Teaching Concurrent Programming Concepts Using Scratch in Primary School: Methodology and Evaluation

https://doi.org/10.3991/ijep.v8i4.8216

Eleni Fatourou, Nikolaos C Zygouris(✉),
Thanasis Loukopoulos, Georgios I Stamoulis
University of Thessaly, Lamia, Greece
nzygouris@uth.gr

Abstract—Computer programming can help children develop problem solving and analytical skills. Thus, many countries have included computer science in the curriculum of primary school. Given differences in culture, available infrastructures, as well as the age pupils are introduced to computer science, forming a computer science curriculum still remains a challenge. Towards this end, this study focuses on exploring the potential merits of introducing concurrent programming concepts early in the learning process. The basic premise is that although concurrent programming at its full details is a rather advanced topic even at university level, it is everyday practice to perform two or more tasks simultaneously that might need (or not) some sort of synchronization. Therefore, the tutor can capitalize on everyday experience to explain basic concepts on concurrency. Such correlation between life experience and concurrent programming challenges may expand the cognitive functions of the pupils and provide them with further background to improve analytical thinking. The proposed curriculum for fifth and sixth grade primary school was adopted in seven classes in Greece. Results indicate that uninitiated to programming pupils at the age of ten (fifth grade) were able to comprehend basic concurrency topics, while pupils at the age of eleven (sixth grade) with some programming familiarity were able to understand more advanced concepts.

Keywords—concurrent programming, computer programming, constructivism, Scratch, primary school

1 Introduction

Computer programming is considered a basic literacy in the digital age helping children to develop creative problem solving skills, logical thinking and mental flexibility. As indicated by European Schoolnet in [6], only ten European countries have fully integrated computer programming in their primary school curriculum, as of 2015. Given differences in culture, available infrastructures, as well as the age pupils are introduced to computer science, the challenge of forming a computer science curriculum that not only offers basic background but expands the cognitive horizon and cultivates the imagination of students, still remains a challenge.

Early computer science courses typically focus on structured programming concepts such as: control/selection constructs and iterations. The primary educational effort (particularly in the case of Greece) is tailored towards applying these building blocks into single thread execution scenarios, overlooking scenarios with concurrent multiple threads. However, concurrency rises more than often in educational programming platforms such as Scratch. For instance, when two independent entities, e.g., sprites, move and act in a labyrinth, it is not uncommon that racing conditions appear, whenever the entities require simultaneous access to a common resource, e.g., some treasure object. As a result, pupils might experience "unexpected" program behavior and occasional program crashes. The teacher has then two practical options: (i) either overlook the problem, diminishing its importance, and continue focusing on single thread correctness criteria, or, (ii) attempt to give a thorough explanation of the reasons of such "unexpected behavior", thus, introducing concurrency issues to pupils, albeit in an ad-hoc, unstructured and unplanned manner which might prove discouraging.

Motivated by: (i) the apparent "knowledge gap" concerning concurrency that exists on many typical early computer programming syllabuses, (ii) the fact that pupils are accustomed to multitasking in their everyday life and (iii) the importance of multithreading and multitasking in modern software and hardware, this study investigates the enrichment of a typical syllabus with multithreading concurrency issues. The main aim is to introduce the pupils to the basic challenges of concurrent programming in a systematic manner, without sacrificing the level of detail contained on a typical syllabus as far as simple single thread structured programming is concerned. The developed syllabus was tailored and evaluated for the education system of Greece, whereby pupils are introduced to programming concepts at the age of ten using Scratch.

The rest of the paper is organized as follows. Section 2 presents the related work from the particular standpoint of the pedagogical approach followed (constructivism). Section 3 illustrates the methodology together with the proposed syllabuses. Section 4 gives an overlook of the main computer programming pitfalls experienced by students and the pedagogical approach to tackle them. Section 5 includes the evaluation setup and discusses results. Finally, Section 6 provides the concluding remarks.

2 Related work

The core approach of the study adheres to the constructionist theory, according to which the learning process is not only transmitted from teacher to pupil, but rather constructed in the mind of the pupil in the form of active learning [16], [18]. Constructivism theorists such as Piaget and Papert view children as the builders of their own cognitive tools, as well as their external realities [1]. Moreover, Papert believes that programming has a tremendous potential to improve classroom teaching [13]. Thus, the dominant theory of learning, supports that knowledge is actively constructed by the pupil and not passively absorbed from text books and lectures [1].

Constructivism has been intensively studied by researchers of science and mathematics education [1]. However, there has been much less work on constructivism in computer science education [2]. In 1980 computer programming projects become to appear with more frequency in schools, but observations of student learning did not always match the powerful claims. The ideal vision of students' becoming better due to hands on Logo learning collided with the documented reality of students' difficulties to learn even fundamentals of Logo [11]. Nowadays, research has entered into a new phase of multidisciplinary theory based protocols [5], [11]. The initial vision of teaching and learning computer programming has been altered. The focus of current researches is on understanding the conditions under which the skills that are learned in programming can transfer to cognitive development of learners [3]. For instance in [10] it was concluded that programming in pairs (a common situation due to laboratory restrictions in schools) has limitations, while in [7] it was pointed out that despite its original limitations, the newer versions of Logo with enhanced graphics and interface might find applications in pre-school ages.

Visual programming languages such as Scratch have been widely adopted recently as the means for early introduction to programming concepts. Scratch uses blocks, which the pupils drag and drop to form their scripts. An avid research interest exists on how to fine tune the learning process with Scratch in order to achieve the best pedagogical results in primary schools [8], [12], [17], but also in elementary ones [14]. Towards, this end the research presented in this paper aims at filling a tutoring gap that often appears when following a classic introductory syllabus to Scratch programming, namely the teaching of concurrency concepts.

3 Methodology

3.1 Educational context and targets

The application of the methodology was performed in the Greek primary school whereby computer programming is taught at the last two years of the school (ages ten and eleven). The official curriculum involves a total of 12 weeks of hourly laboratory lessons per year at each class (fifth and sixth grade). Applying the constructionist theory in the present study required the design of programming challenges that are incremental in nature and led after a certain point to concurrency problems that were self evident. It is straightforward that the proposed syllabuses should adhere to official curriculum constraints (for evaluation reasons). The rather limited timeframe for the computer programming courses offered a serious challenge in defining the educational goals and design a subsequent plan to achieve them.

The research conducted involved both classes that were already familiar with basic structured programming concepts and classes with no prior programming experience. As a result, it was decided that two different projects should be implemented. Instructional scaffolding was used for the learning process. Each project was split into equally hourly tasks. Pupils worked on the same file, extending or changing game functionality. The teaching approach followed, was to introduce the notion of multiple running threads early on and incrementally build knowledge on concurrency issues according to the assigned tasks. A Scratch player often executes "simultaneous" scripts, so when different scripts are triggered by the same event, pupils must check if race conditions apply and if so, learn a way to tackle the problem. Building around this feature, tasks were escalated in order for pupils to achieve the following educational targets:

T1. Implement concurrent moves of a single sprite;

T2. Implement concurrent moves of two or more sprites;

T3. Synchronize sprites using time primitives;

T4. Synchronize sprites using messages;

T5. Distinguish local, sprite-level variables from global variables;

T6. Synchronize sprites using condition variables.

The first four educational targets concerned both the beginners and the more advanced classes, while the last two were only attempted for pupils with prior programming knowledge. It should be noted that the aforementioned six learning objectives were on top of the classic targets related to basic structural programming constructs.

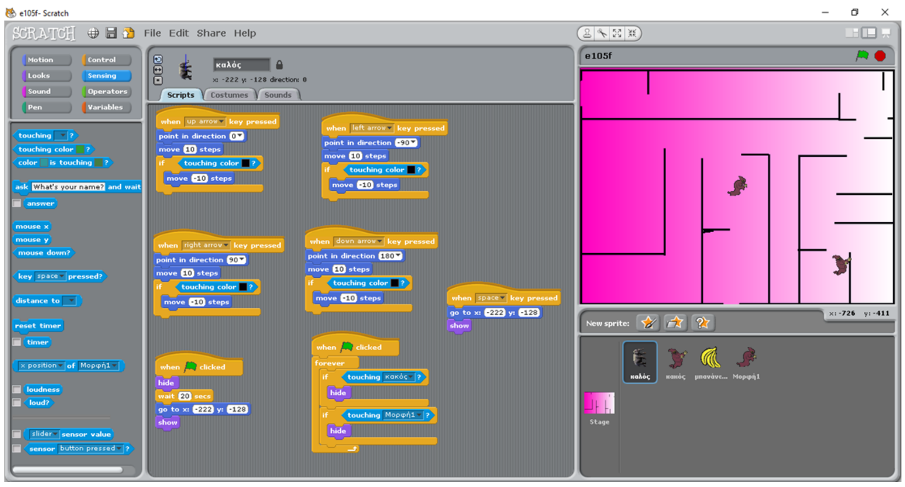
3.2 Beginners' syllabus

The beginners' project concerned a maze game whereby a hero sprite tries to capture a trophy, while chased by enemy sprites. Maze games are very common Scratch projects and are suitable for beginners. As a testament a google search for "maze", performed on 16/5/2017 on the site https://scratch.mit.edu returned roughly 140.000 results. The syllabus presented in Table 1 is designed to incrementally build fundamental programming knowledge, while introducing concurrency concepts and solutions, in a gradual self evident manner. It also contains a midterm and a final project presentation. In the table the intermediate checkpoints for achieving the desired learning objectives are also shown. Aside from T2 for which two checkpoints exist, one for handling two sprites and one for more than two, all other learning objectives were associated with a particular week. On the respective week, the progress of student projects was evaluated according to the corresponding learning objective by the teacher and without the students being aware that an evaluation took place. This was done in order to provide better guidance to students both before and after the midterm project milestone.

Table 1. Beginners' syllabus outline

	Plan	Objectives	Concurrent issues addressed
1	Draw a maze stage. Make a new sprite (hero).	Draw a maze. Distinguish sprite from stage.	N/A
2	Use commands to move the sprite from the beginning of the maze to the end.	Execute a sequence of blocks.	N/A
3	Make the sprite move using arrow keys.	Use a trigger event to start a script.	(T1) Concurrent scripts on a single sprite. Pupils explain what happens if accidentally the same key is used for up and down movement.
4	Draw two or more sprites as enemies. Use command forever to make the enemies move around constantly when green flag is clicked.	Use iteration (forever)	(T2) Two sprites move at the same time. But no synchronization issues exist yet.
5	Create trophy sprite for the hero and use command if to make trophy disappear when touched by the hero.	Use iteration and condition. Use hide command.	A script loops until a condition applies but no synchronization issues apply yet.
6	Add a script to the hero, to make it disappear when touched by an enemy.	Use iteration and condition. Use hide command.	(T2) Concurrent scripts of two or more sprites. Place an enemy on food and make hero touch them
7	Midterm presentation		
8	Make nice guy, enemies and food appear in certain places when game starts.	Initialization. Use show command.	Concurrent scripts of two or more sprites. If show command proceeds move, then race conditions apply.
9	Make stage present an introductory message.	Use wait command.	(T3) Synchronize sprites using time primitives. Upon the game starts all sprites wait for a few seconds for the salutation to disappear
10	Make stage present a winning or losing message.	Use message passing.	(T4) Synchronize sprites using messages.
11	Add any functionality to the project.	Self assessment.	Self assessment.
12	Final project presentation		

An example screenshot from a final project handed down by an average performing fifth grade student is shown in Fig. 1.



```
when up arrow key pressed
point in direction 0
move 10 steps
if touching color ■ ?
    move -10 steps

when left arrow key pressed
point in direction -90
move 10 steps
if touching color ■ ?
    move -10 steps

when right arrow key pressed
point in direction 90
move 10 steps
if touching color ■ ?
    move -10 steps

when down arrow key pressed
point in direction 180
move 10 steps
if touching color ■ ?
    move -10 steps

when space key pressed
go to x: -222 y: -128
show

when ⚑ clicked
hide
wait 20 secs
go to x: -222 y: -128
show

when ⚑ clicked
forever
    if touching κακός ?
        hide
    if touching Μορφή1 ?
        hide
```

Fig. 1. Example of a final project by an average performing fifth grade student. Three sprites and a thesaurus exist (bananas). The hero sprite (elephant) is controlled by 7 scripts. It can be observed that two scripts execute when green flag is pressed. The initialization correctly uses time based synchronization (wait 20 secs) but is not completely correct (the two scripts should have been combined into one).

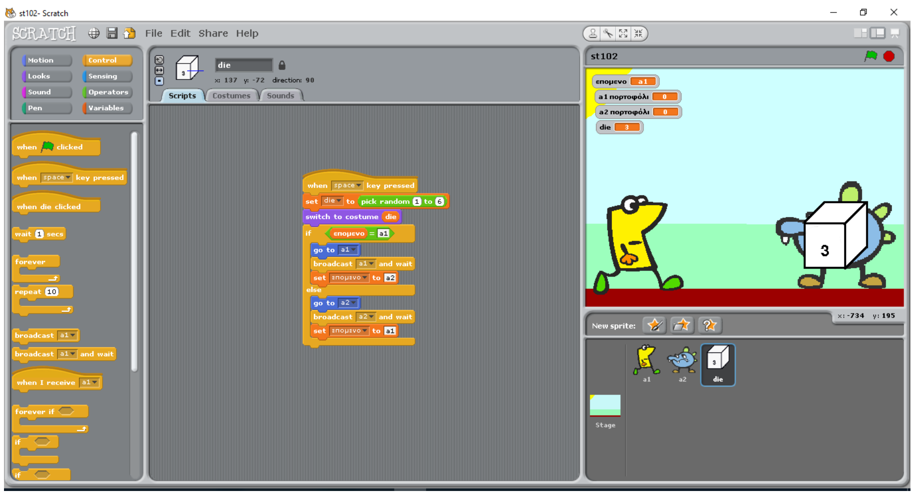
3.3 Advanced syllabus

The more advanced classes were already introduced (previous year) to basic programming concepts with Scratch. However, the syllabus used for the introduction differed from the one in Table 1, focusing only on structured programming constructs using a single thread view of the executed scripts. Therefore, it was deemed that knowledge on concurrency concepts should be built from scratch, albeit at a faster pace compared to the beginners. Table 2 illustrates the syllabus for the advanced classes. As it can be observed, apart from the heaviest workload on concurrency topics (T5 and T6 learning objectives are not included in Table 1), it contains two hours (instead of one) of free project additions and self-assessment. This served two purposes. Firstly, it encouraged a self-motivation attitude and secondly it served as a means of equalizing the effects of "missed hours" between classes that completed the 12 hour schedule and those that only completed 11 hours (due to national holidays).

Table 2. Advanced project outline

	Plan	**Objectives**	**Concurrent issues addressed**
1	Create two sprites representing players. Create a second costume for each one holding a die. Make them change costume on space pressed.	Distinguish sprites and cos-tumes.	(T2) Concurrent scripts of two or more sprites.
2	Change in previous game. Make sprites change costume every 2 seconds.	Use timer.	(T3) Synchronize sprites using time primitives.
3	Delete costumes with the die. Create a die sprite. On click, the die goes to the other player. Use a variable to hold the dice owner.	Use variables.	N/A
4	When a player receives the die, says "I got it"	Use messages.	(T4) Synchronize sprites using messages.
5	Draw a die and cast it randomly when clicked.	Use random function.	N/A
6	Make the die turn for 2 seconds until it shows the result when space is pressed.	Use wait.	(T1) Concurrent scripts of a single sprite. When space is pressed twice within 2 se-conds the interrupt is ignored
7	Midterm presentation		
8	Create a local variable pocket for each player and a global for the die. Initialize them. Each time the die is cast, it adds one to the players' wallet.	Declare and use local and global variables.	(T5) Distinguish local, sprite-level variables from global variables.
9	Give or take money from the players depend-ing on the value of the die and the player's turn.		(T6) Synchronize sprites using condition variables.
10	Add any functionality to the project.	Self assessment.	Self assessment.
11	Add any functionality to the project.	Self assessment.	Self assessment.
12	Final project presentation.		

An example screenshot from a final project handed down by an average perform-ing sixth grade student is shown in Fig. 2.



```
when [space] key pressed
set [die] to (pick random 1 to 6)
switch to costume (die)
if <(επομενο) = a1>
    go to [a1]
    broadcast [a1] and wait
    set [επομενο] to a2
else
    go to [a2]
    broadcast [a2] and wait
    set [επομενο] to a1
```

Fig. 2. Example of a final project by an average performing sixth grade student. The die script involves notifying the two players concerning whose turn it is. Notification was done by broadcasting and corrected to broadcast and wait by the teacher. This was necessary in order to block the dice script until players' scripts finish. Without blocking there is a risk that the dice script will continue executing and perform the assignment that follows the broadcast, which changes players' turn. Thus, a player could miss a turn.

4 Common programming pitfalls

The common programming errors made by students during the study can be categorized across two dimensions. The first concerns whether the error was related to concurrency (C) or not. The second is about the difficulty of detecting the error. Errors that proved difficult to be detected by students are denoted by (H) while the easier ones by (E). It should be noted that (E) errors usually led directly to abnormal behavior of sprites in the screen. Thus, students asked for teacher's help before moving to other aspects of their projects, leading to consolidating knowledge in a stepwise fashion.

Hard to be detected errors (H) rose from two causes. The first one was due to students not checking in a thorough manner the behavior of their programs. For instance an error that would have otherwise been apparent is hidden because the test runs never included the corresponding triggering event. The second type of hard to detect errors concerned the inherent at concurrent programming difficulty of reproducing errors. Namely, an error that occurs at some run (due to a particular script/thread execution combination) may prove difficult to repeat as script/thread execution sequence changes among different runs. This might cultivate a tendency at students of ignoring a particular faulty run because the remaining ones were correct, presumably attributing the fault to some system or environment glitch. In order to increase the likelihood of detecting immediately both kinds of (H) errors, the following steps were taken:

- A list of test cases was given to students, together with instructions of covering most or all event triggering cases;
- Students were instructed (especially in lessons where concurrency issues were involved) not to ignore faulty runs but alert the instructor immediately;
- Students were instructed to run their code repeatedly;
- Before a lab session, students' projects were checked for errors by the instructor.

The list of common errors follows.

- Wrong order of commands (E). For instance a sprite first moves and then turns instead of doing the opposite. Such errors occurred mostly at beginner level;
- Incorrect use of iteration (E). Infinite loops as well as lack of loops were a common mistake both at beginner and at advanced level;
- Wrong structure of events (E). The actions concerning an event, e.g., right arrow click, are erroneously characterized as belonging to another event, most commonly green flag click, i.e., program start event;
- Multiple scripts for the same event creating faulty behavior (C, E). This was quite common among beginners particularly at the initialization event (green flag click). The essence of the error is that students did not comprehend (at that point) that the actions invoked for the event should be done in a sequential manner. Instead, by splitting the actions into two or more scripts the actions' execution order could be arbitrary. For instance, consider an initialization script that first places a hero sprite at a safe position and then shows it. If these two actions are split into two scripts then it is possible to first show the hero script at an uninitialized position, whereby a monster sprite is placed ending the game abruptly.
- Incorrect identification of the sprites that need synchronization (C, H). For instance, many students synchronized the thesaurus with the hero sprite, but not with the enemies. This might create a conflict if the hero and an enemy advance to the thesaurus "simultaneously".
- Using conditional global variables for synchronization without proper initialization (C, E). Variable initialization errors occurred unexpectedly often hence, although simple in nature are reported here. As a side note it should be mentioned that synchronization using global/shared variables, requires in principle the variables to be atomic, i.e., no two threads should gain access concurrently. Although in Scratch atomic variables are not directly supported, the selected game play (die casting) made it difficult for racing conditions to appear. Thus, given the available time schedule, a necessary compromise was reached whereby students were not taught of atomicity issues but were taught of how to use atomic variables for synchronization.
- Using broadcast messages instead of broadcast and wait (C, H). The difference between the two message sending primitives is quite subtle. The second primitive pauses the script until all the receivers of the message terminate.

Piaget and Vygotsky as constructivists suggest that students bring their prior knowledge and experiences into any learning process which in turn influence the way they respond to new information. It is further suggested that students frequently resist changing their minds until data to the contrary is overwhelming. [4]. If students' already constructed problem solving models cannot be implemented in solving newly faced problems they (with the appropriate guidance of the tutor) form models that become plausible and fruitful [9]. Part of what qualifies as good teaching methodology discovers what students already believe and creates the required cognitive disagreement leading to the hard work of adjusting their conceptual understanding [15]. Such conflicts occurred often in the classroom and learners (with teacher's guidance) had to reconstruct their ideas when the desired outcome was not shown in the screen of their computer.

5 Evaluation

5.1 Participants

The syllabuses described in Sec. 3.2 and 3.3 were evaluated in 7 primary school classes in Greece. The total number of participating students was 123 of which 66 males (age range 10-11 years old M=10.59, SD=0.495) and 57 females (age range 10-11 years old M=10.51, SD=0.504). According to age and prior knowledge at computer programming the evaluation group exhibited the following characteristics: (i) 55 students were of fifth grade primary school and completely novice to computer programming, (ii) 68 students attended the sixth grade and (iii) among the sixth grade student only 42 had attended a computer programming lab at fifth grade, while 26 were novices. It should be mentioned that such differences on the knowledge level among students of the same grade are not uncommon in Greek primary schools since there is no fixed Computer Science curriculum (just generic guidelines) and there are no fixed standards concerning lab hardware (many schools experience hardware shortage).

The basic syllabus presented in Sec. 3.2 was followed by the fifth grade students and the 26 novice students of sixth grade for a total of 81 students. The advanced syllabus (Sec. 3.3) was followed by the 42 sixth grade students that had some prior experience with programming in Scratch. Of the 123 participants, 83 had a personal computer station, while 40 students had to share a computer at groups of two and sometimes three. The implementation of the curriculum took place during school time in the class of Informatics. Additionally, all children that participated in the present study did not have a history of major medical illness, psychiatric disorder, developmental disorder or significant visual or auditory impairments according to their medical reports available at their schools.

Table 3. Performance in the first four objectives of Sec. 3.1 (123 total students)

Learning Objectives	Students Achieving Objective	Ratio (%)
T1	105	85.4%
T2	108	87.8%
T3	67	54.5%
T4	26	21.1%

5.2 Comprehension of learning objectives

First, results are presented concerning the evaluation of the learning objectives and how many students managed to accomplish them. The evaluation was done over the final project. Table 3 summarizes the performance of the students on the first four learning objectives which were common in both the beginner and advanced syllabus.

From the results, it is clear that the first two objectives that concerned the concurrent movement of single and multiple sprites were achievable by the vast majority of the students. It is also moderately encouraging that more than half of the students managed to successfully implement sprite synchronization using time primitives (T3). This is presumably due to the fact that time based synchronization is closer to real life experiences rather than message based (T4) which was only successfully incorporated in the projects of roughly 1 out of 5 students. Delving more on the results, Table 4 characterizes students' performance based on age. It also includes results from one way ANOVA between the performance of the two age groups.

Table 4. Performance according to age (fifth grade: age 10, sixth grade: age 11, **p<0.01)

Learning Objectives	Fifth grade students (55 total)		Sixth grade students (68 total)		ANOVA
	Students Achieving Objective	Ratio (%)	Students Achieving Objective	Ratio (%)	F
T1	41	74.5%	64	94.1%	9.491**
T2	46	83.6%	62	91.2%	1.121
T3	32	58.2%	35	51.5%	0.443
T4	4	7.3%	22	32.4%	12.742**

As it is apparent, the majority of students successfully comprehending synchronization using messages belong to age group of eleven years old (sixth grade). This is an indication that T4 topic was not taught at fifth grade to a sufficient extend (only at week 10 according to syllabus) and at least one more lecture was needed. Given the tight constraints on primary school schedule in Greece it might be worth considering removing the topic of T4 from fifth grade and use the extra slot to further improve comprehension of T3. Similarly, the especially high ratios for T1 and T2 in sixth grade reveal a possible option of adapting the advanced syllabus so that T1 and T2 context occupies one instead of two weeks, leaving the extra slot to be used for deepening the comprehension of T4.

Table 5. Performance in the last two objectives of Sec. 3.1 (42 students)

Learning Objectives	Students Achieving Objective	Ratio (%)
T5	17	40.5%
T6	20	47.6%

Evaluation according to learning objectives T5 and T6 is presented in Table 5. Recall, that the advanced syllabus containing T5 and T6 related topics was only followed by sixth grade students that had some prior experience with Scratch. The results show that a portion (in the mid-range) of students, managed to acquire the extra background offered by T5 and T6 successfully.

5.3 Statistical analysis

Analysis according to gender revealed that aside from T2 there were no other statistically significant performance differences between male and female participants. In T2 male students achieved a better understanding (M=1.06, SD=0.24) compared to female students (M=1.19, SD=0.40), while ANOVA gave F=5.113 with p=0.025. This result is somehow surprising since T2, i.e., concurrent move of multiple sprites is an easier topic when compared for instance against T3 which involves time based synchronization.

Subsequently, a one way ANOVA was performed in order to identify differences between the group of children that did not have to share their computer station and those who did. Results are presented in Table 6. As it can be observed, sharing a computer has a detrimental effect on performance that is statistically significant for all but the first and easiest to comprehend task.

Table 6. Differences between children that had their own computer and children that were required to share a computer (* $p<0.05$, ** $p<0.01$)

Learning Objectives	One child per computer station (83 students)		Shared computer stations (40 students)		ANOVA
	Mean	**SD**	**Mean**	**SD**	**F**
T1	1.13	0.34	1.18	0.39	0.38
T2	1.07	0.26	1.23	0.42	6.073**
T3	1.37	0.49	1.63	0.49	7.174**
T4	1.73	0.44	1.90	0.31	4.501*

Next, correlation analysis was done in order to identify possible connections among the learning objectives and provide with further hindsight as to the strengths and weaknesses of the syllabuses. Table 7 presents the analysis for the basic syllabus while Table 8 for the advanced one.

Table 7. Correlation analysis for the basic syllabus (* $p<0.05$, ** $p<0.01$)

	T1	**T2**	**T3**	**T4**
T1	1	0.646**	0.515**	0.159
T2	0.646**	1	0.445**	0.024
T3	0.515**	0.445**	1	0.275*
T4	0.159	0.024	0.275*	1

Table 8. Correlation analysis for the advanced syllabus (* p<0.05, ** p<0.01)

	T1	T2	T3	T4	T5	T6
T1	1	0.513**	0.08	0.05	0.12	0.27
T2	0.513**	1	0.20	0.06	0.09	0.03
T3	0.08	0.20	1	0.11	0.07	0.25
T4	0.05	0.06	0.11	1	0.42**	0.47**
T5	0.12	0.09	0.07	0.42**	1	0.38*
T6	0.27	0.03	0.25	0.47**	0.38*	1

In the basic syllabus T1, T2 and T3 are correlated with each other and these correlations are statistically significant. On the other hand T4 exhibits a weak correlation only with T3. These results further indicate that the first three learning objectives are well organized and sufficiently covered within the basic syllabus. They also suggest that T4 as a learning objective is rather well placed in the syllabus (after T3). Judging from the fact that T4 is not correlated with T2 a possible change in the syllabus to ameliorate results on T4 could involve shrinking the time devoted to T2 from 3 lectures (week 4 to 6) to 2 and increasing by 1 the lectures related to T4.

As far as the advanced syllabus is concerned (Table 8) results show that T1 and T2 exhibit the strongest correlation (similarly to the basic syllabus), but also T4, T5 and T6 are correlated or moderately correlated with each other. This can be viewed as a further testament that the advanced syllabus is well structured regarding the more complex topics it tackles. It also suggests (together with the high scores on T1 and T2 at Table 4) that T1 and T2 could be shrunk in length (1 week each in the advanced syllabus) and/or their teaching being merged with T3.

5.4 Summary of results

The main findings of the evaluation are summarized as follows:

- The first two learning objectives i.e., concurrent scripts on a single sprite (T1) and concurrent movement of multiple sprites (T2), were achievable by the *vast majority* of students both at the basic and at the advanced levels;
- Tackling simple racing conditions that occur during concurrent sprite movement using time based synchronization (T3) was achievable by roughly *half of the students* (both at basic and at advanced level);
- Message based synchronization (T4) proved to be a tough concept for beginners, while at the advanced level roughly *one third* of the students mastered it;
- Distinguishing between local (per sprite) variables and global ones (T5) and consequently using conditional variables for synchronization (T6) were mastered by roughly *4 out of 10* students that followed the advanced syllabus;
- As a general rule gender did not affect performance;
- Lab infrastructure played a significant role (it is favorable to have one working station per student);

- The following main correlations between learning objectives exist: (i) T1 and T2 have significant positive correlation in both syllabuses, (ii) T4, T5, and T6 have significant positive correlation in the advanced syllabus and (iii) T3 is correlated with all the remaining objectives in the basic syllabus, but is independent in the advanced.

It should be mentioned that the 6 learning objectives related to concurrency were incorporated to the syllabuses in addition to the classic topics taught such as: sequential programming structures and user interface concepts. Thus, the success ratios on the objectives should be viewed as extra gains. From this standpoint, both performance and correlation results indicate that both syllabuses are well structured overall, given the 12-hour timeframe that should be followed. Nevertheless, room for improvement exists and can be summarized as:

- Message based synchronization (T4) proved too complex to successfully convey it to beginners within one hourly lecture. Thus, unless the curriculum length is officially expanded, within the current 12 hours time limitation two options are available: (i) increase T4 lectures by one (possibly shrinking the T2 related lectures) or (ii) remove T4 objective from fifth grade and use the time slot for deepening the understanding of the first three objectives (particularly T3). Based on results from Table 4, it seems that T4 is better suited for more mature audience (sixth grade) making option (ii) more attractive;
- Based on the high success ratio on T1 and T2 at the advanced syllabus, a valid option would be to shrink their cover by one lecture devoting the extra time slot to T3 or T4.

6 Conclusions

The main purpose of this study was to investigate the introduction of concurrent programming concepts into a typical early computer programming syllabus using Scratch as a learning tool. Synchronization issues (race conditions) typically rise when building games with multiple interacting sprites, something that is a common approach to learning computer programming with Scratch. Instead of resorting to ad-hoc explanations when such errors inevitably occur that are difficult to understand by students only properly introduced to sequential program execution, this work advocates the systematic incorporation of concurrency issues in the followed syllabus. For this reason, learning tasks were built in a structured approach so that pupils incrementally build knowledge on concurrency issues, while also acquiring knowledge on classic structured programming topics and not missing the fun of game design. With the exception of only one objective at fifth grade, by the end of the 12-week course a large portion of the pupils achieved the 6 extra educational targets with ratios varying from 32.4% to 94.1% depending on the objective and students' ages. More importantly, pupils demonstrated for the largest part an ability to "think concurrently". This was also manifested by the fact that no "unexplained" program behavior was reported as such at the end demonstration, but was rather attributed correctly to racing conditions.

Summarizing, we can state that the results of the study illustrate the usefulness of introducing concurrent programming concepts in a structured way in primary school education. On the other hand, not all educational targets were successfully accomplished by all pupils, with timetable restrictions and infrastructure shortages playing role. Thus, suitable fine tuning of the presented syllabuses can bear further merits to the proposed approach.

7 References

[1] Ackermann, E.: Piaget's constructivism, Papert's constructionism: What's the difference. Future of Learning Group Publication 5(3), 438 (2001).

[2] Allen, J.P., Pianta, R.C., Gregory, A., Mikami, A.Y., Lun, J.: An interaction-based approach to enhancing secondary school instruction and student achievement. Science 333(6045), 1034-1037 AAAS (2011).

[3] Brennan, K., Resnick, M.: New frameworks for studying and assessing the development of computational thinking. In: Proceedings of the 2012 annual meeting of the American Educational Research Association, pp. 1–25, Vancouver, Canada (2012).

[4] Chinn, C.A., Brewer, W.F.: The Role of Anomalous Data in Knowledge Acquisition: A Theoretical Framework and Implications for Science Instruction. Review of Educational Research 63(1): 1-49 (1993). https://doi.org/10.3102/00346543063001001

[5] Clements, D. H., Sarama, J.: Research on Logo: A decade of progress. Computers in the Schools 14(1-2), 9-46 (1997). https://doi.org/10.1300/J025v14n01_02

[6] European Schoolnet, Computing our future Computer programming and coding - Priorities, school curricula and initiatives across Europe, http://www.eun.org/c/document_library/get_file?uuid=3596b121-941c-4296-a760-0f4e4795d6fa&groupId=43887, last accessed 2017/05/10.

[7] Fessakis, G., Gouli, E., Mavroudi, E.: Problem solving by 5-6 years old kindergarten children in a computer programming environment: A case study. Computers & Education 63, 87-97 (2013). https://doi.org/10.1016/j.compedu.2012.11.016

[8] Franklin, D., Hill, C., Dwyer, H.A., Hansen, A.K., Iveland, A., Harlow, D.B.: Initialization in Scratch: Seeking Knowledge Transfer. In: SIGCSE 2016, 217-222. ACM (2016).

[9] Jonassen, D., Strobel, J., Gottdenker, J.: Model Building for Conceptual Change. Interactive Learning Environments 13(1-2): 15-37 (2005). https://doi.org/10.1080/10494820500173292

[10] Lewis, C.M.: Is pair programming more effective than other forms of collaboration for young students?. Computer Science Education 21(2), 105-134 (2011). https://doi.org/10.1080/08993408.2011.579805

[11] Mayer, R.E.: Teaching and learning computer programming: Multiple research perspectives. Routledge (1988).

[12] Meerbaum-Salant, O., Armoni, M., Ben-Ari, M.: Learning Computer Science Concepts with Scratch. Computer Science Education 23(3), 239-264 (2013). https://doi.org/10.1080/08993408.2013.832022

[13] Papert, S.: Mindstorms: Children, computers, and powerful ideas. 2nd ed., New York, NY: Basic Books, (1993).

[14] Sáez-López, J.-M., Román-González, M., Vázquez-Cano, E.: Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. Computers & Education 97: 129-141 (2016). https://doi.org/10.1016/j.compedu.2016.03.003

[15] Hyslop-Margison, E. J., & Strobel, J. (2007). Constructivism and education: Misunderstandings and pedagogical implications. *The Teacher Educator*, *43*(1), 72-86. https://doi.org/10.1080/08878730701728945

[16] Tsihouridis, C., Vavougios, D., Ioannidis, G.: The Effect of Switching the Order of Experimental Teaching in the Study of Simple Gravity Pendulum-A Study with Junior High-School Learners. In: International Conference on Interactive Collaborative Learning, pp. 501-514, Springer, Cham (2016).

[17] Wilson, A., Moffatt, D.C.: Evaluating Scratch to Introduce Younger Schoolchildren to Programming. In: 22nd Annual Workshop of the Psychology of Programming Interest Group, pp. 64-74, (2010).

[18] Zygouris, N.C., Vlachos, F., Dadaliaris, A.N., Oikonomou, P., Stamoulis, G.I., Vavougios, D. et al.: The Implementation of a Web Application for Screening Children with Dyslexia. In: 19th International Conference on Interactive Collaborative Learning, pp. 415-423, Springer, Cham (2016).

8 Authors

Eleni Fatourou is a PhD candidate at the University of Thessaly, department of Computer Science and an ICT and Computer Science teacher at 1st and 3rd primary schools of Perama. Previously, Eleni Fatourou has worked as a Computer Science teacher at Secondary education, Vocational training and as distance learning trainer of trainers. She has also been a java developer. She graduated from Electronic and Computer Engineering, University of Crete and was awarded a Master's Degree in e-Learning by the Department of Digital Systems, University of Piraeus. Email: efatourou@sch.gr

Nikolaos C. Zygouris received his Ph.D in Clinical Neuropsychology from the Department of Special Education. He is adjunct Lecturer at Department of Computer Science of University of Thessaly, Lamia, Greece. His main research domain is in electrophysiological assessment, learning disabilities, web applications, clinical neuropsychology, anxiety, depression, cognitive psychology and educational psychology. He has authored more than 50 papers in journals, book chapters and major conferences. Email: nzygouris@uth.gr

Thanasis Loukopoulos received his Ph.D. degree in Computer Science from the Hong Kong University of Science and Technology. He is currently an Assistant Professor at the Department of Computer Science and Biomedical Informatics of the University of Thessaly, Greece. His main research domain is in parallel and distributed systems with interests including: green computing, cloud computing, WSNs, scheduling, load balancing, video coding parallelization and educational aspects of parallels systems. His work appeared in over 60 publications. Email: luke@dib.uth.gr

Georgios (George) Stamoulis was born in Lamia, Greece in 1966. He got his Diploma from the Department of Electrical and Computer Engineering at the National Technical University of Athens in 1989. He continued his studies at the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign where he was awarded the M.S. (1991) and Ph.D. (1994) degrees. After one year as a Visiting Assistant Professor at the University of Iowa, he joined Intel Corp. working on CAD tools for power analysis and optimization as a senior CAD

Engineer (1995-1996), as lead of the PowerCAD group (1996-1998), as Manager of the Santa Clara Annex of the Strategic CAD Laboratories (1998-1999) and as power manager of the Pentium M project, which became the Centrino platform (1999-2001). In 2001 he was elected Assistant Professor at the Department of Electronic and Computer Engineering at the Technical University of Crete. In 2003 he became an Associate Professor at the Department of Computer and Telecommunications Engineering at the University of Thessaly. In 2009 he became Professor. From 2003 to 2007 he was elected Associate Head and from 2007 to 2011 Head of the Department. Since 2013 he is the Head of the newly formed Computer Science Department. His research interests focus on the analysis and optimization of average and maximum power of integrated circuits, the analysis and optimization of the maximum voltage drop on the power supply lines of integrated circuits, low power design, reliability analysis and optimization, and the application of massively parallel and deep-learning techniques to the aforementioned problems. He has authored more than 100 papers in journals and major conferences, claims three US patents and has more than 800 references to his work. He has also founded two startups in the high-tech area. He is a member of the IEEE and the Technical Chamber of Greece and participates in the program and technical committees of several international conferences. Email: georges@uth.gr

This article is a revised version of a paper presented at the International Conference on Interactive Collaborative Learning (ICL2017), held September 2017, in Budapest, Hungary. Article submitted 20 December 2017. Resubmitted 10 January 2018. Final acceptance 18 February 2018. Final version published as submitted by the authors.