A real-world approach to motivate students on the first class of a computer science course

ALESSIO BELLINO, Facultad de Ingeniería y Ciencias, Escuela de Informática y Telecomunicaciones, Universidad Diego Portales, Chile
VALERIA HERSKOVIC, MICHAEL HUND, and JORGE MUNOZ-GAMA, Department of Computer Science, Pontificia Universidad Católica de Chile, Chile

A common belief among students is that computing is a boring subject that lacks a connection to the real world. The first class (one 80-minute session) in an introductory computer science course may be an appropriate instance to combat such a belief. Previous studies have used course-wide interventions, e.g. games and physical/tangible devices to improve students' motivation. However, although other approaches help motivate students, they may lack real world context or have a high cost of deployment. This paper proposes a novel real-world based approach to introduce programming concepts in the first class of the introductory computer science course. This approach, called Protobject-based, is applicable to courses with over 100 students, has a low deployment entry barrier, requires low investment and may be used creatively to implement different experiences. Furthermore, the Protobject-based approach has an equivalent motivational effect - at least in the short-term - to a Game-based approach even if it is entirely focused on the real world. The low requirements of the approach make it especially suitable for an 80-minute first class in an introductory computer science course. The Protobject-based approach has been preliminarily validated and compared to a pure game-based approach with a study with 376 participants, and we present the analysis of motivation questionnaires, a pre-test and post-test, and a homework assignment given to the students. We posit that more research into initiatives such as this one - that can show students how computer science can impact the real world around them - is warranted.

CCS Concepts: • **Social and professional topics → Computational science and engineering education**.

Additional Key Words and Phrases: cs1

ACM Reference Format:
Alessio Bellino, Valeria Herskovic, Michael Hund, and Jorge Munoz-Gama. 2021. A real-world approach to motivate students on the first class of a computer science course. *ACM Trans. Comput. Educ.* 1, 1, Article 1 (January 2021), 24 pages. https://doi.org/10.1145/3445982

1 INTRODUCTION

Many students believe computing to be "boring, solitary, and lacking real-world context" [50]. To change this perception, students need to be motivated adequately from the first introductory course to programming. Several approaches have been proposed to improve students' motivation in introductory computer science (*CS0* and *CS1*) courses, e.g.: game design and development

Authors' addresses: Alessio Bellino, alessio.bellino@udp.cl, Facultad de Ingeniería y Ciencias, Escuela de Informática y Telecomunicaciones, Universidad Diego Portales, Santiago, Chile; Valeria Herskovic, vherskov@ing.puc.cl; Michael Hund, mahund@uc.cl; Jorge Munoz-Gama, jmun@uc.cl, Department of Computer Science, Pontificia Universidad Católica de Chile, Santiago, Chile.

1946-6226/2021/1-ART1 $15.00
https://doi.org/10.1145/3445982

(e.g. [14, 23]), programmable physical/tangible tools (e.g. [14, 28]), media computation [44]/media scripting [38], data science [12], music, robotics, security [49], art and creative coding [16], and web development [41]. Some approaches to teaching computational thinking or programming at the university level do not change the content of the course but rather its methodology, introducing e.g. pair programming, collaboration, gamification, or a change in grading [44]. Most interventions can improve students' passing rates by approximately one third [44]. Nevertheless, although these approaches may motivate students to learn programming, it can be argued that most of them are not effective to motivate students with the impact programming can make on the real world.

Programmable physical/tangible tools offer "a positive, exciting, and productive learning experience" [15], especially targeting real-world aspects. Nevertheless, an effective deployment of a first class based on such paradigm could be remarkably challenging: components need to be assembled and programmed and this requires a time frame usually longer that a single class, especially if instructors aim to demonstrate the impact of programming in the real world. Moreover, such tools may be not accessible/available in some educational contexts (e.g., budget problems).

To address the aforementioned drawbacks, we propose a novel approach for introducing programming concepts during a single introductory class (i.e., the first class of a course, in our case, 80 minutes long), designed to specifically maximize real-world experience with a low implementation cost. The approach is based on the building of interactive systems by leveraging common objects (e.g., when someone sits on their reading chair, the overhead light will turn on) using a common smartphone and intuitive software. Our approach is based on the use of (1) Protobject, a light, flexible and versatile prototyping tool able to sense interactions with objects by using just a smart phone and (2) IFTTT ("if this then that"), a free web-based service to create simple conditional rules.

The aim of this paper is to present this new approach, which we call Protobject-based, comparing it with a more traditional Game-based approach. Our proposal is used to introduce programming concepts in the first class of an introductory programming course. The study involved 376 students, including the analysis of pre-tests, post-tests, questionnaires regarding motivation, and a homework assignment. The experiment was performed in two sections simultaneously, with two professors with similar teaching evaluations, in two consecutive semesters, alternating the approach used for each professor. Our results shows that, at least in the short-term, the Protobject-based approach improves students' attitude regarding the relationship between programming and real life when compared to the Game-based approach, while preserving a similar general motivational power. Considering, moreover, that our real-world-based approach is flexible (allowing instructors to tailor examples according to available items), that the resulting demonstrations are short enough to be used in a 80-minute class, and that it requires a small effort to be implemented (gathering everyday objects, and preparing the demonstrations), it may be a suitable alternative to use for introducing the first programming concepts when compared to alternative programmable physical/tangible approaches. Naturally, further work is needed to ascertain whether these effects can be sustained over time.

This paper is organized as follows. In Section 2, we present the context in which the study was carried out. Section 3 presents the related work, and Section 4 presents our proposal. Section 5 presents the comparison between both approaches; in Section 6 we present the results of this comparison; in Section 7, we discuss our results; and finally, Section 8 discusses some limitations and Section 9 our conclusions.

2 *PROGRA*: AN INTRODUCTORY COMPUTER SCIENCE COURSE FOR ENGINEERING STUDENTS

Pontificia Universidad Católica (UC) is a well-ranked university [1] located in Chile. The university has over 26,000 undergraduate students, each of whom, when entering the university, applies to a specific program (e.g. medicine, engineering, law, primary education, etc). Within the Engineering study program, Computer Science is one of the possible majors (and professional titles) students can select. Only one of these programs (called *college*) is similar to international college programs in which students take 2 or 4 years of courses from several disciplines. In the case of this particular university, students then transition into one of the other existing programs.

The *Introduction to Programming* course, usually called *Progra* by students, is a first-year course for all Engineering students, as well as a mandatory course for students from the astronomy, statistics, and physics programs. Students from a few other programs, e.g. mathematics, and college, can choose to take *Progra* from a short list of electives. Students from almost any other university program can take the course as an elective, and there are therefore usually a small number of students from other university programs such as arts, economics, or design. This results in a course in which most students (over 90%) are not future computer science majors.

The *Progra* course has around 1,300 students per year, divided into 4-5 sections in the first semester and 7-9 in the second semester (plus one summer session). The course has two 80 minute teaching sessions and one 160 minute lab session per week. Each section has between 100 and 130 students. The large number of students and sections has made it necessary for all the lecturers to have one shared class schedule, in which the same contents are seen in all sections week by week. Each lecturer is free to teach the content as they feel is best, but there is little room for divergence in the topics that are taught.

The course is taught in the Python programming language and covers the following contents: an introduction to algorithms, variables and expressions, if, while, functions, strings, lists, files, object-oriented programming, search, sorting, and recursion. The first class of the course is used to present its rules regarding methodology, evaluation, ethics expectations, among others. The second class is used to provide an introduction into what an algorithm is, and as this is the only class scheduled for this purpose, lecturers typically try to provide a motivational experience that can, at the same time, teach students important concepts.

3 RELATED WORK

It is important to note that since our study focuses only on the first class, our aim is to help students understand the potential of computer programming, as well as to motivate them. We consider approaches for K-12 education (since our students recently came from high school, and our programming course is not intended only for students of computer science [20]), as well as approaches at the university level for CS0/Cs1.

To present the related work, we first introduce programming languages that have been used to teach introductory programming, and then we discuss teaching approaches used by instructors that have a real-world focus, which includes initiatives at high school and university level (CS0/CS1). Then, we introduce Protobject, IFTTT - which is used to trigger actions according to Protobject detection - and discuss our approach in relation to the previous works.

3.1 Programming languages

To introduce programming or computational thinking concepts, researchers have mostly used *textual*, *visual*, or *tangible* programming languages (or combinations thereof). Computational thinking can even be taught without a programming language. For example, at Virginia Tech, a

Computational Thinking for Computer Science (CT4CS) course - that did not use programming languages at all - was implemented to show students that computer science is not "just programming" [19]. Another such initiative for CS0 used stories instead of code to explain computational concepts [36]. This section briefly describes textual and tangible languages, and discusses visual languages in more detail, as these are most closely related to our approach.

Text-based, or textual, programming languages offer a more authentic experience regarding learning "real" programming, which can be motivating for some students, although their use may be more challenging for young students as well as for their teachers [15]. One such approach is the Logo programming language, which has a reduced set of instructions specifically created for teaching programming in schools [14]. At the university level, textual languages are generally used. One study compared Python versus C for an introductory programming course, finding that Python generally yielded better student outcomes than C [45].

Tangible languages are enjoyable and easier to use for younger children [15]. An experiment using a tangible programming language called Tern, that consists of connectable wooden blocks, found that tangible languages are better suited for whole-class activities and motivating children, while graphical interfaces were better for small-group activities [17]. Another such experience with a robot programming platform called PROTEAS with 28 commands and 16 parameters, in the form of connectable cubes, found the tangible version especially suited for younger children and girls [40].

Most studies for K-12 have used visual programming languages, which have been found to be engaging and support learning key concepts [15]. Visual approaches have also been used for CS1 courses, improving pass rates, perhaps because simplifying syntax can help students at risk of failing [44].

Block-based programming languages such as Scratch and Alice are two popular and widely used visual languages, which also have large online communities that provide learners with support from others [18]. Scratch 2.0 even allows users to create projects that use the computer webcam as a sensor [35]. One two-week intervention in a CS1 using Scratch was found to be useful for novices to learn basic programming concepts and eased the transition to a text-based programming language [30]. Another visual programming tool, App Inventor for Android, is especially engaging for teens as it allows them to create applications for their mobile devices [35]. Blockly Games is another popular block-based programming language, available via a browser. Block-based languages have been found to help later on learn a textual language, and their success may stem from how they address three barriers to learning a programming language, since (1) blocks rely on recognition instead of recall, (2) blocks allow organizing code into chunks, reducing the need for punctuation, and (3) blocks provide constrained direct manipulation, helping users avoid errors [7].

Event-driven programming, introduced through a graphical interface early in a first computer science course, was found to be effective and natural for university students in a CS1 course [10].

3.2 Teaching computational thinking with a real-world focus

Among the three main reasons for leaving a computer science major, students at Georgia Tech described not seeing a "relevant connection to the real world" [9]. The CT4CS course at Virginia Tech introduced concepts such as concurrency, with students stating that one of the reasons they found this topic interesting was that it dealt with "real life or practical situations" [19]. At Cal Poly, the CS0 course is especially focused on showing students how computers can help solve real-world problems [49]. This course allows students to choose an application domain of their interest - e.g. robotics, computational art, security, music [49]. No matter what the specific topic chosen by students, students have had improved grades and pass rates in subsequent computer science courses after taking this course [49].

In learning computational thinking, students should be building programs about things that "matter to them", which in many cases translates to courses based on games or digital stories [25]. Games in particular have been widely used, e.g. through platforms such as Alice, Greenfoot, Pex4Fun and Hour of Code [14]. Students may share the games they develop with their families and friends, which creates a sense of ownership and stimulates students to want to learn more about programming [37]. Some initiatives, e.g. RoboCode, have also introduced a competitive aspect into the game [31].

Robotics have also been used to bridge the gap between programming and the real world. One such initiative is the use of Lego Mindstorms, which are a powerful, non-intimidating tool to introduce programming concepts to novices, allowing students to observe robots interact with the real world [14]. Students (12 and 13 years old) who used Lego Mindstorms as a target platform (with a visual programming language) to learn programming had more positive emotions than those who used a desktop computer, without consequence to how much they learned [28].

Other initiatives have sought to teach Arduino, e.g. one 8-hour long student-led seminar taught students programming through four mini-projects: a light-emitting diode (LED) flasher, creating colors using red, green and blue LEDs, dimming a LED, and turning a LED on/off with a light-dependent resistor [26]. Such a seminar requires Arduino components (hopefully, a large number to allow students to creatively build their own projects), which requires a large budget when teaching many students. Similarly, other initiatives have used a Raspberry Pi as an introductory domain of work [32]. One such initiative at a CS0 course sought to allow students to learn by doing as well as understand the impact of computing and its consequences [21].

At a university level, several initiatives have sought to ground CS0/CS1 courses in a domain that is relatable to students, to improve their understanding of the material. For example, some CS1 courses have turned to media computation or game themes [44].

4 PROTOBJECT: USING A RAPID PROTOTYPING TOOL TO INTRODUCE PROGRAMMING CONCEPTS

Protobject [8] is a rapid prototyping tool that enables observation of the states of any stationary object (e.g. home appliances, doors, lights, windows, curtains, umbrella stands) as input for prototyping and testing smart systems. In many cases, Protobject can be used to sense users' interactions with objects. In fact, some interactions can be detected by recognizing changes of object states: for example, the action of closing a door (interaction) changes its state from "open" to "closed"; the action of taking an umbrella (interaction) from a stand changes its state from "umbrella present" to "umbrella absent". Protobject is designed to exploit state changes for triggering actions. For example, when somebody picks up the phone receiver answering a call (state change), Skype status changes automatically to "busy" (triggered event).

4.1 How Protobject works

Protobject works as a multipurpose camera-based sensor and is able to recognize visually observable properties of any stationary object using computer vision techniques. Protobject is composed of an app for Android - which works as a Wi-Fi camera - and an application for a personal computer - which receives and recognizes images.

Key features of Protobject are listed below.

- Protobject is able to sense objects without altering them or their surrounding environments; hence it is flexible and easily adaptable to changes.
- Protobject supports visual training sessions to let designers visually define the regions of interest in the captured images and take snapshots of those regions to teach the system what

states have to be detected and what events are associated with them. At run time, Protobject can match the camera live stream against the stored images of states to detect the current state and trigger the associated events.

- Protobject supports the detection of multi-value discrete states, either multi-value (e.g., closed-door/slightly-open-door/half-open-door/open-door), or binary (e.g., open/close, present/absent). Continuous state changes (e.g. the rotation degree of a knob) can be detected by adding physical markers on the observed objects (e.g., a multicolour band around the knob).

4.1.1 Smartphone App. The smartphone app turns an Android smartphone into a Wi-Fi camera that sends a video stream in M-JPEG format to the desktop app through the IP address displayed on the screen (Fig. 1).

Fig. 1. A smartphone turned into a Wi-Fi camera that displays the IP address to connect.

4.1.2 Desktop App. Protobject desktop app works in design and detection mode. The design mode lets users define and record the possible states of involved objects to build a database of states. The detection mode lets Protobject detect the current state of an object and its state changes.

The design mode lets users visually create regions by semi-transparent rectangles around the target objects (e.g., heater and window in Figure 2 - label 1 and 2); assign a name to each region in the associated panel (e.g., "Window" - label 3); and record the states to be detected by taking snapshots of objects in the desired states by the camera-button (e.g., label 4 in Figure 2). Pictures of the captured states are reported on the right (e.g., window closed and open - labels 5 and 6), and a name is given to each of them by typing a string in the field next to pictures (e.g. "Closed" and "Open" - label 7 and 8). When a new object is created and states are defined, they can be saved in a JSON file in ptj format (*.ptj), which records name, position and size of regions along with pictures of associated states. Saved objects can be loaded, modified and stored again to modify the prototype under development.

In detection mode, Protobject starts monitoring the selected scene, and for each region the real-time state is displayed by showing the name associated with it (e.g., label 1 and 2 in Figure 3), and a similarity index (the number displayed with green background) that measures the similarity between the current image and the recorded images. The displayed value is computed using a linear combination of Pearson correlation among RGB components [10] and cosine similarity among HoG descriptors of the images. States are refreshed with a frequency that can be visually selected by a sliding cursor (label 3 in Figure 3). Protobject can send event messages with the detected states to trigger actions in external components by WebSocket channels that can be enabled on the control panel (label 4 in Figure 3).

4.2 Motivation for its use

Protobject can be used to build working prototypes in a short amount of time and we believe it can be used in programming lessons to show how programming can be useful in everyday life by adding interactive behaviour to common objects. Protobject deliberately uses a 'low tech' approach by leveraging widely available resources, e.g. smartphones, low skill requirements and existing objects to be reinvented and reused.

4.3 IFTTT

If This Then That (IFTTT) is a free web-based service that allows users to link services through simple conditional statements called *applets*. Applets are triggered by a change in one service (e.g. receiving a notification) and can perform actions in another service (e.g. turn on a relay switch). Protobject is one such service, which can be connected with IFTTT using the maker webhook channel (https://ifttt.com/maker).

4.4 Discussion of the related work in relation to our approach

In regard to programming tools, we use Blockly, a state-of-the art visual programming language in which we integrated a few custom additional components to allow the definition of interactive behaviour leveraging Protobject and IFTTT (Fig. 4). We also added a Log window to show messages and let users understand what is going on.

Our approach mostly resembles the teaching context of tangible construction kits (like Arduino), but may be more flexible, adaptable and accessible, since it just requires a standard computer and a smartphone. Broadly speaking, we propose that Protobject can be used for introducing basic programming concepts focusing on real-world problems because its low setup requirements allow it to be used on-the-fly in a short time frame (an 80-minute class).

4.5 Theoretical framework

Our work finds its theoretical foundation in situated learning, an approach to teaching where students are encouraged to deal with real problems in everyday life and learn from real experiences [43]. Learning, in this way, is thus situated in everyday life and the students "construct" [13, 22] their knowledge on the basis of real experiences.

A situated learning activity is based on some assumptions [4, 46]: first, that learning is based on the action usually done in everyday activities; second, that knowledge is acquired in a situational way, and transferred to similar situations; third, that learning is the result of social situations concerning the way of thinking, perceiving reality and solving problems; and finally, that learning is not separated from the real world, but exists as a whole of actions and situations that occur in everyday life.

One of the approaches used to bring situated learning into educational environments is simulation [2], also based on case-based scenarios [2, 24]. In an analogous way, we also bring scenarios into the classroom through the simulation of real cases (see Figures 5 and 6) where programming finds its usefulness in everyday life.

Finally, it is worth noting that in technical contexts such as engineering, situated learning – also through simulation – has led to an improvement in students' motivation [11, 24].

5 METHOD

5.1 Context

Two sections, which we will call sections *Game-based* and *Protobject-based*, participated in an experiment to compare our approach with a more traditional game-based approach (which the

Fig. 2. A screenshot in design mode. Heater (1) and window (2) are regions of interest. The window is selected, and the associated states ("Closed" and "Open") are displayed on the right panel.

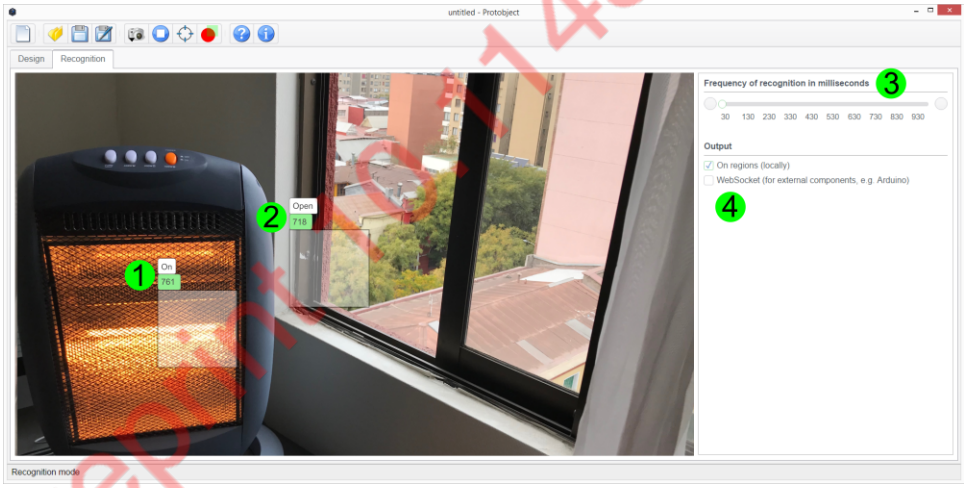

Fig. 3. Snapshot of heater and window displaying their current state ("On" and "Open" respectively) in recognition mode.

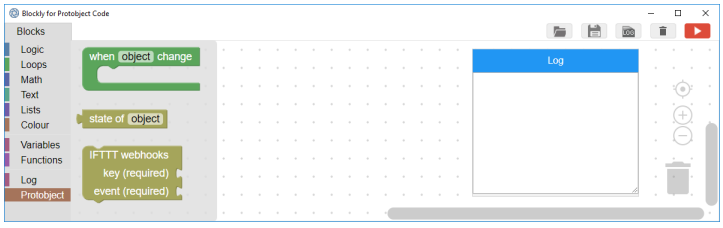

Fig. 4. Blockly interface with Protobject components and Log

professors had been using in previous semesters). The experiment was carried out in the first class of the *Progra* course for the second semester of 2018 (which we refer to as 2018-2) and for the first semester of 2019 (2019-1). The two sections had the same schedule (so the sessions were simultaneous), identical classrooms, and almost the same number of enrolled students (in 2018-2, section Game-based had 129 while section Protobject-based had 128, and in 2019-1 both had 105). The lecturers for the two sections were both full-time faculty with similar student evaluation ratings and who had both won teaching awards for this course. We scheduled the experiment for the second class of each semester, after the course rules had been explained. In 2018-2, one professor taught the Game-based session and the other the Protobject-based session, switching in the 2019-1 semester to avoid any bias due to the professor.

A two-group pretest-posttest design was used in our experiment: we compared data from Game-based and Protobject-based groups to evaluate the impact of Protobject in changing students' attitudes toward programming. During the session, the two groups carried out two different classes, as described in Table 1. It is relevant to note that we chose everyday objects (e.g. a boiler, a phone) for the demonstration, but similar demonstrations could be achieved by using other objects, according to availability.

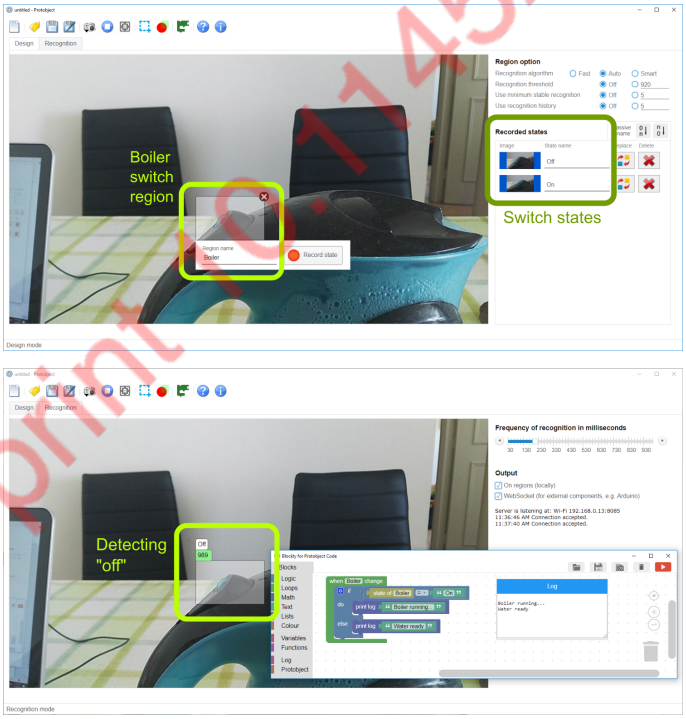

Fig. 5. The boiler prototype: The electrical boiler had a physical switch that moved when the water was boiling. We trained Protobject to detect the two possible states (positions) of the boiler switch (on and off), then the states could be leveraged to run the prototype (see Blockly code).

5.2 Data collection

Data was collected in each stage of the experiment.

Fig. 6. The phone prototype: we trained Protobject to detect the two possible states of the phone (available and busy), then the states could be leveraged to run the prototype (see Blockly code). Note that the lamp is connected through a Sonoff relay (which costs 5 dollars) compatible with IFTTT so that it could be triggered by the Blockly code.

At the beginning of the class, we asked students from both sections to fill out a questionnaire composed of three different sections:

(1) CASF: 16 questions aimed to measure computer attitude of computer science freshmen [33], translated by the researchers into Spanish.
(2) CS-Impact: 7 questions on a 5-point Likert scale about computing's impact in the world, aimed at understanding students' perceptions about computing (Table 2).
(3) Computational-Thinking: three questions selected from a questionnaire for measuring computational thinking abilities, adapted from the Spanish version [39].

At the end of the class, we asked students to fill another questionnaire composed of three sections:

(1) Today-Class: 6 questions on a 5-point Likert scale about the class they had just experienced, aimed at measuring whether students liked the class and whether they felt they had learned something (Table 2).
(2) CS-Impact: the same questions used in the initial questionnaire (Table 2).
(3) Computational-Thinking: similar to the one used in the beginning of the class but with different questions. This prevented students from remembering the previously given answers.

The questions, translated into English, are shown in Table 2. The CASF and Today-Class sections were used to compare the control and experimental groups in regard to attitude towards CS (results in section 6.1) and satisfaction about the class (results in section 6.2). The Computational-Thinking and CS-Impact questions were used to compare groups before (pre-test) and after (post-test) the class (results in section 6.3).

A week after the class, we received the homework from both groups, which in the case of the Protobject-based section were classified according to their feasibility and field of application (results in section 6.4). Before sending the homework, participants of both groups of 2019-1 were asked to answer four open questions that were analyzed qualitatively. The participants of the preceding semester (2018-2) did not answer the questions. Rather, we used the preliminary results of that semester to build the questions appropriately. The questions were the following (the Game-based group was asked the questions in terms of Blockly, while the Protobject-based group was asked in terms of Protobject):

(1) Considering <Blockly/Protobject>, do you believe computer science may be entertaining? Explain your answer.
(2) Considering <Blockly/Protobject>, what do you think about the relationship between programming and game creation? Do you think that programmers go beyond only creating games? Explain your thoughts.
(3) Considering <Blockly/Protobject>, do you think that a similar approach could be used to teach K-12 students? Explain why.
(4) Considering <Blockly/Protobject>, do you think that computer science could be useful for non-experts in computing? Explain your answer.

5.3 Data analysis

5.3.1 Quantitative data. Since most of the data we collected turned out to be not normal (according to Shapiro–Wilk test), we used non-parametric tests instead of parametric tests. We used the Mann–Whitney U test to discover significant differences between groups (unpaired samples). Next, we used the Wilcoxon signed-rank test to discover significant differences between pre-test and post-test (paired samples). Since Likert responses are ordinal data, only the usage of non-parametric tests – like the Wilcoxon signed-rank one – is appropriate. The Wilcoxon signed-rank test provides positive and negative ranks, which let us understand how many people changed their attitude toward higher or lower values. Note that higher positive ranks (in relation to negative ranks) do not necessarily mean that attitude toward a concept positively improved since it depends on the nature of the question (e.g., for inverted questions, the attitude improves when detecting a negative ranking).

For any pre-test/post-test comparison, we also computed the effect size by dividing the Z value of the Wilcoxon signed-rank by the square root of the number of observation using the Cohen criteria of 0.1 = small effect, 0.3 = medium effect, 0.5 = large effect [34].

5.3.2 Qualitative data. To analyse the responses to the open questions, first, we identified the frequency of words and groups of words used by participants for every question by using stemming (to reduce the words to their basic form, e.g., arriving → arriv) and cluster analysis (to identify

Table 1. Description of lesson plans and homework for both sections

	Game-based	Protobject-based
Attendance	• **2018-2**: 103 • **2019-1**: 75	• **2018-2**: 107 • **2019-1**: 91
Lesson Plan	*(1) Algorithms.* Brief introduction to what an algorithm is. *(2) Blockly Games Demonstrations.* Both professors did their usual first class to introduce programming concepts. One professor interactively discussed the plane seat calculator demo (https://blockly-demo.appspot.com/static/demos/plane/index.html), in which the concept of variables and mathematical operations is introduced. Then, they discussed and solved nine levels of the Blockly Games Maze (https://blockly-games.appspot.com/maze) in an interactive exercise with the class. This excercise introduced the concepts of instructions by having the person reach his goal location, instruction order, cycles by repeating instructions until the person reaches the end of the maze, and if/else by turning according to maze constraints. The other professor interactively drew figures using Blockly Turtle (https://blockly-games.appspot.com/turtle), introducing the concepts of instruction order, cycles by repeating instructions to draw a circle, functions and variables, and then solved some levels of the Blockly Maze game to introduce the if/else concept.	*(1) Algorithms.* Brief introduction to what an algorithm is. *(2) Blockly Games Turtle.* One professor used the Blockly Turtle game (https://blockly-games.appspot.com/turtle) to explain the concepts of: instruction by drawing a line, instruction order by drawing a square, cycles by drawing a circle, and functions by drawing rotated squares, while the other professor used the plane seat calculator demo and Blockly Maze to introduce concepts such as instructions, variables, if/else and cycles. *(3)Protobject.* Protobject was introduced as a way to connect Blockly programming to real objects. Two examples were built on-the-fly using Protobject and Blockly: (1) a smart boiler that notifies users when the water is ready (Figure 5), and (2) a system that recognizes whether a phone is in use or not. *(4) IFTTT.* We introduced if/else through IFTTT, explaining that it can be used to connect different services depending on their state. We built a final Protobject example using IFTTT and Blockly: a do not disturb system that turns on a lamp to signal the busy state when receiving/answering a call on a office phone and turns it off (signaling availability) when the call is finished (Figure 6).
Homework	The students were invited to solve the tenth level of the maze game and to explore the Blockly Turtle game and hand in a drawing made through the game.	The students were invited to think of a way in which they could use Protobject (e.g. in their home or university), and hand it in either as a drawing and a descriptive paragraph, or to implement it using Protobject and hand in a Youtube video link.

Table 2. CS-Impact and Today-Class questions

CS-Impact	1	Computer science can be fun.
	2	I'm afraid I won't do well in this course.
	3	Programming is difficult and complex, and only a few computer geeks can understand it.
	4	I believe computing is boring.
	5	Progammers create games and some other things.
	6	It would be possible for children to learn to program in school.
	7	Programming is only useful for a few computer experts.
Today-Class	1	I liked today's class.
	2	Today's class was not very interesting.
	3	Today's class was confusing.
	4	I felt like I learned a lot today.
	5	Today's class was motivating.
	6	I feel that I have a better idea about what programming is after today's class.

common clusters of words, e.g., real word). We considered a word or a cluster of words to be relevant when the frequency in the text was at least 2. We only considered clusters of two and three words.

Next, we read the context in which these extracted words or clusters of words were used to interpret the results. This method of analysing qualitative data is common in social science, and is based on the assumption that although researchers have better interpretative power, the computer has a better capability of identifying common patterns in the text (i.e., frequency of word and word clusters) [42].

6 RESULTS

6.1 Group comparison

To analyze the CASF questionnaire, we computed the composite scores on the five factors of the questionnaire: hardware usage anxiety, self-confidence, computer as beneficial tool, engagement with computer, long lasting negative consequence (Figure 7). No significant differences were found in any factors ($p>0.05$) using the Mann–Whitney U test. Therefore, we can state that the groups are comparable to carry out further analyses.

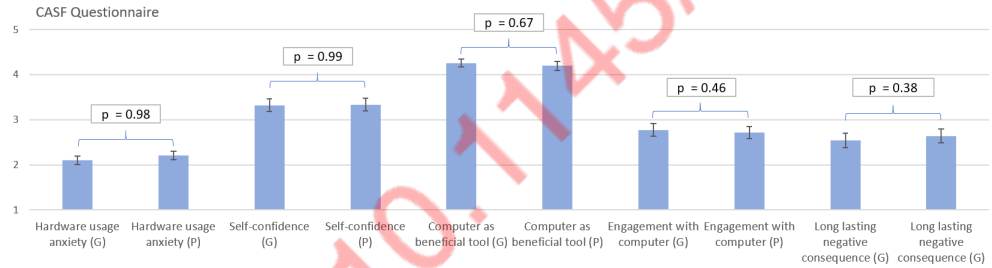

Fig. 7. Comparison of Game-based (G) and Protobject-based (P) groups regarding CASF questionnaire with its factors. None of the differences are significant ($p > 0.05$) according to the Mann–Whitney U test. Error bars denote 95% CI.

6.2 Lessons comparison (Today-Class questionnaire)

Regarding the lesson evaluations, the different items of the questionnaire measure the same dimension (Cronbach's Alpha: 0.77) and, therefore, we computed the composite score and carried out the Mann–Whitney U test among the scores. Such a test ($U=16688$, $p=0.42$) suggests that there are no significant differences among the two groups ($M=4.18$, $SD=0.60$ for Game-based; $M=4.13$, $SD=0.59$ for Protobject-based, see Figure 8).

6.3 Pre/post test comparison (CS-Impact) and Computational-Thinking

Regarding pre/post test questions, we analyzed them using a Wilcoxon Signed-Ranked to detect significant differences among pretests and posttests in the Game-based and Protobject-based groups.

In regard to the question "Computer science can be fun" (Figure 9), we detected significant differences in the Protobject-based group ($Z=-3.152$, $p=0.002$, positive ranks=45, negative ranks=19) with a small effect size ($r=0.22$), but not in the Game-based group ($Z=-0.226$, $p=0.82$, positive ranks=30, negative ranks=34).

In regard to the question "I'm afraid I won't do well in this course" (Figure 10) we detected significant differences in both groups ($Z=-4.663$, $p<0.001$, negative ranks=66, positive ranks=15 for the Protobject-based group, and $Z=-2.636$, $p=0.008$, negative ranks=47, positive ranks=21 for the

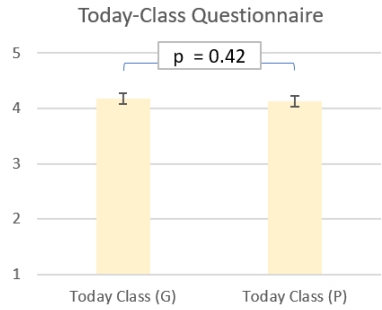

Fig. 8. Comparison of Game-based (G) and Protobject-based (P) groups regarding the composite score of the Today-Class questionnaire. The difference is not significant ($p > 0.05$) according to the Mann–Whitney U test. Error bars denote 95% CI.

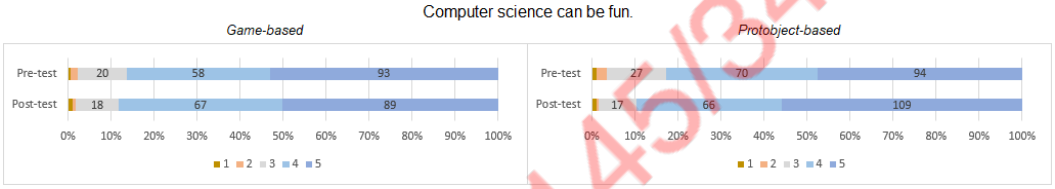

Fig. 9. Likert responses represented through stacked bars with the number of responses for each value.

Game-based group). Therefore, both groups reduced their fear with a small effect size (r=0.19) for the Game-based group, and a medium effect size (r=0.33) for the Protobject-based group.

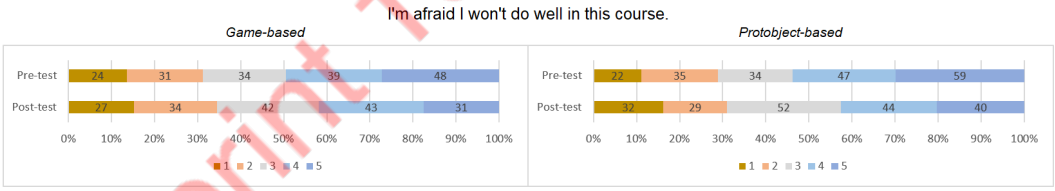

Fig. 10. Likert responses represented through stacked bars with the number of responses for each value.

In regard to the question "Programming is difficult and complex, and only a few computer geeks can understand it" (Figure 11), we detected significant differences in both groups (Z=-4.816, p<0.001, negative ranks=66, positive ranks=22 for the Protobject-based group, and Z=-6.166, p<0.001, negative ranks=72, positive ranks=13 for the Game-based group) Therefore, both groups increased their self-confidence toward programming with a large effect size (r=0.46) for the Game-based group, and medium (r=0.34) for the Protobject-based group).

In regard to the question "I believe computing is boring" (Figure 12), we detected significant differences in the Protobject-based group (Z=-4.441, p<0.001, positive ranks=19, negative ranks=59) with a medium effect size (r=0.31), but not in the Game-based group (Z=-1.569, p=0.11, positive ranks=23, negative ranks=35).

In regard to the question "Programmers create games and some other things" (Figure 13), we detected significant differences in the Protobject-based group (Z=-4.39, p<0.001, negative ranks=65, positive ranks=26) with a medium effect size (r=0.31), but not in the Game-based group (Z=-1.005, p=0.31, positive ranks=27, negative ranks=33).

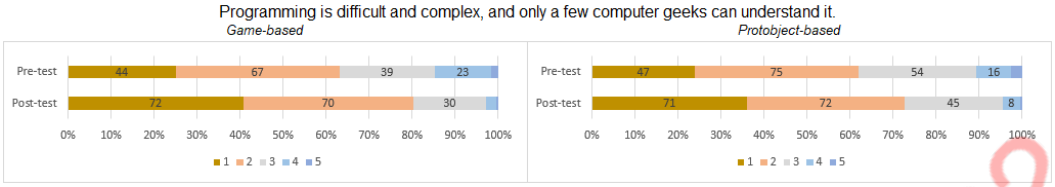

Fig. 11. Likert responses represented through stacked bars with the number of responses for each value.

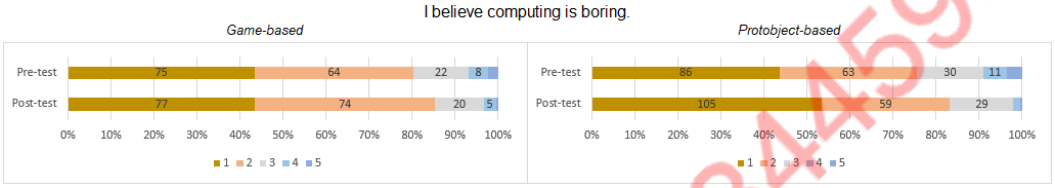

Fig. 12. Likert responses represented through stacked bars with the number of responses for each value.

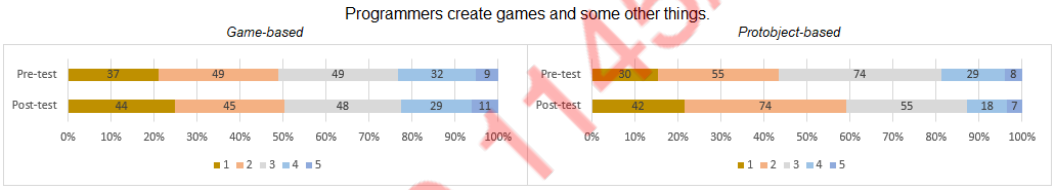

Fig. 13. Likert responses represented through stacked bars with the number of responses for each value.

In regard to the question "It would be possible for children to learn to program in school" (Figure 14), we detected significant differences in both groups (Z=-4.047, p<0.001, negative ranks=14, positive ranks=44 for the Protobject-based group, and Z=-3.066, p=0.002, negative ranks=19, positive ranks=45 for the Game-based group) Therefore, both groups changed their attitude positively on teaching to children with a small effect size (r=0.23) for the Game-based group, and medium (r=0.29) for the Protobject-based group).

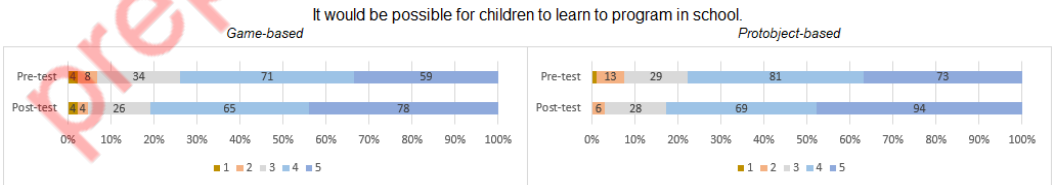

Fig. 14. Likert responses represented through stacked bars with the number of responses for each value.

In regard to the question "Programming is only useful for a few computer experts" (Figure 15), we detected significant differences in both groups (Z=-3.270, p=0.001, negative ranks=56, positive ranks=28 for the Protobject-based group, and Z=-2.300, p=0.021, negative ranks=50, positive ranks=25 for the Game-based group) Therefore, both groups changed their attitude positively on the usefulness of programming for general users with a small effect size (r=0.17 for the Game-based group and r=0.23 for the Protobject-based group).

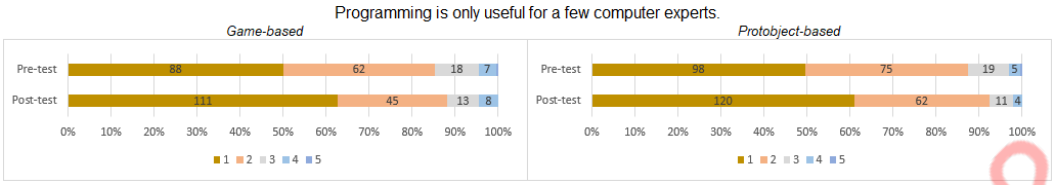

Fig. 15. Likert responses represented through stacked bars with the number of responses for each value.

Table 3. Pre-test/Post-test comparison. The group that is in italics outperformed the other in terms of effect size (r) by at least 0.10.

Question	Group	Pre-test/Post-test comparison		Effect size (r)	p-value
		- ranks	+ ranks		
Computer science can be fun	Game	34	30	-	0.82
	Protobject	19	45	0.22	0.002
I'm afraid I won't do well in this course	Game	47	21	0.19	0.008
	Protobject	66	15	0.33	0.001
Programming is difficult and complex, and only a few computer geeks can understand it	*Game*	72	13	0.46	0.001
	Protobject	66	22	0.34	0.001
I believe computing is boring	Game	35	23	-	0.11
	Protobject	59	19	0.31	0.001
Programmers create games and some other things	Game	33	27	-	0.31
	Protobject	65	26	0.31	0.001
It would be possible for children to learn to program in school	Game	19	45	0.23	0.002
	Protobject	14	44	0.29	0.001
Programming is only useful for a few computer experts	Game	50	25	0.17	0.021
	Protobject	56	28	0.23	0.001

In regard to questions aimed to check computational thinking abilities (Figure 16), both groups significantly improved their results (Z=-3.897, p<0.001, negative ranks=24, positive ranks=57 for the Protobject-based group, and Z=-3.996, p<0.001, negative ranks=31, positive ranks=67 for the Game-based group) with a medium effect size (r=0.30 for the Game-based group, and r=0.27 for the Protobject-based group).

Table 3 summarizes all the pre-test/post-test results showed in this section.

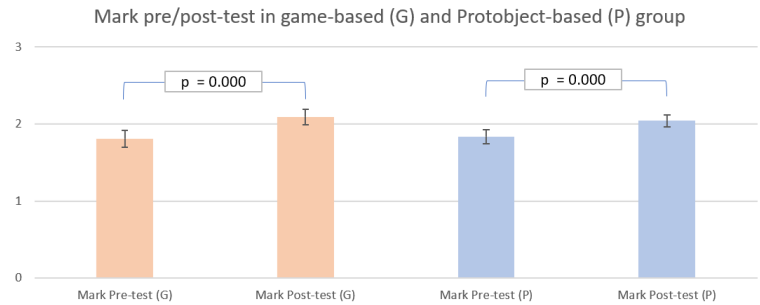

Fig. 16. Marks pre/post test in both groups. All differences are significant. Error bar denotes 95 % CI.

6.4 Homework submissions

Regarding the homework based on Protobject, 172 students submitted their work. Nine of them submitted a video showing a prototype that really worked. The rest (163) submitted a drawing where they indicated the area in which Protobject could be trained by indicating the possible states to be recognized. Then, they "drew" the algorithm using Blockly blocks. 37 submissions were not viable, namely, they could not be actually programmed using Protobject (e.g., send a notification when the user is hungry). The rest (135) of the submissions were viable. Considering the viable submissions only, students generated 55 different unique ideas. All of them used notifications (e.g., send a notification when a home appliance finishes its work).

In regard to the homework based on Blockly Turtle for the Game-based section, 201 students submitted their work. Out of these, 48 were abstract drawings (lines, circles), 21 were logos (e.g. football clubs, technology companies, the Olympic rings), 13 were faces, 13 were shapes such as stars, flowers and rainbows, 12 were animals, and the rest were cartoon-like characters, stick figures, objects (guitars, a cup of coffee), landscapes (e.g. a starry sky), among others.

6.5 Qualitative analysis

In this section, we present the results for each of the open questions that were asked of the students when they turned in their homework.

6.5.1 ... do you believe computer science may be fun? Students in the Protobject-based group felt that computer science is entertaining because it focuses on everyday life, allows to solve several problems of daily life, offering tangible, automated solutions, and stimulating creativity in an easy and intuitive way. Students in the Game-based group felt that computer science is entertaining because it is fun, a game, and allows drawing, stimulating the imagination in an interactive, simple and fun way.

6.5.2 ... what do you think about the relationship between programming and game creation? Students in the Protobject-based group thought that programmers go far beyond games alone, creating solutions to facilitate, automate, optimize, and solve problems in everyday life. Students in the Game-based group also thought that programming goes far beyond games. Although they mentioned games, other concepts such as algorithms, life, science, knowledge, business, physics, engineering, mathematics, systems, social creativity and intelligence also emerged quite frequently.

6.5.3 ... do you think that a similar approach could be used to teach K-12 students? Students in the Protobject-based group felt that Protobject could be used in school because it focuses on everyday problems, proposing interactive and tangible scenarios, and is simple, intuitive and didactic. Students in the Game-based group felt it could be used at school because it stimulates thinking in a simple and fun way, and it is a game. They mentioned it being very useful, entertaining and attractive, stimulating the imagination in the creation of the drawings. Students in this group expressed abstract concepts such as mathematics, logic and thought.

6.5.4 ... do you think that computer science could be useful for non-experts in computing? Students in the Protobject-based group thought computing may be useful for non-experts because it allows solving everyday problems (automation, situations) in a simple and intuitive way. For students in the Game-based group, computing is useful for non-experts because it allows achieving objectives, helping people.

7 DISCUSSION

7.1 Experiment Results

Broadly speaking, it is not possible to state that one approach (Protobject- or Game-based) is better than the other. According to the results in Figure 8, students did not express a preference, and differences among the groups are not significant. However, according to pre-test and post-test comparisons, we can observe how students changed their attitude toward some ideas.

In regard to the results on the "Computer science can be fun" statement, we must point out that, despite almost 90% of the students having a very positive attitude before the treatments (i.e., pre-tests with high scores, see Figure 9) (which coincides with a largely positive general perception among students about the *Progra* course, and a growing interest in the computer science major among Engineering students), ranks (provided by the Wilcoxon Signed-Ranked test) suggest that attitudes changed positively and significantly in the Protobject-based group (45 positive changes vs 19 negative changes). Instead, attitude in the students of the Game-based groups changed slightly negatively (30 positive changes vs 34 negative changes) although the difference is not significant. Therefore, introducing programming using Protobject/IFTTT could be more entertaining with a small effect size (r=0.22).

The responses to the statement "I'm afraid I won't do well in this course" suggest that both approaches are effective to reduce students' fear even if with different effect size: medium in the Protobject-based group (r=0.33) and small in the Game-based group (r=0.19). Also considering the responses to the statement "Programming is difficult and complex, and only a few computer geeks can understand it" suggests that both approaches are effective to increase self-confidence toward programming even if with different effect size: large (r=0.46) for Game-based group and medium (r=0.34) for Protobject-based group.

The responses to the statement "I believe computing is boring" show that, despite almost 90% of the students had a very negative attitude before the treatments (i.e., the students *do not* agree with the statement, see Figure 12), the attitudes changed negatively (so it should be interpreted as a positive change according the real meaning of the question) and significantly in the Protobject-based group (19 positive changes vs 59 negative changes). Also the attitude of the students of the Game-based groups changed slightly negatively (23 positive changes vs 35 negative changes) but the difference is not significant. Therefore, introducing programming using Protobject/IFTTT can reduce the boredom of students with a medium effect size (r=0.31). This result basically coincides with the result of the first question - where the Protobject-based approach was judged as more entertaining.

Regarding the statement "Programmers create games and some other things", ranks suggest that attitude changed negatively - i.e., the students *do not* agree with the statement, so it should be interpreted as a positive change - in the Protobject-based group (65 negative changes vs 26 positive changes) but not in the Game-based group, where the distribution remained almost unchanged (33 negative changes vs 27 positive changes). The effect size is medium (r=0.31), and this result was expected since the lesson carried out in the control group was based mostly on a game (Blockly Maze).

The ranks of the question "It would be possible for children to learn to program in school" suggest that the attitude changed positively for both groups with similar effect size (r=0.23 for the Game-based group vs r=0.29 for the Protobject-based group).

Regarding the responses to the question "Programming is only useful for a few computer experts", ranks suggest that attitude changed negatively - i.e., the students did not agree with the statement, so it should be interpreted as a positive change - both in the Game-based group (50 negative

changes vs 25 positive changes) and in the Protobject-based group (56 negative changes vs 28 positive changes) with a small effect size.

Moving to computational thinking abilities, both the approaches let students improve in similar way. Finally, in most of the cases, students were able to develop their ideas (see homework) contextualized in the real world generating many unique ideas. Moreover, it is worth to note that even when students' ideas were not viable using Protobject (e.g., when the user closes their eyes, turn off the light), they were however well-situated in the real world attempting to solve a real problem.

There was a difference in the number of submitted homeworks among both groups - however, this may be because one group was reminded more effectively to turn in the work (e.g. in 2018-2, in the Game-based group, more students turned in homework in this section than those who attended the class).

The answers to our open-ended questions allowed us to better understand how the Game-based and Protobject-based approaches impacted the students' perceptions about computing. Although the responses from both groups were positive, there was a marked difference for students in the Protobject-based group, who related programming to real-world problems that they could solve. They felt Protobject was entertaining because it stimulates thinking in real life, while Blockly Turtle was entertaining because it is a fun game. However, even students in the Game-based group did not believe that programming is only useful for games, showing a good grasp of the potential of programming. When students were asked whether a similar approach could be used for K-12 students, there was a clear opposition between concrete concepts (e.g., real life, situations, tangible) for Protobject and abstract concepts (e.g., math, physics, logic, imagination) for Blockly Turtle. Finally, when asked whether computer science could be useful for non-experts, the semantic field for answers in the Game-based group was quite limited, possibly suggesting that the students did not know what to answer.

7.2 Our approach as compared to the literature

To the best of our knowledge, studies focused only on the first class - as the one we are proposing - are not common in the area of education and computer science. One study in an Indian university carried out a two-week intervention using Scratch, focused on improving student performance and easing the transition to a text-based programming language, rather than on student motivation [30].

Studies about how to best approach the first class of a university course are mixed: some research suggests to focus on the syllabus and distributing class materials, while others suggest other activities, among them those that aim to motivate students [3]. Studies in other areas (e.g. psychology) have found that a first class may have a positive impact in terms of motivation and interest of students [27, 47], and this is potentially true in other areas such as computer science. Therefore, we believe that studying the first class is of considerable interest especially when trying to evaluate and understand the motivational aspect of students who approach computer science for the first time. In addition, our results show that student attitude towards programming has clearly moved towards the real world and the resolution of concrete problems. As previously discussed, some course-wide initiatives have been proposed in which CS0/CS1 courses have a real-world focus (e.g. [49]), or in which students specifically comment that they appreciated understanding computing in the context of real problems (e.g. [19]). Therefore, even if the effect in the mid or long term may be questionable, this change of attitude - evident immediately after the first class - may contribute to encourage the right development of computational thinking in the frame of the real world. On the other hand, educational research has demonstrated that focusing on problem-solving in the real world may foster intellectual curiosity and motivation, among others [6, 48].

Several studies have found students who believe computer science to be boring [5, 50]. Although our intervention lasted only for one class, we found that the Protobject-based approach was useful to change students' attitude. As matter of fact, the Protobject-based group changed perceptions positively regarding computer science being fun and not boring.

One of the reasons students have mentioned for leaving the field of computer science is its lack of connection to the real world. One student felt that in computer science, they were being trained to do tasks, while their new major was "relevant to situations that occur in daily life" [9]. One of the difficulties of first-year computer science students is understanding programming terms that "do not have equivalents in real life" [29]. The Protobject-based approach could potentially provide a connection to real life that students feel is missing from introductory computer science. As briefly noted above, educators have found that a problem-solving curriculum focused in the real world can lead to greater intellectual curiosity, motivation, a better attitude toward education, and better performance at the university [48]. Thus, a curriculum that is designed with real-world problems in mind may make computational thinking relevant to the lives of students and, therefore, is more likely to keep them interested in the subject [6].

Another aspect that deserves to be discussed is the adaptability of the different approaches considering that, on the one hand, we tried to introduce important concepts (e.g. algorithm, if-then-else) and, on the other hand, we tried to do so with the intention of motivating students in a limited period of time (an 80-minute class). To discuss the adaptability of different approaches to our context we only consider their objective requirements. Our game-based approach (turtle/maze) is definitely the most adaptable - as teachers only need a computer and an internet connection. The Protobject approach is similarly adaptable even if it has some more requirements, namely a smartphone and some recycled objects to bring to the scene. Other approaches based on robots and Arduino (e.g., [14, 26, 32]) are more demanding in terms of requirements, i.e. they need components, sensors, cables, and a prototype construction phase, and their use in a limited period of time like that of one class is difficult. Moreover, robot-based and Arduino approaches are more expensive - so their applicability in low-budget contexts (such as UC) is limited. Although both the Protobject approach and the robot-and-Arduino approach have a similar focus based on the real world, it can be said that - based on the objective requirements of each of them - the Protobject approach is more suited to the duration of a single class while maintaining the focus on the real world.

8 LIMITATIONS

There are some limitations to this research that we would like to acknowledge.

8.1 Intervention duration

Our intervention purposefully lasted for only one 80-minute introductory session. This short time frame is not enough to make claims about any medium or long-term effects of either experience, therefore possibly not being enough to truly change student attitudes towards computing. Therefore, carrying out a just a one-lecture intervention, longitudinal information about student interest/motivation are outside the scope of this study

8.2 Unconscious biases

The intervention was carried out by two professors who were a part of this research. The professors taught the Game-based class that they usually teach for this course, and modified the class to incorporate Protobject when it was their turn to teach the Protobject-based class. Although the professors did not try to influence the results in any way, it is possible that unconscious biases, phrases, or attitudes may have led them to affect study results in some way - e.g., they may have been more nervous in the Protobject-based class about some aspect of the demonstration failing,

which may have affected students' perceptions of each class. We would like to acknowledge that these biases, while unintentional may have affected study results.

8.3 Generalization

This experience was carried out at UC, where around 70% of Engineering students are male, and practically all of the students are from the same country, so the results may not be generalizable to other cultures or other populations. Furthermore, since some students are required to take this course while others choose to take it as an elective, the participants of the session are not randomly chosen and could not be representative of a general university population.

8.4 Questionnaire building

To evaluate psychometric dimensions using Likert responses, it is usually preferable to build several questions to measure the same dimension and sum them up to generate a unique metric to evaluate that dimension. In our case - to shorten the questionnaire - we used a single question to measure a dimension. Although it could lead to lower reliability of the measurement, using a single question to measure a specific dimension is not uncommon in human-computer studies.

9 CONCLUSIONS

This work proposes a real-world approach to introducing computer science concepts. Such an approach may be advantageous over tangible-based approaches such as Arduino because of its low investment and possibility of deploying in a course with a large number of students. When comparing game-based and real-world based approaches, the real-world approach does not hinder students' enjoyment or understanding, proving to be a viable alternative.

Broadly speaking, a Game-based approach is not better at motivating students than a real-world-based approach in the short-term. In fact, although both approaches are overall equivalent in terms of students' satisfaction (see Figure 8), the Protobject-based approach was better at changing students' attitudes regarding the relationship between programming and the real world. Moreover, the Protobject-based approach did improve students' attitudes significantly regarding computer science being fun and not boring, whereas the Game-based approach did not. Considering our results, we believe that more research into initiatives such as the Protobject-based approach are warranted. Further work is also needed to understand whether effects from a motivating experience during a first class carry over into the longer term.

ACKNOWLEDGMENTS

This work was supported in part by: CONICYT/FONDECYT (Chile) grant 1181162, Engineering Postdoc UC 2018, and by the Departamento de Ciencia de la Computación UC/Fond-DCC 2017-0001.

REFERENCES

[1] [n.d.]. QS World University Rankings by Region 2019. https://www.topuniversities.com/regional-rankings. Accessed: 2019-01-03.

[2] Stephen M Alessi and Stanley R Trollip. 2000. *Multimedia for learning: Methods and development.* Allyn & Bacon, Inc.

[3] Denise M. Anderson, Francis A. Mcguire, and Lynne Cory. 2011. The first day: it happens only once. *Teaching in Higher Education* 16, 3 (2011), 293–303. https://doi.org/10.1080/13562517.2010.546526 arXiv:https://doi.org/10.1080/13562517.2010.546526

[4] John R Anderson, Lynne M Reder, and Herbert A Simon. 1996. Situated learning and education. *Educational researcher* 25, 4 (1996), 5–11.

[5] Neil Anderson, Colin Lankshear, Carolyn Timms, and Lyn Courtney. 2008. âĂŽBecause its boring, irrelevant and I dont like computers: Why high school girls avoid professionally-oriented ICT subjects. *Computers & Education* 50, 4 (2008), 1304 – 1318. https://doi.org/10.1016/j.compedu.2006.12.003

[6] Charoula Angeli, Joke Voogt, Andrew Fluck, Mary Webb, Margaret Cox, Joyce Malyn-Smith, and Jason Zagami. 2016. A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Journal of Educational Technology & Society* 19, 3 (2016).

[7] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. 2017. Learnable Programming: Blocks and Beyond. *Commun. ACM* 60, 6 (May 2017), 72–80. https://doi.org/10.1145/3015455

[8] Alessio Bellino. 2016. Protobject: a sensing tool for the rapid prototyping of UbiComp systems. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*. ACM, 257–260.

[9] Maureen Biggers, Anne Brauer, and Tuba Yilmaz. 2008. Student Perceptions of Computer Science: A Retention Study Comparing Graduating Seniors with Cs Leavers. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08)*. ACM, New York, NY, USA, 402–406. https://doi.org/10.1145/1352135.1352274

[10] Kim B. Bruce, Andrea P. Danyluk, and Thomas P. Murtagh. 2001. Event-driven Programming is Simple Enough for CS1. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '01)*. ACM, New York, NY, USA, 1–4. https://doi.org/10.1145/377435.377440

[11] Alexandre Cabral, Rolland Viau, and Denis Bédard. 1997. Situated learning and motivation strategies to improve cognitive learning in CE. *age* 2 (1997), 1.

[12] Sarah Dahlby Albright, Titus H. Klinge, and Samuel A. Rebelsky. 2018. A Functional Approach to Data Science in CS1. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE âĂŹ18)*. Association for Computing Machinery, New York, NY, USA, 1035âĂŞ1040. https://doi.org/10.1145/3159450.3159550

[13] Thomas M Duffy and Donald J Cunningham. 1996. Constructivism: Implications for the design and delivery of instruction. (1996).

[14] Francisco Buitrago Florez, Rubby Casallas, Marcela Hernandez, Alejandro Reyes, Silvia Restrepo, and Giovanna Danies. 2017. Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Review of Educational Research* 87, 4 (2017), 834–860. https://doi.org/10.3102/0034654317710096 arXiv:https://doi.org/10.3102/0034654317710096

[15] V. Garneli, M. N. Giannakos, and K. Chorianopoulos. 2015. Computing education in K-12 schools: A review of the literature. In *2015 IEEE Global Engineering Education Conference (EDUCON)*. 543–551. https://doi.org/10.1109/EDUCON.2015.7096023

[16] Ira Greenberg, Deepak Kumar, and Dianna Xu. 2012. Creative Coding and Visual Portfolios for CS1. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 247âĂŞ252. https://doi.org/10.1145/2157136.2157214

[17] Michael S. Horn, R. Jordan Crouser, and Marina U. Bers. 2012. Tangible interaction and learning: the case for a hybrid approach. *Personal and Ubiquitous Computing* 16, 4 (01 Apr 2012), 379–389. https://doi.org/10.1007/s00779-011-0404-2

[18] Yasmin Kafai and Quinn Burke. 2013. Computer Programming Goes Back to School. *Phi Delta Kappan* 95 (09 2013), 61–65. https://doi.org/10.1177/003172171309500111

[19] Dennis Kafura and Deborah Tatar. 2011. Initial Experience with a Computational Thinking Course for Computer Science Students. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 251âĂŞ256. https://doi.org/10.1145/1953163.1953242

[20] Michael Kölling and Poul Henriksen. 2005. 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 '05)*. ACM, New York, NY, USA, 59–63. https://doi.org/10.1145/1067445.1067465

[21] Brian Krupp and Andrew Watkins. 2019. CS0: Introducing Computing with Raspberry Pis. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 832âĂŞ838. https://doi.org/10.1145/3287324.3287488

[22] Susan M Land and Michael J Hannafin. 2000. Student-centered learning environments. *Theoretical foundations of learning environments* (2000), 1–23.

[23] Scott Leutenegger and Jeffrey Edgington. 2007. A Games First Approach to Teaching Introductory Programming. *SIGCSE Bull.* 39, 1 (March 2007), 115–118. https://doi.org/10.1145/1227504.1227352

[24] Les M Lunce. 2006. Simulations: Bringing the benefits of situated learning to the traditional classroom. *Journal of Applied Educational Technology* 3, 1 (2006), 37–45.

[25] Sze Yee Lye and Joyce Hwee Ling Koh. 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior* 41 (2014), 51 – 61. https://doi.org/10.1016/j.chb.2014.09.012

[26] Pablo Martin-Ramos, Maria Joo Lopes, M. Margarida Lima da Silva, Pedro E.B. Gomes, Pedro S. Pereira da Silva, Jos P.P. Domingues, and Manuela Ramos Silva. 2018. Reprint of First Exposure to Arduino Through Peer-coaching. *Comput. Hum. Behav.* 80, C (March 2018), 420–427. https://doi.org/10.1016/j.chb.2017.12.011

[27] Jared J McGinley and Brett D Jones. 2014. A brief instructional intervention to increase studentsâĂŹ motivation on the first day of class. *Teaching of Psychology* 41, 2 (2014), 158–162.

[28] Alexandros Merkouris and Konstantinos Chorianopoulos. 2015. Introducing Computer Programming to Children Through Robotic and Wearable Devices. In *Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE '15)*. ACM, New York, NY, USA, 69–72. https://doi.org/10.1145/2818314.2818342

[29] Iwona Miliszewska and Grace Tan. 2007. Befriending computer programming: A proposed approach to teaching introductory programming. *Informing Science: International Journal of an Emerging Transdiscipline* 4, 1 (2007), 277–289.

[30] Shitanshu Mishra, Sudeesh Balan, Sridhar Iyer, and Sahana Murthy. 2014. Effect of a 2-Week Scratch Intervention in CS1 on Learners with Varying Prior Knowledge. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)*. Association for Computing Machinery, New York, NY, USA, 45âĂŞ50. https://doi.org/10.1145/2591708.2591733

[31] Jackie O'Kelly and J. Paul Gibson. 2006. RoboCode & Problem-based Learning: A Non-prescriptive Approach to Teaching Programming. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITICSE '06)*. ACM, New York, NY, USA, 217–221. https://doi.org/10.1145/1140124.1140182

[32] Vasileios Orfanakis and Stamatios Papadakis. 2017. Using Raspberry Pi and Wyliodrin for teaching novice programmers in Secondary Education.

[33] G.E. Palaigeorgiou, P.D. Siozos, N.I. Konstantakis, and I.A. Tsoukalas. [n.d.]. A computer attitude scale for computer science freshmen and its educational implications. *Journal of Computer Assisted Learning* 21, 5 ([n. d.]), 330–342. https://doi.org/10.1111/j.1365-2729.2005.00137.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2729.2005.00137.x

[34] Julie Pallant. 2013. *SPSS survival manual*. McGraw-Hill Education (UK).

[35] Stamatios Papadakis, Michail Kalogiannakis, Nicholas Zaranis, and Vasileios Orfanakis. 2016. Using Scratch and App Inventor for teaching introductory programming in secondary education. A case study. *International Journal of Technology Enhanced Learning* 8, 3-4 (2016), 217–233.

[36] J. Parham-Mocello, M. Erwig, and E. Dominguez. 2019. To Code or Not to Code? Programming in Introductory CS Courses. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 187–191.

[37] Rathika Rajaravivarma. 2005. A Games-based Approach for Teaching the Introductory Programming Course. *SIGCSE Bull.* 37, 4 (Dec. 2005), 98–102. https://doi.org/10.1145/1113847.1113886

[38] Samuel A. Rebelsky, Janet Davis, and Jerod Weinman. 2013. Building Knowledge and Confidence with Mediascripting: A Successful Interdisciplinary Approach to CS1. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE âĂŹ13)*. Association for Computing Machinery, New York, NY, USA, 483âĂŞ488. https://doi.org/10.1145/2445196.2445342

[39] Marcos Román-Gonzalez, Juan Carlos Pérez-González, and Carmen Jiménez-Fernández. 2015. Test de pensamiento computacional: diseño y psicometría general. In *III Congreso Internacional sobre Aprendizaje, Innovación y Competitividad (CINAIC 2015)*.

[40] Theodosios Sapounidis and Stavros Demetriadis. 2013. Tangible Versus Graphical User Interfaces for Robot Programming: Exploring Cross-age Children's Preferences. *Personal Ubiquitous Comput.* 17, 8 (Dec. 2013), 1775–1786. https://doi.org/10.1007/s00779-013-0641-7

[41] Stephen Schaub. 2009. Teaching CS1 with Web Applications and Test-Driven Development. *SIGCSE Bull.* 41, 2 (June 2009), 113âĂŞ117. https://doi.org/10.1145/1595453.1595487

[42] Christina Silver and Ann Lewins. 2014. *Using software in qualitative research: A step-by-step guide*. Sage.

[43] David Stein. 1998. *Situated learning in adult education*. ERIC Clearinghouse on Adult, Career, and Vocational Education, Center on âĂę.

[44] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A Systematic Review of Approaches for Teaching Introductory Programming and Their Influence on Success. In *Proceedings of the Tenth Annual Conference on International Computing Education Research (ICER '14)*. Association for Computing Machinery, New York, NY, USA, 19âĂŞ26. https://doi.org/10.1145/2632320.2632349

[45] Jacques Wainer and Eduardo C. Xavier. 2018. A Controlled Experiment on Python vs C for an Introductory Programming Course: Students’ Outcomes. *ACM Trans. Comput. Educ.* 18, 3, Article 12 (Aug. 2018), 16 pages. https://doi.org/10.1145/3152894

[46] Arthur L Wilson. 1993. The promise of situated cognition. *New directions for adult and continuing education* 1993, 57 (1993), 71–79.

[47] Janie H Wilson and Shauna B Wilson. 2007. Methods and techniques: The first day of class affects student motivation: An experimental study. *Teaching of Psychology* 34, 4 (2007), 226–230.

[48] Pat Wolfe and Ron Brandt. 1998. What Do We Know from Brain Research?. *Educational Leadership* 56, 3 (1998), 8–13.

[49] Zoë J. Wood, John Clements, Zachary Peterson, David Janzen, Hugh Smith, Michael Haungs, Julie Workman, John Bellardo, and Bruce DeBruhl. 2018. Mixed Approaches to CS0: Exploring Topic and Pedagogy Variance after Six Years of CS0. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 20âĂŞ25. https://doi.org/10.1145/3159450.3159592

[50] Sarita Yardi and Amy Bruckman. 2007. What is Computing?: Bridging the Gap Between Teenagers' Perceptions and Graduate Students' Experiences. In *Proceedings of the Third International Workshop on Computing Education Research (ICER '07)*. ACM, New York, NY, USA, 39–50. https://doi.org/10.1145/1288580.1288586