Creating the computer player: an engaging and collaborative approach to introduce computational thinking by combining 'unplugged' activities with visual programming

Creare il giocatore computerizzato: un approccio collaborativo e coinvolgente per introdurre il pensiero computazionale combinando attività 'unplugged' e programmazione visiva

Anna Gardeli and Spyros Vosinakis*

University of the Aegean, Greece, agardeli@aegean.gr, spyrosv@aegean.gr*

* corresponding author

HOW TO CITE Gardeli, A., & Spyros, V. (2017). Creating the computer player: an engaging and collaborative approach to introduce computational thinking by combining 'unplugged' activities with visual programming. *Italian Journal of Educational Technology, 25*(2), 36-50. doi:10.17471/2499-4324/910

ABSTRACT Ongoing research is being conducted on appropriate course design, practices and teacher interventions for improving the efficiency of computer science and programming courses in K-12 education. The trend is towards a more constructivist problem-based learning approach. Computational thinking, which refers to formulating and solving problems in a form that can be efficiently processed by a computer, raises an important educational challenge. Our research aims to explore possible ways of enriching computer science teaching with a focus on development of computational thinking. We have prepared and evaluated a learning intervention for introducing computer programming to children between 10 and 14 years old; this involves students working in groups to program the behavior of the computer player of a well-known game. The programming process is split into two parts. First, students design a high-level version of their algorithm during an 'unplugged' pen & paper phase, and then they encode their solution as an executable program in a visual programming environment. Encouraging evaluation results have been achieved regarding the educational and motivational value of the proposed approach.

KEYWORDS Computational thinking, Computer science, Project-based learning, Collaborative learning, Unplugged activities, Visual programming.

SOMMARIO Molte ricerche in corso si occupano di come progettare efficacemente corsi, pratiche e interventi didattici volti a migliorare l'efficienza dei corsi di informatica e di programmazione per la scuola. La tendenza è verso un approccio all'apprendimento costruttivista e orientato alla soluzione di

problemi. Il pensiero computazionale, che fa riferimento alla formulazione e risoluzione di problemi in una forma che possa essere efficacemente elaborata da un computer, sollecita un'importante sfida educativa. La nostra ricerca mira ad esplorare modi possibili di arricchire l'insegnamento dell'informatica che si focalizzi sullo sviluppo del pensiero computazionale. Abbiamo preparato e valutato un intervento didattico per introdurre alla programmazione ragazzi dai 10 ai 14 anni. Gli studenti lavorano in gruppo a programmare il comportamento del giocatore computerizzato per un gioco a loro ben noto. Il processo di programmazione è suddiviso in due parti: gli studenti progettano una versione di alto livello del loro algoritmo durante una fase "unplugged" con carta e matita, per poi codificare la loro soluzione in un ambiente di programmazione visiva. Dalla valutazione dell'intervento sono emersi risultati incoraggianti sul valore educativo e motivazionale dell'approccio proposto.

PAROLE CHIAVE Pensiero computazionale, Scienze informatiche, Apprendimento basato su progetti, Apprendimento collaborativo, Attività unplugged, Programmazione visuale.

1. INTRODUCTION

Computational thinking is not simply about understanding how machines work or learning a programming language. It refers to formulating and solving problems in a form that can be efficiently processed by a computer, involving a set of skills considered essential for a wide variety of disciplines (Bundy, 2007). As such, computational thinking raises an important educational challenge. It is fundamental for programming and also related to many of 21st century abilities such as creativity, critical thinking, and problem solving. In the recent years the educational community has focused on introducing computer science and computational thinking skills in K-12 education.

The teaching of introductory computer science courses to elementary and high-school students has benefited from the emergence of numerous visual programming languages, such as Scratch (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010), Alice (Dann et al., 2011) and Greenfoot (Kölling, 2010). These environments allow learners to focus on the essential procedure of constructing a program, where computational thinking and practices come to the surface, without worrying about complex programming syntax. The utilization of an easy-to-use computer environment alone is, nevertheless, not sufficient for the effective acquisition of computational thinking skills. Ongoing research is being conducted on appropriate course design, practices and teacher interventions for improving the efficiency of computer science and programming courses in K-12 education. In a recent review about teaching computational thinking through computer programming, Lye and Koh (2014) argue that there is an apparent gap in this research area for K-12 students. They propose a number of directions towards a more constructivist problem-based learning approach, such as the use of an authentic and engaging problem, the adoption of information processing strategies, scaffolding of program construction, and support for student reflection.

Our research aims to explore possible ways of enriching computer science education in the aforementioned directions and to assess the efficacy of the proposed approach. We have prepared a learning intervention for introducing computer programming to children between 10 and 14 years old; this involves students working in groups to program the behavior of the computer player of a well-known game. The programming process is split into two parts. First, students design a high-level version of their algorithm during an 'unplugged' pen & paper phase, and their output is evaluated by the teacher, who plays the role of the human computer. Then, they encode their solution as an executable program in Scratch, where they can test their algorithm in action and play the game against the computer. We set up a case study to evaluate the intervention in terms of its reception by the students and its efficiency as a learning approach. We exam-

ined a number of aspects, such as the frequency and types of errors made in the two phases, the quality of collaboration between the student groups, and the students' satisfaction regarding the process. Encouraging results have been achieved concerning the educational and motivational value of the proposed approach.

2. THEORETICAL FRAMEWORK

2.1. *Teaching computational thinking*

Computational Thinking is the thought processes involved in formulating a problem and expressing its solution in a way that a computer—human or machine—can effectively carry out (Wing, 2008). It is about logically organizing and analyzing data, representing data through abstractions such as models and simulations, automating solutions through algorithmic thinking (a series of ordered steps), identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combinations, and generalizing and transferring this problem solving process to a wide variety of problems (Sáez-López, Román-González, & Vázquez-Cano, 2016).

While computational thinking may be essential in nearly all disciplines, both in the sciences and the humanities (Bundy, 2007), it remains the core of computer science. Lu & Fletcher (2009) argue that programming is to computer science (CS) what proof construction is to mathematics and what literary analysis is to English. Therefore, programming should not be included in basic CS classes before students have had substantial practice thinking computationally. Programming requires many cognitive skills such as perception, abstraction, analytical thinking and problem solving abilities that are far more important than the syntax of a programming language. Ivanovic, Budimac, Radovanović, & Savić (2015) argue that after adoption and fundamental understanding of the basic (imperative) programming principles, students can easily move to other programming languages. Accordingly, they propose a model for Teaching First Programming Language in which the first step is to teach students to think in an algorithmic way, independent of the programming language.

Despite growing evidence to support the integration of CS into K-12 education, there are also misconceptions and inaccurate perceptions (Armoni, 2011). Often students give up computer science because they think it is boring, confusing, and too difficult to master (Wilson & Moffat, 2010).

Such difficulties may be partially addressed with the use of Visual Programming tools. These environments adopt a visual way of creating code through manipulation of block-based structures. The blocks can fit together only in specific ways, thus producing code that is syntactically correct. This relieves students from the burden of having to learn the complex syntactic patterns of a programming language, and prevents mistakes due to accidental typing or syntax errors. This is especially important for introductory programming, as it saves learners from the typical difficulties of mastering a textual programming language (Wilson & Moffat, 2010, p. 70). Scratch is a well-known educational program based on the visual programming paradigm. It lowers the bar to programming, empowering first-time programmers not only to master programmatic constructs before syntax but also to focus on problems of logic before syntax (Kordaki, 2012).

Nevertheless, the use of these tools alone does not guarantee effective teaching of computational thinking because students tend to use them in a trial-and-error mode rather than thinking as they are doing. Students need guidance in order to focus on the programming process, which is the key to overcoming some of the difficulties that novice programmers face. Sorva, Karavirta, & Malmi (2013), referring to the challenges of learning programming concepts, mention static perceptions of programming, difficulties understanding the computer, misconceptions about fundamental programming constructs, and struggles with tracing and program state. All four of these relate to the lack of a suitable cognitive model on how the computer really

works, not the lack of knowledge of programming language syntax.

2.2. *Activity – based learning*

Today's educators are leveraging technology tools that set challenges focused on active learning in educational contexts. These learning approaches are focused on teaching methodology and student-centered designs (Saez-Lopez et al., 2016).

Active learning is a type of learning that requires student participation. This differs from the traditional teaching model, where the educator gives a lecture while students listen and take notes – a possibly cause of learning disengagement for many students. The National Training Laboratories' pyramid of learning shows that students retain only 5% of the information passed to them in the form of lectures, while with learn-by-doing knowledge retention is 75%. Furthermore, students retain 90% of what they have learned when they teach each other (Lai, Luong, & Young, 2015).

There are different techniques for implementing active learning in class. Faust and Paulson (1998) have presented six types, which include exercises for individual students, questions and answers, immediate feedback, critical thinking motivators and learning in pairs, e.g. pair programming. The last technique is a cooperative learning strategy, which requires students to be divided into groups and enhances their communication and social skills (Lai et al., 2015). Active learning techniques have been used in teaching computational thinking and programming concepts, as well as in other disciplines and courses.

Code.org (Kalelioğlu, 2015) is a blocky coding platform that offers online activities and courses designed to motivate both individual students in self-directed learning and educators in using programming activities in class. Besides blocky activities, the site propose a list of unplugged lessons. These activities can teach the fundamentals of computer science, whether there are computers in the classroom or not, and can be used as a standalone course or as complementary lessons for any computer science course. A similar "unplugged" project has been proposed by Canterbury University. Bell, Alexander, Freeman, & Grimley (2009) describe activities such as games, magic tricks and competitions to show children the kind of thinking that is expected of a computer scientist. These activities involve active and kinesthetic learning. Bell et al. also suggest that an interesting area worthy of further research is to investigate the value of combining "unplugged" activities with programming in visual programming languages such as Scratch and Alice.

2.3. *Games in the classroom*

Game-based learning activities shift from a traditional teacher-centered learning environment to a student centered environment in which the students are much more active and engaged (Sung & Hwang, 2012). Prensky (2007) argues that students are increasingly convinced that school is totally devoid of interest and totally irrelevant to their lives because school lacks engagement. Today's students are used to having those around them - musicians, movie makers, TV stars, game designers - striving hard for their attention . This is what they are also expecting from their educators: to be challenged and engaged.

Zhang, Kaufman, & Fraser (2014) present four main types of implementing games as educational tools in computer science (CS) education: (a) using games to motivate students, (b) making games to teach CS topics, (c) using games as environments to teach CS topics, and (d) using games as examples to teach CS topics. The main topics of CS, where games have been applied as an educational tool are: theory of computation, algorithms and data structures, programming methodology and languages, and computer elements and architecture.

In their study of evaluating games for teaching CS, Gibson & Bell (2013) include unplugged games, besides desktop, mobile and browser based games. Unplugged games are described as "not computer programs but they are played using physical objects such as cards, pen and paper" in a context of game-like teaching

methods. Research also suggests that exposure to rapid game design activities is effective in motivating students on computer science topics.

3. DESIGN OF THE LEARNING INTERVENTION

We have prepared a learning intervention for introducing computational thinking to young students through collaborative programming of the computer player in a known game. Our approach adopts a two-step process, in which students initially design a high-level solution through an unplugged activity, and then refine, encode and test their solution in a visual programming environment.

The main principles of our approach are:

- Using game design as a challenging and motivating factor. The proposed programming activity concerns the creation of a virtual opponent for human or computer players in a computer game (Huang, 2001). The use of a game as a subject, and the fact that students can test their solution by playing against the computer player they created, are expected to increase their motivation and interest. Other advantages are that games are usually gender neutral, children learn in a playful way, and are reasonably error resilient, so that small errors do not ruin the primary outcome (Bell et al., 2009).
- Solving the problem in two stages: unplugged pen & paper and then visual programming. This approach allows students to familiarize with fundamental CS concepts and trains their computational thinking skills, prior to using the visual programming environment. When they formulate and test their algorithm, they can convert it into actual executable code in the Scratch environment and then watch it in action.
- Collaboration in small groups. In the learning activity, students form groups of two or three and work together in both stages (unplugged & programming). Student collaboration is expected to stimulate interaction between group members as they share their knowledge and understanding, and may possibly result in a refined solution.

The learning intervention takes place in a classroom and a computer laboratory, and is addressed to students between 10 and 14 years old with little or no prior experience in programming. The included activities aim to get students in touch with basic concepts of programming, such as loops, if-statements and variables, by teaching them to think in an algorithmic way so that they can transform an algorithm into a computer program.

The selected game for our intervention is the card game UNO. UNO was selected because (1) it is a fun game that most of the students of both genders enjoy playing, (2) students are already familiar with the game so they don't have to analyze and understand the problem they are being asked to solve, and (3) the rules of the game can be simplified, so that no kind of 'smart' strategy is demanded of the programmable computer player, which would make the algorithm more complicated.

The standard UNO deck has 108 cards 80 numbered cards in four different colored suits (Blue, Green, Red, and Yellow), each with 20 cards; 28 action cards (Draw 2, Reverse, Skip, Wild, and Wild Draw Four) that can be used, for example, to force another player to skip their turn, to draw extra cards, etc. The actions card were excluded from our intervention because their deployment entails some kind of play strategy and this would increase the difficulty of programming as the computer player would need to be 'smart' enough to know when and how to use them.

The process runs in two stages. In the first, student groups use a pen and paper algorithm to create the computer player, without using a programming environment. This is the preparation stage, where they analyze the problem and focus on the step-by-step procedure that will give the right result, without being concerned with any syntax rules. In this stage, the algorithm is "run" by human acts, where the instructor plays the role of the computer that executes the algorithm.

In the second stage, students are asked to transform their algorithm into a program in the Scratch environment. Starting from a pre-built version of the game, students have to encode the operation of the computer program. At any time, they can execute the program and test their solution by playing the game.

Figure 1. Unplugged activities during the workshop.

4. CASE STUDY

We implemented a case study of the learning intervention described above, in order to examine the educational value of the proposed activity – based learning model in teaching computational thinking. The study focuses on the difficulties that students face in converting algorithms into programs, and on the motivational and engaging power of using games as a programming exercise.

4.1. *Context and process*

The case study took place in the Technology Laboratory of the Eugenides Foundation, UTech Lab, in Athens. UtechLab aims to put students in touch with technology through workshops and other activities.

For the unplugged activity, we prepared the "paper blocks", a set of blocks printed on paper similar to the ones used in visual languages. These represent the various actions of the game in physical language, such as "PLAY A CARD", as well as typical programming structures found in introductory languages, such as "IF...ELSE". These cards had to be used by the students to construct their algorithm as a potential solution (Figure 1). Their algorithm was then tested by playing with physical cards and having the teacher play the role of human computer, i.e. executing their algorithm.

Figure 2 shows an example of the proposed solution of the algorithm in pseudocode, as represented by the paper blocks.

```
BEGIN
        When it is your turn to play
        CARD  - first card from your hand
        REPEAT
                IF CARD has same color or number as the top card of the discard pile THEN
                        throw CARD
                        your turn is over
                CARD = next card
        UNTIL no more cards in hand
        CARD = draw a card
        IF CARD has same color or number as the top card of the discard pile THEN
                throw CARD
                your turn is over
        ELSE
                your turn is over
END
```

Figure 2. Example solution expressed in pseudo-code.

For the visual programming activity, we prepared a simplified version of the UNO game in Scratch, in which students had to complete the computer player code only (Figure 3). Students could use any of the programming constructs available in Scratch to create their code. They had access to the deck of cards of the computer player (represented as a list), the card in play on the table (represented as number and color), and the following actions: a) draw a card from the deck, b) play a card, and c) pass their turn. Combining those actions with Scratch's programming constructs, they were able to design a simplified behaviour of a computer player. A visual programming language is more similar to physical language than to common programming languages, but it is still a programming language. Consequently, in this activity the difficulty increased, due to the fact that more programming constructs were used. These include variables, lists and counters, which replace phrases like "the card is the same colour or number as the top card in the discard pile". That said, the goal was not to demand deep knowledge of managing programming elements, such as creation or initialization of variables. Therefore, some of the extra commands were given to the students, as an addition to Scratch's programming constructs. For example, the phrase "go to the next card" should be replaced by a counter. The initialization of the counter was given, so the students have a head start.

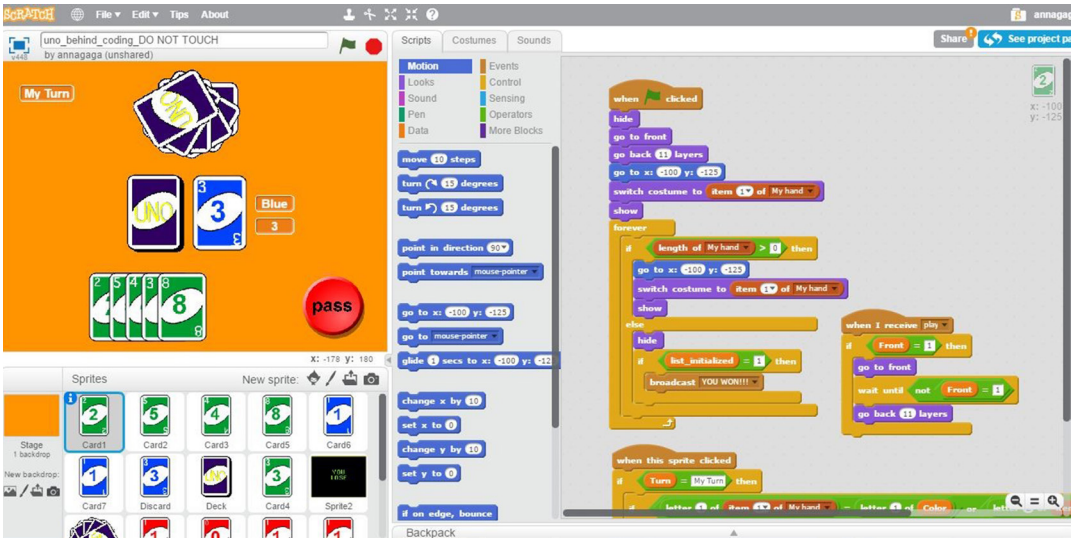

Figure 3. Screenshot of the UNO game implemented in Scratch.

The process during each student workshop was the following:

- introduction to computational thinking: presentation of fundamental concepts of algorithmic thinking and the value of programming in everyday activities - 15 minutes;
- presentation of the game and goals of the laboratory: the rules of UNO game and explanation of the process - 10 minutes;
- team formation: students form groups of three - 5 minutes;
- unplugged activity: student groups have to provide a solution using paper blocks and test it with the instructors - 30 minutes;
- visual programming activity: groups encode their solution in Scratch and test the resulting program - 30 minutes;
- wrap up discussion: students express their general impression and comments and fill in written questionnaires - 15 minutes.

4.2. *The participants*

The participants were 60 students in total (19 female and 41 male) aged between 10 and 14 years old. They were all primary and secondary school pupils in Athens, Greece. Therefore, their background knowledge of the subject, as well as their cognitive development, were similar, regardless of their gender. The case study took place in five workshops, each of which included 12 students arranged in four groups of three, with both female and male students.

4.3. Research design and methods

The aim of our study was to evaluate the proposed learning intervention in terms of student enjoyment, learning effectiveness and collaboration. The data collection methods we used during the study were:

- Observation: students' individual and group behaviour was continuously observed by the instructor to evaluate their engagement, work progress and collaboration;
- error log and analysis: the observed errors in student solutions were logged during both stages. The results were further analysed to identify unique errors and their frequency, and to categorize them.
- think aloud: students were probed to say out loud their actions and the expected outcome when testing their algorithms in both stages, in order to get more insight into their errors;
- evaluation and timing of solutions: the time to completion of the solutions was logged in both stages , and the solutions were evaluated by the instructor.
- questionnaires: students filled in a standard questionnaire provided by Utech Lab, in which they expressed their degree of satisfaction regarding the activities;
- open discussion: students were asked to give their general impression, and to comment on any positive or negative aspects of their experience.

4.4. *Pilot experiment*

Prior to the case study, we ran a pilot experiment to test and improve the pen & paper activity. The experiment took place in the same settings as the case study, with the participation of 24 students arranged in groups of three. The workshops included only the first part, i.e. the unplugged pen & paper activities.

The feedback from the pilot experiment was positive in general, though we observed some misunderstandings of the paper blocks. For instance, the IF ELSE statement consisted of two separate paper blocks, therefore syntax errors were allowed. We shifted to a partial redesign of the paper blocks, so as to avoid syntax errors. This way, we were able to focus on the difficulties novices experienced in understanding computational concepts, considering only logical mistakes.

5. RESULTS

Regarding the satisfaction factor, the majority of the students enjoyed participating in the lab; to the question "In general, how satisfied are you by the lab in which you have participated?" 78% replied "very satisfied", 14% "somewhat satisfied" and only 8% "neither satisfied nor dissatisfied". Enjoyment was also apparent through observation of students' reactions and comments. For instance, some students asked to stay longer to complete the task or to continue playing with the computer player they had created. It was important that the fun should not be over when they had completed the tasks, as they were also able to interact with the outcome of their work.

The average time for completing the task in the second activity (21' 40'') was lower than the first (23' 25''), despite the increased difficulty. While all the groups managed to complete the first activity, where the instructor "ran" the algorithm as the computer player while playing with physical cards, 8 out of the 20 groups (~40%) did not complete the second activity: their Scratch programs did not work as they should have and they were not able to actually play against their computer player. Based on the final deliverables of the activities, combined with the thinking out loud and observation outcome, we can argue that six of these groups did not manage to complete the task because they ran out of time. In order to proceed with the second activity it was necessary to have completed the first, so in two workshops where the students spent more than 30' on the first activity, some groups lagged behind. Observing them during the activities, we argue that with more time these groups would have completed the activity. Therefore, only two groups did not work out as well as we expected regarding understanding and collaboration. Communication among the members was not effective: they kept on discussing the malfunctioning of their program without realizing where the problem lay. Even with the guidance of the instructors, they were not able to identify their mistakes and fix their programs.

m1.1|	Used an IF-ELSE statement instead of an IF statement (Result: forces the computer player to draw a card for every card that is not a match).
m1.2|	Used an IF statement to check if there is a match in the whole set of cards at once, without using a loop.
m1.3|	Missing a command to break the loop after a card is discarded (Result: endless loop)
m1.4|	Missing a command to move the turn to the next player after a card is drawn and discarded.
m1.5|	Missing a command to move the turn to the next player after a card is drawn and it is not a match, so play passes on to the next player.
m1.6|	Did not manage the case when a card is drawn and it is a match.
m1.7|	Extra IF statement to check something that had been already checked (if there is a match, for a second time).
m1.8|	Used a command to move the turn to the next player inside the loop (Result: turn moves to the next player without discarding, even if they have a match).
m1.9|	Used a loop for 7 times (Result: it works only if the player holds exactly 7 cards).
m1.10|	Extra loop for drawing a card (Result: draws a new card for every card that is not a match).

Table 1. Mistakes that were detected in the first activity.

m2.1|	Used a loop for 7 times, which means that it works only if the player holds exactly 7 cards**.**
m2.2|	Extra IF statement to check something that had been already checked (if there is a match, for a second time).
m2.3|	Trying to control who is winning the game.
m2.4|	Used AND instead of OR.
m2.5|	Wrong placement of the counter.

Table 2. Mistakes that were detected in the second activity.

In total, we collected ten unique mistakes in the first activity and five in the second, as shown in tables 1 and 2, and in figures 4 and 5. Of those mistakes, two were repeated in both activities: m1.9 (same as m2.1) and m1.7 (same as m2.2). We can categorize these mistakes in two basic groups regarding the misconceptions that generated them. The first group of mistakes refers to the fact that novices think as humans and attribute human-like reasoning capabilities to machines. The way the computer needs each eventuality to be specified in detail (including, for instance, "obvious" else branches) is something that does not come naturally to many people (Sorva et al., 2013). The second group refers to the fact that students are not yet familiar with the semantics of commands (IF statements, loops etc.), so they didn't completely understand what their result was.

Regarding the pen & paper activity, mistakes m1.1 to m1.6 reside in the first category. For example, the two most common mistakes in the first activity (m1.1, m1.2) were based on the fact that when students entered the code to select the right card to throw, they used an IF statement to check the whole set of cards at once, without using a loop. For them, it sounded obvious that the IF condition should apply to the whole deck. The mistakes m1.7 to m1.10 reflect the misinterpretation of some commands, so they pertain to the second category. For example, five groups used an extra IF statement for something that had been already checked and one group used a command that terminates a loop outside an IF statement, which means that the loop will be terminated after a single round, no matter what. The five mistakes recorded in the Scratch activity all reside in the second category. Furthermore, the mistakes were fewer in the second activity, despite the increased difficulty. But the most important remark is that none of the groups made the same mistake in the first and second activity, which is an indication that the paper block activity has an educational value as a preparation stage in understanding computational concepts.

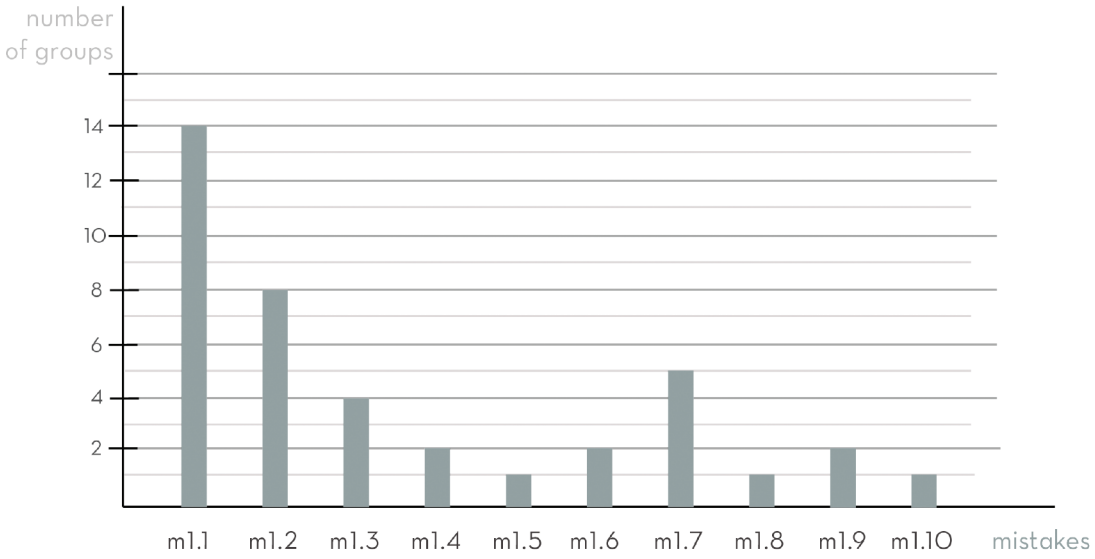

Figure 4. Frequency of mistakes in the "paper blocks" activity.

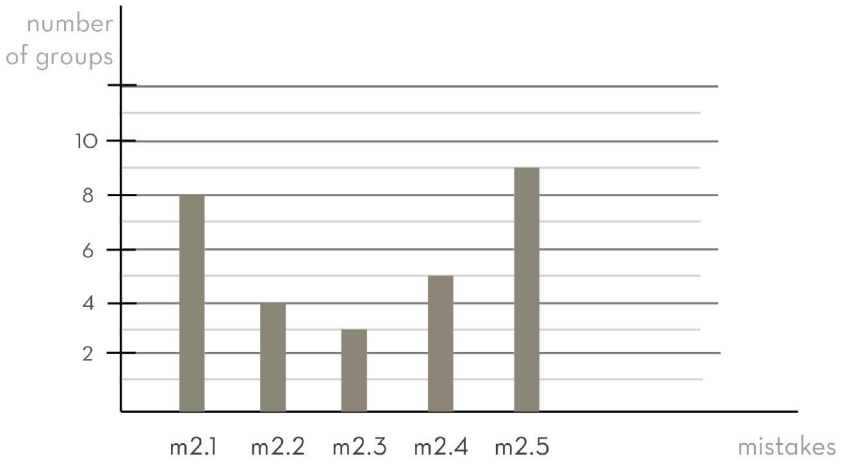

Figure 5. Frequency of mistakes in the Scratch activity.

What makes the activity unplugged is that the instructor "runs" the algorithm as the computer player, while playing with physical cards against the students. The pen & paper activity was an enormous help in the simulation of the analogy of the computer and the human player because the students were able to hold the cards in their hands and reproduce all those actions themselves. During the first activity, while they interacted with the physical cards, the students were constantly being asked to answer aloud to the question "what do you do when you play?". To illustrate this, here is a part of a conversation between the instructor and a student:

Instructor: It's your turn to play. What will you do?
Student: I throw a card (said while throwing a card).
I: But how did you decide which card?
S: I looked at them to see if it is the same color or same number as the one that was thrown before.
I: Did you look at them all at once?
S: Yes, they are all here.
I: Do you think the computer could "look" at them all at once?

Through those questions the instructor was trying to guide the student to understand the fact that it may seem to us that we look all the cards simultaneously, but in fact we don't. We check the cards one by one, but there is no need to give a command to the human brain to do this, because it has been trained to do it subconsciously. By contrast, this needs to be explained to the computer.

Regarding the difficulty of novices to relate the action to the result, in other words to relate program code to the dynamics of program execution in order to learn to reason their actions and practice programming consciously, once again the pen & paper activity proved to be an important asset. The design of the activity did not allow any syntactic mistakes, so we had to deal with logical mistakes only. These became obvious when the instructor did not make the right move, according to the rules of the game. In this way, the students realized that their algorithm was not running as it should; they were then asked to find a way to fix it and were given a second chance to "run" it. If they were having difficulties, the instructor guided them through the solution. By playing with physical cards, students realized their mistakes and above all the consequences of their actions. For example, when they created an endless repetition and the instructor threw five cards instead of one, they saw five cards falling and complained that this broke the rules of the game. And when the instructor drew seven more, one student said while laughing: "*You will probably lose*". The result was that the student remembered this observation when creating the program in Scratch, so he said "*Now we know how to make the computer player lose, if we want to*". By activating their experiential and visual memory on a familiar procedure (that of playing UNO), the students were able to create the connection between the action, in this case the wrong action, and the result, which is a fundamental skill for learning and practicing programming.

The two main characteristics of the interaction among students during the process of problem solving were (1) competition, which was expressed mainly by criticizing the choices of other students in their group and the choices of the other groups, and (2) collaboration between team members. They were impatient to try out their algorithm through playing and they also watched other teams play, so they knew if t others succeeded. Competition played a very important role in keeping the interest and attention of the students. On the other hand, they discussed things with each other before asking the instructors. For instance, in the pen & paper activity, before they began to stick the paper "blocks" on the cardboard sheet, they all needed to agree on the result. In the Scratch activity, some teams decided that one member would handle the computer and the others would guide them, while other teams decided to take turns with the mouse.

6. CONCLUSIONS

In this paper we have presented and evaluated a learning activity for teaching computational thinking through programming. This included both unplugged and visual programming and t centred on the collaborative design of a computer player for a known game. The evaluation results were encouraging, and provided interesting insights about the students' reception of the activities.

The use of a game-related problem solving resulted in an entertaining activity that provided the opportunity to scaffold learning. However, when the difficulty was increased, students were disappointed. Disappoint-

ment can have a negative impact on motivation, but when they overcame the difficult part, the students were excited about what they had achieved. So, it is important that students feel that they can complete the activity, whilst being challenged to provide a good solution.

In addition, the fact that students could instantly view their solution and interact with it, even if they had only made a part of the overall program, had a positive impact. The game gives them the opportunity to expect a result with which they can interact. They made something, which was fun even when the activity was over, which is another important factor for motivation. In particular, UNO proved to be a good choice because of its familiarity; students knew the rules by experience. As a result, we managed to overcome the first part of understanding and analyzing the given problem, and started directly from trying to solve it.

The main conclusions that emerged from the activities are the following:

- By giving students a project based on a familiar game, we overcame the more tedious stage of problem analysis and understanding, and went straight to the "action". That was one of the main motivating factors for students;
- The pen & paper activity prepares students for transforming the algorithm into a computer program, as they have already performed analysis in practice, "recorded" this in their memory and tried out the steps they need to follow in the implementation of the program;
- Students' experiential memory seems to be activated, so they were able to retract their actions and their errors (wrong actions) based on the results they generated, which they remembered experientially and visually.
- working in groups enhances learning, as students not only acquire knowledge from a single source, namely the instructor/educator or even the task assigned to them, but also share knowledge with each other.

We are planning to continue our research in the future by preparing larger and more systematic studies, expanding our approach towards new tools and practices, and applying it in a variety of game-related contexts.

7. REFERENCES

Armoni, M. (2011). The nature of CS in K--12 curricula: the roots of confusion. *ACM Inroads, 2*(4), 19-20. doi:10.1145/2038876.2038883

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology, 13*(1), 20-29. Retrieved from http://www.computingunplugged.org/sites/default/files/papers/Unplugged-JACIT2009submit.pdf

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing, 1*(2), 67-69. Retrieved from http://www.spclab.com/publisher/journals/Vol1No2/N1.pdf

Dann, W. P., Cooper, S., & Pausch, R. (2011). *Learning to Program with Alice (w/CD ROM)*. Upper Saddle River, NJ: Prentice Hall Press.

Faust, J. L., & Paulson, D. R. (1998). Active learning in the college classroom. *Journal on Excellence in College Teaching, 9*(2), 3-24. Retrieved from https://eric.ed.gov/?id=EJ595306

Gibson, B., & Bell, T. (2013, November). Evaluation of games for teaching computer science. In *Proceedings of the 8th Workshop in Primary and Secondary Computing Education* (pp. 51-60). New York, NY: ACM. doi:10.1145/2532748.2532751

Huang, T. (2001). Strategy game programming projects. Journal of Computing Sciences in Colleges, 16(4), 205-213. Retrieved from http://dl.acm.org/citation.cfm?id=378708

Ivanović, M., Budimac, Z., Radovanović, M., & Savić, M. (2015, June). Does the choice of the first programming language influence students' grades?. In *Proceedings of the 16th International Conference on Computer Systems and Technologies* (pp. 305-312). New York, NY: ACM.

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code. org. *Computers in Human Behavior, 52*, 200-210. doi:10.1016/j.chb.2015.05.047

Kölling, M. (2010). The Greenfoot programming environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 14. doi:10.1145/1868358.1868361

Kordaki, M. (2012). Diverse categories of programming learning activities could be performed within Scratch. *Procedia - Social and Behavioral Sciences, 46*, 1162-1166. doi:10.1016/j.sbspro.2012.05.267

Lai, M., Luong, D., & Young, G. (2015, January). A Study of Kinesthetic Learning Activities Effectiveness in Teaching Computer Algorithms Within an Academic Term. In P*roceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)* (p. 44). Retrieved from http://search.proquest.com/openview/f4c2a3fdd27bff8cb819c9f69f32090d/1?pq-origsite=gscholar&cbl=1976352

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin, 41*(1), 260-264. doi:10.1145/1508865.1508959

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior, 41*, 51-61. doi:10.1016/j.chb.2014.09.012

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 16. doi:10.1145/1868358.1868363

Prensky, M. (2005). Engage me or enrage me: What today's learners demand. *EDUCAUSE Review, 40*(5). Retrieved from https://www.learntechlib.org/p/99184/

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education, 97*, 129-141. doi:10.1016/j.compedu.2016.03.003

Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE), 13*(4), 15. doi:10.1145/2490822

Sung, H. Y., & Hwang, G. J. (2013). A collaborative game-based learning approach to improving students' learning performance in science courses. *Computers & Education, 63*, 43-51. doi:10.1016/j.compedu.2012.11.019

Wilson, A., & Moffat, D. (2010). Evaluating scratch to introduce younger schoolchildren to programming. In J. Lawrance, & R. Bellamy (Eds.), *Proceedings of the 22nd annual workshop of the psychology of programming interest group – PPIG2010, September 19–22, 2010* (pp. 64–74). Retrieved from https://

www.researchgate.net/profile/Amanda_Wilson2/publication/266490101_Evaluating_Scratch_to_introduce_younger_schoolchildren_to.pdf

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences, 366*(1881), 3717-3725. doi:10.1098/rsta.2008.0118

Zhang, F., Kaufman, D., & Fraser, S. (2014). Using Video Games in Computer Science Education. *European Scientific Journal, 10*(22), 37-52. Retrieved from http://www.eujournal.org/index.php/esj/article/view/3904/3695