Computers in Human Behavior Reports 2 (2020) 100037

Contents lists available at ScienceDirect

Computers in Human Behavior Reports

journal homepage: www.journals.elsevier.com/computers-in-human-behavior-reports

A blocks-based serious game to support introductory computer programming in undergraduate education

Adilson Vahldick [a,*], Paulo Roberto Farah [a], Maria José Marcelino [b], António José Mendes [b]

[a] Departamento de Engenharia de Software, Udesc Alto Vale, Brazil
[b] CISUC, Department of Informatics Engineering, University of Coimbra, Portugal

ARTICLE INFO

Keywords:
Computer programming learning
Blocks-based approach
Serious games

ABSTRACT

Blocks-based environments have been used to promote computational thinking (CT) and programming learning mostly in elementary and middle schools. In many countries, like Brazil and Portugal, isolated initiatives have been launched to promote CT learning, but until now there is no evidence of a widespread use of this type of environments. Consequently, it is not common that students that reach higher education nowadays are familiar with CT and programming. This paper presents the development of a serious game to support the learning of basic computer programming. It is a blocks-based environment including also resources that allow the teacher to follow the student's progress and customize in-game tasks. Four cycles of experiments were conducted, improving both the game and how it was used. Based on the results of these experiences, the key contribution of this paper is a set of fourteen findings and recommendations to the creation and use of a game-based approach to support introductory computer programming learning for novices.

1. Introduction

Introductory programming learning demands a very practical approach from the students. It is not enough to read and understand the syntax of a programming language or even its control structures. It is necessary that the students create their own programs to solve problems: that is learning by doing (Gomes & Mendes, 2007). However, it is not enough to solve just a few problems. The student needs to engage in an intensive process of programs development focusing on a variety of situations (Robins et al., 2003). It is common that this process causes difficulties and frustrations, that need to be faced with persistence and dedication. The student must feel motivated to stay in this trajectory (Settle et al., 2014). Many tools and pedagogical approaches have been proposed to support the students in their learning process. Maintaining this motivation is the main appeal to use serious games (SG) to support learning, as the fun factor can help to keep the student playing when solving problems in the game (O'Neil et al., 2005).

Blocks-based programming became popular with Scratch (MIT Media Lab, 2020). In these type of environments, actions, manipulation of variables, and control structures are represented by coloured blocks that fit together, following a Lego metaphor (Kelleher & Pausch, 2005). This approach has been used to introduce programming to students due to the ability to visually map complex concepts, the accessibility of all available commands, the ease of use with the drag-and-drop technique, and the description of blocks with natural language (DiSalvo, 2014), (Weintrop & Wilensky, 2015).

The idea behind the use of games in educational contexts is to take advantage of the motivation created by an experience in which the students are out of their everyday setting, being immersed in challenges and being rewarded for their achievements (Whitton, 2010). Both the use of games (López et al., 2016) and its development (Chau et al., 2015) have been used as a strategy to promote learning. The development of logical thinking and problem-solving skills are important, but difficult for many students. The expression of solutions as programs often puts an extra burden on them, as they have an idea about the solution, but fail to express it using a programming language. Games can be used to learn programming if they support computational thinking (CT) and problem-solving skills development (Kazimoglu et al., 2013), (Malliarakis et al., 2017), but often a gap remains when students are asked to solve similar problems using a "real" programming language.

This paper describes the development of a new blocks-based SG to support introductory programming learning called NoBug's SnackBar. This research aimed to (i) identify requirements for an educational game to promote the learning of problem-solving techniques in introductory

* Corresponding author.
E-mail addresses: adilson.vahldick@udesc.br (A. Vahldick), paulo.farah@udesc.br (P.R. Farah), zemar@dei.uc.pt (M.J. Marcelino), toze@dei.uc.pt (A.J. Mendes).

https://doi.org/10.1016/j.chbr.2020.100037
Received 4 September 2020; Received in revised form 19 October 2020; Accepted 28 October 2020
Available online 9 November 2020

programming and (ii) to present a methodology to use that game integrated in an introductory programming course. This led to the definition of three research questions:

RQ1. What are the reasons for using serious games in teaching and learning problem-solving skills in introductory computer programming courses?

RQ2. What features should be part of a serious game to promote problem-solving learning in an introductory computer programming course?

RQ3. How can a serious game be integrated into introductory computer programming courses?

This SG aims to promote the development of computational thinking by creating blocks-based solutions to missions proposed to the student (player). In this environment the student focuses on the logic and structure of programming instead of the mechanics that involve writing textual programs (Kelleher & Pausch, 2005). Once the student has understood how a program works and practiced problem solving, she/he can be introduced to a programming language. The use of technology can make teaching work easier, but it is not enough to drive student learning. Games' potential can only be reached when the teacher is involved, to debrief and help the student transfer ideas out of the game to other situations (Sitzmann, 2011).

This paper is structured as follows. Section 2 presents a compilation of surveys that were used to identify opportunities and gaps for the development of serious games for programming learning. Section 3 describes the research methodology used in the project. Section 4 details the game description and the options that were made in each cycle of the game evolution. Section 5 shows how the teachers can customize the missions proposed to the students. Section 6 presents some lessons learned during the development that may be useful when designing this type of games. Section 7 includes some findings about the use of games in the context of introductory programming courses. Finally, section 8 will reflect on the implications of the results and offers some suggestions for further research.

2. Opportunities in serious games for programming learning

Before starting the development of a game to support programming learning, it was necessary to get familiar with other proposals described in the literature. Five surveys about serious games with similar objectives were consulted to gather a better knowledge about what has been proposed and to look for gaps that could be used as a subsidy to guide the game development.

Malliarakis, Satratzemi and Xinogalos (Malliarakis et al., 2014) analysed 12 games. They pointed out three recommendations: (i) create environments in which the teacher can configure the game to meet her/his needs that always end up depending on her/his pedagogical goals; (ii) monitoring interactions and student progress; and (iii) allow collaborative and cooperative work so that students can help each other in completing tasks.

Vahldick, Mendes and Marcelino (Vahldick et al., 2014) analysed 40 games described in the literature and available online. The gaps identified in this work refer to flexibility and adaptability, so that more teachers can use the games in different contexts and needs. This raised four issues: (i) games could accept more than one representation metaphor, for example visual and textual, and more than one programming language so that the teacher can configure the environment according her/his needs and context; (ii) games should evolve according to the student's progression: initially using a visual language, but at a later stage continue with a conventional programming language; (iii) games must support learning during the resolution of tasks, and not only after their submission or execution; (iv) make the code available with open source licenses, as most games mentioned in the literature are inaccessible, making it impossible to use them as a basis for further research and development.

Harteveld et al. (Harteveld et al., 2014) analysed 36 games described in the literature or available online. The focus of this survey was to identify the most common features in the games. Based on this, they identified some needs, mostly focused on the inclusion of girls: (i) problem solving focusing on collaboration and social aspects (helping another player to overcome their challenges); (ii) games must have clear definition of learning objectives and consider more application of educational philosophies; (iii) increase creativity in terms of gender differentiation, promoting aesthetic aspects for girls; (iv) include narratives, that is, the stories that guide the game should include interactions with other characters and use other mechanics instead of simply solving puzzles.

Miljanovic and Bradbury (Miljanovic & Bradbury, 2018, pp. 204–216) used 49 games identified in the literature or available online. The survey also aimed to analyse the games to identify the most evident common features, and those that may increase the game's adoption. First, the authors consider opening the source code to allow access to the games and their modifications to serve more teachers. Like two other surveys already mentioned, it is also stressed the importance of fostering collaboration and competition within the game. The authors state that in general the game's coverage of the ACM reference curriculum for Fundamental Concepts in Programming is weak. Finally, they found that games do not pay enough attention to the adoption of the best accessibility and inclusion practices.

Shahid et al. (Shahid et al., 2019) analysed 41 articles to identify the gaps that exist in the literature. They concluded that most of them emphasize mechanics and dynamics more than aesthetics, making the games tiring and boring. They pointed out the lack of games directed to the undergraduate level and the incomplete coverage of the items in the ACM curriculum. They also recommended considering the student's previous skills and knowledge and providing automatic feedback when the error occurs.

Based on the conclusions of these surveys, Table 1 lists the recommendations identified.

Table 1
Opportunities to develop serious games.

#	Opportunity	References
01	Teacher configurable environments.	Malliarakis et al. (2014)
02	Monitoring student tasks.	Malliarakis et al. (2014)
03	Collaboration, competition and social aspects.	(Malliarakis et al., 2014) (Harteveld et al., 2014) (Miljanovic & Bradbury, 2018, pp. 204–216)
04	Accept more than one way to solve problems and more than one programming language.	Vahldick et al. (2014)
05	Start with visual strategies for beginners and follow with conventional programming languages as the student progresses.	Vahldick et al. (2014)
06	Provide instructional feedback during the game, not just at the end.	(Vahldick et al., 2014) (Shahid et al., 2019)
07	Open the source code of the games developed.	(Vahldick et al., 2014) (Miljanovic & Bradbury, 2018, pp. 204–216)
08	Clarify the learning objectives and the educational theories used.	Harteveld et al. (2014)
09	Gender differentiation.	Harteveld et al. (2014)
10	Mechanics that include narratives and interaction with other characters, instead of solving puzzles.	Harteveld et al. (2014)
11	Cover all ACM curriculum topics.	(Miljanovic & Bradbury, 2018, pp. 204–216) (Shahid et al., 2019)
12	Adoption of best accessibility and inclusion practices	Miljanovic and Bradbury (2018)
13	Consider the importance of aesthetics.	Shahid et al. (2019)
14	Target audience: undergraduate students	Shahid et al. (2019)

3. Research methodology

From the work described in the previous section three

recommendations were identified to guide the development of NoBug's SnackBar:

1. Do not use the common approach of moving turtles and robots. The game should explore a different approach offering a more meaningful context to undergraduate students, possibly part of their daily life, like a snack bar. Consequently, the commands should represent higher level actions: instead of asking the hero to move a step or turn left, it should be asked to go to a customer or to prepare a hotdog. Higher level commands allow the students to program more complex tasks with a small number of commands, keeping the task solution closer to how they usually think about it in a real situation. For example, it is common to think about "going to the fridge" and not "repeat step until you reach the fridge";
2. Self-learning: students should be able to play and learn without any teacher intervention. The game should lead the student's development through a learning sequence, correcting their solutions while they are playing;
3. Before students create their own solutions in the game, they should experience good programming practices, for example assessing partial or complete solutions to example tasks.

The game research and development process followed the steps indicated in Fig. 1. This model is an adaptation of Design-Based Research (DBR) and the serious games project by Marfisi-Schottman et al. (Marfisi-Schottman et al., 2010).

Design-Based Research (DBR) is concerned with the pragmatic application of learning theories, aiming to narrow the relationship between theory and practice, through tangible learning examples and artefacts, which can be used and reused in the real world (Cocciolo, 2005), (Squire, 2005). Researchers are interested in demonstrating the usefulness of technological means, making the connection between effective learning, the curriculum design and the most sensible way of using them. In addition to the iterative and incremental cycles proposed in DBR, the model considered the process of developing serious games proposed by Marfisi-Schottman et al. (Marfisi-Schottman et al., 2010), which is suitable for minimalist teams and explores pedagogical theories applied to games, in order to create the best teaching and learning conditions.

The research was developed in four cycles, one per semester. The experience did not limit students either in time or space, as it was conducted considering the free use of the game at any time and place, allowing us to possibly evaluate results closer to reality in terms of flow and learning. The artefact that resulted from this research and its development process will be presented in the next section.

4. NoBug's SnackBar: the game and its evolution during the research

4.1. Main ideas behind the game

In the beginning of NoBug's SnackBar design and development process a few decisions were made. It was designed as a web-based game inspired in time management games. The player (student) should control the attendant of a snack bar. Customers (controlled by the game) would make some request (a combination of foods and drinks), and the attendant should perform the necessary steps to fulfil the request. The mission would end when the player fulfils a certain number of requests.

It was decided to use a code metaphor based in dragging and dropping blocks, similar to Scratch. The idea was to avoid using the particularities of any text-based programming language. The game included a set of commands (blocks) to control the attendant and allow him/her to perform the tasks necessary to fulfil the customers' request. There were also blocks to represent variable manipulation and control structures (conditionals and loops). Fig. 2 shows a part of a mission solution and its environment configuration. The objective of this mission is to serve a customer sitting in chair 2 of the counter (represented by A in Fig. 2). He may want to drink a juice or a soda. The service ranges from asking what the customer wants to the order delivery. The first block means that the attendant goes to the position 2 of the bar counter. Then the customer order is stored in a variable called order. The third block represents a conditional and compares the value of order with the constant softDrink. If they are equal, the attendant goes to the fridge (B), takes the drink according to order, and stores it in the variable drink. Otherwise, she/he goes to the fruit box (C), picks the fruit according to order and stores it in the variable fruits. Then she/he goes to the juice machine (D), prepares the juice and stores it in variable drink. After the conditional block, the attendant returns to the customer and delivers the drink. If the delivery matches the order, the player earns "money", and accomplishes the

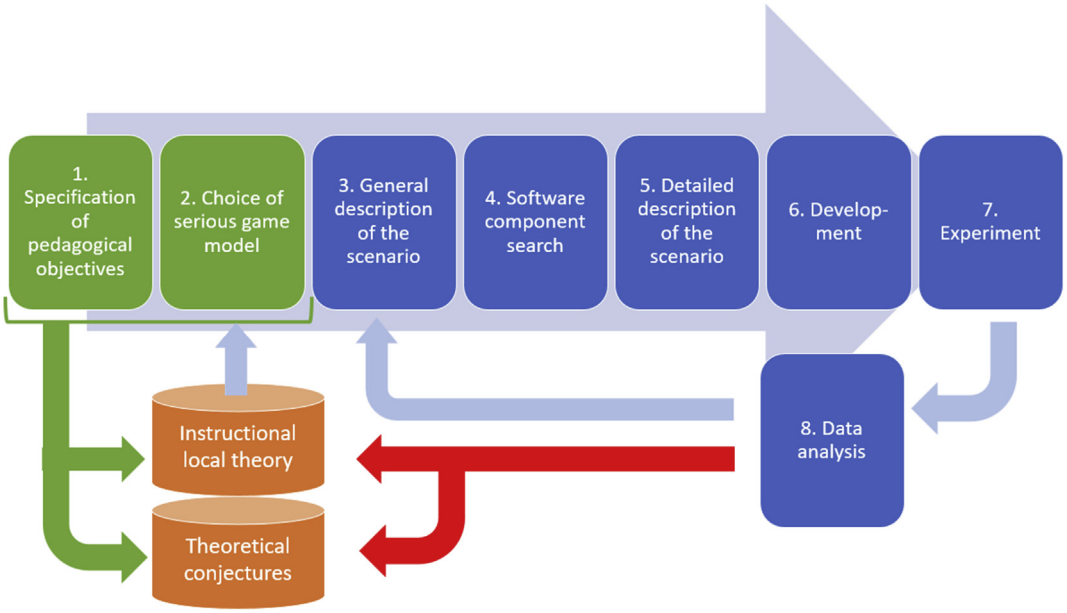

Fig. 1. Model of development and research process.

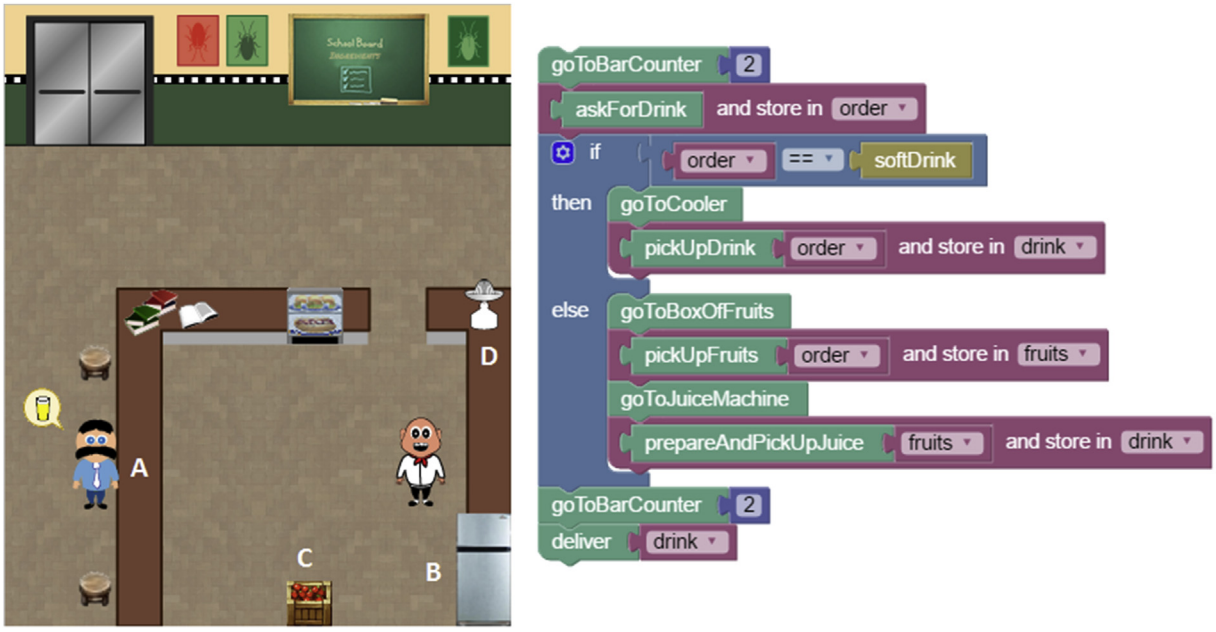

Fig. 2. Example of a mission.

mission goals. Otherwise, the customer gets angry, the player receives an error message, the execution ends, but the student can try again.

4.2. First cycle – pilot test

The goal of the first cycle was to identify basic mechanics and dynamics for the game and select software components that could support these choices. First, a paper prototype (Fig. 3) was developed and evaluated to validate the general game scenario and its types of missions. This prototype was presented to a group of 19 programming students enrolled in the second semester of an undergraduate Informatics Engineering Course. Half of these students had been approved in the introductory programming course in the previous semester. They were asked to solve a simple mission using the paper prototype. After that most of the students stated that they would like a game with these characteristics to be used as a learning resource together with other activities in their programming courses. The blocks system allowed students to solve the proposed problem, regardless of mastering the syntax of a programming language.

A digital prototype was also developed to validate the component that would be used for the construction of resources with blocks. Blockly (Fraser, 2015, pp. 49–50) was used, as it transforms blocks into Javascript code, allowing it to be executed in the browser without the need for compilation and execution on the server side, reducing thus latency by avoiding the transmission between client and server. Fig. 4 shows the

Fig. 3. Paper prototype: (a) – snack bar; (b) – solution by blocks.

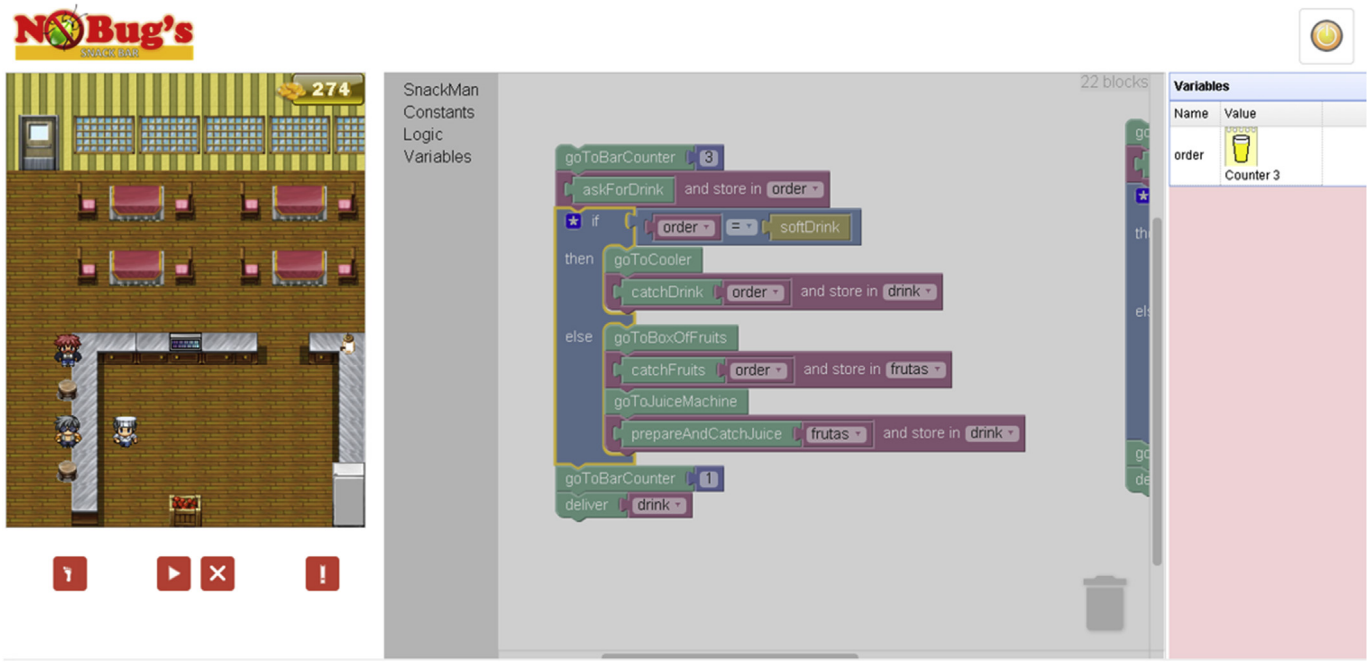

Fig. 4. Pilot version of the main page.

main screen of the pilot version. The key ideas were followed during the whole development. The player creates the solution for the mission in the central area. The animation area (on the left) allows the observation of the customers and the attendant movements based on the proposed solution code. The players can run or debug their code. If they debug, the game shows a list of variables and its values (on the right) and runs a block after each click of the debug button.

The prototype was validated in two sessions, involving two groups of students, who were asked to play 16 tasks organized into missions involving manipulating variables and decision structures. These sessions took 1.5 h each. One of the groups consisted of 19 students from the Master of Education of University of Coimbra and another included 16 students from the first semester of the undergraduate course in Software Engineering of University of Santa Catarina State. This experiment aimed to assess how students with such different characteristics would behave with the game. Through log analysis, students' performance and difficulties were assessed. Fun was evaluated using a survey called EGameFlow (Fu et al., 2009). At the end of this cycle, all the eight steps of the development model described in section 3 were completed for the first time, concluding with the definition of a serious game model for this research.

Conclusions of cycle 1. Students liked the game: the final average of the answers to EGameFlow was 3,8 (in 5) for the Master Education students and 4,1 for Software Engineering students. The lowest results were found in the dimensions of autonomy (3.4) and immersion (3.6) for master's and undergraduate students, respectively. The feeling of loss of control (autonomy dimension) was probably due to the number of topics the students had to learn (sequences, variables, decision structures) during this 1.5 h playing session. This highlighted the need to invest in a feedback system that could help to avoid the player's inertia and provide supporting material to explain basic programming topics. Immersion refers to the degree of involvement, engagement and integration of the player with the game (Brown & Cairns, 2004). Several items were identified that might improve immersion in the game: a scoring system, background music, sounds, a storyline, using own pictures and images, allowing the character to be personalized and developing simpler missions that have solutions with few blocks.

4.3. Second cycle – first version with class integration

In the second cycle, 60 first semester undergraduate students of the Design and Multimedia Course of University of Coimbra were involved, over a period of three months. The game had 57 missions, involving variables, decision structures, repetition structures, arrays and functions distributed in five levels. There was an image of a school in the centre (Fig. 5), where the student had to click to access the list of levels. Around the school, there were some blocked areas that would only be opened if the student could reach a high number of points. The idea was to verify if those very difficult challenges could bring any extra motivation to the students and if this stimulus could influence the student's gaming experience.

There was also a leader board, where students could compare their performance with their peers in three ways: by the amount of points gained, by the total time spent to solve the missions and by the number of attempts. The ranking included the position and values of all the students in the same class. However, a particular student could only see her/his position and the top 10. The three ranking criteria could result in different orderings: for instance, a student could be in the first place in a list, and in the tenth in another.

The game included a system of tips that presented a message to the student if she/he met a particular trigger condition. One of them happened if the student was inactive for some time while solving a mission. Another trigger was activated when there was an error in the execution, or the execution ended without achieving all mission objectives. For example, a possible tip suggested that the student checks if a variable used as a parameter was previously initialized, when the attendant should deliver an order to the customer, and actually picked a product off the shelf.

In the class, in the first lesson, teachers informed the students about the game and how it could be a useful tool especially for those who would feel difficulties. An area was developed for the teacher to monitor the evolution of students in the game. However, one of the researchers was responsible for monitoring the students through this area, and to provide out of class support and obtain feedback, through interviews or exchange of e-mails. This daily interaction allowed good advances in the game, be

Fig. 5. Control panel page.

it in its usability, fun and programming learning development. In addition, the students also answered the EGameFlow survey before starting some missions.

Conclusions of cycle 2. Two students completed the whole game (57 missions) and five students completed eighteen or more missions. Constant monitoring allowed us to identify and implement some tips and some student suggestions still during the cycle. Aesthetically the game was not considered good by many students. The immersion dimension (EGameFlow) again obtained the lowest scores. At the time students would gain more points if they were faster to solve the missions. Students expressed that they were more focused on the time control than in the mission itself. So, we tested another approach to win points: the fewer attempts to reach the solution, the more points the student would receive. This seemed to be more adequate and all the missions were converted to this new point gaining system. As very few students achieved a very high performance, it was not possible to make conclusions about the impact of the blocked areas that existed in the game interface.

4.4. Third cycle – second version with class integration

In the third cycle, 36 students enrolled in the first semester of the undergraduate Software Engineering Course of University of Santa

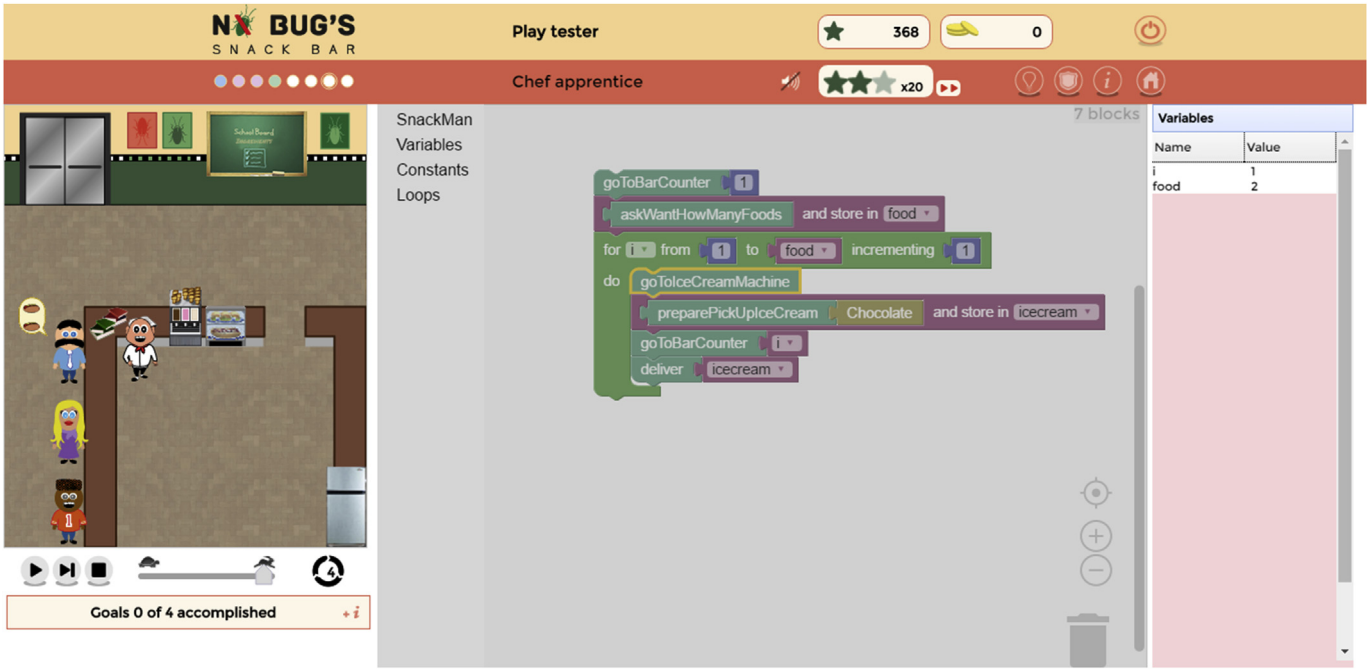

Fig. 6. Interface of the game NoBug's SnackBar.

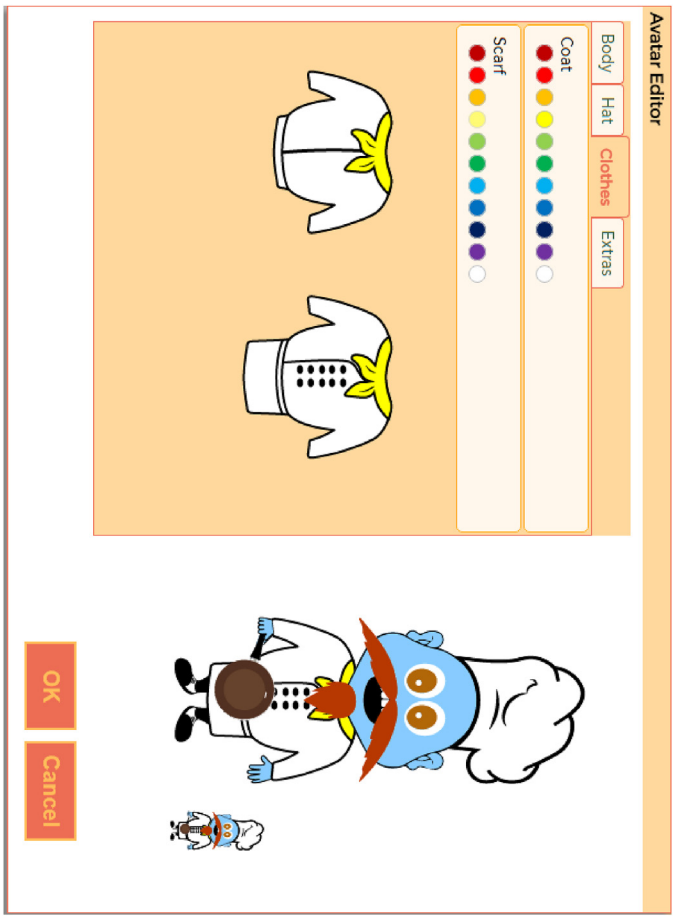

Fig. 7. Avatar editor.

Catarina State were involved, over two months. All changes that resulted from the previous cycle were introduced in the game before the students started to use it. The most visible change was a complete redesign of the game made by a professional designer. Fig. 6 shows the final main page of the game (compare with Fig. 4).

Point winning mechanics was changed, so that the number of points won depended on the number of attempts made to complete a particular mission (less attempts give more points). In the top of Fig. 6, it is possible to observe three stars (each worth 20 points) and two arrows. This means that each star is lost after two unsuccessful attempts. Fig. 6 exemplifies a situation where the student is in the third attempt, because one star is uncoloured (was lost), and the arrows are coloured. After losing the three stars, the student wins very few points in the mission, even if she/he achieves its goals. Each mission has its own limit of attempts per star and also the amount of points per star.

In this version an achievement system was added: students won points, coins or some bonification when they finished a level until a deadline. Teachers could define how students would win these extra rewards (for instance, finishing all the missions of a level until some specific date) and what would be won (game points or coins or extra points in the course). This last possibility allowed teachers to define that the student's performance in the game had some direct influence in their course grades.

Students could also configure the appearance of their attendant. At the beginning, she/he starts bald and with few configuration options: only eyes and skin colour. If the student is a woman, her attendant has eyelashes and accented lips. As the student wins more points, new options were available: Chef's hat and clothes, new special skin colours, haircut type and colours if student is a woman, or moustache type and colours if he is a man. Fig. 7 shows the tab where the clothes in the avatar editor are configured.

Table 2
Relation between the cognitive domain of the revised Bloom's Taxonomy categories and the missions' tasks.

Categories	Task's Types
1. Remembering 2. Understanding	Multiple choice: a solution is provided and a question about it is presented with four answer options. The student selects one of the options. The game runs the solution and verifies the student answer.
3. Applying	Fix errors: a solution is provided with some errors. The types of errors can be (1) incorrect use of comparison or logical operators, (2) erroneous references of variables, or (3) wrong sequence of blocks. The student must correct the mistakes by changing operators, variables or the order of the blocks.
4. Analysing	Sort: all the solution blocks are provided but dispersed in the workplace. The student must sort the blocks in the right order.
5. Evaluating	Fill in the blank: a partial solution is provided with some blocks missing. The student must complete the solution.
6. Creating	Create: the student creates her/his own solution from scratch. There are three different types: (1) the game starts by providing a few blocks to give some suggestion about the solution; (2) without suggestions or constraints; (3) with constraints: quantity of blocks and/or variables in the solution, how many times a type of block can be used in the solution, and a time-limit to solve the problem.

The instructional design was revised for the topics to be introduced more gradually between missions. The topic of functions was removed because it was considered outside the scope of the game, which was focused on the initial barriers' students face when learning programming. At this stage, the game had 73 missions divided in 12 levels: 1-environment, 2 to 4-variable manipulation, 5 to 7-conditional structures, 8 to 10-repetition structures, 11 and 12-arrays. Some levels (1, 2, 5, 8, 9 and 10) were covering the essential topics. The other levels were complementary and optional. For example, after finishing the level 2 (variables), the student could choose to proceed to level 5 (conditional) or to play levels 3 and 4 as complementary missions on variables.

In addition, within each level, the types of tasks asked in the missions were organized considering the Bloom's Taxonomy of educational objectives in the cognitive domain (Anderson et al., 2001). The idea was to get a better alignment between the mission's activities and desired learning outcomes. The more gradual increase in difficulty could also make the game more interesting to the students (Lameras et al., 2017). Table 2 shows the type of the tasks used in the missions and its classification in the taxonomy. This option allowed a better organization the missions' difficulty, providing first some simple tasks, with a lot of given code, and advancing to more complex tasks, including missions where it is necessary to develop programs from scratch. In creation-type missions (highest cognitive domain in Bloom), an assistant was available. When called, this assistant shows the mission solution in pseudocode. This feature could be used by students but costed them some points. The idea was to help students in difficulties to develop their solution after interpreting the provided pseudocode.

In the class, instructional material was incorporated into the explanation of the missions. Materials were produced to allow the teacher to introduce all topics using blocks. Students first learned each topic using blocks, then using a real programming language (Java). Through the achievement system, the teacher added bonuses in the assessments of students who completed the level by a defined deadline.

Conclusions of cycle 3. In the last version there was only one sequence of levels and missions. The less rigid structure of the levels in this version allowed students to avoid being "stuck" in the game when they were unable to complete a mission. Half of the students completed the 35th mission (almost half of the total missions) and only 2 students completed the entire game (73 missions). However, in general, it was possible to observe that despite the students had some freedom to choose their sequence of levels, most of them preferred to follow the suggested sequence of the missions. The pseudocode assistant was useful to many

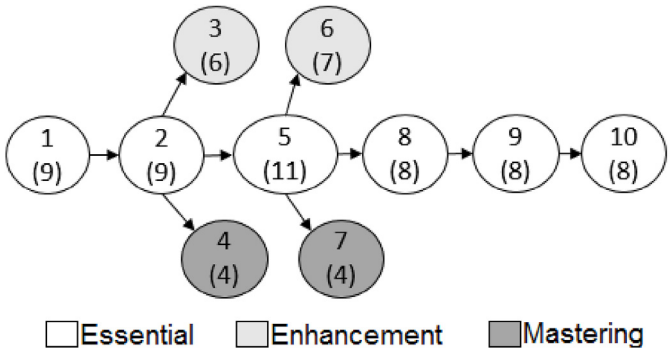

Fig. 8. Learning sequence.

students, although, in average, it took them 6.7 more attempts after using the assistant to reach the final solution, showing that students tried to understand and transform the pseudocode in blocks. Analysing the logs, it was possible to identify some missions that still required a high number of attempts to many students. There was a widespread complaint about reconciling time with class obligations and keeping pace in the game.

4.5. Fourth cycle –third version with class integration

The last development and evaluation cycle involved 33 students enrolled in the first semester of the undergraduate course on Software Engineering at University of Santa Catarina State. This cycle took 45 days. Several changes were introduced in the game: (1) the arrays topic was removed, as it was verified that its blocks representation was complex and inauthentic, making the topic more complicated to learn; (2) level 1 was kept as mandatory, but after solving it, students could choose any mission of any level; (3) two new types of tasks were created. In this version the total number of missions were distributed in 10 levels.

Although the students had liberty to choose their next mission after level 1, the suggested learning sequence is illustrated in Fig. 8. Each circle represents a level, the arrows represent the prerequisites between levels, and within parenthesis is indicated the amount of missions in each level. White levels include the essential missions. Students should learn the basic concepts in these levels. Light grey levels are enhancement levels and dark grey are mastering levels. Students can practice new and more complex situations in enhancement levels. Mastering levels have very challenging missions adequate to the better performing students. The game covers the initial topics usually included in introductory programming courses. They are organized in ten levels with a total of 74 missions: levels 2 to 4: Variable manipulation (19 missions); levels 5 to 7: Conditionals (22 missions); levels 8 to 10: Loops (24 missions: for loop, while loop and the two together). The initial level, level 1 (9 missions) is an introductory level to game playing.

In each essential level the initial missions introduce new concepts, such as variable or conditional using activities in the lower levels of Bloom's Taxonomy (remembering and understanding). As the player advances in the level, the tasks change to higher order categories. The last missions in each level ask for the creation of solutions from scratch. In enhancement levels, the game adopts one of the first four categories to reintroduce a concept or present a new machine in the snack bar. However, they also include several creation missions. Finally, all the missions in mastering levels ask for the creation of solutions, and many of them have constraints to the solution, as the quantity of variables or blocks to be used.

The teaching-learning process involves at least two actors: the student and the teacher. Therefore, it is important to consider both views when developing any technological artefact to improve this process. The teacher needs to monitor the individual evolution of each student and intervene when necessary. The monitoring area has been significantly improved for the teacher to have an overview of the class situation. Fig. 9 illustrates the page that was available to the teacher showing a map with all students' status in terms of missions completed. Each column represented a mission and each row a student. The numbers in the cells represented the number of attempts made by a student in a particular mission. The teacher could click on these numbers to view each attempt.

The green cells represented the missions completed with a number of attempts that falls within the interquartile range of the class. Red represents missions completed with the number of attempts beyond the upper limit of the interquartile range (maximum discrepancy). Missions represented in white were not completed, even though the student had already made some attempts. The uncompleted tasks that already had a higher number of attempts in relation to the rest of the class were displayed in yellow.

This page became useful, for example, for the teacher to help those students who had not yet completed a particular mission, but already had a discrepant number of attempts (cells in yellow). This map could be sorted by the students' names, the number of outliers (students with more discrepancy first), the number of missions done (students with the least completed missions first) and the time taken to complete the mission (students with more time first). According to the ordering chosen the values presented in the cells changed. For example, if sorting by time consumed was selected, cells would display that time in minutes. Ordering always favours showing first the students who appear to need more attention from the teacher.

In the class, in the first lesson the teacher explained how to access to the game and how it would be used in the course; for three weeks the students had to exclusively play, that is, there were no mandatory face-to-face classes; after that a test was applied using only the notation of blocks. In the remaining weeks the course followed a traditional approach, using a common programming language. In the end there was a final exam.

Conclusions of cycle 4. This section presented the final version of the game. In this cycle, half of the students concluded 48 missions and 10 students concluded the whole game (74 missions). Thirty students took the test, resulting in an average grade of 5.4 (±2,33), on a scale of 0–10, with 10 students obtaining a grade greater than or equal to 7.0 (grade necessary to be approved). To analyse the impact of the game experience in the students' final exam performance, we used the Pearson's bivariate correlation between the exam score and the number of missions completed. It was identified a moderate correlation ($r = 0{,}621$, $p < 0{,}0001$), indicating that the students that played more tended to perform better in the final exam.

5. Customize missions in the game

The definitions of the learning sequence and its missions are stored in a XML file accessed by the game. This allows the teachers to customize the learning environment. As an illustration, Table 3 shows the element <objectives> that is the part of a mission definition that allows the specification of its objectives. These include the tasks to be accomplished by the student and how the game can check if they are accomplished.

Fig. 10 illustrates an example of the element (*<objectives>*). It defines the mission type (*missionType*, in the case *fillInGaps* indicates the type "Fill in the blank") according to Table 2, the score (*xpIndividual*, *xpFinal* and *xpTotalRun*), the maximum number of commands to be used (*commQtd*) and extra punctuation (*maxCommands* and *maxCommandsRewardCoins*). There are 6 other attributes, not shown here, that can be used to enable buttons and define other restrictions.

Each <objective> element defines one of the missions' objectives. Its content defines the type of the objective. In the example, there are 3 objectives defined: (1) *deliver* that is accomplished when the customer in position 2 of the counter has been served (defined by the attributes pos and place); (2) *askWantHowManyFoods* that is fulfilled if this type of block was used in the solution for the customer in position 2 of the counter; and (3) *callTimes*, which is fulfilled if the "for" block was used only once in the solution. Some objectives are used to guide the student during the solution development. There is a total of 22 types of objectives

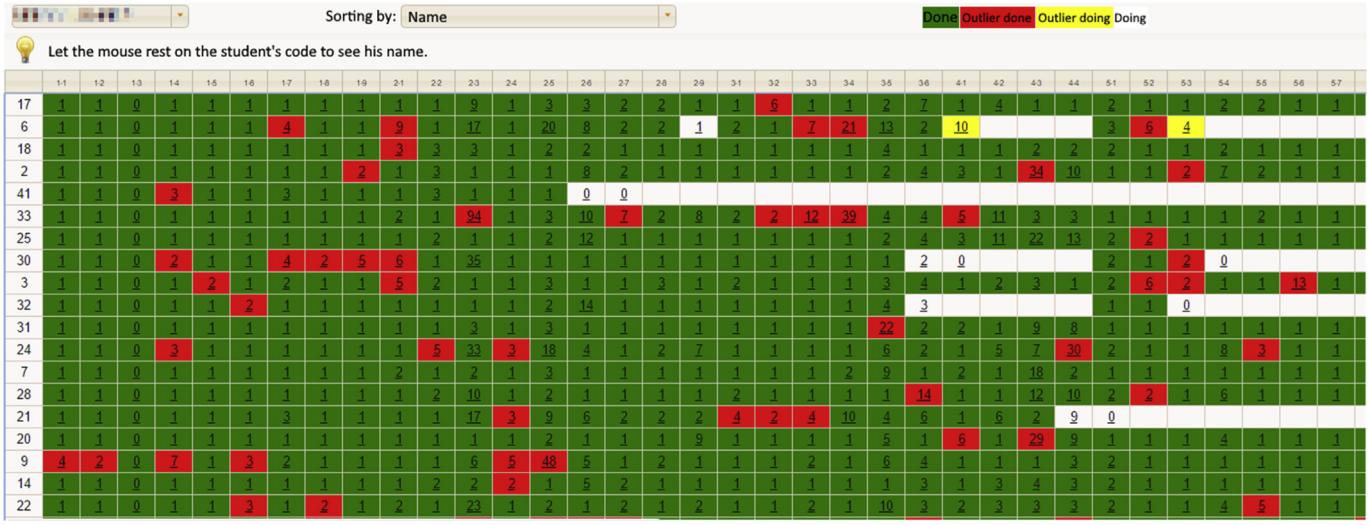

Fig. 9. Monitoring student tasks.

Table 3
Mission objectives definition in XML.

XML Element	Element description
<*explanation*>	Learning content and task description
<*hints*>	Rules that define when a tip is shown
<*help*>	Pseudocode wizard
<*commands*>	Commands available to build the solution
<*customers*>	Customers configuration and their desires
<*objectives*>	Objectives, game restrictions and scoring
<*xml*>	Initial blocks suggested in the solution

```
<objectives xpIndividual="50" xpFinal="10"
 xpTotalRun="6" commQtd="15"
 maxCommands="11" missionType="fillInGap"
 maxCommandsRewardCoins="2" >
    <objective pos="2" place="counter">
      deliver
    </objective>
    <objective pos="2" place="counter">
       askWantHowManyFoods
    </objective>
    <objective block="para" times="1"
          type="controls_for">
      callTimes
    </objective>
</objectives>
```

Fig. 10. Example of objectives setting.

defined in the game.

The process of verifying the student's solution is not based on the analysis of the code produced, but whether during the execution of the mission its objectives were fulfilled. Some objectives are checked during execution and others after it ends. Considering the example of Fig. 10, during execution the game checks whether the order has been delivered to the customer and whether the *askWantHowManyFoods* block has been used, and after execution if the "for" block has been used at most once. This approach allows the student to build a solution that will be evaluated considering its result and not by comparison with an ideal solution already defined in the game.

6. Guidelines to use serious game to support introductory computer programming

The four cycles of experimentation that aimed to develop the game and define how it could be used in classes took 2 years. It was a process based on students' feedback that lead to different changes in the version of the game for the next cycle, and in the strategy used for its integration in the introductory programming courses. During the cycles, fun and learning were key aspects evaluated. Below are listed a set of ideas that we argue are good practices or good considerations for the design of a game to support introductory programming learning. These principles are a refinement of a previous work published after the second cycle (Vahldick et al., 2017).

6.1. Guidelines about game features

P01-Reward points. Every game needs a form to reward the player. This usually happens through some kind of points system. During our studies there were behaviours that emerged from the students' desire to earn as much points as possible. This stimulated students to create strategies to overcome challenges, which is positive for learning.

P02-Reward contingency. Solving programming problems requires student concentration rather than quick thinking. Time pressure makes the student lose calm and adopt a trial and error approach, without necessarily having done a thorough analysis. When using the allocation of points for the number of attempts, the student can use all the time necessary. To avoid losing points, she/he must verify her/his program carefully before running it.

P03-Leaderboard. The leader board was often used by the students, including those with weaker performances. Although the game does not provide any direct competition between players, this ranking comparison was popular and seemed to encourage students to do more and better in the next missions. However, some weaker students asked that their classification would not be visible to the remaining students. Thus, we concluded that the ranking should only display the best classified students (ten in our case). If the student is not in the top, her/his classification appears only in her/his control panel.

P04-Avatar customization. This was a resource that students focused on particularly at the beginning of the game. The possibility of personalizing her/his avatar as she/he receives points offers a feeling of progress. The leader board also displays the students' avatar, offering yet another aspect of fun by allowing some to see the progress of others.

P05-Locked areas. There were blocked areas that could only be opened if the student achieved a certain score. Students showed motivation to do their best to earn the points and complete the missions in

order to unlock these areas. We consider that including this kind of strategy is recommended to keep students stimulated.

P06-Virtual currency. This feature was implemented in the third cycle, when students received points in the course assessments for completing game levels. In the fourth cycle, students received coins that could be used to purchase exclusive items for their avatar or activate the pseudocode wizard. In the third cycle, this system was much more used than in the fourth, showing that including some connection with the course assessment may be a good idea. This can also be a tool to integrate the game with the tasks of the discipline.

P07-Background music. Our experience recommends that the use of background music is made carefully, so that it doesn't add a disturbing factor. The main objective of game playing is learning through problem solving and it is important that the students are fully concentrated in the tasks. The use of different music when the player approaches a losing situation or the time is getting short seemed to add a stress factor to some students, which is should be avoided.

6.2. Guidelines about game-based computer programming leaning

P08-Feedback and support. Some students need to be frequently supported to continue their programming learning development. Therefore, the game should detect when a particular student needs help and the type of support that should be given. In this research support was provided manually, but further research is necessary, so that automatic feedback and pedagogical support can be included in games.

P09-Simple assistant using alternative representations. When students are stuck in a mission, a possibility is to allow them to call an assistant that shows the mission solution using a different representation (for example using pseudocode, flowchart, or any other representation that is easy to understand). Students are responsible for interpreting the representation and building the solution, that is, inserting and dragging the blocks, and changing their parameters, as required by the mission. Experience showed that this assistant was useful, as most students that used it could complete the mission, but often using some more attempts. Of course, using the assistant should have a high point cost to avoid students getting used to use it in every mission, even if they could solve it without assistance.

P10-Access and interact with concluded missions. Data shows that it was positive to allow students to interact with the missions they had already completed, either by consulting, running, debugging or changing the solution, even if it does no longer interfere with the score. Playing a mission again served both to review a concept and to test something that had made the student lose points.

P11-Blocks/commands available. The blocks available in the menus are only those necessary to solve the current mission. Although we may be restricting the students' creation process, we maintained this approach in the game to serve as instruction of what is best for the specific type of problem addressed in the mission. Then, when the student transfers this knowledge to the programming language, it will be the student's task to adapt the type of structure and use the correct commands according to the problem to be solved.

P12-Free choice of missions to be played. Allowing the student to freely choose the next mission to play contributes to prevent her/him from getting stuck in a mission. It could be assumed that this type of possibility would lead the student to ignore the learning sequence suggested by the game, and finally, to follow a learning path less adequate to her/his evolution. However, analysing the log, we were able to conclude that most students tried to advance according to the suggested sequence, and they only tried different missions when they were stuck in the game. In some cases, after playing a different mission they returned to the original mission and were able to solve it.

P13-Cognitive domain and tasks. It is a good idea to organize the tasks proposed to students according to a well-established taxonomy (Bloom's Taxonomy in our case). The first stages involve simpler tasks, such as determining the output produced by running a program, then making minor corrections to a solution with errors, and so on, until arriving to tasks that demand the development of a solution from scratch. This sequence aims to introduce students to functional programs and good practices before the game asks the students to create their own solution.

P14-Simple learning analytics to the teacher. Through simple analyses, such as the number of attempts in each mission, the time spent to solve them, and the number of accesses to support features, it is possible to give useful information to the teacher, allowing an easy identification of students who require special attention, and the kind of support they need. It is essential that serious games can provide this information to increase their usefulness as educational tools.

7. Uses of the game in classes

The game was used in three different semesters with different groups of students. The strategies used to connect the game playing with the formal classes were different from semester to semester. From this work four possible strategies emerged on how to integrate the game with an introductory programming course, especially in its initial weeks.

01-Play optionally. This option may be adequate when the teacher cannot change his/her pedagogical methodology for institutional reasons or believes that current pedagogical strategy is good for most students. In this case, the game can be used as an additional resource particularly directed to students with more difficulties. For example, it could be useful if those students play a set of missions before the corresponding topic is addressed in class. The teacher can monitor the failures and evolution of the students in the administrative environment and thus identify those who need further interventions.

02-Game used in classes. This mode is adequate for situations where the teacher can and is interested to introduce changes in her/his pedagogical strategies. The game and the blocks representation can be used as an alternative form to introduce topics, before the use of a programming language directly. The objective is to understand and use programming structures without the complexity of the language syntax. After, the teacher can present the same examples using the programming language.

03-Game used to prepare the course. This mode is suggested for teachers who want to use the game as a preparation for the course. This can be done before the course starts or in its first weeks. Students should play independently with some teacher support, so that they develop a good knowledge about the different topics the game addresses. After that period, the classes would follow introducing the programming language, taking advantage of the previous game experience.

04-In-game tasks or assignments that complement the course grade. Students often define their work priorities according to the next assessment. If some activity does not count to their grade it is often left behind. During our experiences we also saw this behaviour. Therefore, a small additional score counting to the course final grade could be positive to stimulate game playing. This component should be small in an independent playing scenario, due to possible fraud situations. An alternative, which was tested in a parallel experiment (Vahldick et al., 2018), is to use extra missions as a form of assessment. These missions should be available only during the period of the classes where the test is done. Knowing this methodology would be used, it is likely that the students get more involved with the game to be better prepared for the test.

8. Conclusions and future work

The main objective of the investigation described in this paper was design and implement a game that might help to develop problem-solving skills useful in introductory programming courses. Also, we wanted to study how such a game could be used in the course's context. A new serious game was developed, called NoBug's SnackBar, and experimented four times, in different semesters, using an iterative and interactive methodology, which allowed improvements to be made after each cycle. The experience did not limit the students either in time or space, as they were free to use the game at their own discretion, allowing

us to possibly evaluate results closer to reality in terms of flow and learning.

Now we try to answer each of the three research questions.

RQ1. What are the reasons for using serious games in teaching and learning problem-solving skills in introductory computer programming courses?

Programming learning materializes with practice, that is, it is not enough to read and understand the syntax of a programming language or how basic control structures, work. It is necessary to engage in an intensive problem-solving activity. The student must be motivated to continue this trajectory, no matter the difficulties found. Maintaining this motivation is the main appeal in serious games, as they are designed to use fun as a factor to keep the user playing.

The last three cycles of our work involved the use of different versions of the game. They involved 129 students (63.6% men and 36.4% women) with 70.5% (n = 91) claiming to have no prior knowledge of programming. Regarding the habits of playing digital games, practically the same population claimed to play daily (32,6%, n = 42), occasionally (34.1%, n = 44) and rarely (33.3%, n = 43). It was not surprisingly that the student's behaviour towards the game was also varied. Some engaged seriously with the game, finishing a large number of missions and looking for help when they weren't able to progress, while others didn't show the same level of interest. Some students said they liked the game, but they would rather spend their time with other learning activities.

The data collected showed that the game can be useful to many students, but also that other strategies are also necessary to cover the diversity of students common in higher education nowadays.

As expected, the connection between the game and the course assessment stimulated a lot of students to play the game. So, this may be a strategy to consider carefully.

RQ2. What features should be part of a serious game to promote problem-solving learning in an introductory computer programming course?

We used several tools to draw students' attention to the game, such as getting points to personalize the avatar, three leader boards and an achievement system. We observed that many students were attracted to the game, trying to win points mostly for personal satisfaction. The comparison with their peers didn't seem to be as important. Some students created strategies and tried to improve their performance looking to maximize the amount of points earned in each mission. There was a visible feeling of satisfaction when they were able to win a challenge. The objective of earning more points even induced some second cycle students to draft the mission solution first on paper, in order to minimize the number of attempts. We can conclude that even though scoring may not be a primary objective of a serious game, its use increases the attraction of the game.

The use of blocks instead of direct coding freed the students from the syntactic details of the programming language used in the course. This allowed some of them to make good progress in the game (and in the underlying topics), even with if they had difficulties to manage the language.

The organization of the tasks included in the game according to solid pedagogical principles is also an important recommendation. It is important that the tasks proposed to the student make sense and lead the student in a correct learning progress.

Finally, in sections 6.1 and 6.2 we listed 14 principles to be considered in the development of games for computer programming learning.

RQ3. How can a serious game be integrated into introductory computer programming courses?

In section 7 we included four possible strategies to use the game in the context of an introductory programming course. Our experience showed that a stronger connection with the course induces a higher use of the game. In particular, if the game results are taken into consideration in the course assessment, even using a low weight, the students tend to give more importance to the activity and develop efforts to get those points.

The possibility to use the game in different contexts requires that it is flexible enough for adaptations to be implemented with a minimal effort. This was possible because the game was implemented following the principles of game engines. The missions and the learning organization were not hard coded but included in a database that is consulted by the game when necessary.

Regarding the opportunities identified in section 2, this game answered the following items:

- 01-Teacher configurable environments: as described in section 6, the game allows the teachers to create new in-game tasks and configure the learning sequence;
- 02-Monitoring student tasks: Fig. 9 shows the area for the teacher to track the students' activities;
- 06-Provide instructional feedback during the game, not just at the end: there is a system that presents tips during the production of the solution and that constantly checks what the student is producing; a message is triggered, for instance, when the student is inactive for some time;
- 07-Opening the source code of the games developed: https://github.com/adilsonv77/nobugssnackbar;
- 08-Clarity in the learning objectives and adoption of educational philosophies: Fig. 8 shows the learning sequence that is also displayed to the student. Each level is associated with a learning objective. Table 2 shows the organization of the missions into each level following the Bloom Taxonomy;
- 09-Gender differentiation: the avatar has different skin for boys and girls. Game is absent of violence;
- 13-Aesthetics: after the second cycle, there was an investment to improve the aesthetics of the game. No complaints were received from students after this improvement;
- 14-Target audience: undergraduate students: all aspects of the game were targeted to undergraduate students following their first programming course. In particular, the sequence of topics and tasks were adapted for those courses.

There are still some future research paths that are planned to be pursued soon. The first is to include adaptive features in the game. The current version presents the same basic learning sequence to all students, without considering their performance. This may bore the best students or prove insufficient for weaker ones. The game could provide a different set of missions according to the students' progress: less missions and rapidly increasing in difficulty for those who progress quickly, and more missions, with a slightly increasing difficulty level, for those who have a slower progress. Second, currently, students produce the complete solution on the first attempt, as they earn points according to the number of attempts. However, the ideal would be for them to solve iteratively and incrementally, especially in the case of more complex missions. The game needs to support this good practice, and it can do so by incrementally presenting the objectives of the mission, where the number of attempts is based on this set of objectives, and no longer on the total attempts of the mission. It is also important to vary the number of attempts per star for each new set of goals that is presented. Finally, there is a need to develop a level and mission editor, requiring no knowledge of XML and SQL for the teacher to adapt the game to her/his preferences.

Acknowledgments

First author 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.

Declaration of competing interest

The authors declare that they have no known competing financial

interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.chbr.2020.100037.

References

Anderson, L. W., Krathwohl, D. R., & Bloom, B. S. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Allyn & Bacon.

Brown, E., & Cairns, P. (2004). A grounded investigation of game immersion. In *Human actors in computing systems* (pp. 1297–1300). https://doi.org/10.1145/985921.986048

Chau, B., Nash, R., Sung, K., & Pace, J. (2015). *Building casual games and APIs for teaching introductory programming concepts*.

Cocciolo, A. (2005). Reviewing design-based research. http://www.thinkingprojects.org/wp-content/dbr.doc. (Accessed 14 February 2014).

DiSalvo, B. (2014). Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science. *IEEE Computer Graphics and Applications, 34*(6), 12–15.

Fraser, N. (2015). Ten things we ' ve learned from blockly. In *"IEEE blocks and beyond workshop*.

Fu, F. L., Su, R. C., & Yu, S. C. (2009). EGameFlow: A scale to measure learners' enjoyment of e-learning games. *Computer Education, 52*(1), 101–112.

Gomes, A., & Mendes, A. J. (Sep. 2007). Learning to program-difficulties and solutions. In *International conference on engineering education* (pp. 1–5).

Harteveld, C., Smith, G., Carmichael, G., Gee, E., & Stewart-Gardiner, C. (2014). A design-focused analysis of games teaching computer science. *Proceedings Games+ Learning Society, 10*, 1–8.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2013). Understanding computational thinking before programming: Developed guidelines for the design of games to learn introductory programming through game-play. In P. Felicia (Ed.), *Developments in current game-based learning design and development*. Hershey, PA: IGI Global.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming. *ACM Computing Surveys, 37*(2), 83–137.

Lameras, P., Arnab, S., Dunwell, I., Stewart, C., Clarke, S., & Petridis, P. (2017). Essential features of serious games design in higher education: Linking learning attributes to game mechanics. *British Journal of Educational Technology, 48*(4), 972–994.

López, M. A., Duarte, E. V., Gutierrez, E. C., & Valderrama, A. P. (2016). Teaching abstraction , function and reuse in the first class of CS1 – a lightbot experience. In *21th annual conference on innovation and technology in computer science education* (pp. 256–257).

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2014). Educational games for teaching computer programming. In C. Karagiannidis, P. Politis, & I. Karasavvidis (Eds.), *Research on e-Learning and ICT in Education: Technological, pedagogical and instructional perspectives* (pp. 87–98). New York: Springer.

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2017). CMX : The effects of an educational MMORPG on learning and teaching computer programming. *IEEE Transactions on Learning Technologies, 10*(2), 219–235.

Marfisi-Schottman, I., George, S., & Tarpin-Bernard, F. (2010). Tools and methods for efficiently designing serious games. In *, no. October. 4th European conference on game-based learning* (pp. 226–234).

Miljanovic, M. A., & Bradbury, J. S. (2018). A review of serious games for programming. In *"Joint international conference on serious games*.

MIT Media Lab. "Scratch - imagine, program, share.". https://scratch.mit.edu/. (Accessed 16 March 2020).

O'Neil, H. F., Wainess, R., & Baker, E. L. (2005). Classification of learning outcomes: Evidence from the computer games literature. *Curriculum Journal, 16*(4), 455–474. https://doi.org/10.1080/09585170500384529

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172.

Settle, A., Vihavainen, A., & Sorva, J. (2014). Three views on motivation and programming. In *19th annual conference on innovation and technology in computer science education* (pp. 321–322). https://doi.org/10.1145/2591708.2591709

Shahid, M., Saleem, I., Wajid, A., Shujja, A. H., & ul Haq, K. (2019). A review of gamification for learning programming fundamental. In *2019 international conference on innovative computing (ICIC)* (pp. 1–8).

Sitzmann, T. (2011). "A meta-analytic examination of the instructional effectiveness of computer-based simulation games. *Personnel Psychology, 64*(2), 489–528.

Squire, K. D. (2005). Resuscitating research in educational technology: Using game-based learning research as a lens for looking at design-based research. *Educational Technology, 45*(1), 8–14.

Vahldick, A., Marcelino, M. J., & Mendes, A. J. (2017). Principles of a casual serious game to support introductory programming learning in higher education. In R. A. P. Queirós, & M. T. Pinto (Eds.), *Gamification-based e-learning platform for computer programming education*. IGI Global.

Vahldick, A., Mendes, A. J., & Marcelino, M. J. (Oct. 2014). A review of games designed to improve introductory computer programming competencies. In *44th annual frontiers in education conference* (pp. 781–787).

Vahldick, A., Schoeffel, P., Liz, F. B., Faria, V., Ramos, C., & Wazlawick, R. S. (2018). Maior Frequência na Aplicação de Instrumentos de Avaliação em uma Disciplina Introdutória de Programação : Impactos no Desempenho e Motivação. In *Anais dos workshops do VII congresso brasileiro de Informática na educação* (pp. 739–748).

Weintrop, D., & Wilensky, U. (Jun. 2015). "To block or not to block , that is the question : Students' perceptions of blocks-based programming. In *ACM SIGCHI interaction design and children* (pp. 199–208).

Whitton, N. (2010). *Learning with digital games: A practical guide to engaging students in higher education*. New York: Taylor & Francis.