Exploring Automata Theory with an Educational Activity Using Graph Grammar for K-12 Education

Júlia Veiga da Silva[1], **Braz Araujo da Silva Junior**[1], **Simoné André da Costa Cavalheiro**[1], **Luciana Foss**[1]

[1]Technology Development Center – Federal University of Pelotas (UFPel)
CEP 96.010-610 – Pelotas – RS – Brazil

{jvsilva,badsjunior,simone.costa,lfoss}@inf.ufpel.edu.br

Abstract. *This paper proposes an educational activity for K-12 Education, aligned with the Brazilian National Common Curricular Base, which explores Automata Theory using Graph Grammar. Although several areas of computing are increasingly integrated into the educational context, the theoretical area is still neglected. Due to the scarcity of direct approaches in K-12 Education, this project seeks to bridge this gap. The proposed activity not only develops a specific National Common Curricular Base skill but also indirectly enhances Computational Thinking skills.*

1. Introduction

Driven by the impact of Computer Science (CS) on daily life, initiatives, and efforts have emerged to make education in this area available to everyone. Influenced by Wing's important work on Computational Thinking (CT) [Wing 2006], the perception of CS education has come to understand computing not only as coding or programming but as the ability to solve problems. Today's students live in a world heavily influenced by computing, and many will work in fields that involve or are influenced by it. Therefore, waiting until students are in college to introduce computing-related concepts is no longer sufficient. They should begin working with algorithmic problem-solving and computational methods and tools in K-12 [Barr and Stephenson 2011]. Recent works show the application of different practices allied to CT in K-12, such as unplugged computing [Chen et al. 2023], robotics [Cayetano-Jiménez et al. 2024, Ching and Hsu 2023], and even artificial intelligence [Lee and Kwon 2024, Yim and Su 2024, Sanusi et al. 2023].

In 2022, the Brazilian National Education Council approved the Computing Standards in K-12 education, complementing the National Common Curricular Base (BNCC)[1] [Brazil 2022]. CT is one of the three main axes outlined in this appendix. Despite the recent growth of CT in Brazil [Farias et al. 2023], challenges such as professional development, availability of teaching materials, and the application of effective methodologies must be addressed. Additionally, there is a need to create activities that promote the development of the objects of knowledge outlined by the BNCC for each level of K-12. Clarifying, objects of knowledge are the subjects covered throughout each curricular component, representing the means for developing skills.

[1]Document that defines the essential knowledge, skills, and competencies that all K-12 students in Brazil must develop throughout their schooling.

DOI: 10.5753/sbie.2024.242502

Although computing education has been reignited with the advent of CT and has expanded into K-12, Theoretical Computer Science (TCS) receives relatively less attention in this context. A systematic literature review on TCS in K-12 revealed that topics such as regular expressions, formal languages, and automata theory are much less popular [Silva Junior et al. 2021]. The works covered several approaches, including digital tools, traditional classes, and practical activities unrelated to computers, emphasizing problem-solving as a central element of their educational proposals. However, TCS is underrepresented compared to more popular approaches.

Aiming to make resources available in this overlooked area, this paper presents an educational activity based on automata theory, a sub-topic within TCS, to develop a skill outlined in the BNCC computing appendix for K-12. The activity was developed using GrameStation, a game engine based on Graph Grammar (GG), a formal and visual language for describing systems and verifying properties. The rest of this paper is organized as follows: Section 2 presents some related work and differentiates our proposal from existing ones; Section 3 delves into the theoretical background, exploring concepts concerning automata theory and GG, besides presenting the game engine GrameStation; Section 4 introduces our approach, detailing the activity into the GrameStation; Section 5 presents a discussion about the proposal; and Section 6 concludes the paper, presenting final remarks and outlining potential paths for future research.

2. Related Work

Although not as popular as visual programming languages, some works propose introducing automata in education. For example, an automata puzzle game was proposed to introduce automata theory using gamification to primary and lower secondary school students (9-12 years old) from 36 elementary schools in Japan [Isayama et al. 2016]. In their research, the authors focused only on the concept of Deterministic Finite Automaton (DFA). The game has several stages, divided into *labeling questions*, where students must define the transitions of a given DFA to recognize the provided inputs, and *recognition questions*, where students must identify whether or not a given DFA recognizes a given input.

A total of 90 children played the game; fifteen were 4th graders, forty-one were 5th graders, and thirty-four were 6th graders. Their actions were recorded in logs, that consisted of information regarding when and which buttons were selected, and the values set for those buttons at those times. An analysis of the logs shows that by combining the labeling and recognition data, approximately 60% of children had correct answer rates of 70% or more of the questions. Thus, many of them understood automata concepts well enough to complete the game stages. However, the analysis also shows that while some children in the study (not many) could understand the concepts presented, most of them did not fully understand it, as 80% or more were unable to discover the characteristics of the automata in the recognition questions.

In the Brazilian context, activities that work on automata theory in K-12 were not found, so we will present some examples of activities that work on the topic with undergraduate students. Some works address the scope of this work [Carvalho et al. 2021, Tomizawa and Junior 2021, Vieira and Sarinho 2019, Silva et al. 2010, Leite et al. 2014], and below we detail those that address in more depth the concepts considered in this work.

"Máquina de Senhas", Portuguese for "Password Machine" [Vieira and Sarinho 2019], is a game based on the readaptation of gameplay aspects from the Mastermind game. It follows this format: an example of an automaton is presented to the player, and a sequence of symbols accepted by the automaton is generated. This sequence is invisible to the player. The player must then uncover the hidden sequence from the given automaton. After each move, the player receives information about their current attempt, including whether symbols are in common between the played sequence and the hidden sequence, whether any symbols are in the correct position, and whether the automaton accepts the suggested sequence. The game presents an increasing level of difficulty in the stages regarding the complexity of the challenges, increasing the number of characters in the sequence to be discovered, and the number of symbols in the automaton's alphabet.

Another game, named "Automata Defense 2.0" [Silva et al. 2010], is an educational game designed as a pedagogical complement to the Formal Languages and Automata course. Version 1.0 of the game focuses only on the concept of DFA, while version 2.0 expands the content to also include topics on Nondeterministic Finite Automata (NFA) and Deterministic Pushdown Automata (PDA). The game consists of a tower defense strategy game, challenging players to design automata to succeed. The game features diverse types of creatures, and players must eliminate monsters to avoid losing the score. To do so, they can create towers with automata to attack creatures that have words recognized by the automata. The game also requires strategic reasoning from players, as each added state or transition incurs a cost to be paid with virtual money. The research involved a usability test and a preliminary evaluation of its pedagogical effectiveness. A total of 26 students completed all stages of the research. The pre- and post-session tests covered the topics of DFA, NFA, and general theoretical questions. The game was considered useful as a pedagogical complement but areas for improvement in the interface were identified, such as providing rules within the game and greater differentiation between characters in the activity.

Finally, the game "Chomsky's Mountain" [Leite et al. 2014] is based on Chomsky's Hierarchy, which classifies formal languages into four levels. The game consists of different environments or stages, each representing a level of the hierarchy. The player faces problems related to the languages of each level and must build formal models, such as automata, regular expressions, or grammars, to represent or recognize these languages. As the player solves the problems, new stages are unlocked, becoming progressively more challenging. The goal is to reach the top of the mountain, which represents solving all the problems at all levels of the hierarchy. The study involved administering an electronic questionnaire to students after interacting with the game. The test included 39 volunteer students, of whom 19 were enrolled in the Theory of Computation course and 10 had already completed the course. The results showed that most students positively evaluated their experience with the game (41% rated it a 10, 26% rated it a 9, and 18% rated it an 8), considering it helpful for learning the subject. They also reported that the game helped them better understand concepts and solve problems related to the Theory of Computation. Regarding difficulties, 38% of students faced some kind of difficulty during the game, mainly related to first-time access and interpreting some questions. However, most students said they would prefer to solve Theory of Computation problems through the game, highlighting the importance of the immediate feedback provided by the game.

Compared to related work, the activity presented in this paper stands out for its integration of automata with a formal language, requiring students to manipulate a GG explicitly. This activity not only develops a BNCC skill but also indirectly contributes to the development of CT skills. The definition and execution of an automaton through a GG fosters various CT skills [Silva Junior et al. 2019], including abstraction, where students must understand an automaton that synthesizes a robot's behavior and manipulate a GG that does not define an algorithm as a sequence of steps, but rather as a set of actions (without a predefined order) that can be performed based on the current context of the system (the automaton's state and the current cell of the tape); data representation, where students must interpret the transition function, tape, and robot through vertices and edges (graphs); data collection, where students must define the sequence of information (tape creation) necessary for the robot to fulfill its task; analysis, where students must evaluate the input tape and identify the robot's final state; simulation, where students must execute grammar rules that emulate the robot's behavior in different situations; and pattern recognition, where students must define the match for applying GG rules by searching for the left-hand side of the rule to be applied in the graph representing the system's state.

3. Theoretical Background

This section presents the theoretical context in which our study is situated. It discusses the definitions of automata theory (emphasizing DFA) and GG, besides presenting the tool used to create the activity.

3.1. Automata Theory

An automaton is an abstract model of a system that can follow a specific set of instructions to perform a particular task. It can be thought of as a machine with a finite set of states that can read input symbols, transition between states based on the symbols read, and, in some cases, produce an output. Automata are categorized into different types, such as finite automata or pushdown automata. However, this work focuses on DFA, a subcategory of finite automata. Formally, a DFA is defined as follows [Mogensen 2024].

A DFA is a tuple $M = (Q, \Sigma, \delta, q0, F)$, where: Q is a finite set of states; Σ is a finite set of input symbols, called the input alphabet; $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, which maps a state and an input symbol to a new state, but not necessarily defined for every possible combination of state and input symbol; $q0 \in Q$ is the initial state; $F \subseteq Q$ is the set of accepting (or final) states.

Thus, the automaton reads symbols from an input alphabet and transitions between states based on these symbols and its transition function, δ. The transition function, δ, maps a current state and an input symbol to a new state. This transition process occurs sequentially for each symbol in the input, starting with the initial state, $q0$. After reading the entire input string, if the DFA is accepting state (belonging to set F), then the input string is accepted by the DFA, indicating that the input is part of the language recognized by the DFA. Otherwise, if the DFA is non-accepting state or cannot finish reading the input string, the input string is not recognized by the DFA.

3.2. Graph Grammar

A GG describes a system by modeling its states as graphs (composed of vertices and edges), and events that can change its current state as a set of graph transformation rules

[Ehrig et al. 1997]. A GG must define how its state graph begins, which is called the **initial graph**. Additionally, a GG can differentiate and restrict its elements by declaring them in a **type graph**. For example, Figure 1 illustrates the type (left) and initial (right) graphs of the Pac-Man game as a GG. The type graph defines the elements that compose the game, while the initial graph shows a Pac-Man, a ghost, and fruits on a 3x4 grid of places, and a counter (pink triangle) related to Pac-Man to count how many fruits have been eaten. In particular, the Pac-Man game has four rules (Figure 2): *PacMove*, *GhostMove*, *PacEat*, and *GhostKill*, each represented by a pair of graphs linked by a graph homomorphism.

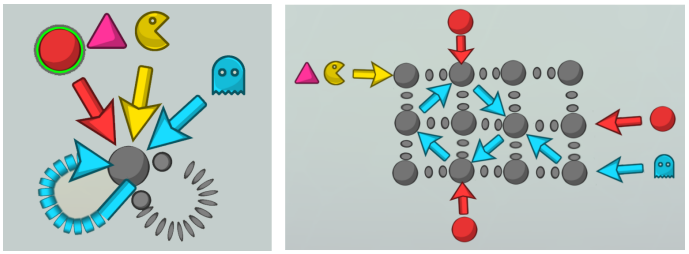

Figure 1. Type graph (left) and initial graph (right) of the Pac-Man game as Graph Grammar

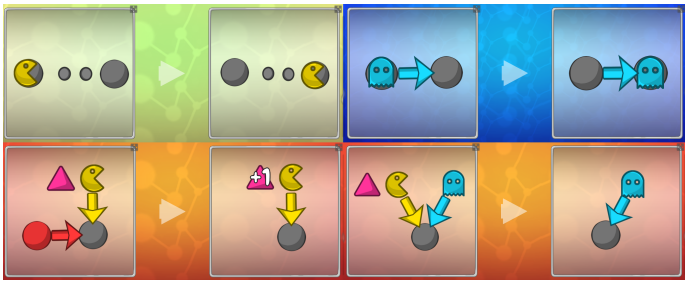

Figure 2. *PacMove* (top, left), *GhostMove* (top, right), *PacEat* (bottom, left), and *GhostKill* (bottom, right)

The pair of graphs representing the rules are the Left-Hand Side (LHS), which expresses a condition to apply the rule; and the Right-Hand Side (RHS) which expresses the consequence of applying the rule. In *PacMove* (Figure 2), for example, LHS defines the condition of having a Pac-Man in a place that has a way to another, and RHS defines the consequence of removing the Pac-Man from the initial place and placing it in another. This representation also implies mappings element by element between graphs (morphisms), so for each element in one graph, it must be said which element (if any) in the other graph corresponds to it. If an element is successfully mapped, it means the rule preserves it, such as the Pac-Man from *PacMove* (Pac-Man is preserved and its mapping to the place is deleted and created). If an element is left unmapped and is in the LHS, then the rule deletes it, such as the Pac-Man from *GhostKill*. If it is not mapped and is in the RHS the rule creates it. Finally, to apply a rule (that is, change the current system state) it is necessary to find a match by mapping LHS elements in their corresponding elements in the state graph.

3.3. GrameStation

GrameStation [Junior et al. 2021] is a GG-based tool used for creating and running games modeled according to this formal language. Since games are represented as GG, it also promotes the development of skills related to CT. These skills are developed both by the person who creates a game (specifies a GG) and by the person who runs a game (simulates a GG) [da Silva et al. 2021]. GrameStation is organized into three modules: Grame Explorer, Grame Builder, and Grame Player (Figure 3). These modules allow users to find, create, and run games, respectively.

In the tool the type graph corresponds to a declaration area, the initial graph refers to the game organization when it starts, and the rules represent the possible actions to be carried out by the player. Finally, to play a game (using the Grame Player module), the user can select, map, and apply the specified rules during the execution. When a rule is selected, the LHS and RHS graphs are shown, and the user should find a match by clicking on LHS elements and then their corresponding elements in the state graph – GrameStation signals when a match is correct or incorrect.

Figure 3. Explorer (top, left), Builder (top, right), and Player (bottom) modules of GrameStation

4. Our Approach

In this section, we present the proposed activity. In subsection 4.1 we describe the methodological steps involved in the conception of the proposal, and in subsection 4.2 we detail the proposed tasks.

4.1. Methodology

The study of finite automata involves several concepts such as states, transitions, and input tape. Since the target audience for this proposal is primary school students, a gradual introduction of these concepts is more appropriate. According to Resnick (2017), it is

important for an activity to be initially designed with simple objectives and tools (low floor) while also allowing for expansion to include more complex concepts (high ceiling). In this context, the methodological steps followed in this proposal were:

1. **Choice of theme**: we selected a theme that would engage the target audience. Our goal is to develop a skill that is part of the BNCC CT axis and is intended for 9th-grade elementary school students. Therefore, the educational activity "The Intergalactic Mission" is a space-themed game designed to teach automata theory to this audience (K-12 students). The game is set in a future where humanity has colonized various planets. The player takes on the role of the Explorer Robot, tasked with helping to colonize a newly discovered planet.
2. **Definition of concepts covered in the activity**: identifying the key concepts to be introduced. The activity is based on the BNCC skill **EF09C003**, defined as the ability to use automata to describe behaviors abstractly, automating them through an event-based programming language. In this context, the activity will be designed in stages, each addressing different automaton models. The first stage will cover concepts related to DFA, including states, (undefined) transitions, initial and final states, acceptance and rejection, input tape, and recognized language. Subsequent stages will address the concepts of NFA and PDA. For each stage, the following steps will be followed:
    (a) **Initial task design**: this task will cover the fundamental concepts necessary for understanding the model under consideration.
    (b) **Design of a task that encompasses all concepts**: designing a comprehensive task that integrates the key concepts of the model under consideration.
    (c) **Implementation of the tasks proposed in the previous items**: putting the previously designed tasks into practice, using GrameStation.
    (d) **Design of intermediate levels**: developing intermediate tasks to bridge basic and advanced concepts.
    (e) **Design of more advanced levels**: creating advanced tasks to challenge and deepen understanding.
    (f) **Pilot case study**: application of the implemented tasks to a group of 9th-grade students to identify potential issues and challenges, as well as to assess understanding of the introduced concepts.
    (g) **Redesign and implementation of all tasks**: Based on the results obtained in the pilot case study, adjustments for all tasks will be analyzed. Subsequently, the implementation of all tasks will proceed.
    (h) **Case study**: a case study will be conducted with a 9th-grade elementary school class to evaluate the effectiveness of the entire stage related to the model under consideration.

In this work, we only focus on the first stage related to DFA. A description of steps (a), (b), and (c) are presented in subsection 4.2, while steps (d) and (e) are described in section 5. Steps (f) to (h) are future work.

4.2. Activity: Designed and Implemented Tasks

The first stage is structured in phases, each represented by a DFA corresponding to a specific task the Explorer Robot must complete. The initial phase focuses on exploring the planet's terrain, utilizing sensors to identify safe zones and key natural resources such

as water and minerals. This paper will delve into this introductory phase, examining the first automaton of the initial phase.

The first phase is an introduction, starting with a simple automaton of a small number of states. The DFA is composed of the following states: *Resting–Earth*; *Traveling–Destination*; *Resting–Destination*; *Exploring–Mine*; and *Traveling–Earth*. The initial state of the DFA is *Resting–Earth*, where the robot starts the activity on its home planet, ready to begin the task. The *Traveling–Destination* state indicates that the robot is en route to the destination planet for exploration. Upon arrival, the robot enters the *Resting–Destination* state, preparing to start exploring the new planet, specifically a mine in this case. After exploration, the robot transitions to the *Traveling-Earth* state, meaning its return journey to the home planet. Transitions between the robot's states are triggered by commands *To Leave*, *To Land*, and *To Explore* (*"Partir"*, *"Aterrissar"*, and *"Explorar"* in Portuguese, respectively). The objective is to guide the robot through the DFA correctly, enabling it to complete the given task.

In the first task, the student receives a completed tape and must, using the appropriate rules, identify the state at which the robot should stop when executing the sequence of commands on the tape. The student then processes and checks the tape to verify if it is correct. If the robot reaches the state specified by the student, the student can advance in the game; otherwise, cannot. The tape never contains an invalid command sequence. In the second task, the student receives the complete automaton, including the final state(s), and a blank tape to construct the sequence of commands that will lead the robot from the initial state to one of the final state(s). In this scenario, the player may create an invalid sequence, resulting in an undefined transition and a "loss" of the phase. Alternatively, if the robot does not reach a final state by the end of the tape, it also results in a "defeat". The student only passes the level if they complete reading the entire tape and end in a final state.

Figure 4 illustrates the type graph of the activity. There are five possible states for the robot, with labeled transitions between these states. The yellow bell vertex demarcates the initial state, the robot is the main character, the spherical vertex with transparency indicates a non-final state, and the flag vertex represents the final state. There is also a vertex corresponding to the tape cells, which can contain command symbols or "whites", along with a red pointer indicating the current position of the tape cell. Additionally, there is a vertex indicating an error, which will be explained further in the rules.

Figure 5 depicts the initial graph, showing all states as non-final and with the robot in the first state (*Resting–Earth*). The tape's current symbol icon indicates that the tape is read from left to right. The stop state vertex (flag vertex) is also included so that, according to specific rules, the student can determine in which state the task will end. Figure 6 provides examples of rules that allow the player to designate the stop state in one of the states. These rules declare that if a state is non-stop (connected to the transparent vertex), it can be made stop. The edge connecting the state to the non-final vertex is then removed, and a new edge is created connecting the state to the final state flag. Since there are five states at this stage, five rules of this type are necessary.

In the first task, the student receives the completed tape and must, in addition to marking the final state, move the robot through the automaton. To achieve this, transi-

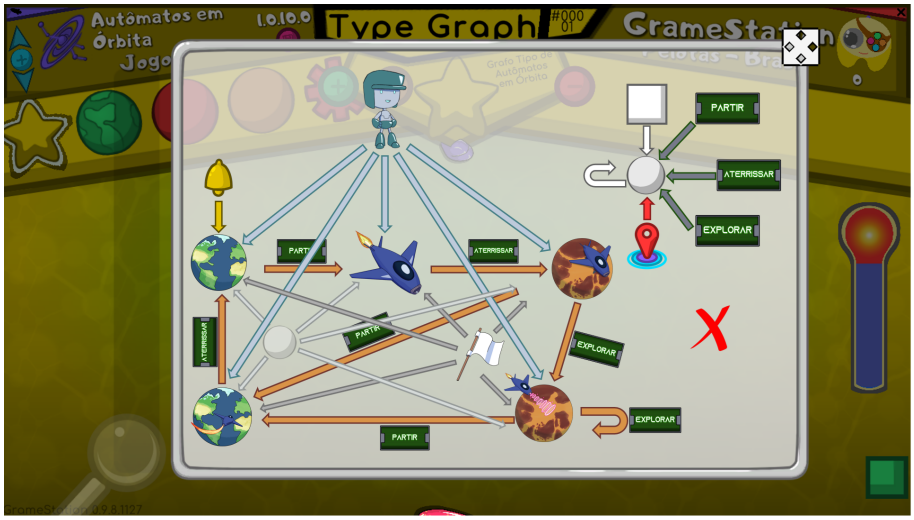

Figure 4. Activity type graph in GrameStation

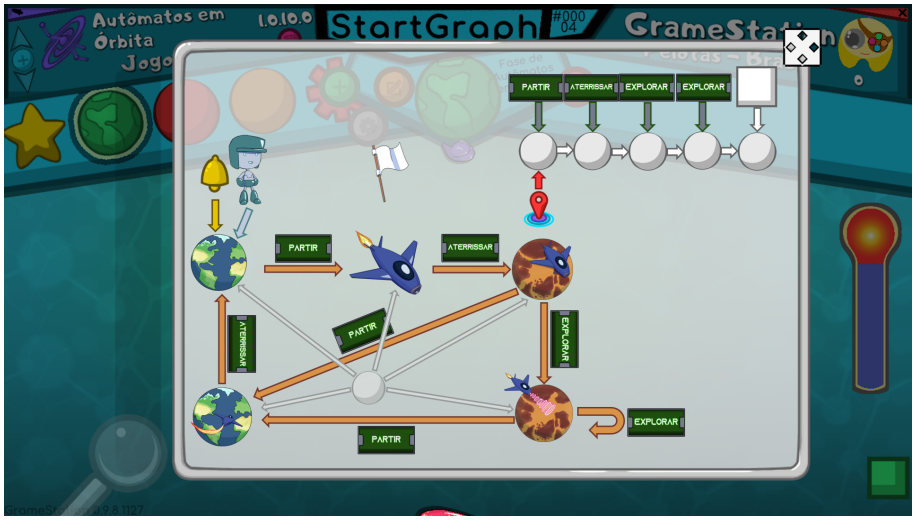

Figure 5. Activity initial graph in GrameStation

Figure 6. Examples of *"FinalState"* rules

tion rules are defined. Examples of these rules can be seen in Figure 7. If the transition between automaton states matches the current symbol on the tape, the transition can be executed. Since there are seven transitions in the automaton presented, seven transition

rules are needed. Figure 7 illustrates the transition from the *Resting–Earth* state to the *Traveling–Destination* state (left) and from the *Traveling–Destination* state to the *Resting–Destination* state (right).

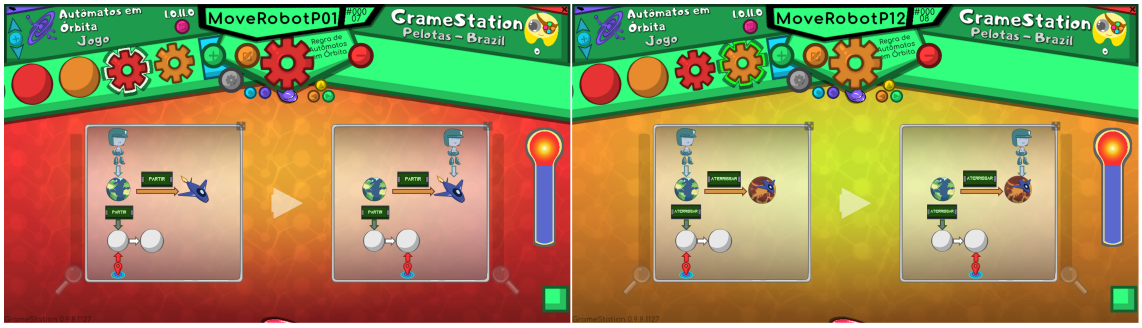

Figure 7. Examples of *"MoveRobot"* rules

In the second task, the student is aware of the final state of the automaton and must construct the sequence of commands on the tape to guide the robot to this state. In this case, the initial graph contains the flag vertex related to one or more states and the tape has only "white" vertices. Therefore, the player must replace the "white" symbols on the tape with the corresponding action labels. Figure 8 illustrates example rules for this task. Since there are three types of labels (*To Leave*, *To Land*, and *To Explore*; *"Partir"*, *"Aterrissar"*, and *"Explorar"* in Portuguese), three rules are required. Figure 8 shows two of these rules. Once the final state is marked and the tape is filled, the player can proceed to guide the robot through the automaton with the transition rules.

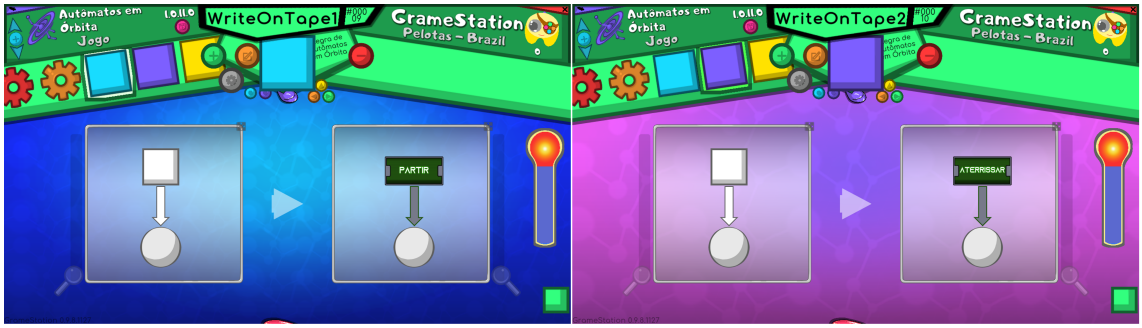

Figure 8. Examples of *"WriteOnTape"* rules

Additionally, rules are implemented to handle mistakes made by the player. In the first task, for example, the error is to finish reading the tape in a state that is not final. That is when the pointer is marking the last cell of the tape, but the state the robot is in has no relation to the flag vertex (Figure 9 on the left). In the second task, for example, selecting a state where the transition does not match the current label on the tape is an error. In this case, the robot is disconnected from the current state, and an error vertex is displayed on the screen. Figure 9 (right) illustrates example rules for this scenario. For instance, in the *Error2* rule, the robot is in the *Resting–Earth* state, but the current symbol on the tape is *To Explore* (*"Explorar"*). Since the only valid transition from the *Resting–Earth* state is *To Leave* (*"Partir"*), this rule results in an invalid command. In the second task, besides invalid transitions, it may also happen that the robot does not reach the final state, which also represents "defeat" in the phase.

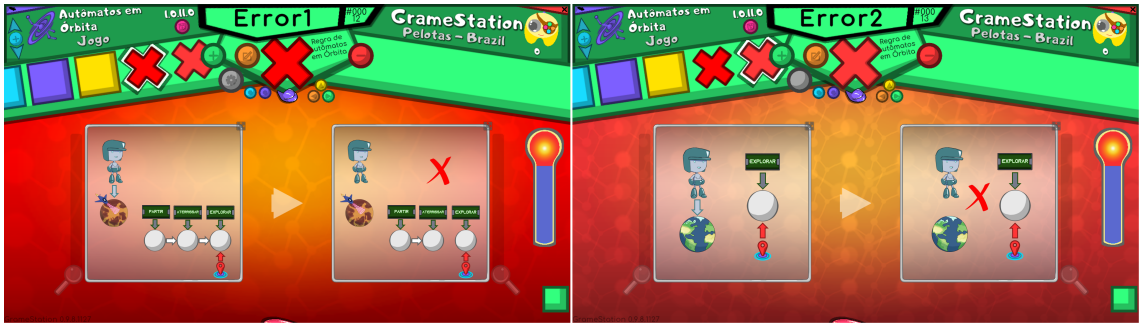

Figure 9. Examples of "Error" rules

5. Discussion

Table 1 presents a brief description of the proposed approaches, listing the covered concepts and highlighting key points to be considered. The tasks already implemented in GrameStation correspond to approaches 1 and 5. Approaches 6 and 7 will support the advanced levels.

Within the proposed approaches, alongside computational concepts, they also foster CT skills. In addition to the relationships already established by Silva Junior et al. (2019) using GG, it is important to note that the manipulation and specification of automata also enhance these skills. The use of automata to represent the robot's behavior during mission exercises *abstraction*, specifically through states representing its current condition and transitions representing actions that can modify this condition. When specifying automata, students define *algorithms* using an event-driven language to solve problems. When defining the tape for the robot to execute a specific mission, they are outlining the sequence of actions (*algorithm*) for the robot to accomplish the mission. Furthermore, to identify the language recognized by an automaton, it is necessary to discern the *patterns* across various input tapes and *generalize* the formation rules for all accepted words. *Debugging* skills can also be developed when students are tasked with defining an input tape that guides the robot to complete a mission. This allows students to simulate the tape processing to verify if the mission was accomplished.

6. Conclusion

This paper presents a proposal for an activity that utilizes automata theory in K-12, specifically using the formal language GG. The activity is designed to be engaging and educational, aiming to help students grasp the abstract concepts of automata theory through practical application in an interactive manner. By completing the activity's phases, students not only learn about automata but also develop CT skills, such as abstraction, data representation, data collection, analysis, simulation, and pattern recognition.

In future work, we intend to expand the activity by including additional phases, gradually increasing its complexity, besides exploring other types of automata, such as non-deterministic ones. Furthermore, we aim to test the activity with the target audience: primary school children.

Approach	Description	Comments	Automata Concepts
1	Students receive a complete tape and need to identify the robot's stopping state before simulating the automaton. Afterward, they must process the sequence of labels/commands on the tape to check if they identified the state correctly.	The tape never contains an invalid sequence of commands.	States and transitions.
2	Students are repeatedly given the same tape and must specify an initial state for the automaton. At each iteration, they simulate the automaton to observe the state in which the task ends.	Since the tape is the same, the task may or may not be completed, depending on the input. The result may vary based on the chosen initial state.	Initial state and transitions.
3	Students must determine which state the automaton will reach after processing a sequence of symbols and indicate whether the task will be completed (i.e., reaches the final state) or not.	There are two cases: (i) the word is read and the automaton stops at a final state, and (ii) the word is read and the automaton does not stop at a final state.	Acceptance or rejection of a word.
4	Given a sequence of instructions, the student must indicate whether the robot can complete the task (i.e., whether it is possible to read the entire tape).	Because the transition function is partial, it may not be possible to complete all instructions on the tape due to undefined transitions.	Undefined transitions.
5	Students are given a complete automaton, including the final state(s), and a blank tape to construct the sequence of commands that will lead the robot from the initial state to one of the final states.	It is possible to create an invalid command sequence, resulting in an undefined transition.	Initial and final states, transitions, acceptance or rejection of a word, and indefinite transitions.
6	Students must determine the required instructions to complete the task by testing different tape options. Ultimately, they must identify in a general manner what is necessary to achieve task completion.	Students may identify only a subset of the language.	Recognized language.
7	Given a language, the students must construct an automaton that accepts that language.	The provided language will be regular, ensuring it is always possible to create an automaton that recognizes it.	Automaton specification.

Table 1. Design of tasks at different levels

References

Barr, V. and Stephenson, C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community? *ACM Inroads*, 2(1):48–54.

Brazil (2022). Normas sobre Computação na Educação Básica. `http://portal.mec.gov.br/index.php?option=com_docman&view=download&alias=182481-texto-referencia-normas-sobre-computacao-na-educacao-basica&category_slug=abril-2021-pdf&Itemid=30192`. Online. Accessed on March 2024.

Carvalho, F., Junior, M. C., and Costa, Y. (2021). Jogos Educativos no Ensino de Autômato Finito Determinístico: Um Estudo de Caso com o Jogo A Factory Disaster. In *Anais Estendidos do XX Simpósio Brasileiro de Jogos e Entretenimento Digital*, pages 472–478, Porto Alegre, RS, Brasil. SBC.

Cayetano-Jiménez, I. U., Martinez-Ríos, E. A., Bustamante-Bello, R., Ramírez-Mendoza, R., and Ramírez-Montoya, M. S. (2024). Experimenting with Soft Robotics in Education: A Systematic Literature Review from 2006 to 2022. *IEEE Transactions on Learning Technologies*, pages 1–18.

Chen, P., Yang, D., Metwally, A. H. S., Lavonen, J., and Wang, X. (2023). Fostering Computational Thinking Through Unplugged Activities: A Systematic Literature Review and Meta-Analysis. *International Journal of STEM Education*, 10(1):47.

Ching, Y.-H. and Hsu, Y.-C. (2023). Educational Robotics for Developing Computational Thinking in Young Learners: A Systematic Review. *TechTrends*, pages 1–12.

da Silva, J. V., Junior, B. S., Foss, L., and Cavalheiro, S. (2021). Adaptação do processo engaged para o desenvolvimento de conteúdos curriculares em uma plataforma de jogos baseada em Gramática de Grafos. In *Anais do XXXII Simpósio Brasileiro de Informática na Educação*, pages 316–327, Porto Alegre, RS, Brasil. SBC.

Ehrig, H., Heckel, R., Korff, M., Löwe, M., Ribeiro, L., Wagner, A., and Corradini, A. (1997). Algebraic Approaches to Graph Transformation. Part II: Single Pushout Approach and Comparison with Double Pushout Approach. In *Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations*. World Scientific Publishing Co., Inc.

Farias, E., Lopes, P., Carvalho, W., and Porfírio, E. (2023). Análise da Adoção de Pensamento Computacional no Contexto Escolar Brasileiro: Um Mapeamento Sistemático da Literatura. In *Anais do XXXIV Simpósio Brasileiro de Informática na Educação*, pages 1625–1636, Porto Alegre, RS, Brasil. SBC.

Isayama, D., Ishiyama, M., Relator, R., and Yamazaki, K. (2016). Computer Science Education for Primary and Lower Secondary School Students: Teaching the Concept of Automata. *ACM Trans. Comput. Educ.*, 17(1).

Junior, B. S., Cavalheiro, S., and Foss, L. (2021). Gramestation: Specifying games with graphs. In *Anais do XXXII Simpósio Brasileiro de Informática na Educação*, pages 499–511, Porto Alegre, RS, Brasil. SBC.

Lee, S. J. and Kwon, K. (2024). A Systematic Review of ai Education in k-12 classrooms from 2018 to 2023: Topics, Strategies, and Learning Outcomes. *Computers and Education: Artificial Intelligence*, 6:100211.

Leite, L., Sibaldo, M. A., de Carvalho, T., and de Souza, R. (2014). Montanha de Chomsky: Jogo Tutor para Auxílio no Ensino de Teoria da Computação. In *Anais do XXII Workshop sobre Educação em Computação*, pages 110–119, Porto Alegre, RS, Brasil. SBC.

Mogensen, T. Æ. (2024). *Introduction to Compiler Design*. Springer Nature.

Resnick, M. (2017). *Lifelong Kindergarten: Cultivating Creativity Through Projects, Passion, Peers, and Play*. MIT Press.

Sanusi, I. T., Oyelere, S. S., Vartiainen, H., Suhonen, J., and Tukiainen, M. (2023). A Systematic Review of Teaching and Learning Machine Learning in K-12 Education. *Education and Information Technologies*, 28(5):5967–5997.

Silva, R. C., Binsfeld, R. L., Carelli, I. M., and Watanabe, R. (2010). Automata Defense 2.0: Reedição de um Jogo Educacional para Apoio em Linguagens Formais e Autômatos. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educaçao-SBIE)*, volume 1.

Silva Junior, B., Cavalheiro, S., and Foss, L. (2019). Revisitando um Jogo Educacional para Desenvolver o Pensamento Computacional com Gramática de Grafos. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 30, page 863.

Silva Junior, B., Cavalheiro, S., and Foss, L. (2021). Theoretical Computer Science in Basic Education: A Systematic Review. In *Anais do VI Workshop-Escola de Informática Teórica*, pages 133–140, Porto Alegre, RS, Brasil. SBC.

Silva Júnior, B. A. d. (2020). Ggasct: bringing formal methods to the computational thinking. Master's thesis, Universidade Federal de Pelotas.

Tomizawa, M. and Junior, M. C. (2021). Automata Toy Factory: Um Jogo Educativo para Ensino de Autômato com Pilha. In *Anais Estendidos do XX Simpósio Brasileiro de Jogos e Entretenimento Digital*, pages 389–397, Porto Alegre, RS, Brasil. SBC.

Vieira, M. and Sarinho, V. (2019). Máquina de Senhas: Um Jogo Digital para o Aprendizado da Teoria dos Autômatos. In *Anais da XIX Escola Regional de Computação Bahia, Alagoas e Sergipe*, pages 54–59, Porto Alegre, RS, Brasil. SBC.

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, 49(3):33–35.

Yim, I. H. Y. and Su, J. (2024). Artificial Intelligence (AI) Learning Tools in k-12 Education: A Scoping Review. *Journal of Computers in Education*, pages 1–39.