Motivational Value of Code.org's Code Studio Tutorials in an Undergraduate Programming Course

Guillaume Nel[1(✉)] and Liezel Nel[2]

[1] Department of Information Technology, Central University of Technology, Free State, Bloemfontein, South Africa
guilnel@cut.ac.za

[2] Department of Computer Science and Informatics, University of the Free State, Bloemfontein, South Africa
nell@ufs.ac.za

Abstract. As part of an instructional strategy to improve undergraduate software development students' basic understanding of programming constructs, students completed a selection of Code Studio tutorials during the first three weeks of their programming course. Block-based environments, such as the one used by the Code Studio tutorials, typically make it easier for students to learn programming as they can focus on concepts instead of syntax. Students are, however, less likely to regard an instructional strategy as meaningful if it presents no motivational value for them. In this paper, Keller's ARCS Model is used to organize the knowledge gained regarding student motivation and the motivational strategies supported by the Code Studio tutorials. Results obtained from analysis of numeric and narrative data collected through a paper-based self-completion questionnaire confirm the high motivation value of the Code Studio tutorials. The results provide insights regarding students' perceptions of Code Studio tutorials as a motivational instructional strategy in an undergraduate programming course. Since students perceive the Code Studio tutorials to have some educational value, further investigations should be conducted to consider more appropriate and effective ways to integrate Code Studio tutorials with undergraduate programming curricula.

Keywords: Block-based programming · Motivation · ARCS model
Undergraduate students

1 Introduction

The difficulties experienced by undergraduate Computer Science students in introductory programming courses have been documented extensively. Some students, especially those with little or no prior programming experience, struggle to form a sufficient understanding of the process of programming and the working of various control structures [5,25,26]. The amount of theoretical concepts

S. Kabanda et al. (Eds.): SACLA 2018, CCIS 963, pp. 173–188, 2019.
https://doi.org/10.1007/978-3-030-05813-5_12

and techniques students need to master can also lead to a loss of interest in programming [3]. Numerous studies have been conducted to identify instructional strategies that could be used to improve students' understanding of basic programming concepts [7,9,16,24]. The past decade has seen a renewed focus on student engagement as a possible strategy to enhance teaching and learning in higher education [27]. This interest in student engagement is grounded in a sound body of literature that has already established a connection between students' involvement in educationally meaningful activities and student success [2,27]. Barkley described student engagement as both *"a process and a product"* that results from *"the synergistic interaction between motivation and active learning"* [2] (p. 8). It is therefore not enough for students to just be actively involved in meaningful learning activities. They also need to be motivated to engage in such activities. In order to sustain motivation, instructional designs should incorporate strategies to address critical aspects related to attention, relevance, confidence, and satisfaction (ARCS) [11]. Before any learning can occur, however, students' attention must be grabbed. One of the attention-grabbing strategies suggested by Keller is to address students' lack of interest [13]. In order to capture software development students' interest, Kelleher and Pausch suggest a change in the programming environment used by beginners [10]. Various studies have been conducted to explore the potential of block-based programming tools such as 'Alice' [28], 'Scratch' [4,5,21], 'MIT App Inventor' for Android [20] and 'Hour of Code' [6,19] as introductory programming environments. The 'Hour of Code' challenge was launched by Code.org in December 2015 as a one-hour introduction to Computer Science.[1] It was originally designed to show that anybody can learn the basics of programming and to broaden participation in the field of Computer Science. In addition to the 'Hour of Code' event, Code.org has since developed an extended catalog of tutorials and courses that can be accessed via their Code Studio website.[2] The main aim of the Code Studio tutorials is to teach additional Computer Science subjects and programming principles to school pupils up to the age of 18 years.

As part of an instructional strategy to improve undergraduate software development students' basic understanding of programming constructs, students completed a selection of Code Studio tutorials during the first three weeks of their programming course. This paper attempts to answer the following two questions:

1. What is the motivational value of Code Studio tutorials for undergraduate programming students?
2. What are students' perceptions regarding Code Studio tutorials as a motivational instructional strategy in an undergraduate programming course?

A short review of relevant literature is in Sect. 2, followed by a discussion of the research design and method in Sect. 3. Data analysis and results are presented in Sect. 4. The paper concludes with a discussion of our findings in Sect. 5 and recommendations for future work in Sect. 6.

[1] https://hourofcode.com/za.
[2] https://code.org.

2 Related Work

2.1 Strategies to Stimulate Motivation in Educational Environments

'Motivation' is often regarded as a vague concept, especially by instructors who want to design learning environments aimed at enhancing students' motivation. In addition to a good understanding of what motivation entails, these instructors also need to consider which *motivational strategies* are best suited for their teaching and learning contexts, and how best to incorporate the chosen strategies as part of their instructional designs. Keller's ARCS model [12] captures four critical aspects (dimensions) that should be addressed in order for students to be motivated to learn: attention, relevance, confidence and satisfaction. He also provides practical strategies that can be used by instructors to achieve each of the four requirements. In using the ARCS model for instructional design [11,13], instructors should first consider strategies to capture students' interest, stimulate inquiry and maintain attention. In order to establish relevance, instructors should set clear goals that are related to the learning material, match students' interests, and provide links to students' prior and future experiences. The suggested confidence strategies should be used to create a learning environment that sets up positive attitudes and boosts students' believes that they can succeed and are in control of their own success. The satisfaction aspect relates students' continued desire to learn to their satisfaction with the process and results of the learning experience [11]. Strategies to promote feelings of satisfaction should describe ways in which intrinsic satisfaction, rewarding outcomes and fair treatment can be promoted. Intrinsic satisfaction is promoted when students feel that *"they have achieved success while studying topics that were personally meaningful to them"* [11] (p. 188). Strategies linked to extrinsic reinforcement (e.g. verbal praise, symbolic rewards and incentives such as marks) can result in rewarding outcomes for students. Instructors should also incorporate strategies to ensure that any rewards given are equitable to the amount of work done by students, and that all students are treated fairly.

2.2 Block-Based Programming Environments as Motivational Tools

In the past decade there has been a steady increase in the use of block-based programming environments to introduce students to programming [29]. These environments aim to make it easier for students to learn programming by focusing on concepts instead of syntax [23]. In this style of programming, blocks (in different shapes and colors) are used to represent the various elements of a programming language (e.g. a control structure, an operator, a variable or a function). Drag-and-drop actions are used to assemble the various blocks (like jigsaw pieces) *"according to a certain planned logic to form a computer program"* [21] (p. 1480). Given the popularity of block-based programming environments, numerous studies have been conducted to explore their use as motivational tools in programming courses of various levels.

Scratch is an open source environment developed by the Lifelong Kindergarten Group at the MIT Media Lab.[3] It facilitates the development of interactive stories, games, and animations that can be shared with others in the online Scratch community. Korkmaz conducted a comparative study to evaluate the effects of Scratch-based game activities on Computer Programming students of an Engineering faculty [14]. The results indicate high levels of motivation but also high levels of negative attitudes towards programming in general. In [5], where Scratch was used in two Computer Science courses for the first three weeks of a 15-week course, the majority of students found the environment to be motivating, funny or easy. A small minority, however, described it as 'difficult' or 'normal'. In [20], where pupils used Scratch for seven weeks, measurements of motivation transition over the 7-week period revealed increased *"intrinsic goal orientation, task value, control of learning beliefs and self-efficacy"* (p. 1042) however no change in extrinsic motivation.

Studies comparing the motivation levels of students using Scratch versus students using traditional text-based programming environments indicated that

- Scratch students were more motivated [5,21];
- Scratch students found the programming environment less boring [21];
- Scratch students were more creative as well as more inclined to create games in their own time [21];
- Scratch students were more motivated to continue with Computer Science studies [4,21];
- The Scratch environment makes it easier for instructors to identify struggling students [5];
- There are no noteworthy differences in retention rates between Scratch and text-programming students [5].

De Kereki urges instructors to consider the impact that an additional learning tool such as Scratch (that is only used for a few weeks at the beginning of a course) will have on students [5]. Students invest time in familiarizing themselves with Scratch, and after a few weeks they have to start anew (without Scratch) in a completely different environment to develop 'real' programs. Such a strategy might have a negative impact on students' motivation towards Scratch and programming in general.

Hour of Code started as a one-hour introduction to Computer Science with the aims to 'demystify code', to show that anybody can learn the basics, and to broaden participation in the field of Computer Science. Although the official 'Hour of Code' event takes place annually in December during 'Computer Science Education Week', all tutorial materials can be accessed throughout the year. The biggest difference between the 'Hour of Code' environment and Scratch is that students cannot create their own programs. In the 'Hour of Code' tutorials, students are confronted with a visualized puzzle-based problem (e.g. a character that needs to move through a maze). Video instructions are used to explain the

[3] https://scratch.mit.edu.

aim of the lesson and to introduce new command blocks (representing programming constructs). Students must then move the relevant blocks from a toolbox to a work space and assemble them in the correct order to create an 'algorithm' that will solve the given puzzle. Students can test their 'algorithm at any stage. During testing, students can observe the character's movements as each of the command blocks in their work space are executed. The Code Studio tutorials (an extension of 'Hour of Code') use the same block-based environment as the 'Hour of Code' tutorials. Instructors can use the Code Studio dashboard to build custom courses for their students by selecting tutorials for different subjects and age groups. The tutorials feature popular themes such as 'Star Wars', 'Minecraft', or Disney's 'Frozen' world. An extensive internet search revealed only a small number of studies that investigated motivational aspects of 'Hour of Code' activities. The following two of those studies are of particular relevance:

The study of [19] investigated the learning motivation of high school pupils and first-year university students (non-Computing majors) during 'Hour of Code' activities. The self-reported Situational Motivation Scale (SIMS) [8] was used to measure four motivational components: Intrinsic motivation (doing an activity because it is interesting or enjoyable); Identified regulation (doing an activity because of its perceived importance and value); External regulation (complying with external demands); and Amotivation (the lack of motivation). Both groups showed high levels of intrinsic and identified regulation. The high school pupils found the activities more intrinsically motivated as these activities are probably *"better suited to their lower age level"* [19] (p. 745). Although both groups showed low levels of external regulation and amotivation, the university group showed significantly higher levels of amotivation. Their high levels of amotivation could be attributed to the fact that they were non-computing students who are *"more oriented in their chosen field of studies and may not be interested much in programming"* [19] (p. 745). Overall, Nikou and Economides regard the 'Hour of Code' tutorial as a valuable example of a well-designed educational activity because of its high level of intrinsic motivation [19], but suggest that it should rather be used for students of lower ages.

In [6], 116 undergraduate students studying different majors (including business, accounting, criminal justice, allied health sciences, geography, hospitality tourism management, and psychology) at two universities completed one 'Hour of Code' tutorial. Pre- and post-surveys were used to determine if the tutorial had any effect on the students' attitudes towards programming and if it improved their basic programming skills. The results showed a significant positive impact on the students' attitude but no significant changes in their programming skills. Based on these experiences, Du, Wimmer and Rada recommend that instructors who want to use the 'Hour of Code' tutorials for programming skill development should *"appreciate the learning objectives of the 'Hour of Code' and integrate the tutorials appropriately into their teaching"* [6] (p. 65).

3 Research Design and Method

We followed a mixed methods approach based on the Framework of Integrated Methodologies (FraIM) as suggested by Plowright [22]. The context of this paper was an introductory (first-year) programming course (OPG1) in the Department of Information Technology at a selected South African University of Technology. This is a foundation course aimed at introducing students to basic computer programming principles and constructs through the use of the C# programming language. The main source of data in the study was the population of the 221 students registered for this course. Students were divided into two class groups with two 85-min theoretical sessions (in a traditional lecture hall) and two 85-min practical sessions per week allocated to each group. During the first three weeks of the semester, four practical sessions (two in week 1 and one each in weeks 2 and 3) were set aside for students to work on Code Studio tutorials. The instructor set up a class group for the 'Accelerated Intro to CS Course' on the Code Studio dashboard. This is a 20-h course originally designed for use with pupils between the ages of 10 and 18. It covers core Computer Science and programming concepts and incorporates selected exercises from the Code.org 'CS Fundamentals' syllabus. Each student had to personally register on the Code Studio website and enroll for the custom course set up by the instructor. Although students were encouraged to complete as many of the tutorials as possible during the four dedicated practical sessions, they did not receive any 'rewards' in the form of marks for the completion of these tutorials. Students were also not required to continue working on the tutorials outside of their practical sessions. It should be noted that the majority of the selected Code Studio tutorials were related to programming concepts that were much more advanced than those covered in the first three weeks of the introductory programming course. However, for the remainder of the semester, the instructor intentionally referred back to specific Code Studio examples whenever she discussed new C# programming concepts related to selection, iteration and OO-methods.

In order to collect data on the students' Code Studio experiences, a survey strategy was deemed most appropriate to manage this relatively large data source [22]. As part of this strategy, data was collected by means of 'asking questions' in a paper-based self-completion questionnaire containing both closed and open-ended questions. The questionnaire was distributed at the end of the semester (week 13) for completion during one of the normal lecture sessions. 148 students (the sample) voluntarily completed the questionnaire.

The questionnaire consisted of three sections: Sect. 1 was based on the Reduced Instructional Materials Motivation Survey (RIMMS) [17]. The original Instructional Materials Motivation Survey (IMMS) [11] is a 36-item situational measure of participants' reactions to self-directed instructional materials they have used. After conducting an extensive validation study of the IMMS, Loorbach, Peters, Karreman and Steehouder devised the 12-item RIMMS [17] which they regarded as a more appropriate post-test tool in their instructional setting than the IMMS. The 12 RIMMS items consist of 3 items for each of the four sub-scales of the ARCS model—attention, relevance, confidence, and satisfac-

tion. For the RIMMS, responses are recorded on a 5-point Likert scale with the response scales ranging from 1 (not true) to 5 (very true). Where necessary, the wording of the RIMMS items we adapted to make it more relevant to the context of our study. Care was, however, taken not to change the substance of the items as these relate to specific attributes of the ARCS model [11]. Section 2 of the survey consisted of open-ended questions aimed at soliciting students' views regarding (1) what motivated them to continue with the Code Studio tutorials outside of class; (2) their reasons for abandoning the Code Studio tutorials; and (3) what they regarded as the main educational value of the Code Studio tutorials. The final section of the questionnaire, Sect. 3, was used to collect basic demographic data from the participants. Numerical data collected through the questionnaire was analyzed in 'SPSS 24' while narrative data was analyzed in 'NVivo 11'.

4 Data Analysis and Results

A total of 148 participants completed the survey. There were 105 (70.9%) male participants and 43 (29.1%) female participants. Although students were not required to work on the Code Studio tutorials outside of their scheduled practical sessions, 33 students (22.3%) indicated that they continued to work on the tutorials in their own time.

4.1 Numeric Data: RIMMS

This sub-section describes the analysis and results of the data collected in Sect. 1 of the questionnaire—the RIMMS questions. Reliability estimates were calculated to show the internal reliability of the scales and a correlation analysis was conducted to determine the relationship between the four ARCS categories. The internal consistency estimate for the overall set of 12 items, based on Cronbach's alpha, shows high internal consistency with a satisfactory value of 0.899 for the total scale: see Table 1. The reliability estimates for the attention, relevance and satisfaction scales were satisfactory (>0.6). The low value for the confidence scale (0.591) can be regarded as questionable and might serve as an indication that the three confidence items did not necessarily measure the same underlying concept in the context of this study. It was, however, decided to retain the confidence data for further analysis since the recorded value is very close to 0.6.

The calculated inter-factor Pearson's correlation coefficients indicate a significant positive relationship between all the ARCS dimensions: see Table 2. The highest correlation was between the confidence and satisfaction dimensions (r = .720) and the lowest between attention and relevance (r = .595). The students' motivation levels were analyzed for each of the four ARCS dimensions as well as for each of the individual items in these dimensions: see Table 3.

In the attention dimension, the total mean score was 3.737, indicating positive motivation levels. Students were positive about how information arrangement (Q1.5; M = 3.811) and quality of the tutorial graphics and sounds (Q1.2;

Table 1. RIMMS reliability estimates and descriptive statistics (N = 148)

Scale	Cronbach's alpha	Cronbach's alpha (standard. items)	Mean	SD	# items
Attention	.732	.739	3.7365	0.89785	3
Relevance	.686	.687	3.7703	0.89582	3
Confidence	.591	.592	3.7095	0.82596	3
Satisfaction	.808	.808	3.6757	1.03085	3
Overall	.899	.900	3.6700	0.78927	12

Table 2. RIMMS correlations between ARCS dimensions

Scale	Attention	Relevance	Confidence	Satisfaction
Attention	1	.595**	.621**	.663**
Relevance	.595**	1	.696**	.678**
Confidence	.621**	.696**	1	.720**
Satisfaction	.663**	.678**	.720**	1

**Correlation is significant at the 0.01 level (2-tailed)

M = 3.743) helped to keep their attention. They were, however, less positive about the role that the variety of the tutorials and characters (Q1.9; M = 3.655) played in this regard.

In the relevance dimension, the total mean score was 3.770. Students were most positive about the relation between the tutorials and the concepts they already knew (Q1.1; M = 3.966). The students were, however, less positive about the worthiness (Q1.7; M = 3.676) and usefulness of these tutorials in their programming course (Q1.10; M = 3669).

The total mean score of the confidence dimension was 3.710. By doing the tutorials, students were positive that they could learn the related programming concepts (Q1.3; M = 3.764) and were mostly confident that they would be able to pass a test on it (Q1.8; M = 3.709). They were slightly less positive about how the organization of the tutorials helped them to learn the related programming concepts (Q1.11; M = 3.655).

For the satisfaction dimension, the total mean score was 3.676—the lowest of the four dimensions. Students were most positive about how their enjoyment of the tutorials fueled their interest in programming (Q1.4; M = 3.750). Overall, they enjoyed doing the well-designed tutorials (Q1.6; M = 5.61 and Q1.12; M = 3.716).

4.2 Narrative Data

This sub-section describes the analysis and results of the narrative data collected in Sect. 2 of the questionnaire through mostly open-ended questions. Inductive analysis [18] was used to make sense of the students' responses. This analysis strategy was chosen as it allowed for the emergence of categories from the data itself. For each question the individual responses (or response segments) were

Table 3. RIMMS motivation level per ARCS dimension (N = 148)

Dimension item	Mean	SD
Attention	**3.737**	**.8979**
Q1.2: The quality of the tutorial graphics and sounds helped to hold my attention	3.743	1.2185
Q1.5: The way the information is arranged for each tutorial helped keep my attention	3.811	1.0456
Q1.9: The variety of the tutorials and characters helped keep my attention	3.655	1.0672
Relevance	**3.770**	**.8958**
Q1.1: It is clear to me how the tutorials are related to the programming concepts I already know	3.966	1.1394
Q1.7: The way in which the tutorials were presented convey the impression that the related programming concepts is worth knowing	3.676	1.1618
Q1.10: The programming concepts covered by the tutorials will be useful to me in OPG1	3.669	1.1272
Confidence	**3.710**	**.8260**
Q1.3: As I worked on the tutorials, I was confident that I could learn the programming concepts	3.764	1.0777
Q1.8: After working on the tutorials for a while, I was confident that I would be able to pass a test on it	3.709	1.1737
Q1.11: The good organization of the tutorials helped me be confident that I would learn the related programming concepts	3.655	1.0862
Satisfaction	**3.676**	**1.0309**
Q1.4: I enjoyed the tutorials so much that I would like to know more about programming	3.750	1.2392
Q1.6: I really enjoyed doing the tutorials	3.561	1.2078
Q1.12: It was a pleasure to do such well-designed tutorials	3.716	1.1898

coded based on the main aspect it related to. The resulting initial codes were then compared for duplication and overlapping. Similar codes were grouped together and, where necessary, unrelated codes were re-coded. Refining of the coding system continued until the remaining codes could be grouped into a small set of categories. Responses that were deemed irrelevant to the asked question were omitted from the analysis.

Motivation to Continue. Since participation in and completion of the Code Studio tutorials were not compulsory, students were asked to indicate what motivated them to continue with these tutorials. 83 students (56.1%) responded to this question. Inductive analysis led to the identification of 96 motivational rea-

sons that were grouped into nine categories. Table 4 provides a summary of these motivational categories together with the number of response segments that relates to each category.

Table 4. RIMMS motivational categories for continuation

Category	Count
Learn/improve skills	20
Relate to subject content	19
Fun	16
Engaging	9
Situational interest	9
Challenge	8
Achievement	7
Subjective norm	6
Ease of use	2

The main motivational aspect identified relates to the way in which the Code Studio tutorials helped students to learn or improve various skills such as programming skills and logical thinking skills. Students were also motivated as they could see the relation between the tutorials and the OPG1 subject content: *"This can be useful to me in OPG1"* and *"I learn more skills that I can apply on OPG1"*. The 'fun' aspect also motivated students as they regarded the tutorials as entertaining, enjoyable and *"a game"*. Three students in particular commented on how the tutorials helped *"to make programming fun"*. While some students were motivated by the engagement provided by the tutorials (*"interesting"*, *"keeps me focused"*, *"keeps me motivated"*, *"is addictive"*), others were motivated by the challenge presented by the tutorials (*"increasing level of difficulty"*, *"challenging stages"*, *"I had to think outside the box"*). As students became *"curious"* and *"intrigued"*, the tutorials sparked situational interest as in [15], since they wanted to *"lean more about programming"*. One student, who had no prior programming knowledge, confirmed that the tutorials *"helped to get started with programming"* while another student explained how the tutorials helped to give him a better idea of the skills required to be a successful programmer. Some students were only motivated by achievement. These students said that *"getting a trophy was the best feeling"*. They were driven by the prospect of *"progress to higher levels"* and *"wanted to finish all the tutorials"*. On the other hand there were students who noted that they were driven by subjective norm (i.e. a perceived social pressure [1]) to complete the tutorials. These students regarded their influencers as *"the OPG1 lecturer"*, *"all the famous people who kept on talking about how exiting programming is"* (in the Code Studio introductory video) or even their *"friends"* and *"fellow classmates"*.

Two students remarked that they were motivated by the *"ease of use"* of the Code Studio interface as it was *"very intuitive"*.

Reasons for Stopping. Although the students were scheduled to continue with the Code Studio tutorials until the end of the third week of the semester, 83 students (56.1%) indicated that they stopped doing the tutorials before the time. Inductive analysis of the response segments revealed 83 reasons for stopping. These reasons were grouped into seven categories: see Table 5.

Table 5. RIMMS motivational categories for stopping

Category	Count
Needed time for other subjects	25
Needed time for OPG1	21
Lost interest	12
No challenge	8
Unrelated to subject	7
Too difficult	6
Learned enough	4

As the semester progressed and *"workloads increased"*, most of the students who responded to this question (54.8%) mentioned that the time they were spending on the non-compulsory Code Studio tutorials could be put to better use. They either needed more time to work on their other subjects or they used the scheduled Code Studio sessions to *"catch up on OPG1 practical assignments"*. Students explained that they were *"struggling with the OPG1 work"* or *"falling behind"* and needed to *"do additional C# exercises"* or *"attend extra classes"* in order to improve their OPG1 marks. Some students lost interest in the tutorials since they were either *"no longer motivated to continue"*, or felt that the exercises were becoming *"boring"* as *"some of the things keep repeating"*. While some students stopped doing the tutorials because they were *"becoming too easy"* or *"no longer presented a challenge"*, others stopped because they became *"too difficult"* or *"complicated"* for them. The relations between OPG1 and the Code Studio tutorials were also no longer as obvious for some students. They could either not see how the tutorials helped with programming or saw no improvement in their programming skills and OPG1 performance. A small number of students indicated that they stopped doing the tutorials because they had *"learned enough"*.

Educational Value. In the third open-ended question of Sect. 2, students were asked to give their views on the educational value of the Code Studio tutorials.

Table 6. RIMMS motivational categories for educational value

Category	Count
Related to subject content	65
Improve logical thinking/problem solving skills	39
Easy/fun way to learn	17
Unsure	3

Analysis of the 115 responses (from 77.7% of students) revealed 124 response segments that were ultimately grouped into four categories: see Table 6.

Although some of the students were unsure (3; 2.6%), the majority of the responding students (65; 56.5%) attributed the educational value of the Code Studio tutorials to the relation it had with the OPG1 subject content. Some of these students made specific mention of the role the tutorials played in improving their *"overall understanding of programming"* as well as their understanding of *"selection structures"*, *"repetition structures"* and *"methods"*. 39 of the responding students (33.9%) linked the educational value of these tutorials to their ability to improve *"logical thinking"* and/or *"problem solving skills"*. For others it was just a *"fun"* and *"easy way to learning programming"*.

5 Discussion

The numeric data indicate that students' overall motivation level when doing the Code Studio tutorials was high. There were also strong positive correlations between all four of the ARCS dimensions showing that the motivational elements of each dimensions played an important role in influencing students' overall motivation. High motivation levels were reported for the attention, relevance and confidence dimensions with a slightly lower motivation level for the satisfaction dimension.

From the narrative data it became apparent that the Code Studio tutorials managed to grab the students' *attention* as the entertaining environment captured their interest and added a fun element to programming. It also managed to stimulate inquiry as they were curious and wanted to learn more about programming. The 'addictive' tutorials helped to keep them focused and maintained their attention. The tutorials also produced *relevance* as students could see the relation between the Code Studio tutorials and the programming concepts they were studying in OPG1. The tutorials also matched their personal goals to learn more about programming. In addition, the tutorials also helped students to improve their programming and logical thinking skills. Motive stimulation was provided through the 'social pressure' students experienced from various individuals (either role models or fellow students). Through completion of the Code Studio tutorials, students were able to build *confidence* as the intuitive and easy to use interface increased their believe that they could successfully complete the given tasks. The tutorials also helped students to develop a sense of

personal responsibility as they were required to work at their own pace and could measure their own progress. The challenging tasks that gradually increased in level of difficulty also provided students with numerous opportunities to succeed. Finally, the Code Studio tutorials also managed to generate *satisfaction* through the provision of intrinsic satisfaction and extrinsic rewards. Students were able to achieve a desirable level of success while learning more about programming concepts—a topic that was personally meaningful to them. The students wanted to progress to higher levels and finish all the exercises. They were also motivated by the 'trophies' they could earn for correctly completing the tutorials. Since students' solution attempts to each of the Code Studio tutorial exercises were evaluated by the system (and not by a human assessor), the assumption can be made that all attempts were evaluated according to the same standards—i.e.: fair treatment of all students.

Despite the initially high motivation levels reported for the Code Studio tutorials, students' satisfaction levels did not remain high throughout. Due to increasing workloads in OPG1 and their other subjects, students no longer regarded the intrinsic and extrinsic rewards provided by the Code Studio tutorials as equitable to the amount of work they had to put in to complete the tutorials—see [11] for comparison. Some of the students could no longer see the relevance of the tutorials to the OPG1 content. The decline in relevance is not surprising as the more advanced Code Studio tutorials focused on concepts such as repetition structures and OO-methods that would only be covered later in the OPG1 syllabus. Levels of confidence also declined as the tutorial tasks became either too difficult or too easy. Attention levels declined as students became bored with the elements that kept repeating in the tutorials. The drops in confidence and attention could be related to Nikou and Economides' observation that these 'Hour of Code' type tutorials are better suited for students of lower age levels [19]. Given Korkmaz's warning about the possible negative impact that a move between programming environments (e.g. from Scratch to text-based) could have on students' motivation [14], it should be noted, however, that none of the students in this study reported any negative attitudes towards programming in general. One significant difference between the 'Hour of Code'/Code Studio and the Scratch environments is that students do not have to invest a significant amount of time to familiarize themselves with the 'Hour of Code' environment. Students are, therefore, much less likely to experience problems in moving over to a completely different environment (like Microsoft Visual Studio) after first spending a few weeks in the 'Hour of Code' environment. It should also be noted that the students in this study already started working in Microsoft Visual Studio environment in the second week of the semester (after only one week of Code Studio tutorials).

Another aspect of this study, which should not be overlooked, is the fact that students only completed the questionnaire at the end of the semester, 10 weeks after conclusion of the Code Studio activities. At that stage their normal C# lectures and practicals already covered most of the programming constructs that were included in the selected Code Studio tutorials. Students were therefore

in a much better position to evaluate the educational value of the Code Studio tutorials with regard to the overall OPG1 course syllabus. They could see the relation of the Code Studio tutorials to the OPG1 course content, the opportunities it provided to students to improve their programming and logical thinking skills, and the way in which the entertaining environment managed to capture their interest.

While the students worked on the Code Studio tutorials, we observed that students tended to follow one of three approaches to solve the puzzle problems. In the first approach, students used a process that included detailed planning and analysis of the whole problem before they started to assemble their command blocks in the work space. The second approach can be linked to 'chunking' where students solved the problem in parts with testing conducted after each iteration. The third approach can be described as trial-and-error. The students who followed this approach would randomly place command blocks in the work space and then make changes based on the test execution results.

6 Conclusions and Future Work

The main aims of this paper were (1) to determine the motivational value of Code.org's Code Studio tutorials for undergraduate programming students, and (2) to gain insights into these students' perceptions of the Code Studio tutorials as a motivational instructional strategy. In this regard, Keller's ARCS Model [11–13] provided a typology to organize the knowledge gained regarding student motivation and the motivational strategies supported by the Code Studio tutorials. Initially, a change in programming environment was identified as a possible attention-grabbing strategy to capture students' interest in programming. Analysis of the numeric data confirmed the high motivational value of the Code Studio tutorials for the targeted group of students. Evidence gathered from the richer narrative data was used to illustrate how integration of the selected Code Studio tutorials served as a motivational instructional strategy. Analysis of the gathered student perceptions also revealed that this particular Code Studio integration attempt was successful in incorporating the following of Keller's suggested motivational strategies [11,13]:

- Strategies to generate and sustain attention: Capture interest, stimulate inquiry and maintain attention.
- Strategies to establish and support relevance: Relate to goals, match interests and tie to experiences.
- Strategies to build confidence: Explain success expectations, provide success opportunities and develop personal responsibility.
- Strategies to promote feelings of satisfaction: Provide intrinsic reinforcement, provide rewarding outcomes and fair treatment.

It can therefore be concluded that this Code Studio integration attempt was successful in achieving each of Keller's four main requirements for motivation: attention, relevance, confidence, and satisfaction [13]. Given the variety of strategies

followed by students in solving the Code Studio exercises, future research could investigate the influence that this type of block-based programming environment could have on students' development processes in a conventional development environment (like MS Visual Studio). The real learning value of the Code Studio tutorials on students' understanding of basic programming constructs such as selection, iteration and methods should also be investigated. Since numerous students pointed out the game-like feel created by the Code Studio environment, the influence of gamification on long-time motivation in Code Studio tutorials could also be investigated.

As noted by Du (et al.), instructors who want to use 'Hour of Code' type tutorials for programming skill development, should *"appreciate the learning objectives of the 'Hour of Code' and integrate the tutorials appropriately into their teaching"* [6] (p. 65). Based on the insights gained from this study, a more detailed investigation could be conducted to consider more appropriate and effective ways to integrate Code Studio tutorials with undergraduate programming curricula.

References

1. Ajzen, I.: The theory of planned behavior. Organ. Behav. Hum. Decis. Process. **50**(2), 17–211 (1991)
2. Barkley, E.F.: Student Engagement Techniques: A Handbook for College Faculty. Jossey-Bass, San Francisco (2010)
3. Bennedsen, J., Caspersen, M.E.: Exposing the programming process. In: Bennedsen, J., Caspersen, M.E., Kölling, M. (eds.) Reflections on the Teaching of Programming: Methods and Implementations. LNCS, vol. 4821, pp. 6–16. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77934-6_2
4. Coravu, L., Marian, M., Ganea, E.: Scratch and recreational coding for kids. In: 14th RoEduNet International Conference – Networking in Education and Research (RoEduNet NER), pp. 85–89. IEEE (2015)
5. De Kereki, I.F.: Scratch: applications in computer science 1. In: 38th Annual Frontiers in Education Conference Proceedings, pp. 7–11. IEEE (2008)
6. Du, J., Wimmer, H., Rada, R.: 'Hour of Code': can it change students' attitudes toward programming? J. Inf. Technol. Educ. Innov. Pract. **15**, 52–73 (2016)
7. Eranki, K.L.N., Moudgalya, K.M.: Program slicing technique: a novel approach to improve programming skills in novice learners. In: Proceedings of the 17th Annual Conference on Information Technology Education (SIGITE 2016), pp. 160–165. ACM (2016)
8. Guay, F., Vallerand, R.J., Blanchard, C.: On the assessment of situational intrinsic and extrinsic motivation: the situational motivation scale (SIMS). Motiv. Emot. **24**(3), 175–213 (2000)
9. Guzdial, M.: Programming environments for novices. In: Fincher, S., Petre, M. (eds.) Computer Science Education Research, pp. 127–154. Taylor & Francis (2004)
10. Kelleher, C., Pausch, R.: Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. **37**(2), 83–137 (2005)
11. Keller, J.M.: Motivational Design for Learning and Performance: The ARCS Model Approach. Springer, Heidelberg (2010). https://doi.org/10.1007/978-1-4419-1250-3

12. Keller, J.M.: Motivational design of instruction. In: Reigeluth, C.M. (ed.) Instructional-Design Theories and Models: An Overview of their Current Status, pp. 383–433. Lawrence Earlbaum Associates (1983)
13. Keller, J.M.: Strategies for stimulating the motivation to learn. Perform. Instr. **26**(8), 1–7 (1987)
14. Korkmaz, O.: The effect of scratch- and lego mindstorms Ev3-based programming activities on academic achievement, problem-solving skills and logical-mathematical thinking skills of students. Malays. Online J. Educ. Sci. **4**(3), 73–88 (2016)
15. Krapp, A., Hidi, S., Renninger, K.A.: Interest, Learning and Development. In: Renninger, A., Hidi, S., Krapp, A. (eds.) The Role of Interest in Learning and Development, pp. 3–25. Lawrence Erlbaum Associates (1992)
16. Lahtinen, E., Ala-Mutka, K., Järvinen, H.: A study of the difficulties of novice programmers. In: Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2005) (2005). ACM SIGCSE Bull. **37**(3), 14–18 (2005)
17. Loorbach, N., Peters, O., Karreman, J., Steehouder, M.: Validation of the instructional materials motivation survey (IMMS) in a self-directed instructional setting aimed at working with technology. Br. J. Educ. Technol. **46**(1), 204–218 (2015)
18. McMillan, J.H., Schumacher, S.: Research in Education: Evidence-Based Inquiry, 6th edn. Pearson Education, London (2006)
19. Nikou, S.A., Economides, A.A.: Measuring student motivation during 'The Hour of Code' activities. In: Proceedings of the 14th International Conference on Advanced Learning Technologies (ICALT), pp. 744–745. IEEE (2014)
20. Nikou, S.A., Economides, A.A.: Transition in student motivation during a Scratch and an App Inventor course. In: Proceedings of the Global Engineering Education Conference (EDUCON), pp. 1042–1045. IEEE (2014)
21. Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., Lahmine, S.: Learning basic programming concepts by creating games with scratch programming environment. Procedia Soc. Behav. Sci. **191**, 1479–1482 (2015)
22. Plowright, D.: Using Mixed Methods: Frameworks for an Integrated Methodology. SAGE, Thousand Oaks (2011)
23. Price, T.W., Barnes, T.: Position paper: block-based programming should offer intelligent support for learners. In: Proceedings of the Blocks and Beyond Workshop (B&B), pp. 65–68. IEEE (2017)
24. Sentance, S., Csizmadia, A.: Computing in the curriculum: challenges and strategies from a teacher's perspective. Educ. Inf. Technol. **22**(2), 469–495 (2017)
25. Soloway, E., Bonar, J., Ehrlich, K.: Cognitive strategies and looping constructs: an empirical study. Commun. ACM **26**(11), 853–860 (1983)
26. Spohrer, J.C., Soloway, E.: Putting it all together is hard for novice programmers. In: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pp. 728–735. IEEE (1985)
27. Trowler, V.: Student Engagement Literature Review. Higher Education Academy, York (2010)
28. Wang, T.C., Mei, W.H., Lin, S.L., Chiu, S.K., Lin, J.M.C.: Teaching programming concepts to high school students with Alice. In: Proceedings of the 39th Frontiers in Education Conference (FIE 2009), pp. 955–960. IEEE (2009)
29. Weintrop, D., Wilensky, U.: 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, pp. 199–208. ACM (2015)