British Journal of Educational Technology *Vol 0 No 0 2018* 1–
doi:10.1111/bjet.12685

Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games

Renny S. N. Lindberg , Teemu H. Laine and Lassi Haaranen

Renny S. N. Lindberg is a doctoral student in the Department of Computer Sciences at the Vrije Universiteit Brussel, Belgium. Teemu H. Laine is an associate professor in the Department of Computer Science, Electrical and Space Engineering at the Luleå University of Technology. Lassi Haaranen is a doctoral student in the Department of Computer Science at the Aalto University. Address for correspondence: Teemu H. Laine, Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Forskargatan 1, 931 87 Skellefteå, Sweden. Email: teemu@ubilife.net

Abstract

An increasing number of countries have recently included programming education in their curricula. Similarly, utilizing programming concepts in gameplay has become popular in the videogame industry. Although many games have been developed for learning to program, their variety and their correspondence to national curricula remain an uncharted territory. Consequently, this paper has three objectives. Firstly, an investigation on the guidelines on programming education in K-12 in seven countries was performed by collecting curricula and other relevant data official from governmental and non-profit educational websites. Secondly, a review of existing acquirable games that utilize programming topics in their gameplay was conducted by searching popular game stores. Lastly, we compared the curricula and made suggestions as to which age group the identified games would be suitable. The results of this study can be useful to educators and curriculum designers who wish to gamify programming education.

Introduction

Computer programming has been an essential tool for computer scientists and engineers for decades. Today, programming is recognized as a core competency skill also outside the engineering field. The demand for workers possessing programming skills is expected to grow drastically in the near future. In Europe alone, there will be over 800,000 ICT job vacancies by 2020 Balanskat and Engelhard (2015). Consequently, several governmental and non-governmental projects, such as the European Coding Initiative and the Code.org, have been established to promote programming education, with emphasis on children.

Likewise, countries around the world have moved toward inclusion of programming education to national school curricula (Alano *et al*., 2016; Australian Curriculum Assessment and Reporting Authority, 2015; Balanskat & Engelhardt, 2015; Department for Education (UK), 2013a; Information Technology Foundation for Education (Estonia), 2015; Ministry of Education (Finland), 2014; Ministry of Education (France), 2015; Ministry of Education (Israel), 2016; Ministry of Education and Science (Korea), 2015). This move has been particularly strong in Europe, as evidenced by a report published in 2015, which surveyed 21 Ministries of Education

Practitioner Notes

What is already known about this topic

- Many countries have started to implement programming education in their curricula.
- Many games utilize programming in their gameplay.
- New techniques to teach children programming are required.

What this paper adds

- A comparison of several countries' approaches to programming education at primary and secondary school levels.
- A review of games that utilize programming in gameplay and what topics they cover.
- Suggestions on what type of programming games might be useful for students at primary and secondary school levels.

Implications for practice and/or policy

- Useful for teachers interested in providing additional learning avenues to their students.
- The identified programming education topics can guide educational game developers to design games with sufficient topic coverage.

around Europe (Balanskat & Engelhardt, 2015) with an aim to discover how programming has been integrated into national curricula around Europe.

Programming has also become a notable topic in the video game industry. In this paper, our definition of programming games covers games that utilize at least one clear programming concept—repetition, conditionals, Boolean logic, and so forth—in gameplay for the purpose of entertainment or education. Several programming games are introduced to the market yearly. Vahldick *et al.* (2014) surveyed commercial games and research projects that were developed for programming education. Several of these games are unavailable, or no longer supported by their developers.

We discovered only two papers surveying curricula with the focus of programming education: Heintz, Mannila & Färnqvist (2016) reported on curricula as they now stand, and Balanskat and Engelhard (2015) conducted a more thorough survey. Moreover, perhaps due to the relative youth of this movement of adding programming into curricula, there are not many papers discussing possible connections between pedagogical objectives in curricula and programming games. Therefore, in this paper we aim to accomplish three tasks:

1. Discover and analyze primary and secondary school curricula around the world that cover programming education.
2. Discover and analyze games that incorporate clear programming concepts in them.
3. Propose games that could support the curricula objectives at different levels.

Background

Computer Science Curricula

Computer science (CS) curricula at university contexts have a history dating back to the 1960s (Conte *et al.*, 1965). Since its first recommendation for a curriculum, the Association for Computer Machinery (ACM), in collaboration with the Institute of Electrical and Electronics Engineers (IEEE), has continued to publish guidelines for CS curricula, with the latest one being the 2013

edition (ACM & IEEE, 2013). CS appeared also briefly in K-12 in the United States in the late 1970s, but transformed rather quickly into other types of computer-assisted education (Resnick *et al.*, 2009). CS has re-appeared in the K-12 curricula only recently due to the awareness of the growing need of CS skills.

A report by Balanskat and Engelhard (2015) surveyed how CS and especially programming have been implemented in the national curricula of 21 European countries, revealing that 16 countries had programming integrated and 2 countries had plans to integrate programming from 2016 onwards. It was noted that programming was mainly integrated into the secondary education level, but that several countries were also on their way to integrating programming into the primary education level. For example, South Korea is planning on integrating programming into all levels by 2018 (Ministry of Education and Science (Korea), 2015). Australia's most recent F-10 (Foundation—Year 10) curriculum from 2015 has programming practices from 3rd year onwards (Australian Curriculum Assessment and Reporting Authority, 2015). The United States does not have a national curriculum with programming education. However, their K-12 Computer Science Framework, promoted by the ACM, Code.org and other notable organizations, aims to provide guidelines for integrating programming into K-12 in individual states (Alano *et al.*, 2016).

Learning Programming by Games

Games have been used for education under different terms, such as: serious games, edutainment, gamification, game-based learning and playful learning. All these emphasize that the game has defined learning objectives (Plass, Homer, & Kinzer, 2015). There has been some debate on how education contents should be implemented in educational games, which has led to the creation of terms related to different approaches. For instance, *edutainment* games are generally perceived to be educational tools where the game is simply a means to an end, whereas *playful learning* is viewed as a game where the process of learning happens through the game (Lode, Franchi, & Frederiksen, 2013; Resnick, 2004).

Other terms that can be loosely tied in with playful learning are *unintentional* and *incidental learning*. The term unintentional learning was used in Boyle *et al.*'s survey on the impacts of digital games, in which they noted that "unintentional learning found in entertainment games could provide insights into engagement and learning in serious games" (Boyle *et al.*, 2016). Incidental learning is described in the following way by Marsick and Watkins (2001): "…when people learn incidentally, their learning may be taken for granted, tacit, or unconscious. However, a passing insight can then be probed and intentionally explored."

Gamification is defined as the use of game elements in a non-game context (Deterding, Dixon, Khaled, & Nacke, 2011). However, the term is also used in other ways, as noted by Plass *et al.* (2015), who state the central quality of gamification as follows: "…it involves the use of game elements, such as incentive systems, to motivate players to engage in a task they otherwise would not find attractive."

The Project Tomorrow, a non-profit group that annually surveys schools in the United States, found that 48% of 38,613 teachers were using games for education in 2015 (Project Tomorrow, 2016). Another independent study surveyed 694 K-8 teachers in the United States on how they use digital games in teaching (Takeuchi & Vaala, 2014). The results suggested that although 74% of the teachers used digital games for educational purposes, they pointed out the lack of curriculum-aligned games and guidance for integrating games into teaching. Similarly, Becker (2007) identified lack of resources for searching available products and tools as one of the difficulties that teachers face when it comes to integrating new technologies into pedagogy. Finally, Takeuchi and

Table 1: Evaluations of programming games

Study	*Purpose*	*Sample*	*Data*	*Outcomes*
Long (2007)	Learning effects and player perceptions of Robocode	83 users of a Robocode discussion forum (age: from <18 to >50)	Questionnaire	Learning effectiveness, motivation, fun, enjoyment
Eagles and Barnes (2009)	Learning effects of a puzzle game	26 university students (age: undisclosed)	Pre- and post-test questionnaire	Learning effectiveness
Liu *et al*. (2011)	Evaluation of a game for learning computational problem solving	117 university students (age: undisclosed)	Pre-questionnaire, log data, questionnaire	Motivation, engagement, learning effectiveness
Lee and Ko (2011)	Evaluation a program debugging game	116 novice programmers (age: 18–59)	Questionnaire, log data	Motivation, engagement
Bromwich *et al*. (2012)	User perceptions of a puzzle game prototype	56 students (age: 13–16)	Observations, log data	Enjoyment, fun, motivation
Esper *et al*. (2013)	Novice's responses to the CodeSpells API	17 children and young adults (age: 8–22)	Pre-questionnaire, video recording, post-interview	Immersion, fun
Lode *et al*. (2013)	Evaluation of Machineers during development	21 school students (age: undisclosed)	Observations, think-aloud, peer-play	Immersion, intrinsic motivation, engagement
Mathrani *et al*. (2016)	Pedagogical evaluation of Light-Bot	44 students of a non-university diploma course (age: undisclosed)	Questionnaire	Learning effectiveness, fun, engagement

Vaala (2014) provided recommendations for improving integration of games into teaching, such as: creating a framework for describing and evaluating educational games, elevating awareness on how to integrate games and creating innovative integration models.

Numerous studies have measured the effectiveness of programming games. As Table 1 shows, positive results have been shown for learning, motivation, engagement, fun and immersion. Yet the studies had notable limitations, such as small sample size and length of the study. Additionally, many focused on older learners.

Block versus Text

One of the features that differentiates learning environments for programming is whether they use a text-based programming language (eg, Python) or a block-based language (eg, Scratch). Under Kelleher and Pausch's (2005) taxonomy of novice programming environments, block-based systems fall under the category of "find alternatives to typing." A key motivation behind block-based systems is that novice programmers often struggle with syntax of a textual language. As block-based languages remove the need for typing syntactically correct instructions, they can improve learning of programming concepts while avoid struggling with syntax.

Perhaps the best known example of a block-based programming is Scratch (Resnick *et al.*, 2009). It allows the construction of programs through a simple interface via dragging and dropping programming constructs with a mouse. This programming method eliminates syntax errors since there is no typing involved. Additionally, Scratch will only allow certain blocks to snap together, thus preventing illogical structures. The idea of Scratch has been adopted to many block-based learning environments (eg, Code.org,[1] Alice,[2] Tynker,[3] Kodu,[4] MakeCode[5] and HopScotch[6]), and further popularized by Google's Blockly[7] library for building visual programming environments.

Block-based systems cannot solve all issues that a novice programmer might encounter. For example, when comparing block-based and text-based approaches, the level of perceived difficulty remains the same (Price & Barnes, 2015). Similar effect was also noted by Lewis (2010) when comparing students interpreting a loop in block-based Scratch and text-based Logo. In advanced topics that require more elaborate coding, block-based systems are less effective. This was discovered by Weintrop and Wilensky (2015) in a study where high school students noted the drawbacks of using blocks to be lack of expressive power and authoring larger, more sophisticated projects. However, block-based approaches are beneficial in some ways. For example, Price and Barnes (2015) noted increased performance in time-on-task and task completion. Moreover, Weintrop and Wilensky (2015) mentioned that the students generally found block-based programming to be easier than a text-based alternative. Finally, while block-based languages are ideal for gently introducing programming concepts to beginners, text-based programming languages, such as Java and Python, might be more suitable for advanced students, as they allow building more sophisticated programs.

[1]https://code.org/

[2]https://www.alice.org/

[3]https://www.tynker.com/

[4]https://www.kodugamelab.com/

[5]https://makecode.com/

[6]https://www.gethopscotch.com/

[7]https://developers.google.com/blockly/

Methodology

The reviewed curricula were searched using Internet search engines and Ministry of Education websites of respective countries. The criterion for selection was for the country to have integrated programming into its national curriculum, or published curriculum guidelines, at least for the primary education.

In total, we reviewed curricula in five countries: Australia (Australian Curriculum Assessment and Reporting Authority, 2015), Finland (Ministry of Education (Finland), 2015), France (Ministry of Education (France), 2015), Israel (Bargury *et al.*, 2012; Ministry of Education (Israel), 2016), and the United Kingdom (Department for Education (UK), 2013a, 2013b). Moreover, we covered the United States (Alano *et al.*, 2016) and Estonia (Information Technology Foundation for Education (Estonia), 2015), although they do not have official curriculum; instead, we analyzed their guidelines for curricula.

To analyze the identified curricula, we first created broad categories based on the curricula to describe differences in foci. The process began by iterating through the curricula descriptions and observing common themes that emerged. These themes were then further abstracted into the categories utilized in Tables 1 and 2. We divided the curricula contents into grades 1–6 (ages 7–12) and 7–12 (ages 13–18). Grades 7–12 were bundled together because very few of the curricula contained recommendations for secondary education.

Vahldick *et al.*'s (2014) survey of 40 games was used as a starting point for reviewing programming games. Firstly, we included also recent games by searching popular game stores, mainly focusing on easily accessible commercial releases on all major mobile and desktop platforms. Secondly, we tested all games that were free to play and searched for reviews, gameplay videos and product descriptions to confirm what programming topics each game covers. Additionally, for games that had no age group specified by developers, we gave our own suggestions based on the use of block-based or text-based programming, and how complex the game is for a novice. Thirdly, we did not limit our review to games that were developed for educational purposes, thus accepting also games that use programming techniques as part of the gameplay. We filtered out games that were no longer supported (over a year since the last update) and that were not accessible. We accepted 11 games listed by Vahldick *et al.* (2014) based on these criteria and discovered 18 new games, thus making our final set to comprise 29 games.

*Table 2: Programming topics in the curricula at grades 1–6 (*indirectly mentioned in the curriculum)*

	Australia	*Estonia*	*Finland*	*France*	*Israel*	*US*	*UK*
Algorithms	x			x	x	x	x
Computational thinking							x
Basic constructs (loops, conditionals, variables)	x					x	x
Creating games							
Data structures							
Digital logic							
Input/output			x				x
Interdisciplinary/robotics		x			x		
Real programming language (object-oriented programming)							
Software design, modularity							
Block-based programming languages	x	x	x*	x	x	x*	x*

Results

K-12 Curricula

We reviewed the K-12 curricula from five countries (Australia, the United Kingdom, Finland, France and Israel), an unofficial curriculum guidelines created for programming education in the United States and programming education guidelines created in Estonia by the Information Technology Foundation for Education. Our intention was to identify similarities and differences in the approaches to programming education. The largest differences between the presentations of the curricula were on how verbose the descriptions of the objectives for each grade were, and on the definitions used for describing programming practices.

Several different terms were used in definitions in the reviewed curricula: Computational Thinking (the United Kingdom, the United States), Algorithmic Thinking (Finland, Israel) and Algorithms (Australia). Though the terms have undeniable similarities, it is challenging to confirm to what extent each country's use of the same term has the same meaning attached to it. This was also noted by Balanskat and Engelhard (2015) while mapping out different definitions used in European countries. When creating Tables 1 and 2, we avoided altering the text used by the curricula, with the exception graphical programming tools. Whenever graphical tools were mentioned, we assumed that block-based tools were meant.

In the following section, we shortly describe the reviewed curricula, and their comparison by grades 1–6 and 7–12.

Australia

The Australian curriculum was released in 2015 and transition to it started in 2016. It is among the most verbose of the analyzed curricula. At grades 3–4, students start to program using visual programs (presumably block-based programming) and the transition to real programming languages is made at grades 7–8. At grades 9–10, a further requirement for covering object-oriented programming is specified.

Estonia

In Estonia, the Information Technology Foundation for Education runs a program called ProgeTiger since 2012 with a purpose to enhance learners' technological literacy and digital competence and targeting at teachers in preschools, primary and vocational education. Additionally, ProgeTiger aims to increase children's interest toward engineering. ProgeTiger has released a program guide for 2015–17 (Information Technology Foundation for Education (Estonia), 2015), with activities aimed at three proficiency levels. These levels are not directly related to formal education levels, thus to fit their guidelines to our comparison tables, we divided the three proficiency levels as follows: Basic—grades 1–6, Intermediate and Advanced—grades 7–12.

The United Kingdom

Programming was included in the United Kingdom's curriculum in 2014 when other revisions were also implemented. The curriculum is divided into five Key stages, each of which represents a specific year group—Key stage 1: Ages five to seven (Years 1–2); Key stage 2: Ages 7–11 (Years 3–6); Key stage 3: Ages 11–14 (Years 7–9); Key stage 4: Ages 14–16 (Years 10–11); Key stage 5: Ages 16–19 (Years 12–13).

*Table 3: Programming topics in the curricula at grades 7–12 (*indirectly mentioned in the curriculum)*

	Australia	*Estonia*	*Finland*	*France*	*Israel*	*US*	*UK*
Algorithms	x			x	x	x	x
Computational thinking			x			x	x
Basic constructs (loops, conditionals, variables)			x	x	x	x	
Creating games		x					
Data structures	x					x	x
Digital logic							x
Input/output							
Interdisciplinary/robotics		x					
Real programming language (object-oriented programming)	x	x	x	x	x*		x
Software design, modularity					x		
Block-based programming languages				x			

Finland

The Finnish curriculum, which implemented programming education in fall 2016, was the least verbose of the reviewed curricula. Programming is not defined as an independent subject; it is integrated into other subjects, such as mathematics. Moreover, instead of listing precise requirements, short sentences are used in the curriculum descriptions. For instance, the curriculum explains about including programming to a mathematics class (grades 3–6): "Encourage the student to create operation instructions in a graphical programming environment" (Ministry of Education (Finland), 2014). This means that the teacher is relatively free to decide how to approach programming education.

France

In 2015, France published their new curriculum that included specifications for programming education at primary schools. The curriculum was subsequently implemented in 2016. The French curriculum description is fairly verbose, and unlike all other reviewed curricula, it is divided into four cycles—Cycle 1: pre-school; Cycle 2: grades 1–3; Cycle 3: grades 4–6; and Cycle 4: grades 7–9. Programming education is set to start from Cycle 2 and the curriculum first instructs the use of block-based programming tools, which are switched to text-based programming tools in Cycle 4.

Israel

Israel introduced programming as a subject in 1976 and the country has been revising the curriculum since. Currently, Israel has programming education integrated into all levels of K-12. However, at grades 1–6, only simple introductory steps toward programming are taken. At grades 7–12, programming pedagogy becomes more serious and detailed. Programming education in these grades is divided into four modules. The first module (grade 7) emphasizes the need to make programming enjoyable by block-based programming (Bargury *et al.*, 2012). The second module (grade 8) focuses on the use of spreadsheets for scientific research. In the third module (also grade 8), which is elective, programming is taught with a real programming language, either using robots or client-side programs, depending on the budget and the teacher. It is notable that client-side program development focuses on HTML5 and JavaScript.

Table 4: Basic information of games suitable for grades 1–6

Name	*Year*	*Genre*	*Platform*	*Languages*
Algotica Iterations	2017	Puzzle	Windows	Eng, Rus, Chi
Cargo-Bot	2012	Puzzle	IOS	Eng
Catos Hike	2014	Puzzle	IOS	Eng
Daisy the Dino	2016	Puzzle	IOS	Eng
Kodable	2014	Puzzle	Web, Windows, IOS, Android	Eng
Move the Turtle	2012	Puzzle	IOS	Eng, Pl, Es
RoboLogic	2009	Puzzle	IOS	Eng, Ger
Robozzle	2009	Puzzle	Web, IOS, Android, Kindle Fire	Eng
Ruby Warrior	2008	Adventure	Web	Eng
CodeMonkey	2014	Puzzle	Web	Eng
Light-Bot	2008	Puzzle	Web, Android, IOS	30
CodeCombat	2013	Role-playing game, adventure, puzzle	Web	50
Codespark	2016	Puzzle	Android, IOS	Eng
Hack'n'Slash	2014	Adventure	Windows	Eng
Machineers	2013	Adventure, puzzle	Windows, Android	8
Turtle Academy	2011	Puzzle	Web	Eng
Robot School. Programming Game	2015	Puzzle	OS X Android	Eng
Lobot—Robot Programming	2016	Puzzle	Android	Eng, Bahasa
Save the animals: Coding Game	2016	Puzzle	Android, IOS	Eng, Chi
Digital Puppet—Programming	2016	Puzzle	Android	Eng
Coding Pirates Game	2016	Puzzle	Web, iOS, Android	Eng, Dan, Ger
Run Marco!	2016	Puzzle	Web, iOS, Android, Kindle	30
Karel Coding: Code Hour	2016	Puzzle	Android	Eng
Minecraft: Education Edition + Code Builder	2016	Simulation, puzzle	Windows, OS X	Eng
Sprite Box	2017	Puzzle	Web, iOS, Android	Eng
Tynker Games	2016	Puzzle, Adventure	Web	Eng

The United States

In the United States, curricula are defined at individual states. The K-12 Computer Science Framework (Alano *et al.*, 2016), used in our review as a representative of the United States curriculum. This framework is prepared by several non-profit organizations, such as the ACM, Code.org and the National Math and Science Initiative, to provide curriculum guidelines for computer science education in the United States. Due to the framework's nature of being a collection of guidelines, it is quite verbose. The focus areas are divided into five categories: Algorithms,

Variables, Control, Modularity and Program. The guidelines for grades 1–2 only include the first four categories and do not expect actual programming to be done. The framework gradually adds objectives and programming activities to the categories as students get older.

Curricula Comparison

Tables 2 and 3 present a comparison of the curricula contents divided into grades 1–6 and 7–12, respectively. In many countries, there is a trend of starting programming education at lower grades using block-based programming tools, although specific tools were often not given. The block-based programming tools were mainly recommended for the first six grades for introducing the working principles of computers and the basic programming constructs, such as loops, conditionals and variables. One notable difference was that some countries' curricula propose to start this at a later stage. For instance, France introduces programming topics at Cycle 4, which encompasses grades 4–6. From grade seven onwards, the focuses in the reviewed curricula are aligned for most parts, with most of the curricula introducing real programming languages, with the exception of the United States. Only Israel had defined the programming languages to be used (HTML5 and JavaScript); other curricula simply stated the requirement to use a text-based general-purpose programming language.

Programming Games

There are more games that lend themselves to teaching programming that can feasibly be tested by teachers. This section aims to give suggestions for games to be used in programming education. Tables 4 and 5 describe basic information of 29 games suitable for grades 1–6 and 7–12, respectively. In Tables 6 and 7, we have analyzed the same games by programming topics and approaches (text-based or block-based) for grades 1–6 and 7–12, respectively. An asterisk marks those programming topics that are not fully utilized. Appendix 1 contains links to game websites and notes whether they can be acquired for free. Our hope is that by providing these results, purposeful use of games to facilitate programming education in K-12 can be increased.

We assigned the games to one or more of the following genres: Action, Adventure, Role-playing, Strategy, Puzzle, Sandbox, Simulation and Sport. Under these genres lie a host of sub-genres and their combinations that can be used to define the alignment of games more precisely. Different genres support different types of learning goals. For example, on one hand, puzzle games naturally lend themselves to practicing particular problems in a structured manner (eg, levels in the game increase the difficulty of puzzles), which can be then be used to practice particular skills (eg, loops, recursion). On the other hand, puzzle games offer a constrained environment with levels that are analogous to isolated exercises. Conversely, simulation games offer a more free-form type of play, thus enabling creative exercises that can be solved in multiple ways over a longer period of time. This approach typically requires that the educator creates or finds suitable scenarios where the learner can experiment with different solution possibilities.

Table 5: Basic information of games suitable for grades 7–12

Name	*Year*	*Genre*	*Platform*	*Languages*
Colobot	2016	Real-time strategy	Windows	Eng, Ger
Human Resource Machine	2015	Puzzle	Windows, Linux, OSX	17
Blockly Games	2011	Puzzle	Web	50

Table 6: Programming topics and approaches in games suitable for grades 1–6

Name	*Variables*	*Conditionals*	*Loops*	*Methods*	*Block-/Text-based*
Algotica Iterations	x*	–	x		Block-based
Cargo-Bot	–	x	x	x	Block-based
Catos Hike	–	x	x		Block-based
Daisy the Dino	–	x	x	x	Block-based
Kodable	x	x	x	x	Block-based
Move the Turtle	x	x	x		Block-based
RoboLogic			x	x	Block-based
Robozzle		x	x	x	Block-based
Ruby Warrior	x*	x	x	x	Text-based
CodeMonkey	x	x	x	x	Text-based
Light-Bot	–	–	x	x	Block-based
CodeCombat	x	x	x	x	Text-based
Codespark		x	x	x	Block-based
Hack'n'Slash	x	x	x	x*	–
Machineers		x	x	x*	–
Turtle Academy	x	x	x		Text-based
Tynker Games	x	x	x	x	Block-based
Sprite Box	x*	–	x	x	Block-based
Robot School. Programming Game	x	x	x	x	Block-based
Lobot—Robot Programming	–	x	x	x	Block-based
Save the animals: Coding Game	–	x	x		Block-based
Digital Puppet—Programming	–	–	x	x	Block-based
Coding Pirates Game	–	x	x	–	Block-based
Run Marco!	–	x	x	–	Block-based
Karel Coding: Code Hour	x	x	x	x	Text-based
Minecraft: Education Edition + Code Builder (MinecraftEdu)	x	x	x	x	Block-based

Table 7: Programming topics and approaches in games suitable for grades 7–12

Name	*Variables*	*Conditionals*	*Loops*	*Methods*	*Block-/Text-based*
Colobot	x	x	x	x	Text-based
Human Resource Machine	x	x	x	x*	Block-based
Blockly Games	x	x	x	x	Block-based
Minecraft: Education Edition + Code Builder (MinecraftEdu)	x	x	x	x	Block-based

Programming is presented in a similar fashion in nearly all of the games, with the character being manipulated by a string of simple commands that either run in a loop or execute once. The biggest difference among these games is whether the commands available are shown as images, or the user has to write them. Only five games added some variation to this: MinecraftEdu, Sprite Box, Colobot, Hack'n'Slash and Machineers. MinecraftEdu handles programming in the familiar fashion of using blocks, but the player can move their avatar freely without coding; the code manipulates a separate agent. In Colobot, the player uses scripts to automate machines. Hack'n'Slash is more about variable manipulation.

Discussion

Most of the reviewed curricula first introduce the concepts of programming to students through play and block-based programming. Terminology varied slightly, with most countries referring to high-level concepts such as algorithms, computational thinking and algorithmic thinking. The Finnish curriculum was least descriptive, simply stating high-level goals such as "encouraging children to create programs with a visual programming environment."

There are several games that are meant for children to understand the basic command sequence. Some of these are more complex than others, and most cover also the basics, such as loops and conditionals. Variables were usually covered either partially or using metaphors, like boxes of different color in Cargo-Bot.

A clear division could be seen in the types of programming games. Games focusing on younger players were mostly rather simplistic, such as Cranes. We struggled to find games that are suitable to players who have a grasp of basic concepts, but have yet to master complex programming skills. One game that does somewhat fill the requirement is MinecraftEdu combined with Code Builder. Games that do show promise of handling both introductory and intermediate levels of programming were added to both lists (MinecraftEdu and Ruby Warrior).

Finally, we discovered several complex games (eg, Screeps,[8] TIS-100,[9] Shenzen I/O,[10] Codefight,[11] CodinGame,[12] Robocode,[13] Untrusted[14] and Elevator Saga[15]) that require a higher level of programming skills, designed to challenge people who program regularly. These could be used at schools to provide practice to highly skilled students.

Listing programming education games and what programming topics they cover does part of the work for educators. However, how these games should be introduced in a classroom environment is another challenge. We are aware that educators are already swamped with their duties (Becker, 2007). Thus, one possibly efficient way to utilize these games would be through incidental learning. Essentially, educators could suggest a suitable game to students to play at their own time. These games could support learning processes by covering the relevant topics in an engaging format.

[8]https://screeps.com/

[9]https://store.steampowered.com/app/370360/TIS100/

[10]https://store.steampowered.com/app/504210/SHENZHEN_IO/

[11]https://codefights.com/

[12]https://www.codingame.com/start

[13]https://robocode.sourceforge.net/

[14]https://alexnisnevich.github.io/untrusted/

[15]https://play.elevatorsaga.com/

Table 8: Limitations and remedies

Limitation	*Remedy suggestion (future work)*
Majority of games are short-form	Utilize game platforms (e.g. MinecraftEdu) that allow educators create new content
Lack of formal training programs for using digital games in classrooms	Develop a short course during which teachers learn about educational digital games and how they can be connected to curricula
Although many programming games cover topics presented in curricula, learning games in general are not well aligned with curricula	Conduct further studies on the suitability of games to specific curriculum subjects; use services such as Teacher Gaming that provide curriculum-aligned versions of commercial games (https://store.teachergaming.com/)
The mapping between grade groups and topics could be done differently depending on the context	Curriculum developers and educators should consider their contextual requirements and adjust the mapping as needed
One topic may be covered by several games	Educators should get familiar with various games to identify the best match for their pedagogical setting and students' preferences
A game covering all topics might not be engaging and pedagogically effective	Test and choose games that match the curriculum objectives for specific topics and the implementation requirements in a given context
There remain gaps in the topic coverage of programming education games, especially for older students	Develop programming games that are aimed at more advanced students
As this study focused on reviewing the curricula documents, real life experiences on implementing the curricula were not investigated	Interview teachers from the respective countries to find out more about the actual situations at schools
Curricula are often provided as guidelines with little instructions for implementation	Devise implementation guidelines by investigating how programming curricula have been implemented in different countries and what the results have been

It should be stressed that we do not see programming games as a silver bullet capable of replacing Scratch-like tools. These games could rather be viable tools in helping to engage with computer science outside schools, alongside graphical programming tools.

Depending on the approach of a given programming game, it can be argued to support various learning theories apart from being an agent for motivation and engagement (see Table 1). The general requirement for the player to actively participate in the game and construct code snippets using block-based or text-based language supports constructivism. Aligned with this, some programming games employ problem-based learning strategies with multiple acceptable solutions to a given problem, which may be ill-defined (eg, MinecraftEdu, Code Combat, CodeMonkey). Although independent learning-by-doing gameplay is the most common approach in the reviewed games, some games were specifically designed to support teaching at schools (eg, MinecraftEdu, Save the Animals, Code Combat, CodeMonkey). These products reflect instructionism, as they can be more teacher-driven. Finally, lack of games that provide the possibility for social learning among the players was evident, as the only game that has this clearly built in is MinecraftEdu.

The emergence of products with teachers in mind is not that surprising, as companies move to fill the demand for programming education games for children. Some of these products come with the benefit of having ready-made tools for the teachers to utilize for additional content

production. Moreover, some games (eg, MinecraftEdu, CodeMonkey, Kodable) offer features for student observation and management, thus helping the teacher efficiently orchestrate the learning environment.

The results of this study improve on Vahldick *et al*.'s (2014) comparison of programming games in several ways. Firstly, Vahldick et al. included several games that were based on research studies and games that were not available anymore; all games in our review are acquirable at the moment of writing this. Secondly, unlike the previous study, we also included curricula in the review. Thirdly, we provide suggestions on the games' fitness to age groups and topic categories. Despite these differences, we confirmed some of the findings of Vahldick et al.: no adaptation to individual learners, little support for multiple programming languages and only a few cases where scaffolding hints are provided during gameplay.

The survey by Takeuchi and Vaala (2014) listed some important limitations that currently exist in integration of digital games into teaching: (i) the majority of the games used by the teachers are short-form; (ii) teachers are learning to teach with digital games via informal means, rather than through formal training programs, thus potentially leaving them unexposed to a wider pedagogical selection; and (iii) games are often not curriculum-aligned. These limitations, together with the limitations emerging from the results of our study, are listed in Table 8. We also provide remedy suggestions that may serve as future work in this area. As the number of available programming games increases, we need long-term pedagogical evaluations to identify the most suitable games for different pedagogical settings. The evaluation studies presented in Table 1 are a good starting point toward this.

Conclusion

We reviewed and compared how seven countries—Australia, Estonia, the United Kingdom, Finland, France, Israel and the United States—approach programming education at primary and secondary schools. We also reviewed games that are easily acquirable and utilize programming in some form in gameplay. Additionally, we created tables that summarize the games, what programming topics they cover and appropriate age groups for the games. The results of this study can be used to overcome some of the issues related to integrating educational games into teaching, especially the lack of correspondence between educational games and curricula (Takeuchi & Vaala, 2014).

The coming generations have a hands-on approach to technology as it revolves around them from early childhood. Even though many of them may never work as programmers, it is important to ensure that they will have a nominal understanding of what makes the technology around them tick. Using games in conjunction with education is by no means a complete solution to achieve this goal, but it might be a viable part of it.

Statements on open data, ethics and conflict of interest

All data for this study were gathered from publicly available government documents, websites and academic articles. There are no conflicts of interest.

References

ACM and IEEE. (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*.

Alano, J., Lash, T., Babb, D., Lee, I., Bell, J., Lyman, C., & Weintrop, D. (2016). *K–12 computer science framework*. Retrieved from https://www.k12cs.org

Australian Curriculum Assessment and Reporting Authority. (2015). *ACARA: Digital technologies: Sequence of content*. Retrieved from https://www.australiancurriculum.edu.au/f-10-curriculum/technologies/digital-technologies/pdf-documents/

Balanskat, A., & Engelhardt, K. (2015). *Computing our future: Computer programming and coding priorities, school curricula and initiatives across Europe*. Brussels: European Schoolnet.

Bargury, I. Z., Haberman, B., Cohen, A., Muller, O., Zohar, D., Levy, D., & Hotoveli, R. (2012). Implementing a new computer science curriculum for middle school in Israel. In *Frontiers in Education* (pp. 1–6). Seattle, WA: IEEE.

Becker, K. (2007). Digital game-based learning once removed: Teaching teachers. *British Journal of Educational Technology, 38*(3), 478–488.

Boyle, E. A., Hainey, T., Connolly, T. M., Gray, G., Earp, J., Ott, M., & Pereira, J. (2016). An update to the systematic literature review of empirical evidence of the impacts and outcomes of computer games and serious games. *Computers & Education, 94*(March), 178–192.

Bromwich, K., Masoodian, M., & Rogers, B. (2012). Crossing the game threshold: A system for teaching basic programming constructs. In *International Conference of the NZ chapter of the ACM's special interest group on human-computer interaction* (p. 56). Dunedin, New Zealand: ACM.

Conte, S., Hamblen, J., Kehl, W., Navarro, S., Rheinboldt, W., Young, D., & Atchinson, W. F. (1965). An undergraduate program in computer science—preliminary recommendations. *Communications of the ACM, 8*(9), 543–552.

Department for Education (UK). (2013a). *Computing programmes of study: Key stages 1 and 2*. Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239033/PRIMARY_national_curriculum_-_Computing.pdf

Department for Education (UK). (2013b). *Computing programmes of study: Key stages 3 and 4*. Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239067/SECONDARY_national_curriculum_-_Computing.pdf

Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From game design elements to gamefulness. In *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments* (p. 9). New York, NY: ACM Press.

Eagle, M., & Barnes, T. (2009). Experimental evaluation of an educational game for improved learning in introductory computing. *ACM SIGCSE Bulletin, 41*(1), 321.

Esper, S., Foster, S. R., & Griswold, W. G. (2013). CodeSpells: Embodying the methaphor of wizardry for programming. In *ACM conference on Innovation and technology in computer science education*. Canterbury, UK: ACM

Information Technology Foundation for Education (Estonia). (2015). *ProgeTiger Programme*. Retrieved from https://www.hitsa.ee/it-education/educational-programmes/progetiger

Heintz, F., Mannila, L., & Farnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in K-12 education. In *Proceedings of the 2016 Frontiers of Education Conference*, IEEE, Erie, PA, USA (pp. 1–9).

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A survey of programming environments and languages for novice programmers. *Science, 37*(2), 83–137.

Lee, M. J., & Ko, A. J. (2011). Personifying programming tool feedback improves novice programmers' learning. *Proceedings of the Seventh International Workshop on computing education research*, 109–116. https://doi.org/10.1145/2016911.2016934

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on computer science education*, ACM, Milwaukee, Wisconsin, USA (pp. 346–350).

Liu, C., Cheng, Y., & Huang, C. (2011). The effect of simulation games on the learning of computational problem solving. *Computers & Education, 57*(3), 1907–1918.

Lode, H., Franchi, G. E., & Frederiksen, N. G. (2013). Machineers: Playfully introducing programming to children. In *Extended abstracts on human factors in computing systems*. New York, NY: ACM Press, pp. 2639–2642.

Long, J. (2007). Just for fun: Using programming games in software programming training and education —A field study of IBM Robocode Community. *Journal of Information Technology Education, 6*, 279–290.

Marsick, V. J., & Watkins, K. E. (2001). Informal and incidental learning. *New Directions for Adult and Continuing Education, 2001*(89), 25–34. Retrieved from https://web.a.ebscohost.com.libezproxy.open.ac.uk/ehost/pdfviewer/pdfviewer?sid=38ad36e0-81e7-47fb-a501-16656c1e131c@sessionmgr4002&vid=3&hid=4104

Mathrani, A., Christian, S., & Ponder-Sutton, A. (2016). PlayIT: Game based learning approach for teaching programming concepts. *OP-Journal of Educational Technology & Society, 19*(2), 5–17.

Ministry of Education (Finland). (2014). *Lisäopetuksen Opetussuunnitelman Perusteet 2014* (4th ed.). Helsinki: Next Print Oy, Helsinki 2016. Retrieved from www.oph.fi

Ministry of Education (France). (2015). *Projet de programmes pour les cycles. Conseil Superieur Des Programmes*. Retrieved from https://cache.media.education.gouv.fr/file/09_-_septembre/22/9/programmes_cycles_2_3_4_469229.pdf

Ministry of Education (Israel). (2016). *Curriculum in computer science and robotics without appendices*. Retrieved from https://cms.education.gov.il/EducationCMS/Units/MadaTech

Ministry of Education and Science (Korea). (2015). *Sw* 중심사회를 위한 인재양성 추진계획. Retrieved from https://www.moe.go.kr/boardCnts/view.do?boardID=294&boardSeq=60077&lev=0

Plass, J. L., Homer, B. D., & Kinzer, C. K. (2015). Foundations of game-based learning. *Educational Psychologist, 50*(4), 258–283.

Price, T. W., & Barnes, T. (2015). Comparing textual and block interfaces in a novice programming environment. In *International Conference on International Computing Education Research* (pp. 91–99). Omaha, Nebraska: ACM.

Project Tomorrow. (2016). From print to pixel: The role of videos, games, animations and simulations within K-12 education: Speak Up 2015 national findings congressional report, 14.

Resnick, M. (2004) Edutainment? No Thanks. I Prefer Playful Learning. MIT Media Laboratory, 4.

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., & Silver, J. (2009). Scratch. *Communications of the ACM, 52*(11), 60.

Takeuchi, L. M., & Vaala, S. (2014). *Level up learning: A national survey on teaching with digital games*. New York, NY: Games & Learning.

Vahldick, A., Mendes, A. J., & Marcelino, M. J. (2014). A review of games designed to improve introductory computer programming competencies. In *IEEE Frontiers in Education Conference* (pp. 1–7). IEEE.

Weintrop, D., & Wilensky, U. (2015). To Block or not to block, that is the question: Students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children—IDC '15* (pp. 199–208). New York, NY: ACM Press. https://doi.org/10.1145/2771839.2771860

APPENDIX A

	Name	*Free*	*Homepage*
1	Algotica Iterations	No	https://store.steampowered.com/app/593330/Algotica_Iterations/
2	Blockly Games	Yes	https://blockly-games.appspot.com/
3	Cargo-Bot	Yes	https://twolivesleft.com/CargoBot/
4	Catos Hike	No	https://hwahba.com/catoshike/
5	CodeCombat	No	https://codecombat.com/
6	CodeMonkey	No	https://www.playcodemonkey.com/teachers
7	Codespark	No	https://codespark.org/
8	Coding Pirates Game	Yes	https://www.codingpiratesgame.com/
9	Colobot	Yes	https://colobot.info/
10	Daisy the Dino	Yes	https://itunes.apple.com/us/app/daisy-the-dinosaur/id490514278?mt=8
11	Digital Puppet—Programming	Yes	https://play.google.com/store/apps/details?id=com.takoyaking.digitalpuppet
12	Hack'n'Slash	No	https://www.hacknslashthegame.com/
13	Human Resource Machine	No	https://tomorrowcorporation.com/
14	Karel Coding: Code Hour	No	https://nclab.com/courses/karel-coding/
15	Kodable	Yes	https://www.kodable.com/
16	Light-Bot	No	https://lightbot.com/
17	Lobot—Robot Programming	Yes	https://play.google.com/store/apps/details?id=com.gemuku.lobot.android
18	Machineers	No	https://www.machineers.com/
19	Minecraft: Education Edition	No	https://education.minecraft.net &https://education.minecraft.net/trainings/code-builder-for-minecraft-education-edition/
20	Move the Turtle	No	https://www.geekkids.me/
21	RoboLogic	No	https://www.digitalsirup.com/app/robologic/?lang=en
22	Robot School. Programming Game	No	https://www.robotschoolapp.com/
23	Robozzle	Yes	https://robozzle.com/
24	Run Marco!	Yes	https://www.allcancode.com/hourofcode
25	Ruby Warrior	Yes	https://www.bloc.io/ruby-warrior#/
26	Save the animals: Coding Game	Yes	https://fifthwisdom.com/games/save-the-animals/
27	Sprite Box	No	https://spritebox.com/
28	Turtle Academy	Yes	https://turtleacademy.com/project/doc/en
29	Tynker Games	Yes	https://www.tynker.com/