In-Browser Implementation of a Gamification Rule Definition Language Interpreter

Jakub Swacha * and Wiktor Przetacznik

Institute of Management, University of Szczecin, 71-004 Szczecin, Poland; wiktorprzetacznik@outlook.com

* Correspondence: jakub.swacha@usz.edu.pl

Abstract: One of the practical obstacles limiting the use of cloud-based gamification applications is the lack of an Internet connection of adequate quality. In this paper, we describe a practical solution to this problem by the implementation of client-side gamification rule processing so that most events generated by players can be processed without the need to involve server-side functions; therefore, only a handful of data have to be transmitted to the server for global state synchronization, and only when an Internet connection is available. For this purpose, we adopt a simple textual gamification rule definition format, implement the rule parser and event processor, and evaluate the solution in terms of performance in experimental conditions. The obtained results are optimistic, showing that the developed solution can easily handle rule sets and event streams of realistic sizes. The solution is planned to be integrated into the next version of the FGPE gamified programming education platform.

Keywords: cloud gamification platform; gamification rule specification; client-side gamification engine; rule processing

1. Introduction

1.1. Problem Context

Gamification can be aptly defined as “the act of applying game-design elements to transform activities, products, services, and systems in a way that provides the kind of experiences similar to those offered by games” [1]. Although gamification can be implemented as non-digital or “unplugged” [2], the bulk of its existing implementations are digital [3]. Therefore, there is a substantial technological component in every digital gamification implementation and a handful of technological barriers that must be overcome for the gamified system to achieve a level of technical quality acceptable for its users. In spite of that, the interest of gamification research is primarily located in its psychological aspects, as it mostly strives to explain why gamification (or its peculiar mechanic) works with at least 118 different theories employed for this purpose [4].

In this paper, we make one small step towards balancing this inequality, as we deal with a problem belonging to the gamification technology area. In particular, it addresses the sub-area of cloud gamification [5]. Cloud gamification platforms typically provide services encompassing all basic functions needed to gamify an application, i.e., including various gamification mechanics and game state maintenance (see Table 1 in [6]).

1.2. Problem Definition

The reliance on an Internet connection is an inherent attribute of all systems using cloud services [7]. If we focus on mobile users, although nowadays, mobile Internet accessibility is widely perceived as abundant, this is simply not true on many occasions. Problems with Internet access do not only happen in less developed countries (see, e.g., [8]) or in rural and sparsely populated areas (see, e.g., [9]) but may also be caused anywhere by obstacles such as thick walls and complex architectural structures (see, e.g., [10]). While the...
total lack of Internet accessibility in specific locations makes it impossible to use a cloud-supported application, there are many more locations in which mobile Internet is accessible, yet poorly. Slow data transmission and frequent connection interruptions may seriously degrade application usability (see, e.g., [11]). This could be particularly frustrating for students trying to use an educational mobile or web application.

In an effort to allow cloud-supported gamified applications to temporarily operate without Internet access, one must move gamification rule processing, at least in the part that does not rely on data acquired from other players, to the client side, leaving only the global state synchronization functions at the server. The technical challenge here lies in developing a lightweight gamification engine that could be run in a web browser environment. Moreover, if gamification is to remain configurable, that is, being allowed to use a single application with different game rule sets rather than embedding the game rules in the application code, a simple format for expressing such rules is needed so as not to make parsing the rules the most complicated part of the lightweight gamification engine.

1.3. Approach and Contributions

In this paper, we address both these challenges by first defining such a format (precisely, an update of a format proposed earlier by the first author [12]) and then presenting an implementation of a parser consuming this format and processing gamification rules it conveys, both of which are run client-side, that is, in the web browser environment (developed by the second author).

As we are aware of no prior work presenting such an in-browser interpreter of gamification rule definitions, we see our key contribution in introducing our solution. We also contribute with an updated (and somewhat simplified) version of the gamification rule specification format and the performance evaluation of the developed software, showing that there are no performance barriers for its application.

The remainder of this paper is structured as follows: Section 2 reports the work so far on client-side gamification libraries and domain-specific languages for conveying gamification rules; Section 3 describes the adapted version of the gamification rule specification language; Section 4 provides the basic facts regarding the software implementation of the lightweight rule parser and processor; Section 5.1 describes the empirical evaluation procedure, as well as the test data used and its technical environment; Section 5.2 presents the results obtained in the tests; in Section 6, the results are discussed in the context of the target application environment; finally, Section 7 summarizes the paper’s content with concluding remarks and an outline of the next steps envisaged for our future work.

2. Related Work

Gamification can, and often is, implemented in information systems by simply extending their code with functions handling game-related events, checking conditions for awarding points, badges, or level-ups, and generating audiovisual feedback to the user.

In the case of web applications, gamification can be implemented server-side and delivered to the client via cloud services, with a number of providers available, e.g., GameUp, GSparks, or PlayFab [13]. It can also be implemented client-side with all processing done in the web browser and, possibly, only global data stored on the server. The latter solution is most typically implemented using JavaScript, a scripting language supported by nearly all contemporary web browsers [14].

There are a number of JavaScript libraries providing basic gamification functionality, including open-source, such as Gamification.js [15], openGamification [16], or score.js [17]. Naturally, using any of these libraries requires the gamification rules to be specified using JavaScript (or another programming language). This is a barrier for those who have an idea of how their gamification system should work but have no programming skills. Moreover, it makes the specification of gamification rules a part of the application code. This means that any time the rules need to be updated (which could be done relatively frequently if the system is to be adapted to the changing user behaviors), the application has to be updated
as well; also, the transfer of rules to another application requires substantial programming effort [12].

The solution to this problem is providing an alternative way to specify the gamification rules and treat their specification as a kind of content that is updated independently from the software and is easy to transfer between applications.

The earliest known solution of this kind was proposed by Herzig ([18], pp. 49–63, 195–218) in the form of Gamification Modeling Language (GaML). It covers a range of gamification elements, including, among others, points, levels, badges, and leaderboards, and supports conditional statements (which may take into consideration, e.g., player status and location) and Boolean constraints in rule definitions, as well as a gamut of possible rule results (from notifications, through points, badges, and items to internal game state changes). However, although GaML has been conceived as “writable for IT experts” ([18], p. 50), it may be too difficult for non-programmers to use.

In 2018, a textual gamification rule definition language was proposed in [12]. As this is the language chosen for our implementation, it will be described in detail in Section 3. While it was conceived and originally intended to be used to gamify mobile museum e-guides, it has been replaced in that application with a visual form-based gamification rule editing tool [19], which does not use any dedicated format for the rules but instead stores them in a relational database (the target is a cloud-based platform processing the rules server-side, so there is no need to transfer them to the client side effectively).

Also, in 2018, work on the Framework for Gamified Programming Education (FGPE) commenced [20]. The primary goal of the FGPE initiative was to provide programming teachers with a set of tools and solutions allowing them to use gamification in their programming courses in a fully configurable manner, meaning that not only the programming exercises could be defined, organized, and ordered as they see fit, but also control over the gamification rules is fully granted to the teachers (more information about FGPE is provided in Section 6). As this requires the separation of the gamification layer from the gamified system (the interactive learning environment in this case) and treats gamification rules as a kind of content (editable and transferable), it motivated the development of a language dedicated to conveying them. The result of this work was GEdIL (Gamified Education Interoperability Language) [21]. As FGPE features a visual, form-based editor (FGPE AuthorKit) for gamified programming exercises and gamification rules [22], which outputs GEdIL files, there was no motivation to make GEdIL simple or suitable for manual writing by humans; consequently, GEdIL is presented in a data-exchange format, which is expected to be generated and read by computers.

There are also other visual gamification rule editors based on forms, both developed by the research community (e.g., GamiTool [23]) and commercial companies (e.g., Game-tize [24]). For visual editing, diagrams can also be used to convey the gamification rules (see, e.g., UAREI [25] or MEEdit4CEP-Gam [26]). However, while visual editors effectively solve the problem of allowing non-programmers to develop gamified systems, they do not address the problem of a simple representation of gamification rules, for which a textual domain-specific language can be a solution.

Recently, Bucchiarone et al. proposed DSL4GaR, a domain-specific language for the definition of gamification, simulation, and deployment [27]. Although its authors declare that “DSL4GaR serves as a tool to raise the level of abstraction for nontechnical people” ([27], p. 6), the rules defined with DSL4GaR look similar to Java source code (see, e.g., Listing 5 in ([27], p. 11)), which makes this declaration dubious as Java code is not particularly readable for non-programmers.

To sum up this section, there are a number of libraries used implement gamification client-side in JavaScript, and there are a number of tools, both textual and visual, dedicated to specifying gamification rules without having to resort to a general-purpose programming language, with their output processed server-side. The novel solution described in this paper strives to merge these two approaches, providing an ability for in-browser
interpretation of gamification rules defined in a simple, dedicated textual format, along with processing of gamification-relevant events.

3. Textual Gamification Rule Definition Language

The textual gamification rule definition language proposed in [12] is based on the specific representation of gamification events. Originally, each such event was described by the following tuple: player, client, area, location, object, action, result, date, and time [12].

Following [12], the rule definition is divided into rule requirements and rule results. The former use the following notation:

```
rule_name:
player_class did action with object in location of area on date at time achieving result
```

Every segment in the proposed notation defines a filter for the respective event description element. Thus, omitting a segment means that the given element’s value is irrelevant for triggering the rule.

The values (of `player_class`, `action`, `object`, etc.) can be constants, indicating a particular element, a wildcard (*), denoting that any value is acceptable (thus, equal in interpretation to omitting that segment in the definition), an element selector (see below), or an expression built using one of the following operators:

- **bigger than** (>), e.g., >10:00;
- **smaller than** (<), e.g., <10:00;
- **bigger than or equal** (>=), e.g., >=10:00;
- **smaller than or equal** (<=), e.g., <=10:00;
- **in the range of** (..), e.g., 10:00..12:00;
- **any of** (,), e.g., 10:00,12:00.

The element selector can be one of the following:

- **any**: denoting that one event involving any of the listed elements is sufficient to trigger the rule; note, this is meant to be used just for emphasis, as `any(X,Y)` works exactly as `X,Y`;
- **all**: denoting that each of the listed elements must be involved in an event to trigger the rule, as a single event can only involve one element; the minimum number of events to trigger the rule is the number of listed elements;
- **seq**: denoting that all of the listed elements must be involved in events exactly in the sequence they were listed; so, any event involving an element other than the next to process will be ignored.

The element selectors originally proposed in [12] include six more that are either omitted or modified in the presented implementation. Two of them, `group` and `member`, are used to define team challenges; however, the intended use of FGPE (for client-side gamification without an Internet connection) makes it impossible to create rules triggered by the actions of other players, so these selectors are not implemented as unusable.

Also omitted are the selectors (**always** and **each**) that are allowed to distinguish rules meant to be triggered multiple times from those meant to be triggered just once (defined with **all**). As this is far from intuitive, in the implementation described here, instead of them, an extra keyword **repeat** can be added at the end of the rule requirements definition to denote a rule that stays active after being triggered. Moreover, adding a number after this keyword denotes the number of times the rule can be triggered before it goes off (e.g., **repeat 3** means the player could trigger the rule only three times).

The two remaining selectors have only been implemented for dates (where they are mainly needed):

- **streak** allows us to define the number of consecutive days during which the respective event must occur at least once for the rule to be triggered (e.g., **streak 3** means the player would have to perform the expected action for three consecutive days to trigger the rule; any day off resets the counter).
• **every** is used only to specify the day of week (given by its English name or a number from 1 to 7; a wildcard or the number 0 denotes any day of week) on which the rule could be triggered (once per day by default unless the **repeat** keyword is used).

The original specification [12] also defines compound rule requirements, which do not refer to events but to the other specified rule requirements combined using selectors. For the sake of clarity, one of the original selectors (**each**) has been omitted in this implementation, which includes the following:

• **any** indicates that the compound rule should be triggered at any time when any of the listed rules have been triggered (e.g., **WonOrLost**: any Won Lost means **WonOrLost** will be triggered every time the player has Won or Lost);

• **all** indicates that the compound rule should be triggered only when each of the listed rules has been triggered (e.g., **WonAndLost**: all Won Lost means **WonAndLost** will be triggered only after the player has Won at least once and Lost at least once);

• **seq** indicates that the compound rule should be triggered only when each of the listed rules has been triggered exactly in the sequence they were listed in (e.g., **WonThenLost**: seq Won Lost means **WonThenLost** will only be triggered when the player has first Won at least once then Lost at least once).

The last type of rule requirements specification presented in [12] covers the rules depending on the game state, and defined as follows:

\[
\text{rule_name: game_state_property subproperty value_set,}
\]

where **game_state_property** is the game state property to be checked, **subproperty** is the specific referenced value, and **value_set** is the set of values needed for the rule to be triggered.

The second part of rule definition is the rule results definition, which specifies the effects of rule execution on the game state and the generated feedback. The notation used for them is as follows (note the semicolon separating the results if there is more than one of them) [12]:

\[
\text{rule_name -> result1 [; result2...],}
\]

Where each result is specified as follows:

result_action [selector] action_arguments1 [action_arguments2... ...] [repeat],

Where

• **result_action** denotes the type of action to be performed (**msg**: message to the player, **reward** for the player, **offer** made to the player (they can accept or refuse), **open** a locked part of the course, **restart** a rule, or **set** the new value of a status variable);

• **action_arguments** denotes the parameters of the result action;

• **selector** defines the order in which the action arguments are to be consumed: **all** indicates that all results should be produced already at the very first rule execution; **seq** indicates that, every time the rule is executed, one of the results should be produced in their order of specification; **random** indicates that, every time the rule is executed, one of the results should be chosen randomly from the list; **random_once** indicates that, every time the rule is executed, one of the results should be chosen randomly from the list, excluding those results that have already been produced; **choice** indicates that, every time the rule is executed, one of the results should be chosen by the player from the list, excluding those results that have already been produced;

• **repeat** keyword indicates that had all action arguments been used in subsequent rule executions, it should start again from the first one; note that this implementation differs from the original specification [12] in shortening the keyword (originally: **repeated**) and moving it from the beginning to the end of the action list; note also, the newly introduced availability of **repeat** in both rule requirements and results specifications allows us to define rules that may be triggered many times (and used in repetitive compound rules) yet directly result in rewards only the first time they are triggered.
4. Software Implementation

The lightweight rule parser and processor software was implemented in Rust, which is a general-purpose programming language that emphasizes performance, type safety, concurrency, and correctness [28]. It ensures memory safety through a “borrow checker”, which tracks references’ lifetimes during compilation. Rust version 1.77.0 (2021 edition, with the latest toolchain as of 27 March 2024) was used. The Rust source code was compiled to WebAssembly, a portable binary instruction format for a stack-based virtual machine [29], so that it can be run in modern web browsers. The choice of WebAssembly as the target language was motivated by its high performance [30].

Implementation depends on a few external libraries:

- **wasm-bindgen** (0.2.92): high-level interactions between Wasm and JavaScript;
- **chrono** (0.4.35): advanced date and time library;
- **clap** (4.5.3): full-featured command line argument parser (used for the purpose of performing the tests);
- **regex** (1.10.4): implementation of regular expressions for Rust (used for rule parsing);
- **serde** (1.0.197): generic serialization/deserialization framework;
- **serde_json** (1.0.114): JSON format serialization.

In the implementation, structural separation of concerns was used. The core of the implementation is the library, which contains the core functionality and exposes publicly what is needed to use it. An associated small program developed for testing contains input data parsing logic, sets up a configuration, and calls appropriate functions in the library.

No “unsafety” was used, which helps to verify the implementation’s correctness. “Unsafe” features require the programmer to uphold memory safety manually. By default, Rust is “safe”, which means it cannot cause undefined behavior without explicit use of “unsafety”. Some behaviors considered undefined are as follows (this list is not exhaustive):

- data races;
- accessing (loading from or storing to) a place that is dangling or based on a misaligned pointer;
- performing a place projection that violates the requirements of in-bounds pointer arithmetic;
- mutating immutable bytes;
- calling a function with the wrong call ABI (application binary interface).

Figure 1 presents the general architecture of the in-browser rule parser and processor, indicating its primary components. Note that, although we call it an interpreter, its mode of operation resembles more a just-in-time compiler, as the whole rule set (imported as the textual list of rules) is parsed right after loading into an internal representation, which is then used to effectively process the gamification-related events generated by the user.

![Figure 1. Architecture of the in-browser rule parser and processor.](image)

The rule parser is implemented using regex expressions, which extract the rule name and a set of key-value pairs that define the rule type and bounds. Note that the caching system prevents redundant parsing. Any key-value pair absent in the rule specification is treated as a “wildcard” (or default value in case of repetition key-value pair) that matches any event unconditionally. A complex rule needs at least one value to be specified in
order to be syntactically correct. The rules considered incorrect are reported as erroneous results with failure details indicated. Otherwise, an instance of rule object is produced. The rule object contains all crucial rule information, such as its name, type, repetition count, completion flag, and relevant elements’ data.

The event processor is invoked whenever an event is triggered and checks the event parameters to identify all matching rule objects, which are then updated in a way depends on the rule type. If all rule conditions have been fulfilled, the rule results are produced, its repetition count is decreased, and if it falls to zero, the rule completion flag is set; therefore, such a rule object is no longer updated unless the rule is reset as a result of another rule.

5. Results

5.1. Test Procedure, Data, and Environment

The goal of the empirical evaluation of the implementation was to verify that the lightweight rule parser and processor are able to effectively handle a high load of player-generated events with a realistic set of processed gamification rules. After analyzing the publicly available FGPE-based gamified programming exercise sets [22], we observed that the number of rules defined in them varies considerably, and we eventually decided to use 14 gamification rules in the tests (11 simple and 3 compound), which is a realistic estimate of the number of rules sufficient to form a well-designed gamification layer of a computer programming course.

The rules used in the tests, expressed according to the format described in Section 3, are listed in Table 1.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>SolvedEasy: player * did submit with * in * of area2 achieving &gt;50 repeat 1</td>
</tr>
<tr>
<td>S2</td>
<td>FirstPerfectSolution: player * did submit with * in * of * achieving 100</td>
</tr>
<tr>
<td>S3</td>
<td>KnowsBuiltins: player * did submit with all(*) in section7 of area1 achieving &gt;50</td>
</tr>
<tr>
<td>S4</td>
<td>EarlyBird: player * did submit at &lt;08:00</td>
</tr>
<tr>
<td>S5</td>
<td>FinishedInteractivePython: player * did submit in section1 of area2 achieving &gt;50</td>
</tr>
<tr>
<td>S6</td>
<td>FinishedNumbers: player * did submit in section2 of area2 achieving &gt;50</td>
</tr>
<tr>
<td>S7</td>
<td>FinishedStrings: player * did submit in section3 of area2 achieving &gt;50</td>
</tr>
<tr>
<td>S8</td>
<td>FinishedVariables: player * did submit in section4 of area2 achieving &gt;50</td>
</tr>
<tr>
<td>S9</td>
<td>TalentedDebugger: player * did submit in section13 of area2 achieving 100</td>
</tr>
<tr>
<td>S10</td>
<td>FinishedExceptions: player * did submit in section28 of area2 achieving &gt;50</td>
</tr>
<tr>
<td>S11</td>
<td>AdvancedInNoTime: player * did submit with all(<em>) in all(</em>) of area3 on &lt;1 June 2024 achieving &gt;50</td>
</tr>
<tr>
<td>C1</td>
<td>FastAndPerfect: all FirstPerfectSolution EarlyBird</td>
</tr>
<tr>
<td>C2</td>
<td>RightOrder1: seq FinishedInteractivePython FinishedNumbers FinishedStrings FinishedVariables</td>
</tr>
<tr>
<td>C3</td>
<td>DebugReady: all FinishedExceptions TalentedDebugger</td>
</tr>
</tbody>
</table>

Note that the test set does not include rules depending on the game state, as this type of rule is more often processed server-side, where the global state of the game (and all relevant variables) could be accessed.

Taking into consideration the difference in processing time of various events and rules, we used a script to generate 10,000 events aiming to cover triggering all of the defined rules. Considering other factors that could affect the processing time, we also prepared five batches containing the same set of rules and events to obtain five distinct time measurements in five subsequent test runs.

The time elapsed during various stages of execution was measured using JavaScript's `performance.now` function that provides a high-resolution timestamp.
The implementation was compiled using the Rust 1.77.0 compiler into WebAssembly. The experiments were performed on a Firefox version 124.0.1 (64-bit) web browser running on a Windows 10 Pro (version 19045.4170) operating system. Note that Firefox is reported as having the slowest script execution (see, e.g., [31]); therefore, the execution times on other web browsers should be even shorter.

The hardware environment in which the experiments were performed is described in Table 2.

**Table 2.** Technical specification of the test environment.

<table>
<thead>
<tr>
<th>Component</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Core i5-4670k 3.40 GHz base frequency (overclocked at 4.30 GHz)</td>
</tr>
<tr>
<td>RAM</td>
<td>16 GB DDR3 SDRAM (800 MHz)</td>
</tr>
<tr>
<td>GPU</td>
<td>AMD Radeon RX 580 8 GB</td>
</tr>
<tr>
<td>Storage</td>
<td>Transcend 230S SSD 512 GB</td>
</tr>
</tbody>
</table>

5.2. **Test Results**

Gamification rule processing consists of two stages performed separately:

- Parsing the rules: carried out after they are loaded from the server, usually only once, after which the rules are stored client-side using an internal representation suitable for the rule processing engine;
- Processing the events: carried out every time a gamification-related event is generated, after which all defined rules are checked for relevancy, and those meeting the criteria specified in their definition are triggered.

Note that the event processing time is of higher importance, as the rule parsing is performed only when the given programming course is loaded on the user’s system (on its first run, after its update, or after the user erases the application’s local storage), whereas the events are processed during the entire time the application is being used. Nonetheless, the rule parsing time also matters, as the full set of rule definitions is parsed as a whole, so the user observes the combined delay, whereas the events are processed one at a time.

Figure 2 presents the rule parsing times of the 14 rules used in the test, measured for the five consecutive test runs.

![Figure 2. Rule parsing times (in ms per rule).](image-url)
As can be observed in Figure 2, the average rule parsing time was approximately 1 ms. The parsing of the whole ruleset was completed, on average, in 15.23 ms, with a minimum measured time of 14.21 ms and a maximum time of 17.53 ms. These times are unnoticeable by the end-user, especially considering the much longer time needed to download the course contents that directly precede the parsing of the rules.

Figure 3 presents the average event processing times, measured for the five consecutive test runs involving 10,000 events each.

![Figure 3. Event processing times (in µs per event).](image)

As can be observed in Figure 3, the average event processing time was below 1 µs, precisely 0.737 µs on average, 0.879 µs maximum, and 0.665 µs minimum. The obtained results indicate there is no need to test how end-users perceive the delay caused by client-side processing of gamification events as times as short as the measured ones are unnoticeable by humans.

6. Discussion

A client-side rule processing engine defines two natural requirements: it not only has to be lightweight (considering the relatively limited computing resources of mobile devices, through which web applications are often used, as compared to a web server), but also the format of the rules must be simple and succinct, making sure definitions can be effectively transmitted over the Internet and processed by the client.

Although gamification rules are often specified using visual form-based tools (see, e.g., [19,22,23]), which are simple to use by humans, from a technical point of view, it is the format they are conveyed in that matters for the complexity and speed of their processing. From the available gamification rule specification formats, we found the one proposed in [12] as particularly suitable for implementing a client-side rule processing engine, considering its relative simplicity in comparison to other such formats (e.g., [18,21]).

While the solution described in this paper is universal, that is it can be applied to gamified web systems of any kind, our work was not carried out without a specific application goal. The intended point of application of the presented solution is the next version of the FGPE platform, an open-source fully configurable e-learning environment for programming education [20]. To understand the specifics of the FGPE platform, some background must be provided first. Games and computer programming education have been paired for a long time. One idea was to make the latter more interesting for students by incorporating...
the former as the subject of programs developed as exercises (see, e.g., [32]). Another was to let students play educational games which require writing code to solve challenges and/or compete with other students; Miljanovic and Bradbury identified 49 examples of such games [33]. While the first approach does not make the programming learning experience game-like, the other focuses on a scope more or less different from what the students would need to write real-world applications. There is yet another option, consisting of the gamification of programming courses. While this could be realized using general-purpose educational tools supporting gamification, such as Moodle [34] or Kahoot! [35], they do not provide the space for coding practice which is essential in learning how to program [36–38]. Coding playgrounds with an automated assessment of students’ solutions are provided within programming learning environments and the best known of them are gamified, e.g., Codecademy [39], Code School [40], and CheckiO [41]. Platforms of this kind most often come as a complete bundle featuring a defined set of exercises and a gamification scheme. This does not suit programming teachers who would like to define their own scope of the course and gamification rules. The FGPE platform is the response to this problem.

The FGPE platform is a complex web-based system comprising several software components, primarily including the exercise editor and manager (FGPE AuthorKit), web service processing gamification rules and maintaining the overall game state (FGPE Gamification Service), and the interactive learning environment (FGPE PLE), allowing students to both solve gamified exercises and receive graded feedback within a web browser; furthermore, teachers are able to organize their programming courses and supervise their students’ progress [22]. Its reliance on sustained web connection is an issue for cases of learning during travel and/or in areas with limited Internet availability.

This is why one of the currently pursued development directions of the FGPE platform is allowing students to use it for long periods without the need for a sustained Internet connection [42]. One of the prerequisites for that to happen is the ability to process at least a part of the gamification rules client-side to avoid querying the FGPE Gamification Service. This means a local gamification rule processor must be developed to work within a web browser. As the complexity of the currently used GEdIL makes it incompatible with the concept of the lightweight gamification rule processor, the idea of adapting the textual representation proposed in [12] for this purpose motivated the work described in this paper.

The obtained results, as presented in Section 5.2, show that the developed in-browser rule parser and processor meet the requirements of the target application. The rule execution times are at least 100 times smaller than those measured in the real-world for the current cloud-based FGPE gamification service installation (which are, as a matter of fact, burdened by the lag introduced by network transmission). This means that implementing the new client-side rule parser and processor in the FGPE platform should improve rather than degrade the user experience of the students who use it (which was already positive [43]), even when ignoring its most important advantage: the new functionality of interacting with the platform without a sustained Internet connection.

The limitations to our findings are due to the fact that the tests were performed not in a real environment, but in an artificial one, and the tested rule set was also not a real one yet developed solely for testing purposes. Regarding the former, it must be noted that, in client-side execution, the times measured in an artificial test setting are not so different from the times measured in the real-world, as the client is used by only one user whereas the server serves many clients with different usage patterns, which impacts the speed with which it handles the requests. Regarding the latter, although we did not use a set of rules acquired from any particular gamified course, the test rules were inspired by the real rules we saw used in such courses. Unfortunately, as we forge a new path, there is no test benchmark against which we could compare our results.
7. Conclusions

In this paper, we adapted the gamification rule specification language defined in [12] for the purpose of a client-side rule parser and event processing engine, which we then implemented using Rust and compiled to WebAssembly for the sake of running it effectively in a web browser environment.

The performed tests involving a realistic number of rule definitions and a large number of processed events showed that both the initial parsing of the rule set defined in the chosen format and processing gamification-related events can be performed in time frames of length below the threshold that could cause a delay noticeable for end-users. Client-side processing, even though running on a considerably less powerful machine, is, in practice, faster than server-side processing: first, because there is no lag due to sending a request and receiving the response over the network for each generated event; second, because the client handles only the events of one user, whereas the server has to deal with events generated by many concurrent users.

The obtained solution is an essential step in our effort to develop a new FGPE-compatible web client that would operate in poor Internet accessibility conditions, conducted within the scope of the FGPE++ international project [42]. The envisaged future work consists of developing an interface, intelligent local cache manager, and synchronization engine so that the results of the gamification rules executed locally could be effectively transmitted to the server to be included in the global game state maintained there (and thus possibly becoming visible to other students and to the teacher of the student playing the game, if there is one) and trigger server-side gamification rules (e.g., dependent on the change of player’s position in the global leaderboard). Once the new FGPE client is complete, its full source code will be published under an open license at the dedicated GitHub repository (https://github.com/FGPE-Erasmus, accessed on 23 May 2024).

Author Contributions: Conceptualization, J.S.; methodology, J.S.; software, W.P.; validation, W.P.; formal analysis, J.S.; investigation, J.S.; resources, W.P. and J.S.; data curation, W.P.; writing—original draft preparation, J.S. and W.P.; writing—review and editing, J.S.; visualization, W.P. and J.S.; supervision, J.S.; project administration, J.S.; funding acquisition, J.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was co-funded by the European Union, grant number 2023-1-PL01-KA220-HED-00016496.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

5. González-Limón, M.; Rodríguez-Ramos, A. Cloud Gamification: Bibliometric Analysis and Research Advances. Information 2022, 13, 579. [CrossRef]


11. Paiva, J.C.; Queirós, R.; Leal, J.P.; Swacha, J.; Miernik, F. Managing gamified programming courses with the FGPE Platform. Information 2022, 13, 45. [CrossRef]


**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.