DELIMITER: A GAME FOR STACK DATA STRUCTURE TEACHING

Fabiana Camargo Bedreski[1]; Inali Wisniewski Soares[2]; Luciane Telinski Wiedermann Agner[3]

Abstract
In Computer Science, the use of serious games applied to education makes it possible to support the teaching of complex subjects, such as algorithms and logic. In this context, the Algorithm and Data Structure addresses the teaching of abstract structures, representing a learning challenge for many students. The Stack is a linear list considered an important resource for resolution of computing problems and is widely used in relevant areas such as Artificial Intelligence and Operational Systems. This paper presents a game intended for supporting the Stack data structure teaching process in a ludic way, making use of game elements as interaction, challenge, and motivation. The aim of the player consists in solving a relevant computing problem by employing the Stack. In this way, this tool allows the student to actively take part in the learning process, in addition to promoting the experimentation of this structure with a real problem.
Keywords: education; serious games; data structures; educational technology.

Resumo
Em Ciência da Computação, o uso de jogos sérios aplicados à educação possibilita apoiar o ensino de disciplinas complexas, como algoritmos e lógica. Neste contexto, Algoritmos e Estruturas de Dados abordam o ensino de estruturas abstratas, representando um desafio de aprendizagem para muitos alunos. A Pilha é uma lista linear considerada um importante recurso para resolução de problemas computacionais, amplamente utilizada em áreas relevantes como Inteligência Artificial e Sistemas Operacionais. Este artigo apresenta um jogo destinado a apoiar o processo de ensino da estrutura de dados Pilha de forma lúdica, utilizando elementos de jogos como interação, desafio e motivação. O objetivo do jogador consiste em resolver um importante problema computacional utilizando Pilha. Desta forma, esta ferramenta permite ao aluno participar ativamente do processo de aprendizagem, além de promover a experimentação desta estrutura com um problema real.
Palavras-chave: educação; jogos sérios; estruturas de dados; tecnologia educacional.

Introduction

Data structures are used to define how information is stored and accessed, being widely employed for the solution of various computing problems. The Stack, a linear data structure, is a LIFO list (Last In, First Out) used to build important applications such as Operational Systems and Compilers (Lafore, 2002; Langsam et al., 1996). However, one of the biggest challenges in

[1] Graduada em Bacharelado em Ciência da Computação pela Universidade Estadual do Centro-Oeste-UNICENTRO, professora do Departamento de Ciência da Computação da Universidade Estadual do Centro-Oeste (UNICENTRO). Email: fabianabedreski@unicentro.br.
[2] Doutora em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná-UTFPR, professora do Departamento de Ciência da Computação da Universidade Estadual do Centro-Oeste-UNICENTRO. Email: inali@unicentro.br.
[3] Doutora em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná-UTFPR, professora do Departamento de Ciência da Computação da Universidade Estadual do Centro-Oeste-UNICENTRO. Email: lagner@unicentro.br.

Stack teaching resides in demonstrating the use of this structure in the solution of real computing problems.

The use of games in education is an important resource that aims to make learning more interesting and increase students' motivation (Leitão et al., 2022; Hussein et al., 2019). Serious games arouse students' curiosity to explore new scenarios with emphasis on the teaching-learning process (Anastasiadis et al., 2018). In Algorithms and Data Structure Courses, games can support the teaching process through attractive graphical interfaces, in addition to allowing students to actively participate in the learning process (Su et al., 2021; Dicheva & Hodge, 2018; Lawrence, 2004).

This paper details the Delimiter game, a pedagogic approach focused on Stack teaching (Bedreski et al., 2021). In this context, the Delimiter provides a collaborative learning perspective, allowing the student to put the concepts learned into practice, in a ludic and interactive way. This game makes it possible to demonstrate the use of Stack data structure in practice for the "scope verification" solution in mathematical expressions. "Scope verification" is a classic Computer Science problem, so the use of stacks is intended for an easier solution of this problem (Goodrich et al., 2014; Feofiloff, 2009; Lafore, 2002). This way, Delimiter game addresses an important computing issue.

1 Methodology

The Delimiter was conceived to support the teaching of Stack data structure in DGBL (Digital Game-Based Learning) context.

1.1 DGBL

Games are usually intended for entertainment and amusement of the player. Nowadays, play is a common activity for most students (Anastasiadis et al., 2018). The use of games in Education aims to promote the engagement and motivation in the execution of a specific activity (Kim et al., 2018; Hussein et al., 2019). In this context, Digital Game-Based Learning (DGBL) refers to the use of digital games in learning environments (Breien & Wasson, 2021).

Researches have proved that educational games applied to education produces positive results, e.g. increasing students' interest and improving performance in the learning process (Anastasiadis et al., 2018). Games arouse curiosity based on elements such as competition, challenge, and reward (Kosa et al., 2016). Also, the use of pedagogic games aims at the engagement and motivation of students, reinforcing concepts through practice (Dicheva & Hodge, 2018).

Education can benefit from the use of games, since more and more students are acquainted with and immersed in digital Technologies (Anastasiadis et al., 2018). Additionally, several researches have pointed out great difficulty of educators catching students' attention by using passive and traditional teaching methodologies (Alsawaier, 2018; Caulfield et al.,2011).

1.2 Stack Data Structure

A data structure can be defined as a method of data storage, aiming at efficient recovery and search, in addition to providing a way to organize information in the memory for a better performance of an algorithm. Data structures are used to help the resolution of various computing problems (Sedgewick & Wayne, 2011; Langsam et al., 1996). Binary Tree, Queue, and Stack are widely known structures, being the third one the focus of the Delimiter tool.

The Stack main feature resides in the insertion and removal of new items being always carried out at the same extremity, called top. Among the main basic operations of the Stack, the functions called Push and Pop can be pointed out. Push adds a new element on the stack top, while Pop removes the element at the top of the stack (Goodrich et al., 2014; Lafore, 2002). This structure is used in several computing problems, such as compilers and operational systems (Sedgewick & Wayne, 2011; Langsam et al., 1996).

1.3 Stack-based Scope Verification

In the context of computer programming, expressions can be defined by the relation between numerical variables and constants, or operands. This relation is defined by using the arithmetical operators (Manzano & De Oliveira, 2005). In its turn, scope refers to the interval defined for each operation present in the logic or arithmetical expressions, making use of scope delimiters to ensure the execution order of those operations. In this way, the verification of these delimiters ensures the expected result through the execution of the operations in the expected order.

The scope verification process in expressions is part of several computing applications, and the Stack can be used to help solve this problem given the insertion and removal order of its elements. Such order defines that the last element added will necessarily be the first one to be removed (Goodrich et al., 2014; Lafore, 2002).

The algorithm that does the scope verification based on the Stack makes use of Push (insertion of an element into the stack) and Pop (removal of an element from the stack) functions.

An expression is composed of the following elements: operands, operators, scope opening characters (`(`, `[`, and `{`), and scope closing characters (`)`, `]`, and `}`). The algorithm analyzes all elements that form the expression, one by one, and executes the following actions (Goodrich et al., 2014; Lafore, 2002):

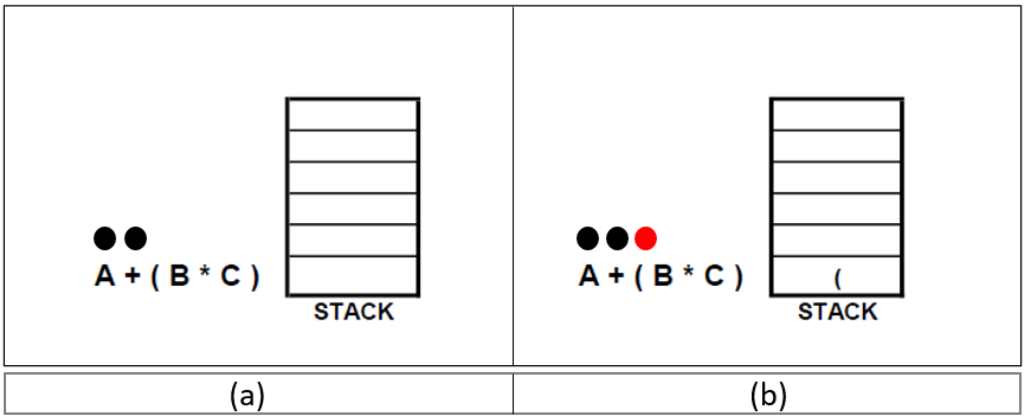
- If a scope opening is found (parentheses, brackets or braces), a Push of this character is done into the Stack.
- If a scope closing is found, the Stack must be analyzed, verifying each case as below:
  - Empty Stack: indicates that the scope closing character does not have a corresponding scope opening. This results in an incorrect expression.
  - Stack is not empty. A Pop operation in carried out in the respective Stack and the removed element is compared to the scope closing character that is being currently analyzed so the compatibility of types is verified. In case of no type correspondence, the expression is incorrect.
- If after the analysis of all elements of the expression, the Stack in not empty, then the expression is incorrect. When this happens, a scope opening is inserted into the Stack for which the corresponding closure was not found.
- In case the Stack is empty after the analysis of all elements of the expression, then the expression is correct.

In order to exemplify this solution based on Stack, the expression "A + (B * C)" will be used to verify whether it is correct with regard to its scopes. The main idea consists in verifying each character in the expression, seeking a scope opening delimiter (parentheses, braces or brackets). In case the character is an operator or operand, it is ignored and the next element is verified. Figure 1(a) shows the initial verification step, in which the dots indicate the characters already verified, i.e. "A" and "+". The next element that is verified is a scope opening delimiter (parentheses). This element is added to the Stack, as illustrated in Figure 1(b).

The next step is the verification of the remaining characters, seeking other scope opening or closing delimiters. When a scope finalizer is found, an element is removed from the Stack. Since the removed item will always be the element on the top, that is, the latest inserted one, it is possible to ensure that the opening and closing order for delimiters of the same type is preserved. To do so, a scope closing delimiter found need be compared to the opening delimiter removed from the Stack with regard to the type (parentheses, brackets or braces).

For the expression to be correct up to that point, compared elements necessarily need be of the same type. If not, the expression is incorrect. Additionally, closing delimiters found, and to which there is no corresponding element in the Stack, indicate an error in the expression.

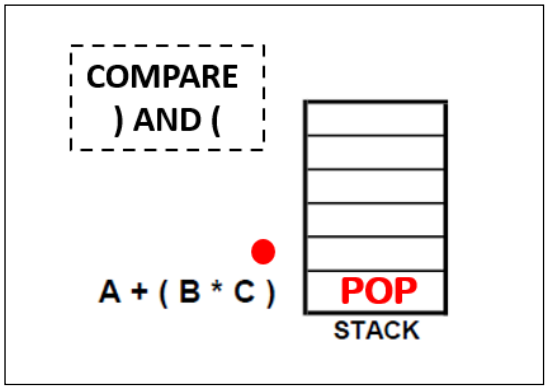
Figure 1 - Initial steps of the scope verification example.



Source: The authors.

Figure 2 shows the verification result of the scope delimiters for the example expression presented in Figure 1.

Figure 2 - Stack element removal in the scope verification example.



Source: The authors.

As the element is removed from the Stack top, the verification is done with the scope closing delimiter found in the expression, leading up to the verification that both are of the same type (parentheses) and evincing that the respective expression is correct. Through the Stack it is also possible the verification of non-finalized scopes, analyzing if through the end of the algorithm execution any element remains in the Stack.

2 Delimiter

The Delimiter proposes a teaching approach for the linear data structure called Stack by means of a digital game that allows the graphic visualization of the structure functioning. Additionally, the game application in the scope verification problem of mathematical expressions can be demonstrated.

Delimiter game was created using Unity4. Unity is a game development tool that provides several animation resources. The game initial screen image is presented in Figure 3. The current version of the Delimiter only supports the Portuguese language. However, one of the future works is to develop an English version of the game.

Figure 3 – Capture of the Delimiter initial screen.

Source: The authors.

The graphical interface definition was designed to be easy and clean, and is based on the following color scheme: purple, green, orange, and shades of gray (Figure 3). Button "Play" enables the player to start the game. Aiming to give the application an identity, a logo was created for the game, making use of the online and free version of the Canva[5] tool.

As shown in Figure 3 upper part, the logo was conceived so as to bring together the main elements of the game: the scope delimiters and the boxes representing the items to be added to the stack. With the objective of emphasizing the game theme, an opening bracket followed by a closing parenthesis was used to form the first letter of "Delimiter". This element was aligned with the orange square, making reference to the boxes used in the game. After pressing button "Play", a screen with expression options is presented to the player, as in Figure 4.

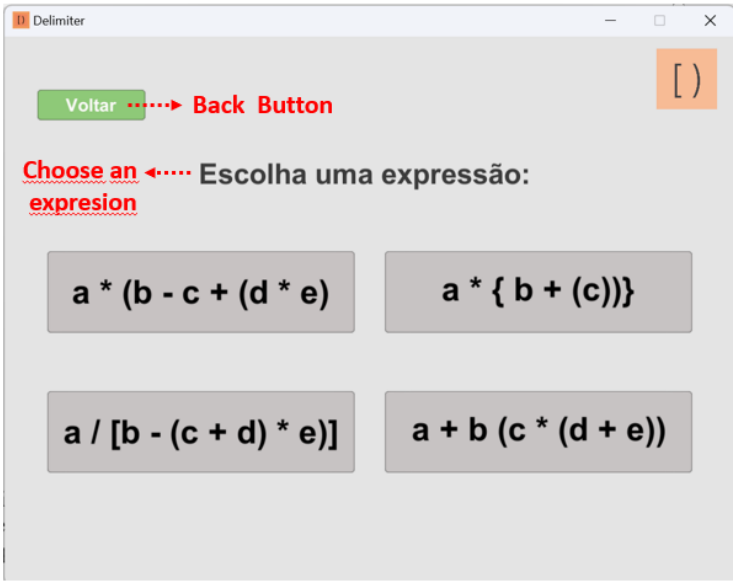
To proceed, the user must select one of the expressions shown. The objective consists in verifying if the scope delimiters of the selected expression are correct. Portuguese text was translated into English and highlighted in red in Figure 4.

It is important to point out that expressions vary in relation to the specific cases that must be covered by the algorithm to solve the problem in question. In this way, expressions present in Figure 4 were defined so as to address different results and, then, demonstrate distinct situations: i) expression "a*(b-c+(d*e)" is incorrect since it contains an open, non-finalized scope; ii) expression "a/[b-(c+d)*e)]" is incorrect since it contains a closing parenthesis without the respective opening; iii) expression "a*{b+(c))}" is not correct because it contains a scope

[4] https://unity.com/pt
[5] https://www.canva.com/pt_br/

finalizer that was not opened; and iv) expression “a+b(c*(d+e))” is correct, since the two scope openings of the parenthesis type were closed accordingly.

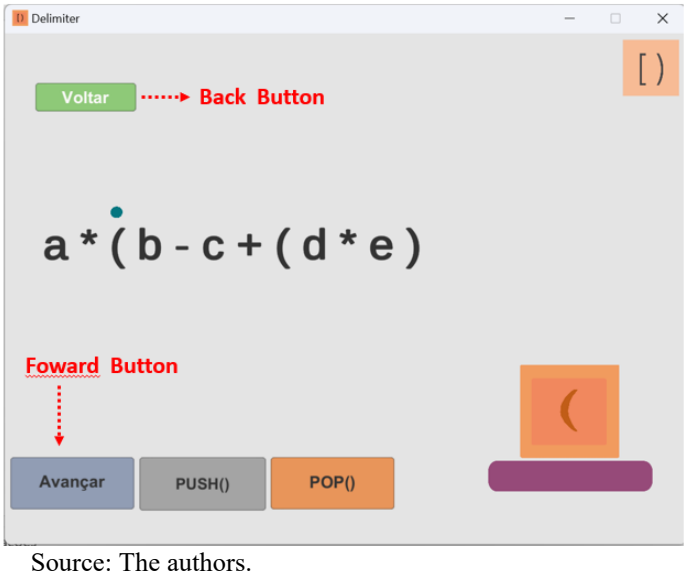
Figure 4 - Selection screen of the mathematical expression.



Source: The authors.

An important element of the Delimiter is about the “location dot”, used to indicate the expression character currently under analysis. Figure 5 exemplifies the location dot, indicating that the character of the expression must be analyzed by the player.

Figure 5 - Location dot indicating that character ‘a’ must be analyzed by the player.

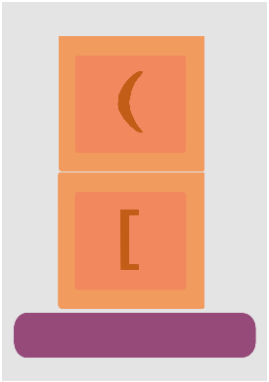


Source: The authors.

An expression may contain characters of the operand, operator, scope opening, and scope closing types. When a delimiter of the scope opening type is found in the expression, the player must add this character to the Stack using the button PUSH(). Likewise, as the location dot is

pointing to a scope finalizer, an element must be removed from the Stack by clicking on button POP(). The Stack items are represented by boxes that can be stacked (Figure 6).

Figure 6 - The Delimiter boxes.



Source: The authors.

Each box (Figure 6) is intended for the storage of the expression scope opening delimiters. In this way, boxes can be stacked according to the order in which they are found in the expression, so that the last delimiter stacked will be the first one to be removed from the stack for the type comparison required.

The scope verification process using Stack comprises previously defined rules. One of the main rules defines that only opening delimiters must be stacked. The player, however, can try to execute another action, such as stacking an operand or an operator. In such cases, the Delimiter interface includes events that set off warning messages indicating that incorrect actions were taken by the player. Besides informing the player about an incorrect action, these messages aim to teach the correct way to move forward and conclude the verification.

In addition to messages correcting actions performed by the player, there are also messages indicating next steps when scope openings or closings appear in the expression, making reference to the Push and Pop operations, respectively.

The stack state verification takes place when a scope closing delimiter is found. At this moment, the player is instructed to proceed with a Pop operation. When the player executes a Pop operation, in case the compared delimiters are corresponding, the box is destroyed and the next element is verified.

This animation sequence aims to highlight the importance of the Stack structure in the solution of the problem. As the opening delimiters are unstacked, by using the POP() button, and the comparisons are made, messages are displayed right above the expression presented, informing the result after each comparison. Whenever an incorrect closing delimiter is found, i.e. as the verification indicates that delimiters being compared are not of the same type, a message is displayed informing the expression status to the player.

2.1 Delimiter Creation and Implementation Details

For the game implementation, the Unity tool was used both in the mechanics and in the application environment building. Unity is a game development platform used to support 2D and 3D projects. This tool provides resources for game development and release in several devices, such as computers and cell phones.

The game interface, including images and the color scheme, was developed with Krita, a free open-source digital painting application. The Delimiter logo drawing was developed with Canva, a web-based free tool for image creation and editing.

The game coding was structured according to the main elements of the scope verification problem in arithmetic expressions, such as the stack and the delimiters. In the implementation context, classes were defined – or scripts as per the Unity Engine nomenclature – to represent the expression to be analyzed by the player, the index that goes through the character chain as well as the Push and Pop operations. A class is a structure that defines a set of objects with common features, allowing the definition of the behavior (methods) of these objects (Sommerville, 2015).

Two Stack data structures were employed in the Delimiter implementation: the first one stores the real delimiters of the character type found in the expression, while the another one stores the objects that represent the delimiters, i.e. the boxes used to indicate to the user which opening delimiters are stacked or were unstacked. In this way, the code of the game developed in programming language can also be used as an example of data structures in real scenarios.

3 Results and discussion

One of the greatest teaching challenges resides in giving students the opportunity to put concepts learnt into practice Delimiter was conceived as a supporting tool for Computer Science teaching, using game elements to provide a more dynamic and motivating learning approach. In so doing, students are able to experiment the application of this data structure to solve common problems in computing applications, since its goal consists in verifying whether a mathematical expression is correct according to its scopes.

In Stack teaching, a regular way to exemplify its functioning in theory consists in comparing its structure with book or dish stacking, usually with static drawings and signs (Lawrence, 2004; Lafore, 2002; Langsam et al., 1996; Wirth, 1985). The Delimiter main feature resides in allowing the visualization of the Stack operation through an animation that simulates a box stack, in which the player is able to either stack or remove a box on/from the top only. In

addition to reinforcing the concept, the “stack” and “unstack” animations allow students to actively participate in their learning process. This way, an element removed through the Pop operation must be the one positioned on top of the Stack, i.e. the last element inserted into this structure.

The use of box images to simulate their stacking aims to facilitate the learning of this structure. Besides enabling students to solve an important computing problem using the Stack, the Delimiter stands out for using graphic elements to make the understanding of this structure easier. Approaches as the one presented in this paper can contribute both with the related theory teaching and the firmer commitment of students, allowing them to play active roles in the knowledge construction.

As future works, new expressions can be added to the game so the players are able to test new possibilities. Also, at this moment the game only supports the Portuguese language. Figures 4 and 5 of this paper show some words of the game screens translated from Portuguese to English (highlighted in red). Therefore, another future work is the creation of the English version of Delimiter.

5 Conclusions

By using teaching-oriented serious games, students can play an important role in their knowledge acquisition, and thus improve their experience through the entire process, in addition to allowing the theory application in real problems and scenarios. Game elements such as rules, challenges, achievements, and rewards can be used to strengthen the engagement and commitment of students in the learning process. Other aspects such as graphic interface, sound components, and animations can also be used to arouse interest and, then, make teaching easier.

Digital games can be defined as a simulation and designed for several purposes, such as entertainment and training. The development of digital games, however, requires the clear definition of their objectives, rules, and requisites, in addition to specific knowledge of implementation technologies. In this way, the development process of a pedagogic game must define easy-to-understand tools, consistent simulations, and reachable challenges for the proposed problems, keeping the focus on the proposed teaching topic (Schell, 2008).

In several science fields, new concepts learning can be challenging, mainly when it includes complex subjects such as Algorithms and Data Structure. Traditional methodologies often raise little interest and motivation in students.

In the Computing Education context, the use of digital games reinforces the understanding of a specific concept, raises higher interest in students who, usually, already

make use of digital games for entertainment purposes. Also, when founded on well-defined rules, digital games allow tracking the learning pace of each student, since it allows the player to retry until he succeeds in each step of the game.

References

ALSAWAIER, R. S. The effect of gamification on motivation and engagement. **The International Journal of Information and Learning Technology**, v. 35, n. 1, p. 56-79, 2018. Disponível em: https://doi.org/10.1108/IJILT-02-2017-0009. Acesso em: 11 dez. 2025.

ANASTASIADIS, T.; LAMPROPOULOS, G.; SIAKAS, K. Digital game-based learning and serious games in education. **International Journal of Advances in Scientific Research and Engineering**, v. 4, n. 12, p. 139-144, 2018. Disponível em: https://doi.org/10.31695/IJASRE.2018.33016. Acesso em: 11 dez. 2025.

BEDRESKI, F. C.; AGNER, L. T. W.; SOARES, I. W. Gamificação Aplicada ao Ensino de Ciência da Computação. *In*: XXX EAIC - Encontro Anual de Iniciação Científica, 2021, Guarapuava, Brasil. **Anais** do XXX EAIC - Encontro Anual de Iniciação Científica, 2021.

BREIEN, F. S.; WASSON, B. Narrative categorization in digital game-based learning: engagement, motivation and learning. **British Journal of Educational Technology**, v. 52, n. 1, p. 91-111, 2021. Disponível em: https://doi.org/10.1111/bjet.13004. Acesso em: 11 dez. 2025.

CAULFIELD, C.; XIA, J.; VEAL, D.; MAJ, S. P. A systematic survey of games used for software engineering education. **Modern Applied Science**, v. 5, n. 6, p. 28-43, 2011. Disponível em: https://doi.org/10.5539/mas.v5n6p28. Acesso em: 11 dez. 2025.

DICHEVA, D.; HODGE, A. Active learning through game play in a data structures course. *In*: **Proceedings of the 49th ACM Technical Symposium on Computer Science Education**, pp. 834-839, 2018. Disponível em:https://doi.org/10.1145/3159450.3159605. Acesso em: 11 dez. 2025.

FEOFILOFF, P. **Algoritmos em linguagem C**. Rio de Janeiro: Elsevier, 2009.

GOODRICH, M. T.; TAMASSIA, R.; GOLDWASSER, M. H. **Data structures and algorithms in Java**. 6. ed. Hoboken: John Wiley & Sons, 2014.

HUSSEIN, M. H. *et al*. Effects of digital game-based learning on elementary science learning: a systematic review. **IEEE Access**, v. 7, p. 62465-62478, 2019. Disponível em: https://doi.org/10.3217/jucs-022-12-1558 . Acesso em: 11 dez. 2025.

KOSA, M. *et al*. Software engineering education and games: a systematic literature review. **Journal of Universal Computer Science**, v. 22, n. 12, p. 1558-1574, 2016. Disponível em: https://doi.org/10.3217/jucs-022-12-1558. Acesso em: 11 dez. 2025.

LAFORE, R. **Data structures and algorithms in Java**. 2. ed. Indianapolis: Sams Publishing, 2002.

LANGSAM, Y.; AUGENSTEIN, M. J.; TENENBAUM, A. M. **Data structures using C and C++**. 2. ed. Upper Saddle River: Prentice Hall, 1996

LAWRENCE, R. Teaching data structures using competitive games. **IEEE Transactions on Education**, v. 47, n. 4, p. 459-466, 2004. Disponível em: https://doi.org/10.1109/TE.2004.825053. Acesso em: 11 dez. 2025.

LEITÃO, R. *et al*. A systematic evaluation of game elements effects on students' motivation. **Education and Information Technologies**, v. 27, n. 1, p. 1081-1103, 2022. Disponível em: https://doi.org/10.1007/s10639-021-10651-8. Acesso em: 11 dez. 2025.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. de. **Algoritmos**: lógica para desenvolvimento de programação de computadores. 17. ed. São Paulo: Érica, 2005

SCHELL, J. **The art of game design**: a book of lenses. Boca Raton: CRC Press, 2008.

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Upper Saddle River: Addison-Wesley Professional, 2011.

SOMMERVILLE, I. **Software engineering**. 10. ed. Boston: Pearson, 2015.

SU, S. *et al*. A game-based approach for teaching algorithms and data structures using visualizations. *In*: ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 52., 2021, Online. **Proceedings** [...]. New York: ACM, 2021. p. 1128-1134. Disponível em: https://doi.org/10.1145/3408877.3432520. Acesso em: 11 dez. 2025.

WIRTH, N. **Algorithms and data structures**. Englewood Cliffs: Prentice-Hall, 1985.