We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

186,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter 9

An Evaluation of Open Digital Gaming Platforms for Developing Computational Thinking Skills

Andoni Eguíluz, Pablo Garaizar and Mariluz Guenaga

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.71339

Abstract

Due to business needs and the growing importance of technology in society, in recent years, the concept of computational thinking has emerged, especially focused on its inclusion in compulsory education as a relevant complement, transversal to traditional subjects. In parallel, various initiatives have developed interactive digital tools for learners to meet this type of thinking through a series of activities commonly framed as games. In this chapter, we evaluate many of the existing free access platforms to propose pedagogical, design, and content approaches with which they can be compared.

Keywords: computational thinking, learning, resources, school, videogames, visual languages

1. Introduction

Computational thinking (CT) allows you to solve problems in a way that a computer (human or machine) can execute. In just a decade, since Wing [1] introduced the concept to the technological community, the movement has been very intense, both in the scientific community and the educational world, as well as in the tools and content available [2]. Thanks to all these efforts, we begin to see signs that the situation is changing, but there are many challenges [3], starting with the still insufficient definition of the concept of CT and its structuring in the classroom [4, 5], which involves multiple initiatives by national and international organizations.

Considering the ubiquity of computers in our digital society, CT appears to be a fundamental skill not only for computer specialists but also for many other professionals. It is still a topic of debate how important and transversal CT should be in compulsory education [5]; however,

INTECH
open science | open minds

© 2018 The Author(s). Licensee InTech. This chapter is distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. CC BY

for practical reasons, the educational world has been paying it increasingly more attention. In recent years, many initiatives have been developed to promote CT in primary and secondary education due to both the social boom in CT and the lack of computer professionals in the present and near future. The Hour of Code, Code Week, or Scratch Day are some of the well-known global events that promote changes in curricular design toward this new digital formation. Most of these initiatives are based on digital platforms where learner programmers can develop and improve their CT skills through games.

It is not this chapter's goal to discuss the advantages of incorporating CT into education, but to analyze the tools, their possible uses, and their limitations.

To delimit the study, we see that there are many common features among most of the current tools geared to developing CT skills like code.org, Alice, Scratch, or Kodable: they are open and free for general use; they focus mainly on primary and secondary school students; and there is an explicit gaming characteristic, which facilitates the use by these learners and applies the proven benefits of games in education [6].

It is also important that all these tools seek to avoid novice programmers having to confront the complexity of text-based computer coding and to improve the learnability [7]. There are several ways to address this problem such as narrative tools, flow-model tools, or specialized output realizations [8]; in this chapter, we focus on the most common tools, which are those that use the block-based visual programming. These tools employ user interfaces based on visual blocks that are moved and placed constructively as an assembly game, usually with the visual abstraction of a puzzle with its pieces and fitting ways. These blocks work as an abstraction of programming components: sentences, data, control structures, procedures, and so on. Consequently, they considerably limit the prior knowledge required to program and reinforce the program structure, eliminating the possibility of syntax errors and focusing only on the logic that exists in the activity that is to be undertaken.

In this chapter, we review a number of existing platforms with features mentioned above. There are articles which discuss some of them [9–11]; our intention is to propose an objective analysis, reviewing their possibilities from different perspectives. From a pedagogical point of view, we study different dimensions that can affect the learning process in or out of class (such as the richness of the proposed interaction, the time that can be invested, and the depth of the exploration). From the point of view of the game, the fun and engagement generated. From a CT point of view, we will analyze what concepts involved in CT each platform covers and to what extent. Finally, we will analyze the degree of adaptation to the personal characteristics of programming learners, their skills, and knowledge. We will consider, among others, aspects such as feedback and interaction, registration possibilities, and learning design.

2. Review of platforms

In our study, we include those platforms accessible online in July 2017 that can be used through the computer, mobile, or tablet without a paid commercial license that do not require additional hardware or devices, that are at least available in the English language, with international and

open distribution, that contain some relevant part of visual programming and that incorporate some game characteristics. In this section, we describe them in an alphabetical order and then analyze them according to their features in the comparison.

2.1. Alice (http://www.alice.org/)

Alice is a free creation tool for small games, animations, and interactive resources in 3D. It is also an interactive animated story development tool that proposes a visual programming interface strongly linked to the development of object-oriented code, with explicit concepts such as methods, event listeners, objects like characters, or classes such as scenes. In its version 3, it also allows a parallel development in which each block movement displays the corresponding Java source code. It is one of the few 3D–oriented environments, making it possible to create games and animations, while adding a degree of complexity to the need to elaborate the scene with its models, skeletons, terrains, or cameras. Its alliance with Disney, Pixar, and EA has facilitated the inclusion of elaborated graphics, which enrich the experience of the project. It was created by the University of Virginia and taken over and maintained by Carnegie Mellon University since 1997.

2.2. App Inventor (http://appinventor.mit.edu/)

App Inventor is a free creation tool for small games, animations, and interactive resources. This tool has a closer approach to professional development than most platforms analyzed and it is probably the least close to a game because its goal is to build apps for an Android device. Users visually design the graphic user interface (GUI) on the one hand and its logical operation on the other, simulated in execution on the device itself or in an Android emulator included. System and development dependency for mobile devices causes event orientation (sensors, button clicks, etc.) to be very present in App Inventor and determines the control flow. It is also important to design the user interface, which has a specific section of the tool (Designer) in addition to code editing (Blocks). It was originally created by Google in 2010 and taken over by MIT in 2012 and oriented to youth and adult audience, not to children.

2.3. Bee-Bot (https://itunes.apple.com/es/app/bee-bot/id500131639?mt=8)

Bee-Bot is a game of successive challenges to learn programming for iPad. Bee-Bot is a bee robot, a physical toy for children of 4 years and above, which also has a free App for iPad that can be used without the toy, geared to preliterate children. It allows children to learn programming concepts about directional motion primitives, while taking their virtual robot through a series of scenarios. It has 12 levels with labyrinths of progressive difficulty. There is also a more complex paid app for children of 7 years and above.

2.4. Beetle Blocks (http://beetleblocks.com/)

Beetle Blocks is a free 3D model creation tool. It is a visual programming environment that allows for modeling three-dimensional shapes. Like a Logo language with its 3D tortoise, you can program the movement of a virtual beetle to generate geometric shapes of all kinds that

are chained with repetitive and alternative control structures and programmable routines. The block language is based on Snap!, which is analyzed in Section 2.20.

2.5. Blockly Games (https://blockly-games.appspot.com/)

Blockly Games is a game of successive challenges to learn programming based on a generalized visual programming library. Blockly (https://developers.google.com/blockly/), developed by Google as open source in 2012, is really a library to make Web applications based on visual programming by blocks (hence its name). Starting from this, other tools have been made such as App Inventor, Microsoft MakeCode, or OzoBlockly. Here we analyze Blockly Games, which is a Blockly-based Google development, proposing a series of educational games to learn how to program in a directed and progressive mode, in a series of levels that are structured into seven sections: Puzzle, Maze (movement to exit a maze with repetitive and alternate structures), bird (continuous 2d motion with twists and x-y displacement based on conditions and boolean expressions), turtle (logo style drawing with repetition over angles and distances), movie (movement of geometric shapes based on a time value that ranges from 0 to 100), pond tutor (a shooting game with angles and forces that introduces text programming corresponding to the visual), and pond (a shooting game with players to be controlled by the computer). It also incorporates two free challenges that can be published on Reddit.

Blockly's development technology is web-oriented (in Javascript), but it also has native support for Android and IOS.

2.6. Cargo-Bot (https://itunes.apple.com/es/app/cargo-bot/id519690804)

Cargo-Bot is a game of successive challenges to learn programming. It is also a game for iPad where children can teach a robot to move boxes inside a factory by using visual programming structures. The game contains 36 puzzles and has the peculiar feature of having been the first game entirely developed on the iPad itself based on Codea (a Lua code editor for iPad). It uses encoding with icons with motion primitives for the crane (down, up, and move box) and calls to repeat procedures as well as box color codes for alternative actions.

2.7. Code.org (https://code.org/) and Code Studio (https://code.org/educate/gamelab)

Code.org is a nonprofit organization created by Harvard computer scientist Hadi Partovi in 2013 having diverse games of successive challenges to learn programming, and tool of creation free of small game, animation, or interactive resource. He has been able to intensively energize the need to bring CT to young people through the activities promoted from his website and, in particular, the creation of a movement called "Hour of Code" that proposes simple digital activities in the form of programming challenges that can be achieved in one hour (or a few more hours) with Blockly-based visual programming. By attracting personalities of the technological world to the promotion, like Mark Zuckerberg and Bill Gates, and later President Barack Obama himself, the movement has been a success not only in the USA but at the global level, managing to mobilize more than 100 million children and adults who use

http://dx.doi.org/10.5772/intechopen.71339

their resources for free to learn to code. Another of the successes of Code.org has been to use elements of existing video games or films to contextualize the challenges of code; and negotiating with corporations to use characters from Plants vs. Zombies, Flappy Bird, Frozen, Star Wars, or Minecraft, which makes the project much more attractive, as well as an important graphic, design, and logic quality. After 4 years of Hour of Code editions, Code.org and its partners have more than 170 different activities identified and accessible on the web (https://code.org/learn).

In addition to the online programming activities, Code.org also deals with the full computing curriculum content in primary and secondary education, provides training materials and resources of different types for free use of schools, and promotes training activities for schoolchildren and teachers (with special attention to women and underrepresented minorities). It has developed a blocks-based visual language, Code Studio (https://studio.code.org), that, Scratch-style, makes it possible to create multimedia resources or video games. This tool distinguishes four types of projects: Draw Something, App Lab (to simulate mobile apps), Play Lab (games or simple stories), and Game Lab (more elaborate games).

All the technology developed by Code.org is open source. The web also incorporates a section for teachers with management features for student groups and a complete dashboard page with detailed control of the development of each student.

2.8. Daisy the Dinosaur (http://www.daisythedinosaur.com/)

Daisy the Dinosaur is a game of successive challenges to learn programming. It is a small game for the iPad aimed at the first contact with programming for the youngest children, who can solve it in a few minutes. Its concept is very simple: the learner moves Daisy the Dinosaur by using some of the nine motion commands in the appropriate sequence. When the challenges are over, you can play in free-play mode to freely move the character.

2.9. Kodable (https://www.kodable.com/)

Kodable is a game of successive challenges to learn programming. Kodable is one of tools with the strongest focus on teachers and classes. It envisages a progression of levels oriented to the whole age of primary education. It has a number of free levels and many more accessible through pay-by-school licenses. The approach is an icon-based visual programming language that allows you to move a character (Fuzz) through an orthogonal maze by collecting the stars. To the motion instructions in all four directions, conditionals are added using colored boxes, loops with counters, and functions to repeat the same code several times. Another series of worlds provide conceptual training complements in programming using games, although they do not use visual programming such as a Tetris-style game for strings and integers, a tower defense for object-orientation, and so on. Teachers have a dashboard with full access to the progress information of their classes and students. In addition, there is a creation mode to generate custom levels, which both teachers and students have access to.

2.10. Kodetu (http://kodetu.org/)

Kodetu is a game of successive challenges to learn programming. It is a maze-solving game that allows an astronaut to be guided to the target in a space station by using visual blocks of motion and rotation, and repetitive and alternative structures. It is based on Blockly Maze and proposes a sequence of 15 levels of progressive difficulty, being the last maze a challenge even for people who already know how to program. It is designed to be able to play in an hour or two. Teachers can generate their own groups and access the information about the learning path covered by their students.

2.11. Kodu Game Lab (http://www.kodugamelab.com/)

Kodu Game Lab is a free small 3D game creation tool. Kodu, originally called Boku, is a visual programming environment developed by Microsoft in 2009, for the Xbox console and Windows OS. Like Alice, the execution environment is in 3D, but the programming orientation is quite different and is not block- but event-oriented. The concept is quite original with respect to other tools: the user can modify the world with a visual editing system of ground and 3D objects and add characters on which rules are created (in when-do form, with what is called in Microsoft Tile-Based Coding): If an event occurs, an action is executed. The events and rules are very oriented to an arcade type game (move, shoot, and collide) in a fixed gravity context. It does not incorporate control structures because the event system itself generates an infinite repetition in a real-time loop, and each rule (when) functions as an alternative.

2.12. Hopscotch (https://www.gethopscotch.com/)

Hopscotch is a free creation tool for small games, animations, or interactive resources. Hopscotch, available only for iOS, is a visual programming language specifically designed to be used on Apple touch devices and is oriented to very simple games. Although the payment license allows you to personalize characters and other improvements, in the free mode you can develop the entire gameplay. Hopscotch also has an online community to share creations, and a web player so that anyone can play the games created from a browser. Hopscotch makes a significant effort to use the resources of the tablet, both to ensure that the entire editing system can be done in a tactile way and to incorporate all the possibilities of the device in programming (tilt, vibration, and acoustic sensors).

2.13. Lightbot (https://lightbot.com/)

Lightbot is a game of successive challenges to learn programming. It is a commercial game for mobile platform (also with a version for Windows and Mac), with a free demo version (code hour) that allows users to play the first levels. The approach of the game modernizes the classic movement puzzles like Robozzle (explained later), with successive challenges of small labyrinths in which you have to illuminate the blue squares with the only instructions to advance, turn, jump, and ignite. To repeat, you can define several subprograms and make recursive calls. The game proposes a staggering of several levels of complexity and has two apps differentiated by difficulty for the age bands of 4–8 and 9 + .

2.14. Made with Code (https://www.madewithcode.com/)

Made with Code is a game of successive challenges to learn programming. It is a Google initiative created in 2014 to encourage school-age girls to develop their first experiences in CT. It includes a lot of educational materials with textual and audiovisual contents, and proposals of projects and activities by using various tools. As far as our analysis is concerned, it has a specific area of visual programming projects (https://www.madewithcode.com/projects/) proposing a series of short activities with Blockly-based visual programming. Some of these activities are programming challenges with basic concepts (like the ones based on *Inside Out* or *Wonder Woman*), and there are more open and creative ones that simply seek to provide tools for girls to propose their own elements based on computational abstractions such as designing a costume pattern (*LED dress*), playing a musical rhythm (*beats*) or creating a visual message based on its components (*code for equality*), with a clear intention to show how many everyday activities and objects have computational components in their design, construction, or use.

2.15. MakeWorld (https://makeworld.eu/)

MakeWorld is a context and path-free creation tool to define programming challenges related to other STEM areas. MakeWorld is one of the few platforms created in Europe, in the framework of a European Union innovation project. It targets primary school children for a first-learning CT environment. Therefore, it minimizes textual aspects and proposes an icon-based action interface, with two levels: worlds (a programming challenge) and stories (a set of challenges linked to a learning sequence). Through these, programming challenges are sought to work concepts of other subjects (mathematics, languages, science …) where they include computational elements such as enumeration, sequencing, identification, classification, cycles, processes, systems, and so on. The concepts of CT are limited to the most basic context: movements in a grid, repetitions through recursive subprograms, and scoring events to manage progress. MakeWorld allows both solving worlds by posing a game with goals and creating worlds (hence its name) to go a step further in the level of abstraction. These worlds can be published and shared in the social community.

2.16. Minecraft (https://education.minecraft.net/)

Minecraft is a free construction game. It is a game based on three-dimensional blocks that allow users to create constructions in a free world, with the intense expression of the creativity of the user. Although its initial objective was only to be a constructionist game, it has evolved and allows users to incorporate complex logics within the objects of the game, including from 2017 a visual programming language and environment, Code Builder (with equivalence in JavaScript). It can be programmed with the Microsoft Visual programming tool (MakeCode) or with the existing Scratch and Tynker tools and allows game elements to change and behave according to the programmed code. After Microsoft bought the original product, it has developed a whole line of education aimed at raising problems and challenges of computational thinking and allowing them to be shared among users in the community. Although Minecraft is a commercially licensed product, educational use has been dropping in price and can be used partially free.

2.17. Robozzle (http://www.robozzle.com/)

Robozzle is a game of successive challenges to learn programming. Robozzle, published in 2009, is, according to its creator Igor Ostrovsky, a "social puzzle game." It poses a very basic programming environment in a maze in which you have to capture the stars with the only instructions to move forward and rotate. To repeat, you can define several subprograms and make recursive calls, and for the concept of alternative you use up to three colors that determine whether the statement is executed or not. The result is a little game of challenges to solve the puzzles that each level poses. The game itself does not propose a staggering of levels of complexity (a priori there are only the basic difficulties of the small tutorial), but it leaves the creation of new challenges to the users themselves, and their evaluation by difficulty and taste, in a community that has created about 10,000 puzzles. It allows CT to be introduced with a simple game, without considering longer control structures or programs, with very few, primitive elements, close to the low level that is behind the programs.

2.18. Scratch (https://scratch.mit.edu/)

Scratch is a free creation tool for small games, animations, and interactive resources. Scratch is a visual programming language created by MIT's Lifelong Kindergarten Lab in 2002, with an editing and execution system in the cloud. With an important orientation to the user community and an open approach, it allows to share and derive programs from others. Due to its significant history and implementation, there are many tutorials for users, teachers, and parents. The structure of programming components differentiates elements such as control structures, events, operators, data, or sensors. The elements that are manipulated by Scratch are configurable images and sounds, allowing you to set up 2D animations and video games of a certain level of complexity.

Scratch was one of the first tools to be established and has therefore greatly influenced most of the ones listed in this chapter. It uses the visual puzzle metaphor for the programming pieces, where each block has a shape that can only be combined with other compatible blocks, and a color that determines the block type.

2.19. ScratchJr (https://www.scratchjr.org/)

ScratchJr is a free creation tool for small games, animations, and resources. It is a visual programming language developed as a derivation of Scratch by the same MIT department, aimed at younger users without reading skills. The interface concept changes (from vertical blocks to simplified horizontal blocks, reducing the number of blocks, and using icons instead of texts) and is aimed at mobile devices, being available for free for Android, IOS, and Chromebook. There is also a version launched in collaboration with PBS Kids, which uses characters from the animated series.

2.20. Snap! (http://snap.berkeley.edu/)

Snap! is a free creation tool for small games, animations, and interactive resources. Snap! is a visual programming language very similar to Scratch, inspired by it in its appearance and

http://dx.doi.org/10.5772/intechopen.71339

type of interaction, but with a series of improvements that make it interesting for a greater range of users: it runs in HTML and JavaScript so that it does not depend on Flash and does not limit the platforms that can be used, allows you to define custom blocks, manage multiuser sessions in real time, nested sprites, generate projects such as executables, undo option, top-level functions, and so on. However, there is a much smaller community of users and projects, and the documentation available for teachers is very inferior.

2.21. SpriteBox (http://spritebox.com/)

SpriteBox is a game of successive challenges to learn programming. It is a game developed for Code Hour by the LightBot company. It has a similar approach, but it raises the level of CT a little by incorporating loops rather than jumps to routines. It also includes a game element in proposing a platform game in which users have to overcome programming challenges that affect the game (create platforms, open gaps, and rebuild the stage). It takes approximately an hour and raises progressive difficulty levels in three consecutive worlds.

2.22. The Foos (http://thefoos.com)

The Foos is a game of successive challenges to learn programming. It is a tablet-oriented game for preschool and primary children (no need for reading). It is based on the idea of a platform game that, in addition to being played as a traditional game, allows the movement to be configured with a block code. Progressive leveling that introduces sequences, repetition, events, and alternatives and ends with possibilities of free play and creation of personalized levels.

The App, created by the Pasadena CodeSpark company in 2014, is commercial, but educational use is free and must be managed by a teacher who will set up the class, invitations, and devices. It has a specific version of code hour that is a subset of the whole game and can be played in a Web browser without installation.

2.23. Tynker (https://www.tynker.com)

Tynker is a free small game creation tool, which includes several successive challenge games to learn programming. It is a commercial project that has a series of free levels and also a school model that can be subscribed to without cost with a set of six phases (each composed of a series of progressive levels), and you can order additional paid phases. The visual environment has vertical blocks similar to Scratch or Code.org, with a lot of context variation and games to choose from. Our analysis deals with the free levels and a specific section defined for code hour, Tynker Hour of Code (https://www.tynker.com/hour-of-code) with a multitude of different levels. Tynker also has a free programming environment, which allows editing games with a Scratch-style editor, allowing for customizing both scenery and objects, as well as the codes that these objects use with all the available blocks, making it one of the most complete tools.

2.24. Waterbear (http://waterbearlang.com)

Waterbear is a free creation tool for small games, animations, and resources. Waterbear is a visual programming language created by Dethe Elza, a Canadian professional (in an open

source development environment), inspired by Scratch but with a language developed to be able to program in a visual way closer to textual programming languages, incorporating elements such as arrays, dates, or diversity of mathematical functions. There is no community of users, and the environment is very self-learning-oriented, practically without didactic material available.

2.25. Platforms not covered

There are many other services and products aimed at learning CT or some of its skills, which we have not considered in this study as they fail to meet the specified conditions. In the first place, there are a whole series of games that require the acquisition of physical devices. Based on classic toys, these include robots and similar devices. Through their connection to the physical world, they enrich the possibilities of previously analyzed tools, limited by a screen and the need of Internet connection. They are an important niche market for companies in the educational toys sector. This is the case of LEGO Mindstorms (https://www.lego.com/mindstorms/), which was already commercialized in 1998 as a result of the collaboration between MIT and the LEGO construction toys company, to incorporate new robotic parts (different types of engines and sensors) controllable by children, using a visual programming language that is installed on the computer or device and that allows the user to write a program that is transmitted to the construction. The FIRST LEGO League began in 1999: an annual international competition with scientific and technological challenges based on this game, with more than 200,000 schoolchildren participating.

In this same line of products, we also find many others that have been appearing the last decade, such as Bee-Bot (https://www.bee-bot.us/), BlocklyProp (https://www.parallax.com/product/program-blocklyprop), Cubelets (http://www.modrobotics.com/cubelets/), Dash the Robot (https://www.makewonder.com/), Edison (https://meetedison.com/), Lego WeDo (https://education.lego.com/en-us/elementary/shop/wedo-2), mBot (http://makeblock.com/), microbit (http://microbit.org/), OzoBlockly (http://ozoblockly.com/), Robbo (https://www.robbo.world/), Sphero (http://www.sphero.com/), and many others.

There are also some mixed physical/digital toys that include a simple but necessary physical part (usually pieces to be placed), connected in some way (Bluetooth) with a digital application that needs the "program" created in the physical world in order to beat the virtual challenge: what is called "tangible programming." This is the case of Puzzlets (http://www.digitaldreamlabs.com/), with several games aimed at primary education or KIBO (http://kinderlabrobotics.com/kibo/), commercialized in dozens of countries.

We should also mention an important category represented by GameMaker Studio (https://www.yoyogames.com/gamemaker), GameSalad (http://gamesalad.com/), Stencyl (http://www.stencyl.com/), Unity (https://unity3d.com/), or Unreal (https://www.unrealengine.com/). These are video game authoring tools which include some possibilities of visual programming. Some of them have been used to learn programming [12], but they are not aimed at learning programming as such, nor are they specifically aimed at children or young people, nor are they commonly used in this type of activity. However, they are not far from this area, and this is already happening with some initiatives such as the Stencyl Educational

Kit (http://www.stencyl.com/education/overview/), GameMaker for Education (https://www.yoyogames.com/education) or GameSalad for Education (http://edu.gamesalad.com/), in all cases with paid licenses.

It is also important to note that text-based programming learning environments continue to exist, as in the 1990s when programming began to be introduced in schools: Basic (such as http://smallbasic.com/) or Logo (e.g., MSW Logo: http://www.softronix.com/logo.html).

There is a growing set of activities for lower educational levels called "unplugged." For example, Computer Science Unplugged (http://csunplugged.org/), a collection of free activities that teaches CT through games and puzzles with cards, paper and pen, strings, and school or household materials, without considering technological tools. Other examples with a more commercial focus are Hello Ruby (http://www.helloruby.com/), Code Monkey Island (http://codemonkeyplanet.com/), CodeMaster and other CODE games (http://www.thinkfun.com/), or Robot Turtles (http://www.robotturtles.com/).

We also find on the market a series of games that do not fit properly with the programming model that is usually classified as visual programming, but which do use concepts of abstraction, algorithms, and resolution of problems that promote CT. This is the case, for example, of SpaceChem (http://www.zachtronics.com/spacechem/), which proposes a series of puzzles in the form of chemical elements that must be manufactured with a specific combination of pathways and operators on atoms and molecules.

Finally, there is another large group of tools like Code Combat (https://codecombat.com/), Code Hunt (https://www.codehunt.com/), CodeHS (https://codehs.com/), CodinGame (https://www.codingame.com/), Colobot (https://colobot.info/), Minetest (http://www.minetest.net/) or Swift Playgrounds (https://www.apple.com/swift/playgrounds/), games for learning programming with text-based languages like JavaScript, Java, Python or Swift, without considering visual programming. They are usually a widely used resource for older students who have spent a few years with visual programming tools.

3. Analysis of platforms

We have analyzed these 24 tools, which we extend to 26, as both Code.org and Tynker really encompass two different approaches that require independent measure. Here we present the information considered about these platforms.

3.1. Classification

The vast majority of the tools can be differentiated into two main groups. The first (46.2% of those analyzed) corresponds to a **set of programming challenges** (e.g., Code.org), in the form of predefined closed levels to solve, usually incorporating new programming structures and increasing the difficulty progressively. The usage sessions can be from a few minutes to a few weeks, where Code.org is making the most remarkable effort to design complete academic trajectories with different levels.

The second group (42.3%) proposes a **visual programming language** in itself (e.g., Scratch), with varying degrees of amplitude, with which the learner can develop his own programs, in principle in a much more open way, which can easily be embedded in a dynamic of project-based learning. Most (27.9%) languages are aimed at programming a 2D game, two are 3D, one is for 3D modeling, and another for mobile device programming. The game component in this group is not really given by the language but by the intention: you can make games but also animations or interactive stories, and really any computer application that the creativity of the user allows (limited by the language primitives, which are not general purpose).

It seems logical to think that with the languages we could define the tools of the first group: that is, with Scratch we could define a programming challenge (or thousands). However, the programming structures themselves are usually not included among the language primitives (i.e., in Scratch, you cannot program a game that raises a programming challenge except with a lot of effort and personalization). In addition, platforms that propose challenges can automatically detect the improvement in each level, give feedback in case of error, and offer the user navigation to the next. In the programming languages, challenges can be proposed, but evaluation and sequencing are foreign to the system. Therefore, there are two different types of tools, although it seems logical that the technologies will continue to approach the possibility of integrating both (as Code.org and Tynker do in a similar way).

At the moment, there are two tools among those analyzed (MakeWorld and Robozzle, 7.7%) that allow users to do both things: users can solve challenges already posed and can also create new coding challenges and publish them for other users to solve. We could then talk about a third set of tools for **creating and solving programming challenges**.

A fourth and last category, represented by Minecraft (3.8%), is that of a **videogame incorporating visual programming in its mechanics**. The main objective of the game is not visual programming (in fact in Minecraft, this feature has appeared after years in which interaction was only possible with text-based languages), but it does incorporate it and allows aspects of CT to work.

3.2. General characteristics

In **Table 1**, we can observe the general characteristics of the 26 analyzed tools. *Type* refers to the classification already mentioned. The country of creation, year of release, and number of languages are indicated. The number of users (in millions) has been indicated, in cases where a reasonably trustworthy approximation has been found.

The hegemony of the USA in this type of tools is prominent. Almost three-quarters (73.1%) of the platforms have been developed there, consistent with the US dominance in Internet services in general, and it may also be a response to an important campaign for interest in basic computing learning at school levels that the USA has been leading for a decade (we can note that in the wake Alice and Scratch, other tools have been emerging continuously). Canada follows it with 11.5%, the same as the whole of Europe, and Australia has its own platform with Cargo-Bot.

The oldest tools are Alice and Scratch, which explains their influence and emphasizes the importance of American universities (MIT and Carnegie Mellon) in the field of visual programming. The implantation data indicate the most widespread: Code.org, Tynker (although it is only in English), and Scratch, although we have not found data on the number of users of some

Game	Type	Ctry	Year	Lang#	User#	Technology						
						Web	Win	Mac	Linux	Andr	iOS	ChrOS
Alice	LANG	USA	1998	23		—	X	X	X	—	—	—
App Inventor	LANG	USA	2010	11	7	X	—	—	—	—	—	—
Bee-Bot	CHAL	USA	2012	1	0.3	—	—	—	—	—	X	—
Beetle Blocks	LANG	USA	2014	39		X	—	—	—	—	—	—
Blockly Games	CHAL	USA	2012	49		X	—	—	—	—	—	—
Cargo-Bot	CHAL	AUS	2012	1	1	—	—	—	—	—	X	—
Code.org - Courses	CHAL	USA	2011	51	430	X	—	—	—	—	—	—
Code.org - Code St	LANG	USA	2014	51	20	X	—	—	—	—	—	—
Daisy the Dinosaur	CHAL	USA	2013	1		—	—	—	—	—	X	—
Kodable	CHAL	USA	2012	1	>1	X	—	—	—	—	X	—
Kodetu	CHAL	ESP	2014	3	0.01	X	—	—	—	—	—	—
Kodu Game Lab	LANG	USA	2009	22	2.5	—	X	—	—	—	—	—
Hopscotch	LANG	USA	2012	3		—	—	—	—	—	X	—
Lightbot	CHAL	CAN	2008	28	7	X*	X	X	—	X	X	—
Made with Code	CHAL	USA	2014	1		X	—	—	—	—	—	—
MakeWorld	CREA	ESP	2016	6	0.002	X	—	—	—	—	—	—
Minecraft	GAME	SWE	2011	11	130	—	X	X	—	X**	X**	—
Robozzle	CREA	USA	2009	1	0.13	X*	—	—	—	X	X	—
Scratch	LANG	USA	2002	72	20	X*	X	X	X	—	—	—
ScratchJr	LANG	USA	2014	7	2	—	—	—	—	X	X	X
Snap!	LANG	USA	2011	39		X	—	—	—	—	—	—
SpriteBox	CHAL	CAN	2016	2		X*	—	—	—	X	X	—
The Foos	CHAL	USA	2014	17	4	X	—	—	—	X	X	—
Tynker	LANG	USA	2013	1	50	X	—	—	—	X	X	—
Tynker - Activities	CHAL	USA	2013	1	50	X	—	—	—	X	X	—
Waterbear	LANG	CAN	2011	1		X	—	—	—	—	—	—

*Browser needs flash plugin, silverlight in case of Robozzle.

**Minecraft has commercial apps for Android and iOS.

Table 1. General characteristics.

significant tools like Alice, ScratchJr, or Blockly. On the other hand, we have the widespread use of Minecraft as a construction game, with no specific data on how many people are using its visual programming possibilities. Availability on the web is clearly a key to massive use, leaving systems that only work on tablets to be more focused on commercial paid licenses, with the iPad holding a predominant place due to its implantation in schools in some countries (like the USA).

For the social data shown in **Table 2**, we have organized the platforms by type, because as you can see, the social options are strongly correlated with the approach of the tool. For those that pose programming challenges, there are no social options, except for Code.org, Blockly Games, and the Foos, which allow users to upload the creations made to the Internet at some levels, which coincide with those that offer a "free mode" (which works in fact as an exception to the rest of the levels, where the challenge has a solution that is either achieved or not.) On the contrary, in the visual programming languages, it is habitual to incorporate an additional community to the language (69% do this), that at least allows users to upload their projects and share them ("share") and, in some cases, more social options such as "like" other users' programs, "report" inappropriate programs, "fav" to bookmark, "follow" another user, or "comm" to comment with free text on the shared creation. In this sense, the two tools of creation and solution work in the same way as the visual languages, with the exception of Robozzle, which is the only one that incorporates "dislike," and the possibility of evaluating other users' puzzles from 1 to 5.

3.3. Learning aspects

Below we consider information relating to the use of these tools in class. First of all, the target age is fundamental, which is shown in **Table 3** where the recommended ages for the different platforms appear, marked according to the indications of the companies themselves, the opinions of the educational community, and the characteristics of the tools. On the one hand, we see that there are various tools for all ages, which is a good news for the educational community. On the other hand, if we consider them by type, as might have been expected, the platforms of challenges are focused on the lowest ages (average age 8.1); the creation and solution of challenges is higher (9.8); and higher still are languages to define games (12.4). This pattern corresponds to the stages that would be desirable if we want to design a longitudinal educational process with these tools: starting with a platform of challenges with the objectives and the path marked, continuing with a creation of challenges based on proposals made by the teacher in a structured and guided way, and ending with a more general-purpose visual language, in a learning environment based on projects and with freedom of personal choice on behalf of the learner to define and carry out the projects.

Table 4 shows aspects of simplicity of installation and use (valued from 0 to 3 according to a defined rubric), richness of interaction (also from 0 to 3), ranges of estimated time dedicated (based on available material and complexity and depth permitted for each system), and material available for teachers (A-D).

You can see the breadth of the range of time dedicated, which in the case of general programming tools such as Scratch or Tynker can vary from a few days to some years (many schools use these tools throughout several years, although not usually continuously). It can also be seen that the challenge systems have a much less ambitious temporal approach, except in the most highly developed levels such as Tynker or Code.org.

Type	Game	Community	Upload allowed	Social options	Remix
Challenges	Bee-Bot	—	—	—	—
	Blockly Games	limited	X*	share*	limited
	Cargo-Bot	—	—	—	—
	Code.org - Courses	—	X*	share*	—
	Daisy the Dinosaur	—	—	—	—
	Kodable	—	—	—	—
	Kodetu	—	—	—	—
	Lightbot	—	—	—	—
	Made with Code	—	—	—	—
	SpriteBox	—	—	—	—
	The Foos	—	X*	like, share*	—
	Tynker - Activities	—	—	—	—
Visual Programming Languages	Alice	—	—	—	exp
	App Inventor	X	—	—	exp
	Beetle Blocks	X	X	fav	X
	Code.org - Code Studio	X	X	share	X
	Hopscotch	X	X	like / share	X
	Kodu Game Lab	X	X	like / share / report	X
	Scratch	X	X	fav / like / report / comm / follow	X
	ScratchJr	—	—	—	—
	Snap!	—	X	—	X
	Tynker	X	X	like / report / share	X
	Waterbear	—	—	—	exp
Creation and playing	MakeWorld	X	X	like / follow / share / comm	X
	Robozzle	X	X	like / dislike / evaluate	—
Game	Minecraft	X	—	—	X

*Only in some levels, exp. = remix from file exported.

Table 2. Social characteristics.

Type	Game	Recommended ages for playing							
		<5	6–7	8–9	10–11	12–13	14–15	16–17	>18
CHAL	Bee-Bot	X	—						
	Blockly Games			X	X	X	X	—	—
	Cargo-Bot	—	X	X	X	—			
	Code.org - Courses		X	X	X	X	—	—	—
	Daisy the Dinosaur	X	X	—					
	Kodable	X	X	X	—	—	—		
	Kodetu			X	X	X	X	—	—
	Lightbot	X	X	X	X	—			
	Made with Code	—	X	X	X	X	X	—	
	SpriteBox	X	X	X	X	—			
	The Foos	X	X	—	—				
	Tynker - Activities			X	X	X	X	—	—
CREA	MakeWorld	—	X	X	X	X	X	—	
	Robozzle		X	X	X	X	—	—	—
LANG	Alice				—	—	X	X	X
	App Inventor						—	X	X
	Beetle Blocks					X	X	X	—
	Code.org - Code Studio				X	X	X	—	—
	Hopscotch			X	X	X	—	—	
	Kodu Game Lab	—	X	X	X	X	—	—	
	Scratch			X	X	X	X	—	—
	ScratchJr	X	X	—	—				
	Snap!			X	X	X	X	X	X
	Tynker			X	X	X	X	—	—
	Waterbear				—	X	X	X	—
GAME	Minecraft						—	X	X

X = recommended, – = viable, space = not recommended.

Table 3. Recommended ages of platforms.

3.4. Engagement

There are no unique or universal expressions of the fun or engagement that a digital activity is capable of producing in a user. Each person has his or her tastes, preferences, and learning; in addition, in the school environment, the way in which an activity is proposed greatly influences its reception. It is not the same for a child to freely choose a game that s/he wants to

Type	Game	Inst [1]	Personal data requested	Us [2]	Int [3]	Apprentice dedication time range	Teacher preparation time range	Av Mat [4]
CHAL	Bee-Bot	0	No	0	0	1 h–10 h	1 h–5 h	B
	Blockly Games	0	No	0	0	1 h-15d	5 h–20 h	C
	Cargo-Bot	0	No	1	1	1 h-5d	1 h–10 h	A
	Code.org - Courses	0	Email (opt.)	0	1	1 h-2 m	5 h–50 h	D
	Daisy the Dinosaur	0	No	0	1	1 h–4 h	1 h–5 h	B
	Kodable	0	No	0	1	1 h-2 m	5 h–50 h	D
	Kodetu	0	Sex, age, school, survey	0	0	1 h–5 h	1 h–5 h	A
	Lightbot	2	No	0	1	1 h–3 h	1 h–5 h	C
	Made with Code	0	No	0	1	1 h–20 h	1 h–20 h	C
	SpriteBox	0	No	0	1	1 h–3 h	1 h–5 h	C
	The Foos	2	Email	0	1	1 h-30d	5 h–30 h	C
	Tynker - Activities	0	Email (opt.)	0	1	1 h-1y	2 h–10 h	C
CREA	MakeWorld	1	Sex, country, age, email	1	2	1 h-2y	5 h–40 h	D
	Robozzle	0	Email (opt.)	1	1	1 h-5d	2 h–10 h	B
LANG	Alice	2	No	3	3	3d-4y	20 h–50 h	D
	App Inventor	3	Google account	3	3	3d-4y	20 h–80 h	D
	Beetle Blocks	0	No	1	2	2d-1y	5 h–40 h	B
	Code.org - Code St	0	Email, age, sex (opt.)	0	2	5d-2y	20 h–50 h	D
	Hopscotch	0	Email (opt.)	2	2	5d-2y	20 h–50 h	C
	Kodu Game Lab	2	No	2	3	5d-2y	10 h–50 h	C
	Scratch	1	Birth date, sex, country, email	2	2	5d-2y	20 h–50 h	D
	ScratchJr	2	No	2	1	1d-2 m	5 h–10 h	D
	Snap!	0	Birth date, email	2	2	5d-2y	20 h–50 h	C
	Tynker	0	Email	1	2	5d-2y	5 h–50 h	C
	Waterbear	0	No	0	2	5d-2y	20 h–50 h	A
GAME	Minecraft	3	Email	3	3	5d-2y	20 h–80 h	D

Inst = Installation simplicity. Us = Usability. Int = Interaction richness. [1–3] evaluated in a range 0–3, where 0 is the simplest and 3 most complex. Av Mat = Available material for teachers [4] is ranged A-D, from no material available (A) to very rich content in many languages (D).

Table 4. Other learning aspects. Installation simplicity.

play when s/he wants, as to be asked to do so by the teacher, more or less obligatorily, within a class. This is a factor that has influenced all educational games from the start. But in any case, we have done the exercise of trying to objectify the "fun" potentially offered by each of our 26 tools, differentiating some of the classic dimensions that influence the experience of the user when it comes to video games and evaluating each of them from 0 (minimum) to 3 (maximum). This gives us an average measure of engagement that is displayed in descending order in **Table 5**.

Game	Type	Sen	Fan	Nar	Cha	Fel	Dis	Exp	Sub	Sto	Engagement
Minecraft	GAME	3	3	3	3	3	3	3	3	2	**2.89**
Code.org - Code Studio	LANG	2	3	2	3	2	2	3	3	3	**2.56**
Tynker	LANG	2	3	2	3	2	2	3	3	3	**2.56**
Scratch	LANG	3	3	2	3	0	2	3	3	3	**2.44**
Hopscotch	LANG	2	3	1	3	2	2	3	3	3	**2.44**
Alice	LANG	2	3	1	3	2	2	3	3	3	**2.44**
Snap!	LANG	2	3	1	3	2	2	3	3	3	**2.44**
MakeWorld	CREA	2	3	2	3	2	2	2	2	3	**2.33**
Kodu Game Lab	LANG	2	3	2	3	0	2	3	3	2	**2.22**
Code.org - Courses	CHAL	2	2	2	3	1	2	2	3	2	**2.11**
ScratchJr	LANG	2	3	1	3	0	1	2	3	2	**1.89**
The Foos	CHAL	2	2	2	3	1	2	1	2	1	**1.78**
Blockly Games	CHAL	1	1	1	3	1	2	2	2	2	**1.67**
Made with Code	CHAL	2	1	1	2	1	2	2	2	2	**1.67**
Tynker - Activities	CHAL	2	2	2	3	1	1	1	2	1	**1.67**
Kodable	CHAL	1	1	2	3	0	2	2	2	1	**1.56**
Beetle Blocks	LANG	1	0	0	3	1	2	2	2	1	**1.33**
Waterbear	LANG	1	0	0	3	0	1	2	2	3	**1.33**
App Inventor	LANG	1	0	0	3	0	2	3	2	0	**1.22**
Cargo-Bot	CHAL	1	1	1	2	1	1	1	2	0	**1.11**
Lightbot	CHAL	1	1	1	3	0	1	1	2	0	**1.11**
Robozzle	CREA	0	0	1	3	2	1	1	2	0	**1.11**
SpriteBox	CHAL	1	1	1	3	0	1	1	1	0	**1.00**
Bee-Bot	CHAL	1	1	1	2	0	1	0	2	0	**0.89**
Kodetu	CHAL	0	1	1	2	1	1	1	1	0	**0.89**
Daisy the Dinosaur	CHAL	1	1	1	1	0	1	1	1	0	**0.78**

Sen = Sensation, Fan = Fantasy, Nar = Narrative, Cha = Challenge, Fel = Fellowship, Dis = Discovery, Exp = Expression, Sub = Submission, Sto = Storytelling.

Table 5. Engagement expressed depending on different dimensions of fun (from 0-min- to 3-max each).

We note that this confirms the logical relationship between the most widespread and the most attractive games (Minecraft, Code.org, Tynker, Scratch). In addition, in general, the languages that allow free creative development and longer periods of engagement are more attractive than those games of challenges whose attraction basically ends when the challenges end. As expected, we see in the lower part the tools that allow shorter periods of engagement and others (like App Inventor) whose complexity and low level coding make the effort invested disproportionate to the attractiveness of the result achieved, from a gaming point of view.

3.5. Computational thinking

It is a fundamental aspect for our analysis to review which of the specific characteristics that are employed in CT are included in the tools analyzed. **Table 6** shows the aspects that each tool includes, or not, along with some significant data such as the number of different blocks that can be used to program (calculated counting all the different blocks that the system allows to be used) or whether the equivalent textual code can be seen in parallel to the visual program that is being developed.

We have not included sequences in the table, which all the tools analyzed have (we cannot imagine a visual programming tool without sequences). We have also seen that flowcharts, a visual tool widely used in programming and in learning programming at the conceptual level, are not used in any of these tools. On the other hand, recursion is used in two different ways: in those tools that do not support loops, (virtually infinite) repeating is performed using recursive calls (this is the case of Cargo-Bot, LightBot, MakeWorld, and Robozzle).

The languages generally support more features and use many more basic blocks to allow greater expressiveness of programming (all have more than 100 constructions, except Hopscotch that limits them due to its orientation to tablets, and ScratchJr that is aimed at the youngest age groups). Conversely, the systems of challenges have much less expressiveness except the four most developed ones which support a large number of levels and greatly diversify the constructions that can be used in each challenge: Code.org, Blockly, Tynker, and Made with Code.

It is also significant that all the languages support events, which speaks of the importance of event-oriented programming in current computing and also shows that the concept of an event that provokes an action has a very natural meaning for learners. Most languages allow multithreading, albeit in a way that is transparent to the learner, who probably does not need to understand the concept to use it implicitly. Only some of the languages (but no challenge tools) allow message passing, object-orientation, 3D, and connection with physical systems. Only one tool (Alice) incorporates the explicit construction of parallel sequences (to launch several blocks in parallel in the same temporal space).

3.6. Design aspects

The last dimension analyzed has been some design considerations of the tools, from the point of view of the approach to the interface, the sequencing of user interaction, and the options available for professors and researchers (see **Table 7**).

Type	Game	Loo	Alt	Deb	Mod	Var	Exp	Blo#	Evn	Thr	Rec	Mes	OO	3D	Txt	Langs	Out
CHAL	Bee-Bot							4									
	Blockly Games	X	X	X	X	X	X	136							X	Js	
	Cargo-Bot		X	X	X			6			X						
	Code.org	X	X	X	X	X	X	>200	X	X					X	Js	
	Daisy	X		X				9	X								
	Kodable	X	X	X	X			7									
	Kodetu			X				9							X	Js	
	Lightbot			X	X			7			X						
	Made w/Code	X	X	X			X	>200									
	SpriteBox	X		X				7							X	Js, Sw	
	The Foos	X	X	X			X	20	X	X							
	Tynker - Act	X	X	X			X	>200	X	X							
CREA	MakeWorld			X	X			17	X	X	X						
	Robozzle		X	X	X			10			X						
LANG	Alice	X	X	X	X	X	X	>200	X	X	X	X	X	X	X	Java	
	App Inventor	X	X		X	X	X	>200	X	X	X	X	X			Java	X
	Beetle Blocks	X	X	X	X	X	X	120	X	X				X			
	Code Studio	X	X	X	X	X	X	>200	X	X	X	X	X		X	Js	
	Hopscotch	X	X		X	X	X	86	X	X	X						
	Kodu						X	>200	X	X				X			
	Scratch	X	X		X	X	X	130	X	X	X	X					X
	ScratchJr	X		X				26	X	X		X					
	Snap!	X	X	X	X	X	X	150	X	X	X	X					X
	Tynker	X	X	X	X	X	X	>200	X	X	X	X			X	Js, Py	X
	Waterbear	X	X			X	X	>200	X			X	X				
GAME	Minecraft	X	X	X		X	X	150	X	X				X	X	Js	

Loo = Loops, Alt = Alternatives, Deb = Visual debugging in execution, Mod = Modules (subprograms), Var = Variables, Exp = Expressions, Blo# = # of code constructs (expressiveness of language), Evn = Events, Thr = Multithreading, Rec = Recursion, Mes = Message passing, OO = OO, 3D = 3D, Txt = Text language equivalent, Langs = Language, Out = Output to physical world (possible connection with robots, sensors, arduino, etc.).

Table 6. Some CT aspects.

In addition to the data given in the table, we have also reviewed the adaptability of the tools but we have not found any. That is, the software always behaves the same regardless of the characteristics of the user (age, gender, educational level, functional diversity, etc) or their behavior (whether the program is wrong or right, better or worse, the game does not change

Type	Game	Prog. Interface	Tut	Help	Free	Reg	Grp	Feed	Dash	Use	Res
CHAL	Bee-Bot	Icons*			X						
	Blockly Games	Ver - blocks	X	X	X						
	Cargo-Bot	Hor - icons	X	X	X						
	Code.org	Ver - blocks	X	X	X	X	X	X	3		
	Daisy	Ver - blocks									
	Kodable	Hor - icons	X	X		X	X	X	3		
	Kodetu	Ver - blocks	X				X	X	2		X
	Lightbot	Hor - icons	X	X							
	Made w/Code	Ver - blocks	X	X							
	SpriteBox	Ver - icons	X	X							
	The Foos	Hor - blocks	X	X		opt	X	X	1		
	Tynker - Act	Ver - blocks	X	X		opt	X	X	3		
CREA	MakeWorld	Ver - icons	X		X	X					
	Robozzle	Hor - icons	X		X	opt				X	
LANG	Alice	Ver - blocks	X	X	X						
	App Inventor	Ver - blocks		X	X	X					
	Beetle Blocks	Ver - blocks	X	X	X	opt					
	Code Studio	Ver - blocks	X	X	X	X				X	
	Hopscotch	Ver - blocks	X	X	X	X					
	Kodu	Graph - icons		X	X						
	Scratch	Ver - blocks		X	X	opt		X		X	
	ScratchJr	Hor - blocks		X	X						
	Snap!	Ver - blocks		X	X	opt					
	Tynker	Ver - blocks	X	X	X	X	X	X	2		
	Waterbear	Ver - blocks			X						
GAME	Minecraft	Ver - blocks	X	X	X	X	X	X	2		

Tut = Integrated tutorial, Help = Integrated help, Free = Free navigation, Reg = User registration needed, Grp = Group creation possible (for teachers), Feed = Feedback for teacher of user's behavior, Dash = Teacher's dashboard (0-no to 3-complete), Use = Public access to users' data, Res = Public research access to user data.

Table 7. Design considerations.

the subsequent levels nor does it provide different information or tutorials.) The only thing that approaches adaptability is the score, which we discuss in the following table.

Reviewing the type of interface that is proposed for the metaphor of "code" (the panel to which you can drag the pieces to develop the program), we see that the most common option is vertical (69.2%), which represents the sequence from top to bottom, and rather less common is horizontal (23.1%) which represents the sequence from left to right (in a few cases with local

adaptation to the languages that are written from right to left). There are two special tools that do not fit into these two schemes: Kodu that proposes a creative graphic interface in a circle where the options are carried out by levels, and each level shows the available options with icons, joined in a circle; and Bee-Bot, which has no explicit code space: each learner has to remember by heart the program sequence that s/he "loads" in the bee (just as happens with the bee-bot physical device).

The preference in vertical interfaces is for blocks (only MakeWorld and SpriteBox, aimed at lower age bands, propose vertical icons), and in horizontal interfaces, icons (except for the Foos and ScratchJr which develop graphically elaborated blocks to represent the repetitive structures mounted on repeating icons). The tendency is to use horizontal structures with younger age groups and vertical ones with older age groups. The blocks usually have the visual form of a puzzle, colored to differentiate the types of construction visually. In some cases like Alice, App Inventor, or Waterbear, the blocks represent concepts that are very close to the corresponding low-level text-based code.

In the table, you can also see the tools provided for the teachers. Those that provide information to the teacher and allow him/her to manage groups of students, often also have an online dashboard in which the teacher, through the web, can consult information on the actions of his/her group. This is fairly complete in the case of Code.org and Kodable (progress by topic, lessons completed), and especially detailed in Tynker (also including the skills worked and the level).

Type	Game	Pre-level	Error	Success	Progress	Progress info accessible
CHAL	Bee-Bot	X		X	X	Stars, points
	Blockly Games	X		X	X	Levels passed
	Cargo-Bot	X	X	X	X	Stars, levels passed
	Code.org	X	X	X	X	Levels passed, Code length
	Daisy	X		X		
	Kodable	X	X	X	X	Levels passed, points
	Kodetu	X		X		
	Lightbot			X		
	Made w/Code	X	X	X		
	SpriteBox			X	X	Levels passed, points
	The Foos	X	X	X	X	Levels passed, stars
	Tynker - Act	X	X	X	X	Levels p., stars, prizes, certifs, concepts
CREA	MakeWorld			X		
	Robozzle			X	X	Levels passed, solution length

Pre-level = Feedback before each level, Error = Feedback after levels failed, Success = Feedback after levels passed, Progress = Explicit info on user's progress in game.

Table 8. Feedback to user.

Public access to data is not common; very few tools show general use data and even less allow access to information for research.

Finally, in **Table 8**, we can see information about the feedback that is given to the user as the system progresses. We only consider challenge games, these being the ones that can guide the learner through an expected series of actions.

4. Conclusions

Throughout this chapter, we have seen that there are a growing number of options to lead a learner through CT. An interesting learning path can be to start with some of the challenge games for a few days and move on from there to a visual programming language that can involve weeks or months of activity. The fun is assured, and there are multiple options, in addition to a diversity of motifs that make use of well-known themes of films or video games to reinforce the experience.

However, there are still some limitations to be considered for the next generation of CT games. There is excessive use of action primitives that have to do with movement and orthogonal rotation (influenced perhaps, as we are all, by Logo and its historical importance); instead of other auditory, rhythmic, or visual options, that also allow for developing algorithmic thought and exploring abilities other than spatial vision: in this sense, we note some of the most recent activities incorporated by Code.org and Made w/Code with multimedia elements. We also find the predominance of blocks; tools like Kodu using flowcharts open possibilities to design new CT tools combining both and other visual expressions, beyond the only visual abstraction of nested blocks. Another widespread lack is that in this nascent field, it is especially important to investigate how users behave and how systems facilitate their learning, so it would be desirable for original digital tools to facilitate the use of open-access information for learning analytics, to allow improvement and provide quantum leaps in the design of new levels and tools. A final important gap is the lack of adaptive learning; practically, all the tools behave always the same, regardless of age, prior knowledge, or the skill shown by the user. It is important that tools begin to use passed user activity to adapt and significantly improve the educational experience, something that should also be especially feasible in this type of fully digital systems.

A key issue in learning is assessment. In the games (which pose a quantifiable, measurable and observable challenge), an adequate assessment is more feasible. In fact, it is already being considered in some platforms: code size—number of blocks—in Code.org, stars for time or level objectives in the Foos and others, and so on. Assessment should be improved to include more key indicators in CT skills, such as program efficiency (number of execution steps) and user behavior in the process, not just the result (number of errors, number of code changes, type of development to the right result, etc). In programming languages, assessment is much more complicated. In the same way that a programming teacher working with a text-based language (e.g., Python or Java) has a complex task to evaluate his students, even more so for a primary or secondary teacher who is not necessarily a computing expert and is not looking for

the same things. Therefore, though remixing can be used to reinforce the skills included in CT [13], the tools still lack the automatic possibilities or even of capacities for teachers to carry out a progressive monitoring of the learner experience in open environments: new mechanisms of analysis and evaluation are needed so that we can verify that students go beyond solving a problem, and study how they solve it and how they progress. Probably, subsystems of the type proposed in Dr. Scratch [14] will be incorporated to facilitate indicators that enrich the process for both learners and teachers. There is also an important need for common criteria to know CT skills so as to develop and evaluate them. In this respect, easily digitized tools such as the CTtest proposed by Román-González [15] can be a great complement for assessment in medium-term training processes, using both games and languages.

Observing only the category of games based on programming challenges, we review the importance of generating more engagement after the challenges are over, incorporating techniques of gamification known as punctuation, classifications, proposals for improvement, and the incorporation of creative levels in which the challenge is not limited by simple quantifiable objectives; a line in which the entities with more resources, such as Code.org and Tynker, are already working. Another key issue in these systems is the scaffolding. In addition to the obvious effort of level design (a need shared by both games and education), guided by the experience of designers, it is a challenge to systematize the process to ensure the good development of learning and progression of motivation, seeking the flow that so many games achieve; it is also necessary to investigate the guidelines for the automatic generation of levels/challenges, in the line already known in many procedural generation video games. It will also be important to consider how to set out the introductions and tutorials to maximize the learning objective and identify the type of thinking that the learner is applying to solve each progressive challenge, as discussed using Kodu in [16]. Another final gap in challenge games is that few systems allow the creation of new challenges for learners or their teachers, limiting the experience and prematurely closing and limiting the learning cycle. Tools like MakeWorld or Robozzle, which not only allow you to play but also to edit new worlds, will be in the next generation of games to increase the personalization of challenges through modification or creative contribution to different challenges, in a characteristic type of remix.

Regarding the programming languages category, there are intrinsic limitations to block-based visual environments compared to text-based languages. This is more noticeable in large programming projects due to limitations on the visibility of the code, code navigation difficulty, or lack of control in source modifications [7]. Bidirectional conversion between visual and text programming language is available in an increasing number of platforms such as Code.org App Lab. This feature allows learners to choose the most useful view depending on the complexity of the project. We also note that game-oriented platforms are also including teachers' dashboards (Code.org, Kodable, Tynker) but are still very limited or nonexistent in the programming languages category, reflecting the lack of assessment tools already mentioned.

Finally, we have included, in our analysis, a small category of videogames that allow using visual programming in their mechanics. In this regard, Minecraft is an interesting example

for game designers. Good videogames can also be designed taking into account specific CT mechanics. We believe this is one of the challenges for the following years in game design, not only for construction games but also for graphic adventures, RPGs, FPS, and many other types of videogames.

Author details

Andoni Eguíluz*, Pablo Garaizar and Mariluz Guenaga

*Address all correspondence to: andoni.eguiluz@deusto.es

Faculty of Engineering, Universidad de Deusto, Bilbao, Spain

References

[1] Wing JM. Computational thinking. Communications of the ACM. 2006;**49**(2):33-35. DOI: 10.1145/1118178.1118215

[2] Buitrago Flórez F, Casallas R, Hernández M, Reyes A, Restrepo S, Danies G. Changing a generation's way of thinking: Teaching computational thinking through programming. Review of Educational Research. 2017. DOI: 10.3102/0034654317710096

[3] Wing JM, Stanzione D. Progress in computational thinking, and expanding the HPC community. Communications of the ACM. 2016;**59**(7):10-11. DOI: 10.1145/2933410

[4] Tedre M, Denning PJ. The long quest for computational thinking. In: Proceedings of the 16th Koli Calling International Conference on Computing Education Research (Koli Calling '16); New York, NY, USA. ACM; 2016. p. 120-129. DOI: 10.1145/2999541.2999542

[5] Denning PJ. Remaining trouble spots with computational thinking. Communications of the ACM. 2017;**60**(6):33-39. DOI: 10.1145/2998438

[6] Kazimogl C, Kiernan M, Bacon L, MacKinnon L. Learning programming at the computational thinking level via digital game-play. Procedia Computer Science. 2012;**9**:522-531. DOI: 10.1016/j.procs.2012.04.056

[7] Bau D, Gray J, Kelleher C, Sheldon J, Turbak F. Learnable programming: Blocks and beyond. Communications of the ACM. 2017;**60**(6):72-80. DOI: 10.1145/3015455

[8] Powers K, Gross P, Cooper S, Mcnally M, Goldman K, Proulx V, Carlisle M. Tools for teaching introductory programming: What works? ACM SIGCSE Bulletin. 2006;**38**(1):560. DOI: 10.1145/1124706.1121514

[9] García-PeñalvoFJ, HughesJ, ReesA, JormanainenI, ToivonenT, ReimannD, TuulM, VirnesM. Evaluation of existing resources (study/analysis). Belgium: TACCLE3 Consortium. 2016; DOI: 10.5281/zenodo.163112

[10] Sandoval-Reyes S, Galicia-Galicia P, Gutierrez-Sanchez I. Visual learning environments for computer programming. In: Electronics, Robotics and Automotive Mechanics Conference (CERMA); IEEE; 2011. p. 439-444. DOI: 10.1109/CERMA.2011.76

[11] Daly T. Using introductory programming tools to teach programming concepts: A literature review. The Journal for Computing Teachers. 2009;Fall:1-6

[12] Panitz M, Sung K, Rosenberg R. Game programming in CS0: A scaffolded approach. Journal of Computing Sciences in Colleges. 2010;**26**(1):126-132

[13] Dasgupta S, Hale W, Monroy-Hernández A, Hill BM. Remixing as a pathway to computational thinking. In: Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing (CSCW '16); New York, NY, USA: ACM; 2016. p. 1438-1449. DOI: 10.1145/2818048.2819984

[14] Moreno-León J, Robles G. Dr. Scratch: A Web Tool to Automatically Evaluate Scratch Projects. In: Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE '15); New York, NY, USA. ACM; 2015. p. 132-133. DOI: 10.1145/2818314.2818338

[15] Román-González M, Pérez-González JC, JiménezFernandez C. Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. Computers in Human Behavior. 2016:1-14

[16] Touretzky DS, Gardner-McCune C, Aggarwal A. Semantic reasoning in young programmers. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17); New York, NY, USA: ACM; 2017. p. 585-590. DOI: 10.1145/3017680.3017787

IntechOpen