View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

Game-Based Coding Challenges to Foster Programming Practice

José Carlos Paiva
CRACS – INESC-Porto LA, Portugal
DCC – FCUP, Porto, Portugal
jose.c.paiva@inesctec.pt

José Paulo Leal
CRACS – INESC-Porto LA, Portugal
DCC – FCUP, Porto, Portugal
https://www.dcc.fc.up.pt/~zp
zp@dcc.fc.up.pt

Ricardo Queirós
CRACS – INESC-Porto LA, Portugal
uniMAD – ESMAD, Polytechnic of Porto, Portugal
http://www.ricardoqueiros.com
ricardoqueiros@esmad.ipp.pt

Abstract

The practice is the crux of learning to program. Automated assessment plays a key role in enabling timely feedback without access to teachers but alone is insufficient to engage students and maximize the outcome of their practice. Graphical feedback and game-thinking promote positive effects on students' motivation as shown by some serious programming games, but those games are complex to create and adapt. This paper presents Asura, an environment for assessment of game-based coding challenges, built on a specialized framework, in which students are invited to develop a software agent (SA) to play it. During the coding phase, students can take advantage of the graphical feedback to complete the proposed task. Some challenges also encourage students to think of a SA that plays in a setting with interaction among SAs. In such a case, the environment supports the creation and visualization of tournaments among submitted agents. Furthermore, the validation of this environment from the learners' perspective is also described.

2012 ACM Subject Classification Applied computing → Interactive learning environments; Applied computing → E-learning

Keywords and phrases games, automatic assessment, graphical feedback, programming, learning, challenges

Digital Object Identifier 10.4230/OASIcs.ICPEC.2020.18

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.

1 Introduction

The amount of time dedicated to practice in programming classes is scarce considering the quantity of knowledge that students need to assimilate. For instance, in a regular sixteen-week semester, introductory programming courses often do not have more than twelve programming assignments [2], which is both too much for teachers to provide meaningful individualized feedback to students and too little for novices to master programming skills. Consequently, it is necessary to provide students with the right tools and resources to increase practice time and maximize its outcome.


© José Carlos Paiva, José Paulo Leal, and Ricardo Queirós;
licensed under Creative Commons License CC-BY
First International Computer Programming Education Conference (ICPEC 2020).
Editors: Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões; Article No. 18; pp. 18:1–18:11

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Automated assessment tools were found useful to grant instant, teacher-free, and effortless feedback for students on attempts to solve exercises [1]. These tools can provide both static and dynamic program analysis. Static analysis involves the compilation of the source code whereas dynamic analysis evaluates the compiled source code, typically providing some input and comparing its output to the solutions' output. The importance and benefits of such approaches in learning are well-documented in the literature [6, 5]. Yet, since these tools are primarily developed for programming contests, their feedback is usually insufficient for learning as is their means to engage and retain learners.

Notwithstanding, tuning dynamic analysis of programs can play a key role in motivating practitioners. For instance, CodeRally [8] and RoboCode [7] are two popular games among programmers in which players are Software Agents (SAs), developed by them, that compete against each other in a 2D environment. The SAs' owners then visualize the graphical feedback of the game unfolding. These games are used both by novice programmers to learn Object-Oriented Programming concepts and more advanced programmers for improving skills while taking pleasure out of it. Some teachers introduced them in programming classes with quite success [3], but only in certain topics as the complexity and costs of creating this kind of games 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, that aims to engage students while working on their programming assignments to increase the time spent on and the outcome of their programming activities. To this end, it fosters students' motivation by challenging them to code SAs. The feedback provided by the evaluation environment consists of a movie displaying the behavior of the SA during the game. These challenges are boosted with 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. On 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 [11]. Hence, this paper focuses on the learners' perspective of Asura.

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

The idea of fostering programming students' motivation through games is not new, neither that of challenging novice programmers to develop an SA to make decisions as a player of a game. The novelty of Asura is that it aims to make the creation of game-based coding challenges, where the learner has to code an SA for a game, simple enough to allow their use in educational contexts for teaching specific programming concepts and increase practice. Therefore, this section introduces some platforms and tools that also provide game-based programming challenges with graphical game-like feedback.

2.1 CodinGame

CodinGame[1] is a web-based platform that proposes several puzzles for learners to practice their coding skills. Most of these puzzles require the user to develop an SA to control the behavior of a character in a game environment, and offer game-like graphical feedback. The SA programmed by the player must pass all test cases (public and hidden) to solve the puzzle. Players can choose one of the more than twenty programming languages available to write their SA, or even solve it in different languages. Once the exercise is solved, players can access, rate, and vote on the best solutions.

Solving these game puzzles and awarding votes on solutions contributes to the leaderboards, level, and badges of the user. There are also contests from time to time in which the player can receive real-world rewards, by defeating other players' agents in a match or being the first to solve all problems. Another mode that CodinGame provides is the *Clash of Code*, where a player competes against other players to be the first to submit the best solution to an exercise. These exercises typically last for five minutes and can be authored by the community, although only text-based test cases are supported in this case.

Even though it has a large variety of challenges, this platform is not adequate to a classroom setting as educational resources are scattered, lacking a proper order for learning. Furthermore, it is a commercial platform and does not allow external persons to author challenges or modules.

2.2 SoGaCo

SoGaCo (Social Gaming and Coding) [4] is a scalable cloud-based web environment that evaluates competitive SAs, developed either in Java or Python, for 2-player board games. 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.

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. The modular architecture of SoGaCo supports different games, but there is no known framework or standard form to develop games for SoGaCo. Nevertheless, it already contains four board games, namely PrimeGame, Mancala, Othello, and 5-in-a-row.

This environment, to the best of the authors' knowledge, is the most similar with Asura. However, it only supports 2-player board games and does not provide any framework to develop new challenges or running tournaments.

3 Asura

Asura is an automatic assessment environment for game-based programming challenges. Its main goal is to provide a means to increase time dedicated to programming practice by engaging students with intrinsic motivators of games while requiring teachers a comparable effort to that of creating a traditional programming problem. Hence, this environment

[1] `http://codingame.com`

enables students to enjoy learning activities using unique features of games, such as graphical feedback, game-thinking, and competition, while including a set of tools designed to support authors during the process of creating Asura challenges, such as a framework and a CLI tool.

This environment follows a multi-component architecture, as depicted in the UML diagram of components of Figure 1, composed of three individual components, named Asura Viewer, Asura Builder, and Asura Tournament Manager, and a component designed as a plugin for an evaluation engine, Asura Evaluator. In this architecture, Mooshak [9] has been selected as the evaluation engine that will connect with Asura Evaluator. 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. Furthermore, its current version includes a pedagogical environment, named Enki [12], which 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. This makes it a perfect fit to also integrate both the Asura Viewer on Enki and the Asura Tournament Manager on the administration interface. Furthermore, the Java ARchive (JAR) of the challenge, produced on the Asura Builder, can also be easily imported on the administration interface and used in dynamic analysis.

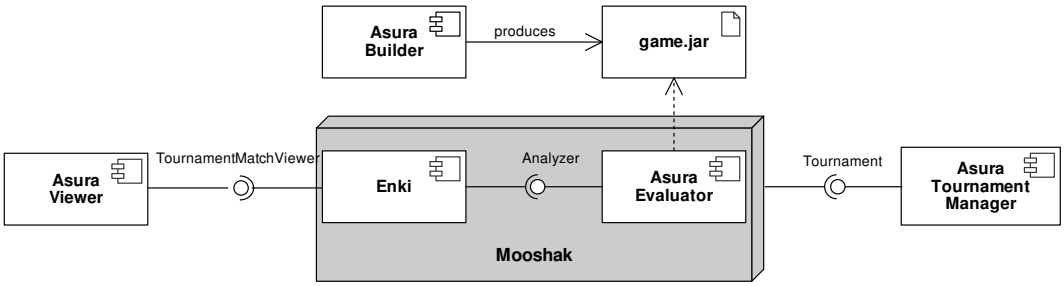

Figure 1 Architecture of the proposed ecosystem of Asura.

3.1 Asura Builder

The Asura Builder is a standalone component composed of multiple tools dedicated to the authoring of game-based coding challenges. It includes 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 framework is accompanied by a CLI tool to easily generate Asura challenges and install specific features, such as support for a particular programming language or a standard turn-based game manager. Even though the authors are required to program the challenges in Java, players can use their preferred programming language to code their SAs.

Since graphics concentrate much of the effort in game development, Asura Builder introduces the concept of game movie, defines a JSON schema to describe it, and provides a builder for objects adhering to it. 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 while the game unfolds with the support of the builder, which, in addition to having methods to construct each frame with ease, includes methods to deal with SA evaluation and premature game termination.

The core of an Asura challenge is the game manager, which among many tasks should ensure that the game rules are followed, decide which player takes the next turn, and declare the winner(s). Asura Builder 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, automatic (de)serialization of JSON or XML to Java objects, and a "contract" for the object that manages the game state. The author only needs to implement the specific parts of the challenge being created, translating mutations of the game state into frames of the game movie.

Moreover, the framework also allows to define wrappers for the SAs. A wrapper consists of a set of methods that aim to give players an 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 instance, a wrapper for a Go player might define a method `addStone(x: int, y: int)` to add a stone of the player in position (x, y) as well as a method `lastStone(): object` to get the last move of the opponent, in order to alleviate the hardness of the exercise. 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.

3.2 Asura Evaluator

Mooshak's evaluation engine follows a black-box approach to grade a submission according to a set of rules while generating a report of the evaluation for further validation from a human judge. The assessment process is twofold, comprising static analysis, which checks for integrity of the source code of the SA and produces an executable program; and dynamic analysis, that involves the execution of the program with each test case loaded with the problem and the comparison of its output to the expected output.

Asura Evaluator inherits the static analysis from Mooshak, only attaching the global and game-specific SA wrappers present in the game JAR file to the command-line. However, the dynamic analysis is completely re-implemented. Instead of test cases based on input and output text files, Asura Evaluator receives as input a list of paths to opponents' submissions. The type of evaluation either a validation or a submission determines the source of the competitors, in case of multiplayer games. Validations do not count for evaluation purposes but are rather a means for the learner to experiment how his/her SA behaves in a match against any existing SA. The opponents are selected by the learner itself from a list containing all the last accepted SAs from the students as well as the control SAs included by the author. On the contrary, submissions are considered as attempts to solve the challenge and as such are evaluated equally for all participants. In this case, the opposing contestants are the control SAs provided by the challenge author.

The opponents' SAs are submissions already compiled and, thus, the component only initializes a process from the compiled sources. After that, it organizes matches containing the current submission and a distinct set of the selected opponents' submissions. The length of this set depends on the minimum and maximum number of players per match, which are specified by the game manager. At this point, the evaluation proceeds on an instance of the specific game manager, which is instantiated from the JAR. The game manager receives the list of player processes indexed by the player ID and crosses the input and output streams of each of these processes with itself. This allows the game manager to write state updates to the input stream of the SA and read actions from it output stream, as typical I/O operations

from the SA perspective. Hence, the execution of the game is completely controlled by the game manager, which is responsible for keeping the SA's informed about the state of the game, querying the SAs for their actions at the right time, ensuring that the game rules are not broken, managing the state of the game, and classifying and grading submissions.

At the end, the statuses obtained from the matches containing the observations, mark, classification and feedback are compiled into a single status which is added to the submission report, and sent to Enki.

3.3 Asura Tournament Manager

The Asura Tournament Manager is a Java library for organizing tournaments among SAs submitted and accepted on an Asura challenge. Tournaments are optional and aim to give a final objective to students by inviting them to engage in a contest realized at the end of the submission time. Hence, the challenge is not just about solving the problem, but also to prepare the SA to win a final competition. During the preparation phase, students can do "friendly" matches (i.e., validations) against any previously approved SA from each other to get an idea of what they can expect to achieve in the tourney. After this phase, instructors can use a wizard embedded in Mooshak's administration user interface to setup the tournament.

Tournaments follow similar models to those found on traditional games and sports competitions. They can have several stages, each of them arranged according to one of the available tournament formats: round-robin, swiss system, single elimination, or double elimination. A stage is composed of a series of rounds in which each competitor either participates in a single match or has a bye (i.e., advances directly to the next round of the tourney in the absence of assigned opponents). Finally, matches are the atom of a tournament. They are generated based on the tournament format, one after another, executed on the Asura Evaluator, and its outcome sent back to the Asura Tournament Manager. The result of a match is a list containing the points obtained by each player as well as any additional features that could be used as tiebreakers, either for the match or rankings.

The interaction with the library is done through the interface `Tournament`, which provides a sequential and seamless way to run the tournament. Firstly, the mandatory metadata of the tournament such as the title, game, and participants should be provided. Then, the stages are added, configured, and started one by one. The configuration options of a stage depend on its format and may include the number of players per match, the minimum number of players per group, the number of qualified players, the maximum number of rounds, the type of result of a match (e.g., win-draw-loss or position-based), and tiebreakers both for rankings and matches. Once the stage has started, the next match to execute can be obtained, activating the wait mode until the result of the match is submitted and processed. At the end, the output of the Asura Tournament Manager is a JSON object complying to the tournament JSON schema.

3.4 Asura Viewer

Asura Viewer is the component responsible for displaying graphical feedback to learners, both in single matches and in tournaments. It consists of a Google Web Toolkit (GWT) widget that transforms the provided JSON either into a movie of a match or an interactive view of the tournament, according to the schema it adheres.

The match mode is where learners can see how their SAs behaved during the match. It presents the JSON output produced during the evaluation of the submission or validation as a dynamic movie, only distinguishable from a game in the fact that it can be pushed back and forth. The widget, presented on the left of Figure 2, mimics that of a media player

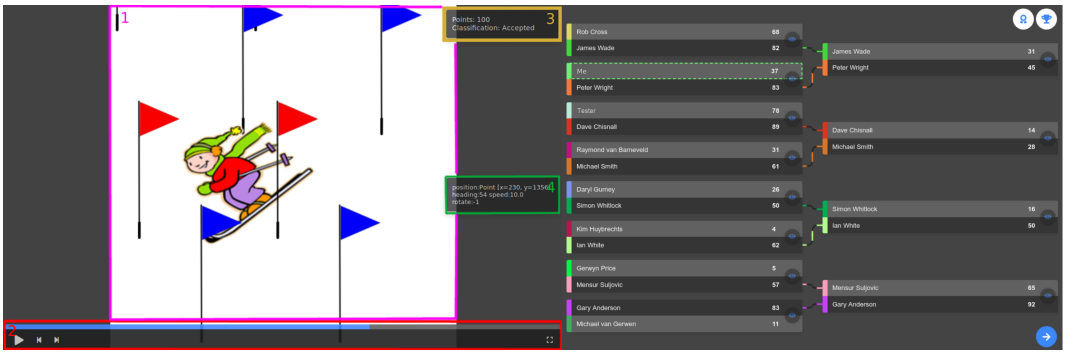

Figure 2 Asura Viewer modes. On the left, the match mode with Slalom Skier game (distinct areas highlighted in different colors). On the right, the brackets view of the tournament mode.

including a slider, a play/stop button, buttons to navigate through the current playlist, and a full-screen button in the control toolbar (red area), a box to show the current status (yellow area), a box to display debugging messages of the SA (green area), and a canvas where the movie is drawn (pink area).

The tournament mode, presented on the right of Figure 2, consists of an interactive user interface which allows students to navigate through the stages of the tournament, visualize specific matches or the whole course of a player, and check the rankings of each stage. It comprises a navigation menu on the bottom right corner to swap the current stage, a contextual menu on the top right corner to switch between standings view (either final or relative to the current stage) and brackets view, and the main area where matches and ranking appear.

4 Validation

An experiment was conducted to validate the acceptance of Asura by learners as well as its effectiveness both in motivating students to increase the time that they dedicate into programming practice and in maintaining or improving the knowledge acquisition and retention rates of traditional programming exercises. This experiment took the form of an open online learning course about the novel features introduced by version ECMAScript 6 (ES6) of JavaScript. The course has been announced to undergraduate students registered in the Web Technologies classes in the past semester, which provided them with some background on JavaScript but not on ES6 features.

A total of 10 students enrolled in the course, of which 1 was female. These students were randomly divided into two groups of 5, control (1 female) and treatment. Each of the groups made a separate branch of the course with the same expository resources but different evaluative resources. The expository resources are lecture notes about the concepts of ES6, including variable declaration, object and array destructuring, arrow functions, promises, and classes, and a compiled ES6 cheat sheet. The evaluative resources are either International Collegiate Programming Contest (ICPC)-like problems (control branch) or Asura challenges (treatment branch). At the end of the course, students from both groups were invited to do an exam composed only of ICPC-like problems to assess the knowledge acquired during the course.

The Asura challenges are different chapters of the same game which is a remake of Asteroids, an arcade space shooter released in November 1979 by Atari, named War of Asteroids. This remake keeps most of the original gameplay of Asteroids, but adds a number

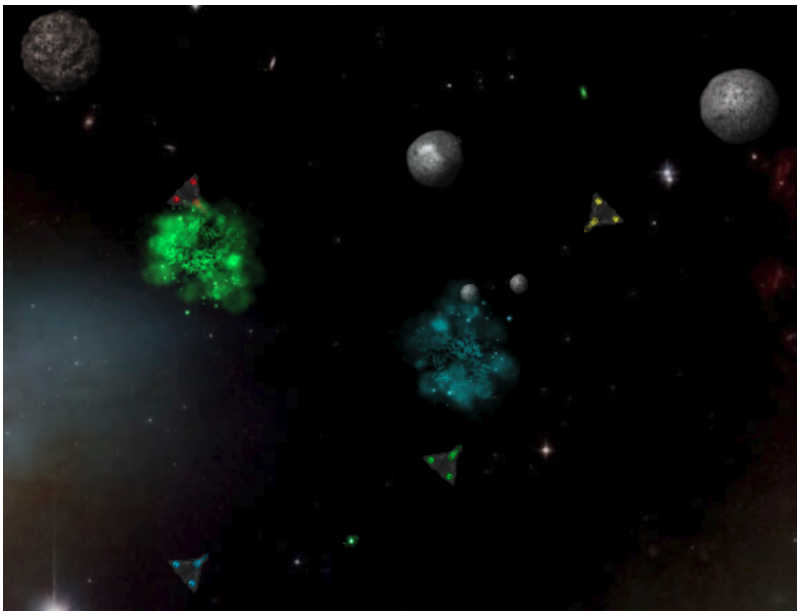

Figure 3 War of Asteroids. Screenshot of a game with four ships competing against each other.

of features; improves graphics (see Figure 3); and replaces the input controls with a program. The game is now played by up to 4 contestants, which replace the saucers. Furthermore, the ships have two additional commands that can be used: activate the energy shield and fire bombs. The ways of earning points as well as the number of points awarded per accomplishment were also modified. The game ends after 10000 units of time (frames) or when a player gets alone in the space.

The game-specific wrapper provides SAs with 7 commands, particularly, `thrust()` thrusts the ship, `steerLeft()` adds -4 degrees to the heading of the ship, `steerRight()` adds 4 degrees to the heading of the ship, `shield()` activates the shield, `firePrimary()` fires a bullet, `fireSecondary()` throws a bomb, and `log(message)` logs a message. Some of these commands are not available in the first chapters, but revealed during the course.

4.1 Results and Analysis

The data gathered during the experiment consists of usage data and questionnaire responses. The usage data is automatically captured by Mooshak 2 into the activity log based on every request sent to the server. This enables the extraction of several metrics such as the number of submissions, number of validations, date and time of activity, and submissions' results. The questionnaire is based on the Lund's model [10], including one section per metric (i.e., *Usefulness*, *Ease of Use*, *Ease of Learning*, and *Satisfaction*) with questions to classify sentences in a 7-value Likert scale and an additional section with free-text questions to collect students' feedback about Asura regarding weaknesses, strengths, and points of improvement.

The questionnaire is part of the exam to guarantee that only students who complete their course and ask for the exam would fill it in. Two of the students (one from each group) have not taken the exam and, thus, they did not answer the survey. The remaining students have finished both the exam and the questionnaire. The outcome from the questionnaire is presented in Figure 4 separated by group, control on the left and treatment on the right.

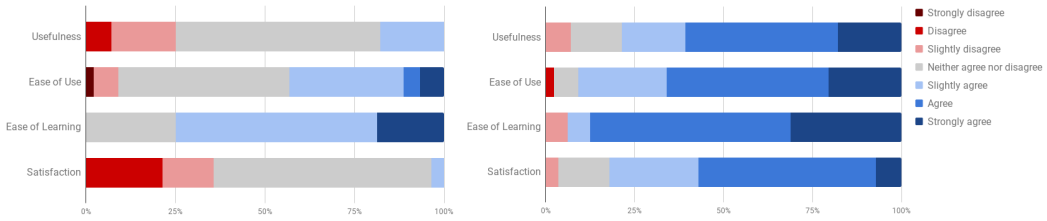

Figure 4 Results of the usefulness, ease of use, ease of learning, and satisfaction questionnaire of Asura: the control group (on the left) and the treatment group (on the right).

The results obtained from the questionnaire reveal improvements in the four metrics: usefulness, ease of use, ease of learning, and satisfaction. For instance, the usefulness of the control group had an average rating of 3.86 whereas the treatment group was 5.50 and the satisfaction valued 3.46 against 5.43, on the control and treatment groups respectively. From the free-text questions, it is possible to identify the main reason for these differences as being the feedback quality since some students in the control group pointed out feedback as a weakness (e.g., "Observations and program input/output could be improved" and "Feedback messages are scarce") while learners in treatment enjoyed the graphical feedback and let suggestions to further improve it in the chosen game (e.g., "On the game map, insert the name of the player above the ship" and "Insert a board that displays the remaining energy of the ship"). However, students have complained about the difficulty of getting started with the game and ES6 at the same time (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."). The user interface of Enki has been criticized by students of both groups either because it lacks some features that they were expecting to have (e.g., terminal and debugger) or it looks bad on some devices (e.g., MacBook Air 13"). In regards to strengths, students emphasized the possibility of learning JavaScript through a game like the War of Asteroids (e.g., "Basic game to learn JavaScript, easy to get used to and easy to program against too.").

The analysis of the usage data aims to estimate the real efficacy of Asura in increasing practice time and, possibly, its impact on knowledge acquisition and retention. It is based on 6 variables, including the group (either control or treatment), number of validations, number of submissions, number of different days in which students have made an attempt (also known as return days), ratio of solved exercises in the course (a number between 0 and 1), and score obtained in the exam (an integer between 0 and 3), and a calculated attribute defined by the sum of the number of submissions and the number of validations, the number of attempts. The comparisons of the number of attempts, return days, the ratio of solved exercises, and exam grades between both groups demonstrate improvements in each of these quantitative metrics, as depicted in the boxplots of Figure 5.

The treatment group made 370 submissions of which 65 have been accepted whereas the control group submitted 44 times of which 16 succeeded. Adding this information to the percentage of exercises that were solved and the return days in both groups, it can be concluded that the students found it difficult to solve both kinds of challenges, but they struggled more in the treatment group than in the control group to overcome their difficulties. Furthermore, students in control only submitted until they solved the exercises while in the treatment they have made 40+ submissions than necessary, all of them in Chapter IV (i.e., the Chapter where competition starts). The amount of validations also reflects these trends,

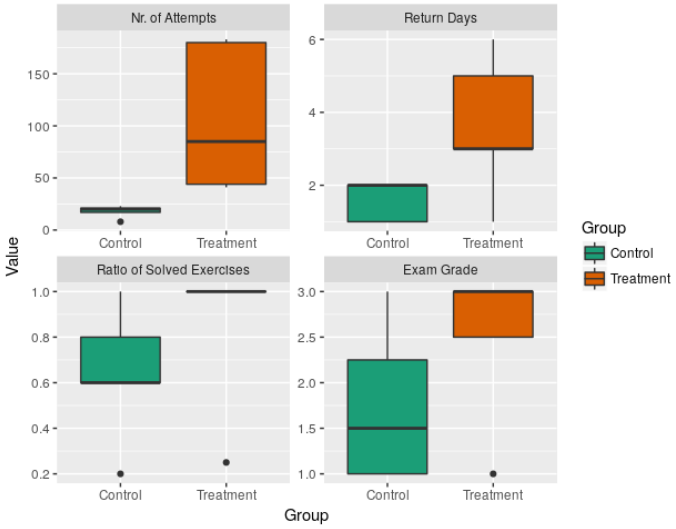

Figure 5 Quantitative comparison of metrics per group: number of attempts, return days, ratio of solved exercises, and exam grade.

yet the analysis suggests that students did not correctly understand the concept of Asura validations (i.e., friendly matches against the opponents) because they only validated 15 times in Chapter IV. The exam, which has been realized only by 4 learners of each group, shows that 75% of the students in treatment have achieved a good grade (between 2 and 3) and 50% in control. Nevertheless, the differences are so expressive due to the low number of participants that allowed other factors to be highlighted in the analysis.

5 Conclusions

This paper presents Asura, an automated assessment environment for game-based coding challenges, that aims to foster students' motivation requiring teachers a similar effort to that of creating traditional programming exercises. This environment offers teachers a framework and a CLI to author game-based programming challenges in which learners code an SA to play a game. These challenges introduce a competitive element in the form of tournaments similar to those realized in traditional games and sports, to further enhance the endeavor to develop better solutions that can beat their opponents.

The analysis of the data collected during the validation demonstrated relative success of Asura in accomplishing its goals. Nevertheless, results are based on a very small sample of students and, thus, insufficient to draw effective conclusions. The majority of the students of the control group expressed dissatisfaction with the amount and quality of feedback provided, whereas students of the treatment group have complained about the difficulty of getting started with the War of Asteroids, while also learning new concepts of ES6. Both groups pointed out a few issues in the user interface of the underlying system, Enki.

The next step is to enrich a repository of challenges already created, as from previous experience, instructors typically prefer to compile a set of existing exercises about a concept and use them.

References

1. Kirsti M. Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005. doi:10.1080/08993400500150747.
2. Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students: Some thoughts and observations. *SIGCSE Bull.*, 37(2):103–106, June 2005. doi:10.1145/1083431.1083474.
3. Esmail Bonakdarian and Laurie White. Robocode throughout the curriculum. *J. Comput. Sci. Coll.*, 19(3):311–313, January 2004.
4. Jens Dietrich, Johannes Tandler, Li Sui, and Manfred Meyer. The primegame revolutions: A cloud-based collaborative environment for teaching introductory programming. In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, ASWEC ' 15 Vol. II, pages 8–12, New York, NY, USA, 2015. ACM. doi:10.1145/2811681.2811683.
5. Stephen H. Edwards. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.*, 3(3):1–es, September 2003. doi:10.1145/1029994.1029995.
6. E. Enström, G. Kreitz, F. Niemelä, P. Söderman, and V. Kann. Five years with kattis — using an automated assessment system in teaching. In *2011 Frontiers in Education Conference (FIE)*, pages T3J–1–T3J–6, October 2011. doi:10.1109/FIE.2011.6142931.
7. Ken Hartness. Robocode: Using Games to Teach Artificial Intelligence. *J. Comput. Sci. Coll.*, 19(4):287–291, April 2004. URL: http://dl.acm.org/citation.cfm?id=1050231.1050275.
8. IBM Community. Coderally, 2013. Last checked on November 2019. URL: https://ibm.com/developerworks/community/blogs/code-rally.
9. José Paulo Leal and Fernando Silva. Mooshak: A Web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003. doi:10.1002/spe.522.
10. Arnold M. Lund. Measuring usability with the use questionnaire. *Usability interface*, 8(2):3–6, 2001.
11. José Carlos Paiva, José Paulo Leal, and Ricardo Queirós. Authoring game-based programming challenges to improve students' motivation. In Michael E. Auer and Thrasyvoulos Tsiatsos, editors, *The Challenges of the Digital Transformation in Education*, pages 602–613, Cham, 2020. Springer International Publishing.
12. José Carlos Paiva, José Paulo Leal, and Ricardo Alexandre Queirós. Enki: A pedagogical services aggregator for learning programming languages. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 332–337. ACM, 2016. doi:10.1145/2899415.2899441.