Learning Analytics Model in a Casual Serious Game for Computer Programming Learning

Adilson Vahldick(✉), António José Mendes, and Maria José Marcelino

Department of Informatic Engineering, CISUC, University of Coimbra, Coimbra, Portugal
adilson.vahldick@udesc.br, {toze,zemar}@dei.uc.pt

Abstract. Games have been used by teachers as a support tool to engage students in learning tasks. As they often record student's performance as learning progresses, it interesting and useful to discuss how that information can be used to assess learning and to improve the learning experience. For instance, teachers can use that information to give personalized attention in classes. In computer programming learning, games can provide an alternative way to introduce concepts and, mainly, to practice them. This paper proposes a model to identify the students' progress considering their performance in programming tasks. The model is demonstrated by an implementation in a casual computer programming serious game.

Keywords: Novice programmers · Learning analytics · Fuzzy systems

1 Introduction

Initial programming learning is known to be complex for many students. Games have been proposed to help students in their initial learning stages, namely to increase their motivation and engagement with the learning process [1]. Two approaches have been used: creating and playing games. In the first approach students are asked to develop small games in order to apply the programming concepts [2]. In the second approach the students play games to reinforce and practice concepts and programming skills [3]. The main idea is to motivate students to the learning activities, shortening the time between theory and practice, and bringing together abstract concepts and concrete activities.

Digital educational environments generate vast amounts of track data that could be used for the development of learning theories and applications [4]. Learning Analytics (LA) rely on data generated by the user's interaction with these environments. LA approach applied in educational games is an alternative to more traditional forms to evaluate learning [5] and it avoids to brake the game-flow experience risking to lose student's interest [6]. We only found in literature one study with LA applied in programming learning games [7]. It proposed a framework with six axes. A mathematical model, relating each axis to a variable, was created to implement this framework. The game rates the student considering each variable and normalizes the data based on a teacher defined ideal behaviour.

C. Vaz de Carvalho et al. (Eds.): SGAMES 2016, LNICST 176, pp. 36–44, 2017.
DOI: 10.1007/978-3-319-51055-2_6

In this paper, we propose a LA model applied in computer programming games focused in the student's performance, rating them automatically based on the performance of their classmates. The model was designed as a Fuzzy Logic Controller (FLC). Fuzzy Logic is closer to human thinking and natural language than other artificial intelligence approaches [8]. The system is modelled using linguistic terms and thus it is easy to represent human knowledge [9].

Casual games usually have smooth learning curves and their assignments are often short [10]. These aspects should also be considered in the design of serious games reducing the time needed to learn the game features and mechanics, and freeing more time to learn [11]. We developed a casual serious game for initial computer programming learning, called NoBug's Snack Bar, using a Blocks-Based Programming (BBP) approach. In BBP the program is constructed through assembling functional blocks [12]. The LA model was tested in this game.

Section 2 presents the design principles followed and the architecture of the FLC to design and implement the LA model. Section 3 describes briefly the developed game and Sect. 4 explains the proposed model. Section 5 demonstrates its implementation and the data gathered by this model. The final section concludes the paper.

2 Fuzzy Logic Controller and Design

The essential part of a FLC is a set of linguistic control strategies based on expert knowledge mapped into an automatic control strategy [9]. A basic configuration of a FLC is depicted by a block diagram such as that shown in Fig. 1.

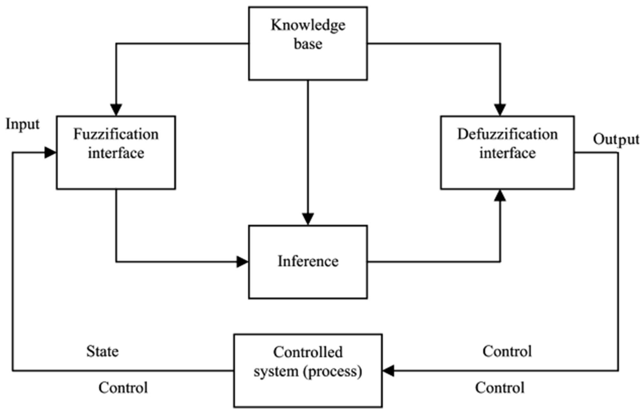

Fig. 1. Configuration of a FLC [9]

The controlled system represents a process that is regulated through a control action. The fuzzification interface is responsible for converting the input data (current state of the controlled system) into suitable linguistic values (fuzzy sets). The knowledge base module contains knowledge about all the input and output fuzzy partitions. The inference module simulates the human decision-making procedure based on fuzzy concepts, inferring fuzzy control actions to employ fuzzy implications and linguistic rules. The defuzzification interface converts the range of output values into the corresponding universe of discourse.

The design procedure of a FCL is divided in several steps as follows [8, 9]: 1-identification of the variables (states and controls); 2-normalization and partition of the variables space; 3-determination of the shapes of the fuzzy sets and their membership functions; 4-construction of the fuzzy rule base; 5-definition of the inference method; 6-determination of the defuzzification strategy.

There are many Fuzzy Logic software packages, as the MATLAB Fuzzy Toolbox and jFuzzyLogic [13]. jFuzzyLogic is an open source library written in Java that supports a Fuzzy Control Language (FCL) defined in the IEC-1131 specification. This specification defines the syntax and semantic of the FCL's components. jFuzzyLogic provides an API that interprets and executes a FCL program. It is also possible to define some or all members of a FLC through Java programming.

3 NoBug's SnackBar

NoBug's Snack Bar game mechanics are inspired in time management games. The player controls an attendant of a snack bar. Customers require some combination of foods and drinks, and the attendant must go to places where they are prepared, fetch them and serve them. The mission ends when the player meets all requests.

Figure 2 shows the game's interface. The animation area (on the left) shows the mission situation and shows the attendant behavior controlled by the player solution. The central area allows the construction of the mission solution. The player can run or debug her/his code. If she/he debugs, then the game shows the list of variables (at the right side of the figure) and runs one block at a time after each click of the debug button.

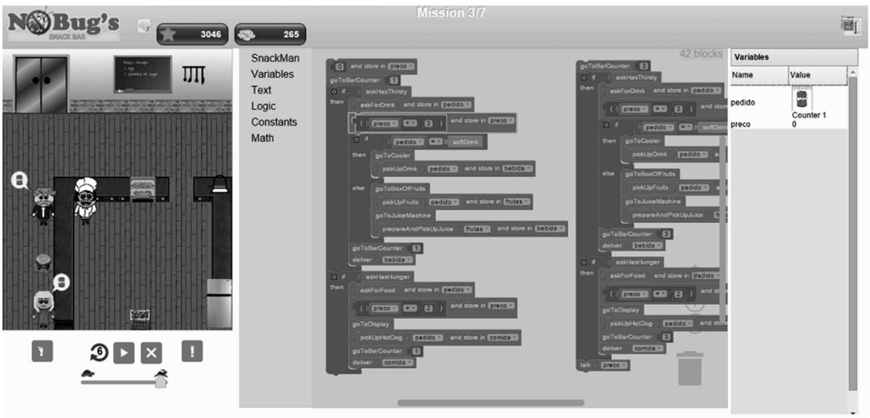

Fig. 2. Game interface

The game covers the initial topics usually included in introductory computer programming courses. It is divided in five levels with 55 missions: 1-Sequence actions (10 missions); 2-Variable manipulation (8 missions); 3-Conditionals (13 missions); 4-Loops (14 missions) and 5-Functions and arrays (10 missions). The first four missions in level one serve only to familiarize the student with basic interface of the game. That is the reason why we will not include them in our statistics in this paper.

4 LA Model in Computer Programming Learning Games

Following the FLC design procedure described in Sect. 2, our initial concerns were the definition of state and control variables, their partition in fuzzy subsets and the assignment of a membership function for each of them. The input variables of the proposed model are the missions' level and the time spent to solve them:

- **Mission:** classify the mission as introductory, development or mastery level.
- **Time Spent (TS):** is the accumulated time spent by the student to solve the last three missions. In our first experiments, we used the total time spent in the missions. However, after some tests, we verified that once a student had a bad performance in any previous mission, this was propagated for a very long time. Then we constrained it to the last three missions. This variable is partitioned into five subsets: *very fast, fast*, *normal*, *slow* and *very slow*. The subsets *very fast* and *very slow* are trapezoidal asymmetrical membership functions and the other three are trapezoidal symmetrical. The universe of discourse range varies according to students' experience. The students' performance in the game depends on several factors, such as the teaching methodology (learning content, assignments, etc.) and the previous programming knowledge or literacy (according to the region or country where the game is being used). To have a general model it is necessary to consider these divergences. We created a *Time Normalization* module to deal with these issues. This module assigns the membership function parameters dynamically, before it fuzzifies the input variables, performing 5 steps (Fig. 3). In the first step, the module retrieves from the game database the time spent in the previous three missions of each student using the Eq. 1:

$$TS_{(i,m)} = \frac{T_{(i,m-1)} + T_{(i,m-2)} + T_{(i,m-3)}}{3}. \tag{1}$$

where i denotes the student identification, $i = 1$ denotes the current player which the system is computing for, m denotes the current mission, $T_{(x,\ y)}$ denotes the time spent on mission y by student x, and $TS_{(i,\ m)}$ denotes the average time spent on the three missions before the m^{th} mission of student i. Thus, $TS(i, m)$ is the crisp value of the input variable TS. The second step identifies and removes students ($i >= 2$) with average time spent that are at least moderate outliers. The third step aims to create five clusters, one for each subset, of average times using the process of hierarchical cluster analysis (HCA) with the complete-linkage method [14]. The fourth step identifies the lowest (l)

and the highest (*g*) values on each cluster (*c1, c2, c3, c4, c5*) where *c1* has the lowest average time values and *c5* the highest values. The final step defines each membership function parameters (*veryfast, fast*, *normal, slow* and *veryslow*) as described in Eqs. 2, 3, 4, 5 and 6:

$$u_{veryfast}(x) = trape\left(x, 0, 0, c1(g), c2(l) + \frac{c2(g) - c2(l)}{2}\right). \tag{2}$$

$$u_{fast}(x) = trape(x, c1(g), c2(l), c2(g), c3(l)). \tag{3}$$

$$u_{normal}(x) = trape(x, c2(g), c3(l), c3(g), c4(l)). \tag{4}$$

$$u_{slow}(x) = trape(x, c3(g), c4(l), c4(g), c5(l)). \tag{5}$$

$$u_{veryslow}(x) = trape\left(x, c4(l) + \frac{c4(g) - c4(l)}{2}, c5(l), c5(g), c5(g)\right) \tag{6}$$

where $c_n(g)$ denotes the greatest value of cluster n, $c_n(l)$ denotes the lowest value of cluster n, and x denotes the parameter that is converted to a membership degree ($u_{membership}(x)$).

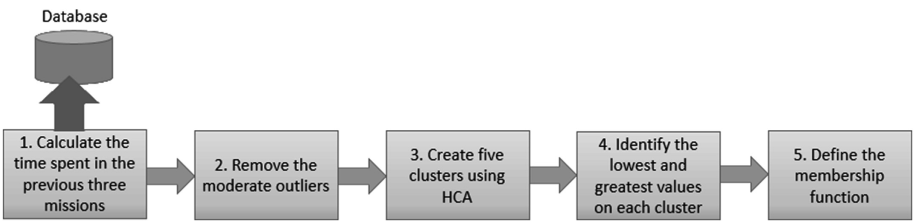

Fig. 3. Time Normalization module

The output variable is the *knowledge level* of the student. This variable is partitioned into three subsets (*bad*, *good* and *excellent*) and their membership function are triangles as defined in Table 1.

Table 1. Membership functions of the output variable *knowledge level*

Subsets	Membership functions
Bad	*trian (0, 0, 11)*
Good	*trian (10, 14, 18)*
Excellent	*trian (17, 20, 20)*

The next step of the FLC design is to define the inference method and form the rule base. The Mamdami inference method was adopted because it does not have nonlinear dynamic equations. The system rates a student according to the time she/he spends to solve the missions. Table 2 summarizes the rule-base, the relation between the two input variables and the output variable. When the player takes a long time to finish a mission, the model assumes that she/he has *bad* knowledge. On the other hand, the model rates the player as *excellent* when she/he finishes the mission *very fast*. In the other rules, the student classification varies according to the mission level. As the introductory missions presents new concepts and do not present challenges, it is expected that the player finishes them quickly. Yet the mastering missions are harder and full of constraints, really challenging the player.

Table 2. Fuzzy rule-base.

Mission	Time spent				
	Very slow	*Slow*	*Normal*	*Fast*	*Very fast*
Introductory	Bad	Bad	Bad	Good	Excellent
Development	Bad	Bad	Good	Good	Excellent
Mastering	Bad	Good	Good	Good	Excellent

Centre of Gravity is defined as the defuzzification method. Figure 4 shows the components relation of the proposed LA model. The ellipses are the input variables. The Time Normalization module accesses the database of the game and the current mission to define which is the time spent by the student and updates the knowledge base. The diamond designates the output variable.

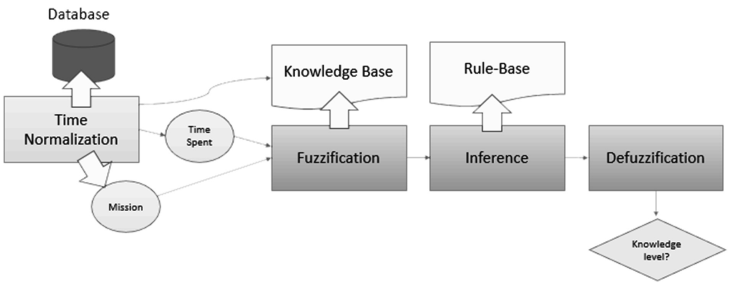

Fig. 4. LA Architecture

5 Implementation and Discussion

The proposed model was instantiated as a FLC in Java with jFuzzyLogic. The code below exemplifies the fuzzy rule-base by FCL. Nine rules were created to cover all the cells in Table 2. The variables definition was suppressed in the code because they were explained in the previous section.

LA model defined by IEC-FCL

```
FUNCTION_BLOCK nobugs_usecode
...
RULEBLOCK OnlyThis
  AND : MIN; OR  : MAX; ACT : MIN; ACCU : MAX;

  RULE 1 : IF TimeSpent IS verySlow THEN
                KnowledgeLevel IS bad;
  RULE 2 : IF TimeSpent IS fast THEN
                KnowledgeLevel IS good;
  RULE 3 : IF TimeSpent IS veryFast THEN
                KnowledgeLevel IS excellent;
  RULE 4 : IF Mission IS introductory AND
           TimeSpent IS slow THEN KnowledgeLevel IS bad;
  RULE 5 : IF Mission IS introductory AND
           TimeSpent  IS  normal  THEN  KnowledgeLevel  IS
bad;
  RULE 6 : IF Mission IS development AND
           TimeSpent IS slow THEN KnowledgeLevel IS bad;
  RULE 7 : IF Mission IS development AND
          TimeSpent  IS  normal  THEN  KnowledgeLevel  IS
good;
  RULE 8 : IF Mission IS mastering AND TimeSpent IS slow
                THEN KnowledgeLevel IS good;
  RULE  9  :  IF  Mission  IS  mastering  AND  TimeSpent  IS
normal
                THEN KnowledgeLevel IS normal;

END_RULEBLOCK
END_FUNCTION_BLOCK
```

We tested our game with 52 students. Figure 5 shows the results obtained in the first 15 missions, divided in introductory (1–7), development (8–11) and mastery (12–15).

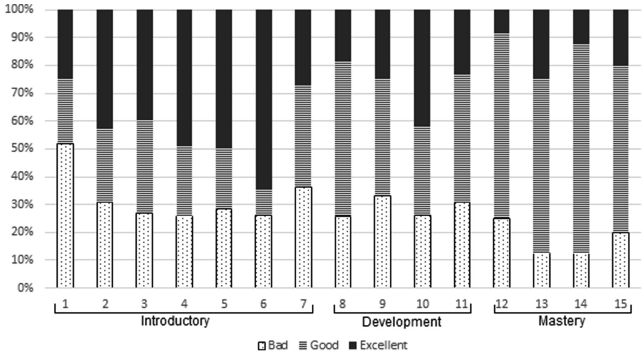

Fig. 5. Distribution of the students' knowledge classification

On average 30% of students are badly classified in introductory missions. This rate could alert the teacher or the game designers to review the missions. However, it is also observable that many students perform very well in the same missions. As the quantity of bad performing students is stable in introductory missions, maybe the teacher should address individually those students. As the students advance in the game, less of them are classified as excellent. This also happens frequently in the classroom: the very well performing students are a small part of the class.

6 Conclusions

Serious games are played in computer programming classes to motivate students overcome the initial natural barriers. However, to maximize the adoption of games in educational settings, it is important that teachers could track the overall progress of the students. In this paper, we presented a LA model based essentially on the time spent by the student to finish each mission. The model classifies the student (as bad, good or excellent) taking into consideration each mission level. We tested the model during a fist experiment. We found out that initially most students were classified as bad or excellent. However, as students advance in the game, they had a more similar performance and more students are classified as good. Although more experiments are necessary to evolve and validate the model, we believe this information can be used by teachers to adapt their lessons giving special attention to less performing students.

Acknowledgments. AV acknowledges the doctoral scholarship supported by CNPq/CAPES – Programa Ciência sem Fronteiras – CsF (6392-13-0) and authorized retirement by UDESC (688/13). We also want to thank the students that played the game and their teachers that allowed us to try it with them.

References

1. Vahldick, A., Mendes A.J., Marcelino, M.J.: A review of games designed to improve introductory computer programming competencies. In: 44th Annual Frontiers in Education Conference, Madrid, Spain, pp 781–787 (2014)
2. Bayliss, J.D., Strout, S.: Games as a “flavor” of CS1. In: 37th SIGCSE Technical. Symposium on Computer Science Education, Houston, Texas, pp. 500–504 (2006)
3. Barnes T., Powell E., Chaffin A., et al.: Game2Learn: building CS1 learning games for retention. In: 12th SIGCSE Conference on Innovation and Technology in Computer Science Education, Dundee, Scotland, pp. 121–125 (2007)
4. Greller, W., Drachsler, H.: Translating learning into numbers: a generic framework for learning analytics. Educ. Technol. Soc. **15**, 42–57 (2012)
5. Shute, V.J., Ke, F.: Games, learning, and assessment. In: Ifenthaler, D., Eseryel, D., Ge, X. (eds.) Assessment in Game-Based Learning. Foundations, Innovations, and Perspectives, pp. 43–58. Springer, New York (2012)
6. Chen, J.: Flow in games (and everything else). Commun. ACM **50**, 31 (2007)

7. Malliarakis, C., Satratzemi, M., Xinogalos, S.: Integrating learning analytics in an educational MMORPG for computer programming. In: 14th International Conference on Advanced Learning Technologies, ICALT 2014, pp. 233–237 (2014)
8. Jantzen, J.: Foundations of Fuzzy Control. Wiley, Chichester (2007)
9. Lee, K.H.: First Course on Fuzzy Theory and Applications. Springer, Heidelberg (2005)
10. Juul, J.: A Casual Revolution: Reinventing Video Games and Their Players. MIT Press, Cambridge (2010)
11. Landers, R.N., Callan, R.C.: Casual social games as serious games: The psychology of gamification in undergraduate education and employee training. In: Ma, M., Oikonomou, A., Jain, L.C. (eds.) Serious Games Edutainment Applications, pp. 399–423. Springer, London (2011)
12. Nor S., Mohamad H., Patel A., et al.: Block-based programming approach: Challenges and benefits. In: International Conference on Electrical Engineering and Informatics, Bandung, Indonesia, pp. 4–8 (2011)
13. Cingolani P., Alcalá-Fdez J.: jFuzzyLogic: a robust and flexible fuzzy-logic inference system language implementation. In: IEEE World Congress on Computational Intelligence, Brisbane, pp. 1090–1097 (2012)
14. Johnson, S.C.: Hierarchical clustering schemes. Psychometrika **32**, 241–254 (1967)