Asura: A Game-Based Assessment Environment for Mooshak

José Carlos Paiva
CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Portugal
up201200272@fc.up.pt
https://orcid.org/0000-0003-0394-0527

José Paulo Leal
CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Portugal
zp@dcc.fc.up.pt
https://orcid.org/0000-0002-8409-0300

Abstract

Learning to program is hard. Students need to remain motivated to keep practicing and to overcome their difficulties. Several approaches have been proposed to foster students' motivation. As most people enjoy playing games of some kind and play on a regular basis, the use of games is one of the most widely spread approaches. However, taking full advantage of games to teach specific concepts of programming requires much effort. This paper presents Asura, a game-based assessment environment built on top of Mooshak that challenges students to code Software Agents (SAs) to play a game, allowing them to test the SAs against each others' SAs and watch a movie of the test. Once the challenge development stage ends, teachers are able to organize game-like tournaments among SAs. One of the key features of Asura is that it provides a means to reduce the required effort of building game-based challenges up to that of creating traditional programming exercises.

2012 ACM Subject Classification Applied computing → Interactive learning environments, Social and professional topics → Computational science and engineering education

Keywords and phrases games, programming, learning, graphical feedback, tournament

Digital Object Identifier 10.4230/OASIcs.SLATE.2018.9

Category Short Paper

Funding This work is partially funded by the ERDF through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, by National Funds through the FCT as part of project UID/EEA/50014/2013, and by FourEyes. FourEyes is a Research Line within project "TEC4Growth – Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact /NORTE-01-0145-FEDER-000020" financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

1 Introduction

In the past recent years, the demand for programmers in the job market has grown rapidly, raising the popularity of programming courses among future undergraduate students who seek for a rewarding career [16]. However, learning to program is hard. Introductory programming courses are considered difficult by many students [4] and often linked to high dropout and failure rates [1]. Many educators consider the lack of abstraction and problem-solving skills as the main sources of difficulties for novice programmers [7]. Nevertheless, these difficulties can be mitigated if students have enough motivation to keep practicing and struggling to overcome them [10].

Consequently, several approaches to foster the engagement of students in programming activities, such as problem-based learning, storytelling, simulations, and competition-based learning, have been proposed and evaluated. Yet, one of the most successful and widespread approaches is gamification, which consists of using game elements and mechanics to engage users in non-game contexts. The most common methods of gamification just give rewards (e.g., badges, experience points, or gifts) to students when they succeed or complete a task. However, rewards are generally weak motivators and its use is controversial, since many argue that they can harm intrinsic motivation and arise long-term educational issues [11]. Other game aspects such as graphical feedback, competitive challenges, goals, collaboration, or abstraction of the physical world, are, typically, much more attractive than rewards. In this sense, some proposals replace the learning activity completely by a game, as is the case of CodeRally[1] and RoboCode [6] which have demonstrated the success of this approach. Unfortunately, the implementation of complete games to teach a specific concept is very costly in terms of time and money.

This paper presents Asura, a game-based assessment environment built on top of Mooshak [9], a system for managing programming contests on the web. Specifically, it integrates into the computer science languages learning environment of Mooshak 2.0, named Enki [13], providing game-based programming challenges to any Enki course. Each challenge asks the student to code a Software Agent (SA) to play a game, requiring the student to understand the rules and goals of the game. After the "break the ice phase", students are challenged to improve their SAs insomuch that they defeat every opponent. While performing this task, students can take advantage of facing any of the existing opponents in a private match, and watch how the match unfolds in a game-like movie. Once the time to solve the challenge ends, educators can organize tournaments, similar to those found on traditional games and sports, among SAs. The tournament is displayed in an interactive GUI, allowing the viewers to control which games they want to see, navigate through stages, and check the ranking. Furthermore, one of the key goals of Asura is to enable teachers to build games with a similar complexity to that of creating an ICPC-like problem. These games can be very simple, such as a number guessing game, or more complex than CodeRally or Robocode. For that, Asura provides a set of helpers including a Java framework and a Command-Line Interface (CLI) tool to support the authoring of challenges.

The remainder of this paper is organized as follows. Section 2 reviews the systems and tools in which Asura lies on as well as environments with characteristics common to Asura. Section 3 describes Asura, its architecture and components. Section 4 provides the guidelines of the experiment that will be conducted to validate Asura. Finally, Section 5 summarizes the contributions of this paper, the expected results of the validation, and the next steps of this work.

2 State-of-the-Art

Asura is a game-based assessment environment designed to integrate into an already existing computer science languages learning environment of Mooshak 2.0, named Enki. The environment aims to offer a way to motivate students to program and overcome their difficulties through practice, requiring from teachers an effort similar to that of creating an ICPC-like

[1] `https://www.ibm.com/developerworks/mydeveloperworks/blogs/code-rally`

problem. It engages students by challenging them to code an SA to play a game, supporting them with graphical game-like feedback to visualize how the SA performs against other SAs. The final goal of an Asura challenge is to win a tournament, like those found on traditional games and sports, among all submitted SAs.

There are already tools in the literature providing some of these features separately. For instance, competition is not a new paradigm in programming learning [2, 8]. Students are increasingly facing programming contests after leaving universities as a part of the recruitment process for top technology companies. So, providing a learning experience with focus on competition may help them in the future. One of the first types of systems to foster competition among programming learners were automatic judges, such as DOMJudge[2], PKU JudgeOnline[3], and Mooshak [9]. Even if these systems were developed for international and regional competitions, teachers of undergraduate programming courses found them useful also as teaching assistant tools [5, 3] to promote competitive programming learning environments and give instant feedback on laboratory classes and exams. In particular, the increasing interest shown on using Mooshak for learning has motivated the development of several extensions, such as quiz evaluation and exam policies.

Recently, Mooshak was completely reimplemented in Java with Graphic User Interfaces (GUIs) using the Google Web Toolkit (GWT). This reimplementation was labeled as Mooshak 2.0[4]. Beyond the changes in the code-base, Mooshak 2.0 gives special attention to computer science learning. For instance, it now includes a specialized computer science languages learning environment – Enki [13] – , which not only supports exercises using typical programming languages, but also diagramming exercises, quizzes, among others. Enki blends assessment and learning, integrating with several external tools, such as a gamification service – Odin [12] – to support the creation of leaderboards, reward students for their successes, among others, and an educational resources sequencing service – Seqins [14] – to offer different learning paths according to the skills of each student.

Currently, there are many other web-based learning platforms that focus on competition to motivate learners, such as HackerRank[5], in which competition is based on leaderboards that consider the number of solved problems and time spent solving them, and CodeFights[6], in which a player "fights" against other player or a bot developed by a company, to complete a set of challenges before the opponent.

Also, game-based approaches for teaching programming have already been applied in several cases. Rajaravivarma [15] proposes a course with a set of challenges, in which the students have to program their own games. Sui et al. [17] presents a browser-based environment that combines gamification and peer-to-peer interaction. This environment challenges the students to write SAs to play simple board games, allowing them to test these SAs against SAs developed by other peers. The environment also provides a debugger interface, which allows the student to check the state of the SA step-by-step.

There are also a few online platforms using games as the central source of engagement, such as CodinGame[7] and Leek Wars[8]. CodinGame proposes several puzzles for learners to practice their coding skills. Most of them require the user to develop an SA to control the

[2] `https://www.domjudge.org/`
[3] `http://poj.org/`
[4] `https://mooshak2.dcc.fc.up.pt/`
[5] `https://hackerrank.com/`
[6] `https://codefights.com/`
[7] `https://codingame.com`
[8] `https://leekwars.com`

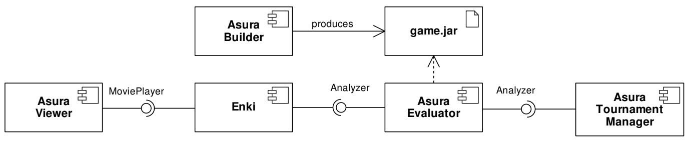


Figure 1 Diagram of components of the architecture of Asura.

behavior of a character in a game environment, and provide a 2D game-like graphical feedback. Leek Wars is a serious game where the player controls a character (a leek) through coding. The goal of the game is to beat other players' leeks during fights, awarding points to climb to the top of the ranking. The result of the fight is displayed as a 2D frame-by-frame movie.

3 Asura

Asura is an environment for game-based assessment in programming learning. Its main features include the graphical game-like feedback, the tournament-based assessment, and the framework and tools that it offers regarding the simplification of the process of building Asura challenges. This environment leverages the SAs evaluation on Mooshak 2.0's program analysis, extending it to support multiple submissions running in the same evaluation environment. This allows students to code their SAs in any programming language supported in Mooshak. On the client side, the Graphic User Interface (GUI) is embedded into Enki's GUI, allowing any course created in Enki to provide Asura challenges out-of-the-box.

The architecture of Asura, presented in Figure 1, is composed of four components: Asura Builder, Asura Viewer, Asura Tournament Manager and Asura Evaluator. The Builder, Viewer, and Tournament Manager are completely new components, whereas the Evaluator is a component that extends Mooshak 2.0 program analysis to support game assessment. In this architecture, Enki makes the bridge between the server and the client, requesting evaluation to the Evaluator and loading the feedback into the Viewer. Each of the next subsections describes a component of Asura, including its role and some implementation details.

3.1 Evaluator

The evaluator engine of Mooshak grades a submission by following a set of rules while generating a report of the evaluation for further validation from a human judge. This evaluation follows a black-box approach. The process consists of two types of analysis: static, which checks for integrity of the source code of the program and produces an executable program; and dynamic, that involves the execution of the program with each test case loaded with the problem.

The Evaluator component of Asura inherits the static analysis of the Mooshak's evaluator engine. The only difference is that the compile command line can include a language-specific player wrapper, present in the `game.jar` file, for complex games. However, the dynamic analysis is completely reimplemented. Instead of test cases with input and output text files, Asura Evaluator receives as input a list of paths of the selected opponents' submissions that generate the input and consume the output of each other. Since these submissions are already compiled (when necessary), the component just initializes a process for each of them. 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 `game.jar`, as well as the game state object. The game manager receives the list of player processes indexed by the player ID and starts the game. 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 state update at the right time, ensuring that the game rules are not broken, managing the state of the game, and classifying and grading submissions. If an SA fails to comply with the rules, the match ends and the SA receives a "Wrong Answer".

During the game, any updates to the state object can passed on to the movie builder. The state object provides the following methods that can be used by the manager to control its lifecycle: `prepare(movieBuilder, players)` which initializes the state and sets up the metadata of the movie, `execute(movieBuilder, playerId, playerUpdate)` that updates the game state with the action of a certain player, `endRound(movieBuilder)` which ends a round of actions, and `finalize(movieBuilder)` which finalizes the game, adding the submission results in the movie, among other things.

Finally, the status 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 the client.

3.2 Builder

The Builder component is a Java framework for building Asura challenges, providing a game movie builder, a general game manager, several utilities to exchange complex state objects between the manager and the SAs, and general wrappers for players in many different programming languages. The framework is accompanied by a Command-Line Interface (CLI) tool to easily generate Asura challenges and install specific features, such as support for a particular programming language, a default turn-based game manager, among others.

Much of the necessary effort for building video games is spent on graphics. To simplify this task, Asura introduces the concept of game movie. 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, such as `title`, `background`, `width`, `height`, `fps`, and `players` (i.e., player names indexed by their ID). Furthermore, the concept of game movie is formally defined in a JSON schema[9]. To allow the manager to easily build the JSON data, the game movie builder provides several methods, such as: setters for each metadata field, `addFrame()` which adds a frame to the movie, `addItem(sprite, x, y, rotate, scale)` which adds a sprite to the current frame in position (x, y) with the given rotation and scale, `addMessage(playerId, message)` which adds a message to the player, setters for observations and classification, and `saveFrame()`/`restoreFrame()` which allow to add a frame state to a stack to restore it later.

The abstract game manager provided by the Builder defines the "contract" of the managers, and provides the necessary utilities for dealing with input/output streams of the processes. Specialized managers must implement the method `manage(state, players)`, determining the order to play, and managing the state of the game accordingly. Some of these specialized managers, such as a turn-based game manager, are already developed and can be easily used in a challenge.

The exchange of state updates between the SAs and the manager is done through JSON. Depending on the programming language, this can be a hurdle and make it very complex

[9] `https://mooshak2.dcc.fc.up.pt/asura/static/match.schema.json`

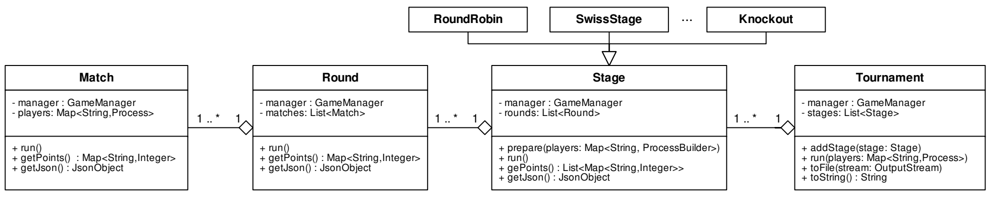


Figure 2 UML class diagram of the Tournament Manager.

for SA's to process it. For that, there are wrappers for players providing several methods to process JSON. Moreover, each game can define its own wrappers providing methods specific to the game, which can be used by SAs.

The documentation for the very first release of Asura Builder is available online[10] and has already been followed by some peers who volunteered to test the system.

3.3 Tournament Manager

The Tournament Manager is the component responsible for managing and running tournaments. A tournament can have any number of stages, each with its own type (pools or knockout) and format (e.g., round-robin, swiss type, knockout, double knockout, among others). Stages are populated with a set of players, those who qualified in the previous stage, which will compose the matches of every round of the current stage. The assignment of players to matches is a task of the class implementing the specific tournament format. These tournaments can be organized in the administrator GUI of Mooshak 2.0, through a wizard developed specifically for this task. This wizard allows to select players individually, add/remove stages, and set some properties of the tournament, such as the number of players per match, the number of qualified players in each stage, and the points awarded to a win/draw/loss.

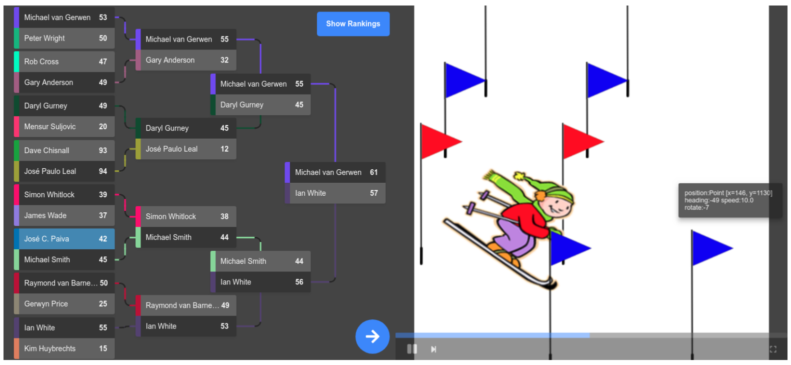
Figure 2 presents the UML class diagram of the implementation of the Tournament Manager. This implementation contains a class for the tournament as well as for each of its phases: stages, rounds, and matches. These classes have very similar methods, such as `run()` which executes the phase, and `getPoints()` which returns either a list of players' points, in group stages, or the players' points. Each match of the tournament is executed in the Evaluator component.

3.4 Viewer

Asura Viewer is a GWT widget with two modes: match and tournament. Figure 3 presents both modes side-by-side. In the tournament mode, it expects a JSON file following the Tournament JSON Schema[11]. This data contains a reference to each match's movie, organized by stages and rounds, as well as partial and complete rankings of each phase. The tournament mode widget consists of an interactive GUI, allowing students to navigate through the stages of a tournament, visualize specific matches or the whole course of a player, and check the rankings of each stage.

The match mode displays the game movie JSON produced during the evaluation phase. The GUI mimics that of a media player, containing a slider, a play/stop button, buttons to

[10] `https://mooshak2.dcc.fc.up.pt/asura/static/asura-builder-documentation.pdf`
[11] `https://mooshak2.dcc.fc.up.pt/asura/static/tournament.schema.json`



Michael van Gerwen 53
Peter Wright 50
Rob Cross 47
Gary Anderson 49
Daryl Gurney 49
Mensur Suljovic 20
Dave Chisnall 93
José Paulo Leal 94
Simon Whitlock 39
James Wade 37
José C. Paiva 42
Michael Smith 45
Raymond van Barne… 50
Gerwyn Price 25
Ian White 55
Kim Huybrechts 15

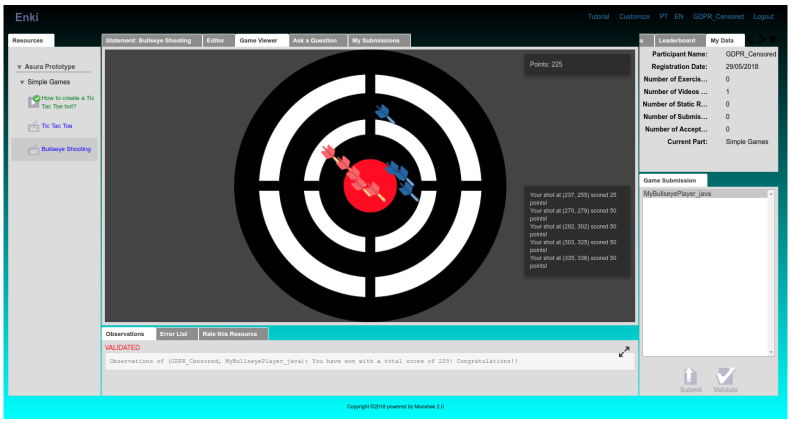
Michael van Gerwen 55
Gary Anderson 32
Daryl Gurney 45
José Paulo Leal 12
Simon Whitlock 38
Michael Smith 44
Raymond van Barne… 49
Ian White 53

Michael van Gerwen 55
Daryl Gurney 45
Michael Smith 44
Ian White 56

Michael van Gerwen 61
Ian White 57

position:Point [x=146, y=1130]
heading:-49 speed:10.0
rotate:-7

Figure 3 Asura Viewer display modes. Tournament mode (on the left) displays a knockout stage of a tournament. Match mode (on the right) presents the graphical feedback provided to a slalom skier.



Points: 225

Your shot at (237, 255) scored 25 points!
Your shot at (270, 279) scored 50 points!
Your shot at (292, 302) scored 50 points!
Your shot at (303, 325) scored 50 points!
Your shot at (335, 336) scored 50 points!

Participant Name:	GDPR_Censored
Registration Date:	29/05/2018
Number of Exercis…	0
Number of Videos …	1
Number of Static R…	0
Number of Submis…	0
Number of Accept…	0
Current Part:	Simple Games

MyBullseyePlayer_java

VALIDATED

```
Observations of (GDPR_Censored, MyBullseyePlayer_java): You have won with a total score of 225! Congratulations!!
```

Copyright ©2018 powered by Mooshak 2.0

Figure 4 Asura Viewer integrated in the GUI of Enki after the validation of the SA.

navigate through the current play-list, a full-screen button, a box to display messages, and a canvas where the movie is drawn. The movie is completely resizeable keeping the original aspect ratio.

This component can be embedded in any environment, provided that the JSON data passed to it adheres to the defined JSON schemas for tournaments and matches. Figure 4 presents a screenshot of the Asura Viewer integrated in the GUI of Enki, after a player validated its SA against an existing SA.

4 Experiment Guidelines

An experiment with Asura will be conducted in the laboratory classes of an undergraduate Object-Oriented Programming (OOP) course at *Escola Superior de Media Artes e Design* (ESMAD) – a school of the Polytechnic Institute of Porto. The purpose of this course is to introduce Javascript and some OOP concepts integrated into its ECMAScript 6 version to students with little to none programming background. The course is composed of two distinct laboratory classes which will be labeled hereafter as Control Group (CG) and Experimental Group (EG), respectively. The homogeneity of the groups will be checked using students' Grade Point Average (GPA). Both groups will have access to an online course in Enki containing a set of programming activities to assimilate knowledge obtained during expository classes. However, CG will have to solve traditional programming exercises, whereas EG will have to develop players for Asura games. These activities will be created by a group of three teachers, pertaining to the same institution as the students, with a single requirement that both sets of activities have identical difficulty. The course will be available online during two weeks, in which students are free to access in and after class.

At the end of the experiment, students and teachers will be asked to fill in online questionnaires about the usefulness of the learning environment. Teachers will have specific questions regarding the difficulty of developing problems for Asura. Students will have questions to assess the level of engagement obtained while using the environment. The results obtained from the online questionnaires as well as usage data collected during the preparation and execution of the course will be analyzed in order to draw conclusions about the effectiveness of Asura. This usage data includes a number of variables, such as the number of solved exercises, the number of submissions per problem, the time spent per exercise, and the number of different strategies used per problem (identified by a threshold on the number of different characters).

5 Conclusions and Future Work

This paper presents Asura, an environment that aims to provide engaging game-based activities with graphical game-like feedback as well as to facilitate the creation of those activities by teachers. From the students' perspective, the main idea of Asura is to challenge students to code an SA that plays a game, taking advantage of the graphical feedback on its performance against other SAs. Once the SA development period ends, teachers can organize tournament amongst SAs. With regard to teachers, Asura provides a Java framework that supports the creation of game-based challenges, particularly in the process of building the game movie.

Asura is a work in progress. It is currently in the final development stage, just lacking the implementation of a few types of tournament stages and some minor improvements in the Builder. The design of each component of Asura, including the way of integrating them with the existing work, is already done. Several game-based challenges were already developed using the framework provided by the Builder.

The next phase encompasses the execution of the experiment described in Section 4. It is expected that students in EG will spend considerably more time in activities, trying different strategies to beat up their colleagues, which would indicate greater engagement. However, some students may be negatively affected by losing and show their displeasure in the questionnaire's responses. Students in the CG will not spend more time in the environment than the necessary amount to solve all problems once. Also, some exercises may not be solved by the end of the experiment. In respect to teachers, they will notice a small increase in difficulty while creating the games, since they need to be familiar with Java and to understand the framework beforehand. The amount of time that they spend doing the graphics is also unpredictable since it highly depends on the quality that they want to reach. Nevertheless, this should not result in a significant difference in terms of time, when comparing with the time of setting up an ICPC-like problem, if they already know Java.

One of the major points of improvement already identified is in the Asura Builder. Many teachers are not familiar with Java, but the framework requires its use. This will certainly prevent or make it very difficult for these teachers to work with Asura. For this reason, the Builder component will be extended to support language-agnostic creation of games.

References

1 Jens Bennedsen and Michael E. Caspersen. Failure rates in introductory programming. *SIGCSE Bulletin*, 39(2):32–36, 2007. doi:10.1145/1272848.1272879.

2 Juan C. Burguillo. Using game theory and competition-based learning to stimulate student motivation and performance. *Computers & Education*, 55(2):566–575, 2010. doi:10.1016/j.compedu.2010.02.018.

3 Ginés Gárcia-Mateos and José Luis Fernández-Alemán. A course on algorithms and data structures using on-line judging. *SIGCSE Bulletin*, 41(3):45–49, 2009. doi:10.1145/1562877.1562897.

4 Anabela Gomes and António José Mendes. Learning to program-difficulties and solutions. In *International Conference on Engineering Education (ICEE)*, 2007.

5 Pedro Guerreiro and Katerina Georgouli. Enhancing elementary programming courses using e-learning with a competitive attitude. *International Journal of Internet Education*, 10(1):27–42, 2008.

6 Ken Hartness. Robocode: Using games to teach artificial intelligence. *Journal of Computing Sciences in Colleges*, 19(4):287–291, 2004.

7 Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. *SIGCSE Bulletin*, 37(3):14–18, 2005. doi:10.1145/1151954.1067453.

8 Ramon Lawrence. Teaching data structures using competitive games. *IEEE Transactions on Education*, 47(4):459–466, 2004. doi:10.1109/te.2004.825053.

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 Scheila Wesley Martins, António José Mendes, and António Dias Figueiredo. A strategy to improve student's motivation levels in programming courses. In *Frontiers in Education Conference*, pages F4F–1–F4F–7, 2010. doi:10.1109/FIE.2010.5673366.

11 Wilbert J. McKeachie. The rewards of teaching. *New Directions for Teaching and Learning*, 1982(10):7–13, 1982. doi:10.1002/tl.37219821003.

12 José Carlos Paiva, José Paulo Leal, and Ricardo Queirós. Odin: A service for gamification of learning activities. In *International Symposium on Languages, Applications and Technologies*, pages 194–204, 2015. doi:10.1007/978-3-319-27653-3_19.

13 José Carlos Paiva, José Paulo Leal, and Ricardo Alexandre Queirós. Enki: A pedagogical services aggregator for learning programming languages. In *Conference on Innovation and Technology in Computer Science Education*, pages 332–337, 2016. doi:10.1145/2899415.2899441.

14 Ricardo Queirós, Paulo José Leal, and José Campos. Sequencing educational resources with Seqins. *Computer Science and Information Systems*, 11(4):1479–1497, 2014. doi:10.2298/csis131005074q.

15 Rathika Rajaravivarma. A games-based approach for teaching the introductory programming course. *SIGCSE Bulletin*, 37(4):98–102, 2005. doi:10.1145/1113847.1113886.

16 Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003. doi:10.1076/csed.13.2.137.14200.

17 Li Sui, Jens Dietrich, Eva Heinrich, and Manfred Meyer. A web-based environment for introductory programming based on a bi-directional layered notional machine. In *Conference on Innovation and Technology in Computer Science Education*, pages 364–364, 2016. doi:10.1145/2899415.2925487.