
Stamatios Papadakis  
Secondary Education Teacher  
Crete Greece  
+306972764917  
stpapadakis@gmail.com

Michail Kalogiannakis  
Department of Preschool Education, Faculty of Education,  
University of Crete  
Crete Greece  
+30694195634  
mkalogian@edc.uoc.gr

Vasileios Orfanakis  
Secondary Education Teacher  
Crete Greece  
+306932430868  
vorfan@gmail.com

Nicholas Zaranis  
Department of Preschool Education, Faculty of Education,  
University of Crete  
Crete Greece  
+306977135389  
nzaranis@edc.uoc.gr

ABSTRACT
The teaching of programming presents many difficulties since its first introduction in the school units. The learning results are unsatisfying arousing questions about the effectiveness of the method used. As it seems, students usually consider programming as a boring and tiring procedure. The problem could be solved if we make the introductory programming easy and entertaining for the students. This can be achieved by reducing the obstacles for the introduction of programming with the use of special programming environments for the beginners such as Scratch and the App Inventor both of which have been created by MIT. Our research focuses on the similarities and differences between the above mentioned programming environments in our effort to find which of the two is the most appropriate for school units.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education]: Computer Science Education – Computer science education, curriculum, literacy

General Terms
Design, Human Factors, Languages

Keywords
Novice Programming Environments (NPEs), Scratch, App Inventor.

1. INTRODUCTION
In our era of innovations students should have lots of skills, including the ones of creative thinking and critical analysis which are known as student skills of the 21st century. Unfortunately most uses of ICT at schools today do not seem to support these specific learning skills. In many cases ICT only enhance old teaching and learning methods [22] though tasks such as pc programming are considered vital for the 21st century student. Students acquire lots of benefits such as the ability of creative thinking and expression of their beliefs, developing rational and analytical thinking and understanding the way ICT are used in our everyday lives [2].

Taking the above into consideration one would expect the increase in students learning programming. However, this number is shrinking more and more over the past years. For example, in Great Britain students learning programming have reduced one third (1/3) the last five years affecting equally the rate in universities as well [29]. Forte & Guzdial argue that the ‘traditional’ teaching approach of programming is more likely to prevent than attract students [5]. The lack of motive is one of the main reasons why students abandon their programming studies [26].

Thus, some programming environments have been developed lately in order to make the programming procedure more friendly and pleasing to beginner students (NPEs, Novice Programming Environments) [27]. Some popular NPEs, such as Scratch and Alice, have reduced the obstacles for introducing programming successfully [24], offering students the ability to put into practice their creative imagination in tasks that are related to their interests [3]. Young people live lately in a world full of smart mobile devices and are fanatic users of both the devices and their applications [28], [32], with many people interested in creating their own mobile applications [25]. Consequently, NPE should adapt accordingly and aim at a mobile-centred audience since they wish to remain in the center of student and youth attention and interests. MIT in cooperation with Google, provide one of the first NPEs, called App Inventor (AI) [24].

In our research there will be a presentation and comparison of the two NPEs: the popular Scratch and the evolutionary AI. Through the presentation of their weak and strong points we will evaluate their suitability as programming teaching tools. Our research can be a useful guide for whoever is considering adopting them.
(especially as regards the newly-introduced AI) as teaching tools to teach programming.

2. PROBLEMS IN TEACHING PROGRAMMING

Since its introduction, the teaching of programming in school units has faced great difficulties while the insufficient learning results arouse doubts concerning the methods used for this introduction [5]. Students usually regard the learning of programming as a difficult and fatiguing procedure [14]. More specifically, cognitive theorists present students’ weakness to solve, to understand the syntax and the semantics of the programming orders [23]. Basic control structures, such as conditions (if, if-else) and loops (while, for), are difficult to comprehend and to be put into practice by beginner programmers [14]. Programming is considered by students as a mysterious and complex procedure which requires specialized technical training and education [4]. Programming lessons are described as too technical, cut off the real world and deprived of creativity [13]. Forte and Guzdial regard as the main reason of failure or early abandoning of the programming courses the idea of students that the programming courses are not useful or interesting [5]. According to [6] the teaching of programming has to be done in such a way so that the cognitive burden is diminished while at the same time the pedagogical value is maximized. Students’ involvement is usually efficient when the teaching framework is enhanced with subjects of direct interest to the students [8].

Margulieux et al. point out that the problem can be dealt with by modifying the introductory programming into an easy and pleasant experience and there are various ways for this effort to be successful [17]. One way is the decrease of endogenous cognitive burden required by beginners for the learning of programming, with the respective decrease of the information number used to solve a problem [23]. The programming ingredients can be isolated so that the quantity of information is decreased and the learners do not try to learn simultaneously different subjects. To begin with, students could be taught in theory the problem solution initializing the cognitive data, which focus on finding solutions without the students to bother so much with the syntax of incomprehensible orders [21].

Papert supported that the programming languages should have a low floor (so that it’s easy for somebody to begin with) and a high ceiling (so that abilities for more and more complicated tasks are offered with the passing of time) [11]. Additionally, the different languages will have to support various types of projects so that people of different interests and learning styles are motivated [10]. The programming languages such as drag-and-drop replace the syntax code with components; an approach which lessens the cognitive burden the one connected with the syntax of the orders allowing the users to focus on the semantic solution of the problem. Not only that but also, these types of languages are considered easy for the users of all ages, cognitive backgrounds and interests allowing them to experiment with the different component parts by simply connecting parts of a code, just like connecting Lego blocks [21].

The environments that include all the above characteristics are called NPEs - Novice Programming Environments [14]. NPEs such as Scratch, Alice, and the Lego Mindstorms NXT, have accepted wide acknowledgement and publicity over the past years since research has shown that NPEs play an important role in attracting and sustaining new programmers in or out of the school environments [3]. A NPE uses visual features instead of programming orders not revealing the complexity of the programming language syntax making it easier for the beginner programmers to understand and use the basic algorithmic structures [24]. NPEs facilitate software development in an entertaining and ‘not threatening’ way. The hope is that the students will not feel stressed or have low self-confidence when they have to deal with computers. Thus, they will be more willing to delve deeper in programming [19]. Nonetheless, we should not forget that the use of a NPE does not solve the problem of command syntax, since the learner will have to deal with it later on during the learning of a second ‘traditional’ language. Still the learner postpones it until the basic programming principles have been understood [29]. Figure 1 (a, b, c) shows the difference between the syntax of a typical program in the traditional language of programming (Hello World!),(Java) and the 2 NPEs (Scratch and App Inventor).

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}

(a) A typical program in Java

(b) A typical program in Scratch

(c) A typical program in App Inventor

Figure 1. Differences in the structure of the program between Java, Scratch & App Inventor.
3. SCRATCH

Scratch is a NPE, which was created by the Lifelong Kindergarten research group of the MIT Media Lab (http://scratch.mit.edu) and is regarded as the premier environment for introducing children to programming [29]. Maloney and Resnick, members of the development team of Scratch, had stated that the development of Scratch, wanted to lower the ceiling programming level, in order for children to start programming earlier [27]. For the same researchers learning programming is like writing. It is appropriate for children to start with simple forms of expression and gradually learn more sophisticated ways to express themselves over time.

The slogan is imagine, plan, share [21]. With Scratch students converted from consumers to producers of media, creating their own interactive stories, games and redistributing free via the Internet [22].

Scratch encourages and facilitates the development of programs using a mix of multimedia elements such as graphics, audio, video etc., to create a new project [4]. Besides, the name Scratch suggests the idea of mixing - modification and mimics the same techniques of dj’s, who play with their vinyl records to create new sounds [21]. Scratch has been named as the YouTube of interactive media, because daily, ‘Scratchers’ from around the world, upload up to 1,000 new projects on the website of the project, the source code freely available for sharing and modification. Within three years the number of projects came from the Scratch website increased from 70,000 to more than 1.8 million [3]. By mid- 2012 it had been translated into more than 50 languages, and each year, worldwide, a day Scratch (day.scratch.mit.edu) is organized.

For the needs of Scratch programming it provides ready characters, scenes and multimedia elements. Additional information can be created by the user with the help of built-in painting, while it may also use external sources [4]. In Scratch each object (sprite) can have one or more scenarios (scripts) connected to it. The scripts add properties to objects, allowing them to act in whatever way the user wishes within one’s work. The scenarios are generated by joining blocks which are organized by category, such as control, traffic etc. A minimum use of the keyboard is required by the user, as the blocks move in the scene using the mouse and are joined together like Lego bricks. Blocks are connected only when there is reasonable editorial, completely eliminating syntax errors, greatly simplifying application development, however, using the same repertoire of commands and structures found in other programming languages [18]. For example, Scratch supports the use of variables, selection and repetition structures and conditional object-oriented and parallel programming [4].

Scratch presents some weaknesses since as pointed out by Harvey et al. it does not have procedures and therefore cannot apply the concept of recursion, one of the central ideas of Logo. Also the support for the data structures is weak [11]. However, these weaknesses are ‘intentional’ as the creators of Scratch deliberately refrained uploading the language features which would act as a deterrent for novice programmers and especially for young children. Also, Scratch does not support classes and inheritance. The University of Berkeley also has developed an extension called BYOB (Build Your Own Blocks), which addresses many of the shortcomings of Scratch. Figure 2 shows the interface of Scratch.

Almost seven years since the first presentation, MIT proceeded, after a long period of testing, to a new version called Scratch 2.0 (http://scratch.mit.edu/overview/). The new version brings several improvements, one of which is the transition from compulsory local use to a browser use (browser-based). Plus, there is only one version for maintenance and growth, while all projects are stored directly in the cloud. With Scratch 2.0 the user acquires new functionality. The integrated paint program acquires the ability to manage vector graphics, there is integrated interaction with web camera, and now the user can create their own tiles prompt, giving additional properties to objects. The sharing of projects has also improved by acquiring social features, like the ability to monitor the progress of projects of other users. However, it is still available for adoption and use locally on the user’s PC (http://scratch.mit.edu/scratch2download/).

4. APP INVENTOR

The App Inventor (AI) is a free web development environment with tiles in order to create applications for smart mobile devices with operating system Android [14], [30]. AI was announced as a pilot project of Google Labs in late 2009, and continued to develop until the end of 2011. The head of the development team was Professor of MIT Harold Abelson [1]. In early 2012 he transferred the work at the Center for Mobile Learning of MIT (Mobile Learning Center) for public use as open source software. Within 18 months after the ‘adoption’ of the MIT, the AI has attracted more than 2 million registered users, 40,000 of whom are active on a weekly basis.

Before the advent of AI, creating an Android application was a difficult and specialized process, because of the many requirements, such as good knowledge of Java and the need of the user familiarity with professional software development tools (e.g. Eclipse, Android SDK, etc.). The AI instead adopts the successful example of the use of visual programming with tiles (e.g. Scratch), adjusting the programming smart mobile devices (e.g. smartphones) [8]. The New York Times has called AI as ‘Do-it-yourself App Creation Software’ [30]. The development environment of AI supports all three popular operating systems.

Figure 3 (a, b) shows the difference in creating a portable audio player in Java and AI.
Combining the increased incentive for users to create portable applications with the advantages of using an NPE, AI could be used as an input programming environment for teaching basic control structures [25], [12]. Liu et al. indicate that, for four years, AI is used in primary and secondary education, while several universities in the U.S. adjust their curriculum by introducing AI for teaching programming (http://mobile-csp.org/) [15]. As Loukides notes, the purpose of AI is to allow people who normally would not ever plan to create applications they would wish, create them without being necessarily programmers [16]. The overall aim of developing AI is to allow users of devices with Android installed software to evolve from consumers to creators [2].

As described in the program’s website [18], AI consists of two main parts which allow users to build their applications:

- **Designer**: this is a website in which the user selects the components to implement and adjust the properties of each component.
- **Author (Blocks Editor)**: this is a feasible window in java, in which the user puts the pieces of code (tiles) in order to edit their 'behavior'.

The AI provides near real-time to adapt to user actions, so the user can directly observe the modification of the application to the Android device. Alternatively, there is the possibility to use the integrated simulator. This is a complete virtual appliance, with the only negative point that is relatively slow compared to a portable device. The final product may be packed in .apk (Android application package) format, or distributed in its web store Google. The structure of AI and the emulator are illustrated in Figure 4 (a, b).

As described in the program’s website [18], AI consists of two main parts which allow users to build their applications:

- **Designer**: this is a website in which the user selects the components to implement and adjust the properties of each component.
- **Author (Blocks Editor)**: this is a feasible window in java, in which the user puts the pieces of code (tiles) in order to edit their 'behavior'.

The AI provides near real-time to adapt to user actions, so the user can directly observe the modification of the application to the Android device. Alternatively, there is the possibility to use the integrated simulator. This is a complete virtual appliance, with the only negative point that is relatively slow compared to a portable device. The final product may be packed in .apk (Android application package) format, or distributed in its web store Google. The structure of AI and the emulator are illustrated in Figure 4 (a, b).

**Figure 3. The difference to create an application in Java and Eclipse compared with App Inventor.**

**Figure 4. The structure of App Inventor and the emulator**

Although the last two years of development many features of Android are constantly incorporated in AI (e.g. using GPS), the AI still has several limitations. For example the inability to create full object-oriented programs, the inability to access sources on the internet without the use of an external API (e.g. Facebook, YouTube, Amazon), and the inability to create files .apk, when the program size exceeds 5 MB [15]. The MIT team is working on creating a new version of AI, which is expected to be released in late 2013 (ai2.appinventor.mit.edu). The biggest difference between the two versions is that the App Inventor version 2 (AI2) will run entirely on network, while incorporating new capabilities and features.

5. **SIMILARITIES - DIFFERENCES**

**SCRATCH & AI**

After the presentation of the two NPE, one would conclude that the AI could be considered as the Scratch on portable devices [31]. Estimates confirmed by Resnick which stated in July 2010 that ‘the members of the Scratch development team share ideas with members of the development team of AI [20]. The two projects have similar goals, although the AI targets smart mobile devices. Some of the discussions have influenced us on what features of Scratch, in particular how the Scratch organizes, accesses and manages the data on the Internet’. In response to a Web user forum Harold Abelson stated that: ‘The roots between AI and Scratch is long and deep, stemming from a Common Vision for the philosophy of education that takes us back to Seymour Papert and Logo, in the late 1960s’ [2]. In another question in the same forum on code sharing, said: ‘There is in fact
a shared code between the two applications. The AI makes use of ‘OpenBlocks’, a project at MIT which uses equally and Scratch’ [29].

Gross and Powers refer that a variety of techniques have been employed to assess novice programming environments. The same researchers classify these techniques into three main categories: anecdotal, analytical and empirical [9]. The first technical assessment, is based on the researcher’s personal view on the changes brought about by the use of NPE in the educational process. The second technical assessment examines a NPE using a specific set of criteria. The third technique uses measurable data in order someone to draw conclusions about the effectiveness of a NPE. In this paper, we used mainly the first technique and some aspects of the third technique such as our direct observation throughout of the student’s engagement with the NPE, the deliverables of students, the attitudes of the students etc.

Taking into consideration international researches [27], [24], [8], [19], [15], and the assessment techniques we referred earlier, we argue that the two NPEs present the following resemblances:

- In AI, but also in Scratch, students move command tiles like connecting puzzle pieces in order to create their programs.
- AI shares the appearance and the look and feel of Scratch, from which it takes a notable number of users. So the transition of a student from Scratch to AI happens easily and without any special problems.
- Programming with AI and Scratch helps the beginner students to focus more on the problem solution and less on the language syntax.
- Both NPEs involve users, allowing them to write programs for situations that are linked directly with their interests, in contrast to the conventional programming teaching where the examples are usually the type of assignments of creating lists of first numbers.

On the other hand, they also have differences which are:

- The initial setup for Scratch is very simple. Installing it is just a matter of downloading the setup file from the Scratch website. On the contrary the setup of AI is not so easy because it needs Java. So one needs to verify that Java is installed on the computer. After Java has been installed, installing AI is quite easy.
- In AI user interface consists of three discernible parts (designing, editor, emulator or mobile device). When the user creates an application one has to use these three parts, whereas in Scratch the user needs to have just a screen in front of him.
- One of the biggest advantages of Scratch is the popular project collection and source code that it offers. AI does not offer such support officially for a similar collection. Just recently the project team of AI has started experimenting to create a similar collection (beta level).
- Scratch does not have procedures (in Scratch 2.0 procedures with parameters are supported). In AI the user can set one’s own procedures with variables as parameters incorporating simple data structures (lists), too.
- AI provides a more integrated environment for the development of object oriented programs, with discernible frameworks between the structure designing, event driven programs. This model makes it easy for the future transition of the user in more elaborate programming languages such as Python or Java through the characteristic Java Bridge.
- AI could offer stronger incentives to the students to get motivated about programming, in contrast to Scratch, since the mobile gadgets (e.g. smartphones) are more ‘tangible’ objects and more linked to the students’ interests as regards the imaginary world of Scratch.
- AI offers abundant learning experiences compared to Scratch, since with AI a student can create whatever one can create with Scratch, but also applications that include physical interactions through the use of drag and drop.
- The execution of orders in AI is not visible, compared to Scratch, where for example the code block to be executed have a white outline to indicate that they are in function.
- AI should incorporate the feature of the downloading of one individual command block from an application and its incorporation in one other, which is similar to insertion and exportation sprites in Scratch.
- The proceedings required for the system preparation for the execution of AI are surely more difficult than the ones in Scratch.
- Scratch provides full support of several languages while AI exists only in English.

6. CONCLUSION-DISCUSSIONS

In conclusion, there is no unambiguous winner. We consider that each framework or educating environment is the one defining the use of Scratch or AI. Both NPEs have their strong but also their weak points; Scratch possibly being the most appropriate for teaching young students or being used in teaching curriculums where the main aim is the pleasant-surface-know-how contact with Scratch and not the deep knowledge about programming. On the other hand, AI is possibly more appropriate for a more official introduction to programming where the final aim is the strengthening of the programming ability and the transition into a conventional language [24].

Thus, for the teaching of programming it’s better to begin with Scratch in Secondary School or the early Junior Classes and then move on with AI in the next classes and school levels. This way the transition between the two environments becomes fairly smoothly due to the look and feel that they convey. AI in comparison with Scratch seems to emerge powerfully as an in-use-programming environment in General High School and in Vocational Schools as it is considered an ideal tool for intriguing the beginner students’ interests with mobile phones [25]. Wolber considers AI as the right tool in the right time [3]. Although Scratch being widely used has proved the merit and the usability of NPEs, AI comes to satisfy the users urge for mobile gadgets. We hope this research will be a useful guide for the teachers who plan introductory lessons and activities on programming, attracting more students in the programming world.

7. REFERENCES

mScience: Sensing, computing and dissemination. Trieste, Italy: ICTP—The Abdus Salam International Centre for Theoretical Physics.

Proceedings of the 42nd ACM technical symposium on 
Computer science education (SIGCSE ’11). ACM, New York, NY, USA, 601-606. DOI: 
http://doi.acm.org/10.1145/1953163.1953329

http://dx.doi.org/10.4236/ce.2013.47A1001