
Using Scratch and App Inventor for teaching introductory programming in secondary education. A case study

Stamatios Papadakis*

Krousonas High School,
Heraklion, Crete, Greece
Email: stpapadakis@gmail.com

*Corresponding author

Michail Kalogiannakis and Nicholas Zaranis

Department of Preschool Education,
Faculty of Education,
University of Crete,
Crete, Greece
Email: mkalogian@edc.uoc.gr
Email: nzaranis@edc.uoc.gr

Vasileios Orfanakis

1st Vocational School of Agios Nikolaos,
Agios Nikolaos, Crete, Greece
Email: vorfan@gmail.com

Abstract: Paradoxically, as the role and significance of computing have increased in society and the economy, and coding is recognised as the fourth literacy, the number of students attending a programming course is in decline. In an attempt to increase interest in computer science (CS), there has been made much effort in developing tools and activities as preliminary learning materials in schools and universities. App Inventor and Scratch strive to engage the novice users by allowing them to write programs about things that connect with their interests in contrast to more conventional programming. In this paper, we focus on the use of these two block-based programming environments as tools to facilitate learning programming for novices. In our analysis, both novice programming environments (NPEs) seemed to be attractive platforms for introducing fundamental concepts in computer programming and both look appealing for majors and non-majors as well.

Keywords: novice programming environments; NPEs; Scratch; App Inventor for Android; AIA.

Reference to this paper should be made as follows: Papadakis, St., Kalogiannakis, M., Zaranis, N. and Orfanakis, V. (2016) 'Using Scratch and App Inventor for teaching introductory programming in secondary education. A case study', *Int. J. Technology Enhanced Learning*, Vol. 8, Nos. 3/4, pp.217–233.

Biographical notes: Stamatios Papadakis is a graduate of the Economics and Business (AUEB) University, Athens, Greece, Department of Information. He received his MSc in Education from the University of the Aegean, Greece, and PhD from the University of Crete, School of Education. He has been working for a series of years as an ICT teacher in public sector secondary education. He has published many articles in journals and has presented several papers in conferences. His research interests include ICT in education, mobile learning, novice programming environments and teaching of programming in primary and secondary education.

Michail Kalogiannakis is Assistant Professor in the Department of Preschool Education at the University of Crete and Associate Tutor at School of Humanities at the Hellenic Open University. He has graduated from the Physics Department of the University of Crete and continued his post-graduate studies at the University Paris 7-Denis Diderot (D.E.A. in Didactic of Physics), University Paris 5-René Descartes-Sorbonne (D.E.A. in Science Education) and received his Ph.D degree at the University Paris 5-René Descartes-Sorbonne (PhD in Science Education). His research interests include science education in the early childhood, science teaching and learning, e-learning, the use of ICT in education, distant and adult education. He has published many articles in international conferences and journals and has served on the program committees of numerous international conferences.

Nicholas Zaranis graduated from the Department of Electrical and Computer Engineering of the NTUA. He took his MSc in ICT from the University of Essex. Also, he received his Master of Science and his PhD from the University of Athens. He has worked for a series of years in the private sector as Electrical Engineer in Hewlett Packard Hellas and in public sector as teacher in Secondary Education. He is currently Associate Professor in the Department of Preschool Education at the University of Crete. His research interests include ICT in education, educational software, and teaching of mathematics using ICT.

Vasileios Orfanakis is an ICT Teacher. He has 15 years of experience in public secondary schools, teaching programming, web design, robotics etc. During the last 4 years he is the technical supporter to more than 100 educational units in the Regional Unit of Lasithi, in Crete Greece. He is a graduate of the Computer Science Department in University of Crete and now he is postgraduate student in Informatics and Multimedia in the Technological Educational Institute of Crete. He has published many articles in journals and papers in conferences. His interests are novice programming environments, mobile computing, robotics and the use of various CMS and LMS.

1 Introduction

In July 2014, European Commissioner Androulla Vassiliou wrote a letter to EU Education Ministers urging them to promote children's opportunity to develop basic computer coding skills in school. In this letter, jointly signed by Neelie Kroes, Vice-President for the Digital Agenda, Vassiliou states, "coding is the literacy of today. Each and every interaction with computers is governed by a code. Programming is fundamental to the understanding of a hyper-connected world" (Barshay, 2014). In the same letter, she states, "coding skills are part of the solution to youth unemployment and

create a growing skills gap in the ICT sector, in which we expect to see a shortage of 900.000 ICT practitioners by the end of 2020” (European Commission, 2014). The above statistics worry, along with predictions from the Bureau of Labor Statistics in the USA that until 2018, computing will be one of the fastest growing job markets (Grover and Pea, 2013). In the USA, Jeannette Wing, corporate vice president at Microsoft Research states that everyone should learn computational thinking (Shein, 2014). According to Wing (2006), “computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science”. Computer science (CS) develops students’ computational and critical thinking skills, thus enabling them, becoming active creators, rather than being passive consumers. This fundamental knowledge is demanded to educate pupils for the 21st century, irrespective of their ultimate field of work or occupation (Code.org, 2015).

Despite the fact that the role and significance of computing has increased in society and the economy (Wilson et al., 2010), students’ motivation to enrol in computing fields is declined (Karakus et al., 2012). Nowadays, it is a challenge for instructors to find new methodologies to attract students based on their interest in computer-related fields and industry needs (Shih et al., 2014). The USA and other countries declared that increasing the number of students who choose to study in scientific fields is a national priority (Harvey and Mönig, 2010). In the UK, the government designated 2014 as the Year of Code, and in September, the new syllabus makes computing education mandatory for all students.

Therefore, all students need to be motivated in order to learn the basic CS principles. Student participation is often successful when the setting is directly related with current themes that are of direct interest to students. Much of what we teach in introductory computing would be right at home in the 1980s PC world, where computer learning was taking place on a computer display screen, excluded from everyday life. The experience of computing today is fundamentally different (Gray et al., 2012). In an attempt to increase interest in CS, much effort has been made developing tools, activities and preliminary learning materials, both for younger children and for high schools and universities (Meerbaum-Salant et al., 2010). Research has shown that novice programming environments (NPEs) play an important role in attracting and retaining new programmers’ interest in computing as they manage to engage students and improve their attitudes towards CS (Papadakis et al., 2014). Additionally, today’s students are learning by using their tablets and smart phones, so mobile services and applications should be extremely engaging for this generation of students (Shih et al., 2014). A new entrant in the field of NPEs is App Inventor for Android (AIA). Unlike other NPEs, the target device for AIA is not a computer, but any Android-based smartphone/tablet (Roy et al., 2012).

2 Problems in teaching programming

Students, comparing CS with other academic fields, find it significantly harder. It is thought to be a ‘nerdy’ and ‘not a cool course to take’ (Kurkovsky, 2007). Program learning is viewed as a hard project (Malan and Leitner, 2007). Significant evidence shows that many novice programmers experience difficulties with learning the programming concepts and applying those concepts in code writing. Novice

programmers find it hard to understand the syntactic details of a language and the computational concepts (Roy et al., 2012). In addition, they face problems with the basic algorithmic structures like looping, conditions, etc. and creating programs with them (Guzdial, 2004).

Moreover, for decades, programming was introduced using enterprise level languages like C, C++, and Java, both in universities and high schools. Despite the popularity of those languages, there has been much debate about the suitability of these languages in education, especially in introductory courses (Pears et al., 2007). The cognitive load required for learning a production/enterprise language is huge (Papadakis et al., 2014). Even the simplest of Java programs which are used on an introductory course's first day, due to their object-oriented nature, look hieroglyphic to a novice programmer (Malan and Leitner, 2007; Wolber, 2011):

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

Languages like Java challenge students to master programmatic overhead before programming itself: students must become masters of syntax before solvers of problems (Malan and Leitner, 2007). The targets of an introductory programming language should be problem solving and algorithm development (Pears et al., 2007). Additionally, researchers assert that traditional programming courses fail to connect programming and CS concepts with students' diverse interests and backgrounds (Forte and Guzdial, 2004). Guzdial and Soloway (2002) indicate that one reason is that teachers use an outdated way of teaching programming as "we teach computer science in much the same way as we learned". Looking the Java initial program, it is hard to be understood by students. In addition, it is not motivating for students to get it run and get a 'Hello World' message on the screen (Wolber, 2011). Today's students, who were born and grown up with interactive media and more lately with the smart mobile devices, do not find programs based on examples of the form 'Hello, World' motivating. In addition, students have little or no interest in learning how to program with common and outdated assignments such as calculating or creating a grade book (Rizvi et al., 2011).

There is a long, rich history of attempts and strategies to ease the process of learning to program and make programming engaging and accessible to a broader population (Kelleher and Pausch, 2007). Some of these strategies include:

- a applying pair-programming and collaborative learning
- b using themes that are attractive to students as multimedia, game and/or animation approach
- c using visualised programming to introduce core concepts before more advanced and in-depth courses (Dekhan et al., 2013).

As a result, many visualised programming tools have been developed and especially blocks languages, in which programs are constructed by connecting blocks that resemble puzzle pieces. These tools have become quite popular in recent years (Chadha, 2014) and are increasingly used to introduce novices to programming (Turbak et al., 2014b).

Although Logo Blocks was the first block language that was developed in 1995, block programming really took off in 2007 when Scratch was appeared (Vasek, 2012). There are many advantages of ‘blocks’ languages. They eliminate the need for typing complete programming statements. Blocks are selected from drawers – menus of related blocks (e.g., math blocks, control blocks) – so it is not necessary for novices to remember or type their names (Turbak et al., 2014a). Additionally, in many blocks languages, blocks belonging to different collections are often distinguished by different colours so users can easily recognise them. Block shapes guarantee that blocks can be connected only in syntactically meaningful ways (Papadakis et al., 2014; Turbak et al., 2014b). This is the most important dimension for blocks languages, whose primary purpose is to reduce syntactic errors in text-based programming. The blocks also contain other useful visual information that is not necessarily readily available in textual programming languages (Chadha, 2014). With these programming environments, novices can focus on learning the concepts, thinking, and problem-solving skills associated with CS principles rather than being hindered by the frustrations of syntax errors that differ in each language (Papadakis et al., 2014; Chadha, 2014). Although blocks eliminate many syntactic errors, runtime errors are still possible, such as type errors and index-out-of-bounds errors (Okerlund, 2014). Several of these introductory computational environments use the three-stage ‘use-modify-create’ progression to help the learner evolve from user to modifier to creator of computational artefacts. Visual programming environments are often followed by exposure to high-level programming languages such as Python and Java (Grover and Pea, 2013).

3 Scratch

Scratch (<http://scratch.mit.edu>) was initially developed in the Lifelong Kindergarten Group at the Media Lab of MIT. Scratch builds on the constructionist ideas of Logo and Etoys (Maloney et al., 2010). Scratch as stated on its website is designed to help young people (ages 8 and up) develop 21st century learning skills (Rizvi et al., 2011). With Scratch, someone develops his ‘fluency skills’ namely his ability to ‘think creatively, reason systematically, and work collaboratively’. Moreover, young people working with Scratch have the opportunity to face significant mathematical and computational issues, like conditions, parallelism, variables and other (Resnick, 2012).

While Scratch inherits many key ideas from Logo, it is also different in several ways. First, in addition to a low floor and a high ceiling, Scratch has wide walls, allowing users to create many different types of projects from interactive greeting cards to physics simulations. Resnick calls these creations ‘projects’ instead of ‘computer programs’ (Barshay, 2014). Sharing is a fundamental part of the Scratch experience and the website provides a place where young creators share their creations with one another on the web. As a result, Scratch has been called the ‘YouTube of interactive media’ (Resnick, 2009). The Scratch Online Community was beta released on March 4, 2007 and started with only 20 participants who were involved in a Scratch workshop (Monroy-Hernandez and Resnick, 2008). Now there is a vibrant online community, where, since December 2015 9.053.352 users had already shared more than 12.030.886 projects and 62.489.843 posted comments (<http://scratch.mit.edu/statistics/>). There is also a separate online community, called ScratchEd (<http://scratched.media.mit.edu>), specifically for educators who help

others learn Scratch (Resnick, 2012). Scratch is available free of charge and has been translated into more than 50 languages (Resnick, 2013). In addition, Scratch software is often redistributed by school systems and educational organisations as One Laptop per Child (OLPC) (Maloney et al., 2010).

Scratch is a blocks language for programming animations, games, and stories. The blocks control the actions of sprites on a canvas. In Scratch, users write programs to manipulate sprites; the default sprite is the Scratch cat. These blocks all have notches, which control flow through statements. Each type of value is represented through a different shape. Scratch has three different types of variables (Boolean, number and text) (Vasek, 2012). Concepts that are often difficult for novices are easier to understand in Scratch because less is hidden. For example, variables can be made visible, helping the user to understand immediately the effect of operations such as clearing or incrementing that variable (Wolz et al., 2009). Scratch supports parallelism across objects (Brennan and Resnick, 2012) and completely hides the compilation process from the user. Scratch 1.4 has 125 command blocks, although some of them do not appear until needed (Maloney et al., 2010). The Scratch language is designed in order to prevent the generation of runtime errors altogether. Single step mode is a feature of Scratch which can be used from users in order to step through their code and check for behavioural and logical errors. Tracking various blocks when they are called is also possible by choosing to have the code outlined (Okerlund, 2014).

Scratch has weaknesses as a programming language. These weaknesses are not oversights; the designers of Scratch deliberately avoided overloading the language with anything a child might find threatening (Harvey and Mönig, 2010). These restrictions are a calculated part of the Scratch design, which bears a primary goal that every facet of the language should be intuitive even to new children. Most notably, it lacks procedures, which convey the impressive phenomenon of recursion, one of the central ideas of CS. Its support for data structures is also weak. Until version 1.3, Scratch had no data aggregation mechanism at all; the lists added in 1.3. Deep structures (lists of lists) are ruled out (Gray et al., 2012). Scratch sprites are objects that own state and behaviour, but they can only be copied. Scratch has no classes or inheritance. A ‘broadcast’ primitive provides a rudimentary form of message passing, but with the limitations that messages cannot be directed to an individual sprite, and the corresponding method scripts cannot take arguments or return values (Brennan and Resnick, 2012). These weaknesses reflect differing goals: Scratch makes no effort to teach object-oriented programming (OOP). In addition, Scratch has no relation at all with Java (Papadakis and Orfanakis, 2014). This means that Scratch is generally not desirable for a CS1 course, although it is sometimes applied as a warm-up in CS1 before moving on to C, Python, or Java (Utting et al., 2010).

Due to Scratch weaknesses, Brian Harvey and Jens Mönig, presented at Constructionism 2010 an extension of Scratch named Build Your Own Blocks (BYOB). BYOB, recently renamed to Snap!, targets the Scratch audience and extends Scratch by adding procedures and higher order functions, effectively adding the power of abstraction to Scratch (Harvey, 2012). Users can create any type of block (expression or statement blocks) and place this block in any category (Vasek, 2012).

A new generation of Scratch, called Scratch 2.0, was released in May 2013 that enables people to program Scratch projects directly in the web browser. This new version enables Scratchers to:

- a share at multiple levels of granularity
- b store ‘persistent data’ in the cloud
- c create projects that react in the physical world by using the webcam as a sensor
- d import sets of specialised programming blocks for continually adding new capabilities to the core language
- e export projects from Scratch to other social-media sites (Resnick, 2012).

With Scratch 2.0, the process of creating a Scratch project will be analogous to creating a collaborative document using Google Doc (Resnick, 2010). There is also a downloadable version of Scratch 2 so anyone can create and view projects offline. Scratch 1.4 versions are also still available, so someone can have both Scratch 1.4 and 2 on his computer. ScratchJr, a mobile version of Scratch, aimed at five to seven year olds was released to the public on July 30, 2014. This app allows children to snap together graphical blocks to make the characters move, jump, dance and sing (Barshay, 2014).

4 App Inventor for Android

AIA (<http://appinventor.mit.edu>) is a visual programming environment for creating mobile applications for Android that uses a Scratch-like graphical blocks of code. It sets a low floor for allowing creative app building while still engaging with complex computational concepts, including procedural and data abstraction, conditional and logical thinking, and debugging (Grover and Pea, 2013; Spertus et al., 2010). In the rhetoric question why one wants to use AIA, as a complementary environment to the other NPEs, Roy (2012) answers that the main reason is because of the increasing usage of smart mobile devices among teens, females and minorities and their potential use as a key for introducing computing. Additionally, NPEs that look and feel like Scratch and Alice but can develop apps for Android and iOS are not common (Grover and Pea, 2013; Roy, 2012).

Google developed AIA in the context of Google Labs. It was firstly announced in July 2009 as an experimental teaching and learning tool for a select circle of American Institutes of Higher Education (Kloss, 2012). Almost exactly one year later, AIA was announced to the public and Google made it available to interested developers as a closed beta version. After the termination of Google Labs, it became open source on December 2011 and the MIT Center for Mobile Learning continued the project under the name ‘MIT App Inventor’ (Schinkel et al., 2013). Since then, AIA environment has become more flexible and friendly by the MIT team by adding new characteristics. During the year 2011, AIA became established as a genuine option to the Java-based Android Software Development Kit (SDK) (Kloss, 2012). The AIA development environment is supported for the three known operating systems, and the resulting apps can be installed on any smart mobile device running Android version 1.5 or greater as an operating system (Pokress and Veiga, 2013).

The advantages of using the AIA are primarily its use as an easy introduction to programming, because no extra coding knowledge is needed as well as quickly and easily demonstrable results (Papadakis et al., 2014; Schinkel et al., 2013). AIA allowing students to focus on the logic for programming rather than the strict syntax of a textual

programming language (Chadha, 2014). AIA comes bundled with a based Android smartphone emulator for live testing. The emulator is quite capable and is a good option if the budget is limited (Papadakis et al., 2014; Roy, 2012), so anyone can develop and test apps even if he does not have a device available. This is important for education and settings in which not every pupil has access to a phone (Wolber, 2011) especially at the K-12 level (Okerlund and Turbak, 2013). AIA requires an active internet connection as well as a Gmail account. Alternatively, there are stand-alone, offline AIA versions, which can be installed locally in Windows, Mac and Linux operating systems such as the AI2 Live Complete and AI2 Ultimate.

AIA has gone through two major iterations. App Inventor Classic (AIA1) refers to an early version and App Inventor 2 (AIA2) is now the primary version as it was released in the end of 2013. The overarching design of both iterations is the same. The environment consists of two parts, the designer for selecting the components of the app, and the blocks editor for setting the behaviour of the app (Papadakis et al., 2014; Chadha, 2014; Pokress and Veiga, 2013). Components include visible user interface items such as text boxes and non-visible items such as the Bluetooth client. AIA2 has a couple of new features such as the ability of adding additional screens while in blocks or designer mode and generally simplifies the creation of blocks programs and changes the way in which variables are declared and used (Okerlund and Turbak, 2013). Additionally, AIA2 as an improved version is much easier to setup. The blocks editor runs in the browser as a different mode in the project. Previously, a Java file called the 'blocks editor' had also to be set up and run, along with installing the AI Setup Software, in order to run AIA1 (Shih et al., 2014). Since December 2015 AIA1 and AIA2 weekly active users where 140,000. Totally, there are 4,000,000 registered users from 195 countries which have already built 12,000,000 applications (<http://appinventor.mit.edu/explore/>).

One of the pedagogical advantages of working with AIA is that students can see their creations while they are building them. This allows students to develop their apps (projects) gradually and encourages them to test as they build. A completed app (project), can be directly downloaded to the connected device, or exported in apk format (Android Application Package) for distribution or upload to Google Play (Pokress and Veiga, 2013). Alternatively, a student can download the whole project in .zip (AIA1) or .aia (AIA2) format for distribution, in order other students to be able to see and edit the source code (Papadakis et al., 2014). The MIT AIA server can store all the projects, allowing students to log into their account and continue their work anytime and anywhere (Shih et al., 2014). AIA provides databases, geolocation, intents to other apps, audio and video contents and other features (Schinkel et al., 2013). The concepts of lists and lists of lists are similar to what other computer languages call arrays and multi-dimensional arrays, respectively (Soares, 2014). The TinyDB and TinyWebDB components allow for persistent storage using simple key-value pairs. Direct data storage on the device is achieved by TinyDB, while access to web information sources and shared databases, is done by TinyWebDB. The TinyWebDB component is significant because as students feel more empowered, creative, they often want to make applications that access web information (Wolber, 2011). AIA has also support for Google Fusion Tables so students can create tables, which can modify with SQL-queries (Schinkel et al., 2013).

AIA is not perfect. It is still limited compared to a normal programming environment. It does not allow creation of new components (new classes) and does not allow control over the priorities of events (Schinkel et al., 2013; Soares, 2014). In addition, although it is easy to create simple programs, it is tiring and time consuming when it comes to ones

that are more complex. In large projects with a lot of procedures and variables, it is hard for a student to maintain them, because there are no hierarchies, except the option to minimise the procedures (Schinkel et al., 2013). Additionally, AIA does not currently offer any searching features so someone must shuffle through the blocks of the entire program to find a piece of code. AIA does not currently offer a means of sharing projects or replicating/reusing parts of code across projects such as Scratch. This is extremely inefficient, and inhibits students from reusing block code in multiple projects and sharing blocks code with others (Chadha, 2014). Only recently, the MIT Center for Mobile Learning has made the App Inventor Community Gallery available, which is an open-source repository of apps created with App Inventor (<http://classic-gallery.appinventor.mit.edu/>) (Soares, 2014). From a purely programming point of view, there is no opportunity to define a local variable except in ‘for each’ and ‘for range’ loops. Only recently, a student who wants to use a procedure in several screens has the ability to copy the blocks in the workspace of another screen (Schinkel et al., 2013) but still cannot access procedures from another screen. Another drawback is that the maximum size of a project is only 5 MB. Larger projects will not be compiled. In fact, projects larger than 5 MB can be compiled if the advanced student uses a tool called AppToMarket (<https://code.google.com/p/apptomarket/>). Although AIA block language eliminates many syntactic errors, runtime errors are still very common, especially index-out-of-bounds errors. AIA1 has two features to support students to debug them: Do It and watch. AIA2 has only the Do It feature. It would be useful to keep track of all variables, like Scratch (Okerlund, 2014).

Educators and learners are a primary audience for AIA as it is first and foremost an academic project (Pokress and Veiga, 2013). As a good introduction to the object-oriented paradigm, it provides a good foundation for acquiring more advanced computing skills in Python and other languages (Morelli et al., 2011). Additionally, in order to give novice programmers some “practice reading code in textual form [to] help their transition to more advanced programming courses” developers of MIT have created the App Inventor Java Bridge, a library that allows integrating AIA components into apps created in Java and Android SDK (<https://code.google.com/p/apptomarket/>) (Soares, 2014). Similarly, there is also a tool for converting AIA block code to a pseudo-Python textual language (Chadha, 2014). As a result, college and high school faculty have successfully used App Inventor in their classes, for over four years (Meerbaum-Salant et al., 2010; Wolber, 2011; Morelli et al., 2011) and there is a new CS Principles course based on AIA (Pokress and Veiga, 2013). AIA1 users with 1.4 million unique e-mail addresses have made over 3.5 million projects, and AIA2 users with nearly 1 million unique e-mail addresses have made over 2 million projects (Turbak et al., 2014a, 2014b; Okerlund and Turbak, 2013).

5 Similarities – differences Scratch and AIA

Many common programming errors are reduced for novice programmers by block languages like App Inventor and Scratch. This is accomplished by providing visual guidance for choosing, setting up, and understanding program structures (Okerlund and Turbak, 2013). Both programming environments are meant to introduce programming to those without prior experience and, as a consequence, the two environments share many

of the same designed goals. App Inventor and Scratch support graphics and sound and strive to engage the users by allowing them to write programs about things that connect with their interests in contrast to more conventional programming instruction where examples might be things like generating prime numbers or sorting a list of numbers (Utting et al., 2010). In this respect, AIA is comparable to Scratch as it aspires to make programming enjoyable and accessible to beginners. The difference, and perhaps an important reason for the attention it has gained, is that App Inventor lets student create apps for smartphones. Due to the widespread use of mobile devices among youngsters, students perceive programming for mobile devices to be more useful and important (Nikou and Economides, 2014). The result is that AIA engages younger students in computing (Morelli et al., 2011).

While both systems allow media to be imported, Scratch includes tools to draw images and record sounds. Moreover, in Scratch it is much easier than AIA to share projects. One of Scratch's main strengths is its popular project gallery where thousands of users share their projects and codes. AIA's website support for an app gallery lacks productivity compared to Scratch due to AIA's current inability to copy blocks between projects (Roy et al., 2012). AIA needs an easily available media library like Scratch, which allows social interaction and peer sharing among its users (Roy, 2012).

Although AIA aims at involving older students than Scratch, it also focuses on self-directed learning, and emphasises tinkerability too. Scratch and AIA are intended to be the means to introduce programming before (or perhaps in) CS1. Scratch has been used as an introduction to both Alice and Greenfoot (Maloney et al., 2010). AIA has been used as an introduction to Java. Command execution in both languages results to visible reactions of reified objects. For example, size, rotation and position of the objects can be changed. High-level commands and hidden low-level details (interaction loop, graphics primitives and display redraw) are also supported by both systems (Utting et al., 2010). The AIA command set is planned to promote a broader diversity of project types than Scratch. AIA and Scratch strive to eliminate gratuitous difficulties, by removing syntax issues, allowing users to concentrate on more important problems, rather than merely struggling to compile their program (Maloney et al., 2010; Utting et al., 2010). In Scratch runtime error messages are avoided through fail soft commands, while a carefully designed concurrency model avoids race conditions (Maloney et al., 2010). In AIA, runtime errors are more common, especially index-out-of-bounds errors due to support of lists and lists of lists.

The AIA's notional machine might surprise readers familiar with Scratch. Scratch has a multi-threaded model in which multiple events can be executed at the same time (Resnick, 2012). AIA has a single-threaded processing model in which only one event handler can be executed at a time. As in AIA, event handlers associated with components initiate all computation and if an event handler executes for a long time, the entire application will appear to be frozen as the system waits for that handler to complete (Resnick, 2012). Unlike Scratch, there is no interleaving between actions within loop bodies in different events in AIA. When the sprite moves, it jumps to its final position without any animation; the single-threaded nature of App Inventor prevents updates to the user interface while an event is running (Turbak et al., 2014a).

Table 1 Differences-similarities in pedagogical characteristics between AIA and Scratch

<i>Pedagogical characteristics</i>		
<i>Description</i>	<i>AIA</i>	<i>Scratch</i>
Computational thinking	Yes	Yes
Accessible and powerful	Yes	Yes
Problem-driven learning	Yes	Yes
Motivational potential	Yes	Yes
Embedded social features	No	Yes
Learnability	Yes	Yes
Target	Novice users	Novice and advanced users

Scratch has a single-window user-interface design with all the key components always visible. On the other hand, AIA interface has three main components. In Scratch, the execution of the code is visible as the executing code blocks have a white outline to denote that they are active. In AIA code, object mapping is not as obvious (Roy, 2012; Roy et al., 2012). The initial setup for Scratch is very simple. Installing is just a matter of downloading the appropriate binary file from the Scratch website. On the other hand, the setup for AIA1 is slightly more complicated as it uses Java (Roy et al., 2012). In AIA2, the initial setup is simpler. As both versions require the Android Debug Bridge (ADB) drivers for interfacing with the device, it could be hard to find the ADB driver for a particular phone or tablet (Roy, 2012) especially for cheaper not well-known brands.

The similarities and differences between AIA and Scratch are presented in the following tables in terms of their pedagogical characteristics and features.

Table 2 Differences-similarities in features between AIA and Scratch

<i>Features</i>		
<i>Description</i>	<i>AIA</i>	<i>Scratch</i>
Classes	No	No
Inheritance	No	No
Sharing sprites	No	Yes
Procedures	Yes	No (SNAP! Yes)
Recursion	Yes	No (SNAP! Yes)
Lists	Yes	No (SNAP! Yes)
Databases	Yes	No
First-class functions	Yes	No (SNAP! Yes)
Multi-threading	Yes (Scratch threading model)	No

6 Evaluation of AIA and Scratch

For the evaluation of AIA and Scratch as tools to facilitate programming learning for novices, we conducted a four-month teaching intervention, which included two experimental groups and one control group (Papadakis et al., 2015). The sample

consisted of 87 Greek students who attended classes in General and Vocational High Schools in the region of Attica, Greece, during the 2014–2015 school year. 35 students in the first experimental group were taught basic programming concepts using AIA and 34 students in the second group using Scratch. The control group (18 students) was taught basic programming concepts with the use of a typical programming language such as Pascal. In order to evaluate how the programming environment shapes the attitudes, perceptions and beliefs of students in programming we used the computer attitude scale (CAS) (Loyd and Gressard, 1984). Additionally, in order to evaluate the students programming knowledge, we used the questionnaire programming knowledge (QPK) (Kleinschmager and Hanenberg, 2011). The research process design included three phases: a pre-test, a teaching-intervention and a post-test phase (Table 3).

The statistical data analysis was performed using the IBM SPSS Statistics for Windows (version 23.0). The standard level of significance used to justify a claim of a statistically significant effect was 0.05. The three groups were equivalent in terms of age ($F(2, 84) = 0.65, p > 0.05$), gender ($\chi^2(2) = 1.01, p > 0.05$) and knowledge in basic programming concepts ($F(2, 84) = 0.82, p > 0.05$) as well as their attitudes towards programming ($F(2, 84) = 0.65, p > 0.05$). For the implementation of the teaching intervention in the three groups, we used the same learning scenarios, which were created specifically for the purpose of the research.

Table 3 The research design

<i>Experimental and control groups</i>	<i>Pre-experimental procedure</i>	<i>Experimental procedure</i>	<i>Post-experimental procedure</i>
Experimental group 1	Pre-test	Teaching intervention	Post-test
experimental group 2			
Control group			

In order to compare the performance of students in CAS and QPK before and after the experimental intervention we used the paired samples t-test with Bonferroni correction. The results indicated that all groups made some progress in their behavioural characteristics and programming achievements over time. The only difference that was found to be statistically significant was between the performances of children in each experimental group (Table 4).

Table 4 The results of the analysis in students' performance in CAS and QPK per group

Experimental group 1 (AIA)	CAS	$t(34) = -5.61, p < 0.05^*$
	QPK	$t(34) = -7.34, p < 0.05^*$
Experimental group 2 (Scratch)	CAS	$t(33) = -4.01, p < 0.05^*$
	QPK	$t(33) = -5.49, p < 0.05^*$
Control group	CAS	$t(17) = -2.30, p > 0.05^*$
	QPK	$t(17) = -3.32, p > 0.05^*$

Note: *Based on Bonferroni correction when $p < 0.016$, then $p < 0.05$.

In addition to the final performance, it was considered useful to investigate the effect of the experimental intervention on the degree of improvement in children's performance in attitude and programming ability (Table 5). According to the results, the greatest improvement in CAS and QPK was demonstrated by students of the first experimental

group (AIA), followed by children of the second experimental group (Scratch), while the worst improvement was demonstrated by children of the control group.

Table 5 Means (M) and standard deviations (SD) of the magnitude of the improvement of students performance

		<i>M</i>	<i>SD</i>
CAS	Experimental group 1	7.58	4.35
	Experimental group 2	4.32	4.04
	Control group	3.20	3.01
QPK	Experimental group 1	8.13	5.28
	Experimental group 2	4.25	4.16
	Control group	3.01	2.80

The results of separate ANOVA indicated that the effect of the intervention on the degree of improvement in students' performance in CAS and QPK was statistically significant, $F(2, 84) = 24.71$, $p < 0.01$ and $F(2, 84) = 22.85$, $p < 0.01$. The statistically significant difference among the three groups necessitated investigating differences between the two experimental groups and the control group, pairwise, using multiple Bonferroni comparisons. The results of analyses indicated that at the end of the study the two experimental groups outperformed the control group. In addition, significant differences were found between the two experimental groups as the first experimental group (AIA) outperformed the second experimental group (Scratch).

7 Comments on the results

The results of this study showed that the techniques of teaching programming using Scratch and AIA contributed so that students significantly improved their performance on the evaluation criterion of attitudes and knowledge in programming. Additionally, in the condition where the teaching intervention with the use of AIA was applied, students had better performance on both evaluation criteria compared to teaching intervention with Scratch and the control group.

This study has some limitations. The duration of the teaching intervention was relatively small. Moreover, empirical data from the school units is not generalisable because of the small sample and its origin from one region of Greece. Nevertheless, since in Greece there are not enough studies on the educational advantages brought by the use of AIA in secondary education, we believe that the present research findings enrich the syllabus. The results of this study are compatible with a variety of studies confirming the superiority of AIA for teaching programming not only in novice programmers but also in collegiate-level programming courses (Morelli et al., 2011; Wolber, 2011). In addition, it confirms the potential positive impact of using smart mobile devices (smartphones and tablets) in the educational process (Zaranis et al., 2013).

8 Conclusions

App Inventor and Scratch strive to engage the novice programming students by enabling them to act on projects, which hold more real life context (Roy et al., 2012) in contrast to more conventional programming (Utting et al., 2010). Those NPEs have successfully lowered the barrier of initial entry to programming satisfying the triplet of low-floor/high-ceiling/wide-walls (Papert, 1980; Gudgial, 2004 as cited in Resnick et al., 2009) as they are based on two critical and apparently opposing criteria: richness and simplicity. As a result, both programming environments seemed to be attractive platforms for engaging students at all levels in the computing curriculum, K-12 through university (Morelli et al., 2011). Especially their combination, in an IT0/CS0 course is unique in introducing fundamental concepts in computer programming and holds some promise as an appealing course for both majors and non-majors.

AIA appears to be a more popular programming environment for use with learners in the teenage group than Scratch because it fascinated them as they can create personalised software for their most precious device, the smartphone/tablet, and literally augment their own reality (Wolber, 2011). Being able to program real-world apps is highly motivating to students and is the key to providing a successful first engagement, which ultimately helps to broaden and diversify the pool of new coders (Turbak et al., 2014a). On the other hand, Scratch possibly is the most appropriate for teaching young students or early learners 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. Thus, for the teaching of programming, it is better to begin with Scratch in primary school or the early junior classes and then move on with AIA in the next classes and school levels. Taking into consideration the development of ScratchJr, a mobile version of Scratch, which aimed at five to seven year olds we think that the combination ‘ScratchJr → Scratch → AIA → Conventional Language’ appears to be a very powerful tool for teaching programming and computing fundamentals. Information technology related majors, STEM fields, and non-majors highlighted the importance of young people’s development as computational creators as well as computational thinkers.

References

- Barshay, J. (2014) *Getting Boys – and Girls – Interested in Computer Coding* [online] http://educationbythenumbers.org/content/girls-computer-coding_1691/31 (accessed 16 January 2016).
- Brennan, K. and Resnick, M. (2012) ‘New frameworks for studying and assessing the development of computational thinking’, in *2012 Annual Meeting of the American Educational Research Association*, Vancouver.
- Chadha, K. (2014) *Improving the Usability of App Inventor through Conversion between Blocks and Text*, Wellesley College Undergraduate Senior Honors Thesis.
- Code.org (2015) *Make Computer Science in K-12 Count!* [online] http://code.org/files/convince_your_school_or_state.pdf (accessed 15 January 2016).
- Dekhan, S., Xu, X. and Tsoi, M.Y. (2013) ‘Mobile app development to increase student engagement and problem solving skills’, *Journal of Information Systems Education*, Vol. 24, No. 4, p.299.
- European Commission (2014) *Vassiliou Urges Education Ministers to Help Kids Crack the Code* [online] http://europa.eu/rapid/press-release_IP-14-887_en.htm (accessed 10 January 2016).

- Forte, A. and Guzdial, M. (2004) 'Computers for communication, not calculation: media as a motivation and context for learning', in *37th Annual Hawaii International Conference on System Sciences*, IEEE Press, New York, pp.1–10.
- Gray, J., Abelson, H., Wolber, D. and Friend, M. (2012) 'Teaching CS principles with app inventor', in *50th Annual Southeast Regional Conference (ACM-SE'12)*, ACM, New York, pp.405–406.
- Grover, S. and Pea, R. (2013) 'Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students', in *44th ACM Technical Symposium on Computer Science Education*, New York, Vol. 2, pp.723–728.
- Guzdial, M. (2004) 'Programming environments for novices', in Fincher, S. and Petre, M. (Eds.): *Computer Science Education Research*, Taylor & Francis, London.
- Guzdial, M. and Soloway, E. (2002) 'Teaching the Nintendo generation to program', *Communications of the ACM*, Vol. 45, No. 4, pp.17–21.
- Harvey, B. (2012) 'The beauty and joy of computing: computer science for everyone. Plenary presentation', in *Constructionism 2012*, Athens.
- Harvey, B. and Mönig, J. (2010) 'Bringing 'no ceiling' to scratch: can one language serve kids and computer scientists?', in *Constructionism*, Paris.
- Karakus, M., Uludag, S., Guler, E., Turner, S.W. and Ugur, A. (2012) 'Teaching computing and programming fundamentals via app inventor for Android', in *Information Technology Based Higher Education and Training (ITHET)*, IEEE Press, Istanbul, pp.1–8.
- Kelleher, C. and Pausch, R. (2007) 'Using storytelling to motivate programming', *Communications of the ACM*, Vol. 50, No. 7, pp.58–64.
- Kleinschmager, S. and Hanenberg, S. (2011) 'How to rate programming skills in programming experiments? A preliminary, exploratory, study based on university marks, pretests, and self-estimation', in *Proc. ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools*, ACM Press, pp.15–24.
- Kloss, H.J. (2012) *Android Apps with App Inventor: The Fast and Easy Way to Build Android Apps*, Addison-Wesley Professional, Boston, USA.
- Kurkovsky, S. (2007) 'Making computing attractive for non-majors: a course design', *J. Comput. Sci. Coll.*, Vol. 22, No. 3, pp.90–97.
- Loyd, B.H. and Gressard, C. (1984) 'Reliability and factorial validity of computer attitude scale', *Educational and Psychological Measurement*, Vol. 44, No. 2, pp.501–505.
- Malan, D.J. and Leitner, H.H. (2007) 'Scratch for budding computer scientists', in *ACM SIGCSE Bulletin*, Vol. 39, No. 1, pp.223–227, ACM, New York.
- Maloney, J., Resnick, M., Rusk, N., Silverman, B. and Eastmond, E. (2010) 'The scratch programming language and environment', *ACM Transactions on Computing Education (TOCE)*, Vol. 10, No. 4, pp.1–15.
- Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M. (2010) 'Learning computer science concepts with scratch', in *6th International Workshop on Computing Education Research*, ACM, Aarhus, Denmark, pp.239–264.
- Monroy-Hernandez, A. and Resnick, M. (2008) 'FEAUTURE: empowering kids to create and share programmable media', *Interactions*, Vol. 15, No. 2, pp.50–53, ACM, New York.
- Morelli, R., de Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E. and Uche, C. (2011) 'Can Android app inventor bring computational thinking to K-12?', in *42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)*.
- Nikou, S.A. and Economides, A.A. (2014) 'Transition in student motivation during a scratch and an app inventor course', in *Global Engineering Education Conference (EDUCON)*, IEEE Press, New York, pp.1042–1045.
- Okerlund, J. (2014) *Improving App Inventor Debugging Support*, Undergraduate Senior Honors Thesis, Wellesley College.

- Okerlund, J. and Turbak, F. (2013) 'A preliminary analysis of app inventor blocks programs', in *IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC 2013)*, San Jose.
- Papadakis, St. and Orfanakis, V. (2014) 'A new programming environment for teaching programming. A first acquaintance with enchanting', *The 2nd International Virtual Scientific Conference – Scieconf 2014*, EDIS – Publishing Institution of the University of Zilina, Slovakia, pp.268–273.
- Papadakis, St., Kalogiannakis, M., Orfanakis, V. and Zaranis, N. (2014) 'Novice programming environments. Scratch & app inventor: a first comparison', in Fardoun, H.M. and Gallud, J.A. (Eds.): *Proceedings of the 2014 Workshop on Interaction Design in Educational Environments*, ACM, New York, NY, USA.
- Papadakis, St., Xefterakis, N., Kardara, M.L., Saldeou, P., Sandali, G., Saranteas, J. and Makrigiannis, P. (2015) 'How the programming environment shapes the attitudes, perceptions, beliefs and knowledge of students in programming. A comparison between scratch and app inventor [in Greek]', *Proceedings of the 9th Panhellenic Scientific Conference of the Informatics in Primary and Secondary Education*, Kastoria, Greece.
- Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M. and Paterson, J. (2007) 'A survey of literature on the teaching of introductory programming', in *ACM SIGCSE Bulletin*, Vol. 39, No. 4, pp.204–223, ACM, New York.
- Pokress, S.C. and Veiga, J.J.D. (2013) 'MIT app inventor: enabling personal mobile computing', in *Computer Science/Computers and Society – ePrintarchive*, Cornell University.
- Resnick, M. (2009) *Making Programming Universally Accessible and Useful*, Proposal to Google [online] <http://www.media.mit.edu/~mres/proposals/Google-grant-proposal.pdf>32 (accessed 6 January 2016).
- Resnick, M. (2010) 'Preparing the next generation of computational thinkers: transforming learning and education through cooperation in decentralized networks', in *Proposal to the National Science Foundation*.
- Resnick, M. (2012) 'Mother's day, warrior cats, and digital fluency, stories for the scratch online community', in *Constructionism 2012 Educational Technology Lab*, pp.52–58, Athens.
- Resnick, M. (2013) 'Coding for all: interest-driven trajectories to computational fluency', in *Proposal to the National Science Foundation*.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. (2009) 'Scratch: programming for all', *Communications of the ACM*, Vol. 52, No. 11, pp.60–67.
- Rizvi, M., Humphries, T., Major, D. Jones, M. and Lauzun, H. (2011) 'A CS0 course using scratch', *J. Comput. Small Coll.*, Vol. 26, No. 3, pp.19–27.
- Roy, K. (2012) 'App inventor for android: report from a summer camp', in *43rd ACM Technical Symposium on Computer Science Education (SIGCSE'12)*, pp.283–288, ACM, New York.
- Roy, K., Rousse, W.C. and DeMeritt, D.B. (2012) 'Comparing the mobile novice programming environments: app inventor for Android vs. GameSalad', in *2012 IEEE Frontiers in Education Conference (FIE) (FIE '12)*, IEEE Press, New York, pp.1–6.
- Schinkel, F., Schlag, A., Sachs, G., Haas, S. and Schäfer, P. (2013) *Evaluation of Android Programming Environments* [online] <http://bit.ly/VHiZ3Z42> (accessed 12 January 2016).
- Shein, E. (2014) 'Should everybody learn to code?', *Communications of the ACM*, Vol. 57, No. 2, pp.16–18.
- Shih, H., Jackson, J., Hawkins Wilson, C. and Yuan, P.C. (2014) 'Using MIT app inventor in an emergency management course to promote computational thinking', in *121st ASEE Annual Conference & Exposition*, Indianapolis.
- Soares, A. (2014) 'Reflections on teaching app inventor for non-beginner programmers: issues, challenges and opportunities', in *Information Systems Education Journal (ISEDJ)*, Vol. 12, No. 4, pp.56–65.

- Spertus, E., Chang, M.L., Gestwicki, P. and Wolber, D. (2010) 'Novel approaches to CS 0 with app inventor for android', in *41st ACM Technical Symposium on Computer Science Education*, pp.325–326, ACM, New York.
- Turbak, F., Sherman, M., Martin, F., Wolber, D. and Pokress, S.C. (2014a) 'Events-first programming in app inventor', in *Consortium for Computing Sciences in Colleges Northeastern Region Conference (CCSCNE 2014)*, pp.81–89.
- Turbak, F., Wolber, D. and Medlock-Walton, P. (2014b) 'The design of naming features in app inventor 2', in *IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC 2014)*, Melbourne.
- Utting, I., Cooper, S., Kolling, M., Maloney, J. and Resnick, M. (2010) 'Alice, greenfoot and scratch – a discussion', in *ACM Transactions on Computing Education (TOCE)*, Vol. 10, No. 4, pp.1–11.
- Vasek, M. (2012) *Representing Expressive Types in Blocks Programming Languages*, Undergraduate Thesis, Wellesley College.
- Wilson, C., Sudol, L., Stephenson, C. and Stehlik, M. (2010) *Running on Empty: The Failure to Teach K-12 Computer Science in the Digital Age*, Association for Computing Machinery.
- Wing, J. (2006) 'Computational thinking', *Communications of the ACM*, Vol. 49, No. 3, pp.33–36.
- Wolber, D. (2011) 'App inventor and real-world motivation', in *42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*, pp.601–606, ACM, New York.
- Wolz, U., Leitner, H.H., Malan, D.J. and Maloney, J. (2009) 'Starting with scratch in CS1', in *ACM SIGCSE Bulletin*, Vol. 41, pp.223–227, ACM, New York.
- Zaranis, N., Kalogiannakis, M. and Papadakis, St. (2013) 'Using mobile devices for teaching realistic mathematics in kindergarten education', *Creative Education*, Vol. 4, No. 7A, pp.1–10.