Authoring Game-Based Programming Challenges to Improve Students' Motivation

José Carlos Paiva[1(✉)], José Paulo Leal[2], and Ricardo Queirós[2]

[1] CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal
up201200272@fc.up.pt

[2] CRACS & INESC-Porto LA & ESMAD/IPP, Porto, Portugal
zp@dcc.fc.up.pt, ricardoqueiros@esmad.ipp.pt

Abstract. One of the great challenges in programming education is to keep students motivated while working on their programming assignments. Of the techniques proposed in the literature to engage students, gamification is arguably the most widely spread and effective method. Nevertheless, gamification is not a panacea and can be harmful to students. Challenges comprising intrinsic motivators of games, such as graphical feedback and game-thinking, are more prone to have longterm positive effects on students, but those are typically complex to create or adapt to slightly distinct contexts. This paper presents Asura, a game-based programming assessment environment providing means to minimize the hurdle of building game challenges. These challenges invite the student to code a Software Agent to solve a certain problem, in a way that can defeat every opponent. Moreover, the experiment conducted to assess the difficulty of authoring Asura challenges is described.

Keywords: Games · Gamification · Authoring · Learning · Programming · Competitive · Graphical feedback

1 Introduction

Motivation is what makes you try to do something [19]. One who feels unable to do an activity, or fails to value it and its outcome, is highly likable to lack motivation to engage in the activity, in which case he/she is said to be amotivated. In education, loss of motivation is one of the most outstanding problems. When students are amotivated, they tend to care less about educational activities and to stop striving to complete them. In programming courses, this results in an unsustainable lack of practice which is accompanied by recurring failures in assessments and later ends up in student dropout [1,6].

Several approaches have been proposed to mitigate this problem, such as problem-based learning [15,17], storytelling [8], pair programming [6],

M. E. Auer and T. Tsiatsos (Eds.): ICL 2018, AISC 916, pp. 602–613, 2020.
https://doi.org/10.1007/978-3-030-11932-4_57

competition-based learning [2,13], and gamification [7,21]. Undeniably, the most widespread approach is gamification, which consists of using game elements and mechanics to engage students. The most common gamification methods typically add extrinsic motivators, such as leaderboards, badges, or levels, to an existing learning environment. Even if these elements can increase students' engagement temporarily, they neither foster correct changes in attitudes and behaviors or longtime commitment. On the contrary, they undermine the intrinsic interest that one might have to perform a task. Completing the activity becomes a means to obtain the reward rather than to develop skills. Moreover, the best way to solve the task is the first to come to mind, taking risks or exploring are not options [10].

Gamification techniques with enduring effects typically rely on different game aspects, such as graphical feedback, game-thinking, collaboration, and in-game challenges. Another well-explored aspect of games in programming learning is competition. Despite the fact that it may have considerably harmful effects [9], it is also true that new graduates are increasingly facing programming contests after leaving universities, as a part of the recruitment process for top technology companies. Hence, a promising approach might be to combine these features. In fact, there are already some attempts to bring competitive games into learning, which challenge the student to code the Software Agent (SA) that controls the player and competes against other SAs. For instance, IBM CodeRally – a Java game-based car rally competition presented at the 2003 ACM International Collegiate Programming Contest (ICPC) World Finals – and Robocode – a Java-based virtual robot game – have been found to have a great potential to catch students and non-students attention [14,16]. Nevertheless, building these challenges involves a complex and time-consuming process which most times educators are not willing to perform.

This paper presents the authoring component of Asura, a game-based programming assessment environment that challenges students to code competitive SAs for a game. Asura is developed on top of Enki [18] – an existing pedagogical environment of Mooshak 2 [12] – and Mooshak 2 itself. The final goal of Asura challenges is to win a tournament among all submitted SAs, at the end of the submission time. The authoring component of Asura, named Asura Builder, is one of its key features. This component aims to provide a simple way of authoring new challenges. As a benchmark, the reference is the well-known ICPC problems whose automated assessment requires coding both a solution program and a test case generator. The goal is to keep the effort of authoring an Asura challenge similar to that of authoring an ICPC problem.

The development of new game-based challenges requires the teacher to extend an existing referee (i.e., manager) to implement the game rules and inquire SAs for their actions, implement the state interface that helps updating the game state, add a problem statement, and upload some image assets for the game. However, more complex challenges may demand the specification of wrappers for SAs submitted by students. Also, teachers can add their own SAs, so that

students can do matches against them since the beginning. There is no limit on the number of SAs submitted by teachers.

The remainder of this paper is organized as follows. Section 2 reviews the state of the art on authoring tools for game-based challenges. Section 3 presents Asura, its concept and architecture. Section 4 details the component for authoring challenges. Section 5 describes the validation of Asura Builder, which aims to assess the increase in difficulty of building an Asura challenge when compared to that of creating an ICPC-like problem. Finally, Sect. 6 summarizes the main contributions of this paper and discloses the next steps on the presented work.

2 State-of-the-Art

Asura is a game-based assessment environment that aims to offer the teacher a way to motivate students to program and overcome their difficulties through practice, requiring a similar amount of effort to that of creating an ICPC-like 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 submitted SAs.

To the best of authors' knowledge, there is no tool to author challenges with these features. Therefore, the next subsections review the state-of-the-art on authoring tools for game-based challenges and present some environments similar to Asura, highlighting their extensibility for new challenges.

2.1 Authoring Tools for Game-Based Challenges

Even though tools to author game-based challenges are scarce, there is much research interest in such tools. Most of the attempts to create platforms for authoring game-based challenges focus on simple, yet attractive, characteristics of games, particularly the storyline, which are easy to adapt to heterogeneous contexts. StoryTec [5] is a digital storytelling platform for creating and experiencing interactive (i.e. non-linear) stories. It encompasses a story editor, an action set editor, a property editor and an asset manager. The story editor manages the structure of the story using an hierarchically organized graph consisting of scenes. The stage editor is a tool similar to a level editor, present in some video games, that enables scene creation using a drag-and-drop interface to insert objects from the assets manager. The action set editor is a visual environment, based on the UML activity diagram, for defining the possible actions and constraints of every scene. The asset manager is where the author can import various types of assets, such as cameras, lights, and models.

<e-Adventure> [20] is an authoring tool for story-driven educational games, that aims to introduce games in the learning process. It supports the creation of *third-person* and *first-person* adventure games by instructors with little or no programming background. Furthermore, it complies with standards and specifications of the e-learning field, allowing to export, modify, and reuse games as learning objects. The tool is an all-in-one game creator with game objects organized by types (e.g., scenes, items, conversations, etc.) which the author can add into the game.

Although these systems were designed for a pedagogical purpose, neither StoryTec or <e-Adventure> are specific for learning to code. Greenfoot system [11] is an interactive object world that provides a framework and an environment to create interactive 2D simulations. On the side of the solver, it features a full-fledged Integrated Development Environment (IDE), including code editing, compilation, object inspection, and debugging. Moreover, Greenfoot also offers graphical feedback to visualize the appearance, location, and rotation of the simulation objects as well as methods to directly interact with them.

2.2 Competitive Game-Based Programming Environments

SoGaCo [4] is a scalable web environment that evaluates competitive SAs, developed in several programming languages, that play simple mathematical board games. Its interactive GUI allows learners to see step-by-step how their programs play the game. Furthermore, it not only promotes competition among students but also collaboration, allowing them to share their SAs through a single bot address (URL).

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 several board games, such as PrimeGame, Mancala and Othello.

CodinGame[1] is a web-based platform with several puzzles that learners can solve 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 provide a 2D 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 more than one language. Once the exercise is solved, players can access, rate, and vote on the best solutions.

The platform enables any user to contribute with programming challenges through a dedicated form. Nevertheless, it only allows them to create challenges based on input/output test cases without any game-like graphical feedback.

[1] www.codingame.com.

3 Asura

Asura is an environment for assessment of game-based programming challenges. The main goal of this environment is to minimize the hindrance of creating new game challenges, while allowing students to enjoy unique features of games, such as graphical feedback, game-thinking, and competitiveness. In Asura, students are challenged to develop an SA to play a game. This SA has the final objective of beating every other SAs in a tournament following a similar structure to those organized on traditional games and sports.

During the development of the SA, students can validate its effectiveness by executing matches against any previously submitted SAs that can be selected from a board in Enki. A match runs on the Asura Evaluator, which evaluates the code of the SA, starts a process with it, and leverages the rest of the evaluation on the game manager. The outcome of the match is a JSON object adhering to the JSON Schema[2] defined for a game movie. This object is given to the Asura Viewer by Enki, which transforms it into an adequate format to display to the learner. Figure 1a) presents the result of a validation of an SA against an opponent in a Bullseye Shooting game being displayed in the Asura Viewer integrated into Enki.

Tournaments can be organized by educators, once the time to code SAs ends. For that, they can use the wizard added to the administrator interface of Mooshak 2, shown in Fig. 1b). This wizard enables the educator to choose among the set of all accepted submissions, determine how much points are awarded per match, and define the structure of the tournament stages. After configuring the format of the tournament, the Asura Tournament Manager organizes and runs the matches of the tournament on the Asura Evaluator. A tournament produces JSON data adhering to the JSON Schema[3] defined for a tournament, which contains a reference to each match's movie, organized by stages and rounds, as well as partial and complete rankings of each phase. This data is presented in an interactive Graphic User Interface (GUI) to the students, allowing them to request the matches they want to see and check details of each phase. For instance, Fig. 1c) displays the interactive GUI of a tournament of Slalom Skying in a knockout phase, whereas Fig. 1d) presents the player's path after clicking in the player's name on the interactive GUI.

The architecture of Asura, depicted in Fig. 2, is composed of four distinct components, namely the Viewer, the Builder, the Evaluator, and the Tournament Manager. These components interact with tools already described in the literature, particularly Mooshak 2 and Enki. The Evaluator is a small package developed inside Mooshak 2, whose main class is the `AsuraAnalyzer`. This class is a specialization of the `ProgramAnalyzer`, the main analyzer of Mooshak 2 which is responsible for grading submissions to ICPC-like problems, for conducting the dynamic analysis using the provided JAR package. Both analyzers implement a common interface – `Analyzer` –, that allows evaluator consumers

[2] https://mooshak2.dcc.fc.up.pt/asura/static/match.schema.json.
[3] https://mooshak2.dcc.fc.up.pt/asura/static/tournament.schema.json.

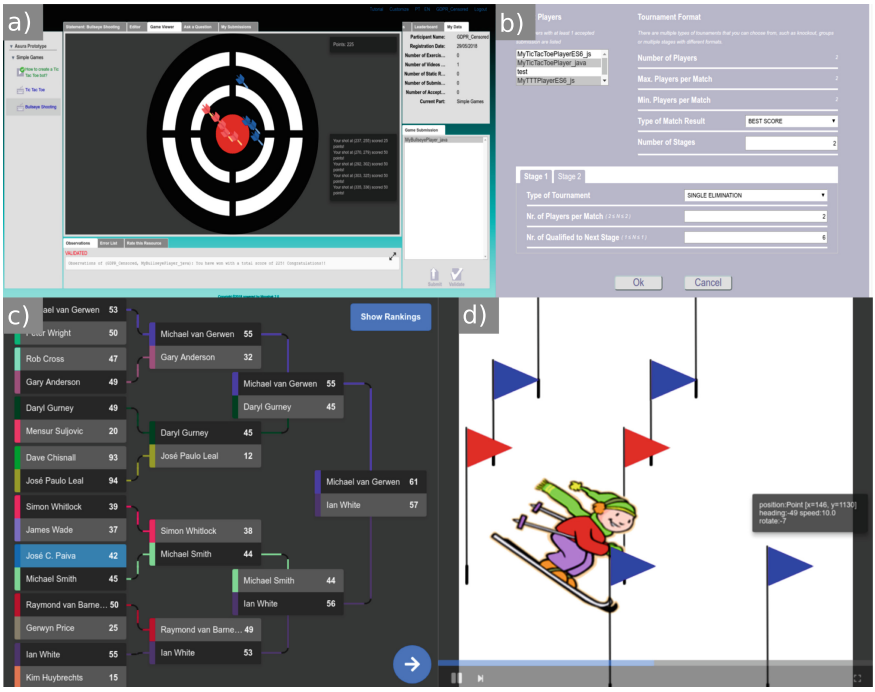

Fig. 1. Screenshots of the various components of Asura

to integrate seamlessly with any of them. In this case, the consumer is Enki. The Tournament Manager handles the set up and execution of the tournament, integrating with the Evaluator and Mooshak 2 administration GUI. The Viewer is an external Google Web Toolkit (GWT) widget that can integrate in any GWT environment, supplying it with an interface `TournamentMatchViewer` to enable them to display either tournaments or matches. Finally, the Builder is an independent component that produces the JAR package used by the Evaluator.

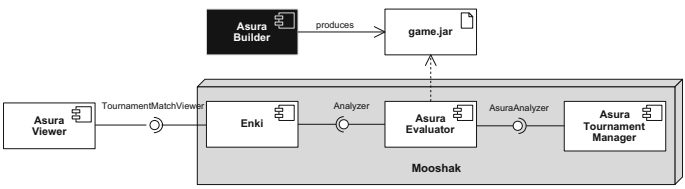

Fig. 2. Diagram of components of the architecture of Asura, highlighting Asura Builder

4 Asura Builder

The Asura Builder is an independent component composed of multiple tools dedicated to the authoring of game-based coding challenges, including a Java framework and a Command-Line Interface (CLI) tool. The Java framework 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 players in several 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. Even though the authors are required to program the challenges in Java, players can use their preferred programming language to code their SAs.

Each of the next subsections describes a sub-component of Asura Builder. Subsection 4.1 describes the builder of graphical feedback. Subsection 4.2 details the referee and state management of the game. Subsection 4.3 presents the communication between the game manager and the players. Subsection 4.4 introduces the two kinds of wrappers that Asura supports to facilitate SA development. Subsection 4.5 provides an overview of the CLI tool.

4.1 Game Movie Builder

Most of the necessary effort for building video games is spent on graphics. They determine the players' first impression on the game and they provide the best feedback of the actions executed during the game. Asura games are not exceptions. In Asura, graphics are abstracted as a game movie, which consists of a set of frames, each of them containing a set of sprites annotated with information about their location and transformations. In this way, the representation of the game movie is very compact since each frame is just a collection of objects, completely described by four numbers. Besides that, a movie also includes metadata information, such as `title`, `background`, `width`, `height`, `fps` – number of frames to display per second –, `anchor_point` – sprite point relative to which coordinates are given –, the set of `players`, and the set of `sprites`.

This abstraction facilitates the construction of graphical feedback by defining a standard way to describe it, independently of the game. Furthermore, Asura Builder offers an interface (and an implementation) to easily build these game movies. The interface provides methods to manage metadata information, add frames, insert and transform items, include messages to players (e.g., logs of their SAs), set the observations and classification of a player, push and pop frame states from a stack, terminate the game movie indicating an error in the Builder component or an error in one of the players, among many others. Updates to the game movie are performed during the game state management.

There is no distinction between game movies built for validations or game movies created during the tournament, so the author does not need to change anything to execute tournaments.

4.2 Game Manager

Every game needs a controller (or referee) to ensure that the game rules are followed. The controller keeps the global state of the game, decides which player takes the next turn, declares a winner, among many other tasks. In the Builder component, these tasks are the responsibility of the `GameManager` who acts as the referee of the game.

The abstract manager provided by the framework connects to the input and output stream of the players' processes to receive their actions and update them with changes on the game state. Specialized managers determine the playing order and manage the game state accordingly. Some of these specialized managers, such as a turn-based game manager, are provided by the framework and can be easily integrated in a new challenge, using the CLI tool.

In order to manage the game state, which is unique to each game, the controller leverages on the `GameState` interface. This interface specifies methods to initialize the state before the game starts, update the game state according to the action of a concrete player, obtain the object that needs to be sent to a certain player to update it about changes to the game state, end the round when all players' commands were executed in that round, finish the game and declare a winner, and much more. Most of these methods receive a game movie builder as parameter, allowing the state object to manage the movie, reflecting any changes made to it.

As a referee, when a player breaks the rules of the game (e.g., takes too much time to play, does an invalid action, does not meet the communication protocol, etc), it must act. If the violation of the player prevents the game from continuing, the game ends marking the infraction of the SA in the game movie. Otherwise, the game proceeds but the faulty SA gets a "Wrong Answer" at the end.

4.3 Communication Manager-Players

The communication between the `GameManager` and the players can be done either through JSON or XML. The `GameManager` sends state updates to the players, containing a type, which identifies the state, and a comprehensive description of the current game state. The messages sent from the players to the `GameManager` contain the action (a command) that the player wants to execute as well as a list of messages for debugging purposes. This communication is handled without any action of the author or players, meaning that they are not aware of the type of messages being exchanged. Yet, the author knows that there is a channel that sends and receives objects. Figure 3 presents the structure of the data models that are exchanged during the game.

4.4 Wrappers

Wrappers are sets of functions, provided by both the framework and the author, that aim to give players an higher level of abstraction, so that they can focus

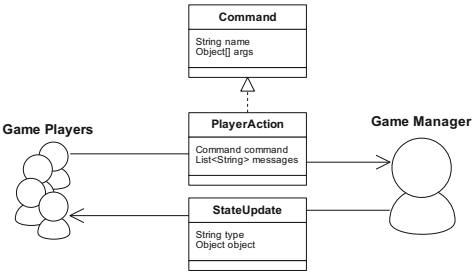

Fig. 3. Diagram of the communication between the manager and the players

on solving the real challenge instead of processing I/O. There are two types of wrappers: global and game-specific.

Global wrappers are defined by the Asura Builder framework. They implement functionality that is common to players of any game in a certain language. This includes methods to read and write JSON, log messages, and call the functions on the abstract and concrete players that implement the game/player-specific functionality.

Game-specific wrappers are provided by game authors and "extend" global wrappers with functionality related to a specific game. For instance, they can implement functions that process state updates, get and set values, or send actions. Besides that, they implement the player lifecycle, i.e., the player loop that reads updates or executes actions.

4.5 CLI Tool

The Command-Line Interface (CLI) tool is a command-line utility, based on cookiecutter,[4] one of the many existing CLI generators. A project with the codebase for an Asura game can be generated with a single command line `asura-cli --generate`. This command makes a series of queries to the user in order to obtain the required information to generate the project. The project generated by this utility is a Maven project containing a skeleton of a Game Manager and a Game State as well as the structure for SAs, wrappers, and skeletons.

Furthermore, the utility also imports pre-built game managers (e.g., `asura-cli --import-manager turn-based`) from a collection, including a turn-based game manager in which players act by turns, an all-at-once game manager in which players act all at the same time, among others. Support for new programming languages can also be managed through the CLI using the commands `asura-cli --add-language <language>` and `asura-cli --remove-language <language>`.

The deployment phase is also supported by the CLI tool. To package the game, the author can use the command `asura-cli --package`. A sample problem statement can also be generated using `asura-cli --add-statement`.

[4] https://github.com/audreyr/cookiecutter.

5 Validation

An experiment to validate the Asura Builder was conducted in an open environment with undergraduate students of the Department of Computer Science of the Faculty of Sciences of the University of Porto, enrolled either in the *First Degree in Computer Science* or the *Integrated Master of Science in Network Engineering and Information Systems* programs. This experiment aimed to assess the usefulness and ease of use of the Builder component on the authors' perspective only. However, students themselves played the role of authors. All students had an average background in Java acquired during the current semester in a Software Architecture course, and had no previous knowledge of Asura.

The experiment consisted on authoring both an ICPC problem and an Asura challenge, following one or more of the provided statements A,[5] B,[6] or C,[7] in increasing order of difficulty. These statements completely describe the challenges to be developed. Yet, they do not constraint the quality of the graphical feedback provided in the Asura game, which were left to the creativity of the authors. At the date of the experiment, the CLI tool was not finished yet. Thus, a Maven archetype was used instead to generate the project, requiring the authors to configure Maven before starting.

At the end of the experiment, students were asked to fill-in an online questionnaire to measure the user acceptance of the Builder. The questionnaire follows Davis' model [3] for evaluating perceived usefulness and ease of use of a system in a 7-value Likert scale, extended with questions about time spent in each type of problem, multiple-choice questions to compare the difficulty of authoring both types (in a 7-value Likert scale), and open text questions to identify weaknesses and strengths, and provide suggestions.

The global results indicate a perceived usefulness of 95.24% and a perceived ease of use of 76.19%. Regarding the comparison of developing an Asura challenge against creating and ICPC-like problem, a value of 73.81% was obtained (highest values are better). Nevertheless, statements regarding difficulty and time had a below average score, such as *I'm capable of developing an Asura challenge faster than an ICPC problem* (28.57%) and *It is easier to develop an Asura challenge than an ICPC problem* (71.42%). When asked to rate, in a Likert scale, the sentence *The additional hurdle of developing an Asura challenge is something that we can disregard considering the gains for students*, students agreed with 42.86%.

The free text answers highlighted the user acceptance of the Builder. For example, *The Framemork for the development of challenges is very flexible and it's mechanics are easy to understand.*

[5] https://mooshak2.dcc.fc.up.pt/asura/static/asura-validation-problem-a.pdf.
[6] https://mooshak2.dcc.fc.up.pt/asura/static/asura-validation-problem-b.pdf.
[7] https://mooshak2.dcc.fc.up.pt/asura/static/asura-validation-problem-c.pdf.

6 Conclusion

Motivating students to learn in practice intensive courses, particularly in programming courses, is challenging. The time dedicated to solve exercises is typically inadequate for the amount of knowledge they have to acquire and techniques they must master. As students start getting bad results, their interest in the learning activities diminishes. To mitigate this problem, it is necessary to find new educational methods that promote coding practice outside of the classes. One of such methods is to wrap learning activities as game challenges.

Even though games are already widely used in programming education, there is a lack of tools for creating them. This paper presents the authoring component of Asura, an environment for assessment of game-based programming challenges. The goal of this component is to minimize the hurdle of creating these challenges, making its difficulty similar to that of creating an ICPC-like problem.

The conducted validation, even though with a very low number of participants due to the final exams, has demonstrated the usefulness of Asura Builder and its ease of use. Nevertheless, there is still a long way to go to achieve the fast creation of game-based programming exercises. The results highlighted a significant difference in terms of time, when comparing the two types of problems. It is expected that the CLI tool can make the generation, addition of features, and deployment phases faster, but not that much. More improvements and extra features are needed, particularly in the game movie builder. The collected comments have also revealed the need to support multiple programming languages in the creation of games, which was already planned in a future release.

The next step is to validate the effectiveness of Asura as an environment to keep students motivated while working on their programming assignments.

Acknowledgments. This work is partially funded by the ERDF – European Regional Development Fund – through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) – as part of project UID/EEA/50014/2013.

References

1. Bennedsen, J., Caspersen, M.E.: Failure rates in introductory programming. SIGCSE Bull. **39**(2), 32–36 (2007)
2. Dagiene, V., Skupiene, J.: Learning by competitions: olympiads in informatics as a tool for training high-grade skills in programming. In: ITRE 2004, 2nd International Conference Information Technology: Research and Education, pp. 79–83 (2004)
3. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. 319–340 (1989)
4. 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, ASWEC 2015, vol. 2, pp. 8–12. ACM, New York, NY, USA (2015). http://doi.acm.org/10.1145/2811681.2811683

5. Göbel, S., Salvatore, L., Konrad, R.A., Mehm, F.: Storytec: a digital storytelling platform for the authoring and experiencing of interactive and non-linear stories. In: Spierling, U., Szilas, N. (eds.) Interactive Storytelling, pp. 325–328. Springer, Berlin (2008)
6. Han, K.W., Lee, E., Lee, Y.: The impact of a peer-learning agent based on pair programming in a programming course. IEEE Trans. Educ. **53**(2), 318–327 (2010)
7. 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. **7**(3), 291–301 (2014)
8. Kelleher, C., Pausch, R.F.: Using storytelling to motivate programming. Commun. ACM **50**, 58–64 (2007)
9. Kohn, A.: No Contest: The Case Against Competition. Houghton Mifflin Harcourt (1992)
10. Kohn, A.: Why Incentive Plans Cannot Work. Houghton Mifflin Company, Boston (1993)
11. Kölling, M., Henriksen, P.: Game programming in introductory courses with direct state manipulation. In: Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2005, pp. 59–63. ACM, New York, NY, USA (2005). http://doi.acm.org/10.1145/1067445.1067465
12. Leal, J.P., Silva, F.: Mooshak: a web-based multi-site programming contest system. Softw. Pract. Exp. **33**(6), 567–581 (2003)
13. Leal, J.P., Silva, F.: Using Mooshak as a competitive learning tool. In: The 2008 Competitive Learning Symposium (2008)
14. Liu, P.L.: Using open-source robocode as a java programming assignment. SIGCSE Bull. **40**(4), 63–67 (2008). http://doi.acm.org/10.1145/1473195.1473222
15. Lykke, M., Coto, M., Mora, S., Vandel, N., Jantzen, C.: Motivating programming students by problem based learning and lego robots. In: 2014 IEEE Global Engineering Education Conference (EDUCON), pp. 544–555 (2014)
16. Morris, C.L., Silberman, G.M.: Programming contests in academic environments. In: fie, pp. F1F7–7. IEEE (2003)
17. Nuutila, E., Törmä, S., Malmi, L.: PBL and computer programming-The seven steps method with adaptations. Comput. Sci. Educ. **15**(2), 123–142 (2005)
18. 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, pp. 332–337. ACM (2016)
19. Ryan, R.M., Deci, E.L.: Intrinsic and extrinsic motivations: classic definitions and new directions. Contemp. Educ. Psychol. **25**(1), 54–67 (2000)
20. Torrente, J., del Blanco, Á., Marchiori, E.J., Moreno-Ger, P., Fernndez-Manjn, B.: <e-adventure>: introducing educational games in the learning process. In: IEEE EDUCON 2010 Conference, pp. 1121–1126 (2010)
21. Utomo, A.Y., Amriani, A., Aji, A.F., Wahidah, F.R.N., Junus, K.M.: Gamified e-learning model based on community of inquiry. In: 2014 International Conference on Advanced Computer Science and Information System, pp. 474–480 (2014)