,

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUCAS GONTARZ FAJARDO

DESENVOLVIMENTO DE UM SIMULADOR DE TESTE DE MESA PARA APOIO AO ENSINO DE ALGORITMOS REPRESENTADOS EM DIAGRAMAS DE BLOCOS

GUARAPUAVA

2026

LUCAS GONTARZ FAJARDO

DESENVOLVIMENTO DE UM SIMULADOR DE TESTE DE MESA PARA APOIO AO ENSINO DE ALGORITMOS REPRESENTADOS EM DIAGRAMAS DE BLOCOS

DEVELOPMENT OF A TABLETOP SIMULATION TOOL TO SUPPORT THE TEACHING OF ALGORITHMS REPRESENTED IN FLOWCHART

Projeto de Trabalho de Conclusão de Curso de Graduação apresentado como requisito parcial para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Diego Marczal

Coorientador: Profª. Drª Renata Luiza Stange

GUARAPUAVA

2026

RESUMO

Este trabalho propõe o desenvolvimento de uma ferramenta para execução passo a passo de algoritmos representados em diagramas de blocos, permitindo acompanhar o fluxo de execução de forma semelhante a um *depurador* de linguagens de programação. A motivação para a criação deste projeto está relacionada à necessidade de oferecer maior suporte didático a estudantes de programação, especialmente nas etapas iniciais do aprendizado, nas quais o uso de diagramas de blocos é adotado como ferramenta de abstração e compreensão de algoritmos. O objetivo principal consiste em permitir que o estudante acompanhe a execução de seus algoritmos de forma detalhada, observando o fluxo de controle e o estado das variáveis durante cada etapa do processo. Dessa forma, busca-se favorecer uma compreensão mais aprofundada do funcionamento dos algoritmos e do raciocínio lógico envolvido em sua execução. A metodologia empregada envolve a utilização dos elementos e das regras de interpretação dos diagramas de blocos já estabelecidos na disciplina de Pensamento Computacional e Linguagem De Programação Visual (PCLPV), adotando-os como base para a implementação de um mecanismo de execução passo a passo, capaz de simular a execução do algoritmo e apresentar ao usuário o contexto de execução em cada etapa. Como resultado, espera-se que a utilização da ferramenta contribua para o processo de ensino-aprendizagem em programação, promovendo maior autonomia ao estudante e favorecendo a compreensão da lógica algorítmica.

Palavras-chave: depurador; diagramas de blocos; validação semântica; lógica de programação; ensino de programação.

ABSTRACT

This work proposes the development of a tool for the step-by-step execution of algorithms represented through block diagrams, allowing the execution flow to be followed in a way similar to a programming language *debugger*. The motivation for creating this project is related to the need to provide greater educational support for programming students, especially during the early stages of learning, in which block diagrams are adopted as a tool for algorithm abstraction and understanding. The main objective is to allow students to follow the execution of their algorithms in detail, observing the control flow and the state of variables at each stage of the process. In this way, the work seeks to promote a deeper understanding of algorithm behavior and the logical reasoning involved in its execution. The methodology employed involves the use of the elements and interpretation rules of block diagrams already established in the PCLPV course, adopting them as the basis for implementing a step-by-step execution mechanism capable of simulating algorithm execution and presenting the execution context to the user at each stage. As a result, it is expected that the use of the tool will contribute to the teaching and learning process in programming, promoting greater student autonomy and supporting the understanding of algorithmic logic.

Keywords: debugger; flowcharts; semantic validation; programming logic; programming education.

LISTA DE ABREVIATURAS E SIGLAS

Siglas

AL	Algoritmos
FP	Fundamentos de Programação
IOO	Introdução à Orientação à Objetos
LP1	Linguagem de Programação 1
PCLPV	Pensamento Computacional e Linguagem De Programação Visual
PCPF	Pensamento Computacional e Fundamentos de Programação
SI	Sistemas para Internet
TSI-UTFPR	Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná - Câmpus Guarapuava

SUMÁRIO

1 INTRODUÇÃO

O ensino de programação introdutória apresenta desafios significativos, especialmente no que se refere ao desenvolvimento do raciocínio lógico por parte de estudantes iniciantes. Embora diferentes abordagens possam ser adotadas, a forma como os algoritmos são introduzidos exerce impacto direto no processo de aprendizagem e no desempenho acadêmico dos alunos.

No curso de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná - Câmpus Guarapuava (TSI-UTFPR), diferentes estratégias pedagógicas foram utilizadas ao longo dos anos para o ensino de lógica de programação. Inicialmente, a abordagem adotada baseava-se no uso da linguagem C como primeiro contato com programação, o que exigia dos estudantes a assimilação simultânea de conceitos lógicos e aspectos sintáticos da linguagem.

Em contraste, outras abordagens frequentemente introduzem o ensino de algoritmos diretamente em linguagens de programação, como JavaScript ou Java, o que pode ampliar a carga cognitiva inicial ao exigir, simultaneamente, a compreensão da lógica computacional e das particularidades sintáticas da linguagem (SUN; LIN; WU, 2024).

A análise histórica do desempenho dos estudantes nas disciplinas iniciais de programação ao longo dos diferentes períodos curriculares fornece indícios quantitativos dos impactos dessas escolhas pedagógicas. A evolução dos índices de aprovação pode ser observada de forma comparativa na Figura 1, que apresenta a distribuição do percentual de aprovação geral por período, e na Figura 2, que detalha o comportamento individual de cada disciplina.

Ao analisar a Figura 1, nota-se que o período de 2011 a 2015, com as disciplinas Linguagem de Programação 1 (LP1) e Algoritmos (AL), apresentou uma distribuição concentrada em valores baixos, com mediana agregada de 26,4% e baixa variabilidade, indicando que o fraco desempenho era consistente entre os semestres. No período de 2016 a 2022, com Pensamento Computacional e Fundamentos de Programação (PCPF) e Introdução à Orientação à Objetos (IOO), a mediana agregada caiu ligeiramente para 22,2%, ainda que tenham ocorrido valores discrepantes (*outliers*) acima de 57%, sugerindo semestres isolados de melhor desempenho possivelmente associados a fatores externos, como perfil da turma ou prática docente.

Cabe destacar que o período de 2016 a 2022 abrange também os anos em que as atividades acadêmicas ocorreram durante a pandemia de COVID-19. Esse registro é importante como referência temporal do contexto em que os dados foram produzidos. Em contraste, o período vigente (2023–2025), com as disciplinas Fundamentos de Programação (FP) e PCLPV, evidencia uma mudança expressiva tanto na mediana (47,7%) quanto na amplitude interquartil, indicando não apenas melhora geral, mas também maior variabilidade positiva nos resultados.

A Figura 2 complementa essa análise ao desagregar os dados por disciplina. Observa-se que a disciplina FP, introduzida na matriz vigente, atingiu mediana de 76,2%, valor substancialmente superior ao registrado por qualquer disciplina dos períodos anteriores. A disciplina

PCLPV, por sua vez, alcançou mediana de 43,1%, desempenho superior ao das disciplinas correspondentes nos períodos anteriores (LP1: 25,4%; AL: 31,4%; PCPF: 25,7%; IOO: 21,6%).

Esses resultados oferecem evidências de que a adoção de uma abordagem baseada em linguagem visual, ao reduzir a barreira sintática inicial, favoreceu a assimilação dos conceitos fundamentais de lógica de programação e contribuiu para a melhoria dos índices de aprovação. Ainda que não seja possível atribuir essa melhora exclusivamente à mudança de linguagem sem o controle de outras variáveis institucionais e pedagógicas, os dados indicam uma forte correlação entre a redução da carga sintática e o aumento do sucesso acadêmico dos estudantes.

Entretanto, a própria análise dos resultados revela que, mesmo com a adoção da abordagem visual, ainda persiste um gargalo importante no processo de aprendizagem, especialmente na disciplina PCLPV, cuja mediana de aprovação permanece abaixo de 50%. Esse cenário sugere que, embora a barreira sintática tenha sido reduzida, ainda existem dificuldades relacionadas à compreensão do fluxo de execução dos algoritmos e ao acompanhamento do estado das variáveis durante sua execução. Parte dessas dificuldades pode ser atribuída à ausência de ferramentas que auxiliem o estudante durante a construção e validação de algoritmos. O ensino por meio de diagramas de blocos é geralmente realizado em cadernos ou em plataformas de desenho, como o Draw.io, nas quais não há suporte à execução do algoritmo. Nesse contexto, o teste de mesa, técnica de rastreamento manual da execução de um algoritmo, variável por variável e passo a passo, é frequentemente realizado sem o apoio de mecanismos que auxiliem sua condução.

A ausência de recursos que apoiem esse processo pode dificultar a verificação do raciocínio desenvolvido pelo estudante, ao mesmo tempo em que soluções excessivamente automatizadas podem reduzir seu envolvimento cognitivo, impactando negativamente o processo de aprendizagem.

Dessa forma, os próprios dados históricos de desempenho do curso indicam que a evolução pedagógica ocorrida até o momento resolveu parcialmente o problema inicial (a barreira sintática), mas revelou uma nova necessidade pedagógica: oferecer ao estudante mecanismos que auxiliem na compreensão dinâmica da execução do algoritmo.

Diante desse cenário, identifica-se como problema central a ausência de uma ferramenta adequada que permita a construção de algoritmos por meio de diagramas de blocos de forma intuitiva, estruturada e alinhada às necessidades do ensino de lógica de programação no curso de Sistemas para Internet (SI). Assim, este trabalho propõe o desenvolvimento de um sistema de diagramas de blocos com um módulo integrado de teste de mesa, no qual o estudante poderá acompanhar a execução passo a passo do algoritmo com rastreamento do estado das variáveis a cada instrução executada, visando facilitar a criação, visualização e compreensão de algoritmos e contribuir para a melhoria do processo de ensino-aprendizagem nas disciplinas iniciais de programação.

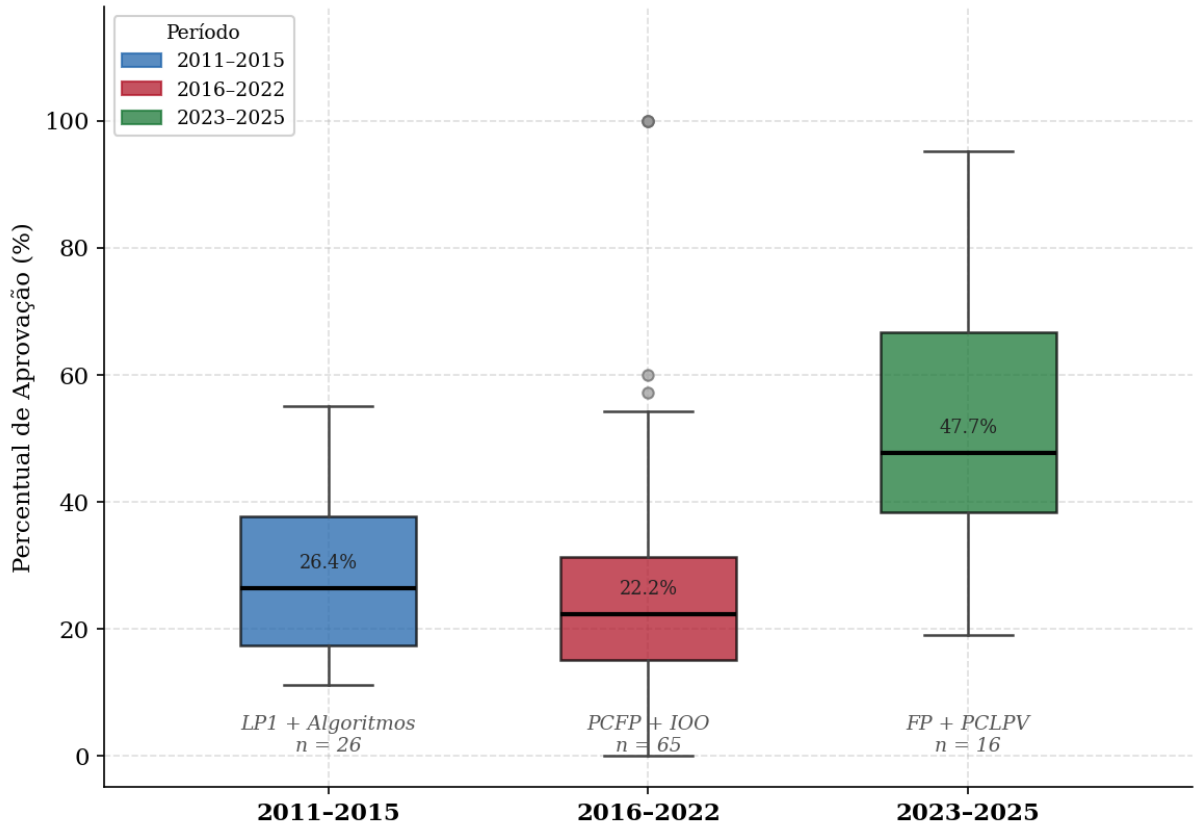


Figura 1 – Percentual de Aprovação Geral por Período Curricular
Fonte: Relatório de Aprovações da Disciplina do Curso de Sistemas para Internet (2026)

1.1 Objetivos

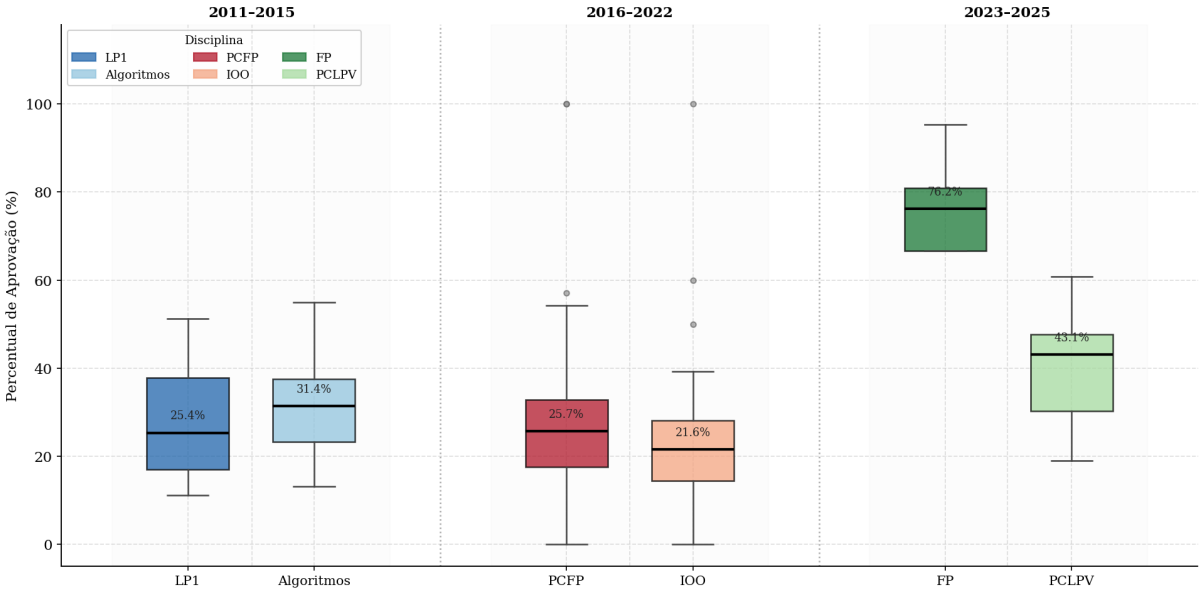


Figura 2 – Percentual de Aprovação por Disciplina e Período Curricular
Fonte: Relatório de Aprovações da Disciplina do Curso de Sistemas para Internet (2026)

Esta seção apresenta os objetivos a serem alcançados durante o desenvolvimento do trabalho.

1.1.1 Objetivo geral

Desenvolver um sistema web interativo capaz de executar, passo a passo, algoritmos representados em diagramas de blocos, permitindo a realização de testes de mesa e auxiliando no processo de lógica da programação.

1.1.2 Objetivos específicos

- Levantar e analisar os requisitos necessários para a construção de um teste de mesa aplicado a algoritmos representados por diagramas de blocos, com base no material utilizado na disciplina de Pensamento Computacional e Lógica de Programação Visual.
- Desenvolver um mecanismo de simulação voltado à execução de algoritmos representados por diagramas de blocos, permitindo a realização do teste de mesa de forma controlada, com a apresentação do estado das variáveis a cada etapa e a análise dos dados de entrada e saída.

2 FUNDAMENTOS E TRABALHOS RELACIONADOS

Este capítulo apresenta os principais fundamentos que embasam este trabalho, abordando conceitos relacionados ao pensamento computacional, à lógica de programação e às estratégias de ensino de algoritmos. Também são analisadas ferramentas utilizadas no ensino de programação, com o objetivo de identificar suas características e limitações, servindo de base para a proposta desenvolvida.

2.1 Fundamentos do Ensino de Programação

O ensino de programação envolve o desenvolvimento de habilidades cognitivas e estratégias que permitem ao estudante compreender, estruturar e resolver problemas de forma sistemática. Nesse contexto, destaca-se o papel do pensamento computacional e da lógica de programação como fundamentos essenciais para a construção de algoritmos e para a aprendizagem de conceitos na área da Computação.

2.1.1 Pensamento Computacional e Lógica de Programação

De acordo com Bower *et al.* (2017), o pensamento computacional pode ser compreendido como um conjunto de habilidades cognitivas utilizadas para analisar problemas, estruturar soluções e expressá-las de forma que possam ser executadas por humanos ou por computadores. Esse conceito envolve estratégias de resolução de problemas fundamentadas em princípios da ciência da computação e pode ser aplicado em diferentes áreas do conhecimento.

Entre os principais elementos que compõem o pensamento computacional, destacam-se os pilares da decomposição de problemas, o reconhecimento de padrões, a abstração e o desenvolvimento de algoritmos. A decomposição consiste em dividir problemas complexos em partes menores e mais gerenciáveis; o reconhecimento de padrões permite identificar similaridades entre problemas ou dados; a abstração envolve focar apenas nos aspectos essenciais da solução; e os algoritmos representam a definição de uma sequência lógica de passos para resolver um problema.

Nesse contexto, a lógica de programação pode ser entendida como a aplicação prática desses princípios. Segundo Xavier (2018), trata-se da organização ordenada e coerente de instruções que compõem um algoritmo, permitindo a resolução de problemas por meio de passos finitos e bem definidos.

Considerando a necessidade de facilitar a compreensão desses conceitos em níveis introdutórios, recursos visuais e estruturados podem contribuir significativamente para o processo de aprendizagem. Conforme Silva e Souza (2018), a aprendizagem torna-se mais significativa quando o estudante participa ativamente da construção do conhecimento. Nesse sentido, repre-

sentações visuais de algoritmos auxiliam no desenvolvimento do pensamento computacional e na compreensão da lógica de programação.

2.1.2 Diagramas de Blocos como Ferramenta de Ensino

Os diagramas de blocos são utilizados no ensino de algoritmos por permitirem a representação visual e estruturada da lógica de um programa. Essa abordagem facilita a compreensão do fluxo de execução, das decisões e dos processos envolvidos na solução de problemas computacionais. Segundo Manzano (2010), essa representação auxilia no desenvolvimento do raciocínio lógico ao permitir a visualização das etapas do algoritmo antes de sua implementação. Estudos como o de Crews e Ziegler (1998) indicam que estudantes iniciantes apresentam melhor desempenho e menor incidência de erros ao utilizar abordagens baseadas em fluxogramas.

2.1.3 Teste de Mesa e Compreensão do Fluxo de Execução

O teste de mesa consiste na simulação manual da execução de um algoritmo, permitindo acompanhar o comportamento das variáveis e identificar possíveis inconsistências lógicas. De acordo com Araújo (2022), essa técnica possibilita verificar se o algoritmo atende ao comportamento esperado. Além disso, conforme Forbellone e Eberspächer (2005), o teste de mesa contribui para o desenvolvimento do raciocínio lógico ao exigir que o estudante acompanhe passo a passo a execução do algoritmo. Pressman e Maxim (2016) destaca que essa prática permite identificar falhas conceituais que nem sempre são evidentes na execução automatizada.

2.1.4 Depuração como Estratégia de Aprendizagem

A depuração é o processo de identificação e correção de erros em programas. No contexto educacional, essa prática assume um papel pedagógico relevante ao estimular a reflexão sobre o funcionamento do algoritmo. Segundo (MCDOWELL; HELMBOLD, 1989), a depuração promove um aprendizado ativo, no qual o estudante formula hipóteses, testa soluções e refina seu raciocínio. Essa abordagem está alinhada ao construcionismo (SILVA; SOUZA, 2018), no qual o erro é parte essencial do processo de aprendizagem. Além disso, a depuração favorece o desenvolvimento de habilidades metacognitivas, permitindo que o estudante compreenda melhor suas estratégias de resolução de problemas e melhore seu desempenho em situações futuras.

2.2 Ferramentas Computacionais no Ensino de Programação

O uso de ferramentas computacionais no ensino de programação tem como objetivo facilitar a compreensão de conceitos abstratos e apoiar o desenvolvimento do pensamento computacional (WING, 2016). Diferentes abordagens têm sido propostas, variando entre ambientes visuais, representações gráficas e linguagens textuais simplificadas.

2.2.1 Scratch

O **Scratch** é uma das ferramentas mais difundidas no ensino introdutório de programação (MIT Media Lab, 2024). Desenvolvido pelo *MIT Media Lab*, o Scratch utiliza uma abordagem baseada em blocos gráficos encaixáveis, permitindo que os usuários criem programas sem a necessidade de escrever código textual. Essa característica contribui para a redução de erros sintáticos e favorece o aprendizado de conceitos fundamentais da programação, como sequência, repetição, estruturas condicionais e eventos. A ferramenta possibilita a criação de algoritmos por meio do arraste de blocos, integrando recursos como animações, controle de eventos e respostas dinâmicas, o que estimula a criatividade e o engajamento dos aprendizes.

A Figura 3 apresenta a interface do ambiente Scratch, na qual é possível observar a organização dos blocos de programação utilizados para a construção de algoritmos de forma visual.

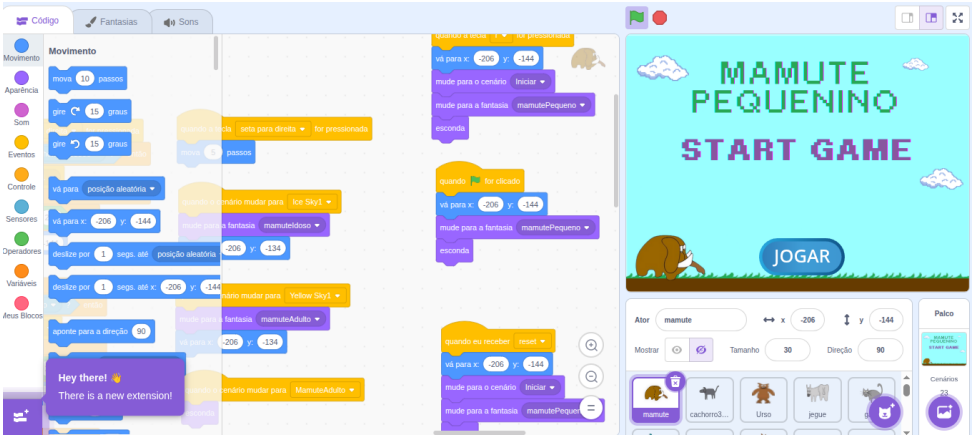


Figura 3 – Interface do ambiente Scratch com programação baseada em blocos
Fonte: (MIT Media Lab, 2024)

Apesar de sua eficácia no ensino introdutório, o **Scratch** não possui foco na representação formal de algoritmos por meio de diagramas de blocos estruturados nem na realização de testes de mesa. Sua abordagem é orientada a eventos e animações, o que pode dificultar a transição para modelos mais tradicionais de construção e análise de algoritmos.

2.2.2 Blockly

O **Blockly** é uma biblioteca de programação visual desenvolvida originalmente pelo Google (Google LLC, 2024). A ferramenta utiliza blocos gráficos manipuláveis por meio da técnica de arrastar e soltar (*drag-and-drop*), permitindo que conceitos fundamentais de programação sejam representados de forma visual. A biblioteca oferece grande flexibilidade para desenvolvedores, possibilitando a criação de ambientes personalizados de ensino e a geração automática de código em linguagens como JavaScript, Python e PHP.

A Figura 4 apresenta a interface de demonstração da biblioteca Blockly, evidenciando o ambiente de programação visual baseado em blocos.

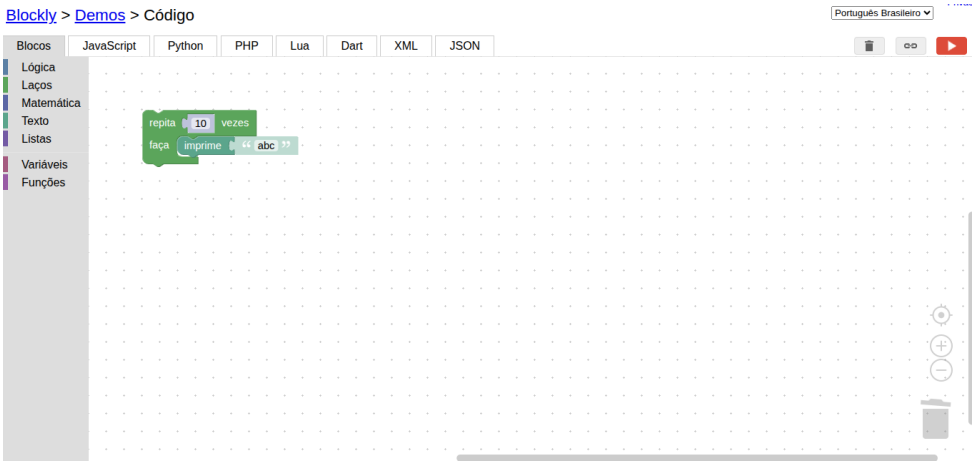


Figura 4 – Interface de demonstração da biblioteca Blockly
Fonte: (Google LLC, 2024)

No entanto o **Blockly** não é um ambiente educacional, mas sim uma biblioteca base sobre a qual ambientes educacionais podem ser construídos.

2.2.3 Lucidchart

O **Lucidchart** é uma ferramenta online de diagramação desenvolvida pela empresa Lucid Software (Lucid Software Inc., 2024). A plataforma é utilizada para a criação de fluxogramas, diagramas UML e outros tipos de representações gráficas utilizadas na modelagem de processos. No contexto educacional, o Lucidchart pode auxiliar no desenvolvimento do raciocínio lógico ao permitir que estudantes representem visualmente a estrutura de algoritmos antes da implementação em código.

A Figura 5 apresenta a interface da ferramenta Lucidchart utilizada para a criação de diagramas e fluxogramas.

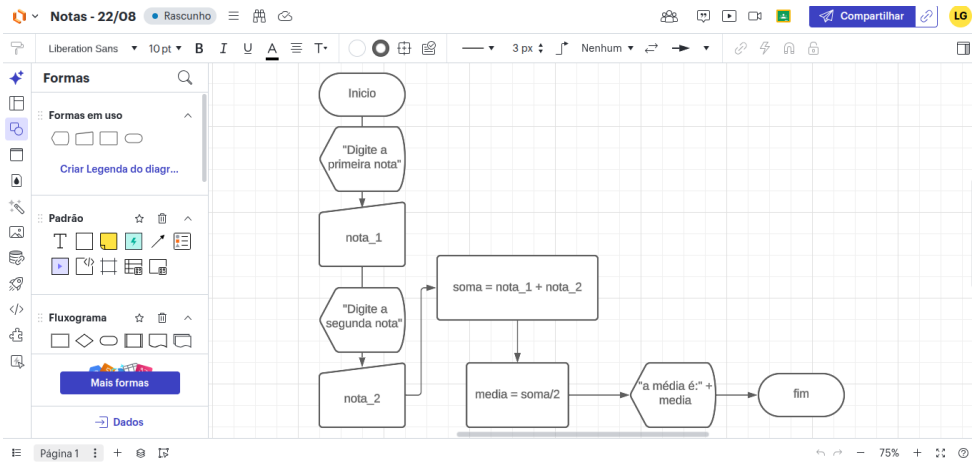


Figura 5 – Interface da plataforma Lucidchart para criação de diagramas
Fonte: (Lucid Software Inc., 2024)

No entanto, o **Lucidchart** não possui capacidade de execução ou validação de algoritmos, limitando-se à representação gráfica de fluxos e estruturas lógicas. Além disso, a plataforma não foi concebida com foco pedagógico específico para o ensino de programação, sendo uma ferramenta genérica de diagramação. Dessa forma, seu uso no contexto educacional restringe-se à criação de diagramas com os recursos visuais disponíveis, que em alguns casos podem divergir dos blocos e estruturas propostos em abordagens didáticas voltadas ao ensino de algoritmos.

2.2.4 Portugol

O **Portugol Studio** é um ambiente de programação educacional que utiliza uma linguagem baseada na língua portuguesa (Portugol Studio, 2023). Essa abordagem facilita o entendimento inicial das estruturas fundamentais da programação, como variáveis, estruturas condicionais e laços de repetição. A ferramenta é amplamente utilizada em cursos introdutórios de programação, pois permite que os estudantes concentrem seus esforços na compreensão da lógica do algoritmo, reduzindo a complexidade sintática comum em linguagens tradicionais.

A Figura 6 apresenta a interface do ambiente Portugol Studio.

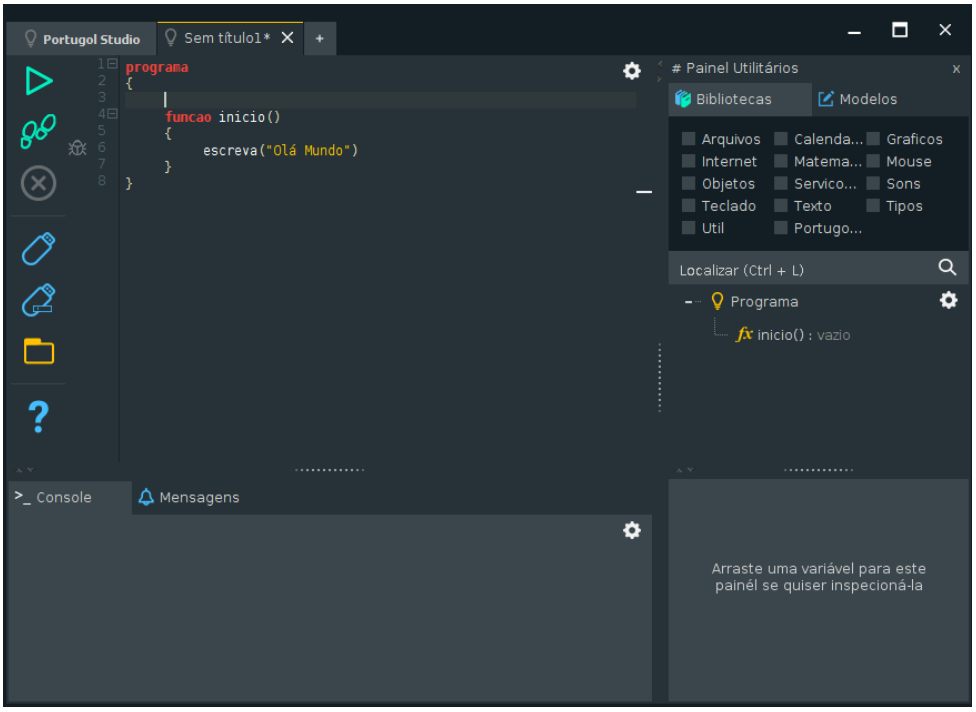


Figura 6 – Interface do ambiente de programação Portugol Studio
Fonte: (Portugol Studio, 2023)

O **Portugol Studio** não possui recursos visuais baseados em diagramas, o que não condiz com a abordagem inicial da disciplina, que utiliza fluxogramas.

2.2.5 Flowgorithm

O **Flowgorithm** é uma ferramenta educacional desenvolvida para auxiliar no ensino de lógica de programação por meio da construção de algoritmos utilizando fluxogramas (COOK, 2023). A ferramenta permite que o usuário crie algoritmos de forma gráfica e acompanhe sua execução passo a passo.

Essa abordagem visual contribui para a compreensão do fluxo de execução de um programa e para a identificação de possíveis erros lógicos durante o desenvolvimento do algoritmo.

A Figura 7 apresenta a interface da ferramenta Flowgorithm.

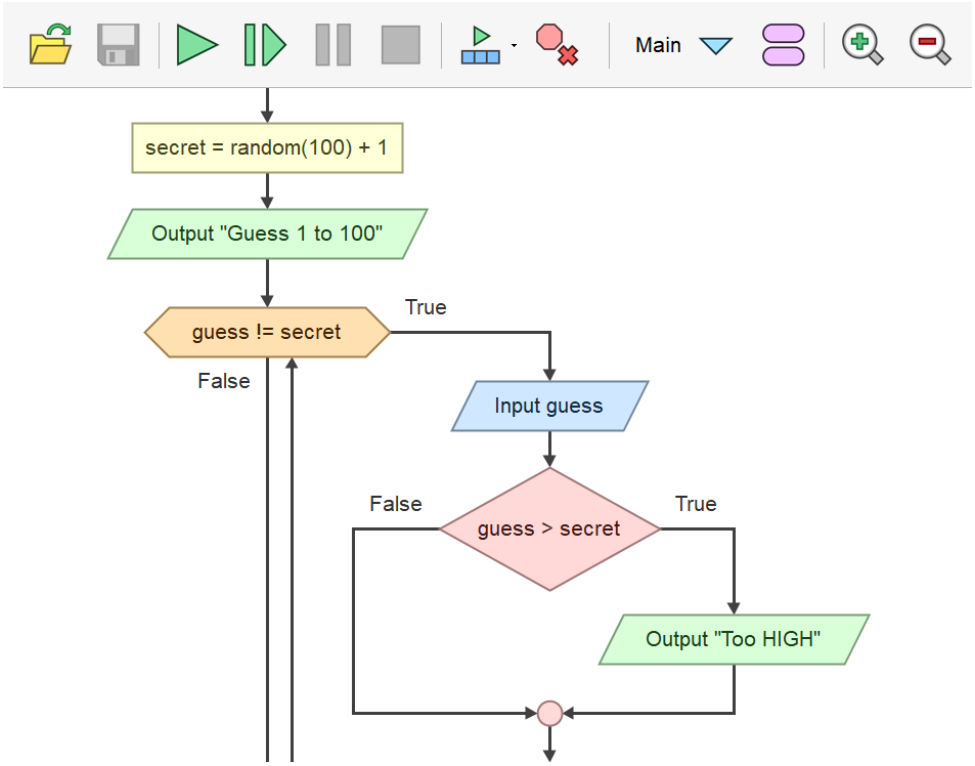


Figura 7 – Fluxograma do algoritmo de adivinhação de número desenvolvido no Flowgorithm
Fonte: Cook (2023)

No entanto, embora o **Flowgorithm** possibilite acompanhar a execução dos algoritmos por meio de funcionalidades de execução passo a passo, pausa e interrupção, a ferramenta não apresenta uma abordagem explicitamente voltada ao teste de mesa, com foco na representação organizada do estado das variáveis e do contexto de execução em cada etapa do algoritmo.

2.2.6 Análise Comparativa das Ferramentas

A análise das ferramentas apresentadas permite identificar diferentes abordagens no ensino de lógica de programação, variando entre ambientes baseados em blocos, representações gráficas e linguagens textuais simplificadas. Cada uma dessas abordagens contribui de maneira específica para o processo de ensino-aprendizagem, porém apresenta limitações quando considerada de forma isolada.

A Tabela 1 apresenta um comparativo entre as principais características das ferramentas analisadas.

Tabela 1 – Comparativo entre ferramentas de ensino de algoritmos

Ferramenta	Blocos	Fluxograma	Execução	Teste de Mesa	Foco Educacional
Scratch	Sim	Não	Parcial	Não	Alto
Blockly	Sim	Não	Não	Não	Médio
Lucidchart	Não	Sim	Não	Não	Baixo
Portugol	Não	Não	Sim	Parcial	Alto
Flowgorithm	Não	Sim	Sim	Parcial	Alto

Nesse contexto, o presente trabalho propõe o desenvolvimento de uma ferramenta que busca preencher essa lacuna no ensino de algoritmos, integrando a construção de soluções por meio de blocos com a execução de um **teste de mesa realizado passo a passo**. O sistema permite que o estudante execute o algoritmo de forma incremental, acompanhando o fluxo de execução, o estado das variáveis e a validação da lógica durante a própria construção do diagrama.

Dessa forma, o teste de mesa deixa de ser apenas uma atividade teórica e passa a atuar como um mecanismo ativo de verificação, possibilitando a identificação de inconsistências lógicas, prevenindo possíveis erros e auxiliando o estudante na validação de sua ideia à medida que o algoritmo é desenvolvido. Além disso, a ferramenta possibilita a conversão do fluxograma para uma linguagem de alto nível, permitindo a visualização do algoritmo em contextos reais de código.

3 METODOLOGIA

O desenvolvimento deste trabalho será conduzido por meio de uma abordagem incremental, orientada à construção de um sistema capaz de representar e simular a execução de algoritmos de forma visual e pedagógica.

A proposta do sistema não se limitará à execução automática de algoritmos, mas terá como foco a construção de um mecanismo que reproduza, de forma assistida, o processo de teste de mesa, amplamente utilizado no ensino de lógica de programação. Dessa forma, as decisões metodológicas buscarão integrar representação visual, execução controlada e acompanhamento do estado do algoritmo.

3.1 Abordagem Geral

A metodologia foi estruturada em dois eixos principais:

1. modelagem da representação dos algoritmos por meio de diagramas de blocos;
2. desenvolvimento de um mecanismo de execução orientado ao teste de mesa.

O primeiro estabelece a forma de representação dos algoritmos, servindo como base para o processo de execução. O segundo, que constitui o foco deste trabalho, concentra-se no desenvolvimento de um mecanismo capaz de interpretar essa representação e permitir a realização do teste de mesa, acompanhando a execução do algoritmo e o estado das variáveis a cada etapa.

Essa organização permite separar a construção do algoritmo de sua execução e análise, mantendo independentes os processos de modelagem e de validação.

3.2 Tecnologias utilizadas

A solução será desenvolvida seguindo uma arquitetura cliente-servidor, visando a separação de responsabilidades entre interface e processamento.

No frontend, serão utilizadas tecnologias voltadas à construção de interfaces interativas, como React e TypeScript, além da biblioteca React Flow, empregada na criação e manipulação de diagramas de blocos. Para o versionamento de código será utilizado o Git, enquanto o GitHub será empregado para o armazenamento e gerenciamento do repositório do projeto. O Docker será adotado para a padronização do ambiente de desenvolvimento, garantindo maior consistência entre os diferentes ambientes utilizados durante o processo de desenvolvimento (Meta Platforms, Inc., 2024; GitHub, Inc., 2024; Git Project, 2026; Docker, Inc., 2024; React Flow Team, 2026).

O backend será desenvolvido com Ruby on Rails, sendo responsável pelo gerenciamento de dados, persistência e apoio às funcionalidades da aplicação. A execução dos algoritmos, bem como o controle passo a passo característico do teste de mesa, será realizada no frontend por meio de TypeScript e React, permitindo maior interatividade e resposta imediata ao usuário (Ruby Core Team, 2023; Ruby on Rails Core Team, 2024; Meta Platforms, Inc., 2024).

A persistência de dados será realizada por meio do sistema gerenciador de banco de dados PostgreSQL (PostgreSQL Global Development Group, 2024).

3.3 Validação

A validação da solução será realizada por meio de testes funcionais, com o objetivo de verificar a coerência do mecanismo de execução dos algoritmos representados em diagramas de blocos e garantir a consistência dos resultados produzidos pelo sistema. Para isso, serão definidos algoritmos de referência contendo diferentes estruturas de controle, incluindo sequências simples, estruturas condicionais e laços de repetição.

Os testes serão conduzidos utilizando diferentes conjuntos de entradas, permitindo avaliar o comportamento do sistema em múltiplos cenários de execução. Durante a validação, serão analisados aspectos como o fluxo de execução percorrido, a atualização do estado das variáveis ao longo da simulação e a correspondência entre os resultados obtidos pelo sistema e aqueles esperados para cada algoritmo.

Os critérios adotados para determinar o funcionamento correto da solução consistem na execução adequada das estruturas algorítmicas representadas nos diagramas, na atualização consistente das variáveis durante cada etapa do processo e na equivalência entre os resultados gerados pelo sistema e os resultados esperados a partir da execução manual do teste de mesa tradicional.

Assim, busca-se garantir que o mecanismo implementado seja capaz de reproduzir de forma confiável o comportamento esperado dos algoritmos representados, permitindo sua utilização como ferramenta de apoio ao processo de aprendizagem.

4 ANÁLISE E PROJETO DO SISTEMA

Este capítulo apresenta a análise e o projeto do sistema proposto, descrevendo os principais elementos que compõem a solução, bem como as decisões de projeto adotadas. São detalhados os componentes responsáveis pela representação dos algoritmos, as regras estruturais definidas e o mecanismo de execução baseado em teste de mesa.

4.0.1 Diagramas de Blocos

Cada bloco do diagrama possui uma função específica, estando associado a operações como entrada de dados, processamento, tomada de decisão e saída de informações. As conexões estabelecidas entre os blocos definem a ordem e o fluxo de execução do algoritmo, permitindo representar de forma visual e estruturada as etapas necessárias para a resolução de um problema.

A Tabela 2 apresenta os símbolos utilizados na construção dos diagramas de blocos adotados neste projeto. Esses elementos possuem semântica bem definida e representam diferentes unidades lógicas de execução, sendo fundamentais para a interpretação dos algoritmos representados no sistema.

Os símbolos adotados têm como base o material didático utilizado na disciplina de PCLPV[1], sendo empregados como referência para a modelagem da sintaxe e da semântica dos algoritmos considerados neste trabalho.

[1] Disponível em: Material de Pensamento Computacional e Linguagem de Programação Visual (PCLPV).

Tabela 2 – Símbolos de fluxograma e suas utilidades

Representação	Nome	Utilidade
	Início/Fim	Indica o ponto de início e término de um algoritmo ou processo.
	Entrada	Representa a operação de entrada de dados no sistema.
	Saída	Indica a apresentação dos resultados ao usuário.
	Memória	Representa a memória do algoritmo, utilizada para definição de variáveis.
	Processo	Representa operações como cálculos e atribuições.
	Fluxo	Indica a direção do algoritmo.
	Decisão	Representa um ponto de escolha baseado em condição.
	Conector	Liga partes distintas do fluxograma.
	Sub-rotina	Representa uma subrotina reutilizável.

A Figura 8 ilustra um exemplo de diagrama construído com base nas definições.

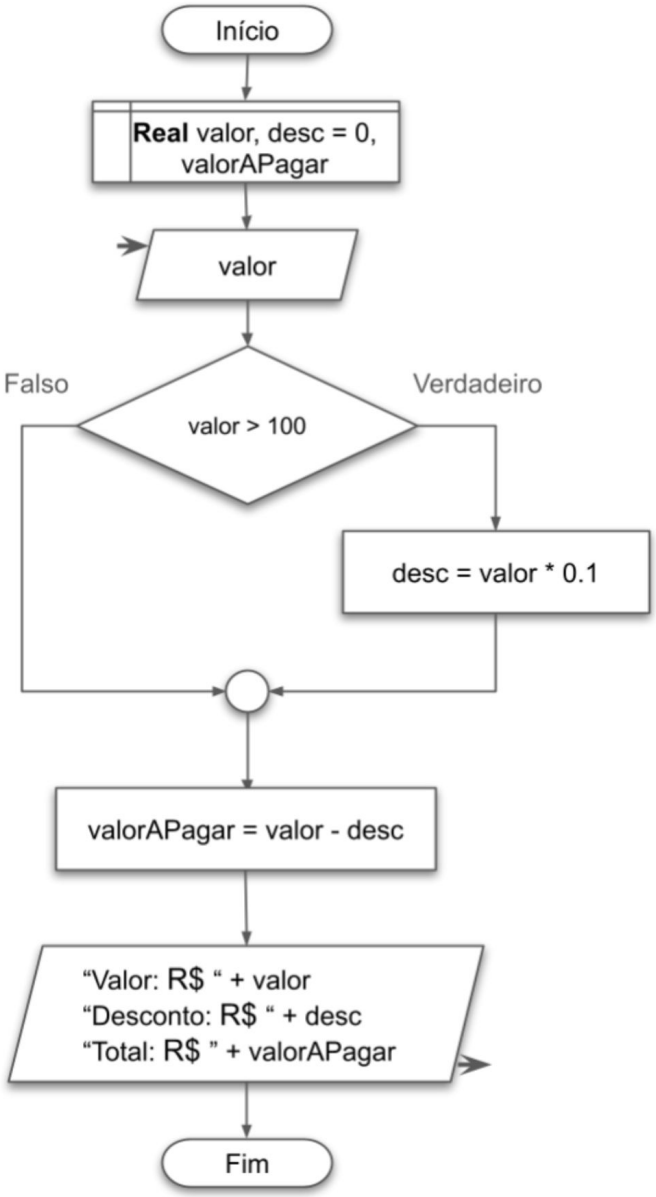


Figura 8 – Exemplo de diagrama de blocos
Fonte: ***Domine a Arte de Aprender a Programar e Pensar Computacionalmente***

4.0.2 Simulação de Execução e Teste de Mesa

Para simular a execução do diagrama de blocos é utilizado um artefato chamado teste de mesa. Esse processo consiste na simulação passo a passo do fluxo de execução do algoritmo, permitindo acompanhar, de forma estruturada, a evolução do estado das variáveis ao longo da execução.

Conforme ilustrado na Figura 9, o processo de teste é composto por duas representações complementares:

(a) o diagrama de blocos do algoritmo, que define a sequência lógica das operações,

(b) a tabela de execução, que registra o estado das variáveis a cada passo.

Passo	Instrução	VARIÁVEIS DO ALGORITMO						Saída
		Real				Inteiro		
		vmKm	dm	dKm	t	h	min	
1	Início							
2	Definição das variáveis							
3	Entrada	100.0						
3.1	Entrada		240000.0					
4	Processo dKm = 240000 / 1000			240.0				
5	Processo t = 240.0 / 100.0				2.4			
6	Processo h = (**inteiro**) 2.4					2		
7	Processo min = (**inteiro**) ((2.4 - 2) * 60)						24	
8	Saída							2 horas e 24 minutos
9	Fim							

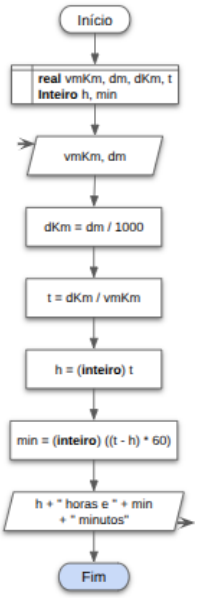


Figura 9 – Teste de mesa realizado de forma manual
Fonte: Documentação da disciplina PCLPV

O sistema integra essas duas perspectivas, permitindo que cada bloco do diagrama seja associado diretamente a uma etapa da simulação. A execução é iniciada a partir do bloco de início do algoritmo, seguindo o fluxo definido pelas conexões entre os blocos. Inicialmente, ocorre a definição das variáveis e, em seguida, a leitura dos dados de entrada. A partir desse ponto, cada bloco de processo ou decisão é interpretado sequencialmente, atualizando o estado interno do sistema. Durante a execução, o sistema mantém uma estrutura de dados que representa as variáveis do algoritmo, incluindo seus valores atuais e tipos associados. A cada etapa, uma nova linha é registrada na tabela de teste, contendo:

1. o passo de execução;
2. a instrução correspondente ao bloco executado;
3. os valores atualizados das variáveis;
4. eventuais saídas geradas pelo algoritmo.

No exemplo apresentado na Figura 9, observa-se a execução de um algoritmo para solução do exercício Tempo de Viajem.

Esse modelo permite não apenas verificar a correção dos resultados finais, mas também identificar erros lógicos durante o processo de execução, uma vez que o usuário pode acompanhar detalhadamente cada modificação no estado das variáveis. Além disso, a associação direta entre blocos e passos da tabela favorece a compreensão do fluxo de execução, especialmente em contextos educacionais.

4.1 Requisitos

Os requisitos do sistema foram organizados em dois módulos principais:

- **Módulo de Construção de Algoritmos:**

  Este módulo é responsável pela criação e edição dos diagramas de blocos, oferecendo suporte à representação visual dos algoritmos. Destaca-se, entretanto, que esse módulo não faz parte do escopo deste trabalho, sendo desenvolvido em um projeto complementar por Emanuel de Oliveira Andrade (ANDRADE, 2026). Sua descrição é apresentada apenas para contextualizar a integração entre os componentes da solução proposta.

  - permitir a construção de algoritmos por meio de blocos visuais;
  - possibilitar a definição do fluxo de execução a partir de conexões entre blocos;
  - oferecer suporte a estruturas sequenciais, condicionais e de repetição;
  - permitir a definição de variáveis e expressões associadas aos blocos;
  - garantir a consistência estrutural do diagrama, impedindo conexões inválidas.

- **Módulo de Execução e Teste de Mesa:**

  Este módulo é responsável pela interpretação dos diagramas construídos, simulando sua execução de forma controlada. Dentre os requisitos associados, destacam-se:

  - executar algoritmos de forma passo a passo;
  - percorrer o fluxo do diagrama com base nas conexões entre blocos;
  - manter e atualizar o estado das variáveis durante a execução;
  - apresentar a evolução do estado das variáveis ao longo do tempo;
  - identificar inconsistências lógicas durante a execução do algoritmo.

Embora distintos, os módulos são complementares, uma vez que os diagramas construídos no módulo de edição constituem a entrada para o processo de execução realizado pelo módulo de interpretação.

4.2 Protótipo Funcional

Com o objetivo de validar a proposta do sistema e explorar aspectos relacionados a organização da interface e integração dos componentes, foi desenvolvido um protótipo funcional utilizando a plataforma Lovable (Lovable, 2026), uma ferramenta baseada em inteligência artificial que permite a geração de sistemas a partir de descrições em linguagem natural.

A utilização dessa ferramenta teve como principal finalidade a viabilização da proposta, possibilitando a visualização prática do funcionamento do sistema antes de sua implementação definitiva. Por meio da construção iterativa baseada em *Prompts*, foi possível simular diferentes abordagens de interface, organização dos blocos de algoritmos e apresentação do teste de mesa.

O protótipo permitiu avaliar elementos fundamentais da proposta, como a disposição dos componentes na interface, o fluxo de interação do usuário durante a execução do algoritmo e a forma de exibição do estado das variáveis ao longo do processo. Além disso, possibilitou a identificação de ajustes necessários para melhorar a clareza visual e a experiência do usuário.

Durante esse processo, foram realizadas discussões com os orientadores, visando alinhar as decisões de projeto às melhores práticas pedagógicas e técnicas. Essas interações contribuíram para a definição mais precisa dos elementos que compõem o sistema, como os tipos de blocos utilizados nos diagramas, o comportamento do mecanismo de teste de mesa e a integração entre os diferentes componentes da aplicação.

Dessa forma, o uso do protótipo se mostrou fundamental para validar a viabilidade da solução proposta, orientar decisões de projeto e reduzir incertezas antes da etapa de desenvolvimento completo do sistema.

A Figura 10 apresenta um algoritmo sequencial de soma de dois números, representando a interface inicial do sistema com a área de modelagem do algoritmo e o painel de teste de mesa.

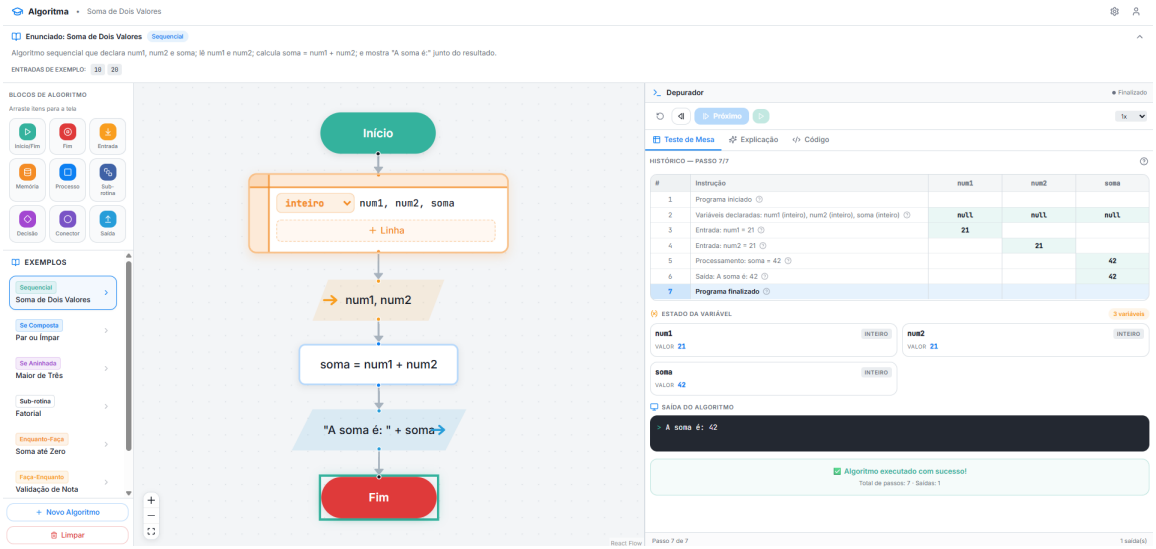


Figura 10 – Protótipo inicial da interface com diagrama sequencial de execução de algoritmos. Fonte: Elaborado pelo autor.

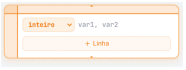
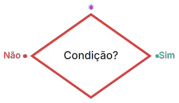
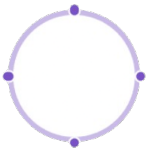
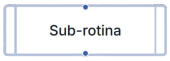
Ressalta-se que os protótipos representam uma versão preliminar da interface, estando sujeitos a ajustes e melhorias ao longo do processo de desenvolvimento, especialmente em relação à usabilidade e organização dos componentes.

4.2.1 Definição dos Símbolos Conceituais da Plataforma

Com base na modelagem do sistema, foram definidos os símbolos conceituais utilizados na construção dos diagramas de blocos. Esses símbolos seguem a representação clássica de fluxogramas, adaptados para implementação na ferramenta React Flow, utilizada no protótipo da aplicação.

A Tabela 3 apresenta a correspondência entre a representação conceitual (base teórica) e a implementação visual utilizada no sistema.

Tabela 3 – Correspondência entre símbolos conceituais e implementação no protótipo

Elemento	Representação Conceitual	Implementação no Protótipo
Início/Fim		
Entrada		
Saída		
Memória		
Processo		
Decisão		
Conector		
Sub-rotina		
Setas		

Fonte: Elaborado pelo autor com base na disciplina de Pensamento Computacional e protótipo do sistema.

A Figura 11 apresenta a implementação de um algoritmo responsável por verificar se um número informado é par ou ímpar, desenvolvido como parte do protótipo do sistema.

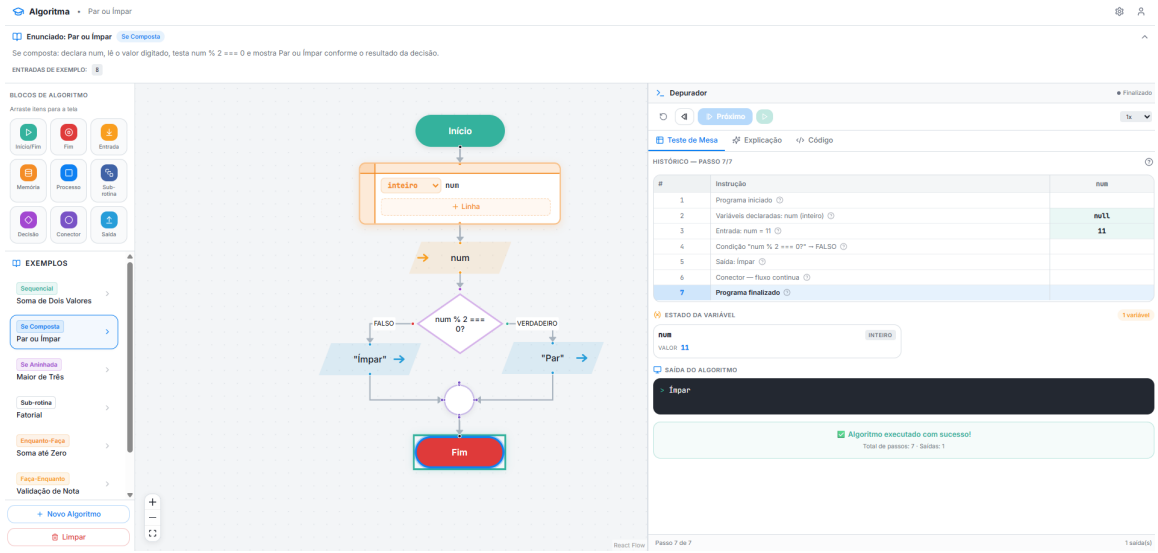


Figura 11 – Protótipo inicial da interface para o algoritmo de decisão (estrutura SE composta). Fonte: Elaborado pelo autor.

A Figura 12 apresenta o diagrama de blocos do algoritmo, no qual é possível observar o fluxo de execução e o ponto de término do processo. Esse fluxo está relacionado com o teste de mesa apresentado na Figura 13.

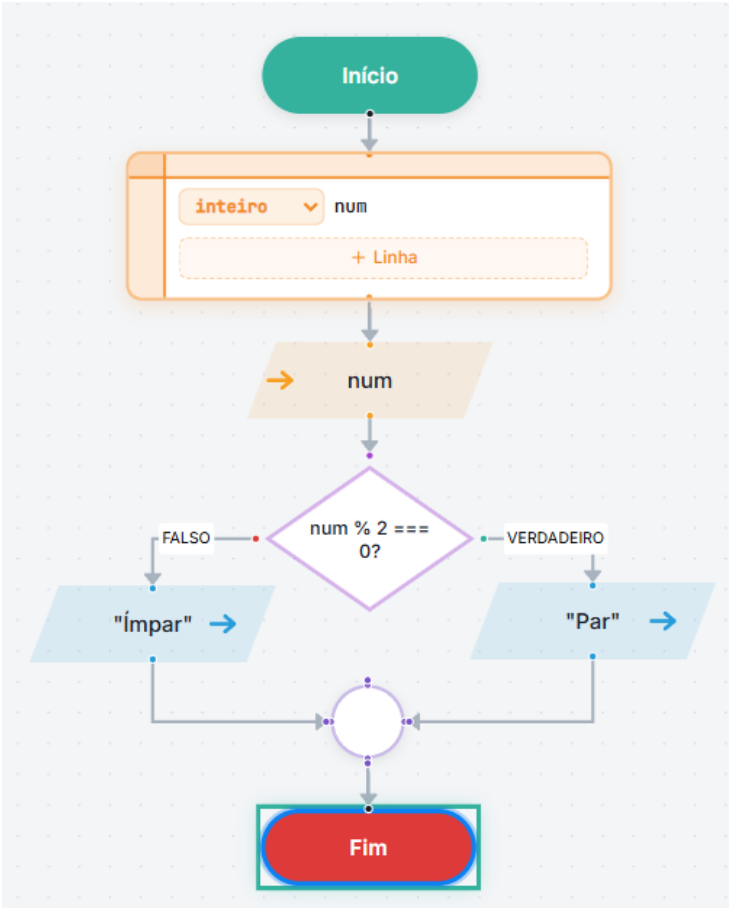


Figura 12 – Diagrama de blocos do algoritmo SE composto para verificação de número par ou ímpar, com destaque para o fluxo de execução e a estrutura lógica do processo. Fonte: Elaborado pelo autor.

A Figura 13 evidencia o teste de mesa, que constitui uma das principais contribuições desta etapa do trabalho. Nela, é possível acompanhar a execução passo a passo do algoritmo e estado das variáveis declaradas, com a identificação do estágio atual de processamento, além da visualização dos valores das variáveis e dos resultados intermediários e finais obtidos durante a simulação. Destaca-se o passo 7, que representa o término da execução do algoritmo conforme o diagrama da Figura 12.

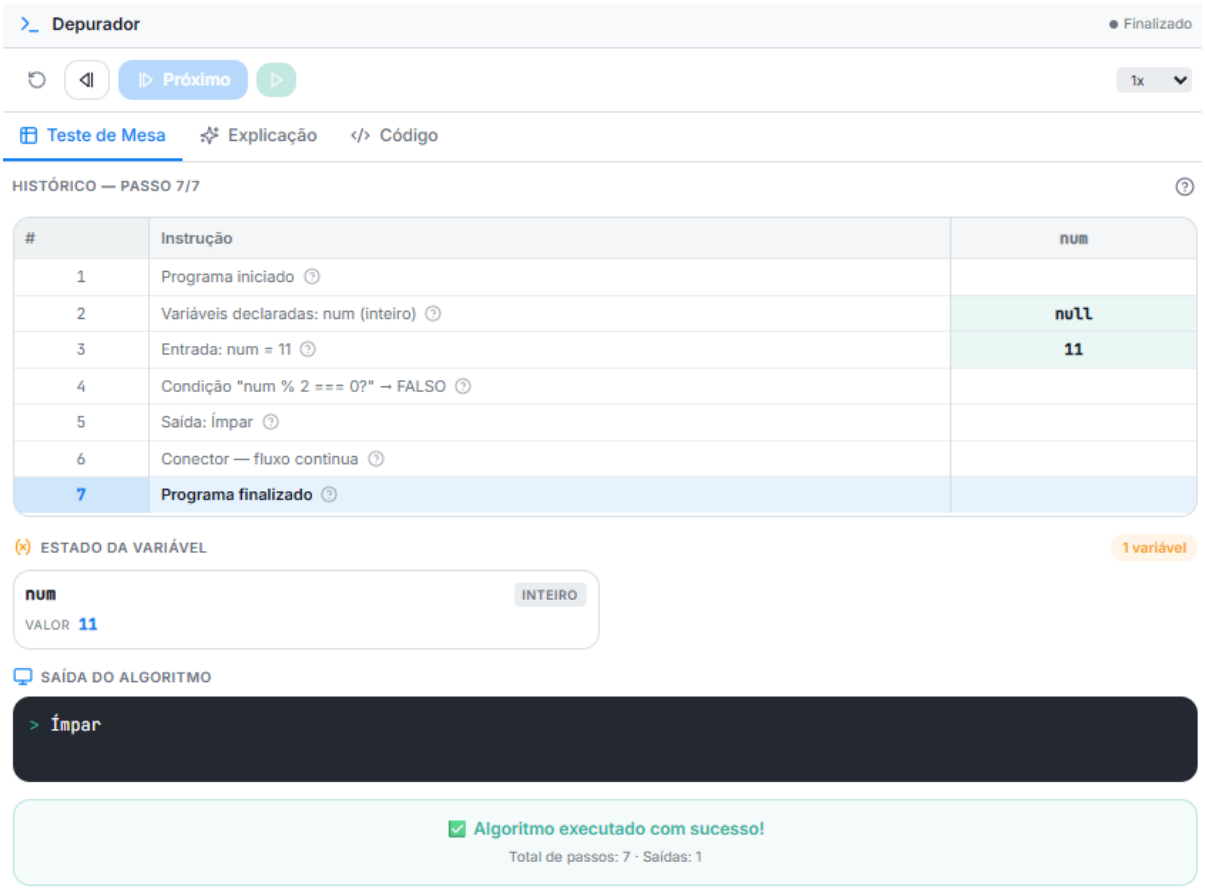


Figura 13 – Teste de mesa do algoritmo SE composto para verificação de número par ou ímpar, com acompanhamento passo a passo da execução e evolução das variáveis.
Fonte: Elaborado pelo autor.

A Figura 14 apresenta a etapa de explicação da execução do algoritmo, auxiliando na compreensão do estado atual do processamento e das transições entre as etapas do fluxo.

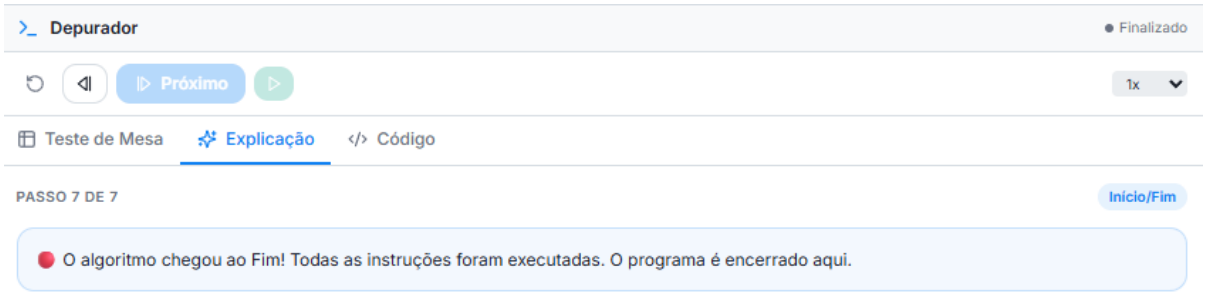


Figura 14 – Etapa de explicação da execução do algoritmo SE composto para verificação de número par ou ímpar, destacando o estado atual do processamento.
Fonte: Elaborado pelo autor.

Por fim, a Figura 15 apresenta a implementação do algoritmo em JavaScript, permitindo a análise direta do código-fonte e sua relação com a modelagem apresentada no diagrama de blocos.

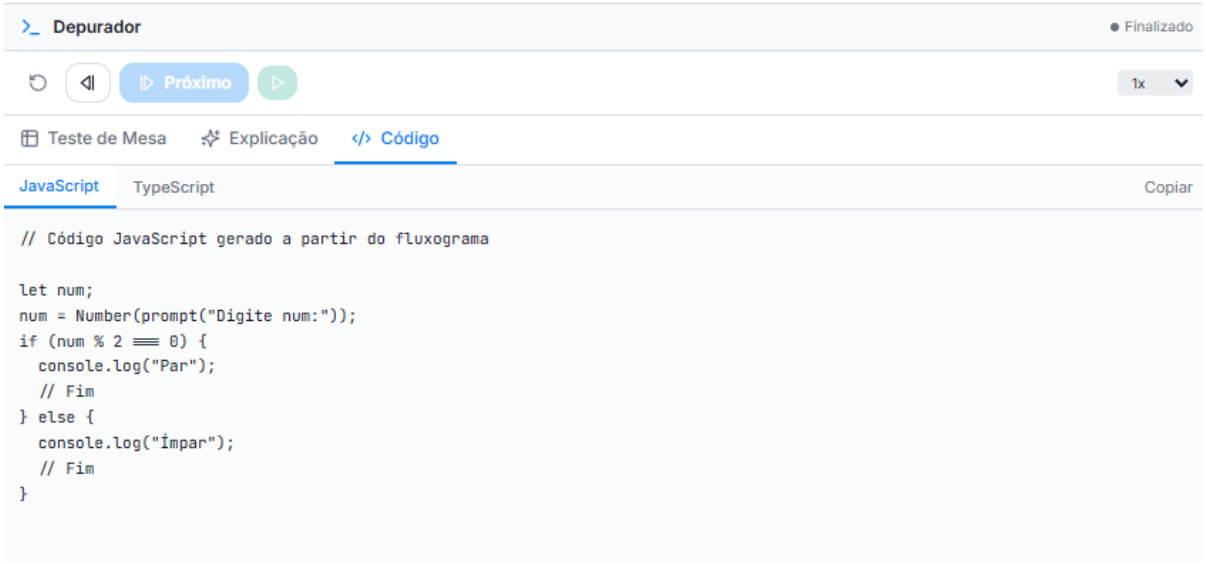


Figura 15 – Implementação em JavaScript do algoritmo SE composto (par ou ímpar), com comparação ao diagrama de blocos.
Fonte: Elaborado pelo autor.

5 CONSIDERAÇÕES FINAIS

5.1 Considerações Finais

Este trabalho apresenta a proposta de desenvolvimento de um sistema web voltado à construção e execução passo a passo de algoritmos representados por meio de diagramas de blocos, contendo um módulo para teste de mesa baseado no diagrama produzido. A proposta tem como foco principal apoiar o processo de aprendizagem de lógica de programação, permitindo a visualização clara da execução dos algoritmos.

Durante o desenvolvimento da proposta, foram definidos os requisitos funcionais do sistema, modelados os fluxos lógicos dos algoritmos e elaborados protótipos de interface que representam a experiência de uso da aplicação. Além disso, foi especificado o módulo de teste de mesa, que permite a simulação passo a passo da execução dos algoritmos, possibilitando o acompanhamento das variáveis e dos resultados intermediários.

Os resultados preliminares indicam que a abordagem proposta possui potencial para contribuir com a compreensão do comportamento dos algoritmos, especialmente no contexto educacional, ao aproximar a teoria da execução prática de forma visual e interativa.

Como continuidade deste trabalho, pretende-se realizar estudos relacionados a grafos, compiladores, interpretadores, máquinas de estado e análise simbólica, visando ampliar as capacidades do sistema e auxiliar no desenvolvimento do módulo de teste de mesa. Após o levantamento e análise dessas tecnologias, serão desenvolvidos os primeiros protótipos funcionais do sistema, com foco na execução passo a passo dos algoritmos e no acompanhamento das variáveis durante o processo de simulação.

REFERÊNCIAS

ANDRADE, E. O. **Desenvolvimento de uma Ferramenta para o Ensino de Lógica de Programação por Meio de Diagramas de Blocos**. 2026. Trabalho de Conclusão de Curso em desenvolvimento, Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná – Câmpus Guarapuava.

ARAÚJO, L. F. **Lógica de Programação**. Curitiba: Intersaberes, 2022.

BOWER, M. *et al.* Improving the computational thinking pedagogical capabilities of school teachers. **Australian Journal of Teacher Education**, v. 42, p. 53–72, 2017. Disponível em: https://api.semanticscholar.org/CorpusID:151346430.

COOK, D. **Flowgorithm - Visual Programming Language**. 2023. https://flowgorithm.org. Acesso em: 25 jan. 2026.

CREWS, T.; ZIEGLER, U. **The Flowchart Interpreter for Introductory Programming Courses**. Bowling Green, 1998.

Docker, Inc. **Docker Documentation**. 2024. https://docs.docker.com. Acesso em: 27 fev. 2026.

FORBELLONE, A. L. V.; EBERSPäCHER, H. F. **Lógica de Programação: a construção de algoritmos e estruturas de dados**. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

Git Project. **Git**. 2026. https://git-scm.com. Acesso em: 19 maio 2026.

GitHub, Inc. **GitHub Documentation**. 2024. https://docs.github.com. Acesso em: 09 fev. 2026.

Google LLC. **Blockly: A Visual Programming Editor**. 2024. https://developers.google.com/blockly. Acesso em: 19 fev. 2026.

Lovable. **Lovable: AI-powered platform for generating applications from prompts**. 2026. Accessed: 2026-05-01. Disponível em: https://lovable.dev.

Lucid Software Inc. **Lucidchart Visual Workspace**. 2024. https://www.lucidchart.com. Acesso em: 30 jan. 2026.

MANZANO, J. A. N. G. **Algoritmos: Lógica para Desenvolvimento de Programação e Computação**. São Paulo: Érica, 2010. Conteúdo utilizado da página 11.

MCDOWELL, C. E.; HELMBOLD, D. P. Debugging concurrent programs. **ACM Computing Surveys**, v. 21, n. 4, p. 593–622, dez. 1989.

Meta Platforms, Inc. **React: A JavaScript library for building user interfaces**. 2024. https://react.dev. Acesso em: 16 mar. 2026.

MIT Media Lab. **Scratch - Imagine, Program, Share**. 2024. https://scratch.mit.edu. Acesso em: 07 mar. 2026.

Portugol Studio. **Portugol Studio - Ambiente de Programação Educacional**. 2023. https://portugol.dev. Acesso em: 11 fev. 2026.

PostgreSQL Global Development Group. **PostgreSQL Documentation**. 2024. https://www.postgresql.org. Acesso em: 22 fev. 2026.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: uma abordagem profissional**. 8. ed. Porto Alegre: McGraw-Hill, 2016.

React Flow Team. **React Flow Documentation**. 2026. https://reactflow.dev/. Acesso em: 19 maio 2026.

Ruby Core Team. **Ruby Programming Language**. 2023. https://www.ruby-lang.org. Acesso em: 12 fev. 2026.

Ruby on Rails Core Team. **Ruby on Rails Framework**. 2024. https://rubyonrails.org. Acesso em: 05 mar. 2026.

SILVA, A. R. d.; SOUZA, M. V. d. O construcionismo de seymour papert e suas contribuições para a educação. **Cadernos de Educação**, v. 17, n. 34, p. 1–15, 2018. Disponível em: https://revistas.fucamp.edu.br/index.php/cadernos/article/view/2820/1766.

SUN, Y.; LIN, J.; WU, Y. Block-based versus text-based programming: A comparison of learners' programming behaviors, computational thinking skills, and attitudes toward programming. **Computers & Education**, Elsevier, v. 200, p. 104789, 2024. ISSN 0360-1315.

WING, J. M. Pensamento computacional. **Revista Brasileira de Ensino de Ciência e Tecnologia**, v. 9, n. 2, p. 1–10, 2016. Tradução do artigo original "Computational Thinking" (2006). Disponível em: https://periodicos.utfpr.edu.br/rbect/article/view/4711/pdf.

XAVIER, G. F. C. **Lógica de programação**. [*S.l.*]: Senac, 2018.