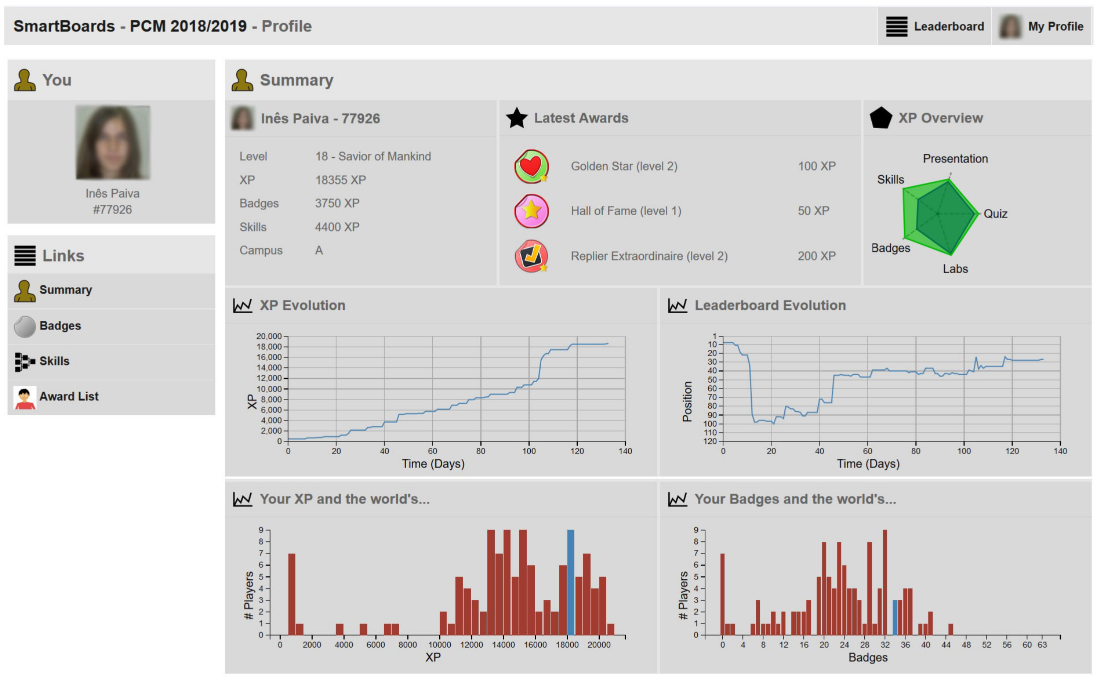
turn "on" hall lamp

Figure 2.8: Rule Editor used in the Internet of Things.

2.3 Gamification

As previously stated, gamification has been a subject of study as a learning method and has shown promising results in motivating students. This happens because games are interactive and engaging, something that education lacks sometimes. Gamification tries to combine gaming features and education, making it more interesting for students. Competition, the feeling of accomplishment and improvement, and receiving feedback and rewards for effort motivates students and drives them to be better.

MCP consists of a semester-long MSc course in the Information Systems and Computer Engineering degree at IST and will be the primary environment of this thesis. This course has been using gamification for the last several years and is an excellent example of how it can motivate students [17], clearly showing promising results in increasing motivation and participation. It uses six game elements: XP, levels, a leaderboard, challenges, badges, and a skill tree. Students are awarded XP for completing course tasks and can track their progress (Fig. 2.9) and their colleagues' progress using the leaderboard (Fig. 2.10), which was created using Smartboards [3]. Badges are given for achieving certain goals and the skill tree is composed by a set of tasks that the students can complete to earn more points. The way the course was created provides competition by providing feedback through points and badges, autonomy by offering different ways to reach higher levels, and relatedness by using forums that create a sense of community. When comparing the non-gamified version of the course with the gamified one, the teachers concluded that the results got better in the later, with an increase of participation and activeness.


SmartBoards - PCM 2018/2019 - Profile

Leaderboard | My Profile

You

Inês Paiva
#77926

Links

- Summary
- Badges
- Skills
- Award List

Summary

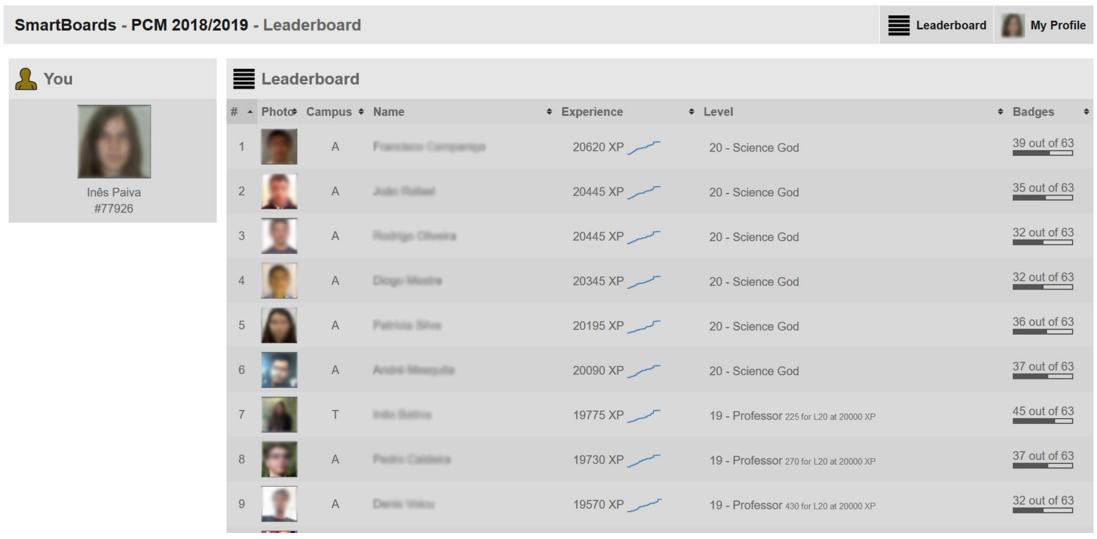
Inês Paiva - 77926

Level	18 - Savior of Mankind
XP	18355 XP
Badges	3750 XP
Skills	4400 XP
Campus	A

Latest Awards

Golden Star (level 2)	100 XP
Hall of Fame (level 1)	50 XP
Replier Extraordinaire (level 2)	200 XP

Figure 2.9: PCM student's profile on Smartboards.


SmartBoards - PCM 2018/2019 - Leaderboard

Leaderboard | My Profile

You

Inês Paiva
#77926

Leaderboard

#	Photo	Campus	Name	Experience	Level	Badges
1		A	[redacted]	20620 XP	20 - Science God	39 out of 63
2		A	[redacted]	20445 XP	20 - Science God	35 out of 63
3		A	[redacted]	20445 XP	20 - Science God	32 out of 63
4		A	[redacted]	20345 XP	20 - Science God	32 out of 63
5		A	[redacted]	20195 XP	20 - Science God	36 out of 63
6		A	[redacted]	20090 XP	20 - Science God	37 out of 63
7		T	[redacted]	19775 XP	19 - Professor 225 for L20 at 20000 XP	45 out of 63
8		A	[redacted]	19730 XP	19 - Professor 270 for L20 at 20000 XP	37 out of 63
9		A	[redacted]	19570 XP	19 - Professor 430 for L20 at 20000 XP	32 out of 63

Figure 2.10: PCM leaderboard.

Apart from MCP, there have been other studies where gamification has shown promising results in software engineering courses [18], providing various challenges through the weeks and assessing what the students have learned each week, concluding that the topic that was part of the challenge that week was the topic that students engaged more with. This is because gamification provides rewards, challenges, achievements, learning, motivation, and user engagement.

Gamification was also use in other areas such as: Improving viewers engagement in TV commercials

[19], breast cancer diagnosis [20] and smartphone applications [21].

2.4 Discussion

Rule-based systems have been in use for several years and have shown excellent results, using an inference engine and a rule base to analyze facts and act on them. The study in Section 2 helped identify different ways of creating rules and understand them by analyzing real examples. These examples helped understand which are the main advantages and disadvantages of using each of the methods studied. Although there are many ways of creating these systems, few examples of these approaches are used in some cases.

Table 2.1 analyzed some crucial characteristics and whether or not the methods studied had them. First, having a visual approach when creating rules can make it easier for people to interact with the system. Also, allowing multiple inputs is essential for the system created because working with only two, like in the matrices, will prevent the creation of more complex rules. Then, intuitivity, which we related with if it was easy or not for people outside of the computer engineering area to use the system, it was concluded that propositional statements and flowcharts failed in this case. Another aspect that could be interesting to implement is using a tool that gives the user possible ways to complete the rules based on other rules already created. In this case, both drag and drop and rule editor approaches covered this. Finally, in the use of these methods in touch devices, some cannot be used because of the complexity of the schemes requiring bigger screens.

Table 2.1: Rule editing approaches' characteristics.

	Visual	Multiple Inputs	Expert Knowledge	Autocomplete
Propositional Statements		x	x	
Natural Language Processing		x	x	
Audio		x	x	
Flowcharts	x	x		
Matrices	x		x	
Drag and Drop	x	x		x
Rule Editor	x	x		x

In our case, we will be using a visual approach with a rule editor because it does not require programming skills to create rules, allowing the end-user to create and edit rules without having the technical knowledge that some of the other approaches require. Also, a graphical approach should be easy to learn and use, giving the user almost step-by-step guidance on how to proceed. It is essential to make sure that the editor that will be created allows for multiple inputs, and that it has an auto-complete feature that helps create new rules.

Another thing that will be explored is implementing a way for the system to deal with grade retractions, which is a feature that was not mentioned in any of the articles studied, but it is essential in our case. Some of the rules will depend on the teacher’s grades, and this feature will make sure that the system undoes actions and performs changes whenever these grades change.

3
Gamecourse

Contents

Gamecourse is a platform, developed in PHP [1], that was created to support MCP and its gamification features. It represents the link between teacher and students, and it is where all the information regarding the course is stored. It allows students to share their work with others, receive grades, check their progress, among other things.

In the beginning, Gamecourse consisted of a set of static web pages that were generated by a script that had to be manually run and that displayed the game elements of MCP [22]. In 2013, an MSc student started working on a web application to replace this script as his MSc project, but it was never finished [23]. Then, in 2016 André Baltazar developed SmartBoards, a web application composed of a leaderboard and profile pages for the students that allowed for customization. Three years later, Alice Dourado improved the system by making it more flexible, scalable, and configurable, which allowed it to be applied to courses other than MCP [24]. In the same year, Matilde Nascimento created a web application that allowed teachers to keep track of the students that attended lectures [2]. Nowadays, apart from myself, two other MSc students are working on this platform, Diana Lopes on the back-end [1] and Patrícia Silva on the front-end [25]. In this chapter, we will explain the aspects of GameCourse that were important to our system, and in chapter 5, we will explain how both of them interact.

3.1 Architecture

Gamecourse consists of a system that works with a set of web applications that work together to provide a good gamification experience for MCP students (Fig. 3.1). These applications are the following:

- GoogleSheets: spreadsheet where each sheet is reserved for a teacher. Sometimes teachers must give a grade or a badge manually, so this is where they will do it.
- Moodle: is a free and open-source Learning Management System (LMS) that is used in MCP as a place for students to share works, receive grades, participate in forums, and answer quizzes.
- ClassCheck: it is a web application developed by Matilde Nascimento [2] to help teachers keep track of the students that attend lectures by giving students a code composed of letters and numbers to confirm their attendance.
- Moodle: learning platform where students publish their works, receive grades and complete quizzes.
- QR module: application used to store the participations that the students have in class. Any time a student answers one of the teacher's questions the teacher gives the student a QR code that should be redeemed and will enter the system as a participation in class.

[1]https://www.php.net/

Each of the applications has a specific plugin created by Diana Lopes [1] that retrieves data from them and stores it in a relational database. Apart from these applications, there is also Smartboards. To provide the necessary information to fill SmartBoards, a teacher must run a script that uses a set of text files as input and generates a new file with all the awards that were unlocked by the students. This process has to be performed manually, which wastes the professors' time. Furthermore, it runs all the data from the beginning of the semester every time that it runs, which is very inefficient.

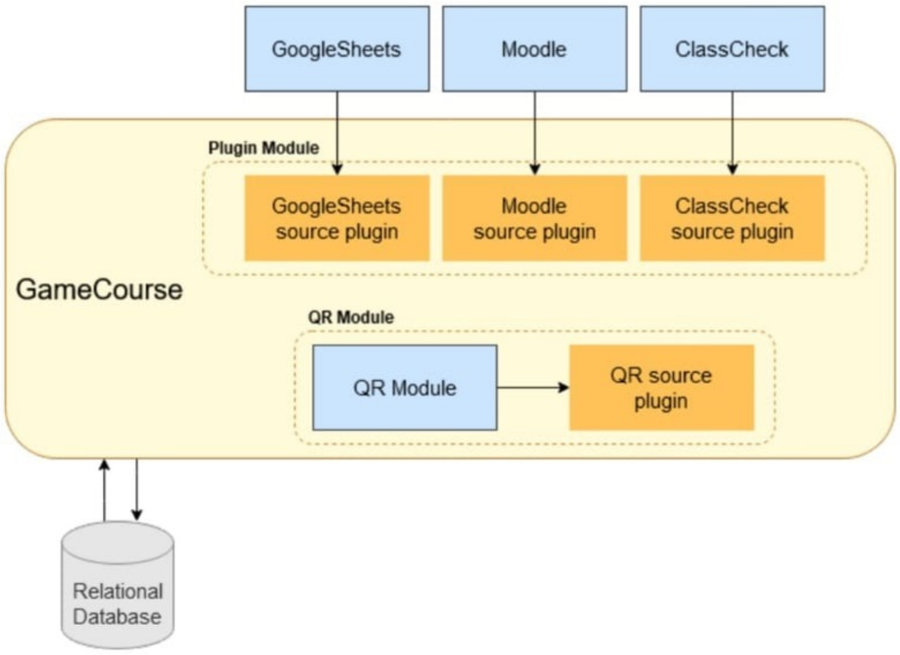


Figure 3.1: GameCourse's architecture [1]

3.2 Database

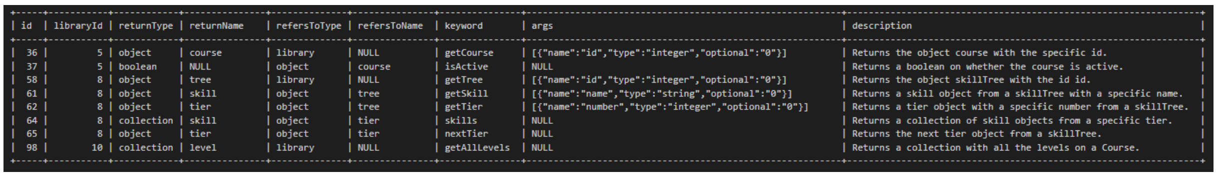
One of the main components of GameCourse is a database that stores all the information regarding the courses created. This database consists of tables that can either regard the system's essential components or be related to specific modules. The main tables are automatically created by the system when the course is created, while the tables related to specific modules will only be created when the module is enabled.

Although some of the tables are filled automatically, one of the main ones, which is the one that stores all the information from the external sources, requires a few extra steps. First, the administrator has to enable the plugin module, which allows the system to retrieve data from the data sources. It is then necessary to set a periodicity and configure each of the plugins needed for the course. After performing these steps, the system is ready to retrieve information from its sources. This information will consist of all the data that may be important for calculating awards for the students. From ClassCheck, the system will retrieve attendances; from Moodle, it will retrieve posts and grades from quizzes and skills; from the QR module, it will retrieve students' participations in class; finally, from GoogleSheets, it will retrieve any other grade or badge that the teachers need to award manually. This data is stored in a table called participation. Each one of the lines in this table refers to an action performed by a student for a course. From now on, every time we mention the term participation, we are referring to one of these actions. In Appendix A we provide a full version of Gamecourse's database entity-relationship model.

3.3 Expression Language

Gamecourse has its own expression language, created to be a uniform and versatile way of interacting with the system's different modules. This language is structured with libraries, each with its functions. Some of these functions are always available because they are related to the system's basic concepts, whereas others only become available after the corresponding module is enabled for the course.

The expression language is described in a dictionary (Fig. 3.2) that has a list of all the libraries and functions available for a course, along with information about the functions, such as its arguments, definitions, keywords, respective libraries, and what data type it returns.

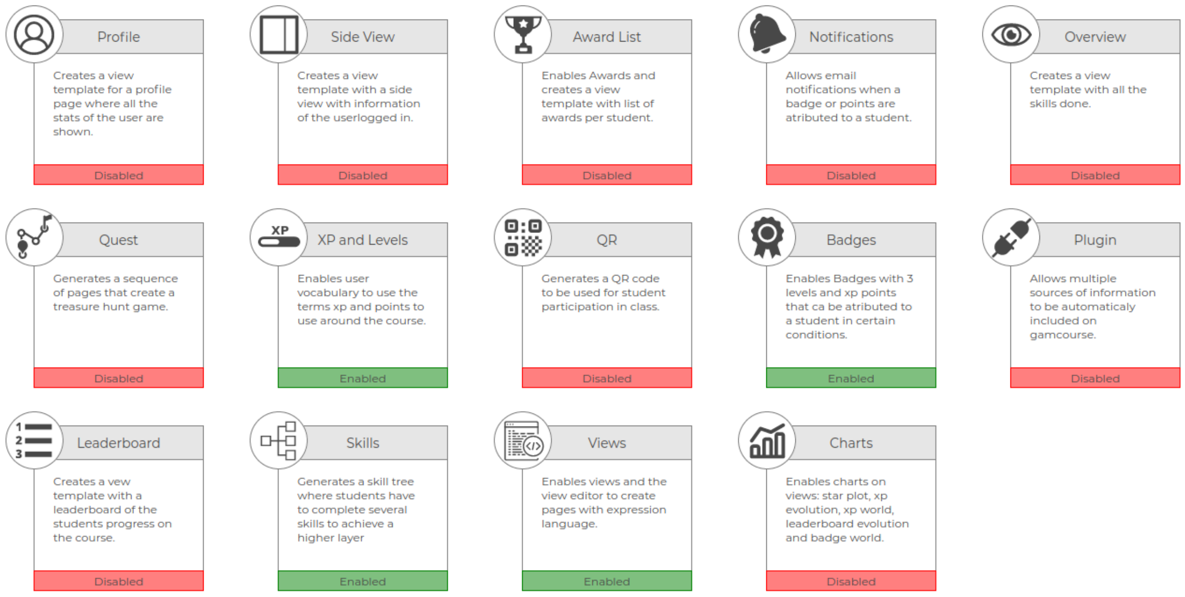


```
+-----+-----------+------------+---------------+--------------+---------------+--------------+----------------------------------------------------------+-----------------------------------------------------------------+
| id  | libraryId | returnType | returnName    | refersToType | refersToName  | keyword      | args                                                     | description                                                     |
+-----+-----------+------------+---------------+--------------+---------------+--------------+----------------------------------------------------------+-----------------------------------------------------------------+
|  36 |         5 | object     | course        | library      | NULL          | getCourse    | [{"name":"id","type":"integer","optional":"0"}]          | Returns the object course with the specific id.                 |
|  37 |         5 | boolean    | NULL          | object       | course        | isActive     | NULL                                                     | Returns a boolean on whether the course is active.              |
|  58 |         8 | object     | tree          | library      | NULL          | getTree      | [{"name":"id","type":"integer","optional":"0"}]          | Returns the object skillTree with the id id.                    |
|  61 |         8 | object     | skill         | object       | tree          | getSkill     | [{"name":"name","type":"string","optional":"0"}]         | Returns a skill object from a skillTree with a specific name.   |
|  62 |         8 | object     | tier          | object       | tree          | getTier      | [{"name":"number","type":"integer","optional":"0"}]      | Returns a tier object with a specific number from a skillTree.  |
|  64 |         8 | collection | skill         | object       | tier          | skills       | NULL                                                     | Returns a collection of skill objects from a specific tier.     |
|  65 |         8 | object     | tier          | object       | tier          | nextTier     | NULL                                                     | Returns the next tier object from a skillTree.                  |
|  98 |        10 | collection | level         | library      | NULL          | getAllLevels | NULL                                                     | Returns a collection with all the levels on a Course.           |
+-----+-----------+------------+---------------+--------------+---------------+--------------+----------------------------------------------------------+-----------------------------------------------------------------+
```

Figure 3.2: Examples of functions from Gamecourse's expression language.

3.4 Modules

Gamecourse allows the use of a set of modules (Fig. 3.3), which can increase the complexity of the course's gamification experience. The administrator of the course has the power to enable or disable the modules as he sees fit. Some modules require other modules to be enabled, and enabling a module adds more functionalities to the course's platform by allowing more features to the game. That is why Gamecourse automatically adapts itself to make sure that it can handle the increased complexity by changing the database and adding new functions to the expression language. For each module, there is a configuration page where the administrator can define its essential features and configure the module's view (i.e., how the module's page is presented to the users).


Profile
Creates a view template for a profile page where all the stats of the user are shown.
Disabled

Side View
Creates a view template with a side view with information of the userlogged in.
Disabled

Award List
Enables Awards and creates a view template with list of awards per student.
Disabled

Notifications
Allows email notifications when a badge or points are atributed to a student.
Disabled

Overview
Creates a view template with all the skills done.
Disabled

Quest
Generates a sequence of pages that create a treasure hunt game.
Disabled

XP and Levels
Enables user vocabulary to use the terms xp and points to use around the course.
Enabled

QR
Generates a QR code to be used for student participation in class.
Disabled

Badges
Enables Badges with 3 levels and xp points that ca be atributed to a student in certain conditions.
Enabled

Plugin
Allows multiple sources of information to be automaticaly included on gamcourse.
Disabled

Leaderboard
Creates a vew template with a leaderboard of the students progress on the course.
Disabled

Skills
Generates a skill tree where students have to complete several skills to achieve a higher layer
Enabled

Views
Enables views and the view editor to create pages with expression language.
Enabled

Charts
Enables charts on views: star plot, xp evolution, xp world, leaderboard evolution and badge world.
Disabled

Figure 3.3: Available modules for Gamecourse.

4

Gamerules

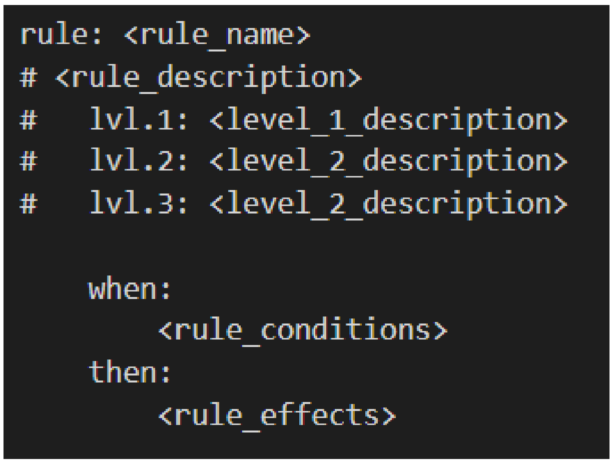
Contents

Gamerules is a rule system that was developed in Python [1] by an MSc student from IST, as a project thesis, but was never finished. This project's objective was to create a rule system that would substitute the script that was being used for MCP, a gamified course, to calculate the course's awards. This was necessary in order to make the system more automatic and efficient.

Although this was never finished, the system created already had a few essential features implemented that would later be very helpful for Autogame. This chapter will explain further and in detail how Gamerules works and some of its most important features.

4.1 Rules

A rule-based system is a system that applies human-made rules to manipulate data. Gamerules, being a rule-based system, requires the creation of a set of rules to work with. These rules consist of text files, each containing a rule that follows a specific structure (Fig. 4.1).



```
rule: <rule_name>
# <rule_description>
#   lvl.1: <level_1_description>
#   lvl.2: <level_2_description>
#   lvl.3: <level_2_description>

    when:
        <rule_conditions>
    then:
        <rule_effects>
```

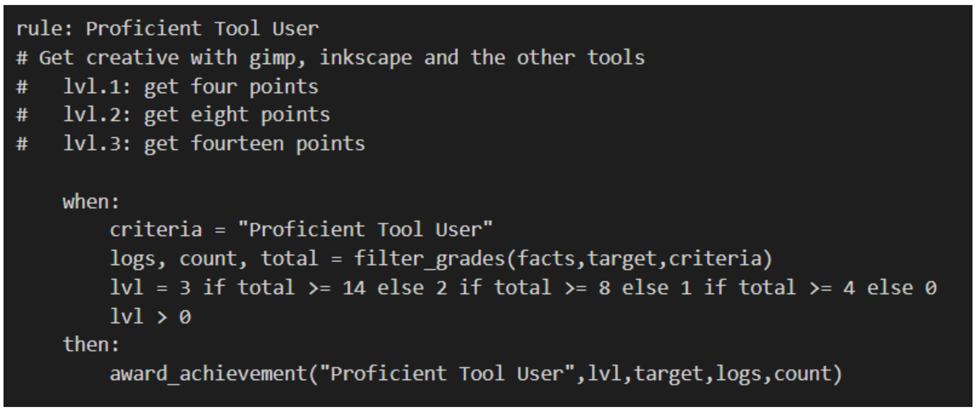
Figure 4.1: Generic rule used in Gamerules.

As shown in Fig. 4.1 rules are composed by four parts:

- **Name**: it is the name of the rule. This is very important because there are functions that refer to specific rules by using its name.
- **Description**: it only exists to make the rule more readable. The system will not do anything with this information.
- **When**: it is composed of assignment statements and precondition statements. The preconditions need to be fulfilled in order to fire the effects of the rule
- **Then**: it is a set of one or more effects that should happen if the preconditions are met.

[1] https://www.python.org/

Although Gamerules was never finished, João Rego, its developer, had already created a set of rules with the intent of using them for MCP. This set of rules is composed of 25 rules that refer to the course's badges. However, some of them are incomplete and do not work correctly. In Fig. 4.2 we provide the code for the *Proeficient Tool User* badge as an example of a rule that was part of Gamerules.



```
rule: Proficient Tool User
# Get creative with gimp, inkscape and the other tools
#   lvl.1: get four points
#   lvl.2: get eight points
#   lvl.3: get fourteen points

    when:
        criteria = "Proficient Tool User"
        logs, count, total = filter_grades(facts,target,criteria)
        lvl = 3 if total >= 14 else 2 if total >= 8 else 1 if total >= 4 else 0
        lvl > 0
    then:
        award_achievement("Proficient Tool User",lvl,target,logs,count)
```

Figure 4.2: Gamerules rule: Proeficient Tool User badge.

A rule is composed of a set of assignments, functions, and conditions. Any assignment that is done on the when block can also be used in the then block. Functions used in the rule's body, such as filter_grades and award_achievements, have to be defined using a particular decorator, which will be explained in the next section. In this case, the rule uses the filter_grades function to filter the awards that are important for this badge. It will then compute the badge level by using one of the outputs of the function used. After, it checks if the condition is true, and finally, if it is, it will award the badge.

4.2 Rule parser

After creating the rules, the system must know how to interpret them. This is where the rule parser comes in. The rule parser is responsible for going through every character of the rule and separating it in name, description, when and then. To accomplish this, the parser looks for keywords. First, it saves everything after "rule:" as the rule name. Then, it ignores every comment because this is not necessary for any calculation. The conditions are everything that comes after "when:" and are saved and compiled one by one. And finally, as for the effects, it looks for everything after "then:".

Given the fact that this parser works by analysing the rule file character by character and compiling each line of the then and when blocks separately, it doesn't have previous knowledge. This means that it cannot run statements that occupy more than one line, such as *for* and *while* loops. However, the user can easily go around this by creating new functions anytime a rule needs more complicated code than what the rule parser can deal with.

To facilitate the use of additional functions on the rules, two function wrappers were created, one for functions meant to be used in the when block, and one for the ones meant to be used on the then block. These wrappers are @rule_function and @rule_effect, respectively. Whenever it is necessary to add a new function, we create it, import it to the *gamefunctions.py* file with the respective wrapper, and it is ready to be used within the rules.

Gamerules already had a list of functions implemented, which were being used in the rules. This functions are shown in Table 4.1 and are detailed below:

- **rule_unlocked**: it receives a target and the name of a rule and returns True if the target has unlocked any rule with that name. This is very useful if it is necessary to chain rules. This way, we can use the fact that a student has fired another rule as the precondition for a rule.
- **effect_unlocked**: this function works precisely as rule_unlocked. However, it refers to effects and not to rules. If the target has unlocked a given effect, this function will return True.
- **award_achievement**: it is used to give the awards to the students. It receives a target, achievement name, level, and a list of logs that contributed to getting this target this badge. It will then return a particular object, called *Prize*, that has the list of awards and the respective logs.
- **award_grade**: works similarly to award_achievement. However, it is used for awarding grades. In this case, the function receives a description and XP instead of badge name and level. It will also compile everything into a Prize object.
- **award_treeskill**: again, similar to the two functions above. This one awards a skill from the skill tree. It only receives the target, skill name and logs. It will also create a Prize object.

Table 4.1: Gamerules's auxilar functions for MCP.

Name	args	description	output
rule_unlocked	rule_name (str), target (int)	Returns True if the rule has been unlocked	result (bool)
effect_unlocked	effect_name (str), target (int)	Returns True if the effect has been unlocked	result (bool)
award_achievement	badge_name (str), target (int), lvl (int), contributions (list)	Create a Prize object with the awards regarding badges	Prize (object)
award_grade	description (str), target (int), xp (int), contributions (list)	Create a Prize object with the awards regarding grades	Prize (object)
award_treeskill	skill (str), target (int), contributions (list)	Create a Prize object with the awards regarding skills	Prize (object)

4.3 Gamerules' input and output

Gamerules is a system that was intended to only use text files. Its input consists of a set of text files with all the information from the course. It had three categories of files: course, gamification, and logs. The first one consists of two files, one is a list of teachers, and the other is a list of students. There are three files for the second category: one for badges, one for levels, and one for the skill tree. Finally, there are moodle logs, moodle quiz grades and spreadsheet logs for the third category. All of these last files had to be downloaded from their respective source every time the system ran. The system would then go through all the files before firing the rules and create all the respective objects.

Apart from the configuration files mentioned above, the system also received a list of rules to calculate the awards. These rules were text files with a set of characteristics (Section 4.1) and were stored in a folder called rules so that the system knew how to access them.

As for the system's output, it is also a text file, called indicators. This file consists of a dictionary with all the awards that were attributed to the targets. Each student is used as a key in this dictionary and the respective value is another dictionary with all the student's awards. Each of these awards has a name, a list of logs that were used to calculate that award, and a set of arguments depending on the type of award. For skills, there is a rating, for badges there is a level and for grades there is the amount of XP. This output was intended to be used as the Smartboards' input, which allows the students to check their progress on the course.

4.4 Gamerules Procedure

Gamerules is a rule system, so it works by receiving facts and rules and calculating results. In this case, the files described in the last section apart from the rules, work as facts. So, the first thing that Gamerules does before even firing the rule system itself is going through all of these files and saving its data as system objects.

After gathering all the information necessary, it creates a *RuleSystem* object using the path for the rules folder. When this object is created, it automatically loads the rules folder and compiles the rules using the rule parser. Then, it calls a function that will fire the rule system, using the list of targets, logs, and scope as arguments. The first two arguments come from the text files used as input, whereas the scope can refer to any variables that can be manually added to be accessible in the rules, for example, additional variables, such as number of classes.

The rule system, when fired, will go through every target and fire each rule individually. When a rule is fired, the system executes its preconditions, and if all the preconditions are valid, it will execute the effects. If any of the preconditions is False, it will not execute the effects, and it will automatically continue to the next rule.

Gamerules also creates a data file to save important information. If it is the first time the system is running, it will create this data file, and if it is not, it will use the existent one. This file stores information on the rules, so the system only has to compile the rules one time, and the next time it runs, it will only access the data file instead of compiling the rules again. If there is any change made to the rules, the user needs to delete this file manually so that the system compiles the rules again and acknowledges the changes. This data file is also how the functions rule_unlocked and effect_unlocked know if a rule/effect has been unlocked.

Finally, with the results from the rules' effects, the system creates the indicators file that stores every award that was calculated.

4.5 Issues

Although Gamerules has the basic features of a rule system working, some issues make it impossible to use in MCP. The system that we need to create needs to fulfill some requirements that are not fulfilled by Gamerules. Here are some of the issues that we had to be dealt with:

- First, automation was one of the main requirements that the rule system needs to fulfill. We need a system that runs on its own without needing any intervention from a professor. Gamerules does not have any mechanism to deal with this.
- We need an incremental system. This was set at the beginning of the project. We need a system that runs only the necessary rules for the necessary data. At its current state, Gamerules needs all the logs from the semester to calculate a correct output, which is very inefficient.
- We need to make sure that the system knows how to deal with grade retractions. This is key because if a teacher needs to change a grade, the system has to delete or update the previous award instead of having two awards for the same data. Gamerules does not have any feature that allows the system to make these changes.
- Finally, it is fundamental that the system is integrated into Gamecourse and communicates with it. Also, we need it to work with Gamecourse's database instead of working with static files. At this point, Gamerules is an entirely separate system from Gamecourse and has zero communication with it. Consequently, it is necessary to change the text files manually if there is any change in the log files.

5 Autogame

Contents

The original plan for Autogame was to create a rule-based system to complement MCP's gamification experience and replace the script that was used in previous years. However, when we planned this system, we did not know what Gamerules was already able to do because it was unfinished, and we never had any document describing its state. At the beginning of this project, a considerable amount of time was spent understanding how Gamerules worked, what it could do, and what was usable for our system. After this process was finished, we concluded that the base of the rule system was already working, so we focused on its integration with Gamecourse, which was already a monumental task.

Autogame (Fig. 5.1), as is, is a rule system that, although being independent, is completely integrated with Gamecourse. It receives all of its inputs from Gamecourse, although in different stages of the process, and writes its output directly in the system's database. The configuration and rule files are located in the system's root folder. Then, it gets its targets by retrieving a list of new participations from the database and getting the students from that list. Moreover, when running, each rule calls functions that communicate with Gamecourse to retrieve the participations to calculate the respective awards.

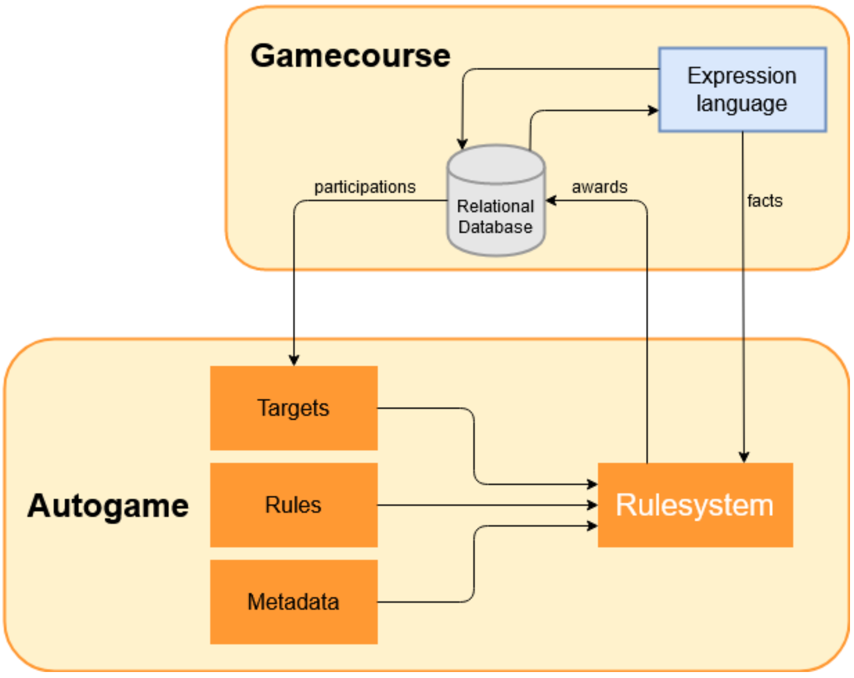


Figure 5.1: Autogame's architecture.

In terms of architecture, the main difference between this system and its ancestor is its inputs and outputs. Gamerules' had a static list of participations for the course, whereas Autogame only retrieves each rule's essential participations. Besides that, Gamerules had a complete list of students, which would be used as targets every time the system ran, whereas Autogame only retrieves the essential students to use as targets from the database. All the other changes made are not reflected in the system's architecture but will be explained in detail in this chapter.

5.1 Upgrading to python 3

One of the first problems that we encountered when trying to create Autogame was the fact that Gamerules was written in version 2 of python, which no longer made sense when this project began since there was already a version 3. So, the first step on this road to Autogame was updating Gamerules to python 3. This was achieved with the help of a tool from python called *2to3* [1] and a lot of research about what was being used and the best replacement for it. Most of the changes made were simple ones, such as exchanging function names, how a variable was encoded or how to make imports. However, given the system's complexity, this took a fair amount of time. After finishing this process and replacing the text files that were missing as input, we were already able to run Gamerules despite some issues that would be solved later on.

5.2 From text files to database

Gamecourse is a system that has been evolving since its creation, and as we previously discussed, one of the main changes developed by Diana Lopes [1], was a set of plugins that retrieve information from the external data sources to the system's database. When Gamerules was first created, it needed to retrieve data from each source, which no longer applies in our case because everything is now in one place (Gamecourse's database). With this improvement, we had also to update Autogame so that it knew how to communicate with this database.

We established a connection with the database by using MYSQL [2] libraries. The system needs a username and password to connect with the database, which the course administrator must place in a file called *credentials.txt*. This is essential to avoid manually writing this information in all the required functions. Instead, the user only writes it in the credential file, and the system has a function that reads the file any time it has to access the database.

Besides all of the above, it is essential to remember that Gamecourse had its data scattered in different sources, and Gamerules' objects and functions were created according to the layout of this information. With the creation of a uniformed database, some of the old objects no longer made sense and had to be replaced by one that fit the new participation format.

There are three situations when Autogame needs to access Gamecourse's database. First, to check when was the last time the system ran, which we explain in section 5.6. Second, to retrieve the rules' targets, also explained in section 5.6. Third, to write the system's output. In this last case, our system will write one line per student and achievement. The awards are written in a table called award. Every award has a course and a student, and the remaining elements depend on the type of award. There are

[1] https://docs.python.org/3/library/2to3.html
[2] https://www.mysql.com/

four types:

- Badge: each badge may have up to three levels, and each of the levels has its own line, so one badge may have up to three lines for the same student. A badge award has a badge name, badge level, and correspondent XP.
- Skill: each line corresponds to one skill a student has completed from the skill tree. A skill award has a skill name and correspondent grade and XP;
- Grade: corresponds to a quiz, laboratory, or presentation grade. A grade award has a name, number of quiz/lab, and grade.
- Other: in MCP's case this type is only used to award the initial bonus. Apart from the course and student, these awards only have an award name and correspondent XP.

In Fig. 5.2 we can show an Entity-Relationship model for the tables that are used in Autogame. A complete version of Gamecourse's diagram is provided in Appendix A.

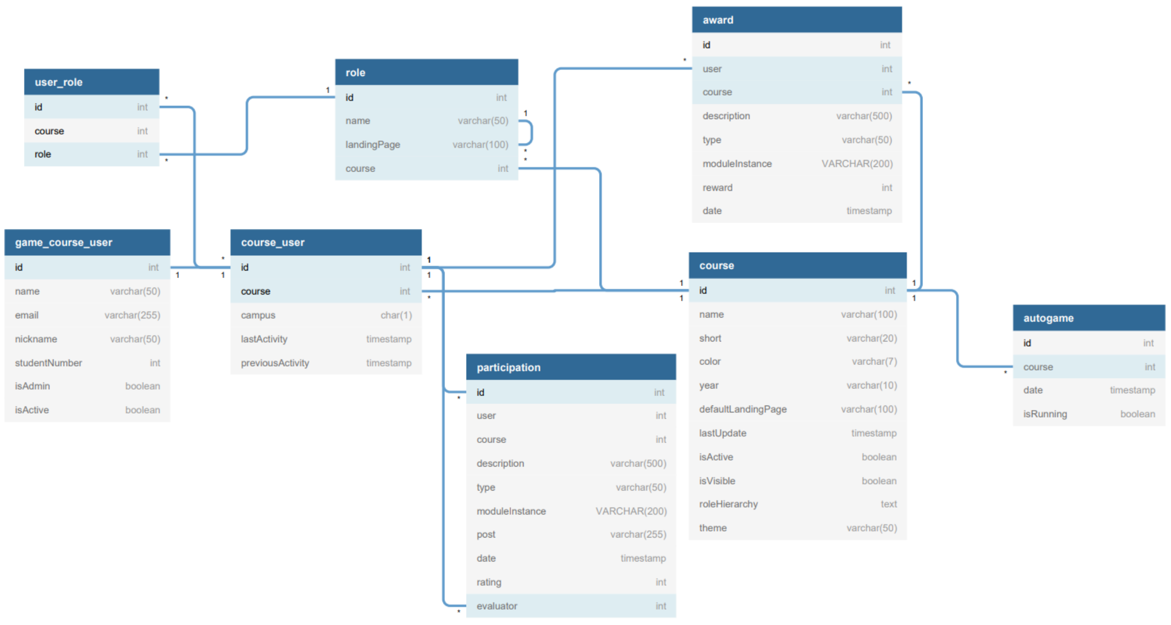


Figure 5.2: Gamecourse's Entity-relationship model - small version.

5.3 Communication with Gamecourse

To fully integrate Autogame with Gamecourse, it was essential that both could establish communication. This was important for two main reasons:

- **Expression language**: as we explained in section 3.3, Gamecourse has its own expression language that consists of a set of functions created to make it easier to communicate with the different modules of the system. Given the fact that in the future, we expect to have an interface on Gamecourse to help users create rules, it made perfect sense that the functions from the expression language could be used as a rule function or effect. To make this possible, Autogame had to be able to communicate with Gamecourse so that it could run the functions and return an output.
- **Firing Autogame**: one of the key features of Autogame is that it runs automatically. In order to do this, we needed Gamecourse to be able to call our system when needed.

Communication between the two systems would be easy to accomplish if both systems were developed using the same language. However, this is not the case. Gamecourse is in PHP, whereas Autogame was developed in Python. To create a channel for them to communicate, we did some research to find the best solution for our problem. We looked for the best Remote Procedure Call (RPC) frameworks that could help us, and we found Apache Thrift [3], which seemed like the right solution. Unfortunately, it had issues with some Ubuntu versions, which was a deal-breaker. Then, we found out about GRPC [4], another RPC system, developed by Google. Again, after trying to configure it, we concluded that this would not work either, this time because it had very little PHP support. Finally, we decided to create a Transmission Control Protocol (TCP) socket in PHP, which would act as a server, listen and wait for connections, and a Python socket that would act as a client and connect to the server when needed. After some trial and error, we managed to establish communication between both systems, using a PHP script and a python script, which we run manually.

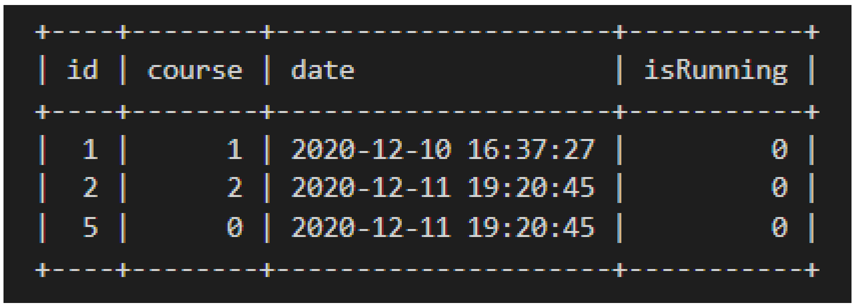
After succeeding in establishing this connection, we first focused on being able to run Gamecourse's rules. For this, we created a new version of the rule parser that called Gamecourse every time one of its functions was called in a rule. This worked fine when we had only one rule and one function, but when we started adding more rules, the server socket would close after running the first function. This happened because the server was configured to close after the first client connection, which did not work. After all, we needed to create one client and connect to the server every time a new function appeared in a rule. To solve this problem, we changed the server so that it only closes after receiving a message sent by Autogame after it finished running all the rules.

The second situation where both systems have to communicate is when Gamecourse calls the rule system. This happens every time that one of Gamecourse's plugins retrieves new data from the data sources. For this, having two scripts that had to be run separately (one for the client and one for the server) was not an option. So, we created a new script that creates a Gamerules object with functions to open the socket, check if the socket is already in use, and run the rule system, using *shell_exec*. And

[3]https://thrift.apache.org/
[4]https://grpc.io/

every time that one of Gamecourse's plugin finds new data to add to the database, it will also run our system.

Finally, to make sure that the rule system does not run over itself for a course or avoid trying to open a server socket when it is already open, we created the table in Fig. 5.3. This table has a line for each active course of the system and one for the server, which is the one with course number zero. Autogame will only run for a course or create a server socket if the respective line's *isRunning* is at zero.



```
+----+--------+---------------------+-----------+
| id | course | date                | isRunning |
+----+--------+---------------------+-----------+
|  1 |      1 | 2020-12-10 16:37:27 |         0 |
|  2 |      2 | 2020-12-11 19:20:45 |         0 |
|  5 |      0 | 2020-12-11 19:20:45 |         0 |
+----+--------+---------------------+-----------+
```

Figure 5.3: Autogame table.

5.4 Changing rule parser

As we mentioned in the previous section, the system must acknowledge a Gamecourse function in the rule files. This is one of the rule parser's most important jobs, and since Gamerules did not have to deal with it, this was never implemented until now.

The rule parser works by finding keywords inside the rule files, which is how it knows how to separate the different parts of the rules (name, description, when, and then). By giving the user the possibility of using functions that belong to Gamecourse, we created a need to add a new rule piece so that the system knows how to deal with this properly. To accomplish this, we establish that all these functions had to follow the following structure:

$$GC.library\_name.function\_name(arguments)$$

Then, we updated the parser so that it was able to recognize the prefix *"GC"* and enters a loop that works as follows:

1. Saves all characters between the two dots as the library name.
2. Stores the characters between the second dot and the open parenthesis as the function name.
3. Saves everything inside the parenthesis as the arguments.
4. Creates a string that consists of: *gc("library_name","function_name", arguments)*

Finally, we created the *gc* function, which opens a client socket to communicate with Gamecourse. This way, when Autogame compiles this string created by the rule parser, it calls this function that communicates with Gamecourse. It also sends the course that the rule system is running for and all the information necessary for Gamecourse to run the function. On the server-side, it separates these arguments and runs the function. If the function's output is a list, it encodes each item and sends them separately. Then, Autogame decodes them and saves them as a *Logline* object in a list. These loglines are lines from the participation table, which means that each one refers to an action performed by a student. We created this object so that it is easier to access any participation element if necessary.

5.5 Getting system's targets

When Autogame was designed, one of its requirements was that it had to be an incremental system, meaning that it would only run new participations that occurred after the last time the system had run. This was a high priority because the old system always ran everything, which was inefficient and wasted a lot of the administrator's time.

To meet this requirement, we needed to find a strategy for the system to run as few participations as possible while computing the results correctly. Only running new participations was never an option because the system would not keep in memory the previous ones. It needs every participation to compute a correct result. It seemed like an impossible task to generate a pleasing result without running everything from the beginning of the semester, but we would not have an efficient system without this. To solve this problem, we shifted our focus from trying to run as few participants as possible to running for the least amount of targets as possible. While results may vary if we do not consider all the participations, targets are independent of each other, so running the system only for a portion of the targets would not impact the output.

The first step to put this into practice was to create a criterion for which targets the system would run each time. However, it did not make sense to risk having some students with updated awards and some students with outdated results. So we had to find a balanced solution. The solution we found was: running for every target with new participations since the last time the system ran. This means that we avoid recalculating awards for students who did not perform any action or received any grade that would change the system's last output.

To accomplish this, we added a new column to the autogame table (Fig. 5.3), which saves the last time the system ran. Every time our script is called, it checks if the system is not already running for the course; if it is not, it updates the course's line on the table with the new timestamp and sets the system as running. We chose to update the timestamp in the beginning because if we updated it at the end, it would create an interval of time between the system starting and finishing where the participations that occurred in-between would not be taken into account. This way, the system would always know the last time it ran and could easily access this information.

Finally, we created a function responsible for retrieving the targets even before we run the rule system itself. This works by using the course and the timestamp of the last time the system ran as arguments. Then, the function accesses the participation table and, using these arguments, retrieves a list of all the students that have participations posterior to the timestamp. Apart from selecting the data from the participation table, we also added a *join* clause to the query so that it also checks game_course_user and course_user tables to make sure that all the targets have the role of students. Finally, it saves these targets in a dictionary that will then be used as input for the rule system.

However, changing the targets was not enough. Because Autogame is always recalculating awards, it was necessary to know how to deal with its output. We could not just write everything without taking into account what was already in the award table because if we did, we would have multiple repeated lines. To avoid this, we made sure that the system compares its output with what was already calculated before. In the next section, we clarify in detail how and why this was done.

5.6 Grade retractions

Another requirement that we set for Autogame at the beginning of this project was that it had to know how to deal with grade retractions and other changes that may affect what the system had already calculated before. This requirement is essential because professors may need to change a grade, and this requires the system being able to retract what was previously awarded.

To deal with this, we needed to know how Gamecourse deals with these changes. If the system just added a new line to the participation table with the new grade, this might have been a problem because our system would need to make sure that it only used the most recent grade. Fortunately, Gamecourse deals with this by updating the line that was already in the database and changing its grade and timestamp. Given that the timestamp is updated, our system reruns this line and recalculates its output. However, it is necessary to change what had been previously awarded. So, how does the system do this?

First, it is important to explain that we faced this problem differently depending on what kind of rule we were dealing with. As mentioned in Section 5.2 the awards are divided into badges, skills, grades, and others. For each one, we created a function that writes the outputs in the database. This is how it works:

- **For badges**: in this case, each badge can have up to three lines on the table, depending on the level the student has reached. When the system recalculates a badge, adding or deleting more than one line may be necessary. The system checks the number of lines on the database and compares it to the level that has been calculated. If there are more lines than the new level, the system computes the difference between these two values and deletes that number of lines, starting with the highest level. If the new level computed is higher than the number of lines already in the database, the system will award the remaining lines.

- **For skills**: in this case, each student cannot have more than one line per skill. The system starts by checking if there is already something in the database regarding that skill. If there is not, it will insert the line with the corresponding grade and XP. If there is, there are two possibilities: if the new rating for this skill is lower than three, then the student no longer has this skill complete, and the line is deleted; if the rating is higher than three and different from the grade already in the database then Autogame updates the line with the new grade.

- **For grades**: in this case, the rules for awarding quiz/laboratory grades retrieve all the lines from the table that correspond to this type of participation. So, the function that awards these grades has to go through all the logs and award grades considering the quiz/laboratory number. There is a *for* loop that will go through the list of lines and either insert or update the corresponding lines. Each lab or quiz only has one corresponding line, and it is not necessary to delete lines because if a student already has something in the database, it will continue there even if the new grade is zero. For the course presentations, which are also awarded with the same function, the system works similarly, but it is unnecessary to check the number because there is only one.

- **For others**: as we mentioned in Section 5.2, this type of award is only used to award the initial bonus. This bonus is given to all the students at the beginning of the course. This is always awarded for every target and only needs to be awarded one time. It never has to be updated or deleted. So the system only checks if the line is already in the database, and if it is not, it adds it.

This method of dealing with retractions always makes sure that the database's awards are correct, but this implementation created a problem. Most of the rules related to badges had a condition to ensure that a rule was only fired if the level calculated was superior to zero (so, no awards for badges with level zero). However, there is a possibility that a student was previously awarded a badge, and after the system recalculates, this student no longer deserves it, and subsequently, the rule will not be fired. So, the badge will not be deleted from the system, which obviously cannot happen. To solve this issue, the rules were changed so that, even if the new estimated level is zero, the system will fire the rule and delete any necessary awards. If there are no lines to be erased, the system will continue to the next rule.

5.7 MCP's configuration

Autogame is an adaptable system, however, its main objective, and focus of this dissertation, is to be used in MCP. This section is intended to show how we configured the system and created the rules to fit MCP's purpose. Here, we explain:

1. The configuration of the system.
2. How we created the rules, giving some meaningful examples
3. The auxiliary functions that were necessary to implement in order to obtain the correct results.

5.7.1 Configurations

The first step to use Autogame for a course is taking care of its configurations. For that, it is necessary that the course already exists in the database. The *id* of this course is used to refer to it within the system.

First, we created a configurations file on the *config* folder, located in the root folder of Autogame, with the following name: $config\_ < course\_id >$. In this file, we created a dictionary called *METADATA* where we can put any information that needs to be accessed in the rules, such as initial bonus or number of classes for each campus. If these values ever change, we can change them here, and the rules will collect the correct values the next time the system runs. How this works is by adding this dictionary to the scope of the system. If it is necessary to add other data, we can add it to the config file and then to the scope of the rule system.

Then, we created a rule folder for the course. This folder must be created in the *rules* folder, locate on the root folder of Autogame. Also, its name must be the course id; otherwise, the system will not know how to access the folder. This is where the rule files must be stored and where the system saves the generated data files.

5.7.2 Rules

The rule files are one of Autogame's most essential inputs since they define how the system calculates its output. Although the rules' structure did not change from Gamerules to Autogame, many changes had to be made to ensure that the system could perform everything it needed to perform to generate correct results. The process of creating this rules took a lot of going back and forth, checking results, and making small changes to reach the correct output. Each rule was individually run, checked, and re-checked until its output matched the previous system's output.

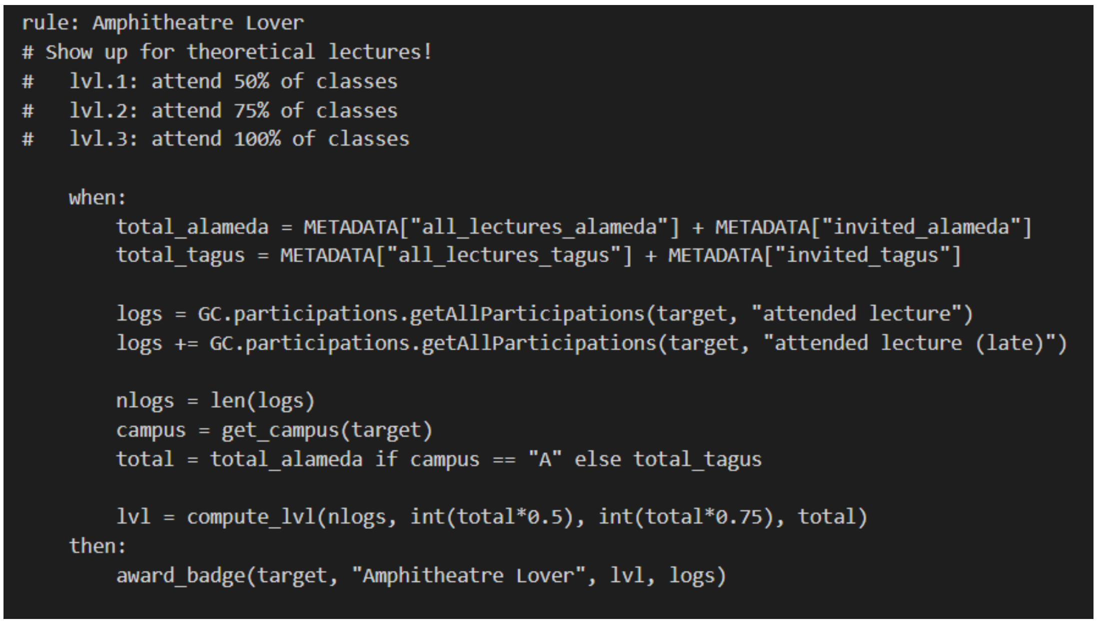
As we mentioned in Section 5.2 and later on in Section 5.6, rules can be divided in fours categories: badges, skills, grades and others. To describe the process of creating the rules, we will explain, for each category, the rule's general structure and give one significant example. The complete set of rules can be found in appendix B.

5.7.2.A Badge rules

One of the main gamification features of MCP is its set of badges. In total, we have a list of 25 rules that award badges. Although these badges are very different from each other, all these rules follow a similar structure, which consists of:

1. Getting any necessary data from the configuration file.
2. Using a Gamecourse function to retrieve the list of participations relevant for this badge.
3. Using auxiliary functions to calculate any additional data.
4. Computing the level of the badge for the target.
5. Awarding the respective level.

In Fig. 5.4 we provide the rule for *Amphitheatre Lover* badge. To get this badge, the students have to attend 50, 75, or 100 percent of the classes, depending on the level. To compute if this badge should be awarded, we start by getting the number of classes from both university campuses from the configuration file. Then, we use the expression language to get all the participations for this student that are from type ”attended lecture” and ”attended lecture (late)”. Then, we use an auxiliary function to check the student's campus. After getting the campus and checking the length of the participations list for this student, we use another function to compute the level. Finally, we award the badge, if necessary, by writing it on the database with its respective level. In the badge's case, there are no conditions so that the system can deal with grade retractions, as explained in Section 5.6.



```
rule: Amphitheatre Lover
# Show up for theoretical lectures!
#   lvl.1: attend 50% of classes
#   lvl.2: attend 75% of classes
#   lvl.3: attend 100% of classes

    when:
        total_alameda = METADATA["all_lectures_alameda"] + METADATA["invited_alameda"]
        total_tagus = METADATA["all_lectures_tagus"] + METADATA["invited_tagus"]

        logs = GC.participations.getAllParticipations(target, "attended lecture")
        logs += GC.participations.getAllParticipations(target, "attended lecture (late)")

        nlogs = len(logs)
        campus = get_campus(target)
        total = total_alameda if campus == "A" else total_tagus

        lvl = compute_lvl(nlogs, int(total*0.5), int(total*0.75), total)
    then:
        award_badge(target, "Amphitheatre Lover", lvl, logs)
```

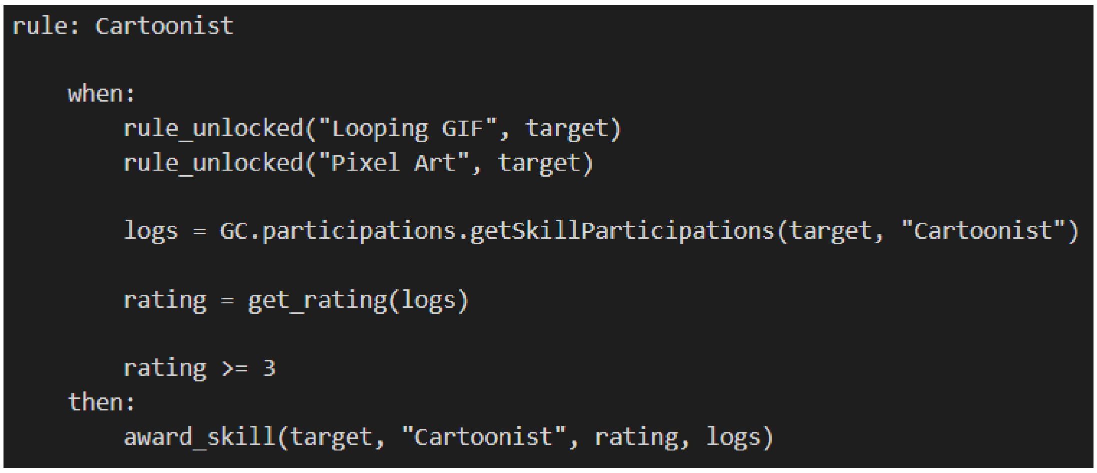
Figure 5.4: Amphitheatre Lover badge rule.

5.7.2.B Skill rules

The second category of rules that we created is the skill rules. The skill rules refer to a skill in the skill tree of the course. Each skill has a tier, and skills in tier 2 or above need two pre-requisite skills from the tier below to be completed. To check if the precedent rules have been completed, we use a function that returns true if a rule has been fired. If a skill rule has been fired, it means that the skill has been completed. So, this kind of rules have the following structure:

1. Checking if this skill is unlocked.
2. Using a Gamecourse function to retrieve the list of participations for this skill.
3. Using an auxiliary function to get the rating for this skill.
4. Creating condition so that the rule only fires if the rating is equal or greater than three (the minimum rate to complete a skill).
5. Awarding the respective skill.

Fig. 5.5 represents the rule for the Cartoonist skill. First, the system checks if the two precedent skills have been completed. If they have been, it continues, if not, the conditions have not been met, so it continues to the next rule. Then, it retrieves the related participations relative to this skill. There should be only one participation because if a teacher decides to change a grade, the line for this skill should be updated, and there will never be more than one line. However, if for some reason there are more than one, the next function, which is the one that checks the skill rating, will return the rating for the line with the more recent timestamp. Then, we use a condition so that the rule only fires if the rating is equal to or greater than three. This condition exists because the skill is only completed if it has this minimum rating. And, if the rule fired even though the student did not have this minimum rating, the function *rule_unlocked* would return an incorrect output. Finally, if the conditions are met, the system writes the new award in the database.



```
rule: Cartoonist

    when:
        rule_unlocked("Looping GIF", target)
        rule_unlocked("Pixel Art", target)

        logs = GC.participations.getSkillParticipations(target, "Cartoonist")

        rating = get_rating(logs)

        rating >= 3
    then:
        award_skill(target, "Cartoonist", rating, logs)
```

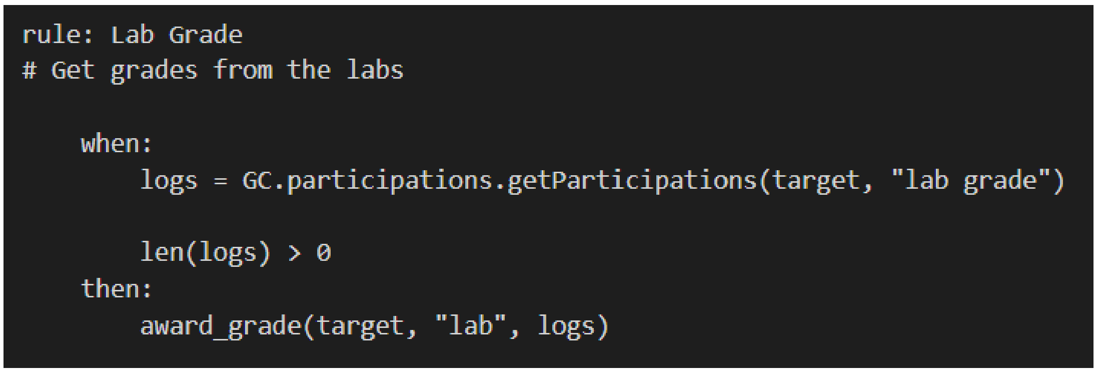
Figure 5.5: Cartoonist skill rule.

All the rules for this category have the following filename structure: $Skill < skill\_tier >< skill\_name > .txt$. The reason behind this is because the system runs the rules by filename, and we need it to run the rules from the lower tiers first. If this was not the case and the system ran a rule from a higher tier that had precedent skills that were not yet fired, then there was no way to know if the skill had been unlocked or not.

5.7.2.C Grade rules

For this category, it is only necessary to retrieve the participations from Gamecourse, create a condition, and award the grade.

In Fig. 5.6 we provide the rule for awarding laboratory grades. For this rule, we begin by retrieving the participations that have ”lab grade” type. There can be more than one participation in this case, one for each laboratory. Then we use a condition so that the rule only fires if the list of participations has a length greater than zero. Finally, we award the lab grade for each of the participations retrieved. If there is more than one participation for a specific laboratory, the system will only award the more recent grade.

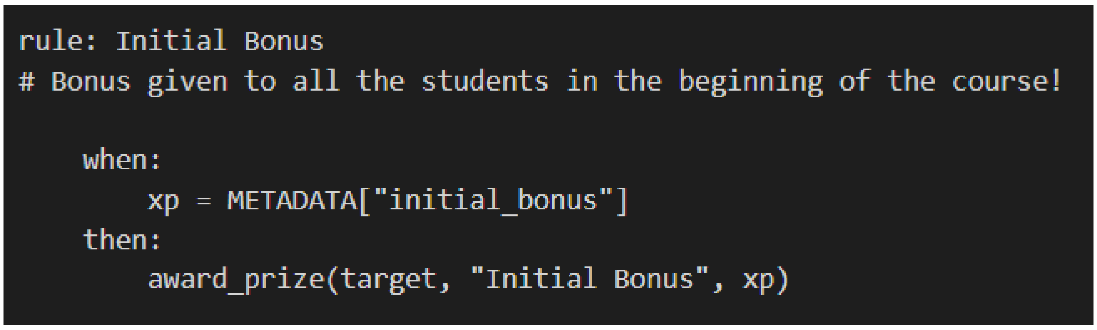


```
rule: Lab Grade
# Get grades from the labs

    when:
        logs = GC.participations.getParticipations(target, "lab grade")

        len(logs) > 0
    then:
        award_grade(target, "lab", logs)
```

Figure 5.6: Laboratory grade rule.

5.7.2.D Other rules

In MCP's case this category includes only one rule, which awards the initial bonus. This kind of rule is the simplest one. As shown in Fig 5.7, it only needs to perform two steps. First, it retrieves the initial bonus from the configuration file, and then, it awards that amount of XP for each student.



```
rule: Initial Bonus
# Bonus given to all the students in the beginning of the course!

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

Figure 5.7: Initial Bonus rule.

5.7.3 Auxiliary functions

Although Gamerules already had a set of functions created to be used in the rules files, Autogame works differently, so the functions that existed did not fulfill our needs. Gamecourse's expression language made our task a lot easier because it provided a set of functions that allowed us to retrieve filtered information from the database. However, we still lacked some code to help us perform additional actions.

The functions used to write the awards in the database (described in Section 5.6) are examples of functions added to Autogame to ensure that the rules work as intended. Apart from these, as we were creating the rules, we found new challenges that could not be solved with the existing functions, so we began developing new ones. All the functions that were created with the intent of being used within the rules are described below and in Table 5.1.

- **get_campus**: it receives a target as input and, by connecting with the database, checks its campus and returns it. It can either return an ”A” for Alameda or a ”T” for Taguspark.

- **compute_lvl**: it is used on the badge rules. It receives from two to four values, depending on how many levels the badge has. The first value is the number of participations the target has, and the next ones are the thresholds needed to achieve each of the levels. It computes the level to which the first value belongs and returns an integer that is the computed level.

- **get_rating**: it is used in the skill rules. It receives a list of one or more Logline objects. If the length of the list is zero, it returns a rating of zero; if it is one, it returns the rating of that participation; and if it is greater than one, it returns the rating for the most recent participation.

- **compute_rating**: this function is used for badges that need to add multiple participations' ratings. Its argument is a list of Logline objects. It goes through every participation on the list and adds the respective ratings, returning the total value.

- **filter_quiz**: this function was created specifically for the *Quiz Master* badge. In MCP the worst grade from the quizzes does not count. This is done by ignoring the last quiz grade and creating a new one which is the difference between the last one and the worst one. This difference is then added to the database. When we retrieve the database's participations, this participation comes in the list too. The problem is that we do not need and do not want this participation for this badge. So, we created this function specifically for erasing this object from the list, if it exists. It can also be modified to erase any other participation from any other list if necessary.

- **filter_excellence**: this is another function that was specifically developed for one badge, in this case, *Lab Master* badge. For this badge, it is necessary to check how many of the participations have the maximum laboratory grade. The key here is that the maximum grade is not the same for all the classes. From now on, we will call tier each group of laboratory classes that has the same maximum grade. For example, in MCP's case, the first five laboratory classes have a maximum grade of 150 XP and the last five a maximum of 400 XP. This means that there are two tiers in this course, each with 5 classes. So, this function receives a list of participations, a list of maximum grades (for MCP: [150, 400]), and the number of classes in each tier (for MCP: [5, 5]). It will then go through each participation and check if it has the maximum grade for the tier it belongs to. If it does, it will be added to a new list, which will then be the output of this function.

Table 5.1: Autogame's auxilar functions for MCP.

Name	args	description	output
get_campus	target (int)	Returns targets's campus	campus (str)
compute_lvl	val (int), *lvls (int)	Returns level in which "val" is inserted	level (int)
get_rating	logs (list)	Returns rating for the most recent log	rating (int)
compute_rating	logs (list)	Computes the sum of the ratings of the logs	total_rating (int)
filter_quiz	logs (list)	Erases element of list referent to "quizz 9"	final_logs (list)
filter_excellence	logs (list), tiers (list), classes (list)	Returns only the logs with maximum grade for the corresponding laboratory class	final_logs (list)

6 Evaluation

Contents

To ensure the quality of Autogame, we conducted a thorough evaluation to cover all the aspects of the system that might not have been working as intended. This evaluation process was composed of two phases: correctness tests and performance tests. The first phase consisted of running a set of tests to evaluate the system's output and its expected results. The second one intended to evaluate the system's performance by creating extreme scenarios and checking the time and memory the system needed to run. This chapter shows how we conducted this evaluation, its results, and a discussion on what we were expecting versus what we got.

6.1 Correctness tests

The first phase of Autogame's evaluation was a correctness tests. These correctness tests had the purpose of finding out if the system was creating a correct output. We defined as a correct output a text file generated by the previous system, which consisted of a list of all the awards given to MCP's 2019/2020 students. However, our system did not create such a file, so to compare both outputs, the first step was to add code to Autogame to create this file with the same structure as the one produced by the previously used script. To do that, we made it so that it wrote the awards to the file every time the system was supposed to write on the database. The goal was to create a file that compiled all the semester's awards, so at this point, we did not worry about having to write or delete awards. This file was created with all the participations on the database so, the system did not need to perform deletes or updates in order to produce a correct result.

After changing our system to create this file, we proceeded to compare it to the same file from the previous system. To do this, we created a function that read through both the files and saved their information on a dictionary. This dictionary had a key with every rule name, and each of the rule names had a new dictionary where the students were the keys. Finally, for each student and rule name, we created a list of awards. With this structure created for both system, we started the comparison process.

Once we had both files, we needed to compare the results. We decided to create a set of tests using python's package unittest [1] which allowed us to create personalized tests to assess the correctness of our output. This evaluation consisted of a set of tests, each for a specific rule, that compared the number of students who received that award and then if the specific awards were correct (levels, grades, number of quiz/laboratory). For this tests we used the students from MCP which are 99, and the rules created for the course, which are 49. In table 6.1 we provide the results of the tests that failed when comparing the number of students that received a specific award.

Apart from the rules described in the table above, we had some other issues with rules that had the correct number of students, however, some of the award's attributes were incorrect. One of the reasons

[1] `https://docs.python.org/3.0/library/unittest.html`

Table 6.1: Badges that were awarded for a different number of students for both systems.

	Students	
Award	**Previous system**	**Autogame**
Amphitheater Lover Badge	86	87
Right on Time Badge	86	87
Tree Climber Badge	62	60
Lab Master Badge	86	86

this happened was that the data our system uses is not 100% equal to the data used to generate the awards file last year. This was because some of the data was lost and had to be added manually. Also, and some of the previous system's data was not updated when the file was created. Here is an explanation of all the awards that had incorrect outputs and why this happened:

- **Amphitheatre Lover and Right on Time**: both these badges use almost the same participations. Somehow, when creating the database, we added participations for a student incorrectly. Upon running the tests, we compared our results to the previous ones and what was in the leaderboard from last year, and we took these participations from our database. The test was then run again with the new data, and everything matched the correct results.

- **Tree Climber**: in this case, we discovered that the previous system was incorrectly awarding this badge. This badge is awarded when a student completes skills for tiers 2, 3, and 4 of the skill tree, each tier corresponding to a badge level. However, last year's file awarded this badge even if the student did not have the minimum rating to have the skill completed. A student would have level 3 of this badge if he/she tried a skill from tier 4, even if the rating was a zero. This was not supposed to happen, so we kept our results as the correct ones. The difference between the number of students who received this award in both systems is due to this miss calculation.

- **Lab grades**: in this case, the difference between files was that some grades were different. Our system was awarding the grades taken from google sheets. However, the other file did not have the correct grades for some students. This was most likely because the teacher corrected these grades, and then the participations file for the script used last year was not updated.

- **Lab Master**: in this case, we had some students that had more levels on this badge than what was supposed to happen. This was due to the fact that we were using the wrong thresholds in our rules, so it was easier to get the maximum level. Upon correcting this, the system passed the test.

Finally, upon making the necessary adjustments to the system's data or the previous file's incorrect awards, our system passed every test. The only one that had a different output, in the end, was the rule for the Tree Climber badge, which, as we explained above, was not being correctly calculated.

6.2 Performance tests

Good performance is a critical feature in any information system. Ensuring the system can handle a lot of users and rules was one of our main priorities when developing Autogame. To put the system to the test, we created some extreme scenarios and checked how long the system took to run and how much memory is used. In the next sub-sections, we describe the process of creating these tests, their results, and a commentary on them.

6.2.1 Increasing system's targets

To evaluate the system, we wanted to test how it would behave in terms of memory usage peak and running time for different numbers of targets. For this, as for the correctness tests, we created a set of tests using python's unittest library.

The first step was to create a new course used only for testing and adding a lot of new students, more specifically 1000 students. To create these students, we first found a list of random names online, then chose a student with many participations and used her as an example. Then, using a python script that we developed, we added the new students to the database and generated the same participations from the MCP student for each new one. This way, we created 1000 new targets, with the same interaction with the course as the person that had the most interaction last year.

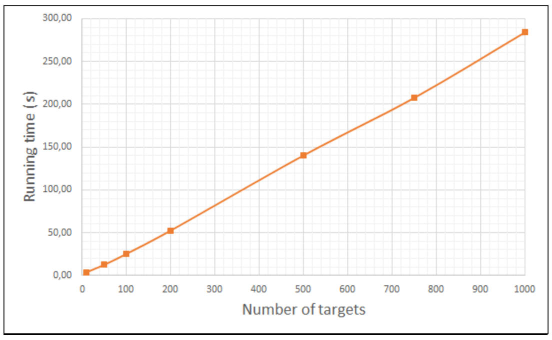
Then, we changed the function that retrieves targets from the database to have a new argument used as the maximum number of students. Inside this function, we added a limit to how many students the query could retrieve. By doing this, we made it easier to run a rule system for different numbers of targets in the tests.

Finally, we created the seven different tests that would run Autogame for different numbers of students, using all of MCP's 49 rules. Using python's libraries time [2] and resource [3] we managed to calculate the running time, in seconds, and memory usage peak, in Mebibytes, for each test. In table 6.2 we provide the results of our evaluation. We ran the tests three times and averaged the results to ensure that they are as accurate as possible and avoid having external factors altering our results. Also, in Fig. 6.1 we show the evolution of the system's load with the increase of the number of targets.

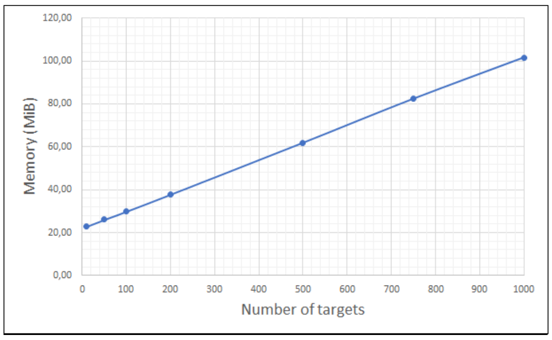
[2] https://docs.python.org/3/library/time.html
[3] https://docs.python.org/3/library/resource.html

Table 6.2: System's running time (s) and memory usage peak (MiB) when increasing the number of targets, with 49 rules.

	1		2		3		Average	
Targets	**Time**	**Memory**	**Time**	**Memory**	**Time**	**Memory**	**Time**	**Memory**
10	2,96	22,96	2,36	22,88	3,91	22,75	3,55	22,87
50	11,44	26,14	11,02	25,96	13,93	26,01	12,32	26,03
100	22,33	29,93	28,79	29,91	23,22	29,81	25,03	29,88
200	52,67	37,91	59,77	37,77	44,34	37,69	52,34	37,79
500	146,48	61,96	151,69	62,15	116,42	61,81	140,15	61,98
750	225,29	82,61	205,29	82,78	189,29	81,94	207,43	82,44
1000	289,68	101,66	281,57	101,85	282,49	101,20	283,89	101,57



(a) Running time (s).



(b) Memory usage peak (MiB).

Figure 6.1: System's characteristics when increasing the number of targets, with 49 rules.

6.2.2 Increasing system's rules

Another aspect of the system that we wanted to test was its ability to deal with a large quantity of rules. This may be important in the future for MCP or even other courses that starts using our system. To test how the system deals with a larger number of rules, we conducted a similar experiment as in the previous sub-section.

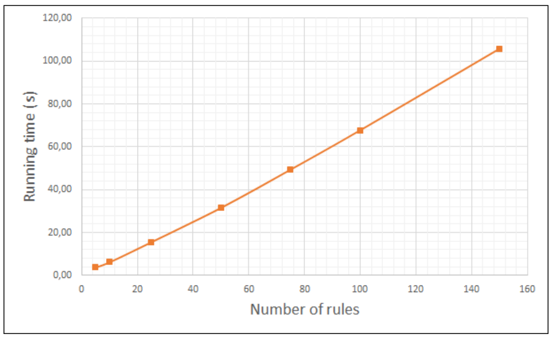
For this case, we needed many rules, more specifically, 150, which is three times the number of tules in MCP. To ensure that the rules' complexity was not a factor here, we used the same rule repeatedly but with a different name every time. We chose one with medium complexity to create these rules and developed a script that created equal text files with two differences: the name of the file and the rule's name inside the file. This way, the system was running the same rule every time but was dealing with it as if it were a different one, so it always inserted new awards instead of updating the previous ones.

After having the rules, we started creating the tests using the same method as in the previous experience. However, in this case, we made the number of targets a constant, 100 students, which is the number of students in MCP. Each test created run the rule system with a different rule folder. Each folder had a different number of rules inside, from 5 to 150. Then, using the same libraries as before, it calculated the running time and memory peak usage.

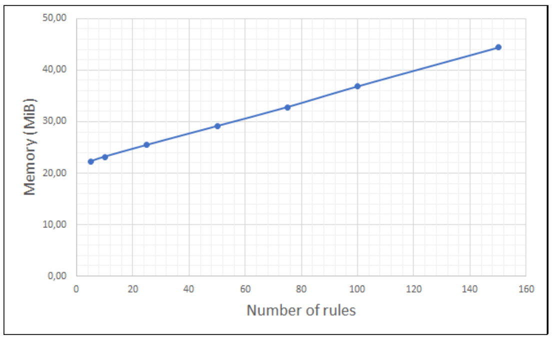
In table 6.3 we provide the results of the three times that we run the rules and the average results. In Fig. 6.2 present the average values to show the system's evolution when we increased the number of rules.

Table 6.3: System's running time (s) and memory usage peak (MiB) when increasing the number of rules, with 100 targets.

	1		**2**		**3**		**Average**	
Targets	**Time**	**Memory**	**Time**	**Memory**	**Time**	**Memory**	**Time**	**Memory**
5	3,30	22,38	4,27	22,23	2,84	22,21	3,73	22,27
10	5,76	23,07	5,40	23,31	5,56	23,13	6,20	23,17
25	14,09	25,44	15,71	25,50	14,05	25,43	15,56	25,46
50	27,13	29,23	32,90	29,15	29,59	29,00	31,57	29,13
75	44,12	32,87	53,95	32,82	45,16	32,74	49,42	32,81
100	65,11	36,77	71,92	36,88	62,33	36,84	67,78	36,83
150	113,17	44,39	96,34	44,45	97,59	44,32	105,73	44,38



(a) Running time (s).



(b) Memory usage peak (MiB).

Figure 6.2: System's characteristics when increasing the number of rules, with 100 targets.

6.2.3 Discussion

Autogame is a rule system that runs every time new data is added to Gamecourse's database. In our system, performance assessment is vital because this system runs many times a day, and it needs to be able to run as fast as possible and use the least amount of the computer's resources as possible. In both Fig. 6.2 and Fig. 6.1 we can conclude that, by taking the system to an extreme scenario, both in terms of number of targets and in number of rules, the system maintains a linear evolution. This gives us an idea of the system's load when using a certain amount of targets and rules.

In MCP's case, we had 99 students and 49 rules, which means that the system will take around 25 seconds to calculate all the awards for the semester. However, it is essential to remember that this is an incremental system and will never need to calculate all the course awards simultaneously. The system only runs when new participations are added to the database and most likely will have a low periodicity. This means that it will only run for students with new participations since the last time the system ran. So, for MCP the running time during the semester will more likely be between 3 and 15 seconds. As for the memory, for the system to run all of MCP's awards for the semester it takes the system around 30 MiB. However, as discussed for the time, the system normally will not run for all 100 targets, it will run for around 5 to 25 targets depending on the periodicity. This means that the system's memory used will be somewhere between 22 and 25 MiB. This memory usage is not at all high, which means that any computer can easily run Autogame without it taking too much of the system's memory.

7 Conclusion

Contents

For this dissertation, we developed a scalable and automated rule system for a gamification course. Most of this work was put into integrating this system in Gamecourse, the system used for MCP. However, this can be adapted to other courses and similar systems.

To develop our system, we initially had a longstanding process of understanding and update Gamerules, the previous of the rule system that was never finished. Gamerules already had implemented some of the rule system's main features. However, it needed much work in order to be usable in the context that we wanted it to work in, MCP. Some of the first steps we had to take consisted of just trying to understand a system that had zero documentation and changing some minor details to make it work as it was.

Upon understating the basics of Gamerules, we started the process of creating Autogame and integrating it with Gamecourse. This was a significant part of our work, from creating access to the database, making the communication between PHP and Python, and going through every scenario possible to ensure that the rules could perform every task necessary. It required adapting the system over and over again until we found the solution that best fitted our needs.

The evaluation showed that Autogame was generating the correct results, which was vital for our work. The performance tests showed that it was capable of running in a matter of seconds and without overusing the system's memory.

Overall, the system in its current state can hopefully be used in MCP in the next semester. It will provide an automatic way for the system to calculate the course's awards and will make the task of keeping the gamification experience up and running that much easier.

7.1 Future work

Although the system is working fine, and without any issues, it can always be improved. One of the areas where the system is lacking is in its interaction with the user. Creating text files for each rule and adding configuration files is not the most user-friendly way of using a rule system. Therefore, the next step can be creating a User Interface (UI) for the system, where the user will have the option of creating rules on the browser. This can also make the creation of rules easier by giving the users a list of existing functions. Actually, there is already another MSc student, Ana Nogueira, working on developing an interface for Autogame in Gamecourse. It will certainly make the system more pleasing to use and add new and better features.

Another thing that can always be improved is the system's libraries and functions. This would open the door to more possibilities within the rules and make them more flexible. Some of the system's functions were specifically created to fit a MCP rule and will not probably be useful for any other rules in the future.

Bibliography

[1] D. Lopes, “Gamecourse beyond,” Master’s thesis, Instituto Superior Técnico, Universidade de Lisboa, 2021.

[2] M. Nascimento, “I am here!” Master’s thesis, Instituto Superior Técnico, Universidade de Lisboa, 2019.

[3] A. Baltazar, “Smartboards,” Master’s thesis, Instituto Superior Técnico, Universidade de Lisboa, 2016.

[4] M. A. Ghahazi, M. H. F. Zarandi, M. H. Harirchian, and S. R. Damirchi-Darasi, “Fuzzy rule based expert system for diagnosis of multiple sclerosis,” *IEEE Conference on Norbert Wiener in the 21st Century (21CW)*, 2014.

[5] A. R. C. Semogan, B. D. Gerardo, B. T. T. III, J. T. d. Castro, and L. F. Cervantes, “A rule-based fuzzy diagnostics decision support system for tuberculosis,” *Ninth International Conference on Software Engineering Research, Management and Applications*, 2011.

[6] W. B. Guo, X. P. Hu, and J. Liu, “Rule-based reasoning in onboard devices: An intelligent route guidance system,” *IEEE International Conference on Service Operations and Logistics, and Informatics*, 2006.

[7] Y. Shanliang, F. Yuewen, Z. Peng, and H. Kedi, “Implementation of a rule-based expert system for application of weapon system of systems,” *International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, 2013.

[8] M. Stanojevic and S. Vranes, “A natural language processing for semantic web services,” *EUROCON 2005 - The International Conference on ”Computer as a Tool”*, 2005.

[9] H. Isahara, “Resource-based natural language processing,” *International Conference on Natural Language Processing and Knowledge Engineering*, 2007.

[10] H. S., O. M.J., and D. A.K., “A framework for the automatic extraction of rules from online text,” 2011.

[11] C. D'Este, D. Reid, and B. H. Kang, "A robotic interface to a medication review expert system," *International Symposium on Ubiquitous Multimedia Computing*, 2008.

[12] D. Hooshyar, R. B. Ahmad, M. H. N. M. Nasir, and W. C. Mu, "Flowchart-based approach to aid novice programmers: A novel framework," *International Conference on Computer and Information Sciences (ICCOINS)*, 2014.

[13] M. Mosconi and M. Porta, "A data-flow visual approach to symbolic computing:implementing a production-rule-based programming system through a general-purpose data-flow vl," *Proceeding 2000 IEEE International Symposium on Visual Languages*, 2000.

[14] J. Poli and J. Laurent, "Touch interface for guided authoring of expert systems rules," *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2016.

[15] A. Kulpa, J. Swacha, and K. Muszynska, "Visual rule editor for e-guide gamification web platform," *Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2019.

[16] T. Tuomisto, T. Kymäläinen, J. Plomp, A. Haapasalo, and K. Hakala, "Simple rule editor for the internet of things," *International Conference on Intelligent Environments*, 2014.

[17] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, "So fun it hurts – gamifying an engineering course," *International Conference on Augmented Cognition*, 2011.

[18] B. S. Akpolat and W. Slany, "Enhancing software engineering student team engagement in a high-intensity extreme programming course using gamification," *IEEE 27th Conference on Software Engineering Education and Training (CSEE&T)*, 2014.

[19] R. D. Michele and M. Furini, "Tv commercials: Improving viewers engagement through gamification and second screen," *IEEE Symposium on Computers and Communications (ISCC)*, 2017.

[20] A. L. Brazil and E. Clu, "A virtual environment for breast exams practice with haptics and gamification," *IEEE 5th International Conference on Serious Games and Applications for Health (SeGAH)*, 2017.

[21] A. Tóth and S. Tóvölgyi, "The introduction of gamification: A review paper about the applied gamification in the smartphone applications," *7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*, 2016.

[22] G. Barata, S. Gama, J. Jorge, , and D. Gonçalves, "Engaging engineering students with gamification," *Proceedings of the fifth outing of the International Conference on Games and Virtual Worlds for Serious Applications*, 2013.

[23] J. Amaral, “Gamecourse,” Master’s thesis, Instituto Superior Técnico, Universidade de Lisboa, 2013.

[24] A. Dourado, “Gamecoursenext,” Master’s thesis, Instituto Superior Técnico, Universidade de Lisboa, 2019.

[25] P. Silva, “Gamecourseui,” Master’s thesis, Instituto Superior Técnico, Universidade de Lisboa, 2021.

A

Gamecourse's entity-relationship model

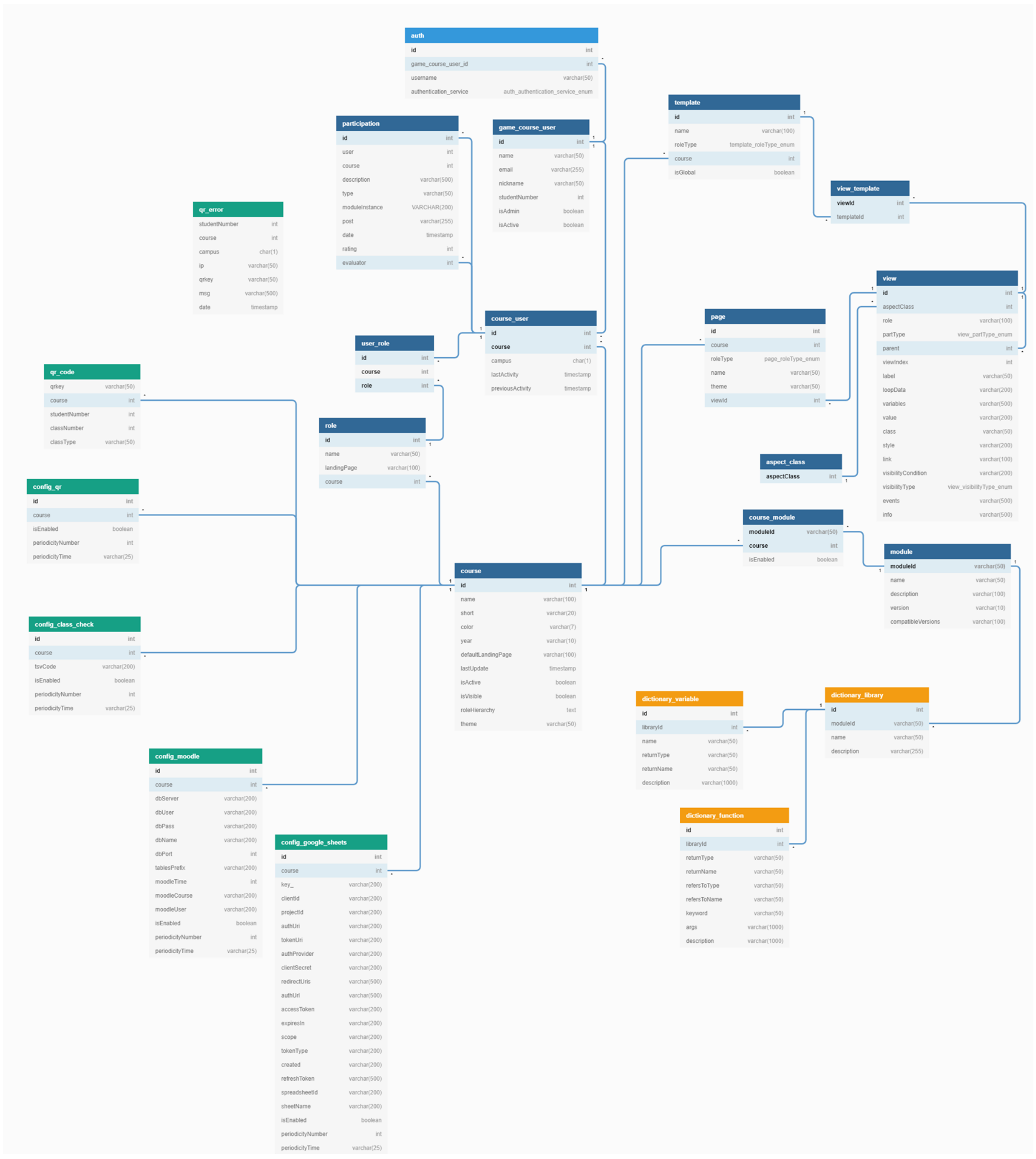


Figure A.1: Gamecourse's Entity-Relationship Model.

B

MCP’s rules

Listing B.1: Badge rules

```
rule: Amphitheatre Lover
# Show up for theoretical lectures!
#  lvl.1: attend 50\% of classes
#  lvl.2: attend 75\% of classes
#  lvl.3: attend 100\% of classes
 when:
  total_alameda = METADATA["all_lectures_alameda"] + METADATA["
      invited_alameda"]
  total_tagus = METADATA["all_lectures_tagus"] + METADATA["invited_tagus"]
  logs = GC.participations.getAllParticipations(target, "attended lecture")
  logs += GC.participations.getAllParticipations(target, "attended lecture (
      late)")
  nlogs = len(logs)
  campus = get_campus(target)
  total = total_alameda if campus == "A" else total_tagus
  lvl = compute_lvl(nlogs, int(total*0.5), int(total*0.75), total)
 then:
  award_badge(target, "Amphitheatre Lover", lvl, logs)
#############################################################################

rule: Apprentice
# Give answers in the 'questions' or 'Labs forums'
# lvl.1: get four points
# lvl.2: get eight points
# lvl.3: get twelve points
 when:
  logs = GC.participations.getForumParticipations(target, "Questions")
  logs += GC.participations.getForumParticipations(target, "Labs")
  points = compute_rating(logs)
  lvl = compute_lvl(points,4,8,12)
 then:
  award_badge(target, "Apprentice", lvl, logs)
#############################################################################

rule: Artist
# Show creativity and quality:
#  lvl.1: get four posts of four points (or higher)
#  lvl.2: get six posts of four points (or higher)
#  lvl.3: get twelve posts of four points (or higher)
 when:
  logs = GC.participations.getParticipations(target, "graded post", 4)
  logs += GC.participations.getParticipations(target, "graded post", 5)
  nlogs = len(logs)
  lvl = compute_lvl(nlogs,4,6,12)
 then:
  award_badge(target, "Artist", lvl, logs)
#############################################################################

rule: Attentive Student
# Find relevant bugs in class materials
# lvl.1: get four points
# lvl.2: get eight points
# lvl.3: get twelve points
 when:
  logs = GC.participations.getForumParticipations(target, "Bugs")
  points = compute_rating(logs)
  lvl = compute_lvl(points,4,8,12)
 then:
  award_badge(target, "Attentive Student",lvl, logs)
#############################################################################

rule: Book Master
# Read class slides
# lvl.1: read slides for 50% of lectures
# lvl.2: read slides for 75% of lectures
# lvl.3: read all lectures slides
 when:
  total = METADATA["all_lectures"]
  logs = GC.participations.getResourceViews(target)
  nlogs = len(logs)
  lvl = compute_lvl(nlogs, total*0.5, total*0.75, total)
 then:
  award_badge(target, "Book Master", lvl, logs)
#############################################################################
```

```
rule: Class Annotator
# Find related resources, more information, about class subjects
# lvl.1: get four points
# lvl.2: get eight points
# lvl.3: get twelve points
 when:
  logs = GC.participations.getForumParticipations(target, "Participation", "
     Class Annotator")
        points = compute_rating(logs)
  lvl = compute_lvl(points,4,8,12)
 then:
  award_badge(target, "Class Annotator",lvl, logs)
#############################################################################

rule: Course Emperor
# Take the course, be the best
# lvl.1: Have the highest course grade!
 when:
  logs = GC.participations.getAllParticipations(target, "course emperor")
  nlogs = len(logs)
  lvl = compute_lvl(nlogs, 1)
 then:
  award_badge(target, "Course Emperor", lvl, logs)
#############################################################################

rule: Focused
# Participate in the Focus Group Interviews
# lvl.1: participate in the interviews
 when:
  logs = GC.participations.getAllParticipations(target, "participated in
     focus groups")
  nlogs = len(logs)
  lvl = compute_lvl(nlogs, 1)
 then:
  award_badge(target, "Focused", lvl, logs)
#############################################################################

rule: Golden Star
```

```
# Be creative and do relevant things to help improve the course
# lvl.1: perform one task
# lvl.2: perform two tasks
# lvl.3: perform three tasks
 when:
  logs = GC.participations.getParticipations(target, "golden star award")
  nlogs = len(logs)
  lvl = compute_lvl(nlogs, 1, 2, 3)
 then:
  award_badge(target, "Golden Star", lvl, logs)
#############################################################################

rule: Hall of Fame
# Get into the Hall of Fame
# lvl.1: one entry
# lvl.2: two entries
# lvl.3: three entries
 when:
  logs = GC.participations.getAllParticipations(target, "hall of fame")
  nlogs = len(logs)
  lvl = compute_lvl(nlogs, 1, 2, 3)
 then:
  award_badge(target, "Hall of Fame", lvl, logs)
#############################################################################

rule: Lab King
# Attend the labs, be the best
# lvl.1: Have the highest grade in the labs
 when:
  logs = GC.participations.getAllParticipations(target, "lab king")
  nlogs = len(logs)
  lvl = compute_lvl(nlogs, 1)
 then:
  award_badge(target, "Lab King", lvl, logs)
#############################################################################

rule: Lab Lover
# Show up for labs!
#  lvl.1: attend 50% of classes
#  lvl.2: attend 75% of classes
#  lvl.3: attend 100% of classes
 when:
  logs = GC.participations.getAllParticipations(target, "attended lab")
  nlogs = len(logs)
  total = METADATA["all_labs"]
  lvl = compute_lvl(nlogs, total*0.5, total*0.75, total)
 then:
  award_badge(target, "Lab Lover", lvl, logs)
#############################################################################

rule: Lab Master
# Excel at the labs
#  lvl.1: top grade in four graded classes
#  lvl.2: top grade in six graded classes
#  lvl.3: top grade in all graded classes
 when:
  tier_1 = METADATA["lab_excellence_threshold_1"]
  tier_2 = METADATA["lab_excellence_threshold_2"]
  logs = GC.participations.getParticipations(target, "lab grade")
  flogs = filter_excellence(logs,[tier_1, tier_2],[5,5])
  nlogs = len(flogs)
  labs = METADATA["all_labs"] - 1
  lvl = compute_lvl(nlogs, 4, 6, labs)
 then:
  award_badge(target, "Lab Master", lvl, flogs)
#############################################################################

rule: Popular Choice Award
# Have the most liked multimedia presentation
# lvl.1: be the third most liked
# lvl.2: be the second most liked
# lvl.3: be the most liked!
 when:
  lvl = GC.participations.getRankings(target, "popular choice award (
      presentation)")
 then:
  award_badge(target, "Popular Choice Award", lvl)
#############################################################################

rule: Post Master
# Post something in the forums
# lvl.1: make twenty posts
# lvl.2: make thirty posts
# lvl.3: make fifty posts
 when:
  logs = GC.participations.getAllParticipations(target, "forum upload post")
  nlogs = len(logs)
  lvl = compute_lvl(nlogs, 20, 30, 50)
 then:
  award_badge(target, "Post Master", lvl, logs)
#############################################################################

rule: Presentation King
# 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)
  lvl = compute_lvl(nlogs, 1)
 then:
  award_badge(target, "Presentation King", lvl, logs)
#############################################################################

rule: Quiz King
# Take the quizzes, be the best
# lvl.1: Have the highest grade in the quizzes!
 when:
  logs = GC.participations.getAllParticipations(target, "quiz king")
  nlogs = len(logs)
  lvl = compute_lvl(nlogs, 1)
 then:
  award_badge(target, "Quiz King", lvl, logs)
#############################################################################
```

```
rule: Quiz Master
# Excel at the quizzes
# lvl.1: top grade in four quizzes
# lvl.2: top grade in six quizzes
# lvl.3: top grade in eight quizzes
 when:
  max = METADATA['quiz_max_grade']
  logs = GC.participations.getParticipations(target, "quiz grade", max)
  final_logs = filter_quiz(logs)
  nlogs = len(final_logs)
  lvl = compute_lvl(nlogs, 4, 6, 8)
 then:
  award_badge(target, "Quiz Master", lvl, final_logs)
#############################################################################

rule: Replier Extraordinaire
# Respond to the gamification questionnaires
# lvl.1: respond to first questionnaire
# lvl.2: respond to both the first questionnaire and the weekly
    questionnaires
# lvl.3: respond to all the questionnaires
 when:
  logs = GC.participations.getAllParticipations(target, "replied to
      questionnaires")
  lvl = len(logs)
  lvl > 0
 then:
  award_badge(target, "Replier Extraordinaire", lvl, logs)
#############################################################################

rule: Right on Time
# Don't be late for class!
# lvl.1: be on time for 50% of lectures
# lvl.2: be on time for 75% of lectures
# lvl.3: always be there on time
 when:
  total_alameda = METADATA["all_lectures_alameda"] + METADATA["
      invited_alameda"]
  total_tagus = METADATA["all_lectures_tagus"] + METADATA["invited_tagus"]
  campus = get_campus(target)
  total = total_alameda if campus == "A" else total_tagus
  logs = GC.participations.getAllParticipations(target, "attended lecture")
  nlogs = len(logs)
  total = total_alameda if campus == "A" else total_tagus
  lvl = compute_lvl(nlogs, int(total*0.5), int(total*0.75), total)
 then:
  award_badge(target, "Right on Time", lvl, logs)
#############################################################################

rule: Squire
# Help your colleagues by writing tutorials of your tree challenges
# lvl.1: get four points
# lvl.2: get ten points
# lvl.3: get sixteen points
 when:
  logs = GC.participations.getForumParticipations(target, "Participation", "
      Tutorials")
  points = compute_rating(logs)
  lvl = compute_lvl(points,4,10,16)
 then:
  award_badge(target, "Squire", lvl, logs)
#############################################################################

rule: Suggestive
# Help your colleagues by writing tutorials of your tree challenges
# lvl.1: get four points
# lvl.2: get eight points
# lvl.3: get twelve points
 when:
  logs = GC.participations.getForumParticipations(target, "Participation", "
      Suggestions")
  points = compute_rating(logs)
  lvl = compute_lvl(points,4,8,12)
 then:
  award_badge(target, "Suggestive", lvl, logs)
#############################################################################
```

```
rule: Talkative
# Participate in Theoretical Lectures!
# lvl.1: participate 2 times
# lvl.2: participate 6 times
# lvl.3: participate 12 times
 when:
  logs = GC.participations.getAllParticipations(target, "participated in
      lecture")
  logs += GC.participations.getAllParticipations(target, "participated in
      invited lecture")
  nlogs = len(logs)
  lvl = compute_lvl(nlogs,2,6,12)
 then:
  award_badge(target, "Talkative", lvl, logs)
##############################################################################

rule: Tree Climber
# Reach higher levels of the skill tree
# lvl.1: reach level two
# lvl.2: reach level three
# lvl.3: reach level four
    when:
        lvl_3 = rule_unlocked("Director", target) or rule_unlocked("Series
            Intro", target)
        lvl_2_a = rule_unlocked("Cartoonist", target) or rule_unlocked("Fake
            Speech", target)
        lvl_2_b = rule_unlocked("Foley", target) or rule_unlocked("Kinetic",
            target)
        lvl_2_c = rule_unlocked("Stop Motion", target) or rule_unlocked("
            reTrailer", target)
        lvl_2 = lvl_2_a or lvl_2_b or lvl_2_c
        lvl_1_a = rule_unlocked("Alien Invasions", target) or rule_unlocked("
            Course Image", target)
        lvl_1_b = rule_unlocked("Looping GIF", target) or rule_unlocked("
            Pixel Art", target)
        lvl_1_c = rule_unlocked("Publicist", target) or rule_unlocked("reMIDI
            ", target)
        lvl_1 = lvl_1_a or lvl_1_b or lvl_1_c
        lvl = 0
        lvl = 1 if lvl_1 else 0
        lvl = 2 if lvl_2 else max(lvl, 0)
        lvl = 3 if lvl_3 else max(lvl, 0)
        lvl > 0
    then:
        award_badge(target, "Tree Climber", lvl)
#############################################################################

rule: Wild Imagination
# Suggest presentation subjects
# lvl.1: suggest a new subject for your presentation
 when:
  logs = GC.participations.getAllParticipations(target, "suggested
      presentation subject")
  nlogs = len(logs)
  lvl = compute_lvl(nlogs, 1)
 then:
  award_badge(target, "Wild Imagination", lvl, logs)
```

Listing B.2: Skill rules

```
rule: Album Cover
# Complete the skill Album Cover with a grade of 3 or more
 when:
  logs = GC.participations.getSkillParticipations(target, "Album Cover")
  rating = get_rating(logs)
  rating >= 3
 then:
  award_skill(target, "Album Cover", rating, logs)
#############################################################################

rule: Audiobook
# Complete the skill Audiobook with a grade of 3 or more
 when:
  logs = GC.participations.getSkillParticipations(target, "Audiobook")
  rating = get_rating(logs)
```

```
    rating >= 3
  then:
    award_skill(target, "Audiobook", rating, logs)
############################################################################

rule: Course Logo
# Complete the skill Course Logo with a grade of 3 or more
  when:
    logs = GC.participations.getSkillParticipations(target, "Course Logo")
    rating = get_rating(logs)
    rating >= 3
  then:
    award_skill(target, "Course Logo", rating, logs)
############################################################################

rule: Movie Poster
# Complete the skill Movie Poster with a grade of 3 or more
  when:
    logs = GC.participations.getSkillParticipations(target, "Movie Poster")
    rating = get_rating(logs)
    rating >= 3
  then:
    award_skill(target, "Movie Poster", rating, logs)
############################################################################

rule: Podcast
# Complete the skill Podcast with a grade of 3 or more
  when:
    logs = GC.participations.getSkillParticipations(target, "Podcasts")
    rating = get_rating(logs)
    rating >= 3
  then:
    award_skill(target, "Podcast", rating, logs)
############################################################################

rule: Reporter
# Complete the skill Reporter with a grade of 3 or more
  when:
    logs = GC.participations.getSkillParticipations(target, "Reporter")
    rating = get_rating(logs)
    rating >= 3
  then:
    award_skill(target, "Reporter", rating, logs)
#############################################################################

rule: Alien Invasions
# Complete the skill Alien Invasions with a grade of 3 or more
  when:
    rule_unlocked("Movie Poster", target)
    rule_unlocked("Podcast", target)
    logs = GC.participations.getSkillParticipations(target, "Alien Invasions")
    rating = get_rating(logs)
    rating >= 3
  then:
    award_skill(target, "Alien Invasions", rating, logs)
#############################################################################

rule: Course Image
# Complete the skill Course Image with a grade of 3 or more
  when:
    rule_unlocked("Course Logo", target)
    rule_unlocked("Reporter", target)
    logs = GC.participations.getSkillParticipations(target, "Course Image")
    rating = get_rating(logs)
    rating >= 3
  then:
    award_skill(target, "Course Image", rating, logs)
#############################################################################

rule: Looping GIF
# Complete the skill Looping GIF with a grade of 3 or more
  when:
    rule_unlocked("Movie Poster", target)
    rule_unlocked("Reporter", target)
    logs = GC.participations.getSkillParticipations(target, "Looping GIF")
    rating = get_rating(logs)
    rating >= 3
  then:
    award_skill(target, "Looping GIF", rating, logs)
#############################################################################

rule: Pixel Art
# Complete the skill Pixel Art with a grade of 3 or more
  when:
    rule_unlocked("Podcast", target)
    rule_unlocked("Course Logo", target)
    logs = GC.participations.getSkillParticipations(target, "Pixel Art")
    rating = get_rating(logs)
    rating >= 3
  then:
    award_skill(target, "Pixel Art", rating, logs)
#############################################################################

rule: Publicist
# Complete the skill Publicist with a grade of 3 or more
  when:
    rule_unlocked("Album Cover", target)
    rule_unlocked("Movie Poster", target)
    logs = GC.participations.getSkillParticipations(target, "Publicist")
    rating = get_rating(logs)
    rating >= 3
  then:
    award_skill(target, "Publicist", rating, logs)
#############################################################################

rule: reMIDI
# Complete the skill reMIDI with a grade of 3 or more
  when:
    rule_unlocked("Album Cover", target)
    rule_unlocked("Audiobook", target)

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

```
 then:
  award_skill(target, "reMIDI", rating, logs)
#############################################################################

rule: Cartoonist
# Complete the skill Cartoonist with a grade of 3 or more
 when:
  rule_unlocked("Looping GIF", target)
  rule_unlocked("Pixel Art", target)
  logs = GC.participations.getSkillParticipations(target, "Cartoonist")
  rating = get_rating(logs)
  rating >= 3
 then:
  award_skill(target, "Cartoonist", rating, logs)
#############################################################################

rule: Fake Speech
# Complete the skill Fake Speech with a grade of 3 or more
 when:
  rule_unlocked("Course Image", target)
  rule_unlocked("reMIDI", target)
  logs = GC.participations.getSkillParticipations(target, "Fake Speech")
  rating = get_rating(logs)
  rating >= 3
 then:
  award_skill(target, "Fake Speech", rating, logs)
#############################################################################

rule: Foley
# Complete the skill Foley with a grade of 3 or more
 when:
  rule_unlocked("Publicist", target)
  rule_unlocked("Pixel Art", target)
  logs = GC.participations.getSkillParticipations(target, "Foley")
  rating = get_rating(logs)
  rating >= 3
 then:
  award_skill(target, "Foley", rating, logs)
```

```
############################################################################

rule: Kinetic
# Complete the skill Kinetic with a grade of 3 or more
 when:
  rule_unlocked("Looping GIF", target)
  rule_unlocked("Alien Invasions", target) or rule_unlocked("reMIDI", target)
  logs = GC.participations.getSkillParticipations(target, "Kinetic")
  rating = get_rating(logs)
  rating >= 3
 then:
  award_skill(target, "Kinetic", rating, logs)
############################################################################

rule: reTrailer
# Complete the skill reTrailer with a grade of 3 or more
 when:
  rule_unlocked("Publicist", target)
  rule_unlocked("Course Image", target)
  logs = GC.participations.getSkillParticipations(target, "reTrailer")
  rating = get_rating(logs)
  rating >= 3
 then:
  award_skill(target, "reTrailer", rating, logs)
############################################################################

rule: Stop Motion
# Complete the skill Stop Motion with a grade of 3 or more
 when:
  rule_unlocked("reMIDI", target)
  rule_unlocked("Alien Invasions", target)
  logs = GC.participations.getSkillParticipations(target, "Stop Motion")
  rating = get_rating(logs)
  rating >= 3
 then:
  award_skill(target, "Stop Motion", rating, logs)
############################################################################
```

```
rule: Director
# Complete the skill Director with a grade of 3 or more
 when:
  opt_1 = rule_unlocked("Stop Motion", target) and rule_unlocked("reTrailer",
      target)
  opt_2 = rule_unlocked("Foley", target) and rule_unlocked("Kinetic", target)
  opt_1 or opt_2
  logs = GC.participations.getSkillParticipations(target, "Director")
  rating = get_rating(logs)
  rating >= 3
 then:
  award_skill(target, "Director", rating, logs)
#############################################################################

rule: Series Intro
# Complete the skill Series Intro with a grade of 3 or more
 when:
  rule_unlocked("Cartoonist", target)
  rule_unlocked("Stop Motion", target) or rule_unlocked("Fake Speech", target
     )
  logs = GC.participations.getSkillParticipations(target, "Series Intro")
  rating = get_rating(logs)
  rating >= 3
 then:
  award_skill(target, "Series Intro", rating, logs)
```

Listing B.3: Grade rules

```
rule: Lab Grade
# Get grades from the labs
 when:
  logs = GC.participations.getParticipations(target, "lab grade")
  len(logs) > 0
 then:
  award_grade(target, "lab", logs)
#############################################################################

rule: Presentation Grade
```

```
# Get grades from the labs
 when:
  logs = GC.participations.getParticipations(target, "Presentation grade")
  len(logs) > 0
 then:
  award_grade(target, "presentation", logs)
#############################################################################

rule: Quiz Grade
# Get grades from the quizzes
 when:
  logs = GC.participations.getAllParticipations(target, "quiz grade")
  len(logs) > 0
 then:
  award_grade(target, "quiz", logs)
```

Listing B.4: Other rules

```
rule: Initial Bonus
# Bonus given to all the students in the beginning of the course!
 when:
  xp = METADATA["initial_bonus"]
 then:
  award_prize(target, "Initial Bonus", xp)
```