information

Article

Fostering Programming Practice through Games [†]

José Carlos Paiva [1,2,*], José Paulo Leal [1,2] and Ricardo Queirós [1,3]

1 Center for Research in Advanced Computing Systems (CRACS)–Institute for Systems and Computer (INESC) Porto LA, 4169-007 Porto, Portugal; zp@dcc.fc.up.pt (J.P.L.); ricardoqueiros@esmad.ipp.pt (R.Q.)
2 Department of Computer Science, Faculty of Sciences, University of Porto, 4169-007 Porto, Portugal
3 Unidade de Investigação em Media Artes e Design (uniMAD)–Escola Superior de Media Artes e Design (ESMAD), Polytechnic of Porto, 4480-876 Vila do Conde, Portugal
* Correspondence: jose.c.paiva@inesctec.pt

† This paper is an extended version of our paper published in Proceedings of the First International Computer Programming Education Conference (ICPEC 2020), Porto, Portugal, 23–24 April 2020.

Received: 29 September 2020; Accepted: 22 October 2020; Published: 24 October 2020

Abstract: Loss of motivation is one of the most prominent concerns in programming education as it negatively impacts time dedicated to practice, which is crucial for novice programmers. Of the distinct techniques introduced in the literature to engage students, gamification, is likely the most widely explored and fruitful. Game elements that intrinsically motivate students, such as graphical feedback and game-thinking, reveal more reliable long-term positive effects, but those involve significant development effort. This paper proposes a game-based assessment environment for programming challenges, built on top of a specialized framework, in which students develop a program to control the player, henceforth called Software Agent (SA). During the coding phase, students can resort to the graphical feedback demonstrating how the game unfolds to improve their programs and complete the proposed tasks. This environment also promotes competition through competitive evaluation and tournaments among SAs, optionally organized at the end by the teacher. Moreover, the validation of the effectiveness of Asura in increasing undergraduate students' motivation and, consequently, the practice of programming is reported.

Keywords: gamification; automatic assessment; programming learning; games; graphical feedback; tournament

1. Introduction

Motivation is what triggers and supports the learning process. Undergraduate students enroll in a new course eager to know more about or start a career in a specific area. Nevertheless, this primary stimulus does not last too long and typically vanishes after the initial difficulties and failures. This loss of motivation is arguably the most prominent concern in programming education nowadays since learning to program is difficult [1] and novice programmers are daunted by the complexity of the domain, which requires a correct understanding of abstract concepts as well as the development of practical skills, such as logical thinking, problem-solving, and debugging [2,3]. These knowledge and skills are acquired through extensive practice.

Unfortunately, the amount of practice time in programming classes is inadequate for students to assimilate such knowledge. As an example, a regular sixteen-week semester introductory programming course often does not include more than twelve programming assignments [4]. This quantity is already tremendous for teachers to be able to give sufficient individualized feedback but short for novices to master programming skills. Consequently, teachers often rely on automated assessment tools to deliver instant, teacherless, error-free, and effortless feedback to students on attempts to solve exercises [5].

Automated assessment tools can offer both static, which involves the compilation of the source code, and dynamic program analysis, which runs the compiled source code and checks if it produces the correct results [6]. The value and advantages of such tools in programming learning are supported by several published research studies [7,8]. Nevertheless, it is still crucial to ensure that there are enough sources of motivation for learners to make the decision to practice, apply the necessary amount of effort, and persist after repetitive failures [9,10].

A widespread and encouraging approach to foster students' motivation is gamification [11,12], which involves the use of game-thinking and game mechanics to engage people [13]. Even though gamification is not a panacea and can be inefficient or even affect students negatively [14], challenges including game elements that motivate intrinsically, such as graphical feedback and game-thinking, are more likely to have long-term positive effects on students [15]. In the specific case of automated assessment of programming assignments, it is possible to tune dynamic analysis of programs to transform the assessment of a program into a game, as the successful games CodeRally [16] and RoboCode [17] illustrate.

CodeRally and RoboCode are games in which players are Software Agents (SAs), developed by practitioners, that compete against each other in a 2D environment visible to the authors. These games have the purpose to teach object-oriented programming concepts. Some teachers introduced them in programming classes with quite a lot of success [18], but only in certain topics, such as the complexity and costs of creating or adapting this kind of game undermines their adoption for teaching other concepts than those they were designed for.

This paper presents Asura, an automated assessment environment for game-based coding challenges, which aims to foster programming practice by delivering engaging programming assignments to students. These assignments challenge students to code a program that acts as a game player (i.e., an SA) providing graphical feedback with the game simulation. These challenges can optionally promote competition among all submitted SAs, in the form of tournaments such as those found on traditional games and sports, including knockout and group formats. The outcome of the tournament is presented on an interactive viewer, which allows learners to navigate through each match and the rankings. From the teachers' perspective, Asura provides a framework and a command-line interface (CLI) tool to support the development of challenges. The validation of this authoring framework has been already conducted and described in a previous empirical study [19]. Hence, this paper focuses on the learners' perspective of Asura.

This paper is an extended version of the work published in [20]. We extend our previous work by adding a much broader review of the state of the art, following the Asura description with a simple example and some further details, and appending an analysis by type of player to the validation.

The remainder of this paper is organized as follows. Section 2 reviews the state of the art on platforms and tools that make use of challenges similar to those of Asura to engage programming learners. Section 3 provides an in-depth overview of Asura, including details on its architecture, design, and implementation. Section 4 describes its validation regarding the growth in motivation and, consequently, practice time. Finally, Section 5 summarizes the main contributions of this research.

2. State of the Art

Learning and teaching programming is hard [1,21]. When students keep struggling for too long, which is common in programming and particularly frequent during the initial stages, they are likely to give up. This is a huge concern among educators in this area and has led to a large body of research, not only to seek new forms of introducing students to programming, but also to boost the interest and motivation of people in coding. Among the most successful approaches are the use of competition and other game-like features in web learning environments to stimulate learners in solving more programming activities [22–24].

Asura uses game elements, particularly the assessment of programming exercises following competition formats and game-like graphical feedback of the behavior of the code, to create a more engaging programming-learning environment. Furthermore, it is embedded in Enki [22] through a special activity mechanism, which enables instructors to also deliver gamified educational contents along with Asura challenges. To the best of authors' knowledge, there are already learning environments that provide some of these features, but none of them includes automatic and competition-based assessment, game-like graphical feedback, and gamified educational content presentation, while supporting educators themselves to create their own courses with activities leveraging all these features.

This section describes some systems that support open online programming courses either without gamification or using only game elements that foster extrinsic motivation, programming learning platforms that stand out for their competitive side, and platforms and games in which learners' code takes action in a game world.

2.1. Platforms for Open Online Programming Courses

The panoply of open online programming courses available on the web is huge as there is a number of platforms supporting them. Some platforms, such as Coursera, Udacity, and edX, offer a wide variety of learning material from top universities' courses, with the possibility, typically at a monetary cost, of getting a verified certificate. Other platforms have less formal courses with a broad range of activities to provide hands-on experience on a specific language or framework (e.g., Codecademy, Khan Academy, Free Code Camp).

The former platforms are designed for people who already have some previous background in the subjects. They aim to improve the knowledge of people in areas that they are already interested in (intrinsic motivation). These platforms are only a means for people to have access to materials that were otherwise difficult to get. So, there is a small or no gamification factor in them. Moreover, creating courses for platforms like Coursera is an arduous task, requiring the agreement and involvement of third parties (e.g., universities).

In less formal open online courses, providers typically use gamification to attract learners, adopting elements such as progress indicators, badges, levels, leaderboards, and experience points. For example, Khan Academy uses badges and progress tracking to engage students to enlist and complete courses. Even that it also allows educators to easily create courses for their learners, it is limited to JavaScript-based programming activities (HTML and CSS are also supported). Codecademy also uses progress indicators and badges. It supports programming languages such as Python, JavaScript, Java, SQL, Bash/Shell, and Ruby. However, creating courses on Codecademy is not yet possible.

In the context of educational institutions, the most widely-used approach to create gamified open online courses is to rely on a Learning Management System (LMS) with game mechanics. LMSs were created to deliver course contents, and collect assignments of the students. However, many of them evolved to provide more engaging environments resorting to gamification. Some of the most notable examples are Academy LMS, Moodle, and Matrix which include badges, achievements, points, and leaderboards. Moodle also has several plugins that offer a variety of other gamification elements.

Another tool that serves this purpose is Peer 2 Peer University (P2PU) [25]. P2PU is a social computing platform that fosters peer-created and peer-led online learning environments. It allows students to join, complete, and quit challenges, and to enroll in courses. By completing learning tasks, courses, or challenges, they can earn badges, which are based on Mozilla Open Badges framework (www.openbadges.org).

However, LMSs and P2PU do not have out-of-the-box support for automatic assessment of programming exercises, which is a somewhat crucial feature for any programming learning environment. Thus, any of the three types of open online courses' providers described above have strengths, but lack the ability to easily create courses, a strategy to engage learners, the possibility to present different forms of learning materials, or the support for a multitude of programming languages. The same happens with other tools described in the literature [26,27].

From the research conducted, Enki [22] is the only tool that blends gamification, assessment and learning, presenting content in various forms as well as delivering programming assignments with automatic feedback, while allowing any stakeholder to create courses freely.

2.2. Competition-Based Programming Learning

Humans are naturally competitive. While learning activities too immersed in competition may produce adverse effects [28], it has proven to be an effective technique to stimulate students to exceed their capabilities [23,29–31]. For instance, programming contests are becoming more and more popular among learners [32]. This is also beneficial for students, who are increasingly facing programming contests after leaving universities, as part of the recruitment process for top technology companies. This section reviews some platforms and tools that use competition as a way to engage students in learning programming.

2.2.1. Automatic Judge Systems

Automatic judge systems are responsible for marking (i.e., assigning a grade to) programming assignments, by comparing the obtained output with the expected output [33]. These systems made it possible to realize programming competitions in a really competitive and engaging way. They have already been used, for more than fifteen years now, to manage programming contests all over the world, ensuring impartial and accurate timely feedback on program evaluation.

Programming contests are, by their definition, competitive activities where there are winners and losers. They are voluntary activities where individuals or teams strive to find solutions to previously unseen problems. Participants generally have a preparation phase where they learn several interesting strategies to understand and solve problems faster. As contests are managed by automatic judges, participants can make as many submissions as they want during competition, and learn from their mistakes. So, losers actually win knowledge which may compensate the negative effects typically associated, by many people, with competitive learning activities [34].

There are many automatic judges available in these days, such as DOMJudge [35], PC^2 [36], PKU JudgeOnline [37], and Mooshak [38]. Most of them born to support programming contests. However, the ability of these tools to support learning in programming classes was explored soon [23]. They are capable of providing immediate feedback to students inside and outside the classroom as well as stimulating, through timed challenges and leaderboards, their effort and persistence in solving activities [24].

2.2.2. HackerRank

HackerRank (https://www.hackerrank.com/) is a platform dedicated to hosting competitive programming challenges. This platform allows the user to code their solutions in more than 30 programming languages, test their code with custom input, and discuss problems with other peers. It has a practice mode where challenges are grouped by domains (e.g., Java, Javascript, Artificial Intelligence, among others), and a contest mode where challenges can be created and sponsored by any organization.

In the practice mode, there are leaderboards, badges, and points. Each domain has a leaderboard and a badge. A badge carries either one, two, three or four stars depending on the total points earned by a user. Typically, challenges proposed in contests are added to a practice domain, once the contest ends, according to the subjects that they cover.

The contest mode has various forms. There are contests that are opened indefinitely, in which challenges are proposed every week/day/month, and there are contests that run for a limited amount of time (e.g., 48 h). Contests have their own leaderboards and badges, that take into account the time spent by the users, since the beginning of the contest, to solve the problem. There is also a particular type of challenge where SAs developed by users run against each others, and the score is assigned based on the result (win, draw, or loss). Contests also enable users to increase or decrease their rating, earn medals (which are given to top solutions), and possibly win cash prizes or jobs at top technology companies.

2.3. Game-Based Programming Learning

Games are usually effective motivators. However, gamified environments are generally very different from games, and fail to use elements that are more prone to motivate users intrinsically such as graphical feedback and in-game challenges. Notwithstanding, there is a subset of gamification where learning is an outcome of playing a full-fledged game (formally known as serious games). For instance, in the context of programming learning, it is common to have games where code replaces the typical input controls. This subsection reviews both platforms where the learning activities are game-based programming challenges with graphical feedback, focusing on their support for authoring new challenges, and games that teach programming concepts.

2.3.1. Games for Teaching Programming Concepts

Serious games are already applied quite extensively in programming learning, either to explain complex concepts in engaging ways or to promote the practice of knowledge and skills harder to assimilate. For instance, Wu's Castle [39] aims to teach loops and arrays by presenting the learner with multiple-choice questions (MCQs) or fill-in blanks (FBs) where the feedback consists of displaying an avatar executing the code (e.g., building snowmen), whereas Robocode [17] challenges the student to develop a complete SA (SAD) that is capable of defeating all the opponents in the arena to practice object-oriented programming (OOP) and structured programming (SP). Some games can also be extended with new challenges or parts (AUT) by instructors who want to reuse it.

To better understand how games are being applied in programming learning, a literature research for serious games that teach one or more programming concepts has been conducted. From a vast collection of games, only those that address both the cognitive and emotional side of the students remained. Most of these games are designed in a way that the player starts by processing small pieces of information until they are both able to evaluate their progress and create solutions to similar problems on their own, following Bloom taxonomy hierarchy [40]. The motivation to keep evolving is mostly based on graphical feedback, well-defined challenges, competition (CMP) and/or collaboration (COL). In the end, 14 games were selected to analyze according to four components: concepts taught, programming languages used, proposed challenges, and key features. Table 1 summarizes the results.

Asura is designed to completely support the automated assessment of SAs on (and the development of) games like these (i.e., that teach and allow to practice specific programming concepts), except for 3D capabilities. Moreover, current experiments focus on studying the impact of a developed game as a learning tool, and not on their adoption as a technique for ongoing learning and practicing of concepts during a course.

Table 1. Overview of serious games to teach programming.

Game	Teaching Concepts	Language	Challenge	Key Features
Catacombs [41]	variables; conditionals; loops	micro-language	MCQs; FBs	3D; explanatory messages; scoreboard; code scaffolding
Saving Sera [41]	variables; conditionals; recursion; loops; quicksort	micro-language	MCQs; FBs; reorder code	code scaffolding; AUT
EleMental [42]	DFS; recursion	C#	avatar navigation; DFS	3D; interactive; explanatory messages;
Prog and Play [43]	variables; conditionals; loops; SP	Ada, C, Compalgo, Java, OCaml, Scratch,	SAD	3D multiplayer; AUT
Wu's Castle [39]	loops; arrays	C++	FBs; MCQs; avatar navigation	interactive
RoboZZle [44]	functions; recursion; conditionals	-	SAD (command stack)	AUT; scoreboard
LightBot [45]	functions; recursion; conditionals	-	SAD (command stack)	3D; AUT; scoreboard
TALENT [46]	variables; conditionals; loops	micro-language	drag-and-drop code; write code	explanatory messages; embedded IDE
RoboCode [17]	SP; OOP; IDE usage	Java	SAD	CMP; multiplayer;
Code Rally [16]	SP; OOP; IDE usage	Java; Node.js	SAD	CMP; multiplayer;
MUPPETS [47]	OOP	Java	SAD	3D multiplayer; COL/CMP; embedded IDE; AUT
PlayLogo 3D [48]	basic concepts	LOGO	SAD (command stack)	3D multiplayer; CMP;
Gidget [49]	basic concepts	micro-language	fix code	embedded IDE; AUT; explanatory messages
Leek Wars	structured programming; object-oriented programming;	JavaScript	write competitive SA; buy items and improve skills	embedded IDE; competitive; collaborative; skills and items; scoreboard

2.3.2. Greenfoot

Greenfoot [50] is an educational Integrated Development Environment (IDE) aimed at teaching object-oriented programming in Java or Stride to 14+ year old students. It supports the development of applications that use 2D animated graphics such as games and simulations by encapsulating the actual graphics code so that programmers can concentrate entirely on programming object behavior.

The purpose of Greenfoot is twofold. On the one hand, it aims to be a pedagogical integrated development environment for novice programmers to create applications with immediate 2D visual feedback, integrating the common tools of an IDE (e.g., editor, compiler, and virtual machine) and educational tools (e.g., interactive object invocation, inspection, and class visualization). On the other hand, it intends to provide a simulation and micro-world framework that offers a higher degree of interaction than existing micro-worlds such as turtle graphics or Gridworld.

To this end, the Graphical User Interface (GUI) of Greenfoot is composed of two windows: the interactive viewer (on the left) and the code editor (on the right). The interactive viewer window contains the Greenfoot world (i.e., the area where the program executes), which is variable in size and holds the scenario's objects, and a class diagram to visualize the classes used in this scenario and their inheritance relations.

2.3.3. CodinGame

CodinGame (http://codingame.com) is a web-based platform for programming learners to practice their coding skills by solving multiple puzzles. Many of these puzzles challenge the user to create a SA to manipulate a character in a game environment and provide graphical feedback to demonstrate the game simulation. To resolve the puzzle, the SA programmed by the player must pass all test cases (public and hidden). Players can select the programming languages of their choice among more than 20 available. Furthermore, once the exercise is solved, the player can view, rate, and vote on the best solutions.

Solving these game puzzles and collecting votes on submitted solutions adds to the user's leaderboards, rank, and badges. There are also periodical competitions in which a player can win physical prizes, either by beating SAs of other players in a match or by being the first to solve the problems. CodinGame also offers another mode, named Clash of Code, where players compete among them, for up to five minutes, to be the first to submit the best solution to an exercise. In this case, the exercises can be proposed by the community, but only text-based test cases are supported and no graphical content.

Although CodinGame has a wide range of challenges, it is not adequate for a classroom setting as its activities are dispersed, and there is no proper order for learning. In addition, it is a proprietary platform and does not allow external persons to manage content.

2.3.4. SoGaCo

SoGaCo (Social Gaming and Coding) [51] is a scalable cloud-based web environment that assesses competitive SAs built either in Java or Python. SoGaCo supports two-player board games, providing an interactive GUI that enables learners to watch step-by-step how their programs perform at each round.

The user interface of SoGaCo has two distinct views: one for editing code and another to test the agents. The latter contains three panels on the left with the user bots, built-in bots, and shared bots from other users, one panel on the center which displays the graphical feedback, and three panels on the right to control the step-by-step visualization of the graphical feedback, show the score, and write game captions respectively. The user starts coding the SA from a skeleton provided by the game author. During the development, he/she can test the bot against any bot present in one of the three bots' panels. After completing it, the single bot address (URL) can be shared with other peers to either compete against it or see its code.

SoGaCo's modular architecture supports various two-player board games, but there is no known process or specific format for creating games, nor for hosting competitions between existing SAs. However, it already includes a variety of board games such as PrimeGame, Mancala, Othello, and 5 in a Row.

3. Asura

Asura is an automated assessment environment for game-based programming challenges. It focuses on maximizing the time that students commit to programming activities by involving them with game elements that motivate intrinsically while minimizing the effort of teachers to a degree comparable to that of producing traditional programming problems. Therefore, it includes a framework and a CLI tool to support the authors through the process of developing challenges comprising unique features of games, such as graphical feedback, game-thinking, and competition.

This environment follows a multi-component architecture, presented in Figure 1, composed of three individual components, named Asura Viewer, Asura Builder, and Asura Tournament Manager, and a component designed to act as a plugin for an evaluation engine, the Asura Evaluator. Each of these components provides an interface to be consumed by other systems, except Asura Builder which produces a Java ARchive (JAR) of the challenge to be imported on the target system. This architecture aims to guarantee that every component is easily replaceable and extendable while allowing them to be integrated independently in different e-learning ecosystems.

An implementation of an ecosystem for Asura also requires the choice of tools both to play the role of the evaluation engine, in which Asura Evaluator integrates, and the learning environment that presents students with learning materials. For the implementation described in this paper, the tool selected for the first role could be almost any programming assessment module of one of the existing open-source automatic judge systems used in programming contests. Nevertheless, the selected system was capable of playing both roles, Mooshak 2 [38].

Mooshak is a web-based automatic judge system that supports assessment in computer science. It can evaluate programs written in several programming languages, such as Java, C, C++, C#, and Prolog. Besides that, Mooshak provides means to answer clarification requests of a problem, reevaluate programs, track printouts, among others. Although originally designed for managing programming contests, it eventually started being used in computer science classrooms to assist teachers in delivering timely and accurate feedback on programming assignments. As of version 2, Mooshak includes a pedagogical environment, named Enki [22]. Enki blends assessment (i.e., exercises) and learning (i.e., multimedia and textual resources) in a user interface designed to mimic the distinctive appearance of an IDE. It integrates with external services to provide gamification features and to sequence educational resources at different rhythms according to students' capabilities.

Figure 1 depicts the implemented architecture of Asura. Asura Evaluator is a small server-side package whose main class is the `AsuraAnalyzer`, a specialization of the main analyzer of Mooshak 2—`ProgramAnalyzer`—for conducting the dynamic analysis using the JAR provided by Asura Builder. Both `AsuraAnalyzer` and `ProgramAnalyzer`, which grades submissions to International Collegiate Programming Contest (ICPC)-like problems, implement a common interface—`Analyzer`—that allows evaluator consumers (i.e., Enki in this case) to integrate transparently with any of them. When running a tournament, Mooshak consumes the Asura Tournament Manager, a Java library providing an interface `Tournament` to organize tournaments. This interface receives a set of players and the structure of the tournament to carry out, and returns the matches to run on the Asura Evaluator, one after another. Either in games or in tournaments, the output of Asura Evaluator is received by Enki and displayed on Asura Viewer, an external Google Web Toolkit (GWT) widget that can integrate in any GWT environment. For that, it has an interface `Viewer` that enables consumers to feed it with a JSON object complying either with a match or tournament JSON schemas.

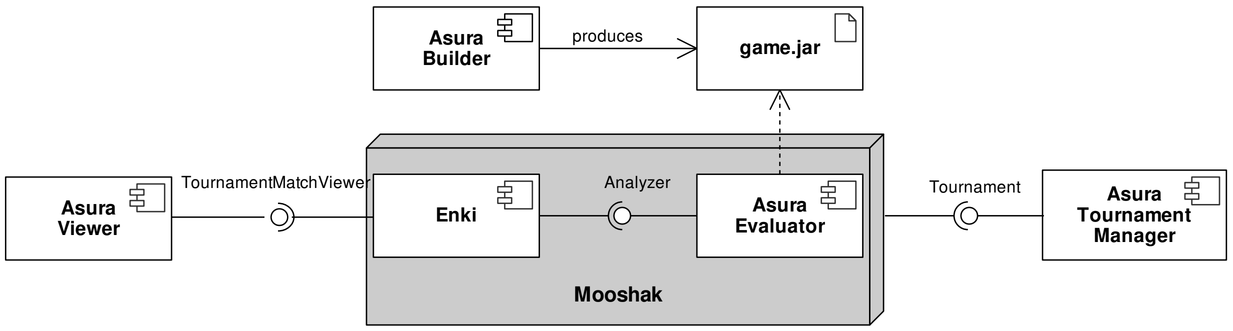


Figure 1. Architecture of the proposed ecosystem of Asura.

3.1. Tic Tac Toe

To ease the comprehension of each component of Asura as well as the interaction among them, the next subsections make use of a simple Asura challenge, named Tic Tac Toe, as the toy domain. Tic Tac Toe is an implementation of the famous 2-player game, coined with the same name, in which the players, X and O, take turns marking the spaces in a 3×3 grid. The winner is the player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row.

This challenge is to be applied in teaching the use of arrays, in Java, to undergraduate students in an *Introduction to Programming* course. To this end, the developed SAs must keep a matrix with the current grid state as they only receive the position (x, y) of the last opponent's choice and must, at least, always play in an empty position to pass the exercise.

3.2. Asura Builder

Asura Builder is a standalone component composed of multiple tools to support the authors through the whole process of creating game-based coding challenges. To this end, it includes both a CLI tool, to easily generate Asura challenges and install specific features (e.g., support for a particular programming language or a standard turn-based game manager), and a Java framework, that provides a game movie builder, a general game manager, several utilities to exchange complex state objects between the manager and the SAs as JSON or XML, and general wrappers for SAs in several programming languages.

The process of creating an Asura challenge starts with the scaffolding of a skeleton using the CLI tool. For instance, in the Tic Tac Toe example, the author would generate an Asura challenge from the Maven skeleton (there are also similar skeletons raw and a Gradle) by running the command in Listing 1 and answering the questions it poses pertaining to the challenge metadata. The skeleton contains a minimal working challenge, including an empty game state, a simple turn-based game manager, and a simple skeleton and a wrapper for Java-based SAs. Since the introduced challenge is targeted to Java, there is no need to manage programming languages. However, it is important to notice that, even though the authors are required to program the challenges in Java, players can use their preferred programming language to code the SAs. For that, authors can also manage support for programming languages through the CLI using the commands in Listing 2, which will add/remove skeletons and wrappers for the specified language.

Listing 1: Generate an Asura challenge from the Maven skeleton through the CLI

```
asura-cli create maven
```

Listing 2: Manage support for programming languages through the CLI

```
asura-cli add-language [language]
asura-cli remove-language [language]
```

After the scaffolding, the actual development phase takes place. The core of an Asura challenge is the game manager, which among many tasks ensures that the game rules are followed, decides which player takes the next turn, and declares the winner(s). The framework provides an abstract game manager containing all the generic code, such as the initial crossing of I/O streams, handlers for timeout and badly formatted input, and automatic (de)serialization of JSON or XML to Java objects. The game manager developed by the author extends this generic one, implementing the specificities of the game. For instance, the game manager of the Tic Tac Toe has to decide which player takes the Xs and Os (which can be done in a random fashion) and, then, query the SA playing with Xs and the one with the Os (i.e., send the last position filled and read the position to play) alternately, updating the game state with their move, until the winning condition verifies (i.e., three equal symbols in a row, column, or diagonal) or all slots of the grid are occupied.

Asura Builder clearly separates the game state from the game manager, defining a “contract” for the object that holds the game state which should be used by the game manager to maintain the state. Hence, the state of a specific game must, at least, implement four mutational methods—`prepare()`, `execute()`, `endRound()`, and `finalize()`—and two informational methods—`isRunning()` and `getStateUpdateFor()` (methods’ parameters omitted for the sake of simplicity). Mutations of the state should generally produce updates in the game movie, a concept introduced by the framework to facilitate the creation of graphics. A game movie consists of a set of frames, each of them containing a set of sprites together with information about their location and applied transformations, and metadata information. The movie should be produced with the support of a provided builder, which includes methods to construct each frame with ease as well as to deal with SA evaluation and premature game termination.

As for the Tic Tac Toe, the `prepare()` method sets the metadata on the game movie builder, particularly the title, the number of frames per second, the default size, the position of the sprite relative to which the objects' coordinates are given, the 3×3 board background, the X and O sprites, and the players. The `execute()` function handles the possible actions from the player. In this case, it supports an action `PLAY` that contains the position in which the player wants to play and an artificial action `SYMBOL` sent by the game manager to indicate who is X and who is O. The `getStateUpdateFor()` returns the update to send to a given player which, in this case, consists of the position where the last symbol has been placed. The `endRound()` generates the frame of the current round based on the previous frame, but adding the new symbol. The `isRunning()` method checks if the number of rounds has exceeded or there is a winning combination, in which case it is not running. The `finalize()` method checks who is the winner and sets the statuses of the players accordingly.

Moreover, the framework also allows us to define wrappers for the SAs. A wrapper consists of a set of methods that aim to give players a higher level of abstraction so that they can focus on solving the real challenge instead of spending time processing I/O or doing other unrelated tasks. They can also be used to increase or decrease the difficulty of the problem by changing the way that the SA interacts with the game, without modifying the game itself. For example, a wrapper for the Tic Tac Toe could define two methods to support the SA, `getLastPlayed(): int[]` which provides the x and y position of the last move and `play(x: int, y: int)` that sends the action to the game manager to play in the given position. Since there are actions common to players of any game (e.g., communication with the manager), those were included in global wrappers. Hence, authors of challenges will only develop game-specific wrappers if they intend to provide a different abstraction layer to the learners.

3.3. Asura Evaluator

The evaluation engine of Mooshak grades a submission according to a set of rules, following a black-box approach. At the same time, it produces a report of the evaluation for additional validation from a human judge (e.g., teacher), if required. The assessment process has two phases: static analysis, which tests the consistency of the SA source code and generates an executable program; and dynamic analysis, which includes running the program with each test case loaded with the problem and comparing its output to the expected output.

Asura Evaluator uses the static analysis from Mooshak as-is, merely appending the global and game-specific SA wrappers encapsulated in the game JAR file to the compile command-line. However, dynamic analysis is re-implemented from scratch, replacing test cases based on input and output text files with game simulations against opponents from a provided list of paths to submissions. The kinds of evaluation, either validation or submission, determine the origin of the contestants, in the case of multiplayer games. Validations are only a means for the learner to experiment with his/her SA in a match against any existing SAs as they do not count for evaluation purposes. The learner can select the opponents from a list containing all the last accepted SAs of the students as well as the control SAs included by the author. On the contrary, submissions count for evaluation and, thus, are evaluated under the same circumstances against the control SAs developed by the challenge author.

The dynamic analysis starts by combining the current submission and a distinct set of the selected opponents' submissions into matches, according to the minimum and the maximum number of players per match indicated on the game manager. As the opponents' SAs are always submissions previously compiled and accepted, the component only needs to start processes from the executable programs. Then, for each match, it initializes an instance of the specific game manager from the JAR, providing it with the list of player processes in the match indexed by the player ID. The game manager crosses the input and output streams of each of these processes with itself, enabling it to write state updates to the input stream of the SA and read actions from its output stream (i.e., from the SA perspective, they are typical I/O operations). Therefore, the game simulation is handed over to the game manager, which should maintain the SAs aware of the game state, query them for actions at the proper time, ensure that the game rules are not broken, manage the state of the game, and classify and grade submissions. Once all matches are completed, the statuses collected from the matches, comprising observations, marks, classifications, and feedback, are combined and added to the submission report.

In the Tic Tac Toe challenge, a submission or validation schedules a set of two-player matches between the submitted SA and either each of the control SAs or the selected opponents' SAs. Firstly, Asura Evaluator will compile the submitted SA's code. Then, for each game, Asura Evaluator initiates a process from the submitted SA's compiled code and another from the previously compiled opponent's code. Finally, it initializes the game manager passing both processes indexed by the players' IDs and handovers the evaluation to it. If, during any match, the submitted SA plays in a previously occupied position or the process fails for any other reason, the evaluation exits with the corresponding negative classification. Otherwise, the program is accepted and the mark is calculated.

3.4. Asura Tournament Manager

Asura Tournament Manager consists of a Java library to organize tournaments among SAs submitted and accepted on an Asura challenge. Tournaments aim to give a final goal to students by encouraging them to engage in a competition run at the end of the submission period, optionally. Hence, the challenge is not just about correctly solving the problem, but finding a strategy to win a final competition. To this end, the preparation phase allows students to do "friendly" matches (i.e., validations) against any previously approved SAs from each other, providing a glimpse of the performance expected in the tourney.

To arrange tournaments, instructors may use a wizard implanted in Mooshak's administration user interface. These tournaments follow the same structure as those conducted in traditional games and sports competitions. They can have multiple stages, each of them organized according to one of the available formats: round-robin, swiss system, single-elimination, or double-elimination. Stages have a series of rounds in which each contestant either participates in a single match or advances immediately to the next round in the absence of assigned opponents (i.e., has a bye). Lastly, matches are the smallest grain of a tournament, generated one after another, executed on the Asura Evaluator, and the result returned to the Asura Tournament Manager. The result of a match includes the points achieved by each player as well as additional features that can be used as tiebreakers, either for the match or rankings.

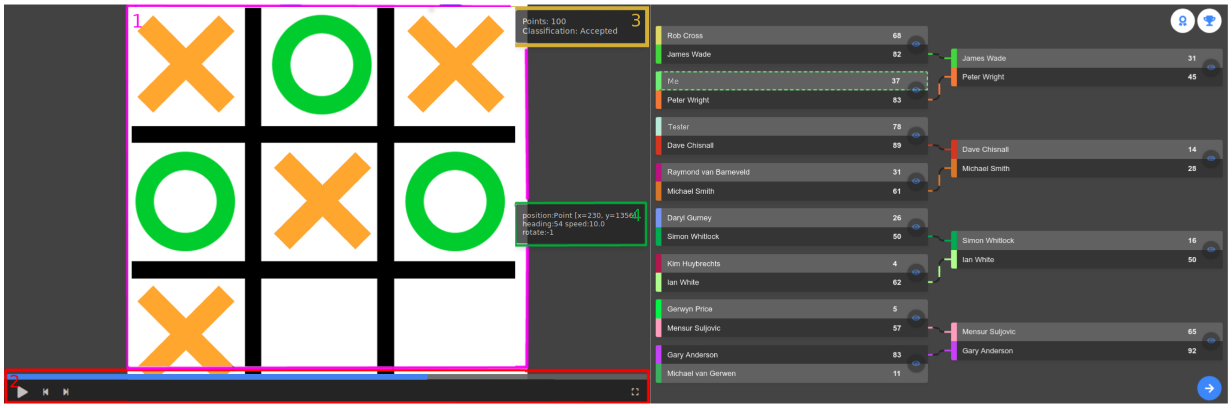
The interface `Tournament` offers a sequential way to interact with the library to organize a tournament. Firstly, the consumer provides the mandatory metadata, particularly the title, game, and participants. Then, it adds and configures the stages, starting them one after another. The configuration options available for a stage depend upon its format and may comprise the number of players per match, the minimum number of players per group, the number of qualified players, the maximum number of rounds, the kind of result (e.g., win–draw–loss or position-based), and tiebreakers both for rankings and matches. Once the stage has started, obtaining the next match activates the wait mode until its result is sent and processed. In the end, the output of the Asura Tournament Manager is a JSON object complying with the tournament JSON schema.

Although the Tic Tac Toe example has the goal of teaching the usage of arrays, the instructor could promote additional programming practice by scheduling a tournament to the end of the submissions' time, so that students would keep improving their SAs even after getting them accepted. Due to the nature of Tic Tac Toe (i.e., two excellent players will always draw), a single group-based format (e.g., round-robin) is adequate whereas elimination formats may fail to classify SAs.

3.5. Asura Viewer

Asura Viewer has the responsibility of presenting the graphical feedback to learners for both single matches and tournaments. Hence, it consists of a Google Web Toolkit (GWT) widget with two distinct modes, particularly match and tournament, that swap seamlessly according to the schema to which the provided JSON adheres.

The match mode displays the game movie described by the JSON produced during the assessment of both submissions and validations, which simulates the game unfolding as if the student is playing the game. In this case, the widget adopts the look and functionality of a media player, as depicted on the left of Figure 2, which includes a control toolbar (red) with a slider to change the time position of the movie and buttons to play, stop, enter full-screen, and navigate through the current playlist; a canvas where the frames are drawn (pink); a box to exhibit the current status (yellow); and a box to present debugging messages of the SA (green).



```
Points: 100
Classification: Accepted
```

```
position:Point [x=230, y=1356]
heading:54 speed:10.0
rotate:-1
```

Rob Cross	68
James Wade	82
Me	37
Peter Wright	83
Tester	78
Dave Chisnall	89
Raymond van Barneveld	31
Michael Smith	61
Daryl Gurney	26
Simon Whitlock	50
Kim Huybrechts	4
Ian White	62
Gerwyn Price	5
Mensur Suljovic	57
Gary Anderson	83
Michael van Gerwen	11

James Wade	31
Peter Wright	45
Dave Chisnall	14
Michael Smith	28
Simon Whitlock	16
Ian White	50
Mensur Suljovic	65
Gary Anderson	92

Figure 2. Asura viewer modes. On the left, the match mode with Tic Tac Toe game (distinct areas highlighted in different colors). On the right, the brackets view of the tournament mode.

The tournament mode offers an interactive view of the complete tournament, allowing users to navigate through the various stages and visualize specific matches or the whole course of a player as well as the rankings of the stage. The GUI of the widget when this mode is active, depicted on the right of Figure 2, holds a previous/next stage navigation menu on the bottom right; a contextual menu, to change between ranks' view (either final or relative to the current stage) and brackets view, on the top right corner; and the central area in which the content appears.

4. Validation

The experiment conducted to assess the effectiveness of Asura in enhancing students' motivation and increasing practice time, as well as its acceptance by learners, took the form of an open online learning course with a duration of two weeks, free of charge, and without participation limits. The course aims to teach the new concepts introduced in version ES6 of JavaScript, and it has been announced to undergraduate students who participated in the Web Technologies classes of the previous semester, which provided them with some background on JavaScript without ES6 features.

From the invited students, 10 (1 female) registered in the course, who were randomly assigned to either of the equal-sized groups: control (1 female) or treatment. Both groups received the same learning materials consisting of lecture notes about the concepts of ES6, including topics such as the variable declaration, object and array destructuring, arrow functions, promises, and classes, and a consolidated ES6 cheat sheet. However, they differ in the programming assignments delivered for training: Asura challenges for the treatment group and ICPC-like problems for the control group. After finishing the course curriculum, both groups were invited to take an exam composed of ICPC-like exercises to measure the skills attained.

The next subsections describe the Asura challenges created for the course, the analysis conducted, its results and limitations. Section 4.1 details the Asura challenges created to teach ES6. Section 4.2 describes the analysis and results. Section 4.3 summarizes the limitations of this study.

4.1. War of Asteroids

The challenges proposed in the treatment group are all chapters of the War of Asteroids. The War of Asteroids is based on Asteroids from Atari Inc, an arcade space shooter released in November 1979 and one of the first major hits of the golden age of arcade games. The goal of Asteroids is to destroy asteroids and enemies (also known as saucers) to grant score points. In the game, the player navigates a triangular ship, initially placed at the center, that can turn left and right, fire projectiles, and thrust forward. Once the vehicle is propelled in a direction, it continues in that direction unless the player applies thrust in a different direction. The graphics consist of simple white lines on a dark background.

The world of Asteroids is a cyclic grid, meaning that objects moving out of the grid disappear and reappear on the opposite position of the screen. The world is populated with asteroids, moving at a constant direction and speed, and saucers, which can change direction and speed. Both can only be destroyed with shots. However, large asteroids first break into medium asteroids, which can, in turn, be blasted into small rocks, that will finally disappear.

War of Asteroids keeps most of the original gameplay of Asteroids, yet adds a number of features; improves graphics (see Figure 3); and replaces the input controls with a program. It is now played by up to four contestants, which replace the saucers. Furthermore, the ships can execute two energy-consuming actions: activate an energy shield and throw bombs. The game ends after 10,000 units of time (frames) or when there is only a single survivor. The methods of earning points, as well as the number of points awarded per accomplishment, were also revised. Players award points for each hit in an asteroid or ship; destroying an asteroid or a ship; activating the protection successfully against a bullet or an asteroid; and remaining health points at the end.

The API defined by the game-specific wrapper allows to control the ship with seven distinct commands: `thrust()` to thrust the ship; `steerLeft()` to rotate the heading of the ship by $-4°$; `steerRight()` to rotate it by $4°$; `shield()` to activate the shield; `firePrimary()` to fire a bullet; `fireSecondary()` to throw a bomb; and `log(message)` to log a message.

Asura Builder allowed producing the entire game movie in about 25 instructions, including animations for explosions, bullets' hits, and rotating asteroids. The movie has been recorded at 60 frames-per-second where 3 minutes weighs approximately 5 MB without compression and 51 KB if compressed with Gzip (http://gnu.org/software/gzip/).

For the validation, the game has been divided into four chapters, each introducing both a new concept of the ES6 and a new feature of the game. The first three chapters are only introductory solo missions, whereas the final chapter expects students to create a complete competitive SA to beat their opponents.

4.2. Results and Analysis

The data obtained from the experiment includes logs automatically registered by Mooshak 2 from the system usage and questionnaire responses. Usage data comprises data about several metrics, in particular, the number of submissions and validations, timestamp of the activity, and results of submitted attempts. The questionnaire follows Lund's model [52], providing data on Usefulness, Ease of Use, Ease of Learning, and Satisfaction gathered from questions with a 7-value Likert scale as well as identifying Asura weaknesses and strengths through free-text questions. As the questionnaire is delivered along with the exam, it is guaranteed that only students who finished the course and do the exam may answer it, which was not the case with two students (one from each group).

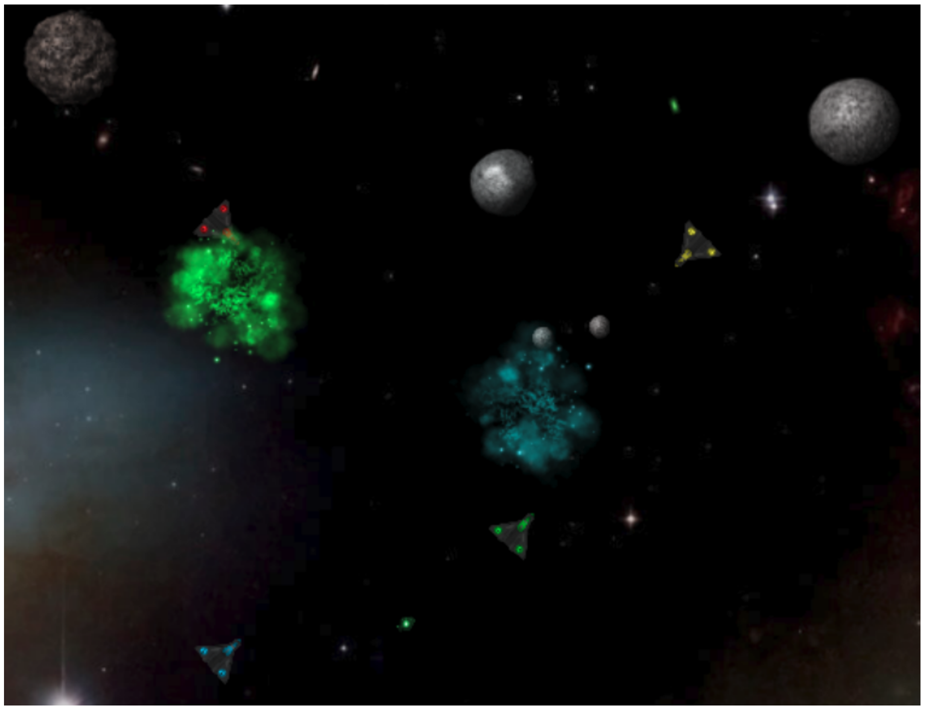


Figure 3. War of Asteroids. Screenshot of a game with four ships competing against each other.

The outcome from the questionnaire is presented in Figure 4 separated by group: control on the left and treatment on the right. These results expose positive advancements in all measured metrics (e.g., usefulness in the control group had an average of 3.86 against 5.50 the treatment group while satisfaction valued 3.46 versus 5.43). The quality of the feedback is perhaps the principal reason for these discrepancies as some students in the control group indicated feedback as a weakness (e.g., "Observations and program input/output could be improved" and "Feedback messages are scarce") in opposition to the treatment group which appreciated the graphical feedback in the free-text questions. Yet, there were complains about the complexity of understanding the game mechanics in parallel with ES6 (e.g., "The idea is good, but when it is used to learn JavaScript since the start, it can be confusing because you have to learn JavaScript, learn the game, and think about the two"), as well as unanimous critics about the user interface of Enki (e.g., "misses terminal and debugger" or "looks bad on my MacBook Air 13").

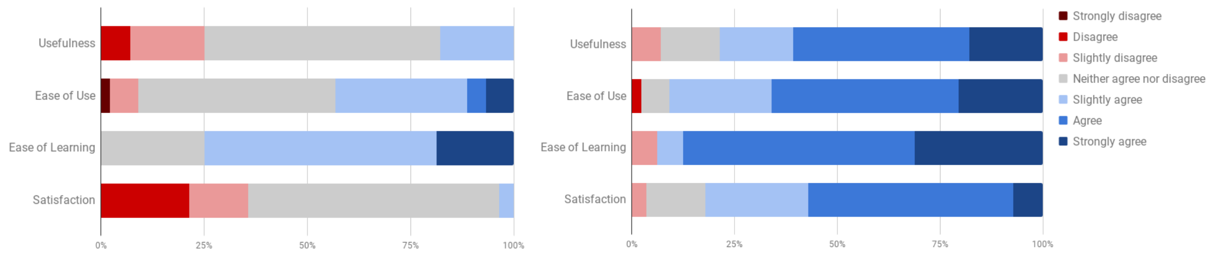


Figure 4. Results of the usefulness, ease of use, ease of learning, and satisfaction questionnaire of Asura: the control group (on the **left**) and the treatment group (on the **right**).

The actual increase in practice time and its corresponding impact on knowledge acquisition and retention is measured from six variables extracted from usage data: number of validations, number of submissions, return days (i.e., count of different days in which a student submitted an attempt), the ratio of solved exercises (i.e., a value of 1 means all activities of the course solved while 0 means no solved activities), and exam score (i.e., an integer between 0 and 3—one point per exercise solved), and group. Figure 5 presents boxplots comparing the number of attempts (derived from the sum of submissions and validations), return days, the ratio of solved exercises, and exam scores of both groups.

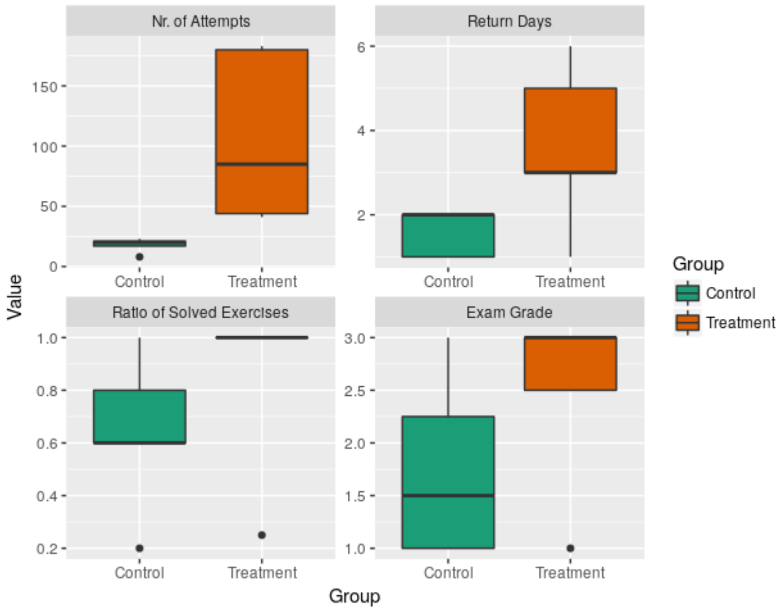


Figure 5. Quantitative comparison of metrics per group: number of attempts, return days, ratio of solved exercises, and exam score.

The number of submissions reveals a considerable discrepancy between the treatment group (370 of which 65 accepted) and the control group (44 of which 16 succeeded), further increased by the differences in the ratio of exercises solved and return days. This discrepancy indicates that students struggled a lot in the treatment group to overcome their difficulties. Moreover, there were 40+ submissions than necessary (i.e., after having an acceptable solution), all of them in the chapter with competition. The validations also reflect these trends, even though students only realized 15 friendly matches against the opponents. The exam shows that 75% of the students in treatment performed well (grade between 2 and 3) against 50% in control.

Students in the treatment group were also invited to fill in a survey to evaluate their gamer personality. This survey follows the Bartle Test [53] to classify players of multiplayer online games into categories based on Bartle's taxonomy of player types (i.e., achiever, explorer, killer, and socializer) [54]. It estimates the percentage that the player's behavior relates to the default behavior of each type of player. The main objective of this analysis was to detect possible correlations between the personality of the learner as a gamer and his/her behavior during the experiment. To this end, four variables were selected: number of submissions, number of validations, number of attempts, and return days. The resulting plots of Spearman's correlation for the distinct taxonomies are presented in Figure 6, annotated with Spearman's correlation coefficient, *r*, and *p*-value. For instance, it is possible to identify a positive monotonic relationship between the killer's ratio and each of the measured values as well as a negative monotonic relationship between the explorer's ratio and the number of validations and return days. Unfortunately, only three learners have completed this test which makes the results statistically insignificant.

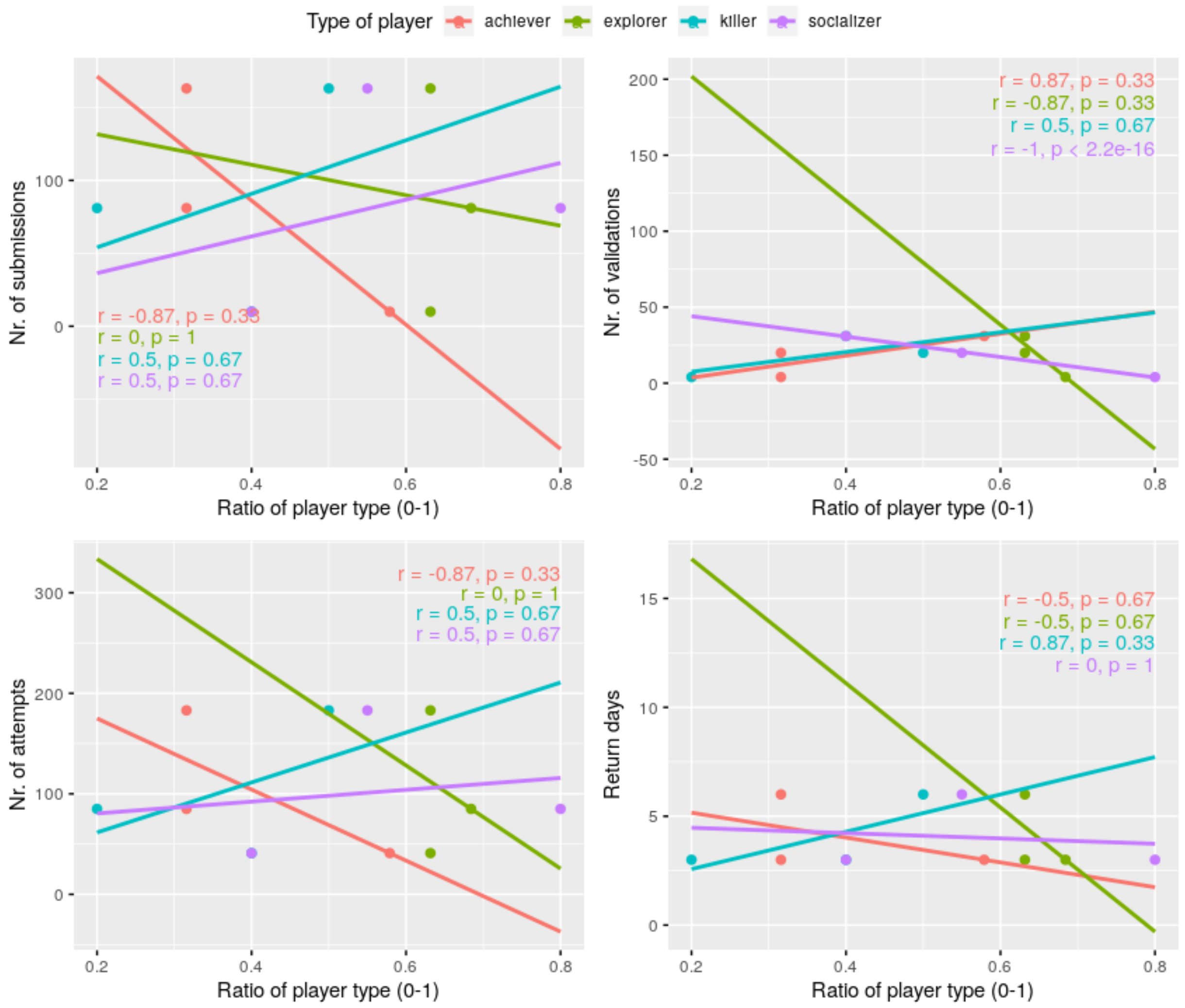


Figure 6. Spearman's correlation between the different player types and the number of submissions, number of validations, total attempts, and return days.

4.3. Limitations

The most significant limitation of this study is the small sample size (10), which prevents drawing reliable conclusions and identifying meaningful relationships from the data (e.g., whether the type of player correlates with individual behaviors during the course). Even though the changes are noticeable (e.g., Figure 4 and 5), they are based on only four students per group, which is insufficient to assert whether differences relate to personal aspects (accentuated by the small size) or "treatment" success. Furthermore, the sample is not representative of the population (i.e., undergraduate programming students), as there is a single female student and all students have at least one and a half years of programming background from the university.

Unfortunately, the experiment could only be conducted during the Spring semester exams' time what partially explains the lack of availability of the target group. The targeted group was already too limited (about 40) at the start due to the requirement of one semester of educational experience in JavaScript to be able to enroll in the course.

5. Conclusions

This paper introduces Asura, an automated assessment environment for game-based coding challenges, which seeks to promote students' motivation, requiring teachers to make an effort comparable to that of creating traditional programming problems. This environment offers teachers a set of tools to author game-based programming challenges in which learners code an SA to play a game. These challenges incorporate competition in the form of tournaments, similar to those used in traditional games and sports, to stimulate further efforts to find better strategies that can defeat all the opponents.

The validation of Asura demonstrated relative success in motivating a small sample of students to overcome their difficulties through practice. Unfortunately, the sample size is too reduced to conclude its effectiveness. Most students of the control group complained about the amount and quality of feedback given, whereas students of the treatment group found it troublesome to get started with the proposed game, the War of Asteroids, while also learning new concepts of ES6. The GUI of Enki has been criticized by both groups with some participants indicating a few issues.

As Asura proposes a new format of challenges, the next step is to grow a repository of challenges that teachers can rely upon to apply to their classes. These challenges should have similar pedagogical purposes to those existing in other formats.

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

Funding: This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CLI	Command-Line Interface
ES6	EcmaScript 6
GUI	Graphical User Interface
GWT	Google Web Toolkit
ICPC	International Collegiate Programming Contest
IDE	Integrated Development Environment
JAR	Java ARchive
JSON	JavaScript Object Notation
LMS	Learning Management System
SA	Software Agent
XML	eXtensible Markup Language

References

1. Robins, A.; Rountree, J.; Rountree, N. Learning and teaching programming: A review and discussion. *Comput. Sci. Educ.* **2003**, *13*, 137–172. [CrossRef]
2. Piteira, M.; Costa, C. Learning Computer Programming: Study of Difficulties in Learning Programming. In Proceedings of the 2013 International Conference on Information Systems and Design of Communication, Lisbon, Portugal, 11 July 2013; pp. 75–80. [CrossRef]
3. Moser, R. A Fantasy Adventure Game As a Learning Environment: Why Learning to Program is So Difficult and What Can Be Done About It. In Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education, TUppsala, Sweden, 1–5 June 1997; pp. 114–116. [CrossRef]
4. Beaubouef, T.; Mason, J. Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations. *ACM Sigcse* **2005**, *37*, 103–106. [CrossRef]
5. Cheang, B.; Kurnia, A.; Lim, A.; Oon, W.C. On automated grading of programming assignments in an academic institution. *Comput. Educ.* **2003**, *41*, 121–131. [CrossRef]
6. Ala-Mutka, K.M. A Survey of Automated Assessment Approaches for Programming Assignments. *Comput. Sci. Educ.* **2005**, *15*, 83–102. [CrossRef]
7. Enström, E.; Kreitz, G.; Niemelä, F.; Söderman, P.; Kann, V. Five years with kattis–Using an automated assessment system in teaching. In Proceedings of the 2011 Frontiers in Education Conference (FIE), Rapid City, SD, USA, 12–15 October 2011; pp. T3J–1–T3J–6. [CrossRef]
8. Edwards, S.H. Improving Student Performance by Evaluating How Well Students Test Their Own Programs. *J. Educ. Resour. Comput.* **2003**, *3*, 1-es. [CrossRef]
9. Law, K.M.Y.; Lee, V.C.S.; Yu, Y.T. Learning motivation in e-learning facilitated computer programming courses. *Comput. Educ.* **2010**, *55*, 218–228. [CrossRef]
10. Pintrich, P.R. The role of motivation in promoting and sustaining self-regulated learning. *Int. J. Educ. Res.* **1999**, *31*, 459–470. [CrossRef]
11. Ibáñez, M.B.; Di-Serio, A.; Delgado-Kloos, C. Gamification for engaging computer science students in learning activities: A case study. *IEEE Trans. Learn. Technol.* **2014**, *7*, 291–301. [CrossRef]
12. Buckley, P.; Doyle, E. Gamification and student motivation. *Interact. Learn. Environ.* **2016**, *24*, 1162–1175. [CrossRef]
13. Zichermann, G.; Cunningham, C. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2011.
14. Toda, A.M.; Valle, P.H.; Isotani, S. The dark side of gamification: An overview of negative effects of gamification in education. In *Higher Education for All. From Challenges to Novel Technology-Enhanced Solutions*; Cristea, A., Bittencourt, I., Lima, F., Eds.; Springer: Cham, Switherland, 2017.
15. Chee, C.M.; Wong, D.H.T. Affluent Gaming Experience Could Fail Gamification in Education: A Review. *IETE Tech. Rev.* **2017**, *34*, 593–597. [CrossRef]
16. IBM Community. CodeRally. 2013. Available online: http://www.alphaworks.ibm.com/tech/coderally (accessed on 1 November 2019).
17. Hartness, K. Robocode: Using Games to Teach Artificial Intelligence. *J. Comput. Sci. Coll.* **2004**, *19*, 287–291.
18. Bonakdarian, E.; White, L. Robocode throughout the Curriculum. *J. Comput. Sci. Coll.* **2004**, *19*, 311–313.
19. Paiva, J.C.; Leal, J.P.; Queirós, R. Authoring Game-Based Programming Challenges to Improve Students' Motivation. In *The Challenges of the Digital Transformation in Education*; Auer, M.E., Tsiatsos, T., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 602–613.
20. Paiva, J.C.; Leal, J.P.; Queirós, R. Game-Based Coding Challenges to Foster Programming Practice. In *First International Computer Programming Education Conference (ICPEC 2020)*; Queirós, R., Portela, F., Pinto, M., Simões, A., Eds.; Schloss Dagstuhl–Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2020; Volume 81, pp. 1–11. [CrossRef]
21. Jenkins, T. On the difficulty of learning to program. In Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, Loughborough, UK, 27–29 August 2002.

22. Paiva, J.C.; Leal, J.P.; Queirós, R.A. Enki: A Pedagogical Services Aggregator for Learning Programming Languages. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, Arequipa, Peru, 25–26 April 2016, pp. 332–337. [CrossRef]
23. Leal, J.P.; Silva, F. Using Mooshak as a Competitive Learning Tool. *Compet. Learn. Symp.* **2008**.
24. Guerreiro, P.; Georgouli, K. Enhancing elementary programming courses using e-learning with a competitive attitude. *Int. J. Int. Educ.* **2008**, *10*, 38.
25. Ahn, J.; Butler, B.S.; Alam, A.; Webster, S.A. Learner participation and engagement in open online courses: Insights from the Peer 2 Peer University. *J. Online. Learn. Teach.* **2013**, *9*, 160.
26. Gogoulou, A.; Gouli, E.; Grigoriadou, M.; Samarakou, M.; Chinou, D. A web-based educational setting supporting individualized learning, collaborative learning and assessment. *J. Educ. Technol. Soc.* **2007**, *10*, 242–256.
27. Hwang, W.Y.; Wang, C.Y.; Hwang, G.J.; Huang, Y.M.; Huang, S. A web-based programming learning environment to support cognitive development. *Interact. Comput.* **2008**, *20*, 524–534. [CrossRef]
28. Kohn, A. *No Contest: The Case against Competition*, 2nd ed.; Houghton Mifflin: Boston, MA, USA, 1992.
29. Chang, L.J.; Yang, J.C.; Yu, F.Y. Development and evaluation of multiple competitive activities in a synchronous quiz game system. *Innov. Educ. Teach. Int.* **2003**, *40*, 16–26. [CrossRef]
30. Lawrence, R. Teaching data structures using competitive games. *IEEE Trans. Educ.* **2004**, *47*, 459–466. [CrossRef]
31. Burguillo, J.C. Using game theory and competition-based learning to stimulate student motivation and performance. *Comput. Educ.* **2010**, *55*, 566–575. [CrossRef]
32. Trotman, A.; Handley, C. Programming contest strategy. *Comput. Educ.* **2008**, *50*, 821–837. [CrossRef]
33. Georgouli, K.; Guerreiro, P. Incorporating an automatic judge into blended learning programming activities. In Proceedings of the International Conference on Web-Based Learning, Magdeburg, Germany, 23–25 September 2019; pp. 81–90. [CrossRef]
34. Revilla, M.A.; Manzoor, S.; Liu, R. Competitive learning in informatics: The UVa online judge experience. *Olymp. Inf.* **2008**, *2*, 131–148.
35. Eldering, J.; Kinkhorst, T.; van de Warken, P. DOM Judge–Programming Contest Jury System. 2020. Available online: https://www.domjudge.org (accessed on 23 October 2020).
36. Ashoo, S.E.; Boudreau, T.; Lane, D.A. CSUS Programming Contest Control System (PC2). 2019. Available online: https://pc2.ecs.csus.edu (accessed on 23 October 2020).
37. Pengcheng, X.; Fuchen, Y.; Di, X. PKU JudgeOnline. 2013. Available online: http://poj.org (accessed on 23 October 2020).
38. Leal, J.P.; Silva, F. Mooshak: A Web-based multi-site programming contest system. *Softw. Pract. Exp.* **2003**, *33*, 567–581. [CrossRef]
39. Eagle, M.; Barnes, T. Wu's castle: teaching arrays and loops in a game. In Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education, New York, NY, USA, 2008; pp. 245–249. [CrossRef]
40. Bloom, B.S.; Max D.E.; Edward J.F.; Walker, H.H.; David R.K. *Taxonomy of Educational Objectives*; Addison-Wesley Longman Ltd.: Boston, MA, USA, 1956.
41. Barnes, T.; Richter, H.; Powell, E.; Chaffin, A.; Godwin, A. Game2Learn: Building CS1 Learning Games for Retention. In Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, New York, NY, USA, 2007; pp. 121–125. [CrossRef]
42. Chaffin, A.; Doran, K.; Hicks, D.; Barnes, T. Experimental Evaluation of Teaching Recursion in a Video Game. In Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games, New York, NY, USA, 2009; pp. 79–86. [CrossRef]
43. Muratet, M.; Torguet, P.; Viallet, F.; Jessel, J.P. Experimental Feedback on Prog&Play: A Serious Game for Programming Practice. *Comput. Gr. Forum* **2010**, *30*, 61–73. [CrossRef]
44. Ostrovsky, I. RoboZZle. 2009. Available online: http://robozzle.com (accessed on 23 October 2020).

45. Aedo Lopez, M.; Vidal Duarte, E.; Castro Gutierrez, E.; Paz Valderrama, A. Teaching Abstraction, Function and Reuse in the First Class of CS1: A Lightbot Experience. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, New York, NY, USA, 2016; pp. 256–257. [CrossRef]
46. Maragos, K.; Grigoriadou, M. Exploiting TALENT as a Tool for Teaching and Learning. *Int. J. Learn.* **2011**, *18*, 1.
47. Phelps, A.M.; Bierre, K.J.; Parks, D.M. MUPPETS: Multi-user Programming Pedagogy for Enhancing Traditional Study. In Proceedings of the 4th Conference on Information Technology Curriculum, Lafayette, IN, USA, 16–18 October 2003; pp. 100–105. [CrossRef]
48. Paliokas, I.; Arapidis, C.; Mpimpitsos, M. PlayLOGO 3D: A 3D Interactive Video Game for Early Programming Education: Let LOGO Be a Game. In Proceeding of 2011 Third International Conference on Games and Virtual Worlds for Serious Applications, Athens, Greece, 4 May 2011; pp. 24–31. [CrossRef]
49. Lee, M.J.; Ko, A. Gidget. 2020. Available online: https://www.helpgidget.org (accessed on 23 October 2020).
50. Kölling, M. The Greenfoot Programming Environment. *Trans. Comput. Educ.* **2010**, *10*, 1–21. [CrossRef]
51. Dietrich, J.; Tandler, J.; Sui, L.; Meyer, M. The PrimeGame Revolutions: A Cloud-based Collaborative Environment for Teaching Introductory Programming. In Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference, Adelaide, SA, Australia, 28 September–1 October 2015; pp. 8–12. [CrossRef]
52. Lund, A.M. Measuring Usability with the USE Questionnaire. *Usability Interface* **2001**, *8*, 3–6.
53. Barr, M. The Bartle Test of Gamer Psychology. 2018. Available online: https://matthewbarr.co.uk/bartle (accessed on 23 October 2020).
54. Bartle, R. Hearts, clubs, diamonds, spades: Players who suit MUDs. *J. MUD Res.* **1996**, *1*, 19.