⌈Araştırma Makalesi / Research Article⌋

Kastamonu Eğitim Dergisi
Kastamonu Education Journal

Kasım 2019 Cilt:27 Sayı:6
kefdergi.kastamonu.edu.tr

Başvuru Tarihi/Received: 06.03.2019
Kabul Tarihi/Accepted: 01.04.2019
DOI: 10.24106/kefdergi.3640

Comparing Three Free to Use Visual Programming Environments for Novice Programmers

Ücretsiz Üç Görsel Programlama Ortamının İncelenmesi ve Karşılaştırılması

Kadir Yücel KAYA[1], İsmail YILDIZ[2]

Öz

Bu çalışma popüler ücretsiz üç görsel programlama ortamının incelenmesini ve karşılaştırmasını hedeflemektedir. Bahsedilen üç farklı ortamı diğer ücretsiz ortamlar arasından seçerken ücretsiz, popüler, ve üretken olması ölçütleri dikkate alınmıştır. Birçok ortam incelendikten sonra MIT App Inventor, Scratch ve Microsoft Kodu Game Lab seçilmiştir. App Inventor ve Scratch ücretsiz ve açık kaynak kodlu ortamlarken, Microsoft Kodu sadece ücretsizdir. Seçilen üç ortam kullanılarak ve incelenerek araştırılmış, ayrıca araştırma alanyazın taraması ile desteklenmiştir. Bu çalışmanın sonuçlarının öğretmen, eğitmen ve öğrencilerin ihtiyaçlarına uygun bir görsel programlama ortamı seçmede yardımcı olabileceği düşünülmektedir. Ortamların incelenmesi, Kodu'nun ilkokul düzeyindeki öğrenciler için daha uygun olduğu sonucunu ortaya koyarken, Scratch'in hedef kitlesinin benzer fakat daha üst yaş kitlesini de içerdiğini göstermiştir. Bu ikisi arasındaki diğer bir fark Kodu'nun tek ürünü 3 boyutlu oyunlarken, Scratch 2 boyutlu oyun ve animasyonlar üretebilmektedir. App Inventor ise ortaokul ve daha üst yaş kitlesini hedeflemekte ve App Inventor'da Android işletim sistemi için mobil uygulamalar geliştirilebilmektedir. Scratch ve App Inventor aynı blok tabanlı kütüphaneyi kullanmakta ve bu ikisi değişkenler, koşullu ifadeler, ve döngüler gibi temel programlama kavramlarını öğretmeye daha uygun bulunmuştur. Seçilen üç ortamın kendilerine özel avantajları ve kendi hedef kitlelerine yönelik özellikleri olduğu görülmüştür. Bu çalışma seçilen ortamların önemli farklılıkları ve özelliklerini araştırmıştır.

Anahtar Kelimeler: app inventor, görsel programlama, kodu, scratch, ücretsiz programlama ortamları

Abstract

This study aims to examine and compare three popular free-to-use visual programming environments. While choosing three environments among other visual programming environments, three criteria were taken into account which are being completely free, popular, and productive. After reviewing several environments, MIT's App Inventor, Scratch and Microsoft's Kodu Game Lab were chosen. While App Inventor and Scratch are free and open source environments, Microsoft's Kodu is only free to use. Selected three environments were investigated through using and examining the environments and literature review. Outcomes of this study can help teachers, instructors and students to choose a relevant visual programming environment based on their needs. Review of the environments showed that while Kodu is more relevant for elementary students, Scratch's target group are similar but also includes higher age range. Another difference between them was that Kodu's sole purpose is to develop games in 3D, Scratch is used for 2D games and animations. App Inventor, on the other hand, targets middle school and higher age range to develop mobile applications for Android OS. Scratch and App Inventor uses the same block-based library which is more relevant to teach basic programming concepts such as variables, conditional expressions, and loops than Microsoft's Kodu. Selected three environments have the advantages of their own and features specifically for their target audience and products. This study investigated the important differences and features of the selected environments.

Keywords: app inventor, free programming environments, kodu, scratch, visual programming

1. Kastamonu Üniversitesi, Eğitim Fakültesi, Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü, Kastamonu, Türkiye; https://orcid.org/0000-0001-7561-980X
2. Kastamonu Üniversitesi, Eğitim Fakültesi, Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü, Kastamonu, Türkiye; https://orcid.org/0000-0003-3048-2840

Atıf / Citation: Kaya, K. Y. ve Yıldız, İ. (2019). Comparing Three free to use visual programming environments for novice programmers. *Kastamonu Education Journal, 27*(6), 2701-2712. doi:10.24106/kefdergi.3640

Extended Abstract

Aim : This study aims to examine and compare three popular free-to-use visual programming environment. Through the examination of visual programming environments, important differences and features were aimed to be specified.

Methodology : Purpose of this study is to examine and put forward characteristic similarities and differences of the selected visual programming environments: Scratch, App Inventor, and Kodu. In this study, content analysis methodology was used to get a detailed insight of the visual programming environments including their categorically separated differences and similarities. In content analysis, researchers create categories and names from the flow of the data instead of predetermined categories and names. Categories and findings are supported with literature review. Research question of this study is: What are the key characteristic similarities and differences between three free visual programming environments?

Findings: After the literature review and examination of the environments some of the key characteristics emerged and dissociated from each environment. Key characteristics includes resources, community and support, online/offline capability, use of programming concepts, product range, and target group. Regarding resources, community, and support Scratch has the richest resources and community. Scratch is also more advantageous about the accessibility which is usable through online and offline interface. Additionally, users can reach the products through a web browser from nearly any modern computer operating system. Two of the important differences between environments were programming expressions and working logics of programming concepts. While App Inventor and Scratch uses similar but simpler version of traditional textual programming concepts with visualization, Kodu uses a different, event-driven only, tile-based programming language. Different than the other two environments, programming was made with sequential events through When/Do rules. App Inventor and Scratch used the same or similar logic with essential programming concepts like variables, conditional statements, and loops. On the other hand, Kodu uses its own game elements that replaces those concepts. Instead of variables, scores; instead of conditional statements, when/do statement; instead of loops, timers were used in Kodu. The last found important difference between the environments are the product types they offer to the users. Kodu is a 3D game development platform that only allows 3D games with provided game objects. Scratch was also limited to develop 2D games and animations. App Inventor, on the other hand, have a more flexible yet more complex product range when compared with the others.

Implications and Future Studies : Rather than selecting an environment and hoping for the best, providing proper guidance and explanation could be crucial for enhancing the learning of non-programmers for essential programming concepts, especially the complex ones. Our examination showed that three popular visual programming environments are separated based on some characteristics and features. Instructors should be aware that advantages, disadvantages, and relevance of an environment for the target audience. Studies showed that each environment has unique properties for a unique audience that could affect the learning outcome of the novice programmers.

1. Introduction

Learning textual programming languages such as C++, and Java is proven to be difficult (Dekhane, Xu, & Tsoi, 2013). Additionally, learning syntax could be one of the barriers that novice programmers face while learning programming. Winslow (1996) stated that novice programmers know the syntax and meaning of statements but they do not know how to create program with this knowledge (cited in Robins, Rountree & Rountree, 2003). Novice programmers sometimes think programming is the production of program text rather than controlling computer's actions at runtime (Sorva, Lönnberg, & Malmi, 2013). Visual programming language is defined as "a programming language that lets users to create programs by manipulating program elements graphically rather than specifying them textually" (Smutny, 2011, p. 358). Visual programming could solve some of the problems of non-programmers with its easy-to-learn and intuitive nature by using drag-and-drop interaction (Hsu & Ching, 2013; Hsu, Lou, & Sun, 2016). These programming environments offer a fun environment for students. (Weintrop & Wilensky, 2017). Immediate visual feedback of the visual programming environments can enhance the motivation of the computing class students (Dekhane, Xu, & Tsoi, p. 307). Visual programming can help novice programmers to avoid syntax errors and make programming a more enjoyable experience (Hsu & Ching, 2013). Focusing on syntax and errors could increase the cognitive load of novice learners. Students are frequently forgetting to write a semicolon or a parentheses and stuck in the same place without knowing what to do (Bennedsen & Caspersen, 2012). Spohrer (1989) also suggest use of visual programming especially for the novices (cited in Robins et al., 2003).

Visual programming languages holds the potential for effectively teaching the programming concepts that students have problems. Studies shows that students find essential programming concepts difficult to understand such as variables, loops, conditionals (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Guzdial, 2003). Additionally, learning programming through visual design can reflect integrative, authentic, meaningful and concrete understanding in target subjects (Baytak & Land. 2010). As the numbers of the visual programming environments are growing bigger, choosing a relevant programming environment is getting harder. Although the programming environment is not the main focus, students have to learn one and it plays an important role in the learning process (Mannila et al., 2006). Overcoming the mistakes and creating error-free programs is a challenge and a struggle for non-majors who are taking introductory programming courses (Wiedenbeck, 2005). Choosing a relevant programming environment could shorten the steps to build an error-free program. Starting with a relevant programming environment as important as the other instructional strategies, since it shapes the perspective of students towards programming and it impacts student-teacher dynamics (Deek & McHugh, 1998; Felleisen, Findler, Flatt, & Krishnamurthi, 2004). This study sought to find the main differences between three popular, free to use visual programming environments regarding their capability to present essential information on programming, their features, and their relevancy for specific audience and purpose.

Visual Programming and Computational thinking

Computational thinking can be considered as a problem-solving toolset that goes beyond information technology to different thinking styles that include algorithmic and parallel thinking (Yadav, Gretter, Good, & McLean, 2017). Computational thinking "allows students to tackle problems, to break them into solvable chunks, and to devise algorithms to solve them" (Mueller, Beckett, Hennessey, & Shodiev, 2017, p. 262). By using the non-syntax focused structure of visual programming environments and the examples connected to real-world could help students embrace the novel thinking styles. Mannila et al. (2006) discussed that for non-majors concentrating on algorithmic thinking and programming is more important than concentrating on programming languages and syntax. To help students to embrace the computational thinking, it should be provided constantly and through unconscious learning. Rather than defining the computational thinking and its step, applying it through the development of programs through algorithms and the steps of computational thinking such as abstraction is more essential for the development of learners (Sadik, Leftwich, & Nadiruzzaman, 2017). Choosing a relevant environment would help instructors and teachers to apply computational thinking principles without having to deal with syntax, or problems of the selected environment.

Block-based and Tile-based Programming Environments

Selected three environments in this study were App Inventor, Scratch and Kodu. Reviewing the logic and library of the environments could be important, since it affects their ability to present certain programming concepts. While selected Scratch and App-Inventor were block-based programming environments, Kodu was a tile-based programming environment. Two of the most popular block-based visual programming environments are Scratch and App Inventor. Both of the environments were developed by MIT. App Inventor and Scratch uses the same block-based programming library: Blockly. Blockly is defined by Google (2018) as a visual code editor that uses interlocking, graphical blocks to represent code concepts like variables, logical expressions, loops, and more which allows users to apply programming principles without having to worry about syntax.

Block-based programming environments offer a puzzle-like composition environment that helps individuals design the relationships and process of the logical framework through the interlocking blocks. To perform these steps, the environment helps the users with visual cues. The user can focus on the logical process instead of language rules and syntax. Since App Inventor and Scratch uses same library programming interface and logic are very similar in both. On the other hand, Kodu uses a tile-based programming logic in which actions and events were put in order by user through the steps into the objects. Kodu is specifically designed for game development and offers special principles derived from game scenarios. Kodu programming environment saves time for the user since there is no chance to make mistakes in the language rules. Kodu's programming interface and logic is entirely event-driven in which programming is formed by sequential tiles through the events triggered by a condition or an action (Fowler, Fristce, & Maclauren, 2012). Its underlying structure and programming logic is different than the mentioned block-based programming environments which will be mentioned in detail in this study.

Scratch

Scratch is an educational programming environment which lets users create interactive media rich projects, created by MIT-media lab and Yasmin Kafai team from UCLA (Kwon, Yoon, & Lee, 2011; Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). Scratch project publicly launched in 2007 and it is growing since then (Maloney et al., 2010). Scratch is a useful and rich environment for novice programmers and learners. Novice programmers can learn programming with interest while creating their own games or animations by using Scratch environment (Kwon, Yoon, & Lee, 2011). Scratch is created mainly for K-12 students (Resnick et al., 2009; Wilson, Hainey, & Connoly, 2012;). Scratch has a site that includes projects, tutorials, forums etc. Users can get help from other Scratch users and examines each other's projects through the Scratch community from the discussion pages. Core audience of the site is children between 8 and 16, however adults also participate to the site (Resnick, et al., 2009). From the release to 2013, more than 3 million projects of 1.5 million registered members (Meerbaum-Salant, Armoni, & Ben-Ari, 2013) have been uploaded which are open source and shared with other users. Latest figures showed that it is growing bigger, since in total nearly 39 million projects were shared, and more than 36 million users registered (Scratch 2019).

While creating their programs with Scratch, students also learn creative and systematic thinking, problem solving skills, mathematical and computational concepts which are parts of computational thinking (Lee, 2011; Resnick, et al., 2009). Scratch provides an easy to design environment in which user can even make changes while program is running (Resnick, et al., 2009). In that way, it also encourages users to experiment. Scratch has recognized as a highly potential first language for first-time programmers (Tangney, Oldham, Conneely, Barrett, & Lawlor, 2010). Mostly used first example of programming, "Hello world" is just a two-piece puzzle in Scratch environment (Malan & Leitner, 2007).

Studies also show that students also find Scratch as a fun to use environment (Malan & Leitner, 2007; Maloney et al., 2010). Scratch environment is coherent with the low threshold (easy to learn), high ceiling (ability to create complex projects) concept (Su, Yang, Hwang, Huang, & Tern, 2013). Wolsz et al. (2009) reported that "after initially learning Scratch, the students' transition to Java or C appeared to be easier" (cited in Meerbaum-Salant, Armoni, & Ben-Ari, 2013).

On the negative side some studies showed that Scratch does not always meet the need to be used as a learning tool for the first step of programming. Maloney et al. (2008) put forward that some of the crucial programming concepts such as variables, boolean logic, and random numbers need to be explained by instructors. Scratch has a very similar interface and logic with AI. However, Scratch is designed and developed for children at its core and products of it are animations and games.

App Inventor

"MIT App Inventor (AI) is a drag-and-drop visual programming tool for designing and building fully functional mobile apps for Android" (Pokress & Veiga, 2013, p.1). App Inventor (2013) was created for users without coding experience to make simple apps for mobile phones which was released in 2010 (Bertea, 2011). According to its own website, App Inventor have over 400,000 unique monthly active users who come from 195 countries who have created almost 22 million apps (MIT App Inventor, 2019). AI has two main components: designer part and blocks editor part (Pokress & Veiga, 2013). While blocks editor is used for behavior of the application (programming part), designer let the user to design and place the components to the screen. In designer screen, there are many components to use, including buttons, textboxes, GPS, Bluetooth, sensors etc. User can just drag and drop those components to the screen and add behaviors to them in the blocks editor (Bertea, 2011). AI's blocks editor, on the other hand, focuses on the functionality of the application. Users can create their applications intuitively without any programming knowledge just by exploring the components (Bertea, 2011; Pokress & Veiga, 2013). Visual cues that AI provides, reduces the chance of errors (Sandoval-Reyes et al., 2011).

AI makes application development considerably easier than traditional programming language (Smutny, 2011). Smutny (2011) lists key features of visual language of AI as; (1) has no syntax, (2) based on idea what happens when component do certain action, (3) no need for a manual - drag and drop, (4) restrict users from making mistakes, (5) concrete, (6) has a powerful library. Hsu and Ching (2013) states that AI has a great potential for novice programmers to develop apps even for professional needs. AI could also provide an easy-to-use environment to let teachers to use as a tool to teach computational thinking. Hsu and Ching (2013) stated that AI motivates students learn programming logic and to engage in creative problem solving process. App Inventor is also relevant to teach in higher education. In Hsu and Ching's (2013) study, participants from higher education defined designing an app with AI as fun and useful. In the same study, even the participants with programming experience find AI satisfying and incorporate it with Java language. Pokress and Veiga (2013) states that college and high-school faculty have successfully used AI in their courses over the 4 years. As the advantages, AI has also some disadvantages. One disadvantage of AI is the fact that, while simple applications are easy to make, complex applications could need deeper knowledge (Bertea, 2011). Additionally, when programs get complicated, block designer gets annoying as well (Hsu & Ching, 2013). If there was an option to switch from blocks editor to text editor, it would be more helpful both for control the program and to be a step for advanced programming.

Kodu

Kodu is an easy-to-use visual programming environment that enables children and beginner programmers to design and construct their own digital games, learn logical thinking and setting ground for more advanced programming (Nygård, Kolås, & Sigurdardottir, 2016; Stolee & Fristoe, 2011). Kodu is introduced in 2009 by Microsoft as a free game creation tool especially for young audience which uses a tile-based system instead of codes or blocks (Nygård, Kolås, & Sigurdardottir, 2016). Unlike Scratch and App Inventor, Kodu is specifically for game creation. Game design is an ideal context to promote children's thinking skills which can help them to make critical decisions and develop their problem-solving skills (Akcaoglu, 2014).

Main difference of Kodu from other two environments is being 3D and comprises some game development tools like terrain editor, lay out tools, character menus (Arilesere, 2014). Coy (2013) states that Kodu's being very high-level language was one of their intentions when designing it to help it become more intuitive. Kodu is more accessible for visually impaired people since it uses big tiles with large icons and using images as well as words make Kodu also a good choice for non-English speakers (Coy, 2013). In a study, it is aimed to have an idea about how fourth and fifth grade students developed their learning skills while learning the Kodu programming and the results showed that students could read and understand the codes written by others and predict the behavior of the program when they learned the rules of Kodu programming (Aggarwal & Gardner-McCune, 2018). It shows that tile-based language of Kodu could be read and understood by others easily which proofs transferability of the easier logic of Kodu environment.

One of the advantage of Kodu to other two environments is the ability to develop 3D games easily and fast. In a traditional 3D game development environment, it would take vast amount of time (if ever) to develop a functional game for a novice programmer. However, a 3D game could be developed in just minutes without the extraneous load. On the other hand, without dealing with the complexity of a 3D game development, users need to follow the rules and scenarios provided by Kodu and it expects users to do them in certain ways (Kelly, 2013).

2. Methodology

Purpose of this study is to examine and put forward characteristic similarities and differences of the selected visual programming environments: Scratch, App Inventor, and Kodu. In this study content analysis methodology were used to get a detailed insight of the visual programming environments including their categorically separated differences and similarities. In content analysis, researchers create categories and names from the flow of the data instead of predetermined categories and names (Hsieh & Shannon, 2005). Content analysis technique is "not restricted to the domain of textual analysis, but may be applied to other areas" (Stemler, 2001, p. 1). Berg (2001) explained content analysis as systematic examination and interpretation of particular body of materials to identify patterns, themes, biases, and meanings. Categories and findings are supported with literature review. Through the content analysis, user interface, programming logic, documentation, forums, and websites of the selected environments were investigated.

Our criteria to choose the environments were (a) being popular, (b) completely free to use, (c) easy to learn from scratch, and (d) being able to program completely visual. In the beginning of the study, through the literature review popular visual programming environments were listed. Among all, 6 programming environments came forward with their visual only programming interface: Scratch, App Inventor, Alice, Kodu, GameSalad, and GameMaker. Examination of the official websites of the environments showed that GameSalad and GameMaker are not free environments and they were eliminated from this study. Remaining four environments were examined according to their capabilities in terms of programming capabilities, relevancies, and features. However, first examination demonstrated that Alice needs learners to know the conceptual programming information that are used in the coding part. Due to this reason Alice was also excluded from this study.

As the final decision to examine and compare three free visual programming environments were App Inventor, Scratch and Kodu Game Lab. To understand the environments and their characteristics and features, researchers used the environment to create basic applications, animations and games. Each environment was examined through using to develop basic programs, reviewing the documentation, and written information. After using the environments their capability to teach programming logic was also investigated, and it was tried to be demonstrated with a basic example in the findings part. Additionally, their categorical differences and unique properties were presented in a table.

Research question of this study is:

What are the key characteristic similarities and differences between three free visual programming environments?

3. Findings

After the literature review and examination of the environments some of the key characteristics emerged and dissociated for each environment. Key characteristics includes resources, community and support, online/offline capability, use of programming concepts, product range, and target group. These characteristics were examined and compared among three selected environments. Comparison and characteristics of the environments could help teachers and students to choose a relevant environment.

Resources, Community and Support

In terms of resources all three environments have enough resource and documentation that could easily be accessed online. However, when it comes to community and support Scratch is come forward among them. Role of the discussion groups is crucial for new learners to seek support and find information (Alzahrani, 2017; Liu, Cheng & Lin, 2013). While App Inventor uses Google Groups as the official discussion group, Scratch and Kodu use their own forums. Number of topics and posts are not visible in App Inventor's site, but it could be said that it is currently active since at the time of writing of this paper there are a number of recent posts. Scratch, on the other hand, has the most active discussion forum. Scratch has more than 240.000 topics, and 2.7 million posts in its discussion forum. On the other hand, Kodu's discussion forum is not very active. It has 375 topics in total, and latest post was made in 2016. Information suggests that Scratch come forward among three, if novice programmers want to learn programming by themselves, since the community support and knowledge repository is stronger.

Online and Offline Capability

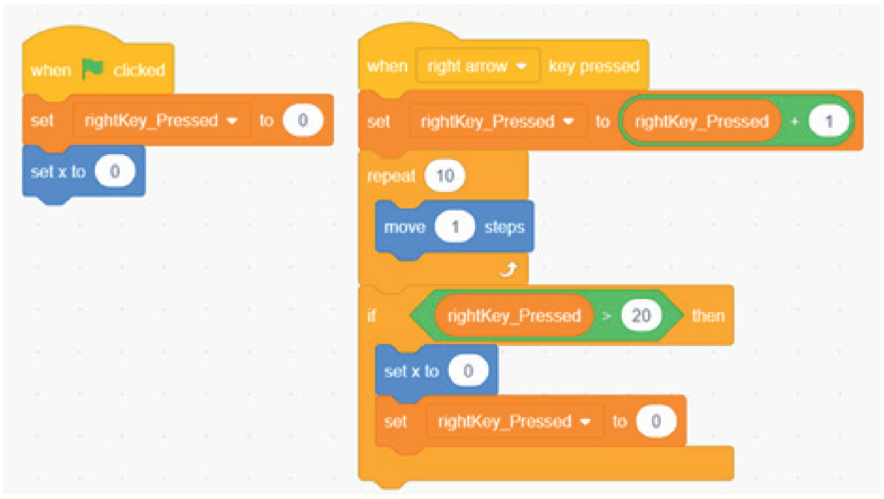
Another aspect to be compared among three environments is online/offline development capabilities of development environments. Having the capability of both offline and online development could be an advantage for different settings and contexts. When the three visual programing environments was compared, online/offline development capabilities of each environment was different than others. App Inventor is an online-only development environment, therefore learners could not reach the development environment without an Internet connection. Users can develop from any operating system through a modern web browser. Additionally, it was seen that due to Server-based live testing, projects were getting slower as the applications are getting more complex.

On the other hand, Kodu has an offline-only development environment. Users need to download and install the setup file to their Windows computers. This restricts users to develop from an online interface. Even though, users can reach other projects through the website of Kodu, they would need to download and install the Kodu setup file, not only for editing the project but also for viewing the game.

The most advantageous one among three was Scratch. Since Scratch has both online and offline development interface, users can develop games or animations through online or offline editor. Users also could view or play the games or animations through the online or offline interface. Users can also share their games or animations as they do in a social media website, because Scratch has a social computing network for sharing projects (Meerbaum-Salant, Armoni, & Ben-Ari, 2013).

Use of Programming Concepts

An important part of a programming environment for novice programmers could be seen as the ability to use and potential to teach essential computer programming concepts. While App Inventor and Scratch uses the same block-based programming interface, Kodu uses a different, tile-based, programming language. Since it involves a variable, a conditional, and a loop, a basic moving character example made with Scratch can be seen in figure 1 below. In the example below a new variable was defined, after a key event was triggered a sprite was moved 10 times to right with a loop. Animation was reset after the conditional statement has met. It showed that Scratch has an easy programming interface with fairly enough programming concepts to be used and learned. A similar example has been tried to be made to compare with the capability of other environments. Examination of programming concepts in Scratch showed that basic programming concepts such as variables, conditional statements, loops etc. can be used effectively. Scratch simplified the essential programming concepts with a colorful interface to provide scaffolding about the categorized differences between them. However, studies showed that it does not necessarily mean that all programming concepts would be fully understood by the students. Meerbaum-Salant, Armoni and Ben-Ari (2013) stated that while the post-test scores of students regarding conditional loops (75%), bounded loops (57.5%), and message passing (62.5%) are acceptable, concepts of variables (7.5%), initialization (17.5%), and concurrency (7.5%) scores are very low which need explicit instruction.

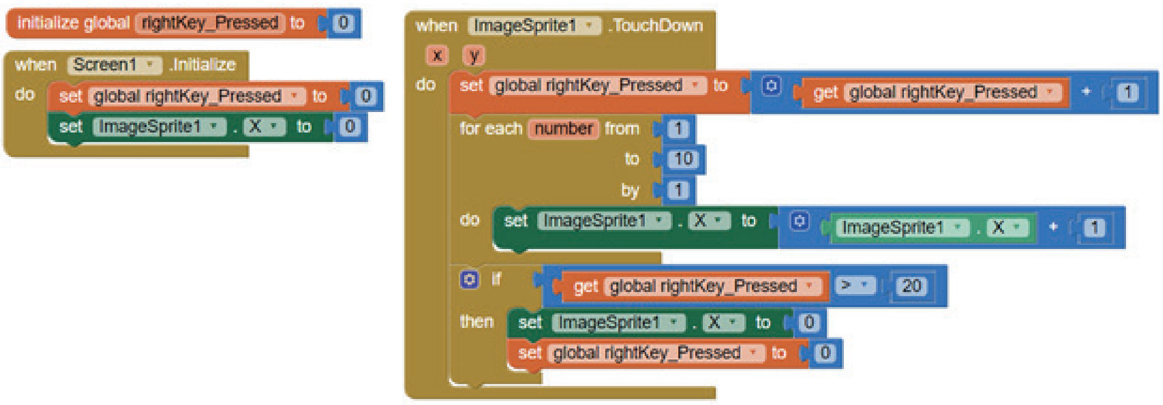


```
when ⚑ clicked
set rightKey_Pressed to 0
set x to 0

when right arrow key pressed
set rightKey_Pressed to rightKey_Pressed + 1
repeat 10
    move 1 steps
if rightKey_Pressed > 20 then
    set x to 0
    set rightKey_Pressed to 0
```

Figure 1. A sample block programming page from Scratch

A similar example prepared with App Inventor which can be seen in Figure 2 below. Since App Inventor uses the same library with Scratch the outcome is very resembling. However, App Inventor needs users to add a canvas and an empty imagesprite to the screen. Additionally, App Inventor has a bit more complex programming logic which resembles traditional programming languages. In App Inventor, there are not much ready-to-use media as Scratch offers to its users. Users need to import any media they need and program to the App Inventor for using them in their application. Same rule applies for the coding part. Each variable needs to be defined, and each event movement need to be made through the mathematical coordinates. It could be said that App Inventor offers a more complex programming palette to use, however, it could also be seen more complicated than other two environments by novice programmers. While Scratch and Kodu could be learned through exploration, App Inventor would need more effort and explanation to be understood.



```
initialize global rightKey_Pressed to 0

when Screen1 .Initialize
do  set global rightKey_Pressed to 0
    set ImageSprite1 . X to 0

when ImageSprite1 .TouchDown
  x  y
do  set global rightKey_Pressed to get global rightKey_Pressed + 1
    for each number from 1
                     to 10
                     by 1
    do  set ImageSprite1 . X to ImageSprite1 . X + 1
    if  get global rightKey_Pressed > 20
    then set ImageSprite1 . X to 0
         set global rightKey_Pressed to 0
```

Figure 2. A sample block programming page from App Inventor

A similar example was made with Kodu, however due to the different nature of Kodu, it was not fully succeeded. In the example seen below, first tile step was used for movement of the character added to the 3D environment. In Kodu, users are only able to use “Scores” as the variables. In the second step, variable was incremented by one in each time user press the right key on the keyboard. At the last step, a conditional statement was used in which when score variable equal to or greater than 10, game ends. Even though the aim was to create a similar program, Kodu restricts us to apply the exact same programming logic. It uses timers as loops, however we could not connect it to change the coordinate of the character, since it uses paths rather than the coordinates.

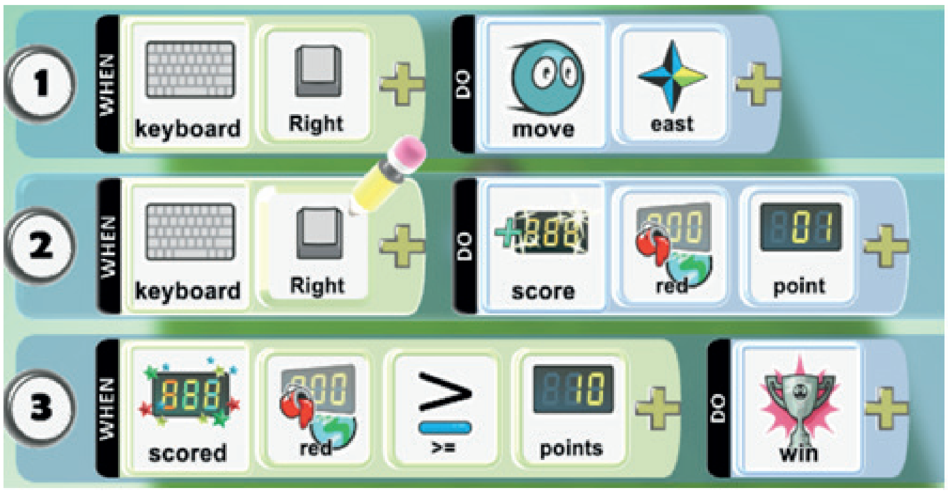


Figure 3. A sample tile programming page from Kodu

Stolee and Fristoe (2011) stated that Kodu has many computer science concepts can be taught by through using Kodu's entirely event-driven programming logic. "Students compose their programs, organized as sequences of WHEN/ DO rules, by selecting icons from a context-sensitive menu, making syntax errors impossible. Rather than using named variables, Kodu uses scores, such as the "red score" and the "blue score," which children intuitively understand from playing computer games." (Touretzky, Marghitu, Ludi, Bernstein, & Ni, 2013, p. 610). Kodu uses score as the only variable which is more challenging to teach variable concept in programming. Stolee and Fristoe (2011) interpreted the assigning and changing different attributes of an object as a local variable, and elements like score interpreted as global variable in Kodu. Even though variables could be interpreted as such by a programmer, the information was not explicitly presented by the environment. Touretzky et al. (2013) also stated that the only conditional statement (When/ Do) in Kodu environment could not replace and provide the logic of the conditional If/Then statement. The best way to provide information about computer science concepts is to explain and make the connection about them.

Kodu might be relevant as a first step environment to change the attitude of elementary student towards programming positively. Since it does not provide a similar implementation of a traditional programming language, it would not be helpful to teach all of the programming concepts through a visual programming language.

In addition to programming concepts involved programming architecture of the environments could also affect the learning and the products of the learners. As it was mentioned before, App Inventor and Scratch uses a block-based library, and their language offers a simple but flexible language for the users. However, Kodu's tile-based architecture restricts the users to assign functions through events by selecting the relevant tile sequentially. After one step completed, user need to align the next step for the selected object or event. Even though this could be stated as a similarity with object-oriented programming, it limits the flexibility of the desired product.

Product Range

App Inventor is the most flexible one among three environments in terms of products. In App Inventor users can develop applications ranging from (nearly) any type of Android OS application, to games, or simply animations. It provides access to phone sensors (orientation, proximity, gyroscope etc.) as well as extensions to be used for Lego Mindstorms. The main limitation in terms of product range for App Inventor is that products are only for devices with Android OS. It can be tested in computers through an emulator but the usable products are for Android OS only.

On the other hand, products of Scratch are usable for both Windows, MacOS and mobile operating systems if designed relevantly for mobile devices. However, the products are limited to 2D games and animations. It also supports extensions such as Lego Mindstorms.

However, with Kodu, users can only develop applications for Kodu. Since Kodu is only available for Windows, and the products are only usable with Microsoft's operating system. It limits the potential of reaching to learners who do not have a Windows PC. Kodu's products are also limited just with 3D games. Having only 3D games restricts users to develop with only provided 3D objects, while others uses image sprites that could be replaced or created by the users.

Target Group

In Kodu's official website (www.kodugamelab.com), Kodu was explained as a visual programming language that lets "kids" create games. "Kodu has an uncomplicated visual approach and is easy to master, even at the earliest stages of grammar school" (Nygård, Kolås, & Sigurdardottir, 2016, p. 416). Kodu's interface and general design is also found to be more relevant for elementary school students after examination for this study, because it has a simpler interface, and limited capability to advance.

Scratch's simple yet powerful interface could be relevant for a broader audience than Kodu's. Studies show that it is suitable for both elementary, and middle school students (Meerbaum-Salant et al., 2013). Scratch's website (Scratch, 2019) shows that statistically, age of their users are ranged from 8-18 (Figure 4). Even though there are users that registered at different ages, Scratch's interface and products are more relevant for children who are between 8-16 years old. Peak of the graphic consist of the 12 years old registered users which could be seen as an important indicator of their target group

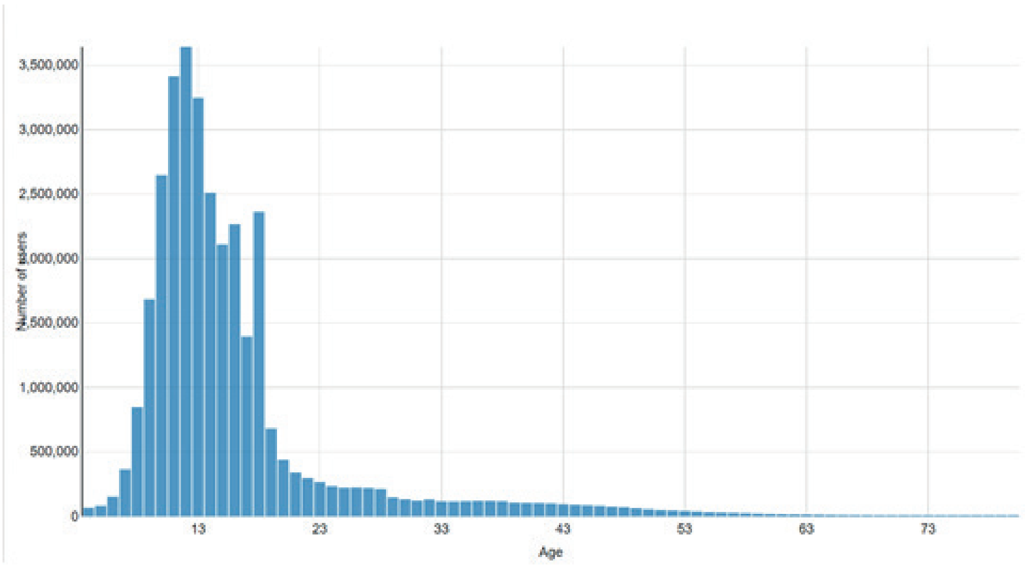


Figure 4. Ages of Registered Users (Scratch, 2019)

App Inventor has a more diverse target group since it was more flexible in terms of the interface and the product range. However, for younger audience the interface could be a bit more confusing. Additionally, the interface was not designed solely for the children. There are a number of studies that used App Inventor used from K-12 to Graduate school (Hsu & Ching, 2013; Morelli et al. 2011; Wagner, Gray, Corley, & Wolber, 2013), however, our examination showed that App Inventor would be more relevant for learner in middle school or above.

4. Implications and Future Studies

Rather than selecting an environment and hoping for the best, providing proper guidance and explanation could be crucial for enhancing the learning of non-programmers for essential programming concepts, especially the complex ones (Denner, Werner, and Ortiz, 2012). Our examination showed that three popular visual programming environments are separated based on some characteristics and features (Table 1). Instructors should be aware that advantages and disadvantages, and relevance of an environment for the target audience.

Table 1. Comparison of the features of Visual Programming Environments

Features	App Inventor	Scratch	Kodu
Online Development	Yes	Yes	No
Offline Development	No	Yes	Yes
Teaching Programming Concepts	High-Complex	Moderate-Essentials	Low-Partial
Programming Architecture	Block-Based	Block-Based	Tile-Based
Programming Structure	Drag and Drop Puzzle Pieces	Drag and Drop Puzzle Pieces	Selectable Stepwise Functions
Extension Support	Yes	Yes	No

OS Support (Products)	Android OS	MacOS, Windows, Linux, and mobile devices	Windows
Sharing Projects	Yes	Yes	Yes
Reaching Source Code	Yes	Yes	Yes (w/ Download)
Target Group	K-12 and Higher	K-8	K-5
Products	Various (exc. 3D)	2D Animation and Games	3D Games
Discussion Forum	Active	Very Active	Not Active
Language Support	12	40	24

This study put forward some similarities and differences of the visual programming environments. Study has shown that each environment has an advantage of its own. While Kodu could be useful for causing a positive attitude change towards programming by providing very simple yet very interactive products to the children, Scratch and App Inventor could be better as a step for textual programming languages. Hsu, Lou, and Sun (2016) also listed some of the limitations of displaying blocks in visual programming languages (1) the readability, (2) the program structure, and (3) the re-use. Using additional tools to support visual programming can be used to overcome these limitations.

For the future studies an experimental comparison of the three environments could be useful. Additionally, a practical implementation following this study could be providing textual programming examples that was provide in the visual programming environments simultaneously. "Highly scaffolded programming environments offer novices a smoother path to early success in computing, but their limited expressiveness must inevitably lead to their abandonment in favor of more powerful conventional languages." (Touretzky et al., 2013, p. 609). In future studies, visual programming environments could be tested to make a connection with textual programming to see their suitability as a step for advanced programming languages. Using CS-Unplugged (Computer Science-Unplugged) activities to support visual programming languages could be one of the useful strategies to enhance the programming understanding of the students as Touretzky et al. (2013) suggested. Effectiveness of visual programming environments could be enhanced with on-paper activities provided by the instructor.

5. References

Aggarwal, A., Touretsky, D. S., & Gardner-McCune, C. (2018). Demonstrating the Ability of Elementary School Students to Reason about Programs. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 735-740). ACM.

Akcaoglu, M. (2014). Learning problem-solving through making games at the game design and learning summer program. Educational Technology Research and Development, 62(5), 583–600. http://doi.org/10.1007/s11423-014-9347-4

Arilesere, F. O. (2014). Kodu game lab - a tool for ensuring quality teaching-learning for pupils in primary schools: case study (school in Northern Finland) (Master's Thesis). Retrieved from Networked Digital Library of Theses & Dissertations. University of Oulu, Finland.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" Programming. ACM Transactions on Computing Education, 14(4), 1–15. https://doi.org/10.1145/2677087

Alzahrani, M. G. (2017). The Effect of Using Online Discussion Forums on Students' Learning. Turkish Online Journal of Educational Technology-TOJET, 16(1), 164-176.

Bennedsen, J., & Caspersen, M. E. (2012). Persistence of Elementary Programming Skills. Computer Science Education, 22(2), 81-107.

Berg, B. L. (2001). Qualitative research methods for the social sciences. Qualitative Research (Vol. Seventh Ed). https://doi.org/10.2307/1317652

Bertea, A. F. (2011). Mobile Learning Applications Using Google App Inventor for Android. The International Scientific Conference eLearning and Software for Education. Bucharest.

Coy, S. (2013). Kodu game lab, a few lessons learned. XRDS: Crossroads, The ACM Magazine for Students, 19(4), 44. http://doi.org/10.1145/2460436.2460450

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? Computers and Education, 58(1), 240–249. https://doi.org/10.1016/j.compedu.2011.08.006

Deek, F. P., & McHugh, J. A. (1998). A survey and critical analysis of tools for learning programming. Computer Science Education, 8(2), 130–178.

Dekhane, S., Xu, X., & Tsoi, M. Y. (2013). Mobile app development to increase student engagement and problem solving skills. Journal of Information Systems Education, 24(4), 299–308.

Felleisen, M., Findler, R. B., Flatt, M., & Krishnamurthi, S. (2004). The TeachScheme! Project: Computing and Programming for Every Student. Computer Science Education, 14(1), 55–77.

Fowler, A., Fristce, T., & Maclauren, M. (2012). Kodu Game Lab: a programming environment. The Computer Games Journal, 1(1), 17–28. Retrieved from www.computergamesjournal.com

Google, (2018). Introduction to blockly. Retrieved March 3, 2019 from https://developers.google.com/blockly/guides/overview

Hsieh, H.F., & Shannon, S. E. (2005). Three Approaches to Qualitative Content Analysis. Qualitative Health Research, 15(9), 1277-1288. http://doi.org/10.1177/1049732305276687

Hickmott, D., & Prieto-Rodriguez, E. (2018). To Assess or Not to Assess: Tensions Negotiated in Six Years of Teaching Teachers about Computational Thinking. Informatics in Education, 17(2), 229-244.

Hsu, Y. C., & Ching, Y.-H. (2013). Mobile App Design for Teaching and Learning: Educators' Experiences in an Online Graduate Course. The International Review of Research in Open and Distance Learning, 14(4), 117-139.

Kelly, J. F. (2013). *Kodu for Kids: The Official Guide to Creating Your Own Video Games*. Que Publishing.

Kwon, D., Yoon, I., & Lee, W. (2011). Design of Programming Learning Process using Hybrid Programming Environment for Computing Education. KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS, 5(10), 1799-1812.

Lee, Y. J. (2011). Scratch: Multimedia Programming Environment for Young Gifted Learners. Gifted Child Today, 34(2), 26-31.

Liu, E. Z. F., Cheng, S. S., & Lin, C. H. (2013). The effects of using online q&a discussion forums with different characteristics as a learning resource. The Asia-Pacific Education Researcher, 22(4), 667-675.

Malan, D. J., & Leitner, H. H. (2007). Scratch for Budding Computer Scientists. SIGCSE (pp. 223-227). Kentucky: ACM.

Maloney, J., Peppler, K., Kafai, Y. B., Resnick, M., & Rusk, N. (2008). Programming by Choice: Urban Youth Learning Programming with Scratch. In SIGCSE '08 (pp. 367–371). Portland, Oregon, USA.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. ACM Transaction on Computing Education, 10(4), 16:1-16:15.

Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. Computer Science Education, 16(3), 211–227.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning Computer Science Concepts with Scratch. Computer Science Education, 23(3), 239-264.

MIT App Inventor, (2019). About us. Retrieved March 5, 2019 from http://appinventor.mit.edu/explore/about-us.html

Morelli, R., de Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E., & Uche, C. (2011). Can Android App Inventor Bring Computational Thinking to K-12? In SIGCSE'11. Dallas, Texas, USA.

Mueller, J., Beckett, D., Hennessey, E., & Shodiev, H. (2017). Assessing Computational Thinking Across the Curriculum. In Emerging Research, Practice, and Policy on Computational Thinking (pp. 251–267). Cham: Springer International Publishing.

Nygård, S., Kolås, L., & Sigurdardottir, H. (2016). Teachers' Experiences Using KODU as a Teaching Tool. Proceedings of the European Conference on Information Management & Evaluation, 416–422.

Pokress, S. C., & Veiga, J. D. (2013). MIT App Inventor: Enabling Personal Mobile Computing. ACM.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Kafai, Y. (2009). Scratch: Programming for All. Communications of the ACM, 52(11), 60-67.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teachin Programming: A Review and Discussion. Computer Science Education, 13(2), 137-172.

Sadik, O., Leftwich, A.-O., & Nadiruzzaman, H. (2017). Computational Thinking Conceptions and Misconceptions: Progression of Preservice Teacher Thinking During Computer Science Lesson Planning. In Emerging Research, Practice, and Policy on Computational Thinking (pp. 221–238). Cham: Springer International Publishing.

Sandoval-Reyes, S., Galicia-Galicia, P., & Gutierrez-Sanchez, I. (2011). Visual Learning Environments for Computer Programming. Electronics, Robotics, and Automotive Mechanics Conference (pp. 439-444). Cuernavaca, Morelos: IEEE Computer Society.

Scratch, (2019). Age Distribution of New Scratchers. Retrieved March 5, 2019 from https://scratch.mit.edu/statistics/

Smutny, P. (2011). Visual Programming for Smartphones. International Carpathian Control Conference, (pp. 358 - 361). Velke Karlovice.

Sorva, J., Lönnberg, J., & Malmi, L. (2013). Students' Ways of Experiencing Visual Program Simulation. Computer Science Education, 23(3), 207-238.

Stemler, Steve (2001). An overview of content analysis. Practical Assessment, Research & Evaluation, 7(17).

Stolee, K. T., & Fristoe, T. (2011). Expressing computer science concepts through Kodu game lab. In Proceedings of the 42nd ACM technical symposium on Computer science education (pp. 99-104). ACM.

Su, A. Y., Yang, S. J., Hwang, W.-Y., Huang, C. S., & Tern, M.-Y. (2013). Investigating the role of computer-supported annotation in problem-solving-based teaching: An empirical study of a Scratch programming pedagogy. British Journal of Educational Technology.

Tangney, B., Oldham, E., Conneely, C., Barrett, S., & Lawlor, J. (2010). Pedagogy and Process for a Computer Programming Outreach Workshop - The Bridge to College Model. IEEE Transactions on Education, 53(1), 53-60.

Wagner, A., Gray, J., Corley, J., & Wolber, D. (2013). Using app inventor in a K-12 summer camp. Proceeding of the 44th ACM Technical Symposium on Computer Science Education - SIGCSE ’13, 621. https://doi.org/10.1145/2445196.2445377

Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program. In Proceedings of the 2005 international workshop on Computing education research - ICER ’05 (pp. 13–24). New York, New York, USA: ACM Press.

Wilson, A., Hainey, T., & Connoly, T. (2012). Evaluation of Computer Games Developed by Primary School Children to Gauge Understanding of Programming Concepts. Proceedings of the European Conference on Games Based Learning, (pp. 549-558). Cork.

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. ACM Transactions on Computing Education (TOCE), 18(1), 3.

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2017). Expanding computer science education in schools: Understanding teacher experiences and challenges. Computer Science Education, 26(4), 235–254. https://doi.org/10.1080/08993408.2016.1257418