RESEARCH ARTICLE

Classification of the programming styles in scratch using the SOLO taxonomy

Anastasios Ladias[1*] **Theodoros Karvounidis**[2] **Dimitrios Ladias**[3]

[1] Ministry of Education, Attica 19005, Greece
[2] Department of Informatics, University of Piraeus, Piraeus 18534, Greece
[3] Department of Informatics, National and Kapodistrian University of Athens, Athens 10679, Greece

Correspondence to: Anastasios Ladias, Ministry of Education, Attica 19005, Greece;
Email: ladiastas@gmail.com

Received: July 29, 2021;
Accepted: September 20, 2021;
Published: September 28, 2021.

Citation: Ladias, A., Karvounidis, T., & Ladias, D. (2021). Classification o f t he P rogramming Styles in Scratch using the SOLO taxonomy. *Adv Mobile Learn Educ Res*, 2021, **1**(2), 114-123. https://doi.org/10.25082/AMLER.2021.02.006

Abstract: The present study attempts to categorize the programming styles of sequential, parallel, and event-driven programming using as criterion, the level of adoption of the structured programming design techniques. These techniques are modularity, hierarchical design, shared code, and parametrization. Applying these techniques to the Scratch programming environment results in a two-dimensional table of representative code. In this table, one dimension is the types of the aforementioned programming styles and the other is the characteristics of structured programming. The calibration of each of the dimensions has been held with the help of the levels of the SOLO taxonomy. This table can develop criteria for evaluating the quality characteristics of codes produced by students in a broader grading system.

Keywords: programming styles, programming genres, Scratch, SOLO taxonomy

1 Introduction

The SOLO taxonomy (Structure of Observed Learning Outcomes) is a means of classifying learning outcomes in terms of their complexity (Biggs & Collis, 1982). SOLO taxonomy describes a hierarchy where each level becomes a foundation on which further learning is built (Biggs & Collis, 1982). It enables the instructors to assess students' work on the performance levels from the lower end (Pre-structural) to the higher end (Extended Abstract):

(1) Pre-Structural Level. The student does not understand, uses irrelevant information, and/or misses the point altogether.

(2) Uni-Structural Level. The student can deal with one single aspect and make apparent connections. The student can use terminology, recite (remember things), perform simple instructions/algorithms, paraphrase, identify, name or count.

(3) Multi-Structural Level. The student can deal with several aspects, but these are disconnected. He/she can enumerate, describe, classify, combine, apply methods, structure, execute procedures, *etc*.

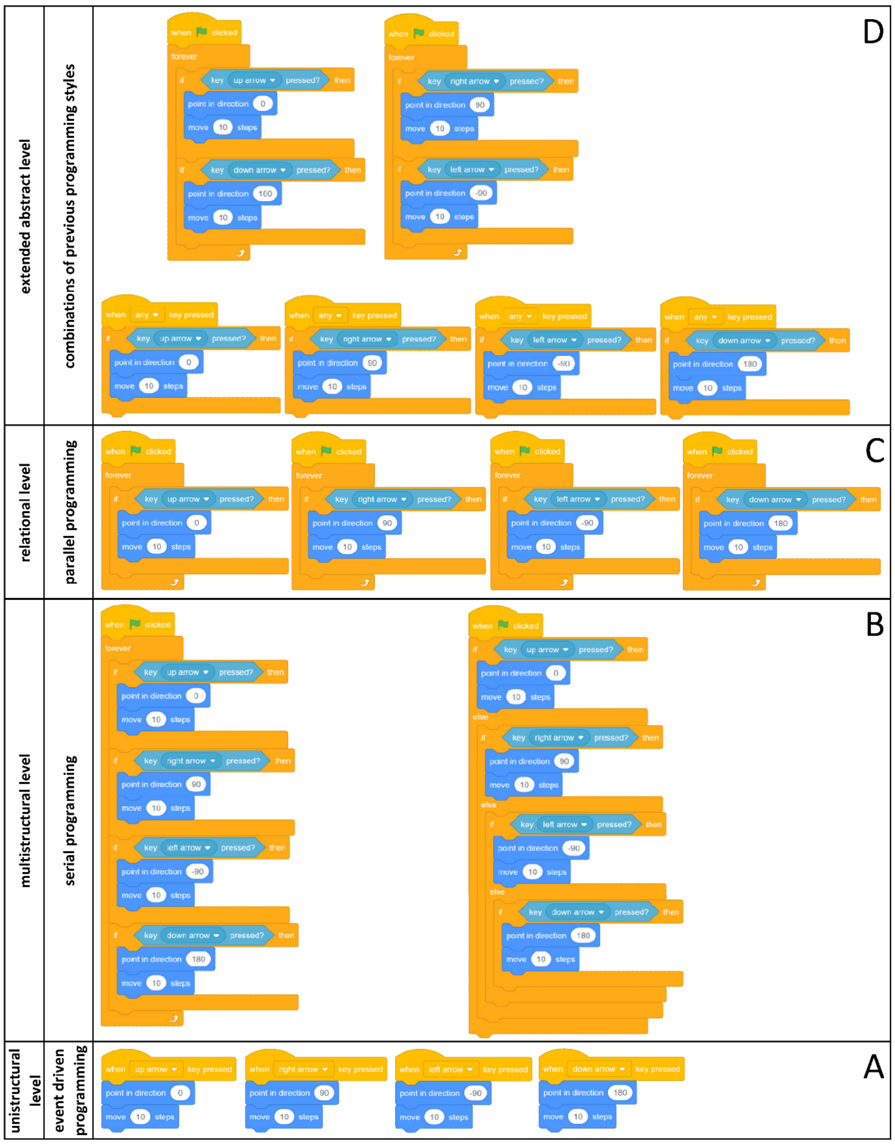
(4) Relational Level. The student may understand relations between several aspects and how they might fit together to form a whole. The understanding forms a structure and may thus have the competence to compare, relate, analyze, apply theory, and explain cause and effect.

(5) Extended Abstract Level. The student may generalize structure beyond what was given, perceive structure from many different perspectives, and transfer ideas to new areas. He/she may have the competence to generalize, hypothesize, criticize or theorize.

Scratch is an educational block-based visual programming environment in which the novice programmer can manage - with relative ease - multimedia elements and engage in authentic digital projects by creating animation, simulations, and games (Resnick et al., 2009). Scratch is designed to support children and novices' learning through experimenting and tinkering, as it encourages learners to engage in creative learning experiences and express their ideas using code (Papadakis, 2020). In the Scratch programming environment, the code consists of modules called scenarios, which are distributed across the objects. Using the metaphor that parallels a theatrical production with the development of a project in Scratch, the scenarios are considered to identify events and control the behavior of the actors-objects, which appear as sprites in a scene (Ladias et al., 2020).

The Scratch programming environment can support various programming styles such as structured programming, parallel programming, event-driven programming, object-based programming, and a form of object-oriented programming (using clones). Therefore Scratch, combined with its ease of learning and handling, is suitable for novice programmers for comparing different programming styles (Vidakis et al., 2014; Papadakis & Kalogiannakis, 2019).

It aims to categorize - using the SOLO taxonomy - each of the three aforementioned programming styles into a dimension corresponding to structured programming. For a novice programmer to be able - for educational purposes - to compare each programming style, the scenario used in this study was used to solve the same simple problem, namely the user remote control of an object in the Scratch scene, using the keyboard arrows.



```
D
when [flag] clicked
forever
  if <key (up arrow) pressed?> then
    point in direction (0)
    move (10) steps
  if <key (down arrow) pressed?> then
    point in direction (180)
    move (10) steps

when [flag] clicked
forever
  if <key (right arrow) pressed?> then
    point in direction (90)
    move (10) steps
  if <key (left arrow) pressed?> then
    point in direction (-90)
    move (10) steps

when (any) key pressed
if <key (up arrow) pressed?> then
  point in direction (0)
  move (10) steps

when (any) key pressed
if <key (right arrow) pressed?> then
  point in direction (90)
  move (10) steps

when (any) key pressed
if <key (left arrow) pressed?> then
  point in direction (-90)
  move (10) steps

when (any) key pressed
if <key (down arrow) pressed?> then
  point in direction (180)
  move (10) steps

C
when [flag] clicked
forever
  if <key (up arrow) pressed?> then
    point in direction (0)
    move (10) steps

when [flag] clicked
forever
  if <key (right arrow) pressed?> then
    point in direction (90)
    move (10) steps

when [flag] clicked
forever
  if <key (left arrow) pressed?> then
    point in direction (-90)
    move (10) steps

when [flag] clicked
forever
  if <key (down arrow) pressed?> then
    point in direction (180)
    move (10) steps

B
when [flag] clicked
forever
  if <key (up arrow) pressed?> then
    point in direction (0)
    move (10) steps
  if <key (right arrow) pressed?> then
    point in direction (90)
    move (10) steps
  if <key (left arrow) pressed?> then
    point in direction (-90)
    move (10) steps
  if <key (down arrow) pressed?> then
    point in direction (180)
    move (10) steps

when [flag] clicked
if <key (up arrow) pressed?> then
  point in direction (0)
  move (10) steps
else
  if <key (right arrow) pressed?> then
    point in direction (90)
    move (10) steps
  else
    if <key (left arrow) pressed?> then
      point in direction (-90)
      move (10) steps
    else
      if <key (down arrow) pressed?> then
        point in direction (180)
        move (10) steps

A
when (up arrow) key pressed
point in direction (0)
move (10) steps

when (right arrow) key pressed
point in direction (90)
move (10) steps

when (left arrow) key pressed
point in direction (-90)
move (10) steps

when (down arrow) key pressed
point in direction (180)
move (10) steps
```

Figure 1 SOLO taxonomy classification levels in parallel, sequential, and event-driven programming styles

2 Structured programming

According to Papert (1991), "the mathematician, George Polya argued that general methods for solving problems should be taught and specifically recommends that whenever we approach a problem, we turn to the question: Can the problem be divided into simpler problems?". An example of Scratch program analysis using (a) processes and (b) messages is shown in Figure 2. The similar topology of the code in both categorization modes is noted.

The problem-solving technique, which analyzes and synthesizes a problem into individual easier-to-manage subproblems, has been adopted in structured programming. Structured programming - including hierarchical and modular programming - involves the defining of the basic functions and breaking them into smaller functions which correspond to program sections (subprograms). These subprograms perform a stand-alone task and are written separately from the rest of the program (Vakali et al., 1999). In the Scratch programming environment, the subroutines can be defined as "My Commands" (procedures) or as scenarios created by sending messages or creating clones.

The use of procedures, messages, and clones in a Scratch program, gives to the code the characteristics of structured programming. Such characteristics are namely the code modularity, the degree of granulation of the modularity and the hierarchical structure of the segments in tree structures. Other characteristics are the exploitation of shared code, the parameterization of the processes that creates multitool processes, providing the code with flexibility and adaptability (Ladias et al., 2020).

In the next section it will be provided examples of event-driven programming codes that apply structured programming design techniques such as modularity, hierarchical design, shared code, and parametrization.

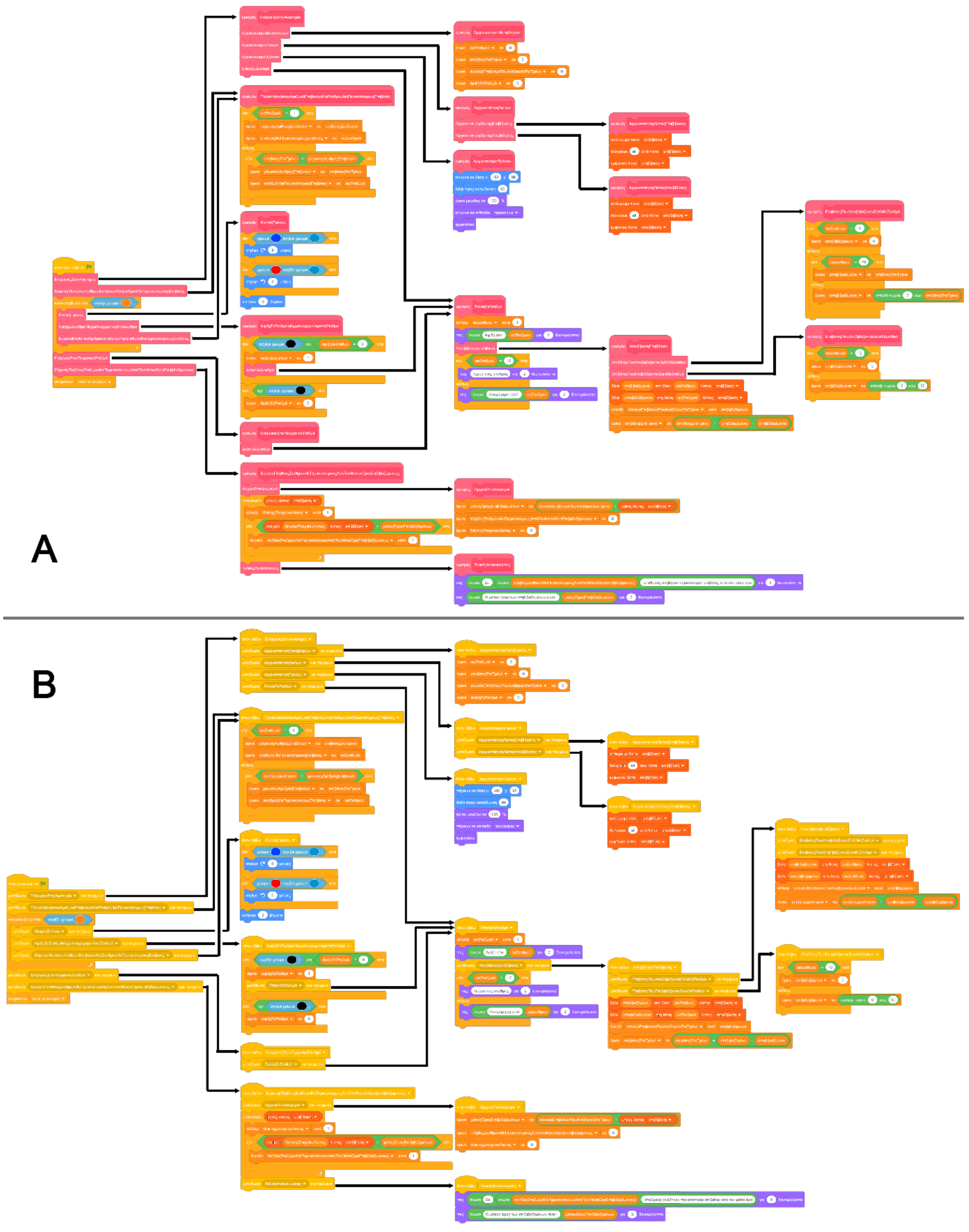
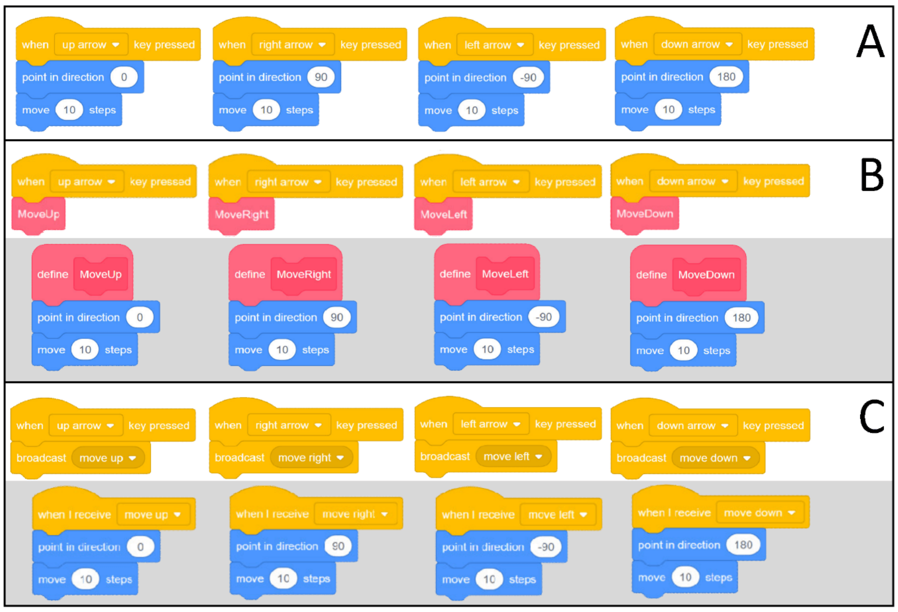


Figure 2 Program development broken down into simpler subprograms. The main program on the left creates a hierarchical (tree-like) structure. Modularity is done by: (A) using procedures, https://scratch.mit.edu/projects/446098222 and (B) sending and receiving messages, https://scratch.mit.edu/projects/446126048

3 Event-driven programming with structured programming elements

3.1 Programming guided by segmented events

An example of event-driven programming is the code in Figure 3A (the same as that corresponding to the monomodal level of the SOLO taxonomy of Figure 1). This code can also acquire structured programming features if, in each of these scenarios, simple subroutines are used (initially). This modularity can be implemented by procedures (Figure 3B) or sending messages (Figure 3C).



```
A
when up arrow key pressed      when right arrow key pressed    when left arrow key pressed     when down arrow key pressed
point in direction 0           point in direction 90           point in direction -90          point in direction 180
move 10 steps                  move 10 steps                   move 10 steps                   move 10 steps

B
when up arrow key pressed      when right arrow key pressed    when left arrow key pressed     when down arrow key pressed
MoveUp                         MoveRight                       MoveLeft                        MoveDown

define MoveUp                  define MoveRight                define MoveLeft                 define MoveDown
point in direction 0           point in direction 90           point in direction -90          point in direction 180
move 10 steps                  move 10 steps                   move 10 steps                   move 10 steps

C
when up arrow key pressed      when right arrow key pressed    when left arrow key pressed     when down arrow key pressed
broadcast move up              broadcast move right            broadcast move left             broadcast move down

when I receive move up         when I receive move right       when I receive move left        when I receive move down
point in direction 0           point in direction 90           point in direction -90          point in direction 180
move 10 steps                  move 10 steps                   move 10 steps                   move 10 steps
```

Figure 3 Example of event-driven programming (A), which implements modularity of its scenarios using procedures (B) or by sending/receiving messages (C)

Comment 1: It should be noted that in Scratch a procedure is defined and called only inside an object. In contrast, the scenarios of messages can be transmitted by one object and received either by scenarios in different objects or by scenarios of the same object. Therefore, the entire code in Figure 3B must belong exclusively to the same object, while the scenarios in Figure 3A and Figure 3B can also belong to different objects.

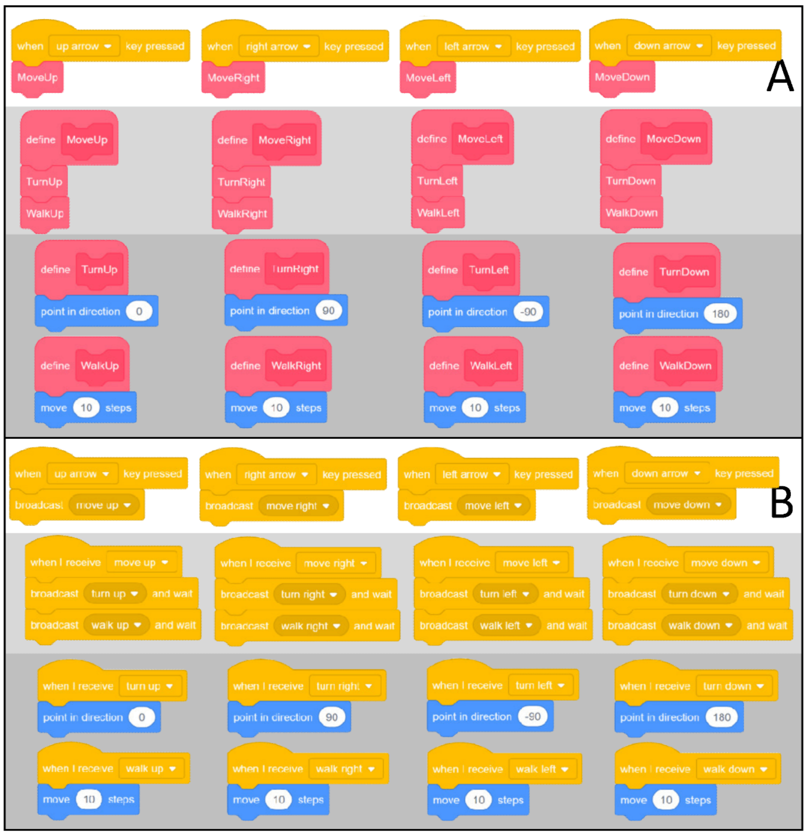
Comment 2: Although the comparison of the program of Figure 3A with programs Figure 3B and Figure 3C shows that the adoption of modularity increases the size and complexity of the code, this is quickly overturned as the overall size of a program increases because the method "divide and conquer". Applied to part-time programming allows the programmer to have control over the flow of the program.

3.2 Programming guided by events with hierarchical design

In each of the programs B and C shown in Figure 3, its code can acquire hierarchical design characteristics if in each scenario the subroutine (process or message reception) is further analyzed in individual subroutines (Figure 4). Thus, in the implementation with procedures (Figure 4A), in the first scenario (when the up-arrow key is pressed), the "Move Up" procedure is first called, which by its definition it then calls the "Show Up" and "Go Up" procedures successively. The same occurs in Figure 4B in the implementation by sending and receiving messages.

Comment 3: When there are more levels of hierarchical design, then the upper levels manage the problem with an abstract approach *hiding the details* which the lower levels they undertake to manage.

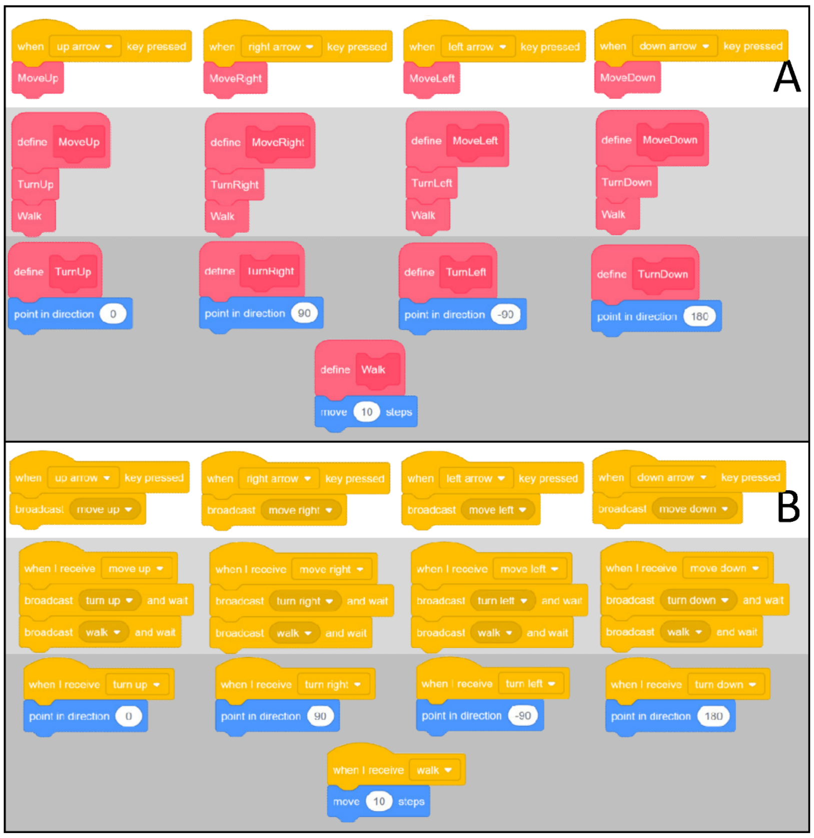
Comment 4: A reasonable question is where the analysis of functions/subroutines towards more simple sections (granulation of modularity). A suggested answer to this question may be that the analysis continues as long as the subproblems/subprograms correspond to some physical processes or logical entities, for which there are words to describe them (Ladias et al., 2017).



```
when up arrow key pressed
MoveUp

when right arrow key pressed
MoveRight

when left arrow key pressed
MoveLeft

when down arrow key pressed
MoveDown

define MoveUp
TurnUp
WalkUp

define MoveRight
TurnRight
WalkRight

define MoveLeft
TurnLeft
WalkLeft

define MoveDown
TurnDown
WalkDown

define TurnUp
point in direction 0

define TurnRight
point in direction 90

define TurnLeft
point in direction -90

define TurnDown
point in direction 180

define WalkUp
move 10 steps

define WalkRight
move 10 steps

define WalkLeft
move 10 steps

define WalkDown
move 10 steps
```

```
when up arrow key pressed
broadcast move up

when right arrow key pressed
broadcast move right

when left arrow key pressed
broadcast move left

when down arrow key pressed
broadcast move down

when I receive move up
broadcast turn up and wait
broadcast walk up and wait

when I receive move right
broadcast turn right and wait
broadcast walk right and wait

when I receive move left
broadcast turn left and wait
broadcast walk left and wait

when I receive move down
broadcast turn down and wait
broadcast walk down and wait

when I receive turn up
point in direction 0

when I receive turn right
point in direction 90

when I receive turn left
point in direction -90

when I receive turn down
point in direction 180

when I receive walk up
move 10 steps

when I receive walk right
move 10 steps

when I receive walk left
move 10 steps

when I receive walk down
move 10 steps
```

Figure 4 Example of event-driven programming (A), which implements modularity of its scenarios using procedures (B) or by sending/receiving messages (C)



```
when up arrow key pressed
MoveUp

when right arrow key pressed
MoveRight

when left arrow key pressed
MoveLeft

when down arrow key pressed
MoveDown

define MoveUp
TurnUp
Walk

define MoveRight
TurnRight
Walk

define MoveLeft
TurnLeft
Walk

define MoveDown
TurnDown
Walk

define TurnUp
point in direction 0

define TurnRight
point in direction 90

define TurnLeft
point in direction -90

define TurnDown
point in direction 180

define Walk
move 10 steps
```

```
when up arrow key pressed
broadcast move up

when right arrow key pressed
broadcast move right

when left arrow key pressed
broadcast move left

when down arrow key pressed
broadcast move down

when I receive move up
broadcast turn up and wait
broadcast walk and wait

when I receive move right
broadcast turn right and wait
broadcast walk and wait

when I receive move left
broadcast turn left and wait
broadcast walk and wait

when I receive move down
broadcast turn down and wait
broadcast walk and wait

when I receive turn up
point in direction 0

when I receive turn right
point in direction 90

when I receive turn left
point in direction -90

when I receive turn down
point in direction 180

when I receive walk
move 10 steps
```

Figure 5 Example of event-driven programming that makes good use of shared code by (A) calling the "Go" procedure and (B) activating the "when I receive the walk" scenario

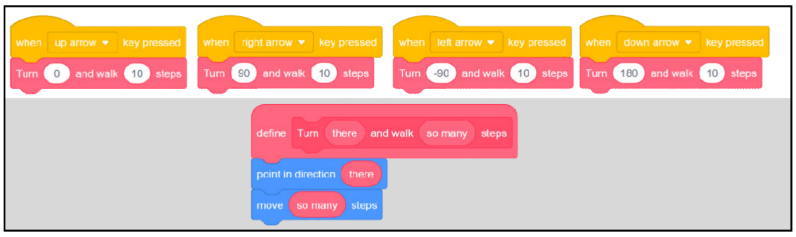
3.3 Shared program-driven programming

In the programs of Figure 4, where the implementations have been held both with procedure and message, we observe that all codes of the last line execute the same command (the “move ten steps”). The four different last line procedures in Figure 4A can be replaced by a shared procedure, as shown in Figure 5A. The four different scenarios of the last line of Figure 4B can be replaced by a shared scenario (Figure 5B).

Comment 5: Identifying the pieces of code that perform the same task and replacing them with shared code has the advantage of limiting the overall code size of a program.

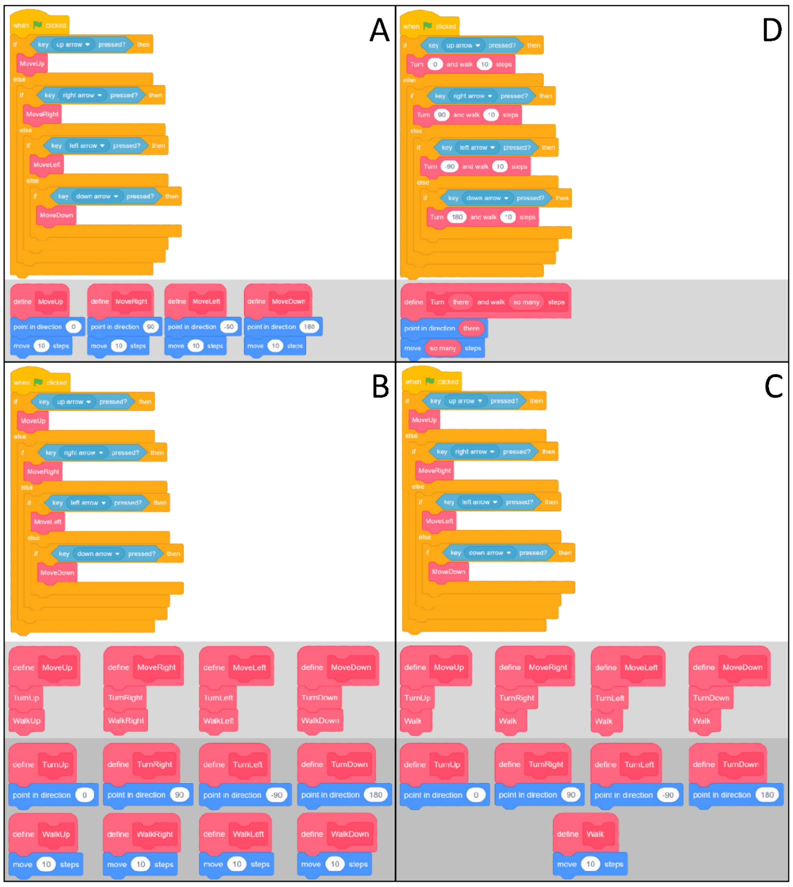
3.4 Programming guided by customizable events

In the programs of Figure 5, which have been implemented using procedures and messages, we observe that all the codes - except of the ones of the initial level - that correspond to each one of the four events, they have a similar structure. This requires that the code should be appropriately configured, as shown in Figure 6, in a way that the detection of each of the four events calls for the same procedure with direction and distance parameters.



```
when up arrow key pressed
Turn 0 and walk 10 steps

when right arrow key pressed
Turn 90 and walk 10 steps

when left arrow key pressed
Turn -90 and walk 10 steps

when down arrow key pressed
Turn 180 and walk 10 steps

define Turn there and walk so many steps
point in direction there
move so many steps
```

Figure 6 Example of an event-driven programming example that makes good use of the capability to parameterize processes. The “Turn there and walk so many steps” process call can be done by passing various values of the “there” and “so many” parameters



```
A
when clicked
if key up arrow pressed? then
  MoveUp
else
  if key right arrow pressed? then
    MoveRight
  else
    if key left arrow pressed? then
      MoveLeft
    else
      if key down arrow pressed? then
        MoveDown

define MoveUp       define MoveRight     define MoveLeft       define MoveDown
point in direction 0  point in direction 90  point in direction -90  point in direction 180
move 10 steps       move 10 steps        move 10 steps         move 10 steps

D
when clicked
if key up arrow pressed? then
  Turn 0 and walk 10 steps
else
  if key right arrow pressed? then
    Turn 90 and walk 10 steps
  else
    if key left arrow pressed? then
      Turn -90 and walk 10 steps
    else
      if key down arrow pressed? then
        Turn 180 and walk 10 steps

define Turn there and walk so many steps
point in direction there
move so many steps

B
when clicked
if key up arrow pressed? then
  MoveUp
else
  if key right arrow pressed? then
    MoveRight
  else
    if key left arrow pressed? then
      MoveLeft
    else
      if key down arrow pressed? then
        MoveDown

define MoveUp   define MoveRight   define MoveLeft   define MoveDown
TurnUp          TurnRight          TurnLeft          TurnDown
WalkUp          WalkRight          WalkLeft          WalkDown

define TurnUp         define TurnRight       define TurnLeft         define TurnDown
point in direction 0  point in direction 90  point in direction -90  point in direction 180

define WalkUp   define WalkRight   define WalkLeft   define WalkDown
move 10 steps   move 10 steps      move 10 steps     move 10 steps

C
when clicked
if key up arrow pressed? then
  MoveUp
else
  if key right arrow pressed? then
    MoveRight
  else
    if key left arrow pressed? then
      MoveLeft
    else
      if key down arrow pressed? then
        MoveDown

define MoveUp   define MoveRight   define MoveLeft   define MoveDown
TurnUp          TurnRight          TurnLeft          TurnDown
Walk            Walk               Walk              Walk

define TurnUp         define TurnRight       define TurnLeft         define TurnDown
point in direction 0  point in direction 90  point in direction -90  point in direction 180

define Walk
move 10 steps
```

Figure 7 Examples of sequential programming in which are applied structured programming techniques, such as (A) modularity of the scenarios, (B) hierarchical design, (C) shared code, and (D) parametrization of the procedure

Comment 6: In the Scratch environment, only the procedures have this potentiality of parametrization, while the messages do not. Locating customizable code snippets has the advantage of creating multi-tooling processes, thus increasing the code's flexibility.

Comment 7: In general, subprograms should perform a single task, to investigate and cover all the cases of the subproblem they are called to solve. They must be autonomous in a way that any changes, corrections or improvements, do not affect outside their limits, while if necessary they can be replaced by another subroutine without the need for any modifications. Moreover, the interface with the code that calls them must be written clearly and visibly, having only one input and one output. They must also have the capability, to be used as black boxes, be small in size, be manageable by the programmer, and be designed with a lego-like philosophy to be reused as elements of library procedures (Ladias, 1991).

The same structured programming design techniques mentioned above will be applied to sequential programming.

4 Sequential programming with structured programming elements

The multidimensional level of Figure 1 shows the "monolithic" serial codes that solve the problem of remote control by the user using the arrow keys on the keyboard. We will work with the code with the nested "if... then... else" structure, seeking to acquire structured programming characteristics using subroutines.

Applying modularity using procedures results in the program of Figure 7A. With a hierarchical design technique, the program evolves into the one of Figure 7B. Subsequently, from the Figure 7B program emerges the shared code of Figure 7C. The previous steps can be implemented by sending/receiving messages depending on Figure 3C, Figure 4B, and Figure 5B programs. By configuring the previous procedures, the program of Figure 7D is obtained.

Comment 8: All comments on structured scheduling characteristics mentioned in the event-driven scheduling section apply to sequential programming.

Then, the same design techniques of the structured programming mentioned above will be applied to parallel programming.

5 Parallel programming with elements of structured programming

The relational level of Figure 1 shows the parallel code that solves the user remote control problem of an object with the keyboard arrow keys.

This code may acquire characteristics of structured programming if within each of these parallel scenarios are used subroutines. While applying modularity using procedures results in the program of Figure 8A. With a hierarchical design technique, the program evolves into that of Figure 8B, from which it also emerges the shared code of Figure 8C. The previous steps can be implemented by sending/receiving messages depending on Figure 3C, Figure 4B, and Figure 5B programs. By configuring the previous procedures, the program of Figure 7D is obtained.

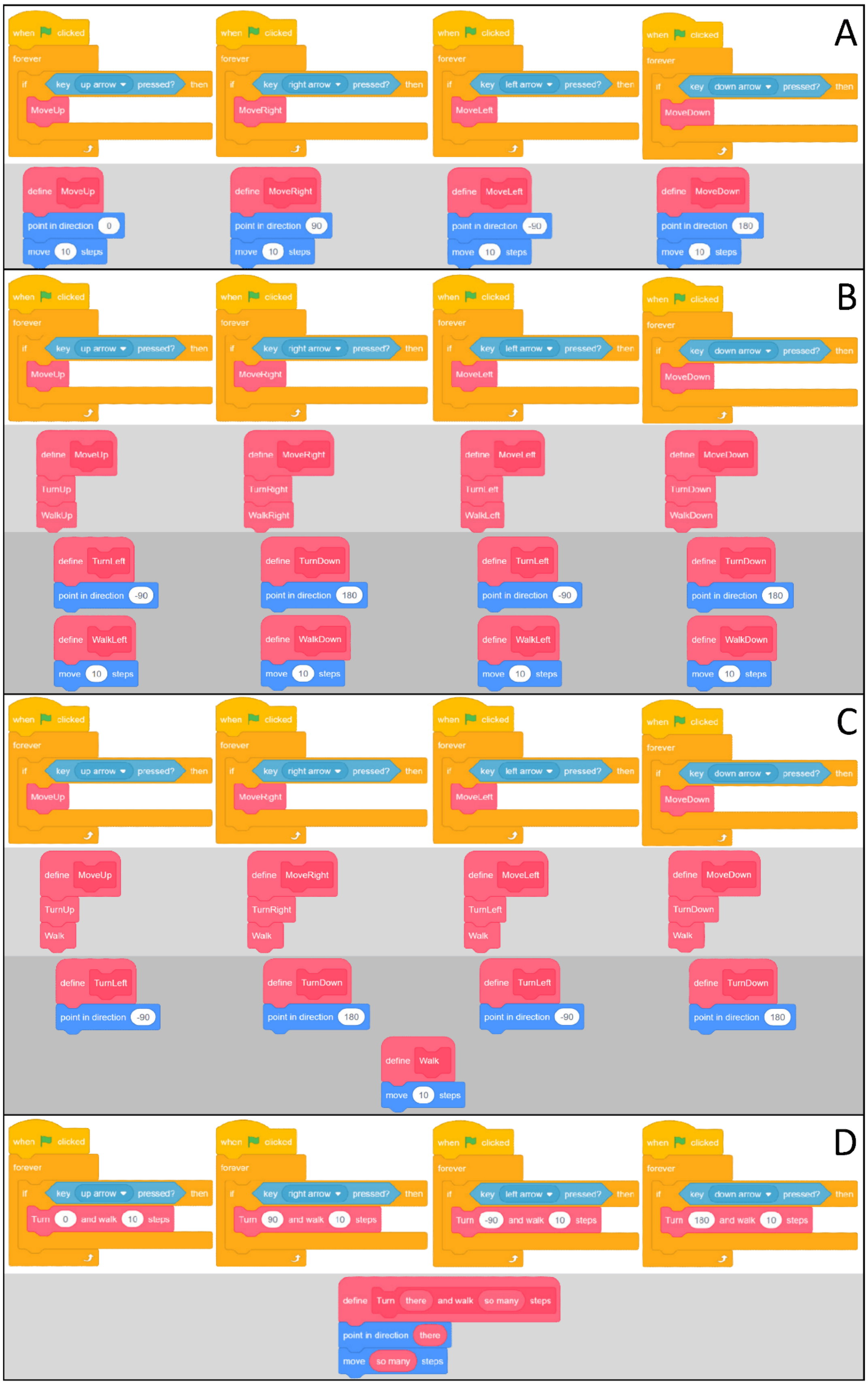
Comment 9: All comments on structured programming features mentioned in the event-driven programming section can also apply to parallel programming.

According to the SOLO taxonomy, all of the previous codes will be attempted to be classified according to the assessed difficulties during their teaching and based on the expected learning outcomes.

6 Programming styles and SOLO taxonomy

As mentioned in the introduction, SOLO (Structure of Observed Learning Outcomes) taxonomy proposes the assessment of knowledge based on the structure of the observed learning outcome, classifying these outcomes into the five hierarchical levels of the SOLO taxonomy.

We will now attempt to correspond, the influences of the various characteristics of structured programming based on the three programming styles (event-based programming, sequential programming, and parallel programming) to the levels of the SOLO taxonomy. First, at the pre-structural level of SOLO taxonomy that refers to the use of unrelated and disorganized information that does not make sense, in terms of the structured programming dimension, all programs contained in Figure 1 are classified, indicating the lack of structured programming influences in the three aforementioned programming styles. Second, at the Uni-structural level,



```
A
when [flag] clicked            when [flag] clicked               when [flag] clicked              when [flag] clicked
forever                        forever                           forever                          forever
  if key up arrow pressed? then  if key right arrow pressed? then  if key left arrow pressed? then  if key down arrow pressed? then
    MoveUp                         MoveRight                         MoveLeft                         MoveDown

define MoveUp                  define MoveRight                  define MoveLeft                  define MoveDown
point in direction 0           point in direction 90             point in direction -90           point in direction 180
move 10 steps                  move 10 steps                     move 10 steps                    move 10 steps
```

```
B
when [flag] clicked            when [flag] clicked               when [flag] clicked              when [flag] clicked
forever                        forever                           forever                          forever
  if key up arrow pressed? then  if key right arrow pressed? then  if key left arrow pressed? then  if key down arrow pressed? then
    MoveUp                         MoveRight                         MoveLeft                         MoveDown

define MoveUp                  define MoveRight                  define MoveLeft                  define MoveDown
TurnUp                         TurnRight                         TurnLeft                         TurnDown
WalkUp                         WalkRight                         WalkLeft                         WalkDown

define TurnLeft                define TurnDown                   define TurnLeft                  define TurnDown
point in direction -90         point in direction 180            point in direction -90           point in direction 180

define WalkLeft                define WalkDown                   define WalkLeft                  define WalkDown
move 10 steps                  move 10 steps                     move 10 steps                    move 10 steps
```

```
C
when [flag] clicked            when [flag] clicked               when [flag] clicked              when [flag] clicked
forever                        forever                           forever                          forever
  if key up arrow pressed? then  if key right arrow pressed? then  if key left arrow pressed? then  if key down arrow pressed? then
    MoveUp                         MoveRight                         MoveLeft                         MoveDown

define MoveUp                  define MoveRight                  define MoveLeft                  define MoveDown
TurnUp                         TurnRight                         TurnLeft                         TurnDown
Walk                           Walk                              Walk                             Walk

define TurnLeft                define TurnDown                   define TurnLeft                  define TurnDown
point in direction -90         point in direction 180            point in direction -90           point in direction 180

                                         define Walk
                                         move 10 steps
```

```
D
when [flag] clicked            when [flag] clicked               when [flag] clicked              when [flag] clicked
forever                        forever                           forever                          forever
  if key up arrow pressed? then  if key right arrow pressed? then  if key left arrow pressed? then  if key down arrow pressed? then
    Turn 0 and walk 10 steps       Turn 90 and walk 10 steps         Turn -90 and walk 10 steps       Turn 180 and walk 10 steps

                         define Turn (there) and walk (so many) steps
                         point in direction (there)
                         move (so many) steps
```

Figure 8 Example of parallel programming in which are applied structured programming techniques, such as (A) the modularity of scenarios, (B) the hierarchical design, (C) the shared code, and (D) the parametrization of procedures

which is limited to the trading of an item, the programs of Figure 3B, Figure 3C, Figure 7A and Figure 8A are classified at this level, in which a one-dimensional and "shallow" modularity is implemented, ignoring the other components of structured programming in the three aforementioned programming styles. Third, at the *Multi-structural level*, although a multi-point perspective is observed, a complete picture has not yet been formed. The programs shown in Figure 4A, Figure 4B, Figure 7B, and Figure 8B are classified. which implement modularity either using procedures or a sending/receiving messages mechanism. They also contain though hierarchical design characteristics, thus producing tree structures in the aforementioned three programming styles. Forth, at the *Relational level*, a holistic perspective is observed. At this level, the programs of all three programming styles mentioned above are classified. In addition to the previous features of structured programming of the multi-structural level, the users now can generate and make good use of existing shared code via multiple shared calling procedures or by activating alternative shared scenarios. Examples of this category are the programs of Figure 5A, Figure 5B, Figure 7C, and Figure 8C. Last, at the *level of Extended Abstract*, in addition to the relational level characteristics, the code is considered a more general case in which the codes of the previous levels are treated as snapshots. Such codes are configured processes that provide flexibility in the program used as "multitools" and library elements with examples of the codes of Figure 6A, Figure 6B, Figure 7D, and Figure 8D.

In the initial work (Ladias et al., 2020), it was created a one-dimensional table with the programming styles (the programming guided by events, sequential programming, and parallel programming) and their classification at the SOLO taxonomy levels. In the present work, each of these programming styles was further examined in terms of their influences on the characteristics of structured programming. The result is a two-dimensional table (see Table 1 below), in which one dimension is the type of programming style and the other is the characteristics of the structured programming, classified to the levels of the SOLO taxonomy. The cells in the table correspond to the codes used in this work. The empty cells of the last line can be filled with a wide variety of codes resulting from "mixes" between programming styles and combinations of individual characteristics of structured programming.

Table 1 Matching of programming styles and the effects on them of the characteristics of structured programming

				Structure of observed learning outcomes (SOLO) taxonomy				
				Pre-structural	Uni-structural	Multi-structural	Relational	Extended abstract
				Structured programming				
					Modularity	Hierarchical design	Shared code	Parametrization
SOLO levels	Uni-structural	Programming styles	Event-driven	1A	3B, 3C	4A, 4B	5A, 5B	6
	Multi-structural		Sequential	1B	7A	7B	7C	7D
	Relational		Parallel	1C	8A	8B	8C	8D
	Extended abstract		Combination of – various styles	1D				

7 Conclusions

This paper is part of a Scratch code evaluation project. The correspondence of programming styles and the SOLO taxonomy levels, combined with students' influences from structured programming elements, can help develop criteria. These criteria can be used in an evaluation system for code written by novice programmers.

Table 1 can also be used by the designers of a programming study program. Specifically, Table 1 can help them suggest the main navigation path in this educational content by following neighboring cells. On the one hand, these can enable exploratory learning with reduced guidance and help from the teacher (scaffolding) and, on the other hand, a spiral approach. In addition, the teacher can use Table 1 as a guide to determine alternative paths to follow to teach the educational content by adapting it to the classroom circumstances.

A suggested step for further investigation is for researchers to look for possible relationships in Table 1 with object-oriented programming features. Scratch is an object-based programming environment considering the analogies to object-oriented environments, whereas Scratch objects correspond to and clone in objects.

References

Biggs, J. B., & Collis, K. F. (1982). Evaluating the quality of learning. The SOLO taxonomy. NY: Academic Press.

Doukakis, S. (2019). Exploring brain activity and transforming knowledge in visual and textual programming using neuroeducation approaches. AIMS neuroscience, 6(3), 175-190. https://doi.org/10.3934/Neuroscience.2019.3.175

Karvounidis, T., Argyriou, I., Ladias, An., & Douligeris, C. (2017). A Design and Evaluation Framework for Visual Programming Codes. EDUCON. Athens.

Ladias, A. (1991). Pascal, a methodical approach. Kleidarithmos: Athens.

Ladias A., Karvounidis T., & Ladias D. (2017). Partitioning the code in a Scratch visual programming environment. In Proceedings of the 11th Panhellenic Conference of Informatics Teachers (PEKAP), Chalkida, Greece.

Ladias, D., Karvounidis T., & Ladias, A. (2020). Categorization in the SOLO taxonomy of programming styles in Scratch. Erkyna, 21, 89-100.

Papert, S. A. (1991). Mindstorms: Children, computers, and powerful ideas. Basic books.

Papadakis, S., & Kalogiannakis, M. (2019). Evaluating the effectiveness of a game-based learning approach in modifying students' behavioural outcomes and competence, in an introductory programming course. A case study in Greece. International Journal of Teaching and Case Studies, 10(3), 235-250.

Papadakis, S. (2020). Evaluating a Teaching Intervention for Teaching STEM and Programming Concepts Through the Creation of a Weather-Forecast App for Smart Mobile Devices. In Handbook of Research on Tools for Teaching Computational Thinking in P-12 Education (pp. 31-53). IGI Global.

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

Vakali, A., Giannopoulos, I., Ioannidis, N., Koilias, Ch., Malamas, K., Manolopoulos, I., & Politis, P., (1999). Development of Applications in Programming Environment (C Lyceum) - Student Book. ITYE "DIOFANTOS. "

Vidakis, N., Barianos, A. K., Trampas, A. M., Papadakis, S., Kalogiannakis, M., & Vassilakis, K. (2019). in-Game Raw Data Collection and Visualization in the Context of the "ThimelEdu" Educational Game. In International Conference on Computer Supported Education (pp. 629-646). Springer, Cham.