BlocklyPar: from sequential to parallel with block-based visual programming

Ana Luisa Veroneze Solórzano
Department of Languages and Computer Systems
Federal University of Santa Maria
Santa Maria, Brazil
0000-0003-0203-8865

Andrea Schwertner Charão
Department of Languages and Computer Systems
Federal University of Santa Maria
Santa Maria, Brazil
0000-0003-3695-8547

***Abstract*—This Innovative Practice Full Paper presents BlocklyPar, a set of three tutorial games to move from sequential to parallel programming using a block-based visual language. Block-based tutorial games are attractive tools for introducing programming to novices. A few of existing tools can express multiple tasks running at the same time, but none of them address parallel programming concepts and terms used in the field of parallel computing. Our tutorial games are targeted for first-year Computer Science students, as a resource to anticipate parallel computing using a self-taught approach with engaging challenges. The challenges involve university students' day-to-day tasks to make the games more meaningful for the audience, thus collaborating with the idea that everyday tasks can benefit from parallel approaches. The first game introduces the programming environment and the sequential blocks; the second introduces the concepts of tasks, resources allocation, and parallel task execution; and the third presents the concepts of computational load distribution and performance metrics for evaluating improvements in a parallel solution. The concepts are expressed through animation components and three new programming blocks. We have conducted preliminary tests with Computer Science students for evaluating the platform usage and parallel programming concepts assessed. The results suggest that the games contribute to the student's learning on parallelism as an extension of practicing sequential programming. It can also motivate students to design parallel solutions to explore today's multi-core and multiprocessor computers.**

***Index Terms*—computer science, parallel programming, tutorial games, block-based programming**

I. Introduction

Block-based tools are alternatives to introduce logic and programming concepts, as conditionals and loops, for novice programmers, inside classrooms and beyond [1]. Visual blocks are a high-level abstraction of textual codes with distinctive shapes and colors that perform actions when nested together using a drag-and-drop interface. Programming with blocks frees the user from the textual programming syntax and improves users' learnability by recognition over recall [2].

The popular block-based tools MIT App Inventor [3] and Scratch [4] provide self-oriented tutorials with step-by-step guidance via modals and videos. Code.org [5], also provides self-oriented tutorials, but inside games with distinct levels. A study showed that puzzle-based approaches could help users learn programming concepts better than using traditional tutorials, since users are free to think about combining the blocks to achieve a given outcome [6].

Tutorial games join engaging puzzles with self-oriented challenges based on constructionist precepts, where users can express their personal solutions using blocks at the same time they learn new skills [7]. Tutorial games can be appropriate for novices, and new programmers, since it uses similar terms and programming concepts used in textual programming (e.g., `if... else`, and `do... while` statements). In fact, some tools provide direct conversion from blocks to the equivalent code snippet in a textual language to approach users to real-world programming.

Parallel thinking is considered one of the fundamental concepts for teaching computational thinking in K-12 education [8], [9]. Parallel programming is a field in Computer Science that applies parallel thinking to better use multicore environments through libraries and APIs applied in textual programming languages [10]. Parallel programming classes are usually elective in CS courses or offered after the students acquire autonomy with sequential programming, for being considered "advanced".

There is still a discussion about teaching parallelism into introductory Computer Science while introducing sequential programming [11]. One of the first papers on the topic shows that parallel computing should be in introductory courses but might be less challenging if students have a basis of computing architecture and data structures [12]. A more recent one reports that it is essential for students to know how to apply parallelism to take advantage of today's multicore processors era since we have no perspective of coming back to only sequential instructions [13]. It is then counterproductive that programmers do not explore all computing resources due to a lack of training to do so.

Considering that, we asked ourselves: how to combine the benefits of using visual blocks to introduce programming in puzzle-based activities, with the challenge of encouraging CS students to think in parallel solutions to explore multicore environments? We present BlocklyPar, a set of tutorial games to practice from sequential computing to parallel concepts using a block-based language. It is a new tool targeted for first-year CS students as a resource to anticipate parallel computing teaching using a self-taught tool with engaging challenges. The

challenges involve university students' day-to-day tasks, thus collaborating with the idea that everyday tasks can benefit from parallel approaches. The games were designed to achieve the following learning objectives:

1) Practice sequential programming with blocks;
2) Introduce parallel programming concepts;
3) Practice the use of parallel approaches;
4) Rethinking a problem in a parallel way;
5) Analyze the performance improvement using parallelism.

The tool is open source and is available online in https://blocklypar.github.io. The rest of this paper is organized as follows. In Section II, we discuss related work on block-based tools exploring parallelism. Section III presents the design of the tutorial games, including the parallel programming concepts we address and the new programming blocks we propose. Section IV presents the three tutorial games, while Sections V and VI present preliminary user tests, results and discussion considering users' feedback. In Section VII, we present our conclusion and future works.

II. Related Work

Usual approaches for teaching parallel programming to novices include theory classes and practice activities. Nezu [14] proposes a 50 minutes practice to present parallel programming elements in intermediate-level programming courses. They use OpenMP, a popular application programming interface for developing parallel code [15]. The students used multi-system Linux environments to implement parallel sort and merge procedures in C with OpenMP and did performance analysis comparing with the serial program.

Other reports propose new textual programming languages to facilitate the teaching of parallel programming [16], [17]. EcoSim uses an English-like syntax to promote parallel thinking among programming learners, applied in a 5-day course for elementary school students with no prior programming knowledge, while Chapel is a language similar to Python and Java that facilitates the implementation of tasks in parallel.

Feng et al. [18] present activities to teach parallel programming to elementary school students and researchers of any educational level using Snap! (a block-based language inspired by Scratch) [19]. They created three new blocks to perform parallel operations in two activities: (i) a producer-consumer problem, a classic computing challenge that requires process synchronization, where the producers are bees, the product is honey, and the consumers are bears; (ii) an activity to develop the notion that parallelism can accelerate operations, presenting one jar and three cups, where the user can serve one cup at a time or the three in parallel. They evaluated their approach with high school students that made free constructions for the activities using sequential and parallel blocks.

Although these tools work well for their purpose, none of them present self-guided tutorials, as used in Code.org and Blockly Games. Introducing parallelism using game tutorials and block-based language can make the activity more attractive and challenging to novice programmers. Scratch, for example, lets users implement parallel tasks without presenting or correlating them to parallel programming concepts. Helena is a block-based programming language for web automation that offers a parallel skip block, used to define two online objects in the same entity and process them in parallel [20]. Parallel is a puzzle game to teach concurrency and parallelism concepts to CS undergraduate students without coding [21]. The player design synchronization mechanisms related to multiple threads execution without using blocks, and the game is not available online. We have not found a tool that proposes using tutorial games for introducing parallel programming using a block-based programming language.

III. Tutorial Games Design

We designed BlocklyPar as a set of tutorial games to address basic parallel programming concepts and terms used in the field. The design is inspired by the Maze game from Google's open-source Blockly-Games[1], a set of eight educational games that teach programming. Blockly-Games and other tools as MIT App Inventor and Code.org use blocks built with Blockly, an open-source programming library from Google for creating block-based programming languages, available in https://github.com/google/blockly.

A. Parallel Programming Concepts

Before implementing the tutorial games, we defined the parallel programming concepts to be introduced based on a Parallel Programming course curriculum. We start with the concept of task execution using one resource that processes a computational load by itself. Then we go to the parallel strategy, where the same amount of load is distributed among more than one resource. Figure 1 exemplifies this situation, where a sequential execution in one resource takes longer to execute than splitting the same task between three resources. The execution time orientation shows that there is a performance gain.

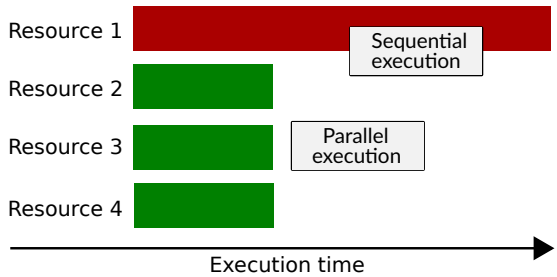


Fig. 1: Execution time gain with a parallel execution compared to the sequential execution.

With these concepts, users can practice a naive performance analysis, where they look at the tutorial challenge and compare the answers using multiple resources or one resource for the same situation. We believe it is a proper approach for non-expert parallel programmers to gradually learn concepts and

[1]https://blockly.games

visualize how parallelism can impact their code. The concepts addressed in the tool are:

1) Tasks: In parallel computing, tasks are instructions assigned to be performed in one or more computational resources, for example, in different cores from multicore processors. In one resource, the tasks run serially, one at a time; in more than one resource, tasks can be distributed between resources and run in parallel.

2) Computational Load: The amount of work each resource will compute, usually measured in time units. A good load balance between resources is crucial in parallel programming. The ideal scenario to accomplish the best performance is to divide the computational load equally among the resources.

3) Parallelism: Parallel programming is an approach to optimize algorithms' performance by splitting independent tasks into smaller tasks and computing them at the same time. Ian Foster proposed a methodology for design parallel programs by (i) partitioning the input data of the application into independent chunks, (ii) determining how tasks will communicate, (iii) grouping tasks that perform less work into larger tasks, and (iv) map the final tasks to run on different processor nodes [22]. Mattson et al. presented another approach where the programmer first analyzes the problem and identifies tasks that could run concurrently to then apply parallel optimizations [23]. These techniques show the importance of following a design pattern, analyzing the problem, the resources, the data, and the goal before inserting parallel approaches.

4) Performance Analysis: It usually considers the time spent to a program finishes executing and compares the result with executions using different systems, data size, or other parameters. The results show whether the approach achieved the maximum computing power or if the program could be implemented less costly. In parallel programming, this is part of a process to improve the performance of a program, where the developer runs experiments using different computing devices, workloads, or parameters, analyzes the results, modifies the implementation if needed, and re-run the experiments to a new evaluation.

B. Parallel Programming Blocks

Blockly is written in JavaScript, a programming language that facilitates the execution as a web application, running on the client-side without dependencies required. The developer can create new programming blocks, build the Blockly code locally, and inject it into a web page. There are functions to create new blocks of existing categories: textual blocks, loop blocks, variables (static and dynamic), procedural, mathematical, and logical. To create a new category, the developer must define the new blocks of this category and implement its behavior using textual code.

We convey the idea of parallelism with sprites moving at the same time on the animation screen. We created three new programming blocks for this, in a new parallel category: (a) sequential task block, (b) resource block, and (c) parallel task block. The `sequential task blocks` (Figure 2) run logic blocks stacked on them to complete a task. The `resource block` (Figure 3) determines which character will execute the task, bringing the notion of resource allocation. The drop-down menu options are according to the number of characters in the level. The character will only move according to the logic blocks stacked on it.

Fig. 2: New blocks to address task execution.

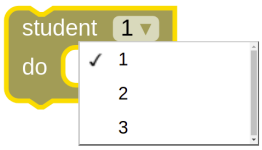


Fig. 3: New block to select the character to perform the task.

The `parallel task block`, the outer block in Figure 4b, expresses the parallel execution of tasks. It repeatedly executes all instructions, performing the task in parallel on the resources allocated. To delegate tasks to more than one character, the user must use one `resource block` to each character that will perform a task in parallel as shown in Figure 4b. Each resource can run different commands, such as in real-world parallel applications where resources can perform over different loads.

To avoid moving fast in introducing new blocks, we kept textual expressions, colors, and formats similar to the other blocks. The `sequential task block`, for example, remains the `parallel task block` by using the textual expression *repeat until*. Moreover, the compose of a parallel answer with the new programming blocks resembles the declaration of OpenMP sessions, as seen in Figure 4, where on the left there is the scope of sections with OpenMP written in C language, and on the right, the blocks used in the game. Each section defines a task to be performed by one resource, as in our games.

IV. Tutorial Games

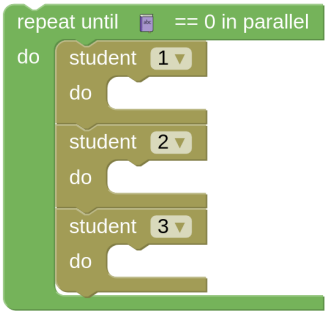
We created a set of three tutorial games with four levels each. Each tutorial has four independent levels with the same challenge to be solved but with varied visual components and blocks available. They have increasing difficulty, so we can gradually introduce the concepts and new blocks without overwhelming the user. The tutorial games flow consists of users visualize the tutorial challenge, check the resources available, think about a solution using the available blocks, and learn from previous levels to identify where parallelism could be applied for the ideal solution.

Our primer target audience is first-year Computer Science students, usually an audience with less knowledge of textual programming, that would benefit from the visual blocks and

```
#pragma omp parallel {

  #pragma omp sections {
    // definition of section 1
  }
  #pragma omp sections {
    // definition of section 2
  }
  #pragma omp sections {
    // definition of section 2
  }
}
```

(a) A parallel region defined with OpenMP sessions.



(b) New parallel task block.

Fig. 4: Comparison between one use of parallelism with OpenMP and the use of parallelism with the new blocks.

focus only on practicing parallel programming. To make the games engaging to the target audience, we created challenges involving day-to-day tasks for university students, such as attending a class after finishing the homework and returning books to the library. They have distinct challenges, scenarios, and programming blocks to command actions on one or more game characters representing the students. The following sections present each tutorial game:

A. First tutorial game

Since some novice programmers are not used to block-based tools and their drag-and-drop dynamics, we created the first tutorial to present the environment, introduce the blocks programming language and the execution workflow. The challenge is to program the character to walk home (Figure 5a). We introduced sequential programming structures of repetition and if-else statements, useful for the next tutorials.

B. Second tutorial game

This game introduces the concepts of task execution, computational load, and resource allocation. The challenge is to guide the character to the classroom after completing all the homework (Figure 5b). Homework is considered completed when the character passes through it, then the computer image is removed from the path and added to the list on the left. The game ends when all homework is done, and the character arrives in the class.

From level one to three, we work with the `sequential task block` varying the path and the blocks available so users can get used to define the task region using the new block. In the last level, we introduce the `resource block` and explain in a pop-up message that from now on, the character will only move when explicitly declared with the new block. Figure 6 shows an example of the `sequential task block` an the `resource block` usage. To associate the concepts from Section III with the components used in this game, we defined what the task, resources, and load are:

- **Task:** Finish all assignments before going to class.
- **Resources:** One character.
- **Load:** Homework to be completed before attending class.

C. Third tutorial game

This game introduces parallelism, load balance, and performance analysis. The challenge is to guide the characters to the library to return all the books borrowed represented in the list on the left of the animation screen (Figure 5c). The character delivers a book upon arrival at the library. The book is decremented from the list on the left, and the character returns to the starting point, repeating the actions until there are no books left.

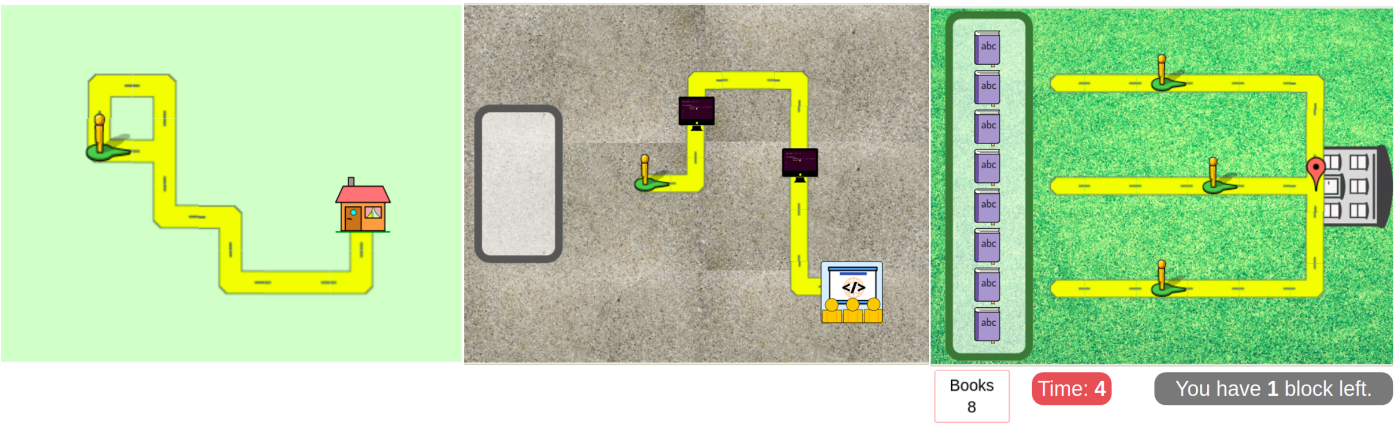
Level one follows the last level of the previous tutorial, requiring the `sequential task block` and the `resource block` to move one character. In the second level, we introduce the `parallel task block` with two characters keeping the number of books and a similar path from the last level. To insist on using one `resource block` per character, level three continues with two characters but with more books. We introduce a new character in the last level, so the user can play using one or two or three and see the performance gains. Figure 7 exemplifies the use of the `parallel task block` with three characters. The concepts in this game are:

- **Task:** Return all the books to the library.
- **Resources:** Two or three characters, depending on the level.
- **Load:** Books to be returned.

The tool is self-guided by using pop-up messages at the beginning of the level, explaining the challenge, and throughout the interactions to present new components and give tips based on user behavior. We also implemented animation components to guide the user on realizing the performance improvement using more than one resource on the third game, as shown in Figure 8. A counter below the animation screen depicts the number of books left, and another box shows the execution time spent at each level. To calculate the execution time, we assigned a runtime to each block. A `move forward` block, for example, counts 1 unit of time, while the block to check a path attached to a conditional block counts 0.5 units of time. It shows a consistent execution time, even if characters take different routes.

V. Preliminary Tests and Results

We applied individual and anonymous questionnaires for two groups of students in CS who volunteered to participate: (i) two male graduate students with in-depth knowledge of parallel programming, and (ii) eleven undergraduate students (six female and five male students), with basic knowledge or without knowledge of parallel programming. Given how long they have been in CS, we estimate their ages to be between 18 and 23 years old. As a preliminary study, we believe that having graduate and undergraduate respondents could bring rich insights about the games, so applying the games to only first-year CS students learning textual programming has not yet been studied.



(a) First tutorial game (b) Second tutorial game (c) Third tutorial game

Fig. 5: The three tutorial games (we omitted the programming screen and other visual components for better interpretation). First game (a) introduces the platform and block-based programming, second game (b) task execution, computational load, and resources allocation, and third game (c) parallelism, load balance, and performance analysis.

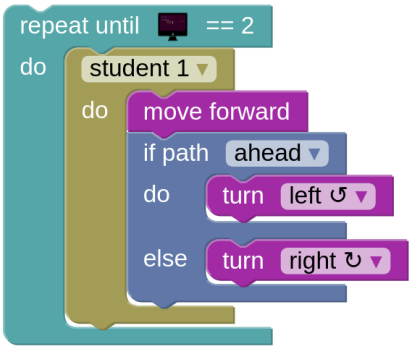


Fig. 6: Using the task block for the second tutorial game.

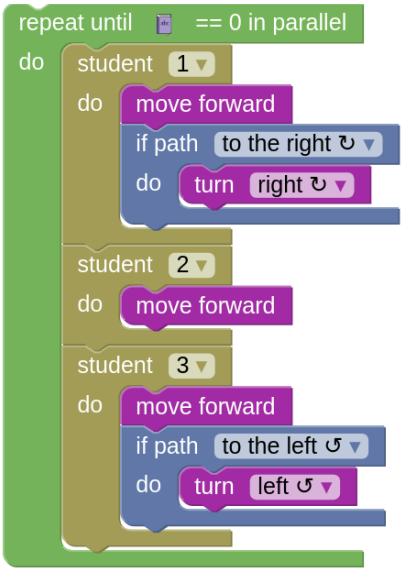


Fig. 7: Using the parallel task block with three characters.

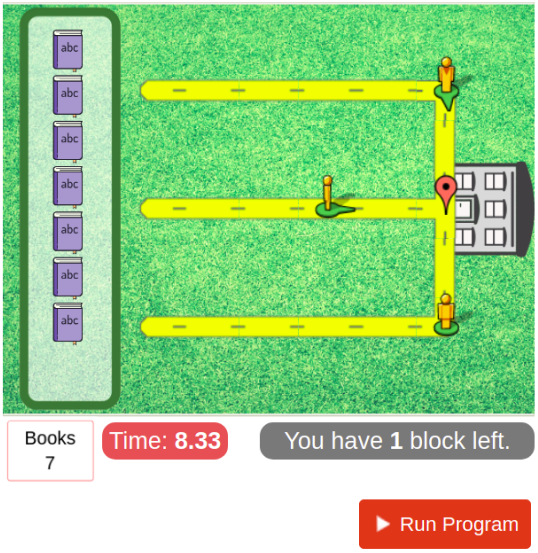


Fig. 8: Animation screen and the resources used to motivate performance analysis.

All tests were conducted in person by the authors of this paper, within a scheduled time frame, during which we observe the students' behavior while they played the games. All users had access to a computer and received the link to BlocklyPar. Users entered the tutorial games web page and were instructed to click the "About" page, which presents an explanation about each challenge and a short overview on the parallel programming concepts involved. After playing the three games in sequence, they answered a questionnaire. We also inserted Google Analytics[2] in the web pages to collect information about user access. Our goals were to monitor the students'

[2]https://analytics.google.com

navigation in the tutorial games, to check the challenges' acceptance by the intended audience, and to evaluate the clarity of the scenarios' components and challenges. Section V-A presents the students' behavior we noticed by observation and by analyzing Google Analytics data, and section V-B presents the results of the questionnaires.

A. Students behavior

Students who had already played Blockly-Games solved the games quickly than those who were unfamiliar with it. Those who have not played Blockly-Games but had experience in playing games understood the drag-and-drop workflow quickly. One undergraduate student with no experience playing games presented difficulty guiding the character to the endpoint using blocks and stacking the blocks to create the algorithm.

The undergraduate students took a similar time to play. All the female students spontaneously wrote down suggestions and improvements to insert in the questionnaire while playing. Three undergraduates presented difficulty with the character orientation in the scenario, taking more time to move the character correctly and solve the first game. Some undergraduates had difficulty understanding the game's goals presented in the pop-up dialogues, as the tool is in English and is not implemented in their native language yet.

Using Google Analytics, we observed that 8 out of 13 students accessed the "About" page as instructed. The average time navigating on this page was two and a half minutes (2'30"). The access had no bounce rate, meaning that the users did not open the page and immediately left. We observed that the last level of the second game took longer to be solved due to the `resource block`. With Google Analytics, we confirmed that the average time on this level was 2'47" and in the previous level was 2'2".

The first level of the second game was solved fast, around 20". However, the first level of the third game took 2'21". We supposed that this happened due to the new scenario, goal, and especially due to the mandatory use of the `resource block`. The last level of the third game took the higher retention of 4'36". We observed that students left the pages after completing the challenge, so the time was spent only implementing the solution.

B. Questionnaire results

Both groups answered multiple-choice and open-ended questions after playing the three tutorial games. The questionnaire aimed to evaluate the tutorials' format: whether the number of levels was enough, the scenarios were attractive, and the challenges were clear. It also included questions to assess the effectiveness of the tutorial games to introduce parallel programming.

The graduate students had already played Maze. They liked the look and functionality of the new blocks, finding the format and images suggestive of what the blocks do. One respondent suggested an explicit presentation of how much faster is the parallel version compared to the sequential execution, and pointed that the tool is more appropriated for users with previous experience with the usual visual programming blocks.

We asked the same open question for both groups: *"If you have already studied Parallel Programming, which concepts did you recognize in the games?"*. Two undergraduates answered: *"Division of tasks, allocation and sharing of resources, simultaneous/concurrent access to data"*, and *"Only that the student would be a thread"*. The graduate students associated the idea of performance improvement when reducing the execution time with parallelism, parallelism in thread-level, and that reducing the execution time is not always linearly proportional to the number of resources used. They also associated load distribution concepts: the competition for shared resources, dynamic load division, and load balancing. One student wrote: *"the reduction in the total execution time is not always linearly proportional to the number of workers"*.

This last statement relates to the third tutorial, where the execution time does not necessarily decrease linearly using more characters. Because of that, the fourth multiple-choice question (Figure 9) asked which of the situations could take half of the execution time to deliver all the books, compared to the case presented in the header. The correct answer is Option 3, since the distance of characters at the limits of the path to the library is the same, so they may have the same cost, while other combinations of two characters have different distances to the library. Three students selected the wrong answer: two of them selected Option 2, that has the right number of characters, but located at different starting points. It indicates that they not compare the cost of the paths that each character walk to understand how much faster it can be.

We asked **four questions** about the new blocks, including the one presented previously. No student answered all questions correctly, but four female students answered three right questions. In the **first question**, we asked which blocks represent task execution. We expected they select the `task blocks` (sequential and parallel), but three selected only the `resource block` and two selected all blocks.

In the **second question**, we presented a screenshot of the third game with one character and asked which visual component represents the load of the problem. The load is the books, which the characters must take to the library. The other option was the users, which would be the resources allocated for the execution of the task; the steps to walk the way, which would be the execution results; and the programming blocks, which are the instructions that determine the movements. Seven students selected the right answer, and the others selected "the steps to walk the way" and "the programming blocks".

The **third question** asked which parallel programming issue is associated with the `resource block`. The correct option is that it associates a task with a specific processor. In the games, it associates the task with the selected character. Nine students answer it right. Other students thought that it defines an instruction related to the character, but the instructions are the blocks stacked on the `resource block`. We believe that this happened because they associated the challenge of the first game with the term "task", being a homework one

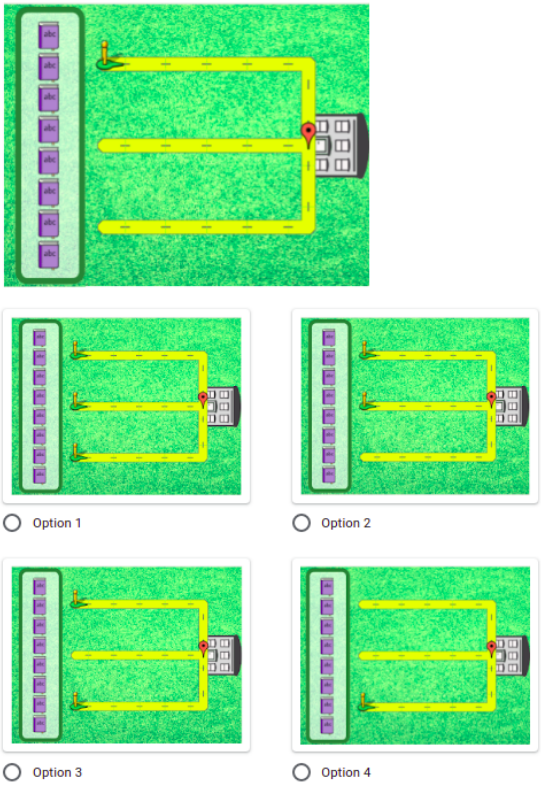


Fig. 9: Fourth multiple-choice question in the questionnaire for undergraduate students.

kind of task. As discussed before, the **fourth question** (Figure 9) asked about execution time and had seven correct answers.

Table I summarizes the questionnaire results for the undergraduate students. Only two users marked all correct options for the first question. For the others, at least half of the students selected the right answer. We notice that the third question, which correlates the Parallel Programming concept with design aspects of the game, had a higher match.

TABLE I: Questionnaire results for the 11 undergraduate students.

Question #	1	2	3	4
Number of correct responses	2	7	9	7

We asked the undergraduates which audiences they think are most appropriate for the games. They could select all that apply: high school students, first-year CS students with or without knowledge of parallel programming, CS students with knowledge of parallel programming, or CS students without knowledge of parallel programming. The majority (ten students) answered first-year CS students with or without knowledge of parallel programming, followed by high-school students (eight answers). One wrote that it could be used with younger students only if they knew programming concepts. Otherwise, it would be necessary to apply introductory programming activities.

The last question of the questionnaires asked for general feedback and suggestions in an open-ended format. We present two insightful feedbacks: (i) "I found the game interesting and that it instigates reasoning, mainly because it proposes exercises with parallel programming concepts (not so explored in similar games that already exist).", (ii) "The game brings a diverse experience, helping to instigate knowledge even without theoretical knowledge. I really liked the idea and the initiative. It is super interesting, especially for beginners in the area!".

VI. Discussion

Our tests indicate that the tutorial games can be suitable for the target audience of first-year CS students. Most of the students understood the idea of performance improvement when distributing the task among more than one resource and the meaning of the visual components. However, based on the questionnaire answers, we concluded that the differences among the new blocks are not sufficiently clear, neither is the parallel programming concept addressed by the `task blocks`.

We need to improve both tutorial games levels and visual components to present the parallel concepts more clearly. One idea is to highlight the counter in the third tutorial, showing the history of execution time for similar challenges using a metric such as the *speedup* to express the performance improvement. We can use more dialogues and pop-ups to point what concept is addressed with each visual component and a complete explanation of the parallel programming concept, perhaps in a dialog window. On the other side, most users understood the meaning of the visual components and the challenge. We plan to insert new levels for each tutorial and two new tutorials: one for users without previous experience with block-based programming and another with more practices using the `parallel task block`.

VII. Conclusion

Parallel applications are essential to efficiently solve computational problems in several fields of research. Since parallel approaches are not mandatory for the correct execution of a program, they are often overlooked. Most programmers are not used to applying parallelism in their programs since the beginning, having to rewrite the code and sometimes rethink the algorithm to implement a parallel solution.

Tutorial games with visual programming blocks are advantageous resources to engage the users in educational activities. However, the existing tools do not offer parallel programming blocks either address parallel programming concepts. We believe that block-based tutorial games can facilitate the introduction of parallel programming for novices.

In this paper, we presented BlocklyPar, a set of tutorial games for introducing parallel concepts. We also presented a preliminary study evaluating the format and challenges of the tutorial with Computer Science students from different periods. The results suggest that our tutorial games can contribute to the students learning of parallelism as an extension of

practicing sequential programming. They also provide insights to improve the platform, create new tutorials, and diversify the levels. A complete evaluation of a wide public of first-year CS students is yet to be done. We plan to apply it in a Parallel Programming course.

The source code of BlocklyPar is available online on GitHub with Apache 2.0 open source licensing in https://github.com/blocklypar. The tutorial games are available online to play in https://blocklypar.github.io. We encourage users and developers to play and modify the code, serving as a base to create similar tutorial educational games.

REFERENCES

[1] D. Weintrop and U. Wilensky, "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*, ser. IDC '15. New York, NY, USA: ACM, 2015, pp. 199–208. [Online]. Available: http://doi.acm.org/10.1145/2771839.2771860

[2] D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak, "Learnable programming: Blocks and beyond," *Commun. ACM*, vol. 60, no. 6, p. 72–80, May 2017. [Online]. Available: https://doi.org/10.1145/3015455

[3] S. C. Pokress and J. J. D. Veiga, "Mit app inventor: Enabling personal mobile computing," *arXiv preprint arXiv:1310.2830*, 2013.

[4] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, p. 60–67, Nov. 2009. [Online]. Available: https://doi.org/10.1145/1592761.1592779

[5] Code.org, "Code.org." [Online]. Available: https://code.org/

[6] K. J. Harms, N. Rowlett, and C. Kelleher, "Enabling independent learning of programming concepts through programming completion puzzles," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2015, pp. 271–279. [Online]. Available: https://doi.org/10.1109/VLHCC.2015.7357226

[7] S. Papavlasopoulou, M. N. Giannakos, and L. Jaccheri, "Exploring children's learning experience in constructionism-based coding activities through design-based research," *Computers in Human Behavior*, vol. 99, pp. 415–427, 2019. [Online]. Available: https://doi.org/10.1016/j.chb.2019.01.008

[8] J. M. Wing, "Research notebook: Computational thinking: What and why?" *The Carnegie Mellon University School Of Computer Science*, 2010.

[9] V. Chiprianov and L. Gallon, "Introducing computational thinking to k-5 in a french context," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '16. New York, NY, USA: ACM, 2016, p. 112–117. [Online]. Available: https://doi.org/10.1145/2899415.2899439

[10] A. Shafi, A. Akhtar, A. Javed, and B. Carpenter, "Teaching parallel programming using java," in *Proceedings of the Workshop on Education for High-Performance Computing*, ser. EduHPC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 56–63. [Online]. Available: http://dx.doi.org/10.1109/EduHPC.2014.7

[11] K. Kirkpatrick, "Parallel computational thinking," *Commun. ACM*, vol. 60, no. 12, p. 17–19, Nov. 2017. [Online]. Available: https://doi.org/10.1145/3148760

[12] D. J. John, "Integration of parallel computation into introductory computer science," *SIGCSE Bull.*, vol. 24, no. 1, p. 281–285, Mar. 1992. [Online]. Available: https://doi.org/10.1145/135250.134567

[13] W. B. Gardner, "Should we be teaching parallel programming?" in *Proceedings of the 22nd Western Canadian Conference on Computing Education*, ser. WCCCE '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3085585.3085588

[14] N. Nezu, "Teaching parallel programming in 50 minutes," *J. Comput. Sci. Coll.*, vol. 31, no. 2, pp. 18–24, Dec. 2015. [Online]. Available: https://dl.acm.org/doi/10.5555/2831432.2831435

[15] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming.* USA: The MIT Press, 2007.

[16] K. Burke, "Chapel: A versatile language for teaching parallel programming: Conference workshop," *J. Comput. Sci. Coll.*, vol. 30, no. 6, pp. 16–16, Jun. 2015. [Online]. Available: https://dl.acm.org/doi/10.5555/2753024.2753030

[17] C. Gregg, L. Tychonievich, J. Cohoon, and K. Hazelwood, "Ecosim: A language and experience teaching parallel programming in elementary school," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '12. New York, NY, USA: ACM, 2012, pp. 51–56. [Online]. Available: http://doi.acm.org/10.1145/2157136.2157155

[18] A. Feng, E. Tilevich, and W.-c. Feng, "Block-based programming abstractions for explicit parallel computing," in *Proceedings of the 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, ser. BLOCKS AND BEYOND '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 71–75. [Online]. Available: http://dx.doi.org/10.1109/BLOCKS.2015.7369006

[19] A. Feng and W. Feng, "Parallel programming with pictures in a snap!" in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 950–957. [Online]. Available: https://doi.org/10.1109/IPDPSW.2016.194

[20] Helena, "Helena - web automation for end users." [Online]. Available: https://helena-lang.org/

[21] J. Zhu, K. Alderfer, A. Furqan, J. Nebolsky, B. Char, B. Smith, J. Villareale, and S. Ontañón, "Programming in game space: How to represent parallel programming concepts in an educational game," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, ser. FDG '19. New York, NY, USA: ACM, 2019. [Online]. Available: https://doi.org/10.1145/3337722.3337749

[22] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[23] T. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*, 1st ed. Addison-Wesley Professional, 2004.