Vilnius university
Institute of Data Science and Digital Technologies
L I T H U A N I A

INFORMATICS ENGINEERING (T007)

MODELING THE SYSTEM FOR INTERACTIVE TASKS DEVELOPMENT: ENGAGEMENT TAXONOMY FOR INTRODUCTORY PROGRAMMING TOOLS

Tomas Šiaulys

October 2020

Technical Report DMSTI-DS-T007-20-08

VU Institute of Data Science and Digital Technologies, Akademijos str. 4, Vilnius
LT-08412, Lithuania
www.mii.lt

Abstract

One of the main ideas behind these introductory programming environments is introducing basic programming concepts more effectively by incorporating different visualization strategies. There have been attempts to classify introductory programming tools, however, certain critical aspects have not yet been discussed within the existing classifications, especially those related to user engagement in the programming environment. In this paper we introduce an engagement taxonomy for introductory programming tools (ETIP) built on a concept of engagement taxonomy for software visualization and previous classifications of programming learning tools. The new taxonomy is then used to inclusively review introductory programming environments for secondary education used today. Our review illustrates how majority of introductory programming tools do not fully explore the ways visualizations could help with tackling the problems of beginner programming comprehension. This report includes excerpts from the paper to be presented at the ISSEP 2020 conference.

Keywords: Introductory programming, software visualization, engagement taxonomy

Contents

1 Introduction

Before working on systems for interactive tasks development, a clear picture about the present systems being used at schools is needed with a particular attention to introductory programming environments. These systems at present are the main tools used to introduce students to computational thinking. The common difference between the introductory programming tools for higher education and secondary education is that the tools for younger audience tend to be more visual and more tangible.

This study focuses on the role of visualization in improving student comprehension of the basic concepts and practices of programming in introductory programming environments for K-12 education. It is important to note that visual programming environments that are widely used across the secondary education, employ both visualization types - programming code visualizations as well as visualizations of program execution. While the role of code visualization has been extensively studied (Xu et al., 2019, Weintrop and Wilensky, 2019), the impact of execution-time visualizations on students' comprehension is less clear. This report includes excerpts from the paper to be presented at the ISSEP 2020 conference.

2 Related work

2.1 Misconceptions of novice programmers

According to literature review by Qian and Lehman (2017) novice programmers tend to have misconceptions about most of the basic programming concepts including variables, conditionals, parameters, loops and even the idea of states and sequential execution. Tracing programs step by step is probably the most important strategy to come over these misconceptions, however, Lister et al. (2004) and Simon (2011) show that it's the ability that most of the novices struggle with. Grover and Basu (2017) show that even students, who completed introductory visual programming courses, keep misconceptions about the basic programming concepts. Authors argue, that even though visual programming environments like *Scratch*, do help learners with the syntactic aspects of programming, conceptual and strategic aspects of programming require additional effort. Role of pedagogy in overcoming these misconceptions also leaves many unknowns as instructors tend to show weak understanding about students' mistakes (Brown and Altadmri, 2014). One of the most popular approaches in trying to tackle the problems of student comprehension is visualization. Visualizations can make abstract programming concepts and hidden automatic runtime processes visible and controllable.

2.2 Taxonomies of programming learning tools

Taxonomy of programming environments and languages for novice programmers by Kelleher and Pausch (2005) is the most cited attempt to classify introductory programming tools. While proposed taxonomy could be criticized for vague descriptions and overlapping categories, it provides an overall view of the key aspects in making programming accessible for novices (K-12 as well as higher education). Taxonomy suggested the category of code visualization with an emphasis of avoiding syntax errors, as well as the category for visualizing program execution, including examples of strategies that programming environments use for visualization. Authors argue that different visualization techniques are similar to "the supports found in many

debuggers". Article provides brief descriptions of 86 systems, some dating back to 1960's.

Built upon the work of Kelleher and Pausch, Taxonomy of programming learning tools (Saito et al., 2017) attempts to describe each learning tool across 11 categories. However, due to the lack of clarity in category definitions, it is not fully clear what certain categories of the taxonomy represent. While taxonomy doesn't explicitly focus on visualization of execution, *support to understand programs* category provides some information about visualization strategies of the learning tools. Proposed taxonomy is then used to classify 43 introductory programming environments designed for kids.

João et al. (2019) used similar approach analyzing 26 most popular block-based and visual programming apps across 27 categories with a focus on pedagogical usefulness. As with the previous classifications, some categories are vaguely defined, particularly those concerning the execution environment.

Different approach was taken by Duncan et al. (2014) in loose classification of 47 tools for introductory programming according to the difficulty level, concepts being introduced, as well as student age, without focusing on visualization. Authors introduce heuristics to classify introductory programming tools into 5 approximately defined categories leaving the classification rather subjective.

2.3 Engagement taxonomies

Metaanalysis of visualization systems by Hundhausen et al. (2002) concludes that visualization proved to be effective in only 13 out of 28 studies and that different learner engagement forms were connected to the effectiveness of visualizations. Following this work Naps et al. (2002) introduced original engagement taxonomy (OET) for program visualization, which defined six different forms of learner engagement in the context of using visualization tools: *no viewing*, *viewing*, *responding*, *changing*, *constructing* and *presenting*. It has been hypothesized that increasing level of engagement would result in better learning outcomes and that the mix of different forms of engagement would be more beneficial than a single engagement form. A survey partially supporting OET was carried out by Urquiza-Fuentes and Velázquez-Iturbide (2009) regarding program visualization and algorithm animation systems.

Building upon the original engagement taxonomy Myller et al. (2009) and Sorva et al. (2013) attempted to improve the categorization of engagement levels. Hypothesizing that collaborative activities of the students and engagement levels are correlated, Myller et al. introduced an extended engagement taxonomy (EET) defining 10 engagement levels: *no viewing*, *viewing*, *controlled viewing*, *entering input*, *responding*, *changing*, *modifying*, *constructing*, *presenting* and *reviewing*. Sorva et al. (2013) criticized OET and EET for mixing different engagement forms and introduced a revised 2-dimentional engagement taxonomy (2DET) differentiating between direct engagement dimension: *no viewing*, *viewing*, *controlled viewing*, *responding*, *applying*, *presenting*, *creating*; and content ownership dimension: *given content*, *own cases*, *modified content*, *own content*. Then 22 systems were classified into categories according to 2DET.

3 Engagement taxonomy for introductory programming tools (ETIP)

Previously defined taxonomies of engagement in software visualization, even though have theoretical basis, are still problematic in using them practically for classification and research of program visualization systems. Attempts to classify

programming environments tend to distribute all the systems across two or three groups and for the further analysis other factors have to be chosen. Sorva et al. (2013) found that 18 out of 22 generic program visualization systems fall under controlled viewing engagement level and own content in ownership dimension of 2DET. Hence understanding what is meant by controlled viewing may be the key in explaining user engagement in using visualization tools. This is especially relevant when discussing introductory programming tools for K-12 since these tools tend to employ visualizations different from the ones used in higher education that were analyzed using previous taxonomies.

Another argument for introducing a new taxonomy for introductory programming is that most of the visualizations cannot be categorized as specially designed for presenting (Sorva, 2013) and even if they were, this would give us more information about the use of visualization in social interactions rather than about individual interaction with the visualization. This remark is consistent with the survey of Urquiza-Fuentes and Velázquez-Iturbide (2009) classifying all the systems that support *changing* level of OET, as well supporting *presenting* level of engagement.

We propose a new engagement taxonomy for introductory programming tools (ETIP) focusing on student engagement in studying the visual execution of programs (**Table 1**). The lower levels of engagement (*no viewing* and *viewing)* in our proposed taxonomy are the same as in 2DET. *Following* is added and *controlled viewing* is split into three levels of engagement in respect to tracing the execution of a program in visual environment. Highlighting the code during execution was suggested by Naps et al. (2002) in the context of algorithm visualization. Nevertheless, this might not be enough to engage learners into tracing the visualization, especially when the certain steps being executed are too complex or the bugs are present. Tracing the execution of visualization can be partially helped by *changing the speed*. Finally, engagement levels of *executing step-by-step* and *rewinding* are adapted from the requirements for the algorithm visualization systems (Karavirta & Shaffer, 2016). For the reasons described above, *presenting* level was not included. *Creating*, *responding* and *applying* levels were omitted as well for being not consistent with the concept of visual student-written program execution. Given that all of the introductory programming environments involving program visualization are expected to promote content ownership of the students, content ownership dimension is as well omitted in the presented taxonomy.

Table 1. The categories of the engagement taxonomy for introductory programming tools

Level of engagement		**Description**
No viewing		There is no visualization, only material in textual format
Viewing		The learner views a visualization with no control over execution of visualization, can only zoom/navigate the environment of program execution.
Following		The learner views the visualization with the executed code being highlighted .
Controlled viewing	Changing the speed	The learner can change the speed of visualization being executed.
	Executing step by step	The learner can view the visualization being executed step-by-step.
	Rewinding	The learner can rewind the visualization at any time during the visualization.

4 Method to select tools

To answer the second research question, as many as possible of all the available programming environments for secondary education were categorized according to the highest level of user engagement of ETIP they allow. All the tools from the previous classifications of programming environments for K-12 education (Duncan et al., 2014, Saito et al. 2017, Joao et al., 2019) were included in the list for the study as well as additional environments from the web search. The general overview of 70 selected environments can be found in the Appendix 1. The list could not be seen as complete, since the number of introductory learning environments for K-12 education is difficult to track. However, most of the most popular visual introductory programming environments are included.

5 Results and discussion

The results from the Appendix 1 table suggest that majority of the tools cover most of the basic programming concepts with exception of the tools focusing on primary education. Also, majority of the tools use the same block model for code construction as well as the same game/puzzle activity type (*Blockly games*, *Code Studio*, etc.). Another common group of introductory programming environments are the classical turtle visualization environments (*Scratch*, *Snap!*, etc.) focusing on the motion of the sprites and drawing. The most common programming languages used for learning were Python and JavaScript. Systems based on gamification elements are just as common as tools allowing for free creation of games, animations, etc.

Appendix 2 shows that only 2 programming tools allow rewinding and only 20% of the introductory programming environments allow executing step by step level of engagement, while majority (47%) allows only viewing of program execution visualization. What is somewhat surprising is that tools created for primary education to a great extent employ low engagement levels, while the systems that fall into high engagement level categories are targeted towards older students.

The results of the study bring some new insights into K-12 programming education. First of all, it seems that in relation to such a large number of educational programming environments, most of the tools stick to the same old models. Of course, new technologies gave visualizations the quality and the possibilities never seen before, but as noted before, even though more enjoyable and emotionally engaging, visualizations are not always effective in improving the learning results. Secondly, having in mind the misconceptions of novice programmers about most of the basic programming concepts, it seems puzzling why so many tool designs do not address the issue of learners' comprehension. In particular, the ability to track the program execution, a skill at the core of understanding the basic programming concepts, seems to be unrepresented in most of the programming environments for schools. These systems could borrow strategies for program visualization from the systems targeted more at higher education. It could be argued that tracing the program execution in a visual environment involves the same strategies as tracing an algorithm visualization, while the systems for algorithm visualization often employ much richer tools for user engagement.

6 Conclusion and future work

This report introduced Engagement taxonomy for introductory programming tools (ETIP) - a model to measure learners' engagement in tracing the program execution in visual environments. There is still a lack of knowledge about the importance of

engagement levels in designing introductory programming tools as well as different types of engagement involved. Suggested taxonomy could be used for future research in studying student engagement levels in visual environments.

The results also suggest that in order to improve learners' comprehension through more engaging interactive tasks, introductory programming environments' design could be improved. This also involves the tools and libraries used in creating these interactive tasks and learning environments.

7 References

1. Brown, N.C. and Altadmri, A., 2014, July. Investigating novice programming mistakes: Educator beliefs vs. student data. In Proceedings of the tenth annual conference on International computing education research (pp. 43-50).
2. Duncan, C., Bell, T. and Tanimoto, S., 2014, November. Should your 8-year-old learn coding?. In Proceedings of the 9th Workshop in Primary and Secondary Computing Education (pp. 60-69).
3. Grover, S. and Basu, S., 2017, March. Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education (pp. 267-272).
4. Hidalgo-Céspedes, J., Marín-Raventós, G. and Lara-Villagrán, V., 2016, October. Learning principles in program visualizations: a systematic literature review. In 2016 IEEE frontiers in education conference (FIE) (pp. 1-9). IEEE.
5. Hundhausen, C.D., Douglas, S.A. and Stasko, J.T., 2002. A meta-study of algorithm visualization effectiveness. Journal of Visual Languages & Computing, 13(3), pp.259-290.
6. João, P., Nuno, D., Fábio, S.F. and Ana, P., 2019. A Cross-analysis of Block-based and Visual Programming Apps with Computer Science Student-Teachers. Education Sciences, 9(3), p.181.
7. Kelleher, C. and Pausch, R., 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys (CSUR), 37(2), pp.83-137.
8. Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppälä, O. and Simon, B., 2004. A multi-national study of reading and tracing skills in novice programmers. ACM SIGCSE Bulletin, 36(4), pp.119-150.
9. Luxton-Reilly, A., Albluwi, I., Becker, B.A., Giannakos, M., Kumar, A.N., Ott, L., Paterson, J., Scott, M.J., Sheard, J. and Szabo, C., 2018, July. Introductory programming: a systematic literature review. In Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (pp. 55-106).
10. Mladenović, M., Boljat, I. and Žanko, Ž., 2018. Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. Education and Information Technologies, 23(4), pp.1483-1500.
11. Myller, N., Bednarik, R., Sutinen, E. and Ben-Ari, M., 2009. Extending the engagement taxonomy: Software visualization and collaborative learning. ACM Transactions on Computing Education (TOCE), 9(1), pp.1-27.

12. Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. and Velázquez-Iturbide, J.Á., 2002. Exploring the role of visualization and engagement in computer science education. In Working group reports from ITiCSE on Innovation and technology in computer science education (pp. 131-152).
13. Papert, S., 1980. Mindstorms: Computers, children, and powerful ideas. NY: Basic Books, p.255.
14. Qian, Y. and Lehman, J., 2017. Students' misconceptions and other difficulties in introductory programming: A literature review. ACM Transactions on Computing Education (TOCE), 18(1), pp.1-24.
15. Rijke, W.J., Bollen, L., Eysink, T.H. and Tolboom, J.L., 2018. Computational Thinking in Primary School: An Examination of Abstraction and Decom-position in Different Age Groups. Informatics in education, 17(1), p.77.
16. Saito, D., Sasaki, A., Washizaki, H., Fukazawa, Y. and Muto, Y., 2017, November. Program learning for beginners: survey and taxonomy of programming learning tools. In 2017 IEEE 9th International Conference on Engineering Education (ICEED) (pp. 137-142). IEEE.
17. Saito, D., Sasaki, A., Washizaki, H., Fukazawa, Y. and Muto, Y., 2017, November. Program learning for beginners: survey and taxonomy of programming learning tools. In 2017 IEEE 9th International Conference on Engineering Education (ICEED) (pp. 137-142). IEEE.
18. Simon. 2011. Assignment and sequence: why some students can't recognise a simple swap. In Proceedings of the 11th Koli Calling International Conference on Computing Education Research (Koli Calling '11). Association for Computing Machinery, New York, NY, USA, 10–15. DOI:https://doi.org/10.1145/2094131.2094134
19. Sorva, J., Karavirta, V. and Malmi, L., 2013. A review of generic program visualization systems for introductory programming education. ACM Transactions on Computing Education (TOCE), 13(4), pp.1-64.
20. Thomas, L., Ratcliffe, M. and Thomasson, B., 2004. Scaffolding with object diagrams in first year programming classes: some unexpected results. ACM SIGCSE Bulletin, 36(1), pp.250-254.
21. Urquiza-Fuentes, J. and Velázquez-Iturbide, J.Á., 2009. A survey of successful evaluations of program visualization and algorithm animation systems. ACM Transactions on Computing Education (TOCE), 9(2), pp.1-21.
22. Vieira, E.A.O., Da Silveira, A.C. and Martins, R.X., 2019. Heuristic Evaluation on Usability of Educational Games: A Systematic Review. Informatics in Education, 18(2), pp.427-442.
23. Weintrop, D. and Wilensky, U., 2019. Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. Computers & Education, 142, p.103646.
24. Xu, Z., Ritzhaupt, A.D., Tian, F. and Umapathy, K., 2019. Block-based versus text-based programming environments on novice student learning outcomes: A meta-analysis study. Computer Science Education, 29(2-3), pp.177-204.

Appendix 1

General information of introductory programming environments for K-12 sorted by the age of target audience.

Name	Code representation	Activity type	Visualization type	release date	age group	conditionals	loops	variables	functions	objects
Code Studio (courses)	blocks	game-puzzle	moving sprite, drawing	2013	4 and up	x	x	x	x	
BotLogic.us	picture-blocks	game-puzzle	moving sprite	2013	4–11					
Kodable	picture-blocks, JavaScript	game-puzzle	moving sprite	2014	4–11	x	x	x	x	x
Lightbot Jr	picture-blocks	game-puzzle	moving sprite	2014	4–8	x	x			
Cargo-bot	picture-blocks	game-puzzle	moving sprite	2012	5 and up	x	x			
Tynker	Blocks, Python, JavaScript, HTML	game-puzzle, creating	moving sprite, drawing, animation, music	2012	5 and up	x	x	x	x	x
Code avengers	Java, JavaScript, Python	game-puzzle, creating	moving sprite/drawing	2012	5 and up	x	x	x	x	x
Lego Bits and bricks	picture-blocks	game-puzzle	moving sprite	2017	5–11	x				
Move the Turtle	picture-blocks	game-puzzle	moving sprite	2012	5–11		x	x	x	
My robot friend	picture-blocks	game-puzzle	moving sprite	2013	5–12					
Cato's Hike	picture-blocks	game-puzzle	moving sprite	2012	5–12	x	x			
Junior Coder	picture-blocks	game-puzzle	moving sprite	2015	5–12	x	x		x	
ScratchJr	picture-blocks	creating	moving sprite, game design	2014	5–7	x	x			
Daisy the dinosaur	blocks	game-puzzle	moving sprite	2012	5–7	x	x			
the Foos (CodeSpark)	picture-blocks	game-puzzle	moving sprite, game design	2014	5–9	x	x			
Codable Crafts	picture-blocks	game-puzzle	moving sprite	2015	5–9	x	x			
Swift Playgrounds	swift	game-puzzle	moving sprite	2016	6 and up	x	x	x	x	x
Run Marco (All can code, hour of code)	blocks	game-puzzle	moving sprite	2014	6–12	x	x			
Codemancer	picture-blocks	game-puzzle	moving sprite	2013	6–12		x		x	
Rapid router (Blockly, Code for life)	blocks/Python	game-puzzle	moving sprite	2014	6–13	x	x	x	x	
Gamefroot	blocks	creating	game design	2017	7 and up	x	x	x	x	x
Bee-bot app / bee-bot emulator	no code	game-puzzle	moving sprite	2012	7–11					
Alice	blocks	creating	game design	1998	7–13	x	x	x	x	x
RobotMagic (Blockly)	blocks, JavaScript	game-puzzle, creating	moving sprite, simulation, game design	2017	7–16	x	x	x	x	x
Code Monkey	CoffeeScript, Python	game-puzzle	moving sprite	2014	8 and up	x	x	x	x	x
Blockly games (Blockly)	blocks	game-puzzle	moving sprite, drawing, animation, music	2012	8 and up	x	x	x	x	
mBlock (Blockly)	blocks, Python	creating	moving sprite, drawing, game design	2011	8 and up	x	x	x	x	
Pencil Code (Droplet editor)	blocks, coffeeScript, JavaScript	creating	moving sprite, drawing, animation, game design, music	2013	8 and up	x	x	x	x	x
Microsoft MakeCode Arcade	blocks	creating	game development	2020	8 and up	x	x	x	x	x
Open Roberta Lab (robot simulation)	blocks	creating	simulation	2016	8 and up	x	x	x	x	
CodeBug (electronics simulation)	blocks	creating	simulation	2015	8 and up	x	x	x		
LearnToMod (Minecraft)	blocks, JavaScript	creating	game design	2015	8 and up	x	x	x	x	x
Penjee	Python	game-puzzle	moving sprite	2014	8 and up	x	x	x	x	x
RoboMind	RoboMind	game-puzzle	moving sprite	2005	8 and up	x	x	x	x	

Turtle Academy	jslogo	creating	moving sprite, drawing, animation	2011	8 and up	x	x	x	x	
Robo Logic	picture-blocks	game-puzzle	moving sprite	2013	8 and up				x	
StarLogo TNG/Nova	blocks	creating	moving sprite, drawing, animation, game design, simulation	2008	8 and up	x	x	x	x	x
NetsBlox	blocks	creating	moving sprite, drawing, game design, simulation	2016	8 and up	x	x	x	x	x
Logo Interpreter	UCBLogo	creating	moving sprite, drawing, game design	2013	8 and up	x	x	x	x	x
SpriteBox Coding	picture-blocks, Swift	game-puzzle	moving sprite	2018	8–13		x		x	
Code Kingdoms (Minecraft, Roblox)	blocks, Java, Lua	creating	game design	2013	8–14	x	x	x	x	x
Hopscotch	blocks	creating	animation, game design	2012	8–14	x	x	x	x	x
Scratch	blocks	creating	moving sprite, drawing, game design	2007	8–16	x	x	x	x	
Snap!	blocks	creating	moving sprite, drawing, game design	2011	8–16	x	x	x	x	x
SmalRuby	blocks/Ruby	creating	moving sprite, drawing, game design	2014	8–16	x	x	x	x	
Pyonkee	blocks	creating	moving sprite, drawing, game design	2014	8–16	x	x	x	x	
Kodu	picture-blocks	creating	game design	2009	9 and up	x	x	x		x
Lightbot	picture-blocks	game-puzzle	moving sprite	2008	9 and up	x	x		x	
Code Combat	python, JavaScript, CoffeeeScript	game-puzzle, creating	moving sprite	2013	9 and up	x	x	x	x	x
Crunchzilla/Code Monster	JavaScript	creating	animation, game design	2015	9 and up	x	x	x	x	x
NetLogo	NetLogo	creating	moving sprite, drawing, animation, game design, simulation	1999	9 and up	x	x	x	x	x
Khan Academy	JavaScript	creating	animation, game design	2014	9 and up	x	x	x	x	x
RoboZZle	picture-blocks	game-puzzle	moving sprite	2010	9 and up	x	x		x	
MIT App Inventor (Blockly)	blocks	creating	app	2010	10 and up	x	x	x	x	x
Kodular (MIT AppInventor)	blocks	creating	app	2018	10 and up	x	x	x	x	x
Thunkable	blocks	creating	app	2018	10 and up	x	x	x	x	x
Looking Glass	blocks	creating	game design	2012	10 and up	x	x	x	x	x
tickle app learn to code	blocks	creating	moving sprite	2014	10 and up	x	x	x	x	
AgentCubes	blocks	creating	game design	2006	10 and up	x	x	x	x	x
Codesters	Python	creating	moving sprite, drawing, game design	2014	11 and up	x	x	x	x	x
CodeSpells	blocks, JavaScript	creating	game design	2015	12 and up	x	x	x	x	x
Gameblox (Blocky)	blocks	creating	game design	2014	13 and up	x	x	x	x	x
App Lab (Code Studio)	blocks, JavaScript	creating	app	2016	13 and up	x	x	x	x	x
Grasshopper (Google)	JavaScript	game-puzzle, creating	animation	2018	13 and up	x	x	x	x	x
Game Lab (Code Studio)	blocks, JavaScript	creating	app	2016	13 and up	x	x	x	x	x
Karel the Dog (CodeHS)	Karel, Java	game-puzzle	moving sprite, drawing	2012	13–15	x	x	x	x	
Coding with Chrome (Blockly)	Blocks, Python, JavaScript, CoffeeScript	creating	drawing, animation, game design	2015	14 and up	x	x	x	x	x

Greenfoot	Java, Stride	creating	game design	2006	14 and up	x	x	x	x	x
Codea	Lua	creating	game design	2011	14 and up	x	x	x	x	x
CodeHS	Python, Java, JavaScript, C++, C	creating	drawing, animation	2012	16 and up	x	x	x	x	x

Appendix 2

Classification of introductory programming tools for K-12 education.

Name	**Viewing**	**Following**	**Changing the speed**	**Executing step-by-step**	**Rewinding**	**Age group**
Code Combat					x	9 and up
Karel the Dog (CodeHS)					x	13–15
RobotMagic (Blockly)				x		7–16
NetsBlox				x		8 and up
Snap!				x		8–16
Cargo-bot				x		5 and up
Rapid router (Blockly, Code for life)				x		6–13
CodeBug (electronics simulation)				x		8 and up
Penjee				x		8 and up
RoboMind				x		8 and up
RoboZZle				x		9 and up
AgentCubes				x		10 and up
App Lab (Code Studio)				x		13 and up
Game Lab (Code Studio)				x		13 and up
Greenfoot				x		14 and up
Code Studio (courses)			x	x		4 and up
Blockly games (Blockly)			x			8 and up
Lightbot Jr			x			4–8
Code Monkey			x			8 and up
Lightbot			x			9 and up
Swift Playgrounds		–	x			6 and up
Move the Turtle	x		x			5–11
StarLogo TNG/Nova	x		x			8 and up
NetLogo	x		x			9 and up
BotLogic.us		x				4–11
Kodable		x				4–11
Lego Bits and bricks		x				5–11
Junior Coder		x				5–12
ScratchJr		x				5–7
Daisy the dinosaur		x				5–7
the Foos (CodeSpark)		x				5–9
Codable Crafts		x				5–9
Run Marco (All can code, hour of code)		x				6–12
Codemancer		x				6–12
Robo Logic		x				8 and up
SpriteBox Coding		x				8–13
Tynker	x	x				5 and up
Logo Interpreter	x					8 and up
Code avengers	x					5 and up
My robot friend	x					5–12
Cato's Hike	x					5–12
Gamefroot	x					7 and up

Bee-bot app / bee-bot emulator	x					7–11
Alice	x					7–13
mBlock (Blockly)	x					8 and up
Pencil Code (Droplet editor)	x					8 and up
Microsoft MakeCode Arcade	x					8 and up
Open Roberta Lab (robot simulation)	x					8 and up
LearnToMod (Minecraft)	x					8 and up
Turtle Academy	x					8 and up
Code Kingdoms (Minecraft, Roblox)	x					8–14
Hopscotch	x					8–14
Scratch	x					8–16
SmalRuby	x					8–16
Pyonkee	x					8–16
Kodu	x					9 and up
Crunchzilla/Code Monster	x					9 and up
Khan Academy	x					9 and up
MIT App Inventor (Blockly)	x					10 and up
Kodular (MIT AppInventor)	x					10 and up
Thunkable	x					10 and up
Looking Glass	x					10 and up
tickle app learn to code	x					10 and up
Codesters	x					11 and up
CodeSpells	x					12 and up
Gameblox (Blocky)	x					13 and up
Grasshopper (Google)	x					13 and up
Coding with Chrome (Blockly)	x					14 and up
Codea	x					14 and up
CodeHS	x					16 and up