proceedings

Proceedings

Prof. Watson: A Pedagogic Conversational Agent to Teach Programming in Primary Education †

Pablo Yeves-Martínez [1] and Diana Pérez-Marín [2,*]

1 IBM, 28002 Madrid, Spain; pabloyeves@ibm.com
2 Universidad Rey Juan Carlos, 28933 Móstoles, Spain
* Correspondence: diana.perez@urjc.es

† Presented at the 13th International Conference on Ubiquitous Computing and Ambient Intelligence UCAmI 2019, Toledo, Spain, 2–5 December 2019.

Published: 21 November 2019

Abstract: Teaching programming in Primary Education has recently attracted a great deal of research interest. One global trend is using multimedia languages such as Scratch. However, it was our belief that by using Pedagogic Conversational Agents that dialog with the students, they have to think how to solve given problems and to write the code to solve them. In particular, the MECOPROG methodology was applied to design the student-agent dialog in Prof. Watson. An experiment with 19 students (11–12 years old) was carried out proving the viability of the approach, which shed some light into alternative procedures to teach programming in Primary Education.

Keywords: Pedagogic Conversational Agent; programming; Primary Education

---

1. Introduction

During the last years, teaching programming at young ages has gained significant relevance, setting new learning trends on Primary Education curriculum. The level of abstraction required to learn programming techniques remains an unsolved challenge, thus, resulting in the need for the development of novel intuitive and easy-to-use tools in order to help students develop the required knowledge to face programming exercises.

This work focuses on the analysis of basic programming concepts such as variable creation or input/output as a basis for more advanced concepts like conditional or loop statements. In this context, one of the worldwide preferred tools to teaching programming in Primary Education is Scratch [1], where the students are required to drag and connect grouped instructions to create different programs. These can be run in a multimedia environment with colored backgrounds, music and movement. This first approach may be useful for a teacher to work on the variable and output concepts, e.g., students can be asked to write a program resulting on the cat saying its name and modify it later on by varying a given value (notice that with a hypothetical "ask" instruction, the input concept could have been included too).

Alternatives to Scratch for teaching programming at Primary Education schools may be based on the Lego WeDo, Mindstorms EV3 robots [2] or even "unplugged" approaches if technological resources are not available [3], such as Code.org printable exercises. However, the effectivity of these approaches has not yet been proven [4].

In addition to the previously described approaches for teaching programming at Primary Education levels, there are multiple Pedagogical Conversational Agents [5]; i.e., interactive systems that engage in a conversation with the students about a specific knowledge area. These agents present specific benefits for the students learning path since including a conversational agent in a learning environment has been proven to produce three positive effects: (i) the Persona Effect [6], causing a positive result on the perception of the learning experience by the student, (ii) the Proteo Effect [7], where students, motivated by the agent characteristics, try to resemble to the agent, learning from them, and (iii) the Protégé Effect [8], where students adopt the teacher role and make an effort to make the agent learn from them.

In order to avoid students' distraction induced in multimedia programming environments, and to bring the students closer to the written programming syntax while keeping the interactive experience, a conversational agent was developed in this work. Previous studies have proven the benefits of the "conversational programming" approach, i.e., to make programming more conversational [9,10]. The core idea is to help students to identify the discrepancies between the programs they intend to write and their actual programming results in a dialogue with the computer.

To guide the dialogue, the MECOPROG methodology [11], described in Section 2, is used. MECOPROG is based on the use of metaphors to teach the basic programming concepts, lowering the abstraction level and trying to get the students to solve the programming challenges without extra multimedia. The reason to use metaphors is that previous research reveals the benefits of using metaphors to teach programming in Primary Education [12].

However, given that it is not always possible to use real-world cooking scenarios with physical programming input devices such as a Wiimote, the use of a computer software with keyboard and mouse input devices was chosen. In particular, Section 3 describes Prof. Watson, a conversational pedagogical agent based on IBM Watson Assistant services.

Dr. Watson has the aim of teaching basic programming concepts step by step and in a dynamical way, offering a battery of exercises to be solved by the students. A pilot experiment with 19 students, 11–12 years old, in a Spanish school is described to prove the approach feasibility. In the school, the students could only use computers with keyboard and mouse. Finally, Section 5 closes the article with the main conclusions derived from this work and future work guidelines.

2. Mecoprog

MECOPROG [11] is a teaching methodology for programming in Primary Education based on the use of metaphors. These metaphors are specifically chosen to catch the students' attention and bring the programming world closer to their daily life. In particular, a common example of this metaphors is cooking, i.e., the concept of a program is shown as a recipe, where a sequence of steps needs to be followed one after the other to achieve a result.

Table 1 groups the metaphors used for learning basic programming concepts through the MECOPROG methodology. The amount of time to be dedicated on each metaphor group depends on the students' answers. The rule is that a new group should not be started until the previous one has been properly assimilated. The aim is to create a meaningful learning [13]. Each metaphor group is further explained in [11] for the teachers use in a classroom.

Table 1. Example of metaphors [11].

ID	Script	Concept	Metaphor
M1	Program, programming, sequence, memory	Program, sequence Memory, variables	Recipe Pantry, box
M2	Input/output instructions	Output Input	PC Screen Keyboard, reflex on PC
M3	Conditionals	Child making decisions	Computer making decisions
M4	Loops	You, laying the table for X people	Computer repeating instructions X times

3. Prof. Watson

Prof. Watson is a web agent powered by IBM Watson artificial intelligence (AI) technology, able to develop and follow a conversation to support the students on the syntax, methodologies and programming paradigm understanding. Through an interactive dialogue, Watson can be used to get the student involved in the understanding of programming concepts by means of an easy to understand high level programming language, taking into account the students' age and previous experience with code.

The application is displayed as a chat interface, showing a welcome message sent by the AI Watson Assistant service, encouraging the students to go through a tutorial on how to use the environment. After finishing this tutorial, a battery of exercises is offered to check if the users have acquired the knowledge as expected.

The agent contents are: Input/Output, variable creation, variable display, screen printing, conditionals, operators (comparators), "if" conditional structure, "if then" conditional structure, "if then else" conditional structure, loops and repeat structure. These contents are displayed in a sequential way throughout the tutorial, and exercises offer them in the same order. Figure 1 shows a sample conversation between a student and Prof. Watson agent (shown left) and the created variables on the right-hand side of the figure.

The conversation is the main element of the agent. It uses pre-defined intentions and entities, and context variables as well, in order to interact with the user and offer the most relevant answers depending on the situation. A context variable is a reference that can be used during the course of the conversation to keep previous states, previous user inputs or any other relevant information to be kept for future answers.

Context variables are defined in the IBM Watson Assistant environment by the dollar sign ($) and consist of a value and a name. The conversation used by Prof. Watson uses different context variables, such as the student's name, defined as $nombreAlumno, the list of names of the variables created by the student and their values, among others. To access to the system context, SpEL functions [11] are used, based on the Spring regular expression language.

In the IBM Watson Assistant conversation environment, the nodes responsible of guiding the user throughout the conversation flow are defined. A node can have children, evaluated in a pre-defined order. In the circumstance that the condition defining a node is fulfilled, the node is executed. If a node is executed, its following child is not evaluated.

Hello! I'm Professor Watson! What's your name?

I'm Pablo Yeves

Hi, Pablo! What do you want to do today?

I'd like to do the tutorial, please

Great! I'll show you how to create a variable. A variable consists of a name and a value. To create a variable, you can write **name = value**. Try to create a variable named apples with the value you like.

apples = 34

Right! Now we have a variable **apples** with a value of **34**, as you can see on the right column. Let's practice now functions. A function is like an action. Do you want to learn how to do them?

sure!

Ok, ask me to write the value of the variable **apples**. You can do it with **write(apples)**. Now it's your turn.

write(apples)

34
Great! Did you know that if you ask me to write between quotation marks "like this" you don't need to create a variable? For example, try to ask me to write the word you like without creating a variable.

write(apples)

Oh, you've forgot the quotation marks…

Write("apples")

"apples"
That's it! Let's go to conditionals now. Do you know what they are?

no...

That's easy, Pablo. A conditional is a structure like **if… then…** Do you want to know a bit more about conditionals?

Ok!

Very well! You can make different types of conditionals, like: **if(4>2)** or **if(apples==3)**. You can compare numbers or variables. Do you need a reminder of the operators?

Figure 1. Example of conversation between a student and Prof. Watson. The conversation has been translated to English for the sake of English readers**.** However, originally the dialogue is in Spanish. The system can be used on-line at profesor-watson.eu-gb.mybluemix.net (only in Spanish).

A node has the following structure:

1. **Node name:** The node name allows its identification. It is not required to name a node, but it is advised to be done in order to differentiate nodes between them.
2. **Condition:** The condition is the expression to be evaluated to execute the node. If it is not fulfilled, the following node is evaluated. In the conditions, specific patterns can be included to match the student's input, and intentions or entities detected by Prof. Watson as well.
3. **Context:** Allows to create or modify context variables if a node condition is fulfilled. This is useful to store the user's given information or save specific conversation states.
4. **Answer:** Represents the text to be given by the system when the condition is evaluated and the required context variables are modified, if necessary.
5. **Final action:** The final action, defined on the "and finally…" field, states what needs to be done after displaying the answer. The service allows three different options: wait for the user to give another input, skip the user input (if the processed node has children) or jump to a different node, waiting to the user reply, evaluating a condition or directly answering with the reply of the pointed node.

4. Experiment

A session involving 19 students of 6th grade of Primary Education (11–12 years old) was performed in a computer science class in a public school of the Madrid region in Spain to prove the feasibility of this work. Around 50% of the students were boys and the rest girls.

Students were allowed to access the system through a given URL (profesor-watson.eu-gb.mybluemix.net) during the class at the computer room of the school, giving them freedom and autonomy to solve the tutorial, and a fraction of the proposed exercises due to time constraints.

At the end of the session, a questionnaire was given to the students to gain insights on their perception about the class. From this, it was revealed that an 89.5% of the students had previously studied programming (notice the high percentage, as in the school where the experiment was made, programming is taught from 4th grade).

Furthermore, Figure 2 shows the "Ranked value" in x axis versus the "Perceived learning level" in y axis for each questionnaire. The results indicate that those students that ranked the class with higher levels also perceived a higher learning level, a blue line corresponding to the least square linear regression has been plotted with illustrative purposes only. Additionally, it is remarkable that the two students that did not previously study programming (corresponding to the 10% of the class) ranked the experience with values of 7 and 9, with a perceived learning value of 8 and 10.

Moreover, as shown in Figure 3a,b, most students evaluated the class as very entertaining, and considered that they have learnt new programming concepts (79% gave a score equal or greater than 8). Regarding Prof. Watson grading, as shown in Figure 4, most students gave an overall neutral score in terms of conversational level. The name understanding by the system was ranked very positively, as it was understood for almost every student. Tutorial length scoring was mainly neutral, with some prevalence to its short duration. Regarding the quantity of examples, students stated a wish of having more to practice with it.

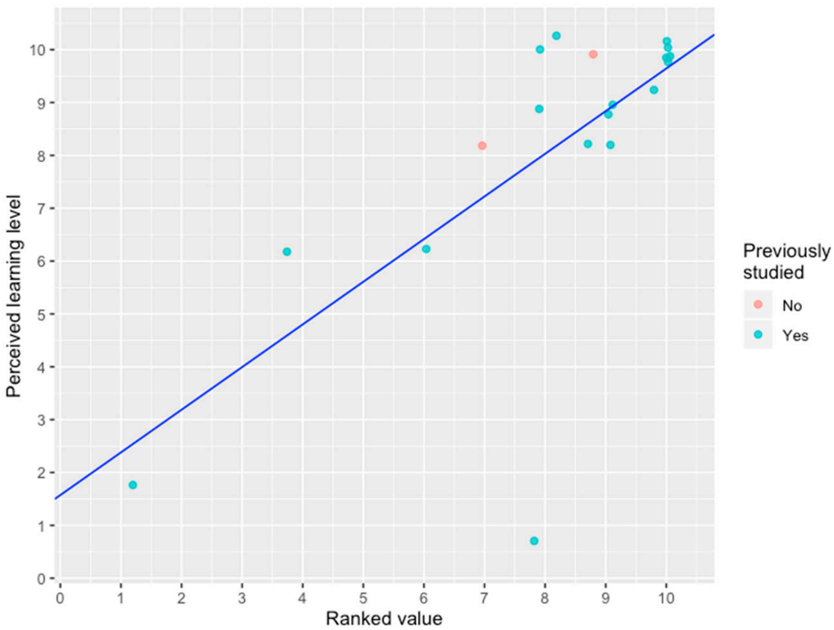

Figure 2. Scatter plot with "Ranked value" in x axis and "Perceived learning level" in y axis for each questionnaire. Jittering has been applied to the avoid data overplotting. Red dots indicate that the student did not study programming before, while blue dots indicate the student had previously studied programming. Furthermore, the blue line corresponding to the least square linear regression has been added for illustrative purposes.

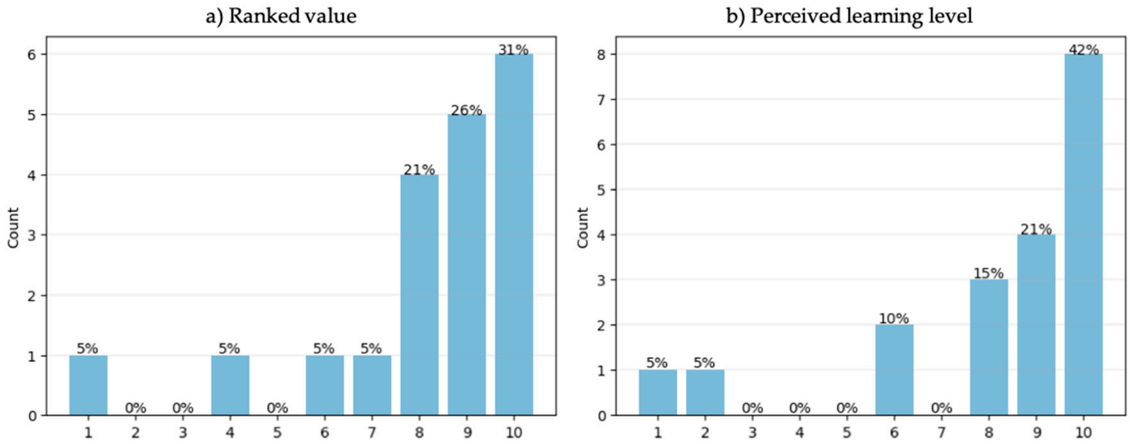

Figure 3. (**a**) Ranked value defined as votes on the question "How entertaining was the lesson?", 1 being the minimum and 10 being the maximum value. (**b**) Perceived learning level defined as votes on the question "Do you think you've learned new things today?" to be valued from 1, being the minimum and 10 being the maximum number of concepts learnt.

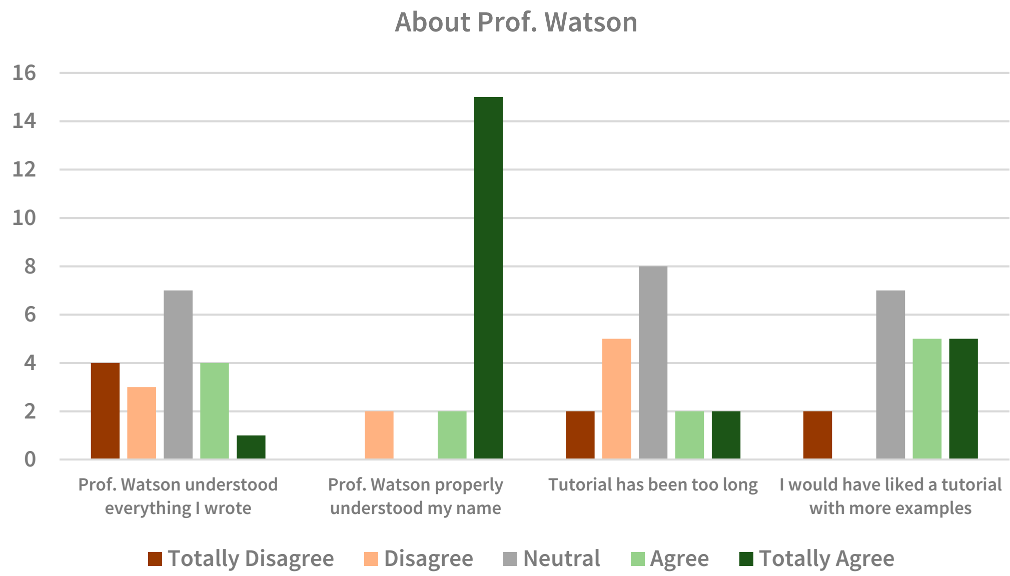

Figure 4. Prof. Watson grading scale.

Considering that the questions posed might be somehow closed, the possibility of the students to say what they thought about the course and what they would enhance was provided to them. Figure 5 shows a summary of the most common comments stated by the students, serving as an analysis of their open answers.

On the question "What aspects of the course did you find more useful?", the variable use and creation were the main aspects, probably because of the way the system works with them, differently from *Scratch*, showing them live on the screen as they are created, in a very visual way. Appreciation to the system is also stated, as well as the use and creation of conditionals.

Finally, regarding ways of improving the system, 15.8% of the students would not enhance the system, and a remarkable 26.3% of the users stated that they should study or practice more before using the system to leverage the full potential of the tool.

Apart from that, the 19 conversations of the students with its associated data were collected. Those generated logs contained relevant information, such as the number of nodes visited per student, the conversation stream they followed or the time the session took. The average conversation session time was 21 min.

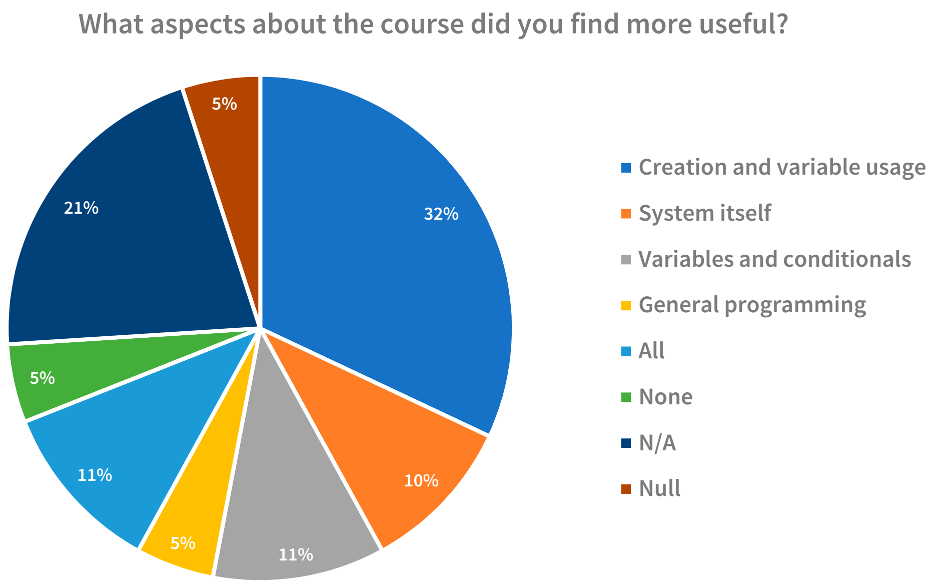

Figure 5. Open comments summary of the students about Prof. Watson contents.

5. Conclusions and Future Work

The design and implementation of conversational agents intended for learning programming present relevant advantages over those used for learning other areas of study. In particular, due to the programming languages structured character and their solid syntax, it is easier to structure, to detect and to code specific intentions, such as variable creation, and conditional or loop statements.

As shown on the acceptance tests, Prof. Watson as a sample conversational agent to teach programming in Primary Education was positively welcomed among the students, being a dynamic and amusing way of understanding and consolidating the programming aspects.

Conversational agents offer a wide range of possibilities, and in particular, this project can be spread in different ways, for instance by adding IBM's Watson Speech To Text and/or Text To Speech, allowing a voice input and output interface to enhance the system accessibility.

On the other hand, the use of cookies or databases may be useful to save the students' progress across sessions, by implementing login credentials when starting the system. There are some storage services available on IBM Cloud, exposed as services, that could be connected to the application and would allow the system to track the users' progress.

The possibility of adding new exercises by a teacher is also a feature that may result interesting to provide increased versatility to the system. Nowadays the exercises are coded with the IBM Watson Assistant web tool, but the fact that the service offers an API to create nodes and conversational flows through a customized, easier interface, will allow people non familiarized with the IBM environment to add new functionalities in an easier way.

Another improvement to be considered would be a tool to help the student on the syntactical errors that they may make on writing the code, such a debugger. Even if the Watson Assistant conversation can be made to detect specific patterns and be more flexible on the code syntax, it is relevant to emphasize to the students the proper use of brackets, parenthesis and keywords.

References

1. Resnick, M.; Maloney, J.; Monroy-Hernandez, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; et al. Scratch: Programming for all. *Commun. ACM* **2009**, *52*, 60–67.

2. Sović, A.; Jagušt, T.; Seršić, D. How to teach basic university-level programming concepts to first graders? In Proceedings of the IEEE Integrated STEM Education Conference (ISEC), Princeton, NJ, USA, 8 March 2014; pp. 1–6.
3. Brackmann, C.; Barone, D.; Casali, A.; Boucinha, R.; Muñoz-Hernandez, S. Computational thinking: Panorama of the Americas. In Proceedings of the International Symposium on Computers in Education (SIIE), Salamanca, Spain, 13–15 September 2016; pp. 1–6.
4. Kalelioğlu, F. A new way of teaching programming skills to children: Code. Org. *Comput. Hum. Behav.* **2015**, *52*, 200–210.
5. Johnson, W.L.; Rickel, J.W. Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments. *Int. J. Artif. Intell. Educ.* **2010**, *11*, 47–78.
6. Lester, J.C.; Converse, S.A.; Kahler, S.E.; Barlow, S.T.; Stone, B.A.; Bhogal, R.S. The Persona Effect: Affective Impact of Animated Pedagogical Agents. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '97, Atlanta, GA, USA, 22 March 1997; doi:10.1145/258549.258797.
7. Yee, N.M.; Bailenson, J. The proteus effect: The effect of transformed self-representation on behavior. *Hum. Commun. Res.* **2007**, doi:10.1111/j.1468-2958.2007.00299.x.
8. Chase, C.C.; Chin, D.B.; Oppezzo, M.A.; Schwartz, D.L. Teachable agents and the protégé effect: Increasing the effort towards learning. *J. Sci. Educ. Technol.* **2009**, doi:10.1007/s10956-009-9180-4.
9. Cameron, S.H.; Ewing, D.; Liveright, M. DIALOG: A conversational programming system with a graphical orientation. *Commun. ACM* **1967**, *10*, 349–357, doi:10.1145/363332.363343.
10. Repenning, A. Making programming more conversational. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Philadelphia, PA, USA, 18–22 September 2011; doi:10.1109/VLHCC.2011.6070398.
11. Pérez-Marín, D.; Hijón-Neira, R.; Martín-Lope, M. A Methodology Proposal Based on Metaphors to Teach Programming to Children. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje* **2018**, *13*, 46–53, doi:10.1109/RITA.2018.2809944.
12. Tarkan, S.; Sazawal, V.; Druin, A.; Golub, E.; Bonsignore, E.M.; Walsh, G.; Atrash, Z. Toque: Designing a cooking-based programming language for and with children. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Atlanta, GA, USA, 10–15 April 2010; pp. 2417–2426, doi:10.1145/1753326.1753692.
13. Ausubel, D.P.; Novak, J.D.; Hanesian, H. *Psicología educativa: un punto de vista cognoscitivo*; Trillas: México city, México, 1983; Volume 2.