Object Oriented Puzzle Programming

A. Ferrari, A. Poggi\textsuperscript{1}, M. Tomaiuolo\textsuperscript{2}
Dipartimento di Ingegneria dell’Informazione
Università degli Studi di Parma
Viale delle Scienze, 181A 43124 Parma
aferrari@ce.unipr.it
\textsuperscript{1}poggi@ce.unipr.it
\textsuperscript{2}tomaiuol@ce.unipr.it

The objective of this work is to present a methodology and an educational environment to facilitate first approach to coding for high school and university students. Object-oriented paradigm is increasingly used in CS1 courses and Block programming reduces the syntactic complexity of text-based programming languages. So our purpose is merge positive features of both and combine Block and Puzzle programming with object-oriented paradigm in a graphic educational environment.

1. Introduction

The introduction to programming and to Computational Thinking in general is a topic of vivid debate. The main objective of this article is to present our graphic educational environment for introducing the object-oriented paradigm using Block and Puzzle Programming.

The theme of coding is generating much interest and various approaches have been proposed for spreading Computational Thinking. In particular, the experiences of the most recent years with regards to introduction to coding have been mostly oriented to a very young audience. Many projects aim to facilitate the introduction of young students and other novices to computer programming. Those include the famous code.org project, “One hour of code” [Wilson, 2014], the Raspberry Pi platform [Upton & Halfacree, 2014], and numerous other efforts [Cooper et al, 2000]. A feature shared by a number of these projects is the use of Puzzle Programming for simplifying the first approach to computer programming, reducing -- and in some sense eliminating -- the syntactic burden.

Our work is oriented to teaching computer programming in a different context. In fact, for our jobs and experiences, we are interested in particular in programming courses for high school students and university freshmen. In these courses, students face at first the typical syntactic difficulties related with coding, but then they also have to deal with the more advanced challenges of design, using one or more paradigms. The common understanding of the latest years, which is also supported by research works and practical experiences, is to introduce the Object-Oriented approach already in CS1 courses.
In various experiences which are described in the scientific literature, the focus is placed on the advantages derived from the use of a development environment realized specifically for educational purposes.

We are working to define an approach which can get the best of both the worlds of Object-Oriented Programming and Puzzle Programming. This approach, which we call Object-Oriented Puzzle Programming, is supported by both a methodology and a tool for the design and development of object-oriented applications. The application is realized by connecting visual objects representing the fundamental tokens of Object-Oriented Programming, in a similar manner to Puzzle Programming. The user/programmer realizes a puzzle of connected blocks, which represents the structure of the application. From this visual structure, it is then possible to generate the code automatically. The set of target languages include the most popular object-oriented languages in the educational context, e.g., Python, Java, Javascript etc.

The rest of the article is organized in the following way. Section 2 presents some background material, the concept of Computational Thinking, the situation of Computer Science in public education, in particular in Europe and in Italy. Section 3 presents the features of block-based programming and object-first approach in CS1 courses. Section 4 discusses methodology and tool we propose for Object Oriented Puzzle Programming and Google Blockly API that underlies our project. Finally, section 5 provides some concluding remarks.

2. Background: the debate about Computational Thinking and Computer Science education in Europe and in Italy

The whole idea of Computational Thinking was popularized by [Wing, 2006]. The article presents Computational Thinking as “a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science.”

With the development of the Computational Thinking trend, other views emerged. In particular, Denning [Denning, 2009] underlines the necessity of distinguishing Computer Science from Computational Thinking; the latter in fact includes only a part of the larger set of skills and knowledge possessed by a computer scientist. According to Denning, Computer Science is a combination of engineering, mathematics, and science.

This acceptance is facilitated by the shifting of focus from computers to computation, as the main interest of Computer Science. A quite popular saying in the early 1990s (attributed also to Dijkstra, among others) was that “Computer Science is no more about computers than astronomy is about telescopes”. The attribution of the quote is disputed but, certainly, such a shift in perspective is necessary to begin seeing computation in nature.

The situation of Informatics education in Europe is not ideal. The title of a recent study, edited by ACM & IE, is quite evocative: “Informatics education: Europe cannot afford to miss the boat” [Gander et al., 2013]. It starts from the consideration that both governments and citizens in EU are conscious of living
in a so-called “Information Society”. But too often they satisfy themselves with the notion that learning to use digital media (i.e., digital literacy) is enough. Instead, digital literacy has to be considered as a very basic practical skill, and not an adequate intellectual tool for facing the new challenges.

To distinguish the more basic skills from the real science behind IT, the report adopt the term “Informatics”, which is already common in continental Europe, but is acquiring the meaning of “Computer Science” also in the Anglo-Saxon world. Informatics is a scientific discipline that was born before actual computing machines, with the studies of Turing and Church in the 1930s. In the following decades, with its development into a full-fledged discipline, Informatics became an interesting mix of mathematical theory and electrical engineering, together with many seminal concepts of its own, including: algorithms; concepts of performance and complexity; data structures; concurrency, parallelism, and distribution; formal languages; abstractions.

In Italy, the introduction of Informatics in the various programs is only partial. A large reform of secondary schools started in 2010. It followed a contrasted discussion, with some articles in the press opposing the idea that everybody should study programming and the foundations of Computer Science, in the same way that it is not necessary to be a mechanical engineer to drive a car. Computer Science researchers and professionals, on the other hand, supported the idea of a more widespread diffusion of Informatics as a scientific discipline, with dedicated hours at least in secondary schools. A manifesto [CINI, GII, GRIN, 2010] was signed jointly by the largest groups in Informatics research, including CINI (Consorzio Interuniversitario per l’Informatica), GII (Gruppo Ingegneria Informatica), GRIN (Gruppo di Informatica). The manifesto embraces the point of view of a previous and more detailed document published by AICA (Associazione Italiana per l’Informatica ed il Calcolo Automatico), with the title “Informatics in high schools in the context of education reform” (published only in Italian) [AICA, 2003].

After the reform, elements of Computer Science are introduced, since the first year, in both technical schools and applied sciences secondary schools. Other schools (classical, scientific and pedagogical secondary schools) should have elements of Computer Science covered during Mathematics lectures. Most of these initiatives are directed toward primary school and based on ludic activities. In addition, the Italian government is promoting initiatives for soliciting secondary school students to become “digital makers.”

3. Approaches for the introduction to Computational Thinking

Coding is certainly an important aspect of Computational Thinking. Denning sustains the idea of equating Computer Science with coding is misleading [Comer et al., 1989]. However, it is certainly true that programming languages are useful and often irreplaceable tools for achieving the goals of Computer Science. The main aim of Computational Thinking is not to foster and simplify the development of applications. However, the proposals for new environments,
tools and languages, which ease the first approach to programming, can be a starting point. After this point, the other fundamental aspects of Computational Thinking, including abstraction and modeling, can be introduced.

“Every student in every school should have the opportunity to learn computer science” is the explicit aim of code.org, one of the most important projects enacted to spread Computational Thinking abilities (www.code.org).

Our attention is directed in particular to block programming proposed as a learning environment for young novice programmers and object-first methodology adopted in introductory computer science courses.

3.1. Block and Puzzle Programming

Like other projects with similar aims, code.org provides young and novice programmers with very easy development tools and environments. With these systems, the development process is based on the creation of puzzles, which are composed of blocks representing the basic elements of the “language”.

This approach is often applied to gaming situations, and it has been advanced in many prior programming environments, including for example Scratch [Resnick et al, 2009] and Alice [Cooper et al, 2000]. These projects propose educational games for children who have not had prior experience with computer programming. By the end of these games, players, novice programmers, are ready to use conventional text-based languages to solve simple problems.

Most of these initiatives are directed toward primary school and based on ludic activities. The MIT App Inventor project [Wolber et al, 2011] is instead directed at a more advanced target. It is a block-based programming tool enabling the creation of simple multimedia application that can be executed on Android devices.

The common aim, which characterizes all these various projects, is not to turn everyone into a computer programmer, but to spread the basic scientific knowledge for understanding our own contemporary society.

The founding idea of block programming, which characterize all the projects, which we have described, is to provide a graphical interface with blocks of diverse types. The user/programmer can combine the blocks in various ways through drag & drop operations, in such a way to form a puzzle representing the solution to a given problem. The blocks represent the basic elements of the language and they are differentiated by form and color, to easy their identification and usage. The difficulties, faced by novices which approach programming for the first time, are essentially of two different kinds: on the one hand it is necessary to formalize the ideas about the solution in the form of an algorithm, on the other hand it is necessary to adhere to the rigorous syntax of a programming language.

Block programming eliminates the possibility to introduce syntactic errors and allows users to focus entirely on the logics of assigned problems and their solutions. In fact, the composition of blocks is rigidly constrained by existing slots, which represent the syntactic constraints of the language.
Although most educational projects and experiences proposing a block programming approach are oriented to a very young audience, projects directed toward high school students are beginning to appear. In [Weintrop & Wilensky, 2015] some points of strength and weakness are analyzed, as they are perceived by high school students which are allowed to move from a textual programming environment to a block-based one.

Students identify the drag and drop composition interaction and the ease of browsing the language as contributing to the perceived ease-of-use of tools based on blocks-based programming. Being less powerful is indicated on the other end as drawback to blocks-based programming compared to the conventional text-based approach.

3.2. Object first vs. Object later

In the last decades, the object-oriented approach is become one of the most adopted programming paradigm [Kölling, 1999]. It is popular both at the educational and professional level. This fact is largely accepted, but there are still a lot of debates focusing on the time in which OOP is best introduced in CS1 courses.

The debate about the “paradigm shift” is still actual, but in the last years in many introductory university courses there has been a move from the procedural paradigm to the object-oriented one.

A comparative analysis [Vilnet et al, 2007] shows that student who have started their curriculum with an object-first approach obtain better results, when they have to design software for solving complex problems. Other studies [Bennedsen & Carsten, 2008] [Cooper et al, 2003] present experiences in which the object-first approach has effectively led to great improvements. They show that students obtain better overall results and the unsuccess rate is drastically reduced. An additional advantage of the object-first approach is to facilitate the comprehension of the object-oriented paradigm, while the shift from a paradigm to another is quite difficult for those who started studying the discipline with an imperative/procedural approach. In [Kölling, 1999] it is argued that “it is the switch that is difficult, not object-orientation.” If the main difficulty lies in the paradigm-shift, from the procedural paradigm to the object-oriented one, then the objects-early approach has the lowest difficulty level, as can be seen in many CS1 courses.

4. Object Oriented Puzzle Programming

In a CS1 course based on the Object-Oriented paradigm, it is important to focus students' attention on design and modeling.

Our project seeks to extend Block programming features to the Object-Oriented paradigm. The tool we propose is a didactic environment for a design driven development of object oriented applications.
4.1. The development environment

The methodology of Object Oriented Puzzle Programming (OOPP) needs to be supported by a development environment. In fact, we’re developing a prototype tool, allowing a user/programmer to design and create simple object-oriented applications, automatically generating executable code, in various target languages, including Python, Java and JavaScript.

In particular, leveraging the API of Google Blockly (developers.google.com/blockly) we have realized a set of higher-level blocks that can be used for object-oriented programming. This way, a visual development environment can be run in any modern browser, and thus further lowering the access barrier for willing users.

The environment is characterized by three fundamental elements: (i) a toolbox containing the available types of blocks, (ii) a workspace to place blocks and compose them in various ways to form a puzzle, and (iii) a textual container for showing the program, automatically generated by converting each block of the puzzle in the desired target language.

Blocks available in the toolbox have been organized in categories and subcategories to ease their identification. Some blocks are part of the standard categories of Blockly: Logic, Loops, Math, List, Variables. Apart from these blocks, we have created additional blocks for object-oriented programming: Classes, Constructors, Methods, Types. New blocks are shaped to fit together only in ways that are syntactically correct, e.g. methods that return values are shaped and defined according to the types of values they return.

The definition of a new class makes it immediately available in the toolbox, as a new datatype that can be used in the application development. To keep the environment simple and intuitive, as much as possible, only the main features of object-oriented programming have been introduced. At the moment, it is not possible to define inner classes, abstract classes or interfaces, explicitly. However, we certainly plan to support encapsulation, inheritance and polymorphism, which encompass all the main use cases for object-oriented design.

We argue that more advanced features, found for example in the UML specifications, are neither necessary nor desirable in such introductory environment. In fact, we foster its future use in the limited case of the initial approach to object-oriented programming. Quite probably, it will never be fit for the development of complex applications from both the point of view of cyclomatic complexity and object-model complexity. We do not consider these uses cases in the application area of this project, which will remain mainly didactic. More in general, writing large programs in most visual programming environments is cumbersome. In fact, as the very authors of Blockly stated clearly: “Blockly is currently designed for creating relatively small scripts … Please do not attempt to maintain the Linux kernel using Blockly :).”

The environment allows users to analyze the puzzle and the generated code at the same time, also side by side in two panels. This is an important step for learning the syntactic features of the target language, which can later be used
to solve more complex problems. The generated code can be executed in the OOPP environment, directly. For interpreted languages, this is quite simple, exploiting the features of the basic distribution of Blockly. The examples provided with Blockly demonstrate the generation of executable JavaScript, Python and Dart code. Instead, the translation to statically compiled languages (e.g. Java) is more cumbersome.

Fig. 1 – Example of some class definition

4.2. Google Blockly architecture and API

Blockly is an open source project developed by Google and comes with a large number of predefined blocks such as control blocks, mathematical functions and so on. Blockly is not a programming language, it’s a web-based visual programming editor that generate code in a specific programming language such as JavaScript, Python and more.

Blockly is a tool for application developers and there are many educational applications that use Blockly. Developers can integrate a resizable Blockly interface into a webpage that allows users (novice programmers) to write programs by plugging blocks together. Users can choose blocks and connect them to design an algorithmic solution to a problem and then generate the code that represents this solution in a programming language.

Google provides an API which allows developer to build new blocks. Then, it is possible to create new puzzles, based on both the newly defined blocks and the standard ones. Both phases of developments, for creating new types of blocks and new puzzles, may be fully completed in a web environment, possibly using only a web browser without any server. The creation of a set of new blocks corresponds to the creation of a new language, with its new syntax and semantics. The syntax is defined by the blocks structure, color, form, and mutual possible connections. For users/programmers this constitutes a
significant advantage; in fact, it is virtually impossible to create puzzle (i.e., programs) with syntax errors, as blocks without a complementary coupling scheme cannot be joined in any way. Drag & drop operations are used to select and join the available blocks of the program. However, their connection is constrained by the types of their slots. Wrong connections are simply not allowed by the web interface, in the first place. In practice, Blockly provides a sort of textarea where only syntactically correct code can be written, with the clause that the code is substituted with visual blocks.

The semantics of the blocks is then defined by providing for each of them the code which they will be translated to. The code can be written in any programming language. The most easily supported and used are JavaScript, Python, PHP and Dart, as they are more simply interpreted and executed. However, it is also possible to generate code in languages that are usually statically compiled (e.g., Java).

The double representation of the program, as both puzzle and textual code, is very interesting and stimulating in an educational environment. In fact, the novice programmer can compare the two versions and identify their mutual bounds.

![Fig.2 – Puzzle programming & Text-based programming](image)

The object-oriented blocks defined in OOPP have been developed using the APIs of Blockly. A very useful tool for the definition of new blocks is BlockFactory, which is a GUI based tool where the main features of the new block can be selected in a toolbox. During the phase of creation, a preview of the new block is shown, with its visual aspect aside from the corresponding JavaScript/JSON code. This code will later be integrated into the puzzle/applications created by the final users. For each new type of block, it is necessary to define its color (which will ease its visual identification), its name and various other attributes, together with its visual shape (which will define the possible connections with other blocks). It is worth noting that, in the case of OOPP, final users are not required to manage this phase. In fact, we have already created the needed generic blocks and they are provided with the environment, ready to be set-up and used directly in the final users' puzzles.

BlockFactory is itself a final Blockly application. Apart from being a useful tool for the definition of new blocks, it can also be considered as an example of the potentiality of block-programming. After having created the first types of blocks, the corresponding generated code is quite clear. Thus, in our project, we created most of the new types of blocks directly in JavaScript. It is possible
to associate each block with various code fragments, which allow the block to be translated into multiple target languages. This way, users can also compare the basic features of those languages, and the different ways they can represent the same puzzle.

Fig.3 – BlockFactory

5. Conclusions

We proposed, and we are working to develop, a project that merge Puzzle Programming and Object-Oriented paradigm. OOPP is a didactic tool for introducing the Object-Oriented paradigm for high school students and university freshmen using Block and Puzzle Programming.

OOPP is a web application based on Google Blockly APIs. We integrate original blocks by creating new blocks for classes, attributes and methods: the basic tokens for Object-Oriented programming. User/programmer with OOPP can create a puzzle: a collection of connected graphical programming blocks. The puzzle can then be automatically translated in Java, JavaScript or Python code.

We think that our project will be useful in CS1 courses as a starting environment for modeling and designing classes. OOPP is a tool for Object-Oriented paradigm not tied up to a specific language. Automatic generation of code from a puzzle of blocks to diverse Object-Oriented languages is a valid help to compare syntactical differences from a unique project.
References


