Autogame

Inês Paiva
Instituto Superior Técnico
Lisboa, Portugal

Abstract

Education plays a very important role in society, shaping young people and giving them 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

1 Introduction

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* [14], a QR module and Smartboards [2]. 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 [2], 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.

This project's objective is to create Autogame, a rule-based system that will use Gamecourse's data to calculate the student's awards and feed them back to the system. This system must be automatic, incremental, efficient, and provide a solution for the cases where a grade retraction happens. This means that the system must be able to go back on awards that were previously calculated if the data changes.

2 Related work

MCP uses Gamecourse, a platform that interacts with other external sources, to provide the course's gamification experience. Apart from that, teachers run a script that uses data

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

from those sources to calculate the course's awards. This is not an efficient or automatic process, so our solution is the creation of a rule-based system.

A rule-based system uses rules to apply knowledge and calculate outputs. These systems have shown results in areas such as medicine, by helping the diagnose of multiple sclerosis [7] and tuberculosis [17]; and in other areas, for example, in the integration of information in onboard devices [8].

In this section, we analyze the different kinds of rule-based systems and their advantages and disadvantages. We divide them into visual and non-visual, depending on 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. 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 [18]. 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.

2.1.2 Natural Language Processing. NLP uses linguistics and artificial intelligence to extract information from text [20]. 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 [10].

This method has been used in a framework that extracts rules from online text [16], 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.

2.1.3 Audio. The use of audio interfaces is a subject that is starting to gain relevance in artificial intelligence, with applications such as Siri and Alexa. D'este et al. have created a medication review robot [5] 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.

2.2 Visual rule systems

This section explains visual approaches to create rules in a rule-based system, gives an example, and describes each of the approaches' advantages and disadvantages.

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 [9].

The use of flowcharts to create rules provides a more visual representation, however, 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.

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, making it impossible for the user to create more complex rules that depend on more than two variables.

2.2.3 Drag-and-Drop. Drag and drop method [15] 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*. This rule editor is as generic as possible, so it can be adapted and used in different domains and by different users, just by changing the primary inputs and outputs that the system works with.

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 [11] by textitBalticMuseums: Love IT! international project [3] which has an e-guide gamification web service for tourists. This editor helps the addition, view, modification, and deletion of rules for this service, which users can use on their mobile devices. Also, in a project regarding the Internet of Things [21]. Here, the authors defend that objects have events, states that can be used as conditions and methods representing the results or actions. Using this information, users could create a great variety of rules that would interact with objects.

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 [4], 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 and their colleagues' progress using the leaderboard, which was created using Smartboards [2].

2.4 Discussion

This study 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.

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 setbystep 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 the articles studied. 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

Gamecourse is a platform developed in PHP 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 [3]. In 2013, an MSc student started working on a web application to replace this script as his MSc project, but it was never finished [1]. 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 [6]. In the same year, Matilde Nascimento created a web application that allowed teachers to keep track of the students that attended lectures [14]. Nowadays, apart from myself, two other MSc students are working on this platform, Diana Lopes on the back-end [12] and Patrícia Silva on the front-end [19].

In Fig. 1 we provide Gamecourse's architecture. This system uses multiple web applications that work together to provide the students with a good gamification experience. These web applications work as external sources of information that provide data that is stored in the system's database. These sources are: GoogleSheets to store any grades the teachers have to give manually; Classcheck used to keep track of the students that attended lectures; Moodle, where the students publish their works, receive grades, and complete the quizzes; and QR module, used to track the students that have participated in lectures. Each one of these sources has its plugin responsible for retrieving its data and storing it in the system's database.

3.1 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.

The data from the external sources referred to in the previous subsection goes to a table called *participation*, where

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

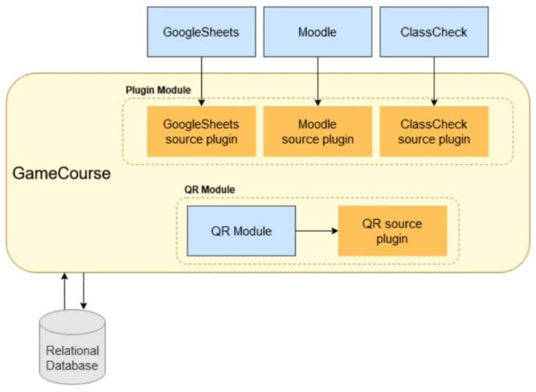


Figure 1. Gamecourse's architecture.

every action done by the students for the course is stored. This is the information used to calculate the course's awards, such as badges, completed skills, and grades. To fill this table, the administrator has to enable the plugin module and set a periodicity with which the plugins will retrieve the sources' data. Then, from ClassCheck, the system will retrieve attendances; from Moodle, it will retrieve posts and grades from quizzes and skills; finally, from GoogleSheets, it will retrieve any other grade or badge that the teachers need to award manually. Each line on this table refers to an action performed by a student, such as attending a lecture or completing a skill. From now on, we will refer to each of these actions that go on this table as participation.

3.2 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 that include 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 that has a list of all the libraries and functions available for a course and information about the functions, such as its arguments, definitions, keywords, respective libraries, and what data type it returns.

3.3 Modules

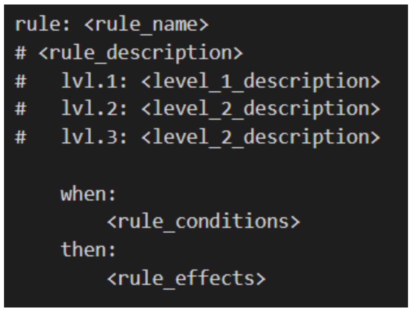
Gamecourse allows the use of a set of modules, increasing the complexity of the course's gamification experience. The administrator of the course has the power to enable or disable the modules as he sees fit. Some modules require other modules to be enabled, and enabling a module adds more functionalities to the course's platform. 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).

4 Gamerules

Gamerules is a rule system developed in Python 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 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 help Autogame.

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 followed a specific structure (Fig. 2).



```
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 2. Generic rule used in Gamerules.

To be valid, a rule must have the following components: name, description, when block, and then block. The name of the rule is essential because it can be used as an argument for another function. The description only exists to make the rule more readable and give an idea of what it does. The when block consists of a set of assignments, functions, and conditions, and the then block is composed of a set of assignments and functions that are only compiled if the conditions in the previous block are true.

4.2 Rule parser

The rule parser is the part of the system that is responsible for interpreting the rules. It does this by going through every character on the rule file and looking for keywords that indicate which part of the rule comes next. The parser starts by saving everything after "rule:" as the rule name. It ignores the comments because they do not need to be compiled and are irrelevant to the system. After the comments, it looks for "when:" and it will store each line of this block. Finally, it saves everything after the keyword "then:" line by line, as the actions of the rule.

The method of compiling the rules line by line makes it impossible for the system to know how to run statements that take for than one line, such as for loops. However, the user can easily go around this by creating auxiliary functions any time the system needs to run code more complex than the rule parser can deal with. These functions can be added to the system using special wrappers. For functions used in the when block, it needs to have the @rule_function wrapper, and for the ones used in the then block, there is the @rule_effect wrapper.

4.3 Input and output

Gamerules, being a rule-based system, needs a set of facts and rules to work with. These facts and rules that the system receives consist of a set of text files. There are four categories of files used as input for this system: course files, gamification files, log files, and rules. The first one consists of two files, one is a list of teachers, and one 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. This last category of files is downloaded for each of the external data sources and will be used to calculate the awards.

As for the system's output, it is 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, and the respective value is a 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. There is a rating for skills, for badges, there is a level, and for grades, there is XP.

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. 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, such as additional variables, such as the 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.

5 Autogame

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 rule system's base was already working, so we focused on its integration with Gamecourse, which was already a monumental task.

Autogame (Fig. 3), 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.

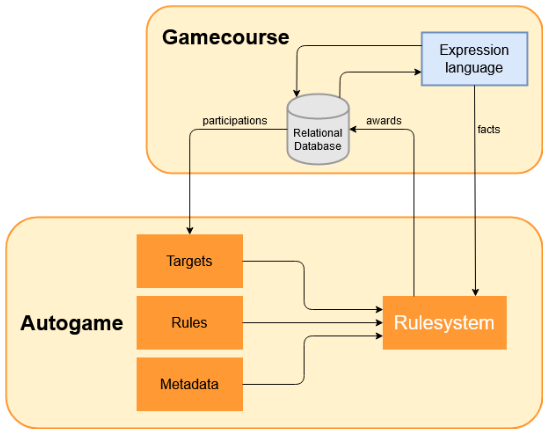


Figure 3. Autogame's architecture.

5.1 Upgrading to python 3

One of the first problems we encountered when trying to create Autogame was 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* [4]. Most of the changes made were simple ones, such as exchanging function names, how a variable

[4]https://docs.python.org/3/library/2to3.html

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 [12], 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 [5] 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.

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 four types: badges, skills, grades, and others.

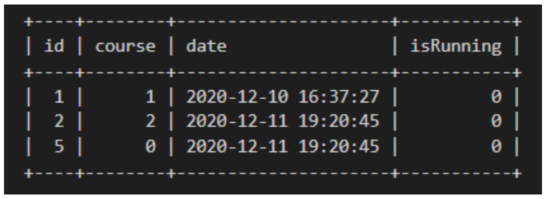
5.3 Communication with Gamecourse

To fully integrate Autogame with Gamecourse, it was essential that both could establish communication. This was important for two main reasons: so that Autogame could use Gamecourse's expression language; and so that Gamecourse could fire Autogame when necessary. 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 researched to find the best solution for our problem. After testing some frameworks, we concluded that the best way to create a channel between both systems was by using a Transmission Control Protocol (TCP) socket. For this, we created a PHP script that would act as a server and a python one that would act as a client. At this point, these scripts were run manually.

The next step was to make sure that the system could run Gamecourse's functions. Each time a Gamecourse function is called in a rule, our system needs to create a client to communicate with the server. To make sure that the server does not close before Autogame finishes running, we made a condition to keep the server socket open until Autogame sends a message for it to close.

For Autogame to run, it needs to be called by Gamecourse. This happens every time one of the plugins retrieves new data from external sources. To make this possible, we created a new PHP script that combines both the PHP server and the Python client and is run by the plugins. This way, whenever Autogame needs to run, it will open a server socket and any clients necessary.

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 Autogame table. 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 4. Autogame table.

5.4 Changing rule parser

After establishing communication with Gamecourse, we had to ensure that our system recognized a Gamecourse function and created the channel to run the function there. To do that, first, we established that all these functions had to be called using a specific structure, and then we changed the rule parser so that it was able to recognize them. The structure is the following:

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

Finally, with the rule parser changed so that it recognizes the "gc" at the beginning of the line and saves everything after it, we created the *gc* function, which opens a client socket to communicate with Gamecourse. This way, whenever the rule parser finds one of these functions, it calls this gc function that communicates with Gamecourse and sends it all the arguments necessary to run 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

[5] https://www.mysql.com/

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. To solve this problem, we shifted our focus from running 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. The criterion is: 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. 4), 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 running 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.

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 to retract what was previously awarded.

To deal with this, before writing the output on the database, Autogame always checks what is already there. We covered every possibility and made sure that it knows how to deal with every situation. For example, there may be up to three lines on the participation table for a badge and target because a badge can have up to three levels. Before awarding a new level, the system checks how many lines are already on the database regarding that badge and that student and compares that value with the new level calculated. Then, depending on the values, it knows if it has to delete, update, or insert lines.

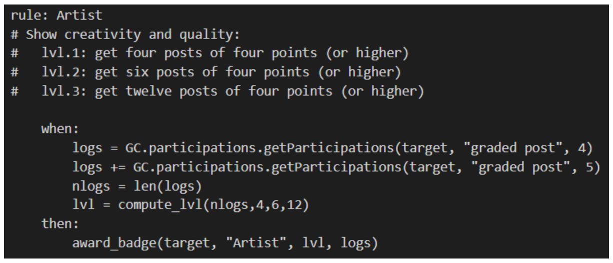
This method of dealing with retractions always makes sure that the database's awards are correct, but this implementation created a problem. Most of the rules related to badges had a condition to ensure that a rule was only fired if the level calculated was superior to zero (so, no awards for badges 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 primary 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.

The first step was to create a config file where we inserted the METADATA dictionary. This dictionary has information that needs to be accessed within the rules, such as the number of lectures or the maximum grade for the laboratories, and by putting it in the config file, the system knows that it has to add it to the scope of the rule system. Then, we create a folder with the rules using the id of the course as the folder's name.

After the first configurations, we proceed to create the rules. These rules all have a similar structure. First, we assign some data using METADATA if necessary; we retrieve students' participations from Gamecourse and then perform some actions on the data using auxiliary functions. We may have one or more conditions for the rule if necessary, and finally, we award the badge if the preconditions are met. In Fig. 5 we provide the code for badge artist rule. In this case, we wanted to award the badge depending on how many posts with a rating greater than three the target had. So, we retrieve all the participations that fit this description, then we count how many there are, and finally, we compute the level that is deserved depending on the number of participations.



```
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)
```

Figure 5. Autogame rule: Artist badge.

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, developed a set of new functions that are now available for any new course that uses Autogame.

6 Evaluation

To ensure the quality of Autogame, we conducted a thorough evaluation to cover all the system aspects that might not have been working as intended. This evaluation process was composed of two phases: correctness tests and performance tests. In this section, we will explain how we conducted these tests and what the results were.

6.1 Correctness Tests

For the first phase of Autogame's evaluation, we conducted a series of tests to assess if the system generates a correct result. To do this, we compared our results using MCP's data from last semester with the results from the previous system. We used a mockup of Autogame that we had created to conduct these tests, where we replicated the database.

Using a python's library called unittest [6], we created a test for each one of the MCP's rules. Each test compared the number of students that received that award for both systems, and if this number was the same, it compared each one of the arguments of the students' awards (badge level, grade, skill rating, etc).

In the end, our system passed almost all of the tests, which the exception of four. Three of these tests failed due to different data being used. This happened because some of the awards calculated last year were not updated at the end of the semester. So, we checked each of them and changed the database's data to have the correct result. After making these changes, there was only one test that our system was failing, the one for the Tree Climber badge rule. We investigated this issue and found out that the previous system was miscalculating this award. For example, level one of this badge is supposed to be awarded if a student completes a skill from tier 2 of the skill tree, which means that the student has to have a rating of at least 3 in one of those skills. However, the previous system awarded the badge disregarding the skill's rating. So, both students who had this badge for the previous system and not ours were in this situation. This means that they were not supposed to have the award in the first place.

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

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 takes to run, in seconds, and how much memory is used.

6.2.1 Increasing system's targets. For the first phase of the performance evaluation, we created a set of tests where we used all the 49 rules from MCP and gradually increased the targets. The target used was always the same but with different names. Then, using python's libraries time [7] and resource [8] we managed to calculate the running time, in seconds, and memory usage peak, in Mebibytes, for each test. To ensure that no external factors were altering the results, we run the tests three different times and calculate the average, provided in Table 1.

Table 1. System's average running time (s) and memory usage peak (MiB) when increasing the number of targets, with 49 rules.

Targets	Time	Memory peak
10	3,55	22,87
50	12,32	26,03
100	25,03	29,88
200	52,34	37,79
500	140,15	61,98
750	207,43	82,44
1000	283,89	101,57

6.2.2 Increasing system's rules. Using a similar process has in the previous sections, we created a set of tests where we gradually incremented the number of rules in the system, using 100 targets, the students from MCP. For this case, we used the same rule repeatedly, but with a different name so

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

that the system would calculate different awards for each rule. Using the same libraries as before, we ran the tests three times and averaged the results, resulting in the values in Table 2.

Table 2. System's average running time (s) and memory usage peak (MiB) when increasing the number of rules, with 100 students.

Rules	Time	Memory peak
5	3,73	22,27
10	6,20	23,17
25	15,56	25,46
50	31,57	29,13
75	49,42	32,81
100	67,78	36,83
150	105,73	44,38

6.3 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. We concluded that both the time and the memory usage peak have a linear evolution for targets and rules for both tests. This gives us an idea of the system's load when using specific numbers of targets and rules. For MCP's case, the system takes around 25 seconds and 30 MiB to run everything. However, it is essential to remember that this is an incremental system and rarely will run for all the students. So, usually, considering that the periodicity with which the system runs will be low, the values will probably be somewhere between 3 and 15 seconds and 22 and 25 MiB.

7 Conclusions

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 rule system that was never finished. 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.

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.

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.

References

[1] J. Amaral. 2013. *Gamecourse*. Master's thesis. Instituto Superior Técnico, Universidade de Lisboa.

[2] A Baltazar. 2016. *SmartBoards*. Master's thesis. Instituto Superior Técnico, Universidade de Lisboa.

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

[4] G. Barata, S. Gama, J. Jorge, and D. Gonçalves. 2011. So Fun It Hurts – Gamifying an Engineering Course. *International Conference on Augmented Cognition* (2011).

[5] C. D'Este, D. Reid, and B. H. Kang. 2008. A Robotic Interface to a Medication Review Expert System. *International Symposium on Ubiquitous Multimedia Computing* (2008).

[6] A. Dourado. 2019. *GameCourseNext*. Master's thesis. Instituto Superior Técnico, Universidade de Lisboa.

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

[8] W. B. Guo, X. P. Hu, and J. Liu. 2006. Rule-based Reasoning in Onboard Devices: An Intelligent Route Guidance System. *IEEE International Conference on Service Operations and Logistics, and Informatics* (2006).

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

[10] H. Isahara. 2007. Resource-based Natural Language Processing. *International Conference on Natural Language Processing and Knowledge Engineering* (2007).

[11] A. Kulpa, J. Swacha, and K. Muszynska. 2019. Visual Rule Editor for E-Guide Gamification Web Platform. *Federated Conference on Computer Science and Information Systems (FedCSIS)* (2019).

[12] D. Lopes. 2021. *GameCourse Beyond*. Master's thesis. Instituto Superior Técnico, Universidade de Lisboa.

[13] Mauro Mosconi and Marco Porta. 2000. 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] M. Nascimento. 2019. *I am here!* Master's thesis. Instituto Superior Técnico, Universidade de Lisboa.

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

[16] Hassanpour S., O'Connor M.J., and Das A.K. 2011. A Framework for the Automatic Extraction of Rules from Online Text. (2011).

[17] A. R. C. Semogan, B. D. Gerardo, B. T. T. III, J. T. d. Castro, and L. F. Cervantes. 2011. A Rule-Based Fuzzy Diagnostics Decision Support System for Tuberculosis. *Ninth International Conference on Software Engineering Research, Management and Applications* (2011).

[18] Yang Shanliang, Fu Yuewen, Zhang Peng, and Huang Kedi. 2013. 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).

[19] P. Silva. 2021. *GameCourseUI.* Master's thesis. Instituto Superior Técnico, Universidade de Lisboa.

[20] M. Stanojevic and S. Vranes. 2005. A Natural Language Processing for Semantic Web Services. *EUROCON 2005 - The International Conference on "Computer as a Tool"* (2005).

[21] T. Tuomisto, T. Kymäläinen, J. Plomp, A. Haapasalo, and K. Hakala. 2014. Simple Rule Editor for the Internet of Things. *International Conference on Intelligent Environments* (2014).