UNIVERSIDADE FEDERAL DO PAMPA

MARINA SILVA

PROPOSTA DE UMA LINGUAGEM COMPOSICIONAL VISUAL PARA ENSINO DE PROGRAMAÇÃO

Bagé
2021

MARINA SILVA

PROPOSTA DE UMA LINGUAGEM COMPOSICIONAL VISUAL PARA ENSINO DE PROGRAMAÇÃO

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientadora: Ana Paula Lüdtke Ferreira

Bagé
2021

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

S586 Silva, Marina

Proposta de uma linguagem composicional visual para ensino de programação / Marina Silva.

69 f.: il.

Orientadora: Ana Paula Lüdtke Ferreira

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pampa, Engenharia de Computação, 2021.

1. Linguagens de programação. 2. Linguagens visuais. 3. Ensino de algoritmos. 4. Educação em Computação. I. Título.

SERVIÇO PÚBLICO FEDERAL

MINISTÉRIO DA EDUCAÇÃO

Universidade Federal do Pampa

MARINA SILVA DA SILVA

PROPOSTA DE UMA LINGUAGEM COMPOSICIONAL VISUAL PARA ENSINO DE PROGRAMAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 2 de outubro de 2021.

Banca examinadora:

_____

Prof. ª Dra. Ana Paula Lüdtke Ferreira

Orientadora

Unipampa

_____

Prof. ª Dra. Sandra Dutra Piovesan

Unipampa

_____

Prof. Dr. Sandro da Silva Camargo

Unipampa

Assinado eletronicamente por **SANDRA DUTRA PIOVESAN**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 21/09/2022, às 10:55, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **SANDRO DA SILVA CAMARGO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 21/09/2022, às 11:29, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **ANA PAULA LUDTKE FERREIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 21/09/2022, às 17:49, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0932233** e o código CRC **57C15666**.

Referência: Processo nº 23100.017451/2021-96 SEI nº 0932233

AGRADECIMENTO

À professora Ana, que me inspirou e me guiou por todo esse trabalho. Ao professor Leandro Camargo e ao professor André Mello, que me fizeram gostar de programação. A mamãe e ao papai, que acreditaram em mim. E a Letícia e ao Gabriel que não me deixaram parar.

RESUMO

Os primeiros semestres dos cursos de graduação na área de Computação tendem a apresentar maiores índices de reprovação e de retenção de alunos. Disciplinas de introdução à programação contribuem para esses índices. Uma das dificuldades enfrentadas pelos alunos nessas disciplinas é o foco em aspectos operacionais da solução de problemas, associado às linguagens que seguem o paradigma imperativo, gerando um excessivo esforço na correção de erros de sintaxe. Este trabalho busca apresentar a linguagem Pandora, uma proposta de linguagem funcional visual composicional voltada para o ensino de programação, focada nos aspectos semânticos da resolução de problemas e na composição de soluções a partir de soluções previamente elaboradas. A linguagem foi definida formalmente utilizando uma gramática de grafos e possui foco em facilitar o reuso de código. Pandora é composta por blocos coloridos com símbolos identificadores, o que a torna acessível a estudantes de diversas faixas etárias. A linguagem pode ser utilizada para o ensino de lógica de programação, algoritmos e linguagens funcionais, além de poder ser usada no ensino de Matemática, com suporte computacional ou com atividades desplugadas.

Palavras-chave: Linguagens de programação. Linguagens visuais. Ensino de algoritmos. Educação em Computação.

ABSTRACT

The first semesters of undergraduate courses in the field of Computing tend to have higher failure and student retention rates. Introductory programming courses contribute to these indexes. One of the difficulties students face in these disciplines is the excessive focus on operational aspects of problem-solving, associated with languages that follow the imperative paradigm, generating an exaggerated effort to correct syntax errors. This work seeks to present the language Pandora, a proposal for a functional visual compositional language aimed at teaching programming, focused on the semantic aspects of problem-solving and on the composition of solutions from previously elaborated solutions. The language was formally defined using a typed graph grammar and focuses on facilitating code reuse. Pandora is composed of coloured blocks with identifying symbols, making it accessible to students of different age groups. The language intends to facilitate teaching programming logic, algorithms, functional languages, in addition to being used for teaching Mathematics, with computational support or with unplugged activities.

Keywords: Programming languages, Visual programming languages, Algorithms teaching, Computing education.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ACM	Association for Computing Machinery
IEEE	Institute of Electrical and Electronics Engineers
UNIPAMPA	Universidade Federal do Pampa
MIT	Massachusetts Institute of Technology
RFID	Radio-Frequency Identification
BEESM	Block-based End-user programming tool for Smart Environments

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a introdução do trabalho, com a motivação e justificativa descritas na Seção 1.1 e os objetivos do trabalho apresentados na Seção 1.2.

1.1 Ensino de algoritmos e programação

O primeiro semestre dos cursos de graduação na área de Computação apresenta altos índices de retenção. Esse fato é atestado pelos dados do Censo da Educação Superior no Brasil[1] e pela quantidade de trabalhos de pesquisa publicados em conferências e periódicos dedicados ao ensino de Engenharia e Computação, tanto no Brasil quanto no exterior.

As causas para a retenção dos discentes no primeiro ano do curso são frequentemente atribuídas a uma suposta "falta de base" de conhecimentos proveniente do Ensino Médio e Fundamental. Essa justificativa pode parecer verdadeira, mas não é suportada pelos dados de desempenho discente.

Em primeiro lugar, alunos que são aprovados com notas no ENEM bastante altas apresentam as mesmas dificuldades com relação às disciplinas de programação do que os alunos que ingressam com notas inferiores. A causa dessa falta de diferenciação pode ser explicada pela diferença nas competências exigidas dos discentes do ensino superior – na forma de conteúdos, habilidades e atitudes – em relação às exigidas no Ensino Médio. O tipo de estudo necessário, o tipo de atividade exigida, o tipo de raciocínio demandado, a forma de apresentação dos resultados encontrados, entre várias outras características do trabalho acadêmico, são fundamentalmente diferentes do esquema de memorização de conteúdos utilizado na maioria das escolas.

As dificuldades encontradas, mesmo para aqueles que eram considerados ótimos alunos no Ensino Médio leva à frustração e ao desapontamento com o curso, o que provoca a também alta taxa de evasão no primeiro semestre. Conteúdos novos, como lógica de programação, que não são vistos no Ensino Básico, possuem os mesmos índices de reprovação das demais disciplinas, o que parece indicar que o problema de fato é a localização do componente no primeiro semestre do curso. Como não é possível eliminar a existência de conteúdos no primeiro semestre, outra abordagem é necessária.

[1] https://www.gov.br/inep/pt-br/areas-de-atuacao/pesquisas-estatisticas-e-indicadores/censo-da-educacao-superior/resultados.

O artigo *Computational Thinking* (WING, 2008) é um dos marcos da discussão sobre ensino de Computação, estabelecendo um conjunto de habilidades e atitudes consideradas fundamentais para todas as pessoas, não somente para profissionais de Computação. O foco desse arcabouço é a solução de problemas, necessária em todas as áreas. Em seu artigo, Wing estabelece diretrizes gerais para o ensino que levam em conta um conjunto de competências a desenvolver para que o objetivo de "pensar computacionalmente" seja atingido.

Em um artigo posterior (WING, 2014), Wing apresenta a definição de Pensamento Computacional como "os processos mentais envolvidos na formulação e na expressão da solução de um problema, de forma que tanto pessoas como máquinas possam resolvê-lo". A habilidade essencial para a mobilização desses processos é a abstração, usada no reconhecimento e definição de padrões, generalização a partir de instâncias específicas e parametrização para adaptação de soluções.

Essas habilidades necessárias ao aprendizado de programação podem ser desenvolvidas ainda durante o ensino básico utilizando conceitos relacionados ao pensamento computacional e até o próprio ensino de algoritmos e lógica de programação. Segundo (WING, 2008) "se quisermos garantir uma base sólida de entendimento e aplicação do pensamento computacional para todos, então esse aprendizado deve ser construído desde os primeiros anos da infância."

No Brasil, a programação não faz parte do currículo do Ensino Básico. Sendo assim, os estudantes chegam ao ensino superior sem ideias sobre o que constitui a atividade de programar. As primeiras experiências no ensino superior costumam ser com linguagens de propósito geral fundadas no paradigma imperativo (PIMENTA, 2019). Quando o paradigma de orientação a objetos é usado, quase sempre a linguagem de programação ensinada é Java.

A complexidade das linguagens escolhidas e a forma como se representam os algoritmos em linguagens com fluxo de execução imperativo faz com que os métodos de ensino direcionem o foco (tanto do professor quanto do aluno) para a sintaxe da linguagem de programação estudada, em vez de destacar o processo de resolução de problemas por meio da programação. Além disso, o alto nível de abstração necessário para a resolução dos problemas apresentados é uma competência que não foi desenvolvida anteriormente com os alunos (GOMES; MENDES, 2007) fazendo com que a atividade de programação acabe com o foco na retirada de erros de compilação e não no pensar a solução de problemas.

A literatura na área de ensino de programação apresenta diversas buscas de soluções para os problemas levantados, utilizadas como forma de facilitar o ensino de programação. Entre essas abordagens estão jogos, ferramentas lúdicas e robótica educacional.

Dentre essas ferramentas, destaca-se o uso do *Scratch*, um sistema do *Lifelong Kindergarten Group* do MIT Media Lab composto por um conjunto de blocos de instruções que podem ser arrastados pela tela, combinados e executados por um personagem. O *Scratch* tem sido utilizado em diversas pesquisas relacionadas ao ensino de computação no ensino básico (SOUZA; CASTRO, 2016) e ao pensamento computacional (ZANETTI; BORGES; RICARTE, 2016).

Essa estrutura de programação através de elementos visuais também é utilizada em outras ferramentas como *App Inventor*, também do MIT, que possui foco em programação para dispositivos móveis e no jogo *NoBug's SnackBa*r que tem como proposta a utilização de blocos para construção da solução de um problema específico em cada fase (VAHLDICK; FARAH, 2020). Ainda assim, a organização em blocos não elimina a questão de que a composição dos blocos pode dar mais ênfase na sintaxe do que na semântica da programação.

Uma das questões trazidas pela Engenharia de Software moderna é o reúso de software. O aproveitamento de especificações, modelos e código aumenta a produtividade das equipes de desenvolvimento e facilita os procedimentos de verificação, pelo uso de artefatos já testados, inclusive em ambiente de produção (PRESSMAN, 2016). O reúso efetivo de artefatos de software exige do desenvolvedor um pensamento voltado à composição de artefatos conhecidos para a solução de um problema novo. A habilidade de compor especificações ou código não pode ser desenvolvida quando os estudantes, a cada nova tarefa, são instados a construir sempre uma solução a partir do zero.

Uma das formas de desenvolver habilidades de composição de artefatos de software é ensinar os alunos a programar por meio de uma linguagem composicional, que exija o uso de componentes já desenvolvidos para a construção de uma nova solução. O paradigma funcional de programação possui essa característica (SEBESTA, 2018), ainda que as linguagens funcionais possuam sintaxes que não favorecem o ensino. Cita-se, como exemplos, a linguagem LISP (TOURETZKY, 1990) ou a linguagem Haskell (THOMPSON, 1999).

Abordagens orientadas a objeto favorecem a composicionalidade, mas ainda assim, quando usam um fluxo de controle imperativo, apresentam os problemas de foco

na sintaxe, já mencionados. Abordagens orientadas a objetos com fluxo de execução funcional, como a linguagem CLOS (KEENE, 1989), facilitam a programação de estruturas de dados complexas, mantendo as vantagens da composicionalidade, ainda que com a sintaxe da linguagem LISP, com parênteses que tornam a leitura do código bastante difícil.

A proposta a ser desenvolvida, a partir da execução deste trabalho, é unir as vantagens da composicionalidade associada à programação funcional, com a estruturação em objetos, herança e polimorfismo de subclasses provida pelo paradigma da orientação a objetos, em uma linguagem visual que possa ser usada por iniciantes com foco na semântica da resolução de problemas, movendo o foco do processo de compilação para o processo de execução de código. Espera-se, com isso, que os estudantes habituem-se a pensar em termos da semântica da resolução de problemas, reúso de especificações, mesmo quando comecem a trabalhar com linguagens textuais usadas na indústria de software.

1.2 Objetivos e organização do trabalho

O objetivo deste trabalho é apresentar uma proposta de linguagem visual composicional para introdução à programação de computadores, focada nos aspectos semânticos da resolução de problemas.

São objetivos específicos deste trabalho:

1. Revisar a literatura sobre ferramentas para ensino de programação, produzindo uma análise crítica sobre seu foco na sintaxe usada e na semântica resultante.
2. Investigar as implementações do Pensamento Computacional em linguagens e artefatos para ensino de programação.
3. Revelar os paradigmas mais efetivos para ensino de programação, com base da literatura.
4. Definir sintaticamente a linguagem proposta e sua semântica (semi)formal, de maneira que ela possa ser implementada em um trabalho futuro.

São resultados esperados deste trabalho e de sua continuidade:

- Aumentar o interesse dos estudantes do Ensino Básico pela área de Computação, por meio de uma estratégia de ensino de programação mais lúdica e interessante.

- Qualificar o ensino de algoritmos e programação no curso de Engenharia de Computação da UNIPAMPA, por meio da descoberta de procedimentos mais efetivos de ensino do tema para estudantes.

O texto está organizado como se segue: no Capítulo 1 é apresentada a introdução do trabalho, com a motivação e justificativa descritas na Seção 1.1 e os objetivos do trabalho apresentados na Seção 1.2. O Capítulo 2 expõe a caracterização do trabalho de pesquisa desenvolvido (Seção 2.1), os procedimentos de revisão da literatura (Seção 2.2) e o ferramental tecnológico usado para o desenvolvimento do trabalho (Seção 2.3). O Capítulo 3 apresenta os trabalhos relacionados selecionados na revisão de literatura. O Capítulo 4 apresenta os conceitos de Linguagens Funcionais (Seção 4.1) e Gramáticas de Grafos (Seção 4.2) utilizados como base para construção da proposta da linguagem Pandora. O Capítulo 5 explicita a proposta da linguagem Pandora (Seção 5.1), sua gramática (Seção 5.2) e alguns exemplos de códigos (Seção 5.3). O Capítulo 6 apresenta as considerações finais.

2 MATERIAL E MÉTODOS

Este capítulo descreve a caracterização e o planejamento da pesquisa na Seção 2.1, a revisão sistemática da literatura, com as questões de pesquisa e *strings* de busca na Seção 2.2 e o ferramental tecnológico utilizado no trabalho na Seção 2.3.

2.1 Caracterização e planejamento da pesquisa

Este trabalho caracteriza-se como um trabalho exploratório de pesquisa bibliográfica e aplicada. A pesquisa bibliográfica visa instrumentalizar conceitualmente e experimentalmente a condução do processo de desenvolvimento. O entendimento das tecnologias já desenvolvidas para o ensino de programação permite a análise dos aspectos sintáticos e semânticos das linguagens utilizadas e os resultados reportados na literatura podem apontar caminhos que devem ou não ser seguidos. Estão em estudo também as implementações em linguagens do arcabouço do Pensamento Computacional. As hipóteses de pesquisa que estão em investigação neste trabalho são as seguintes:

H1 Os elementos estruturantes do Pensamento Computacional não estão traduzidos em linguagens voltadas ao aprendizado de programação.

H2 O paradigma imperativo gera ferramentas mais focadas em sintaxe do que em semântica de programas.

H3 Linguagens visuais funcionais com estruturação orientada a objetos são um arcabouço mais rico para ensino de programação com vistas à composicionalidade e reúso de software.

As hipóteses de pesquisa levantadas não serão respondidas neste trabalho, visto que sem uma implementação e experimentos válidos, não será possível respondê-las. De toda forma, as hipóteses serão guias do trabalho aqui desenvolvido e dos desdobramentos futuros.

O desenvolvimento do trabalho está organizado a partir das seguintes atividades:

1. Revisão sistemática da literatura sobre linguagens visuais/composicionais para ensino de programação e implementações do pensamento computacional em linguagens de especificação/programação.
2. Proposição de uma linguagem visual com elementos funcionais e que facilite

a construção de soluções com base no arcabouço teórico do pensamento computacional.

3. Definição sintática formal e semântica informal da linguagem proposta.

O teste de um subconjunto da linguagem construída, junto a estudantes dos ensinos superior, médio e fundamental não foi possível, em virtude da pandemia. Como não há ferramenta computacional operante, as atividades teriam que ser feitas de forma deplugada e presencial.

2.2 Revisão sistemática da literatura

A etapa de análise e revisão bibliográfica se realiza de maneira sistemática, sendo composta pelas seguintes passos: (*i*) definição das questões de pesquisa relacionadas à revisão; (*ii*) definição das fontes de pesquisa; (*iii*) definição das *strings* de busca; (*iv*) filtragem dos resultados encontrados; (*v*) classificação dos trabalhos encontrados; (*vi*) revisão e análise dos trabalhos selecionados. A seguir, são descritos os métodos, procedimentos e resultados de cada uma dessas fases.

A revisão da literatura foi organizada sobre as seguintes questões de pesquisa:

Q1 Quais são as ferramentas existentes para ensino de programação baseadas em composição de módulos ou funções?

Q2 Quais são os métodos/linguagens de ensino de programação usados na educação básica?

Q3 Existem implementações de linguagens que usem os princípios do pensamento computacional?

As seguintes palavras-chave foram escolhidas para a busca do referencial usado neste trabalho:

Tabela 1 – Fontes de referências bibliográficas

Bibliotecas e repositórios digitais	ACM Digital Library (https://dl.acm.org/) IEEE Xplorer (https://ieeexplore.ieee.org/) Wiley Online Library (https://onlinelibrary.wiley.com/)
Periódicos	ACM Transactions on Computing Education (https://dl.acm.org/journal/toce)
Anais de eventos	International Computing Education Research Workshop (https://dl.acm.org/conference/icer) Information Technology Education Conference (https://dl.acm.org/conference/ite) Innovation and Technology in Computer Science Education (https://dl.acm.org/conference/iticse) SIGCSE: Computer Science Education (https://dl.acm.org/conference/sigcse) Workshop sobre Educação em Computação (https://sol.sbc.org.br/index.php/wei)

Termo	**Sinônimos**
Ensino	ensino, aprendizado, método, estratégia, learning, teaching, method, strategy
Programação	programação, programa, algoritmo, programming, algorithm
Linguagem	ferramenta, linguagem, sistema, software, tool, language, system
Pensamento computacional	"pensamento computacional", "computational thinking"
Propriedades	composicional, modular, visual, compositional
Nível de ensino	"ensino fundamental", "ensino médio", "ensino básico", "high school", "middle school"

As *strings* de busca nas fontes de pesquisa foram construídas por meio da conjunção de cláusulas que contenham a disjunção dos sinônimos em seu interior. As fontes de referências bibliográficas utilizadas são apresentadas na Tabela 1.

A primeira fonte de pesquisa utilizada foi a biblioteca ACM Digital Library, a *string* de busca utilizada foi *learning OR teaching OR method OR strategy AND programming OR algorithm AND software OR tool OR language OR system AND "computational thinking" AND visual OR compositional AND "high school" OR "middle school"* buscando referenciais por título e resumo. A partir dessa *string* se obteve 21.893 resultados referentes a artigos publicados nos últimos cinco anos. Buscando reduzir o número de resultados da busca foram observados quais artigos traziam assuntos

não relacionados diretamente ao trabalho foi adicionada a palavra *programming* antes de *learning, teaching, method e strategy* em cada um dos termos. Também foram adicionados operadores NOT para os termos *"c programming", java e reading*. Após essas alterações, o número de resultados foi reduzido para 61, no entanto ainda haviam publicações com temas divergentes. Assim a *string* foi modificada, utilizando-se a *string "programming learning" OR "programming teaching" OR "programming tool" OR "programming language" AND algorithm OR software AND "computational thinking" AND visual OR compositional AND "high school" OR "middle school" NOT "c programming" NOT java NOT reading* para o título e a *string "programming learning" OR "programming teaching" OR "programming tool" OR "programming language" AND algorithm OR software AND "computational thinking" AND visual OR compositional NOT "c programming" NOT java NOT reading* para o resumo, obtendo-se onze resultados. Desses, 6 resultados responderam às questões de pesquisa e foram incluídos no referencial teórico.

Na ferramenta Google Scholar foi utilizada inicialmente a mesma *string* utilizada anteriormente na busca por título na biblioteca ACM Digital Library, com filtro para o período de 2016 a 2021. Obteve-se 43 resultados. Após algumas modificações, percebeu-se que a ferramenta possui algumas restrições e parece retornar melhores resultados com *strings* mais curtas. Utilizando a *string "programming learning"OR "programming teaching" OR "programming language" OR "computational thinking" AND "high school" OR "middle school" NOT "c programming" NOT java* obteve-se catorze resultados e com a *string "linguagem de programação" OR "ensino de programação" AND "pensamento computacional" AND "ensino médio" OR "ensino fundamental" NOT "linguagem c" NOT java NOT desplugada NOT segurança* o retorno foi de 85 resultados. No entanto, muitos desses resultados se mostraram não relevantes para o trabalho. Assim, foram priorizadas as buscas em outras bases de dados.

Na biblioteca IEEE Xplorer, inicialmente foi utilizado o modo de busca Advanced Search, no entanto muitos dos resultados não possuíam relação com o trabalho, de modo que preferiu-se utilizar o modo de busca Command Search, com a busca pelos referenciais por título. A primeira *string* utilizada foi *"programming learning" OR "programming teaching" OR "programming tool" AND "programming language" AND "computational thinking" AND visual OR compositional AND "high school" OR "middle school" NOT Python NOT Java* que retornou 136 resultados publicados nos últimos cinco anos. No entanto, poucos resultados relevantes foram encontrados. Após algumas alterações, foram

identificados alguns erros na sintaxe da *string* e ela foi alterada de modo a manter os operadores NOT no início da *string* e utilizar dos parênteses quando necessário, por fim após essas verificações a *string NOT "c programming" NOT python NOT java AND "programming learning" OR "programming teaching" OR "programming tool" AND "programming language" AND "computational thinking" AND visual OR compositional AND ("high school" OR "middle school")* retornou 66 resultados. Desses, 24 resultados responderam às perguntas de pesquisa e foram adicionados ao referencial teórico.

Nos anais do Workshop sobre Educação em Computação (WEI) foi utilizada a *string* ensino de programação *OR* aprendizado de programação *AND* ferramenta *OR* linguagem que retornou 57 resultados. Desses catorze foram adicionados ao referencial teórico.

Nos anais do Workshop de Informática na Escola (WIE) foi utilizada a *string* ensino de programação *OR* aprendizado de programação *AND* ferramenta *OR* linguagem que retornou 53 resultados. Desses dezenove foram acrescentados ao referencial teórico.

Na busca no periódico *ACM Transactions on Computing Education* foi utilizada inicialmente a *string 'programming learning' OR 'programming teaching' OR 'programming tool' OR 'programming language' AND algorithm OR software AND 'computational thinking' AND visual OR compositional AND 'high school' OR 'middle school'* que retornou 78 resultados referentes aos últimos cinco anos, após algumas alterações utilizou-se a *string 'programming learning' OR 'programming teaching' OR 'programming tool' OR 'programming language' AND algorithm OR software AND 'computational thinking' AND visual OR compositional AND 'high school' OR 'middle school' NOT woman NOT java* que retornou 16 resultados. Desses, quatro foram adicionados ao referencial teórico.

A prioridade de análise foi definida para os trabalhos publicados nos últimos cinco anos. Os critérios de inclusão do trabalho na revisão são os seguintes:

- Trabalhos com propostas de linguagens específicas para o ensino de programação em todas as fases do ensino básico e no início do ensino superior. Não serão considerados artigos que trabalhem com linguagens textuais de propósito geral. Esses últimos somente serão contabilizados para propósitos estatísticos.
- Trabalhos que usem linguagens específicas para o ensino de programação, desde que analisem os resultados do aprendizado com metodologia de análise de dados quantitativos e mensuráveis.
- Trabalhos que usem o arcabouço do pensamento computacional para o ensino de

programação.

No terceiro passo foi realizada a leitura dos resumos, introduções, conclusões e referências dos trabalhos selecionados com a finalidade de identificar a relevância dos trabalhos para a realização pesquisa, como também para identificar potenciais trabalhos relevantes que não foram encontrados pelos métodos de pesquisa utilizados. O quarto passo compreendeu a classificação dos trabalhos relevantes, em que foi atribuída uma nota de 1 a 5 conforme a relevância e similaridade do trabalho encontrado com a pesquisa em desenvolvimento, da seguinte forma: 5 - trabalhos que abordam o mesmo problema, com os mesmos objetivos; 4 - trabalhos que abordam o mesmo problema, com objetivos similares; 3 - trabalhos que abordam um problema similar, com objetivos similares; 2 - trabalhos que abordam um problema similar, com objetivos distintos; 1 - trabalhos sem relação com o problema ou com os objetivos.

Por fim, no quinto e último passo foi realizada uma análise abrangente dos trabalhos, por ordem decrescente de relevância. Na análise dos trabalhos, novas fontes puderam ser identificadas e foram incorporadas ao método. O processo de busca terminou quando nenhum novo trabalho relacionado com índice de relevância maior que 3 foi encontrado no processo.

2.3 Ferramental tecnológico

O texto do trabalho foi desenvolvido em LaTeX usando o modelo unipampa.cls, disponibilizado no site do curso. Como suporte tecnológico para a escrita, foi usada a plataforma de escrita colaborativa Overleaf (http://overleaf.com).

Para a construção dos símbolos da linguagem foram utilizados o Google Apresentações, o Canva e o Programa de manipulação de imagem do GNU (GIMP). O Google Apresentações faz parte do conjunto de ferramentas *Google Workspace* (workspace.google.com/intl/pt-BR/products/slides/) com foco em construção de apresentações em slide. Um dos recursos oferecidos pela ferramenta é a possibilidade de adicionar formas e diagramas às apresentações. Esse recurso foi utilizado para a construção dos rascunhos iniciais dos blocos da linguagem. O canva (www.canva.com) é uma ferramenta de design com um conjunto de templates que facilita a criação de material visual. Essa ferramenta foi utilizada para a construção dos blocos da linguagem utilizando formas simples e texto na fonte *Schoolbell*. Por fim, foi utilizado o GIMP

(gimp.org), ferramenta de edição de imagens para a construção da gramática e dos códigos em Pandora.

A ferramenta Figma (figma.com) foi utilizada para a prototipação das telas de uma ferramenta Pandora.

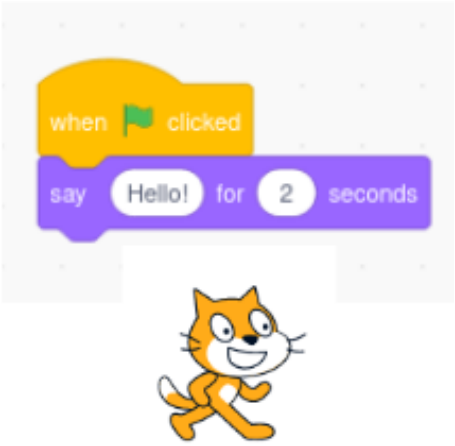
3 TRABALHOS RELACIONADOS

Esse capítulo descreve as principais ferramentas e linguagens de programação com foco em ensino encontradas na literatura. Na Seção 3.1 é feita a descrição da linguagem de programação visual Scratch e outras ferramentas derivadas, na Seção 3.2 são apresentadas as ferramentas Lego Mindstorms e Lego Education We Do, na Seção 3.3 é descrita uma série de ferramentas tangíveis, na Seção 3.4 são apresentados alguns jogos voltados para o ensino de programação, na Seção 3.5 são descritas algumas ferramentas visuais que não seguem os eixos observados nas seções anteriores e por fim, na Seção 3.6 são apresentados os resultados obtidos a partir da revisão sistemática da literatura realizada no trabalho.

3.1 Scratch e linguagens baseadas em Scratch

Dentre as linguagens de programação visuais com foco em ensino encontradas na literatura destaca-se pela sua comunidade de usuários o *Scratch* (scratch.mit.edu/statistics), ferramenta do MIT com sintaxe baseada em blocos. Os blocos da linguagem Scratch representam diferentes instruções que serão executadas por um personagem animado. Possuem diferentes cores de acordo com o tipo de instrução (movimento, aparência, controle, eventos, som, operadores, variáveis, etc) e são conectados por meio de uma função de arrastar e soltar para formar um programa.

Figura 1 – Blocos e personagem da ferramenta Scratch



Fonte: (scratch.mit.edu)

Além de amplamente utilizada, a linguagem de blocos do Scratch serve como base

para outras ferramentas desenvolvidas com foco em ensino. A BEESM (Block-based End-user programming tool for Smart Environments) (SERAJ; AUTEXIER; JANSSEN, 2018) é uma ferramenta de programação visual em ambiente web que pode ser utilizada para programação de robôs, microcontroladores e ambientes inteligentes. Ela emprega a linguagem de blocos baseada em Scratch, além da possibilidade de visualizar esse código em linguagem PHP e na linguagem de programação do Arduino. Segundo os autores essa abordagem de blocos reduz erros de sintaxe e facilita a manipulação das estruturas de código. Outro fator muito interessante a respeito da ferramenta é a facilidade que o usuário possui em prototipar seus projetos.

O ambiente de programação visual em blocos da BEESM é construído utilizando Google Blockly, uma biblioteca open source para adicionar código baseado em blocos em uma aplicação. A biblioteca Google Blockly facilita a construção de linguagens em blocos, pois fornece uma gramática e uma representação para programação que os desenvolvedores podem usar em seus aplicativos (PASTERNAK; FENICHEL; MARSHALL, 2017). Os blocos disponibilizados pela Google Blockly são visualmente semelhantes aos blocos do Scratch.

Outra ferramenta utilizada para o ensino de programação com blocos baseados em Scratch é o App Inventor, ferramenta lançada pela Google e atualmente mantida pelo MIT, que permite o desenvolvimento de aplicações Android. O App Inventor possui blocos para a programação de funções específicas de dispositivos móveis, além de uma tela Design para construção da parte visual da aplicação, o que facilita a criação de aplicativos Android por iniciantes na programação.

Além da BEESM e do App Inventor pode-se mencionar uma série de soluções que utilizam o Scratch para a programação de dispositivos com propósito educacional como Arduino, Lego Mindstorms e Raspberry Pi. Dentre esses trabalhos encontram-se as ferramentas Make Code, Scratch4Arduino e Modkit que tornam possível a utilização do Scratch para a programação com Arduino e as linguagens DuinoBlocks for Kids, Lofi Blocks e Sucre4Kids que possuem estruturas de blocos coloridos semelhantes ao Scratch e também são voltadas para a programação de placas Arduino. As ferramentas Enchanting e ScratchGPIO também utilizam o Scratch, mas para programação de Lego Mindstorms NXT e Raspberry Pi, respectivamente.

Outra ferramenta utilizada em alguns trabalhos foi plataforma Hora do Código do site Code.org que consiste em diversos jogos e tutoriais temáticos que utilizam principalmente a linguagem Scratch e linguagens baseadas em Scratch com o objetivo

de ensinar programação para crianças. Outra aplicação da linguagem Scratch observada em alguns trabalhos foi o uso em atividades voltadas para alunos de ensino fundamental e médio com o objetivo de trabalhar conceitos relacionados às disciplinas de matemática e ciências, bem como diversas atividades voltadas para o incentivo do pensamento computacional e das habilidades relacionadas.

3.2 Kits Lego Mindstorms e Lego Education We Do

Outros exemplos de ferramentas utilizadas no ensino de programação, principalmente com foco em alunos do ensino fundamental e médio, são os kits de robótica da Lego (PINTO-LLORENTE et al., 2016): Mindstorms EV3 e Education WeDo. Até 2019, ambos os kits utilizavam a linguagem de blocos LabView composta por blocos quadrados conectados na horizontal representando as funções a serem executadas pelos componentes do kit (motores, sensores e tela).

O kit Lego Education WeDo, que tem direcionamento para alunos das séries iniciais do ensino fundamental, possui blocos mais coloridos, símbolos maiores e com menos detalhes em relação à linguagem utilizada pelo kit Mindstorms EV3, o que facilita a diferenciação dos blocos. Já o kit Mindstorms EV3 (bem como seu antecessor NXT) possui blocos com mais informações, permitindo alterações como direção ou número de giros de um motor. A partir de 2019, o kit Mindstorms EV3 passou por uma alteração no software da ferramenta e começou a utilizar a linguagem Scratch para a programação dos projetos. Apesar das diferenças entre as linguagens é possível construir códigos equivalentes aos desenvolvidos com a linguagem anterior.

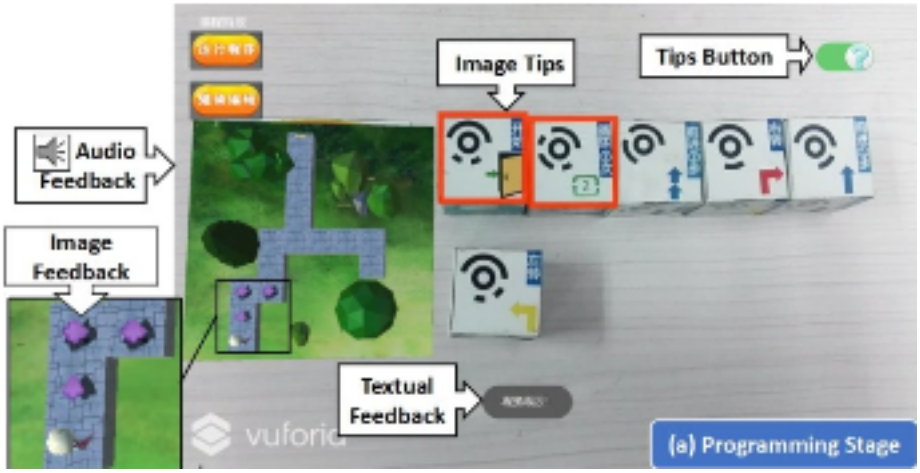
3.3 Linguagens de programação tangíveis

Além das linguagens mencionadas, várias iniciativas buscam criar ferramentas e linguagens acessíveis para diversos níveis de ensino. Uma tendência observada na literatura recente foi o desenvolvimento de ferramentas tangíveis, principalmente visando alunos das séries iniciais do ensino fundamental e da educação infantil. Segundo (HORN; JACOB, 2007), linguagens de programação tangíveis são similares às linguagens de programação baseadas em texto ou visuais, no entanto em vez de utilizar imagens e palavras na tela do computador, linguagens tangíveis utilizam objetos físicos para

representar vários elementos de programação, comandos e estruturas de controle de fluxo.

Ferramentas como os blocos tangíveis desenvolvidos por (HATTORI; HIRAI, 2019), ou a ARCat (DENG et al., 2019) e a Ar-maze (JIN et al., 2018) aliam artefatos tangíveis e realidade aumentada como forma de tornar a ferramenta acessível a alunos mais jovens e facilitar o trabalho em equipe. Os softwares mencionados leem e interpretam os códigos construídos pelos usuários através da câmera de um smartphone ou tablet e executam o algoritmo utilizando realidade aumentada. Esse tipo de ferramenta tem um grande potencial de auxiliar na motivação dos alunos em aprender programação, além de eliminar a necessidade das escolas de possuírem laboratórios de informática com um computador por aluno.

Figura 2 – Imagem da ferramenta Ar-maze



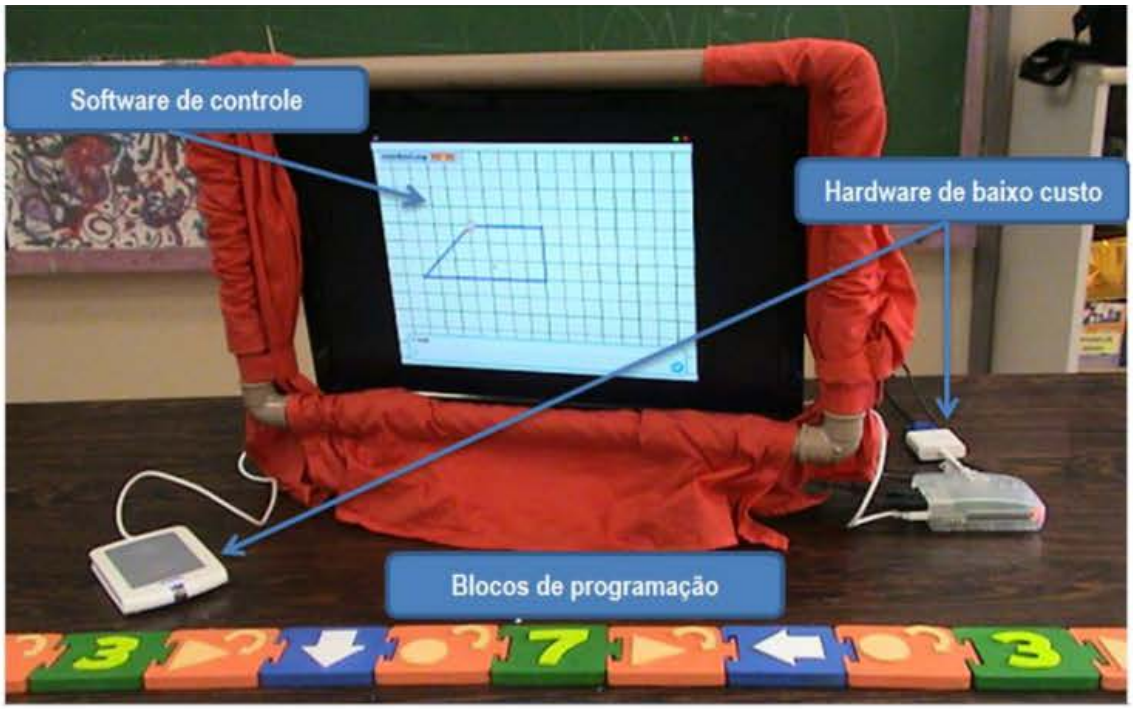
Fonte: (JIN et al., 2018)

Outro aspecto observado foi o formato dos blocos utilizados em cada linguagem, na Ar-maze esses blocos são pequenos cubos de madeira que podem ser conectados com imãs. Cada bloco representa um tipo de movimento que o personagem do software pode realizar, além de instruções como início e fim do programa e loop.

Outras ferramentas que seguem esse modelo de blocos tangíveis são Cubetto, TaPrEC (CARBAJAL; BARANAUSKAS, 2018) e Scottie Go!. Cubetto é composta por várias peças no formato de setas que são encaixadas em um tabuleiro como instruções sequenciais que movimentam um robô no formato de um cubo de madeira. TaPreEC possui blocos de madeira coloridos com uma etiqueta RFID no verso que são encaixados em uma sequência horizontal e uma placa Raspberry Pi que executa o programa e mostra o resultado em um monitor. O jogo Scottie Go! possui um conjunto de cartões coloridos com instruções que são encaixados em uma sequência vertical e escaneados por uma

aplicação móvel que mostra a execução dos comandos por um personagem.

Figura 3 – Imagem da ferramenta TaPrEC



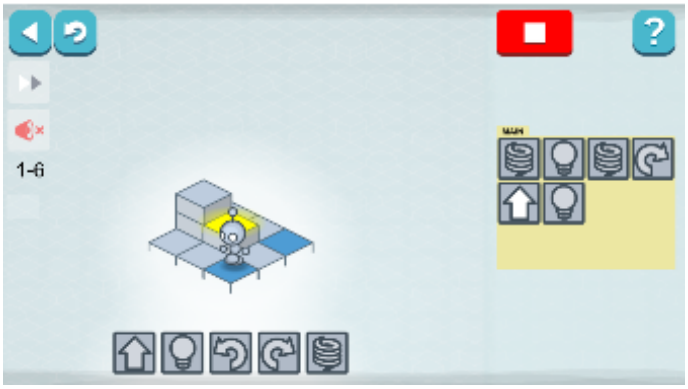
Fonte: (CARBAJAL; BARANAUSKAS, 2018)

3.4 Jogos

Outro tipo de ferramenta utilizada para o ensino de programação, são os jogos. Uma tendência observada foi jogos em que cada fase representa um labirinto ou quebra-cabeça e o usuário precisa construir um conjunto de instruções para guiar o personagem para o fim do labirinto ou para o objetivo da missão.

Exemplos desses jogos são Lightbot (SOUZA et al., 2018), Robotizen (DANTAS et al., 2019), Move (TOLEDO et al., 2020), NoBug's Snack Bar, VR-OCKS, Jogo RPG de (SARKAR; SARKER; HOSSAIN, 2016), e ELIS (TERAN; ARAúJO; PIRES, 2019). Uma característica em trabalhos que utilizam essas ferramentas é a melhora na motivação dos alunos ao trabalhar com jogos e ferramentas que utilizam gamificação.

Figura 4 – Imagem do Jogo LightBot



Fonte: (lightbot.com)

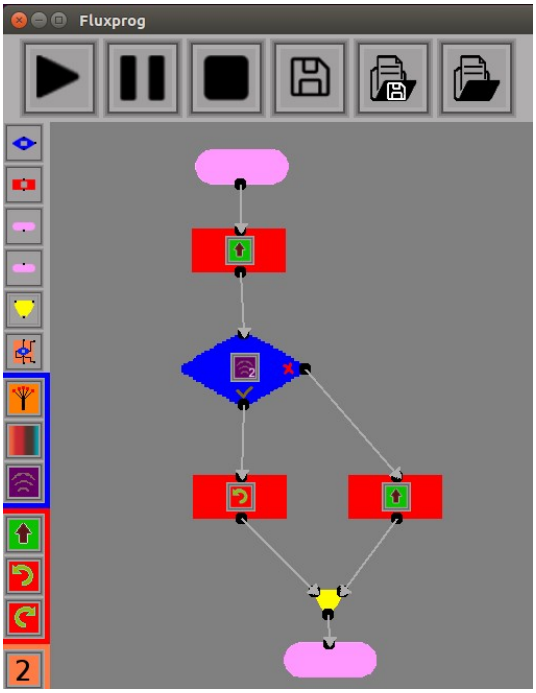
3.5 Outras linguagens visuais com foco em ensino

Além dessas ferramentas, outro padrão observado foi o das ferramentas (FABRO et al., 2019) e Modelix (NETO et al., 2018) e que utilizam linguagens baseadas em fluxogramas. Fluxprog possibilita a programação de dispositivos construídos com Arduino e de robôs em um simulador virtual 3D. Os blocos utilizados no fluxograma possuem uma diferenciação de cor para cada tipo de comando: início e fim do programa, loop, ações, decisões e merge. Modelix também tem foco em robótica educacional, permitindo a programação de placas Arduino.

Outro modelo de linguagem visual encontrado na literatura é o da linguagem Stride (PRICE et al., 2016), semanticamente igual ao Java, que utiliza quadros (frames) para o ensino de programação. Os quadros são semelhantes aos blocos, organizados de forma vertical. No entanto, possui espaços definidos para inserção de comandos em forma textual. O Stride visa reduzir erros de sintaxe cometidos pelo desenvolvedor iniciante. Foram realizadas atividades com alunos do ensino fundamental com um grupo de alunos utilizando Java e outro utilizando Stride, segundo o autor o grupo trabalhando com Stride progrediu mais rápido na execução da atividade, passando menos tempo editando a sintaxe. Isso sugere que a edição baseada em quadros é uma ferramenta útil para reduzir a carga de sintaxe para programadores iniciantes e pode levar a uma melhora na performance de tarefas em programação.

Já a linguagem Visual Programmer tem foco no ensino de programação para alunos surdo e utiliza símbolos para representação das instruções. A ferramenta permite que o código simbólico seja convertido para linguagem C possibilitando que o aluno surdo

Figura 5 – Imagem da ferramenta FluxProg



Fonte: (FABRO et al., 2019)

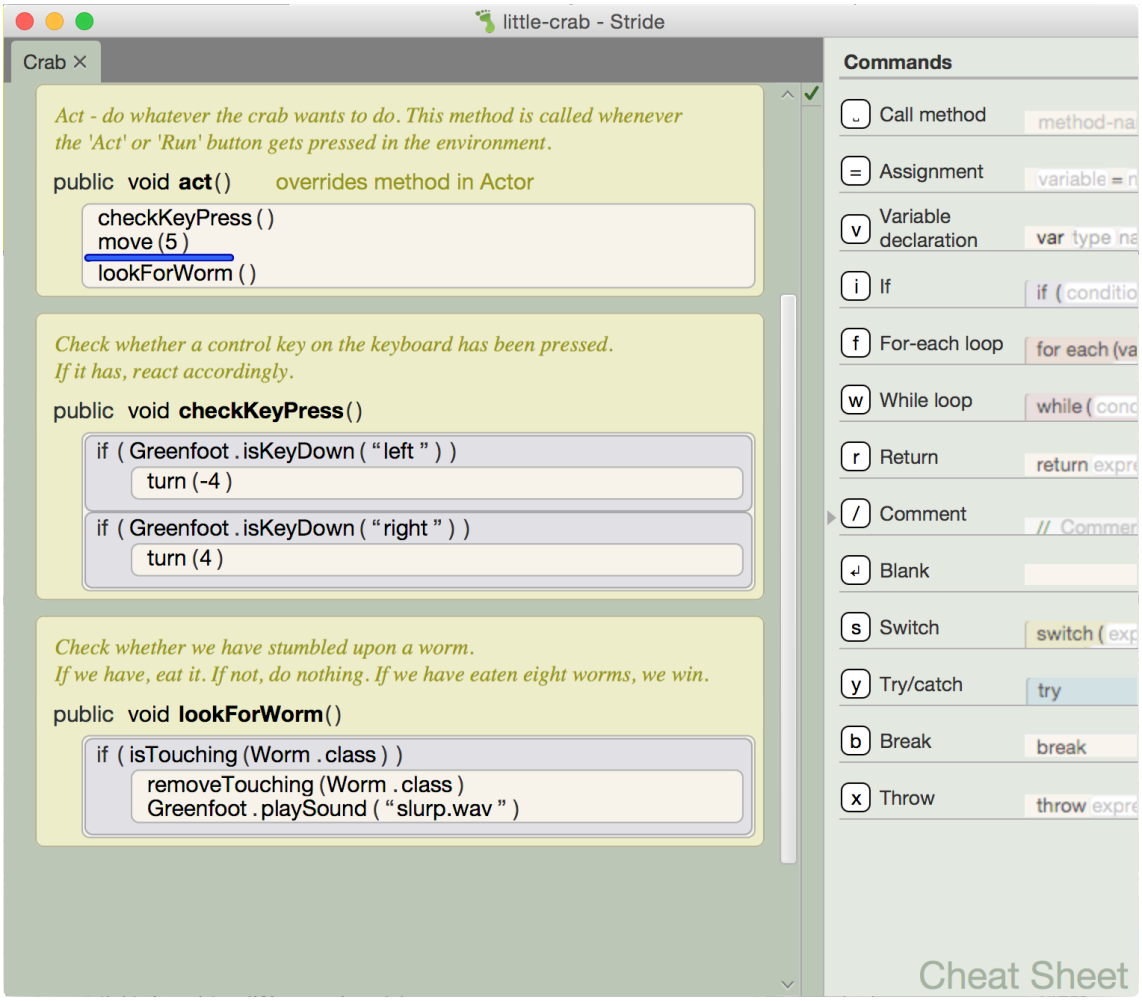
compreenda a aplicabilidade de cada instrução dentro do código-fonte (ANDRADE et al., 2019).

Uma linguagem que também mostrou um padrão diferente do observado em outros trabalhos foi a linguagem Viscuit (WATANABE et al., 2020). Uma linguagem de programação visual voltada para alunos de educação infantil. Por ter como usuário crianças não alfabetizadas, o Viscuit não possui informações textuais e é composto somente por imagens. A linguagem segue um sistema de reescrita: o código é construído com regras representadas por uma estrutura semelhante a um óculos (dois círculos conectados). A criança desenha as imagens a serem utilizadas ao longo do código. Quando o código é executado, a imagem do círculo à esquerda é substituída pela imagem do círculo à direita. É possível também utilizar imagens prontas que representam comando de clicar ou executar sons.

3.6 Sistematização dos resultados

Os trabalhos e resultados discutidos nesse capítulo estão sistematizados nas tabelas Tabela 2 e Tabela 3. Na coluna nível de ensino, estão descritos os níveis de ensino para o qual a linguagem foi construída, já na coluna nível de ensino utilizado estão

Figura 6 – Código na linguagem Stride



Fonte: (PRICE et al., 2016)

enumerados os níveis de ensino nos quais alguma atividade utilizando a linguagem foi realizada. Os níveis enumerados foram Educação Infantil (EI), Ensino Fundamental 1 (EF1), Ensino Fundamental 2 (EF2), Ensino Médio (EM), Ensino Superior (ES) e Não Especificado (NE) quando nenhum nível de ensino foi descrito como foco da linguagem. Na coluna características foram enumeradas as principais características observadas na linguagem relacionadas a organização de blocos vertical (OBV), como observado na linguagem Scratch, organização de blocos horizontal (OBH), como observado na linguagem usada pelos kits Lego, o uso de blocos tangíveis (BT), o uso de realidade aumentada (RA), o uso de realidade virtual (RV), o uso de jogos, a possibilidade de desenvolvimento para dispositivos móveis Android, para Arduino, para Lego Mindstorms e para micro:bit, e casos de linguagens baseadas em regras, fluxogramas ou quadros.

A literatura estudada neste trabalho mostrou a existência e o uso de linguagens de

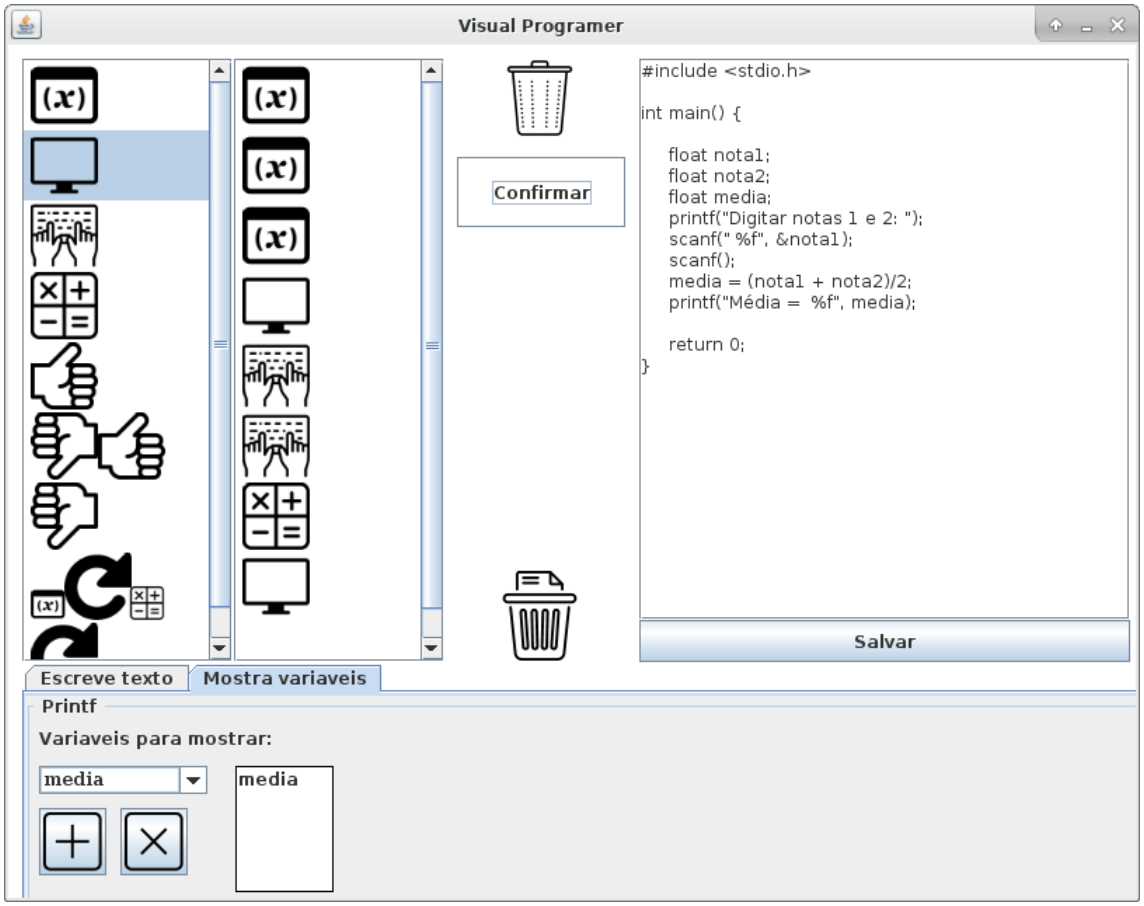
Tabela 2 – Análise das linguagens

Linguagem	Referência	Nível de ensino	Paradigma sintático	Fluxo de execução	Características	Trabalhos que utilizam a linguagem	
						Nível de Ensino Utilizado	Número de artigos
Scratch	(RESNICK et al., 2009)	EF1/EF2/EM	Blocos	Imperativo	OBV	EF1/EF2/EM ES	33
Lego Mindstorms	(education.lego.com/en-us/downloads/mindstorms-ev3/software)	EF2/EM	Blocos	Imperativo	OBH	EF1/EF2/EM ES	7
App Inventor	(WOLBER, 2011) (googleblog.blogspot.com/2010/07/app-inventor-for-android.html) (appinventor.mit.edu)	NE	Blocos	Imperativo	Dispositivo móvel Android OBV	EF2/EM	6
LightBot	lightbot.com/	NE	Blocos	Imperativo	Jogo OBH	EI/EF1/EF2 EM	4
Alice	(CONWAY, 1997)	EF2/EM/ES	Blocos	Imperativo	OBV	EF2/EM	2
TaPrEC	(CARBAJAL; BARANAUSKAS, 2018)	EF1/EF2	Blocos	Imperativo	BT OBH	EF1/EF2	2
Pencil Code	(BAU; BAU, 2014) pencilcode.net/	NE	Blocos	Imperativo	OBV	EF1/EF2/EM	2
BEESM	(SERAJ; AUTEXIER; JANSSEN, 2018)	NE	Textual Blocos	Imperativo	Arduino OBV	-	1
Lego Education We Do	education.lego.com/en-us/downloads/wedo-2/software	EF1	Blocos	Imperativo	OBH	EF1	1
Programming Tool with Tangible Blocks and AR	(HATTORI; HIRAI, 2019)	EF1	Blocos	Imperativo	BT RA OBV	EF1	1
AR-maze	(JIN et al., 2018)	EF1	Blocos	Imperativo	BT RA Jogo OBH	EF1	1
Arcat	(DENG et al., 2019)	EF1	Blocos	Imperativo	BT RA Jogo OBV	EF1	1
FluxProg	(FABRO et al., 2019)	NE	Diagramas	Imperativo	Baseado em Fluxograma Arduino	EM/ES	1
SimBA-PLT*	(SU; LIN, 2018)	EF1/EF2/EM	Blocos	Imperativo	Arduino OBV	EF1/EF2/EM	1
Jogo RPG	(SARKAR; SARKER; HOSSAIN, 2016)	EF1	Blocos	Imperativo	Jogo OBH	EF1	1
MakeCode	www.microsoft.com/en-us/makecode	EF2/EM	Blocos	Imperativo	OBV Arduino/Lego Mindstorms/ micro:bit	EF1	1
Jogo Move	(TOLEDO et al., 2020)	ES	Blocos	Imperativo	Jogo OBV	ES	1
Jogo Robotizen	(DANTAS et al., 2019)	EF2	Blocos	Imperativo	Jogo OBV	EF2	1
MDS e Visual Programmer	(ANDRADE et al., 2019)	NE	Blocos	Imperativo	Baseada em Símbolos	EM	1
Stride	(PRICE et al., 2016)	NE	Quadros	Imperativo	Baseado em quadros	EF2	1
Blockly	(FRASER, 2015) developers.google.com/blockly		Blocos	Imperativo	OBV	EF2	1

Tabela 3 – Análise das linguagens cont.

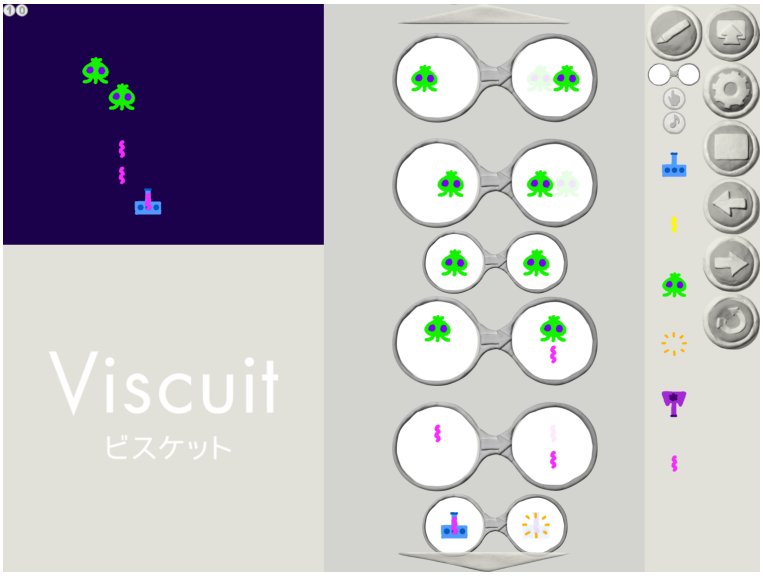
DuinoBlocks for Kids*	(QUEIROZ; SAMPAIO, 2016)	EF1	Blocos	Imperativo	Arduino OBV	EF2	1
Jogo NoBug's Snack Bar	(VAHLDICK et al., 2016)	ES	Blocos	Imperativo	Jogo OBV	ES	1
Jogo Elis	(TERAN; ARAúJO; PIRES, 2019)	NE	Blocos	Imperativo	Jogo OBV	EF2	1
App Lofi Blocks*	lofiblocks.com/en/	NE	Blocos	Imperativo	Arduino OBV	EF2/EM	1
Modelix*	www.modelix.com.br/	NE	Diagramas	Imperativo	Baseado em fluxograma Arduino	EF2	1
Sucre4Kids	(TRILLES; GRANELL, 2018)	EF1	Blocos	Imperativo	Arduino OBV	EM	1
Scratch Jr.	www.scratchjr.org/	EF1	Blocos	Imperativo	OBH	EF1	1
ROBOTC Graphical	www.robotc.net/	NE	Blocos	Imperativo	OBV Arduino/Lego Mindstorms	EF1	1
Scottie Go!	scottiego.com/en/	EF1	Blocos	Imperativo	BT OBV	EF1	1
VR-OCKS	(SEGURA et al., 2020)	EF2	Blocos	Imperativo	Jogo RV OBH e vertical	EF2/ES	1
Snap!	(CARRASQUER, 2019)	NE	Blocos	Imperativo	OBV		1
Stencyl	www.stencyl.com/education/overview/	NE	Blocos	Imperativo	OBV	EF2	1
Viscuit	(WATANABE et al., 2020)	EI	Regras	Lógico	Baseado em regras	EI	1
Cubetto	www.primotoys.com/cubetto/	EI	Blocos	Imperativo	BT OBH	EI	1
Enchating	en.scratch-wiki.info/wiki/Enchanting_(Scratch_modification)	NE	Blocos	Imperativo	OBV Lego Mindstorms NXT	EF2	1
Modkit	(MILLNER; BAAFI, 2011)	NE	Blocos	Imperativo	OBV Arduino	EF2	1
Scratch4Arduino	http://s4a.cat/	NE	Blocos	Imperativo	OBV Arduino	EF2	1

Figura 7 – Imagem da ferramenta Visual Programmer



Fonte: (ANDRADE et al., 2019)

Figura 8 – Imagem da ferramenta Viscuit



Fonte: (WATANABE et al., 2020)

programação visual voltadas para diversas faixas etárias. Da Educação Infantil ao Ensino Superior, iniciativas mostram que esse tipo de linguagem pode ser utilizado com sucesso em diferentes níveis de ensino.

O estudo da literatura também permitiu observar em diversos trabalhos o uso de ferramentas visuais com foco no incentivo ao pensamento computacional. Essas abordagens são feitas principalmente em conjunto com a robótica educacional, além de alguns trabalhos com programação tangível. Outra aplicação interessante das linguagens visuais foi em iniciativas interdisciplinares no Ensino Fundamental e no Ensino Médio, em trabalhos onde a programação era utilizada como um meio para o aluno compreender conteúdos das áreas da Matemática ou Ciências.

Também foi possível verificar que as linguagens visuais encontradas na literatura utilizam em sua maioria o paradigma imperativo. De todas as linguagens analisadas, somente uma – que usa o paradigma lógico – não utiliza o paradigma imperativo.

Foi observado um uso considerável da ferramenta Scratch no ensino de programação em diversos níveis de ensino. Dos trabalhos analisados 33 utilizaram Scratch. Além disso, outros 22 trabalhos utilizaram ferramentas com linguagens semelhantes ou baseadas em Scratch. É possível verificar o uso dos blocos coloridos representando diferentes tipos de instrução, organizados de forma vertical nessas ferramentas, como mostrado na Figura 9.

Figura 9 – Blocos da ferramenta Scratch como exemplo de uma linguagem com organização de blocos vertical

Fonte: Autora (2021)

Outro padrão observado foi o uso de linguagens com blocos predominantemente quadrados, organizados de forma horizontal e com uso frequente de símbolos maiores e mais simples para a representação das instruções, exemplificado na Figura 10. Dos trabalhos analisados 19 utilizaram ferramentas com esse padrão, dentre elas os kits Lego, Scratch Jr., as ferramentas tangíveis Ar-maze, TaPrEC e Cubetto e os jogos Lightbot, Robotizen e o jogo RPG de (SARKAR; SARKER; HOSSAIN, 2016).

Figura 10 – Blocos da ferramenta Scratch Jr. como exemplo de uma linguagem com organização de blocos horizontal

Fonte: Autora (2021)

Em geral, esses dois padrões de organização de blocos na vertical semelhante ao Scratch e organização de blocos na horizontal semelhante ao LabView/Lego foram os mais utilizados nos trabalhos analisados. As ferramentas que não seguiram esses padrões foram FluxProg, Modelix, Visual Programmer, VR-OCKS, Viscuit e Stride.

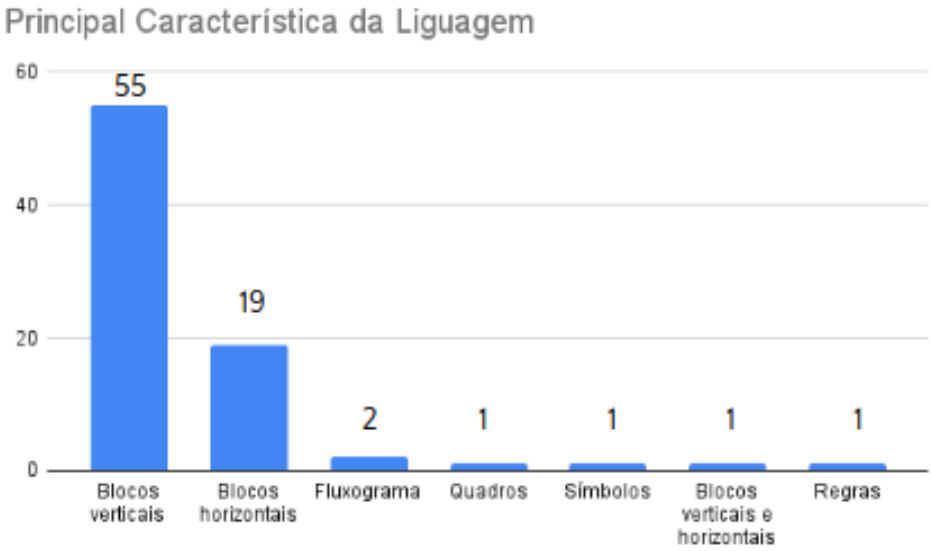
A Figura 11 apresenta um gráfico de barras com esses padrões observados nas linguagens, mostrando que 55 trabalhos utilizaram linguagens com organização de blocos vertical, 19 trabalhos utilizaram linguagens com organização de blocos horizontal e que as exceções foram seis trabalhos que utilizaram linguagens tendo como principais características fluxogramas, quadros, símbolos, regras ou uma combinação da organização de blocos verticais e horizontais.

VR-OCKS utiliza um misto dos dois padrões, com organização de blocos na horizontal na maior parte das instruções, mas utilizando uma abordagem de organização de blocos na vertical quando se trata de estruturas de repetição.

FluxProg e Modelix utilizam linguagens baseadas em fluxogramas, o que parece promissor, principalmente se for considerado que fluxogramas já são utilizados por professores em aulas introdutórias de algoritmos e programação.

A linguagem Stride é bastante semelhante à linguagem Java e possui uma estrutura de quadros. Pode ser uma alternativa interessante para transição de uma turma de alunos que estuda programação visual com blocos para uma linguagem textual orientada a

Figura 11 – Gráfico de barras apresentando a principal característica de cada linguagem



Fonte: Autora (2021)

objetos.

Visual Programmer é composta por símbolos com mais detalhe, em relação às linguagens com organização de blocos na horizontal, com uma representação mais próxima das instruções da linguagem C. Assim como Stride também pode ser uma ferramenta útil na transição de uma linguagem visual para uma linguagem textual.

Viscuit também se destacou como a única linguagem analisada a não seguir o paradigma imperativo.

4 REFERENCIAL TEÓRICO

Neste capítulo é abordado brevemente o referencial teórico utilizado como base para a construção da linguagem Pandora. Na Seção 4.1 está descrito o paradigma funcional e na Seção 4.2 as gramáticas de grafos.

4.1 Linguagens Funcionais

As linguagens funcionais utilizam o paradigma de programação funcional que tem como base as funções matemáticas. Segundo (SEBESTA, 2018) "uma linguagem de programação puramente funcional não usa variáveis, nem sentenças de atribuição, liberando o programador de preocupações relacionadas às células de memória do computador no qual o programa é executado." Isso possibilita que as funções não tenham efeitos colaterais pois uma função não pode alterar valores externos e não possui variáveis de estado então sempre que uma função receber um mesmo argumento retorna a mesma saída.

As linguagens funcionais não possuem construções de iteração, como funções *for* ou *while* nas linguagens iterativas. Para execução de repetições é usado o mecanismo de recursão.

A operação mais relevante no paradigma funcional é a composição de funções. Dadas duas funções $f : A \to B$ e $g : B \to C$, a composição de f com g, denotada por $f \circ g$ é a função que para cada elemento $x \in A$ retorna o valor $g(f(x))$.

LISP e Haskell são duas linguagens funcionais utilizadas tanto no mercado[1] quanto na academia. Abaixo, alguns exemplos da sintaxe dessas linguagens, de forma a justificar tanto a escolha do paradigma funcional como uma abordagem visual com outro tipo de organização.

A sintaxe da linguagem LISP é baseada em listas, em que o primeiro elemento da lista é o nome da função invocada e os restantes são os seus argumentos. O construtor de funções *defun* é também uma função da linguagem. Abaixo, um exemplo de código para calcular as raízes de uma equação de segundo grau, segundo a fórmula de Báskara. Note-se que a estrutura aninhada das chamadas segue o esquema de composição matematicamente definido.

```
(defun raizSegGrau (a b c)
```

[1] O software AutoCAD (http://www.autodesk.com.br), por exemplo, é totalmente escrito em LISP.

```
(let (delta (- (* b b)(* 4 a c)))
     (x1 (/ (+ (* -1 b) (sqrt delta)) (* 2 a)))
     (x2 (/ (- (* -1 b) (sqrt delta)) (* 2 a)))
(list x1 x2)))
```

A sintaxe da linguagem Haskell é diversa, um pouco mais parecida com o modelo de programação sequencial usado em linguagens imperativas, mas ainda com forte estrutura matemática. A declaração da função estabelece seu domínio e contradomínio (Haskell é uma linguagem tipada, ao contrário de LISP) com os nomes dos parâmetros formais logo abaixo. Note que os valores de saída da função também têm nomes, pois a função pode admitir saída composta. O código do mesmo cálculo das raízes de uma equação de segundo grau encontra-se a seguir.

```
raizSegGrau :: Double -> Double -> Double -> (Double, Double)
raizSegGrau a b c = (x1, x2)
    where
        x1 = (-b - sqrt delta) / (2*a)
        x2 = (-b + sqrt delta) / (2*a)
        delta = b**2 - 4*a*c
```

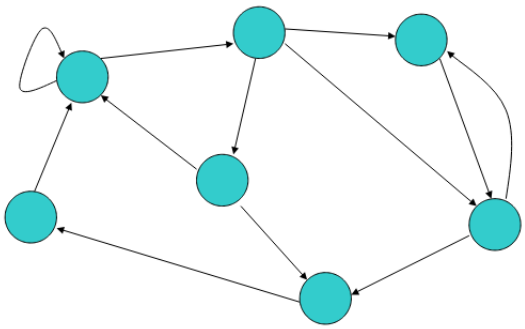
O mecanismo de iteração das linguagens funcionais é a recursão. Abaixo, exemplos em LISP e Haskell, respectivamente, do cômputo da função fatorial.

```
(defun fatorial (n)
    (if (eq n 0) 1
        (* n (fatorial (- n 1)))))
```

```
fatorial :: Integer -> Integer
fatorial 0 = 1
fatorial 1 = 1
fatorial n = n * fatorial (n-1)
```

O encadeamento da composição de funções é um foco de dificuldade por parte dos alunos que estão começando a trabalhar com linguagens desse tipo. Ainda que a notação matemática para a composição tenha essa característica, não há necessidade de uma linguagem nesse paradigma usar essa notação. Uma composição de funções é um encadeamento sequencial que pode ser expresso dessa forma. Ou seja, a composição das funções f e g pode ser expressa como $f;g$, com a semântica usual da saída de f ser entrada de g (para funções de aridade 1). A representação sequencial tem duas vantagens, no contexto deste trabalho: (i) possibilitar a construção da composição por meio de blocos,

Figura 12 – Exemplo de Grafo



Fonte: (JORNADAS..., 2006)

facilitando sua utilização e entendimento por parte dos alunos e (ii) facilitar a migração para linguagens imperativas textuais no futuro.

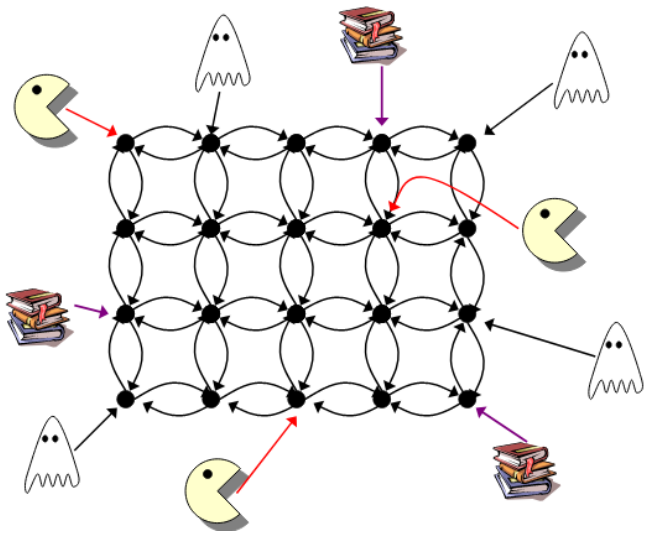
4.2 Gramática de Grafos

Grafos são estruturas algébricas capazes de expressar relacionamentos de qualquer tipo. Qualquer diagrama utilizado na área de computação pode ser descrito matematicamente como um grafo (FERREIRA; RIBEIRO, 2006). Grafos podem ser um recurso muito útil para representar e comunicar informações de forma visual, inclusive para não especialistas na área de Computação. Formalmente, um grafo é definido como uma tupla $G = (V, E \subseteq V \times V)$, onde V é o conjunto de nodos ou vértices e E é o conjunto de arcos ou arestas, definidas como pares de nodos. Pode-se também usar a definição algébrica, em que um grafo é representado como uma tupla $G = (V, E, src, tar)$, em que V é o conjunto de nodos, E é o conjunto de arestas, src é a função de origem que para cada aresta retorna seu nodo de origem e tar é a função de destino que para cada aresta retorna seu nodo de destino. Essa representação não permite a existência de mais de uma aresta que conecta os mesmos dois nodos. Assim, cada aresta possui um nodo de origem e um nodo de destino, possuindo uma identificação única.

Um grafo pode ser um grafo rotulado quando possuir rótulos relacionados aos nodos ou às arestas, diferenciando-os com alguma informação adicional que se queira representar. Isso facilita a utilização do grafo como um modelo ou representação de elementos reais. Um exemplo de grafo rotulado é apresentado na Figura 13, onde os nodos do grafo representam elementos de um jogo Pac-Man.

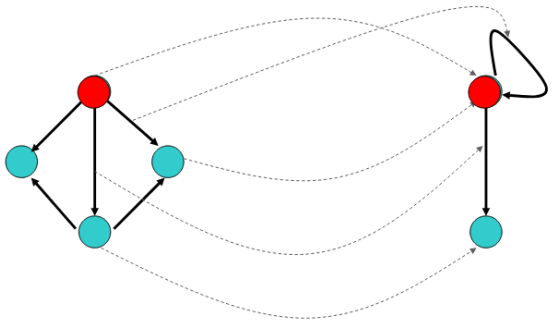
Morfismos são mapeamentos utilizados para representar relações entre duas

Figura 13 – Exemplo de Grafo Rotulado



Fonte: (JORNADAS..., 2006)

Figura 14 – Exemplo de Morfismo de Grafos

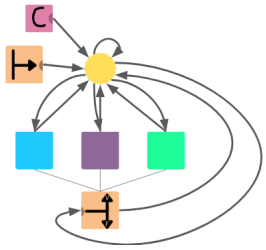


Fonte: (JORNADAS..., 2006)

estruturas algébricas, preservando a estrutura que está sendo mapeada. Formalmente, um morfismo entre dois grafos $G_1 = (V_1, E_1, src_1, tar_1)$ e $G_2 = (V_2, E_2, src_2, tar_2)$ é um par de funções (f_V, f_E), em que f_V mapeia os nodos de G_1 nos nodos de G_2 e f_E mapeia as arestas de G_1 nas arestas de G_2, de forma que a origem e destino das arestas mapeadas seja preservada. Ou seja, para qualquer aresta $e \in E_1$ tem-se que $f_V(src_1(e)) = src_2(f_E(e))$ e $f_V(tar_1(e)) = tar_2(f_E(e))$. A Figura 14 mostra um exemplo de morfismo de grafos. Um morfismo é dito parcial quando mapeia um subgrafo de G_1 em G_2.

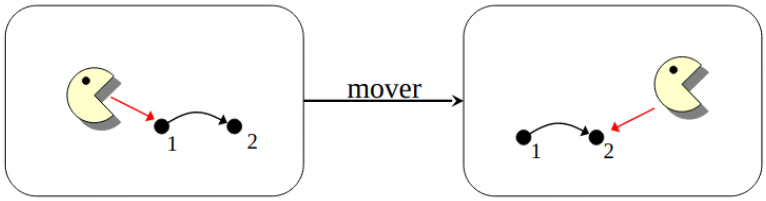
Um exemplo de utilização de morfismo é em grafos tipados. Um grafo tipado é um grafo que possui morfismo total com um grafo tipo. Esse grafo tipo define quais os tipos de nodos e arestas que podem ocorrer em um determinado grafo. Um exemplo de grafo tipo para a aplicação do Pac-Man pode ser observado na Figura 15. Note que o

Figura 15 – Exemplo de Grafo Tipo



Fonte: (JORNADAS. . . , 2006)

Figura 16 – Exemplo de Regra de uma Gramática de Grafos



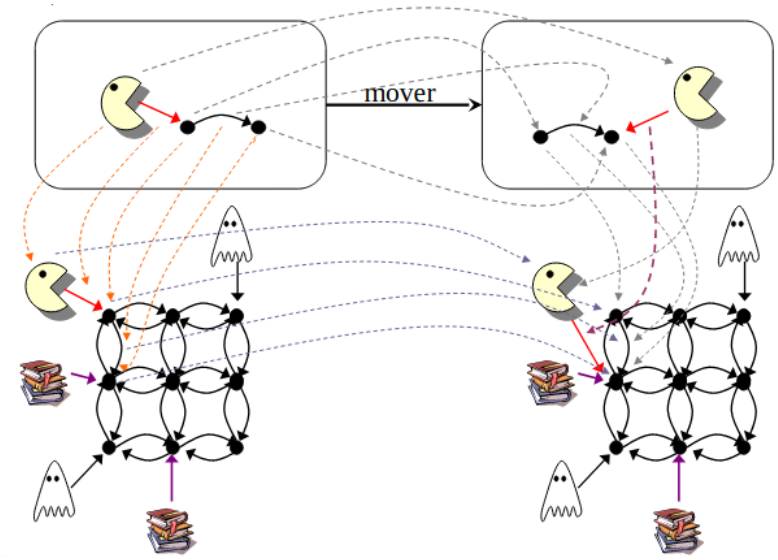
Fonte: (JORNADAS. . . , 2006)

grafo tipo estabelece exatamente quais são os nodos e arestas que que podem ocorrer no grafo que representa o jogo Pac-Man. A tentativa de colocar um arco entre um Pac-Man e um fantasma, por exemplo, não seria possível, visto que essa aresta não aparece no grafo tipo.

Utilizando esses conceitos é possível aplicar um sistema baseado em regras em um grafo, construindo uma gramática que represente através de um conjunto finito de regras, um conjunto de grafos que pode ser gerado a partir dessas regras. Assim, gramáticas de grafos são uma generalização de gramáticas de Chomsky para grafos. A vantagem do uso de uma gramática de grafos está na apresentação visual e intuitiva, o que facilita a compreensão da representação inclusive para não especialistas em métodos formais (FERREIRA; RIBEIRO, 2006).

Para que seja possível a aplicação de uma regra da gramática em um grafo é necessário que exista um morfismo total ou parcial entre o que está do lado esquerdo da regra e o grafo, bem como o que está do lado direito da regra e o grafo após a aplicação da regra. Um exemplo de regra de uma gramática de grafos pode ser observado na Figura 16 que apresenta a regra a ser aplicada para mover o Pac-Man no jogo representado por um grafo. A regra determina que quando o nodo Pac-Man possui uma aresta que o liga a um nodo 1, que por sua vez está ligado a um nodo 2, é possível criar uma aresta que liga o

Figura 17 – Exemplo de aplicação de Regra de uma Gramática de Grafos



Fonte: (JORNADAS. . . , 2006)

nodo Pac-Man ao nodo 2 e remover a aresta que o liga ao nodo 1, representando assim o movimento do Pac-Man. Na Figura 17 é possível observar que a regra mover pode ser aplicada naquela situação, pois existe um morfismo entre a regra e o grafo que representa o jogo.

Gramáticas de grafos são usadas para representação de estruturas dinâmicas, incluindo sintaxe de linguagens de programação visuais (ZHANG; ZHANG, 2003; FERREIRA; RIBEIRO, 2006). Neste trabalho, gramáticas de grafos serão usadas para apresentar a sintaxe da linguagem Pandora.

5 A LINGUAGEM PANDORA

Esse capítulo descreve a proposta de linguagem construída. Na Seção 5.1 é descrita a estrutura geral da linguagem, na Seção 5.2 é especificada a gramática da linguagem Pandora na Seção 5.3 são apresentados alguns exemplos de códigos Pandora e na Seção 5.4 é discutida a concepção de uma ferramenta para programação Pandora.

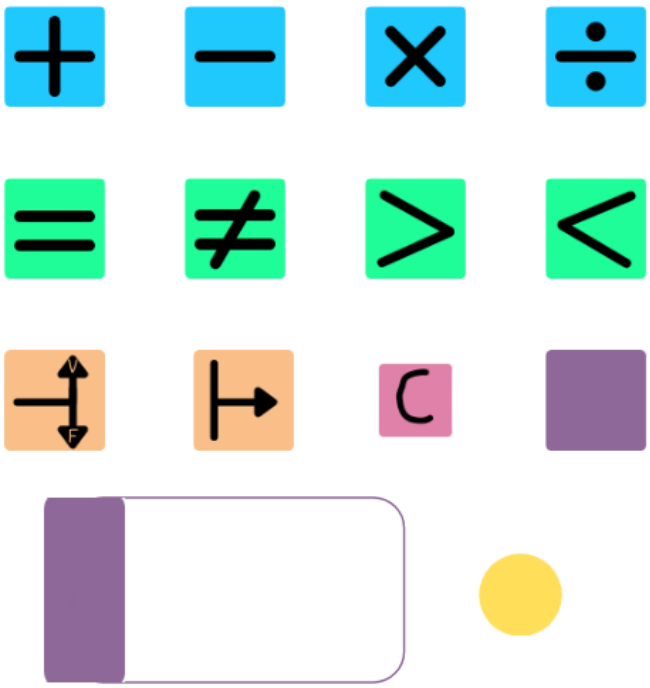
5.1 Estrutura e símbolos da linguagem

A partir das linguagens estudadas optou-se pelo desenvolvimento de uma linguagem funcional visual seguindo uma organização de blocos horizontal. A estrutura de blocos visa tornar o visual mais interessante, ainda que mantendo todos os elementos de uma programação textual. A organização horizontal visa reproduzir a estrutura de programação das linguagens textuais, de forma que a migração da programação em Pandora para uma linguagem de programação textual possa ser feita sem mudanças bruscas de paradigma. A definição formal da sintaxe da linguagem Pandora foi utilizada uma gramática de grafos tipada.

Os símbolos utilizados na linguagem Pandora são blocos no formato de quadrados coloridos para a representação de funções e círculos para representação de fluxo de dados no programa, nomeadamente parâmetros e valores de retorno das funções. Pandora disponibiliza inicialmente onze funções que podem ser combinadas e encapsuladas em uma nova função criada pelo usuário. A declaração dessa nova função é representada por um retângulo roxo e branco: na área roxa é inserido o nome da função e na área branca são inseridas as funções que a compõem. As funções equivalentes a operações aritméticas (soma, subtração, multiplicação e divisão) são representadas por quadrados azuis. As funções equivalentes a operações de comparação (igualdade, diferença, maior que e menor que) são representadas por quadrados verdes. A função que retorna um valor constante é representada por um quadrado rosa. A função equivalente a um teste condicional é representada por dois quadrados beges representando a função se, que divide o fluxo de execução em dois caminhos e a função decisão que mostra a saída do caminho executado. Já as funções criadas pelo usuário são representadas por um quadrado roxo. Esses símbolos serão conectados por setas representando a passagem de um valor ou por uma linha simples representado o fluxo de execução no caso específico das funções se/decisão. Todas as representações de função são nodos do grafo da linguagem. A

Figura 18 apresenta todos os nodos que representam funções na linguagem Pandora.

Figura 18 – Nodos de representação dos elementos da linguagem



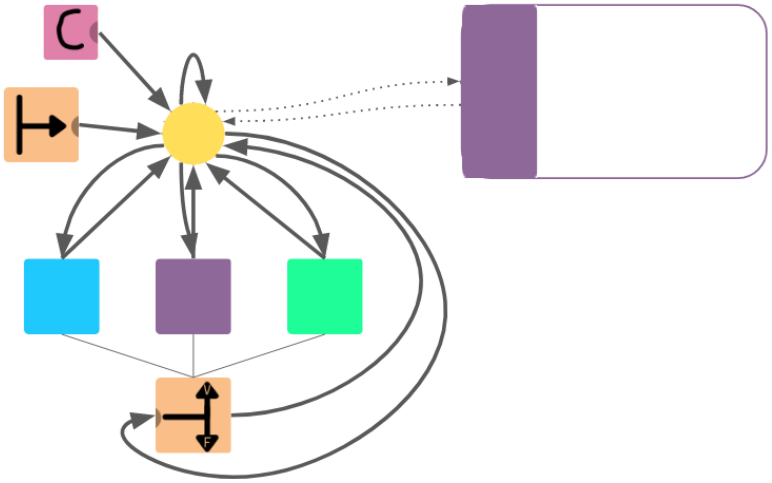
Fonte: Autora (2021)

Cada programa da linguagem Pandora é um grafo tipado onde os nodos representam as funções e os valores e as arestas os fluxos de dados e de execução do programa. A opção pelo uso de uma gramática de grafos para descrição da sintaxe foi devido à facilidade de representação de informações visuais, foco da linguagem Pandora. Os programas são grafos tipados sobre o grafo tipo mostrado na Figura 19. Conforme o grafo tipo uma função constante, uma função decisão ou uma função qualquer podem gerar como saída um valor representado pelo círculo amarelo. O próprio círculo pode passar o seu valor para outro símbolo igual. Um valor também pode ser uma entrada para uma função se ou uma função qualquer.

5.2 Gramática da linguagem Pandora

A gramática da linguagem Pandora é composta por 20 (vinte) regras que serão utilizadas para a construção dos programas. Nas Figuras 20, 21 e 22 pode-se observar essas regras. Além dos símbolos apresentados anteriormente, também é utilizado um

Figura 19 – Grafo tipo da linguagem Pandora



Fonte: Autora (2021)

quadrado cinza menor como símbolo não terminal, para permitir a definição de quantas funções forem necessárias. A seguir, as regras da gramática são explicadas, para um melhor entendimento:

Regra (i) : determina que dado um símbolo não terminal *s*, pode-se substituí-lo por uma definição de função *f* com um valor interno conectado a um valor de saída.

Regra (ii) : estabelece que dado um símbolo não terminal *s*, pode-se substituí-lo por uma definição de função *f* com um valor interno conectado a um valor de saída e outro símbolo não terminal.

Regra (iii) : determina que dada uma definição de função *f* com um valor interno *x* conectado a um valor de saída *y* pode-se acrescentar a essa função um valor de entrada conectado a outro valor interno.

Regra (iv) : estabelece que dada uma definição de função *f* pode-se inserir no corpo da função *f* uma função qualquer a com um valor de saída.

Regra (v) : estabelece que dada uma função qualquer *a*, inserida no corpo da função *f*, com um valor de saída *x* pode-se adicionar à função a um valor de entrada.

Regra (vi) : estabelece que dada uma definição de função *f* com um valor de entrada *x* conectado a um valor interno *y* e uma função qualquer *a*, inserida no corpo da função *f*, com uma entrada *z* pode-se conectar o valor *y* com a entrada *z* de modo que o valor de entrada da função *f* e o valor de entrada da função a sejam iguais. Essa regra só pode ser aplicada se não houver outro valor conectado ao valor *z*.

Regra (vii) : determina que dada uma definição de função *f* com um valor de saída *z* conectado a um valor interno *y* e uma função qualquer *a*, inserida no corpo da função *f*, com uma saída *x* pode-se conectar o valor da saída *x* com o valor *y* de modo que o valor de saída da função *f* e o valor de saída da função *a* sejam iguais. Essa regra só pode ser aplicada se não houver outro valor conectado ao valor *y*.

Regra (viii) : define que dada uma definição de função *f*, uma função qualquer *a* com um valor de saída *x* e uma função qualquer *b* com um valor de entrada *y*, ambas inseridas no corpo da função *f*, pode-se determinar que *x* e *y* possuem o mesmo valor de modo que a saída da função *a* torna-se a entrada da função *b*, conectando as duas funções. Essa regra só pode ser aplicada se não houver outro valor conectado ao valor *y*.

Regra (ix) : define que dada uma definição de função *f*, uma função de decisão *a* com um valor de saída *x* e uma função qualquer *b* com um valor de entrada *y*, ambas inseridas no corpo da função *f*, pode-se determinar que *x* e *y* possuem o mesmo valor de modo que a saída da função *a* torna-se a entrada da função *b*, conectando as duas funções. Essa regra só pode ser aplicada se não houver outro valor conectado ao valor *y*.

Regra (x) : estabelece que dada uma função qualquer *a* com um valor de saída *x* pode-se substituir a função *a* por uma função constante. Essa regra só pode ser aplicada se não houver um valor de entrada conectado a função *a*.

Regra (xi) : define que dada uma função qualquer *a* com com entradas *x* e *y* e uma saída *z* pode-se substituir a função *a* por uma função de adição. Essa regra só pode ser aplicada se não houver outro valor de entrada conectado a função *a*.

Regra (xii) : define que dada uma função qualquer *a* com com entradas *x* e *y* e uma saída *z* pode-se substituir a função *a* por uma função de subtração. Essa regra só pode ser aplicada se não houver outro valor de entrada conectado a função *a*.

Regra (xiii) : define que dada uma função qualquer *a* com com entradas *x* e *y* e uma saída *z* pode-se substituir a função *a* por uma função de multiplicação. Essa regra só pode ser aplicada se não houver outro valor de entrada conectado a função *a*.

Regra (xiv) : define que dada uma função qualquer *a* com com entradas *x* e *y* e uma saída *z* pode-se substituir a função *a* por uma função de divisão. Essa regra só pode ser aplicada se não houver outro valor de entrada conectado a função *a*.

Regra (xv) : define que dada uma função qualquer *a* com com entradas *x* e *y* e uma saída

z pode-se substituir a função a por uma função de igualdade. Essa regra só pode ser aplicada se não houver outro valor de entrada conectado a função a.

Regra (xvi) : define que dada uma função qualquer a com com entradas x e y e uma saída z pode-se substituir a função a por uma função de diferença Essa regra só pode ser aplicada se não houver outro valor de entrada conectado a função a.

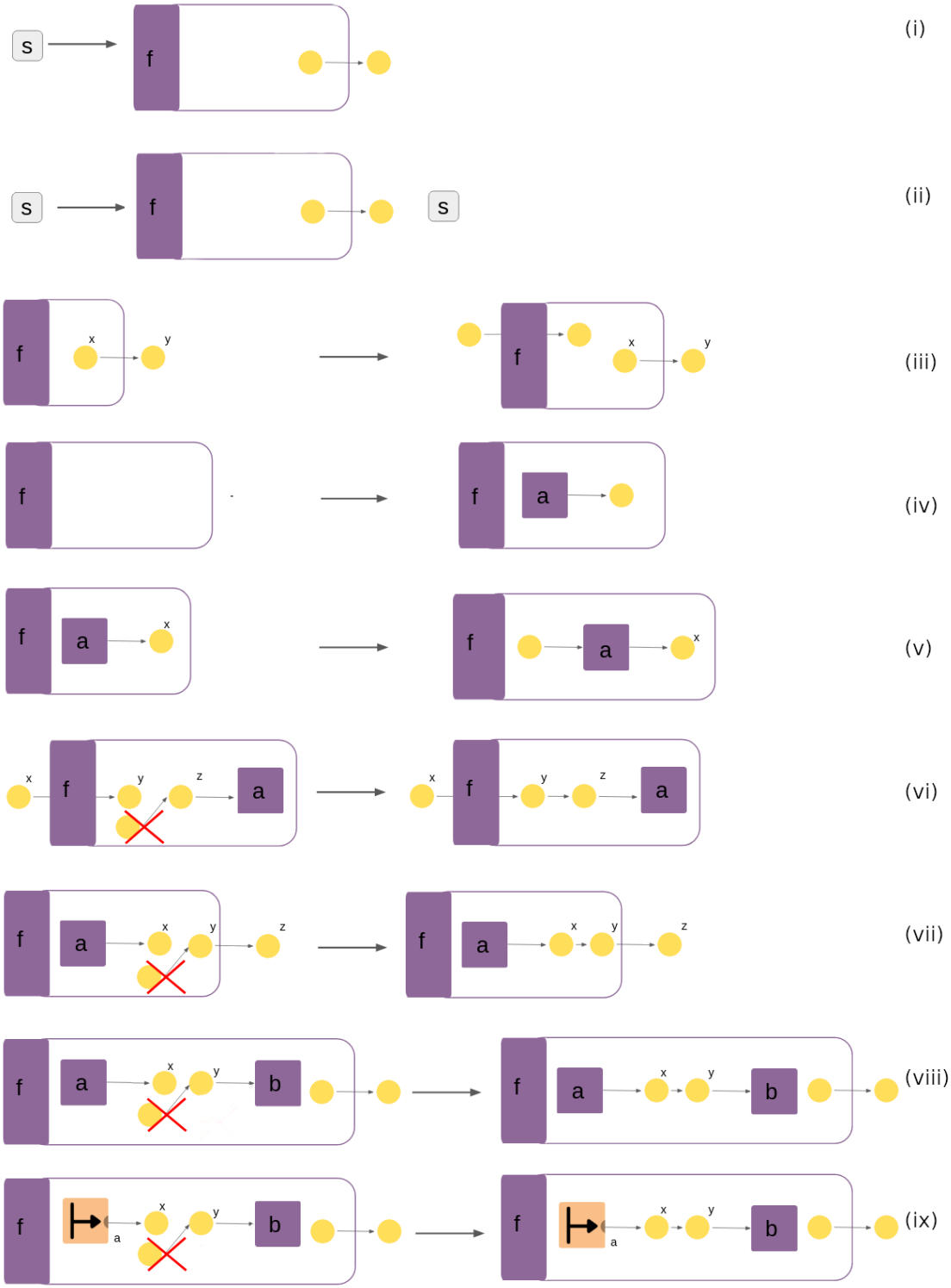
Regra (xvii) : define que dada uma função qualquer a com com entradas x e y e uma saída z pode-se substituir a função a por uma função de maior que. Essa regra só pode ser aplicada se não houver outro valor de entrada conectado a função a.

Regra (xviii) : define que dada uma função qualquer a com com entradas x e y e uma saída z pode-se substituir a função a por uma função de menor que. Essa regra só pode ser aplicada se não houver outro valor de entrada conectado a função a.

Regra (xix) : determina que dada uma definição de função f e uma função qualquer a, inserida no corpo da função f, com um valor de saída x, pode-se utilizar este valor x como entrada de uma função se, que por sua vez será conectada a duas funções quaisquer b e c que possuem valores de saída. Esses valores serão conectados à função decisão que possui um valor de saída.

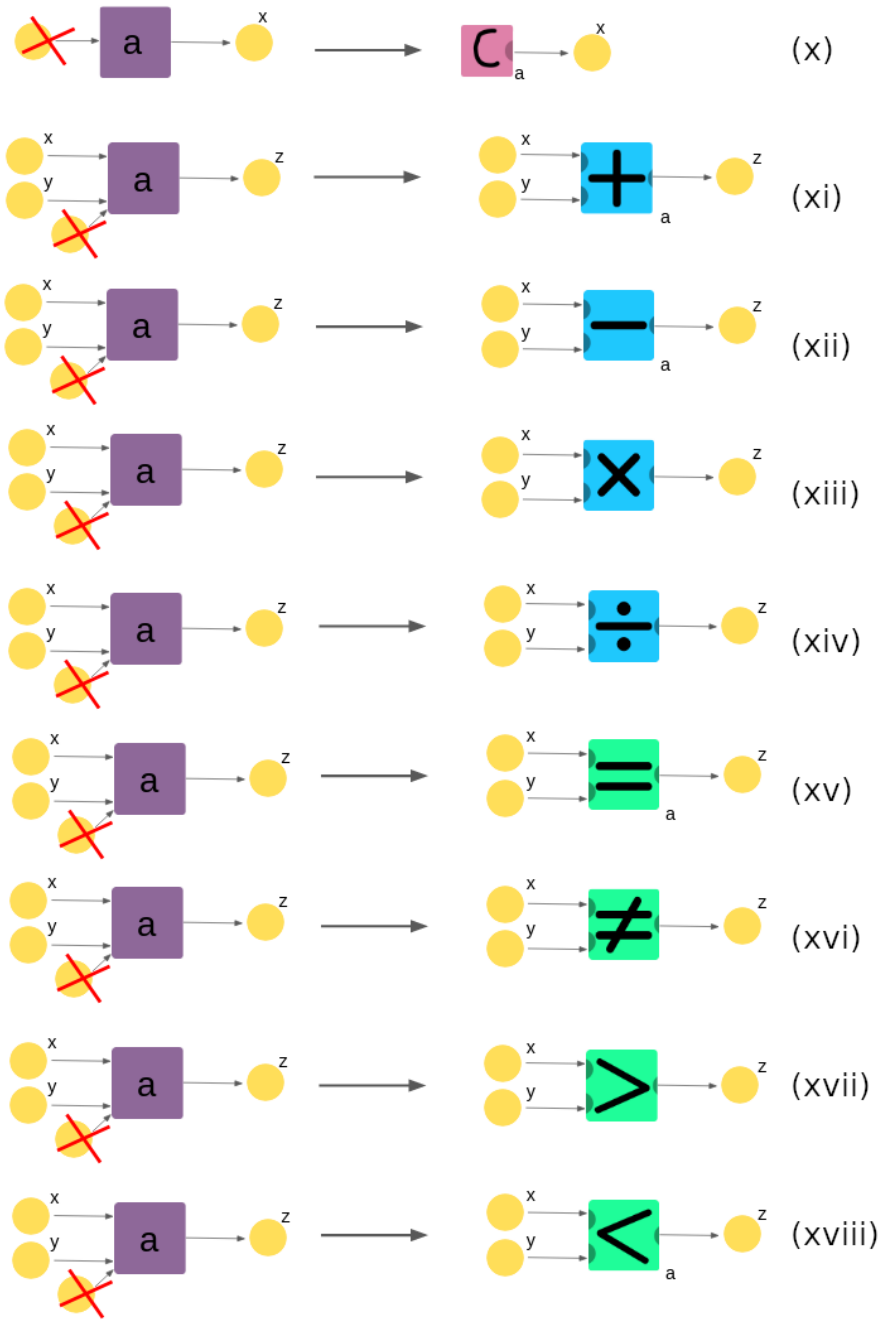
Regra (xx) : define que dada uma definição de função f, uma função qualquer c com um valor de entrada r e um valor de saída s e uma função se p conectada a duas funções quaisquer a e b com valores de saída y e x, respectivamente, por sua vez conectados a uma função decisão q com um valor de saída z, sendo que as funções a, b, c, p e q estão todas inseridas no corpo da função f, é possível utilizar o valor de saída x da função b como entrada da função c, conectando-o ao valor r, de modo que o valor de saída s da função c passa a estar conectado à função de decisão q.

Figura 20 – Gramática da linguagem Pandora



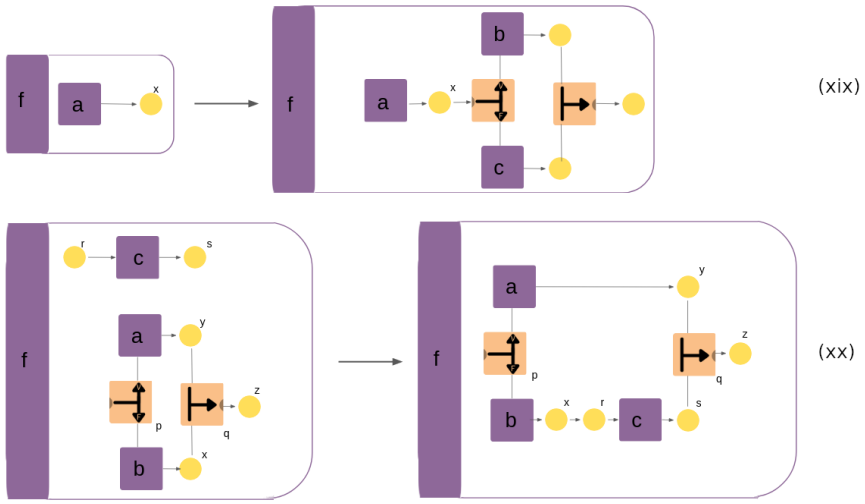
Fonte: Autora (2021)

Figura 21 – Gramática da linguagem Pandora (cont.)



Fonte: Autora (2021)

Figura 22 – Gramática da linguagem Pandora (cont.)



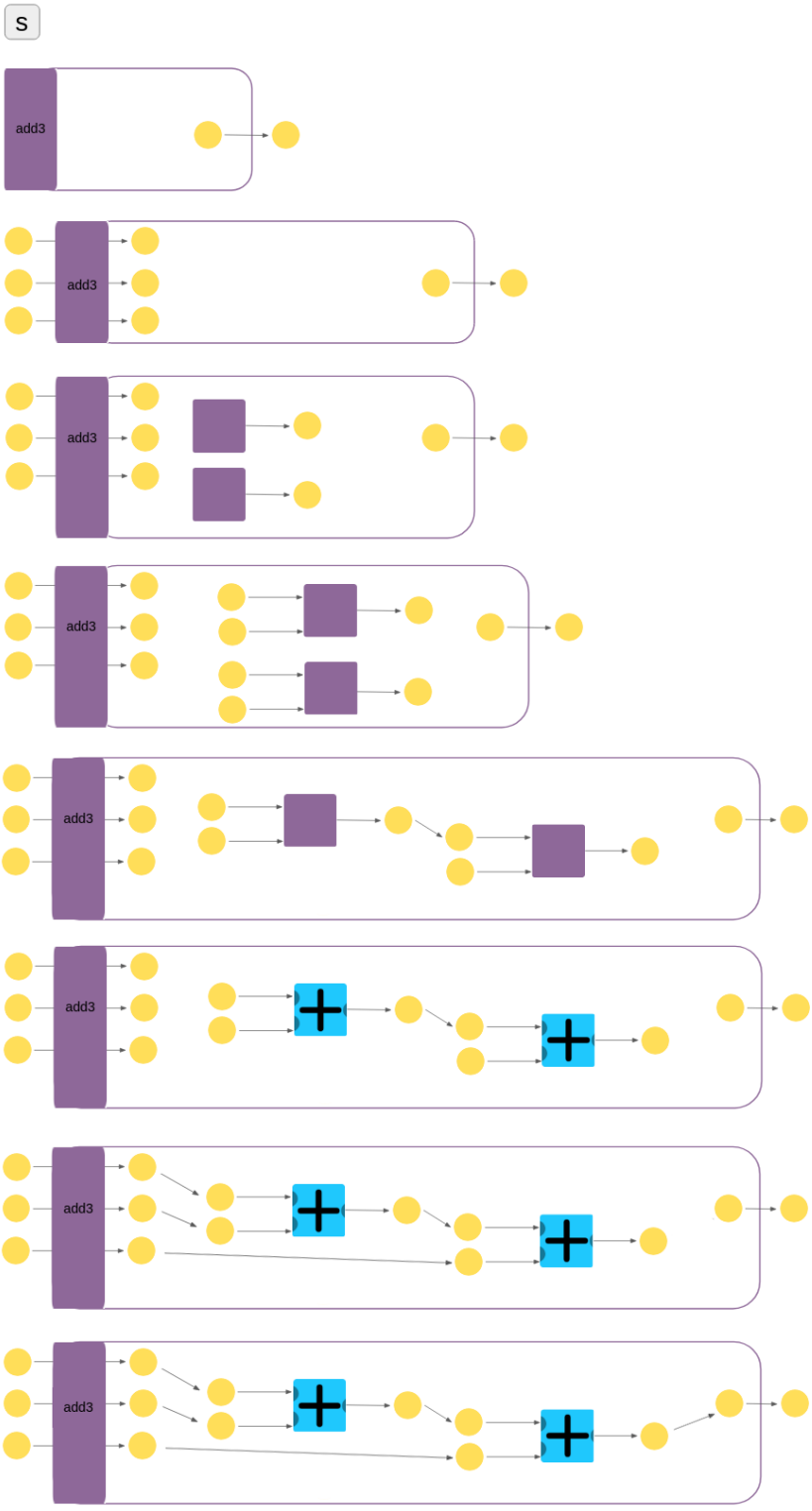
Fonte: Autora (2021)

5.3 Exemplos de programas em Pandora

A partir da gramática construída, é possível escrever programas na linguagem Pandora. Na Figura 23 está representada a derivação de um código Pandora para adição de três valores. Inicialmente existe apenas um símbolo não terminal, ao aplicar a regra (i), o símbolo não terminal é substituído pela definição da função add3. Após, aplica-se três vezes a regra (iii) adicionando assim, três valores de entrada à função add3. Em seguida é aplicada duas vezes a regra (iv) que adiciona duas funções no corpo da função add3 e a regra (v) duas vezes em cada uma dessas funções, adicionando valores de entrada. Após, é aplicada a regra (viii) que permite que a saída da primeira função seja conectada à primeira entrada da segunda função. Em seguida é aplicada a regra (xi) em ambas as funções, identificando-as como funções de adição. E por fim, são aplicadas as regras (vi) e (vii) que conectam as entradas da função add às entradas das funções de adição (as duas primeiras entradas da função add3 equivalem às duas primeiras entradas da primeira função e a terceira entrada da função add equivale a segunda entrada da segunda função) e a saída da segunda função adição à saída da função add3. Desse modo, é construído um código na linguagem Pandora que recebe três valores em sua entrada, passa os dois como parâmetro para uma função que realiza a soma desses valores; o resultado dessa soma é passado como parâmetro para outra função que o soma com o terceiro valor de entrada da função; o resultado dessa segunda soma é passado como valor de saída da função add3.

Na Figura 24 está representada a derivação de um código Pandora para cálculo

Figura 23 – Derivação de código Pandora para adição de três valores



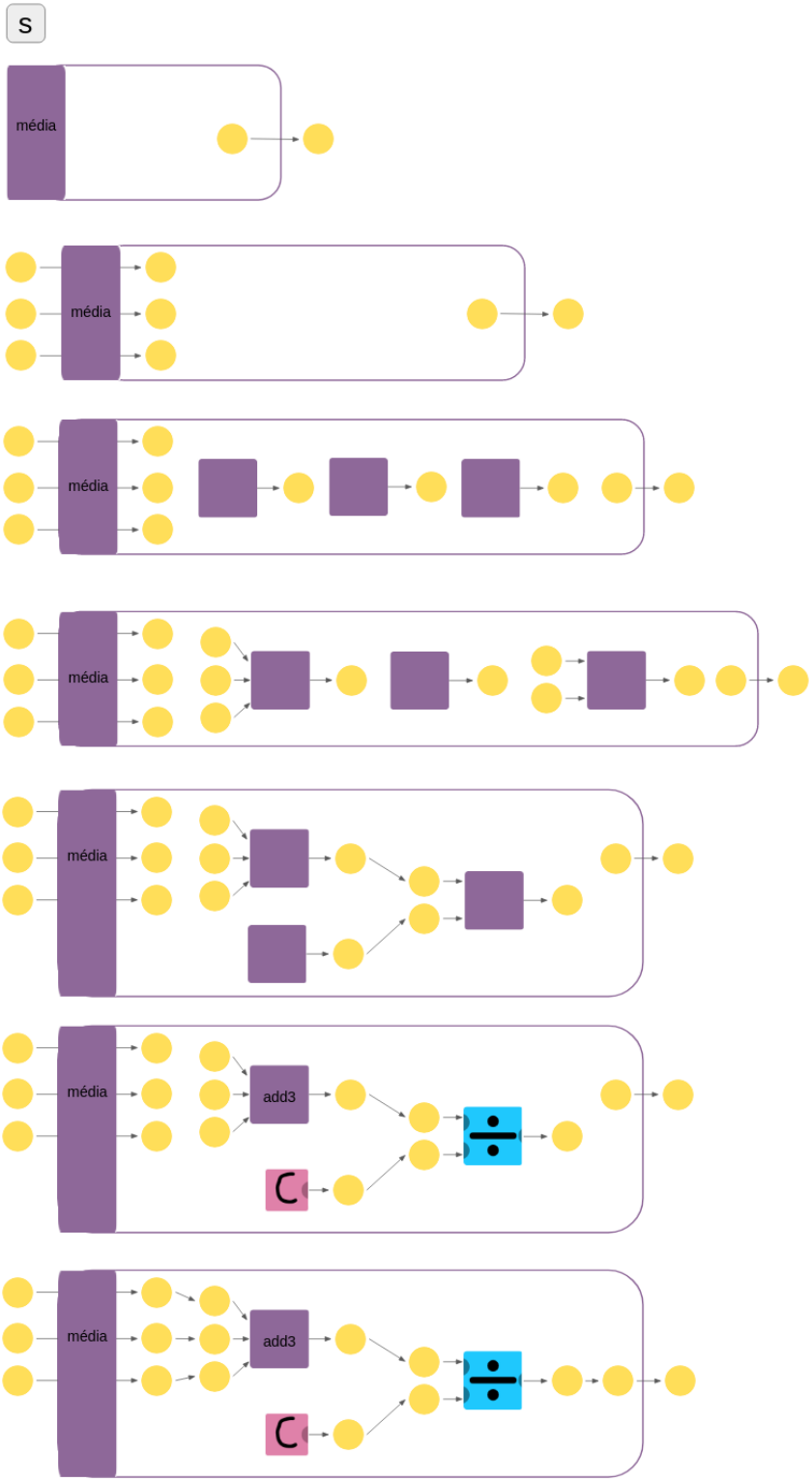
Fonte: Autora (2021)

da média de três valores. Para facilitar a construção do código utilizou-se a função add3 para soma de três valores, construída no exemplo anterior. A princípio um símbolo não terminal é utilizado e a regra (i) é aplicada, substituindo-o por uma definição da função média. Após, é aplicada a regra (iii) três vezes, adicionando à função três valores de entrada e três vezes a regra (iv) adicionando três funções ao corpo da função média. Em seguida é aplicada a regra (v) três vezes na primeira função e duas vezes na terceira função adicionando valores de entrada a essas funções. Logo, é aplicada a regra (viii) duas vezes, ligando a saída da primeira função com a primeira entrada da terceira função e a saída da segunda função com a segunda entrada da terceira função. Em seguida são aplicadas as regras (x) e (xiv) definindo a segunda função, como uma função que retorna um valor constante, a terceira função como uma função de divisão e a primeira função como a função add3. Por fim, são aplicadas as regras (vi) e (vii) e os valores de entrada da função média são conectados aos valores de entrada da função add3 e o valor de saída da função divisão é conectado à saída da função média. Assim, é criado um código que recebe três valores como entrada, realiza a soma deles, através da função add3 e divide esse valor pela constante 3, retornando assim a média das três entradas.

Na Figura 25 e na Figura 26 está representada a derivação de um código para cálculo da área de uma circunferência. A princípio se utiliza um símbolo não terminal e se aplica a ele a regra (i) transformando-o na definição da função área. Após, se aplica a regra (iii), que adiciona uma entrada à função área e a regra (iv) três vezes incluindo três funções no corpo da função área. Em seguida se aplica a regra (v) duas vezes nas duas primeiras funções, adicionando uma entrada a cada função. Logo, se aplica a regra (viii) duas vezes, conectando a saída da primeira função à primeira entrada da segunda função e conectando a saída da terceira função à segunda entrada da segunda função. Em seguida, se aplica a regra (xiii) nas duas primeiras funções, identificando-as como funções de multiplicação e a regra (x) na terceira função identificando-a como uma função que retorna um valor constante. Por fim, se aplicam as regras (vi) e (vii) conectando a entrada da função área nas duas entradas da primeira função de multiplicação e a saída da segunda função de multiplicação na saída da função área. Desse modo, é construído um código em Pandora que recebe um valor equivalente ao raio r de uma circunferência, realiza a função de multiplicação obtendo o valor de r^2 e multiplicando-o pela constante π, retornando na saída o valor da área dessa circunferência.

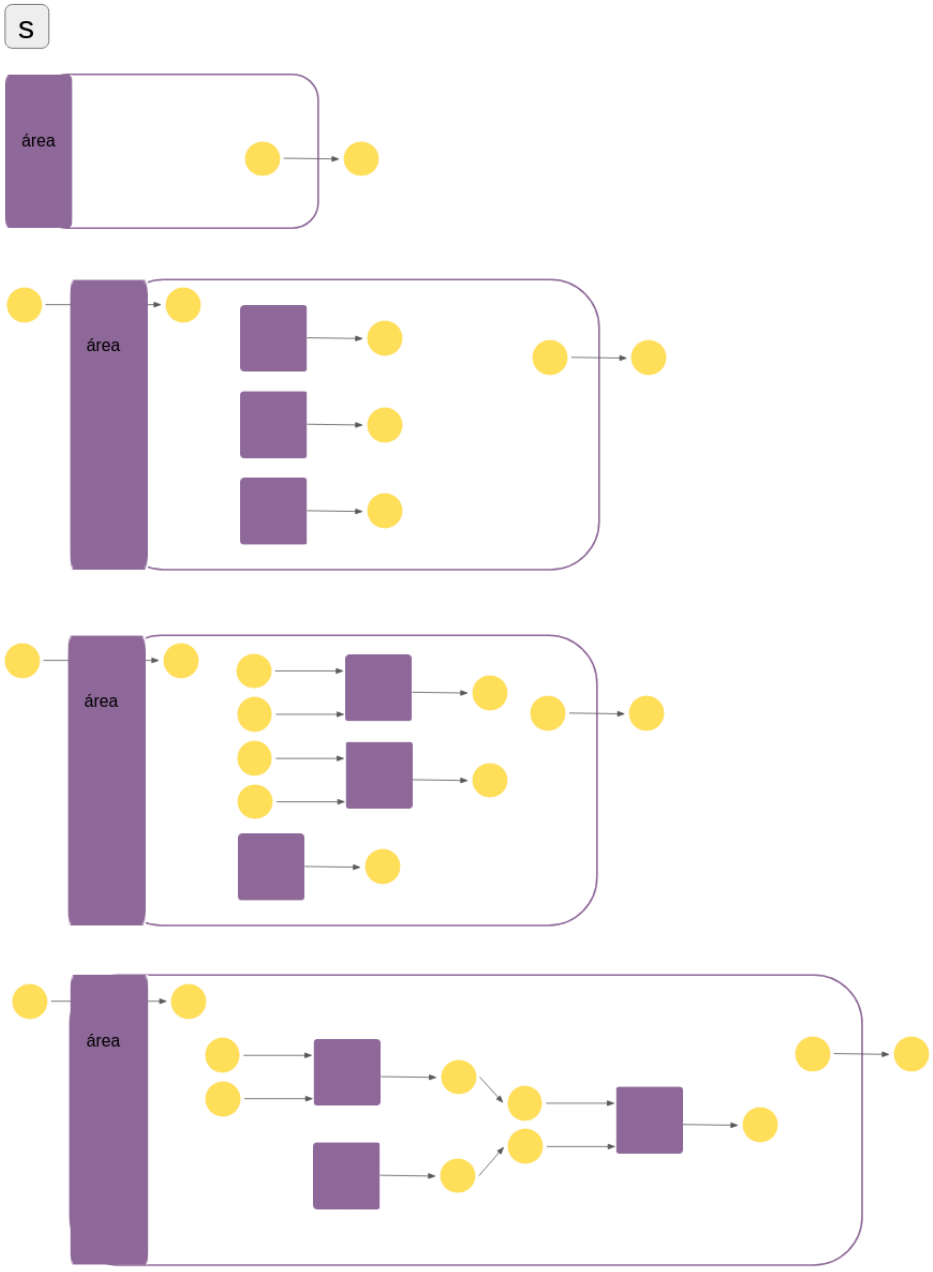
Na Figura 27, na Figura 28 e na Figura 29 é apresentado o exemplo da derivação de uma função que realiza o cálculo do fatorial de um número. Inicialmente existe apenas

Figura 24 – Derivação de código Pandora cálculo da média de três valores



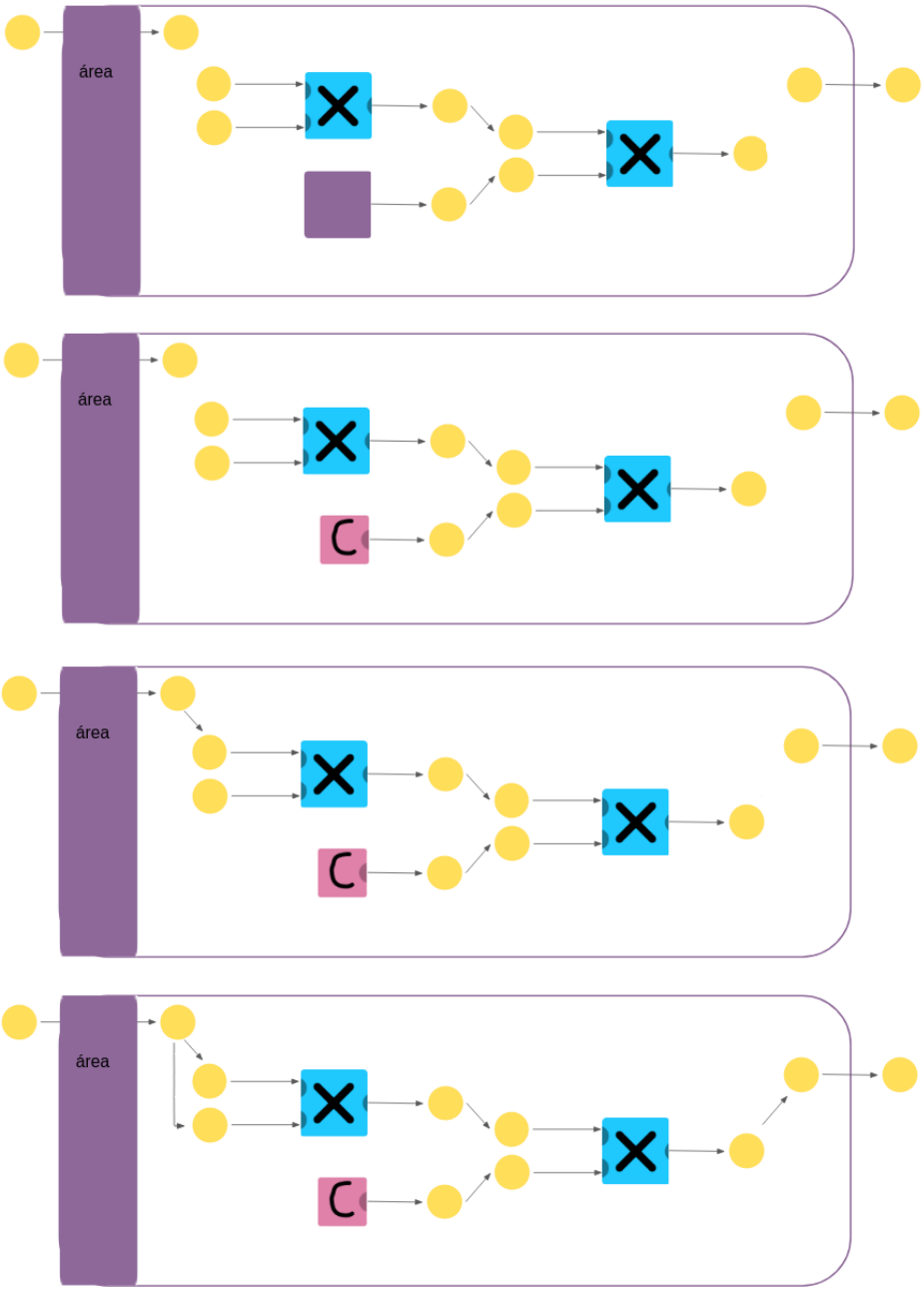
Fonte: Autora (2021)

Figura 25 – Derivação de código Pandora para cálculo da área de uma circunferência



Fonte: Autora (2021)

Figura 26 – Derivação de código Pandora para cálculo da área de uma circunferência cont.



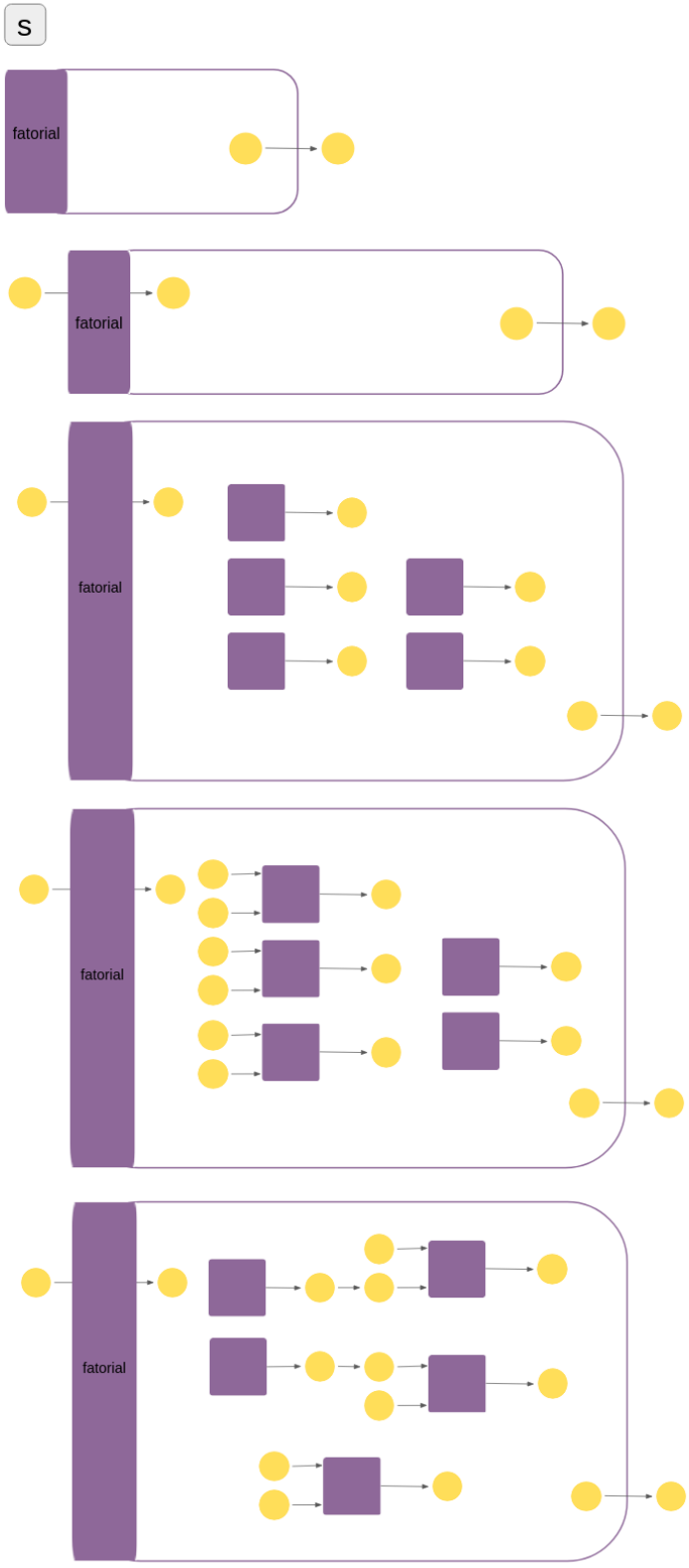
Fonte: Autora (2021)

um símbolo não terminal, ao aplicar a regra (i), o símbolo não terminal é substituído pela definição da função fatorial. Em seguida é aplicada a regra (iii) que adiciona um valor de entrada à função e cinco vezes a regra (iv) que adiciona cinco funções ao corpo da função fatorial. Após, é aplicada a regra (v) duas vezes nas três primeiras funções, adicionando dois valores de entrada em cada uma. Em seguida é aplicada duas vezes a regra (viii) conectando a saída de uma das funções que não possui entradas na segunda entrada da primeira função e a saída da outra função que não possui entradas na primeira entrada da segunda função. Logo, se aplicam as regras (xix), (v) e (xx) na segunda função, adicionando a estrutura das funções se/decisão na sua saída, acrescentando uma entrada à função do ramo F da função se e conectando a terceira função no ramo F da função se. Após, é aplicada novamente a regra (viii) conectando a saída da primeira função à entrada da função no ramo F da função se. Em seguida, se aplicam as regras (x), (xii), (xiii) e (xv), identificando as duas funções sem valores de entrada, bem como a função do ramo V da função se como funções que retornam valores constantes, a segunda função como uma função de igualdade, a primeira função como uma função de subtração e as outras funções no ramo F da função se como a própria função fatorial e uma função de multiplicação. Por fim aplicam-se as regras (vi) e (vii) conectando o valor de entrada às entradas das funções de igualdade, subtração e multiplicação e a saída da função de decisão à saída da função fatorial. Desse modo, obtém-se uma função que recebe como parâmetro um valor n, e o compara com uma constante igual a zero, se forem iguais o valor um é passado como saída da função fatorial; se os valores forem diferentes é subtraído um do valor n e esse valor de $(n-1)$ é passado como entrada de uma nova instância da função fatorial, sua saída é multiplicada pelo valor n e passado como saída da função fatorial.

5.4 Ferramenta Pandora

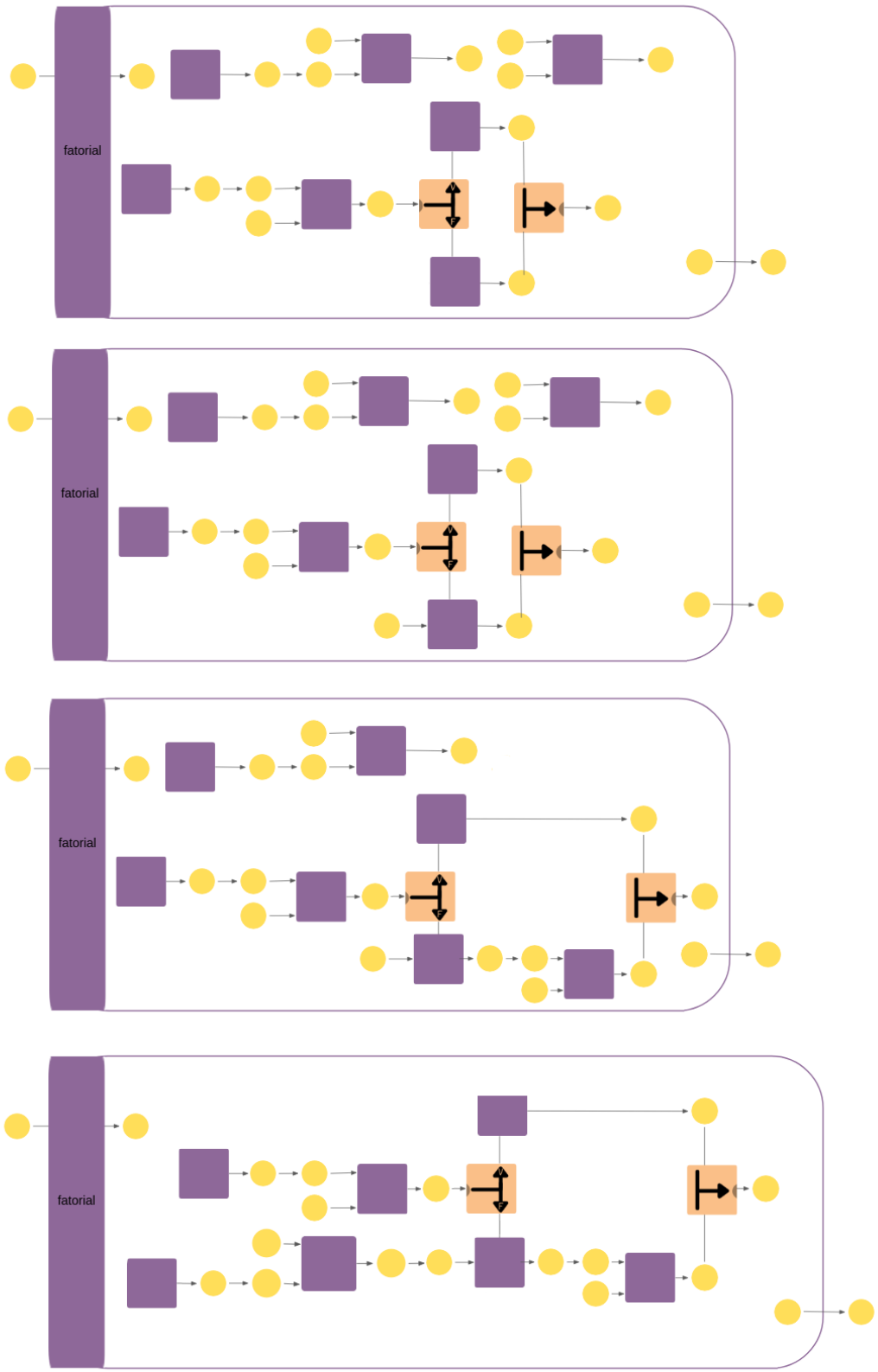
Foi considerada uma ferramenta para facilitar a programação na linguagem Pandora. Foram elaborados protótipos de telas da ferramenta, apresentados na Figura 30, na Figura 31 e na Figura 32 . A ferramenta disponibilizaria o conjunto de blocos padrão da linguagem Pandora permitindo ao usuário arrastá-los até a área para construção de programas e de novas funções. Essas funções seriam salvas e disponibilizadas como blocos em outra aba para que pudessem ser reutilizadas facilmente na construção de outras funções. A ferramenta também possibilitaria a tradução de um programa Pandora para uma linguagem textual.

Figura 27 – Derivação de código Pandora para cálculo de fatorial de forma recursiva



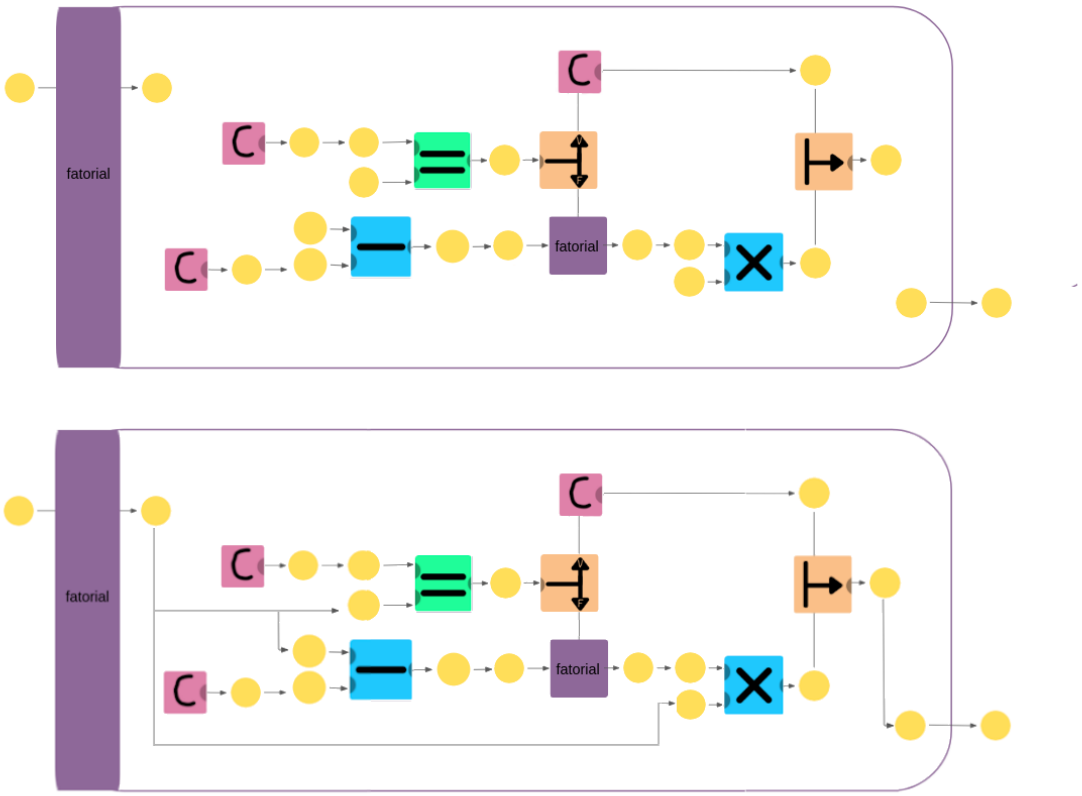
Fonte: Autora (2021)

Figura 28 – Derivação de código Pandora para cálculo de fatorial de forma recursiva cont.



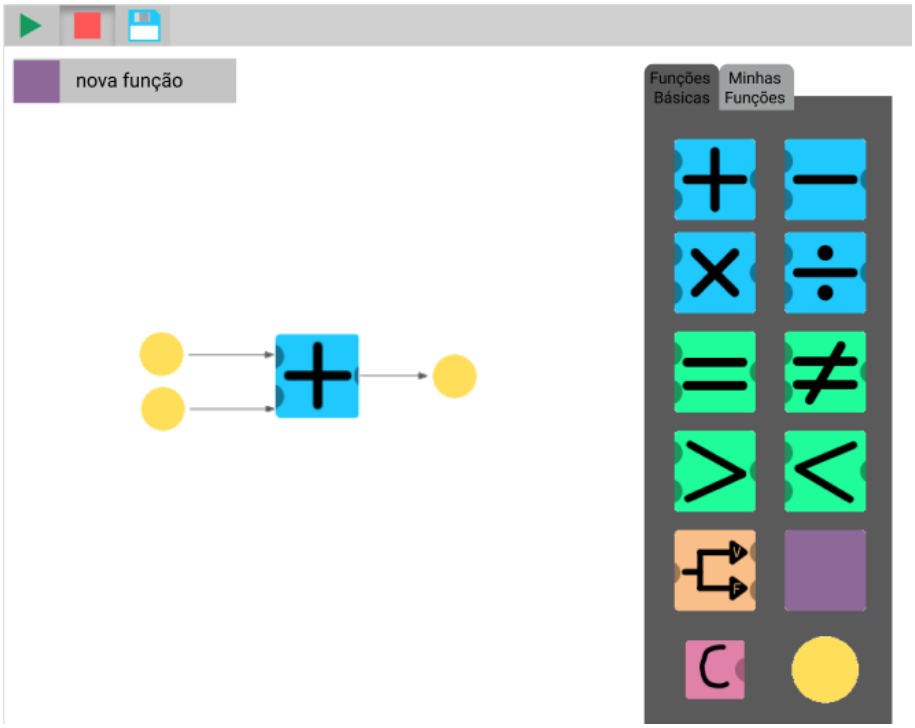
Fonte: Autora (2021)

Figura 29 – Derivação de código Pandora para cálculo de fatorial de forma recursiva cont.



Fonte: Autora (2021)

Figura 30 – Protótipo de ferramenta Pandora- Uso dos blocos padrão



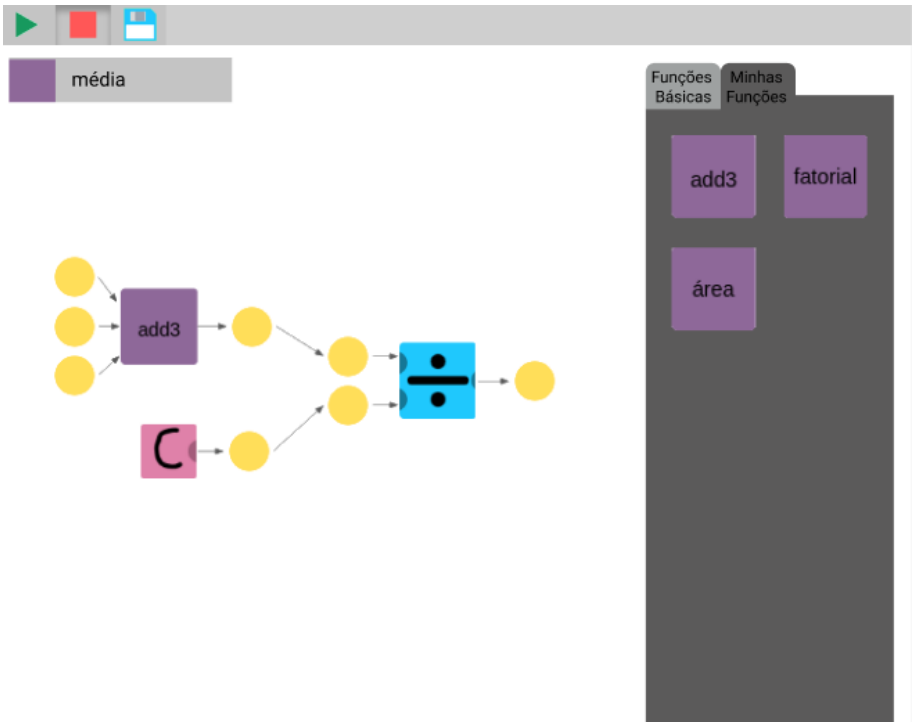
Fonte: Autora (2021)

A Figura 30 apresenta a tela da ferramenta e a construção de uma função chamada ‘nova função’ que utiliza a função adição disponível entre os blocos padrão da linguagem na aba ‘Funções Básicas’. A Figura 31 mostra a construção do código para cálculo da média de três valores na função ‘média’ essa função utiliza a função ‘add3’ previamente construída e disponibilizada na aba ‘Minhas Funções’. A Figura 32 mostra a execução da função ‘média’.

5.5 Atividades Desplugadas

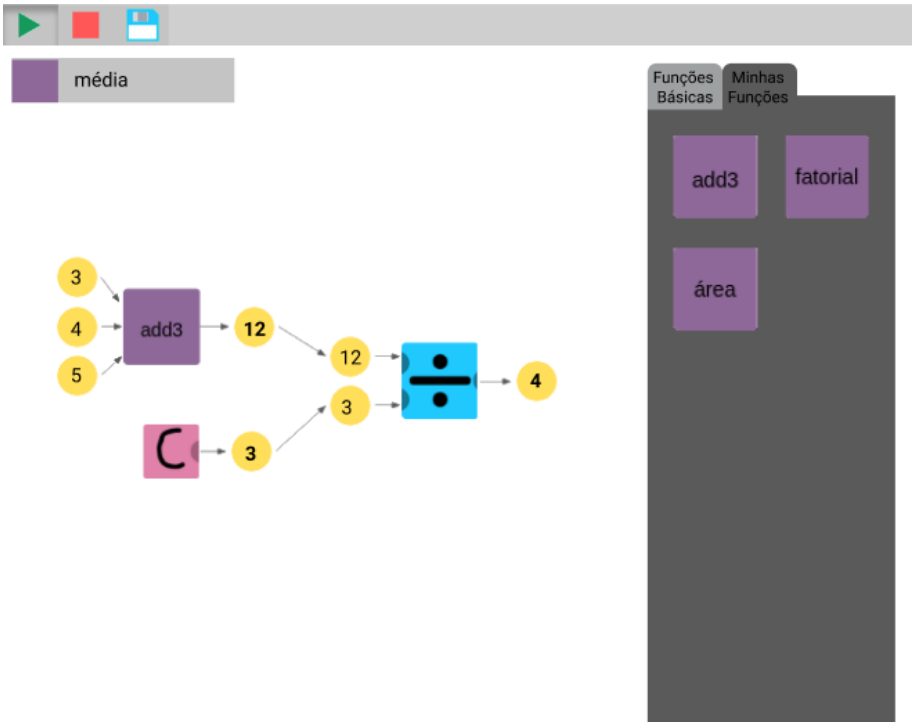
Está disponibilizado no apêndice A um material para que a Linguagem Pandora possa ser utilizada no ensino de programação de forma desplugadas. Estão disponíveis para impressão um quadro onde podem ser construídos os programas, com espaço para inserção do nome da função construída - de forma semelhante ao exemplificado na Ferramenta Pandora na seção anterior - e um espaço disponível para organização dos cartões que representam os blocos da linguagem. Assim, os alunos podem construir diversos códigos utilizando os cartões com os blocos impressos. As setas que conectam os blocos podem ser desenhadas com lápis ou caneta.

Figura 31 – Protótipo de ferramenta Pandora- Uso de funções do usuário



Fonte: Autora (2021)

Figura 32 – Protótipo de ferramenta Pandora- Execução do programa



Fonte: Autora (2021)

6 CONSIDERAÇÕES FINAIS

Seguindo os objetivos do trabalho, foi construída a linguagem Pandora. Uma linguagem de programação visual funcional Turing-completa com foco no ensino de programação. A linguagem pode ser utilizada por alunos de diversas faixas etárias, pois é composta por símbolos acessíveis e com pouca informação textual, buscando facilitar a compreensão da finalidade de cada função. A linguagem Pandora também tem como foco o reúso de código, facilitado pela representação das funções como símbolos geométricos que contém outras funções, permitindo ao usuário adicionar funções construídas anteriormente em novos códigos. O uso do paradigma funcional foi conveniente na construção da linguagem visual Pandora pois possibilitou a composição dos blocos em função da representação do encadeamento sequencial.

O protótipo de ferramenta Pandora idealizado neste trabalho tem como objetivo facilitar o desenvolvimento e a execução de programas Pandora possibilitando ao aluno a edição do código através de recursos como arrastar, soltar e conectar os blocos da linguagem, além de contribuir para o reúso de código através da aba de funções salvas que permite ao usuário localizar rapidamente e adicionar ao código funções previamente desenvolvidas por ele.

Pode-se utilizar a sintaxe da linguagem construída em sala de aula, tanto para o ensino de algoritmos e lógica de programação como para o primeiro contato dos alunos com uma linguagem funcional. Outra aplicação interessante para Pandora é em atividades desplugadas, principalmente em turmas de ensino fundamental. A linguagem Pandora também pode ser utilizada como forma de avaliar a efetividade do paradigma funcional em linguagens visuais voltadas para o ensino, visto que não são frequentes linguagens visuais que utilizam esse paradigma.

A partir da linguagem Pandora construída neste trabalho é possível desenvolver uma série de trabalhos futuros como a implementação da ferramenta Pandora descrita anteriormente, o desenvolvimento de um compilador Pandora e a incorporação de tipos abstratos de dados e orientação a objetos à linguagem. Além disso, podem ser desenvolvidos trabalhos que investiguem o potencial da linguagem Pandora de facilitar o aprendizado de alunos de algoritmos e programação avaliando a experiência dos alunos nessas disciplinas e sua taxa de aprovação, bem como seu desempenho em outras disciplinas que utilizem os conhecimentos básicos de lógica de programação e linguagens funcionais. Também seria interessante analisar a efetividade da linguagem Pandora em

diferentes níveis de ensino e em comparação com outras linguagens de programação utilizadas por estudantes.

REFERÊNCIAS

ANDRADE, G. et al. Metodologia didático simbólica como alternativa para o ensino de programação de computadores a alunos surdos. In: **Anais do XXVII Workshop sobre Educação em Computação**. Porto Alegre, RS, Brasil: SBC, 2019. p. 473–482. ISSN 2595-6175. Available from Internet: <https://sol.sbc.org.br/index.php/wei/article/view/6652>.

CARBAJAL, M.; BARANAUSKAS, M. Programação tangível e construção de significado: criação de símbolos para o ambiente taprec junto com professoras de ensino fundamental. In: **Anais do XXIV Workshop de Informática na Escola**. Porto Alegre, RS, Brasil: SBC, 2018. p. 323–332. ISSN 0000-0000. Available from Internet: <https://sol.sbc.org.br/index.php/wie/article/view/14344>.

DANTAS, I. et al. Ensino de lógica de programação no ensino fundamental utilizando o jogo robotizen: um relato de experiência. In: **Anais do XXVII Workshop sobre Educação em Computação**. Porto Alegre, RS, Brasil: SBC, 2019. p. 51–60. ISSN 2595-6175. Available from Internet: <https://sol.sbc.org.br/index.php/wei/article/view/6616>.

DENG, X. et al. Arcat: A tangible programming tool for dfs algorithm teaching. In: . New York, NY, USA: Association for Computing Machinery, 2019. (IDC '19), p. 533–537. ISBN 9781450366908.

FABRO, J. A. et al. Programming teaching using flowcharts in a simulated environment focused on introducing practical obr. In: **2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)**. [S.l.: s.n.], 2019.

FERREIRA, A. P. L.; RIBEIRO, L. Programação orientada a objeto com grafos. In: **Jornadas de Atualização em Informática (JAI)**. Campo Grande: Sociedade Brasileira de Computação SBC, 2006.

GOMES, A.; MENDES, A. J. Learning to program - difficulties and solutions. In: . [S.l.: s.n.], 2007. (International Conference on Engineering Education – ICEE 2007).

HATTORI, K.; HIRAI, T. An intuitive and educational programming tool with tangible blocks and ar. In: **SIGGRAPH '19: ACM SIGGRAPH 2019 Posters**. [S.l.: s.n.], 2019.

HORN, M. S.; JACOB, R. J. K. Designing tangible programming languages for classroom use. In: **Proceedings of 1st international conference on Tangible and embedded interaction**. [S.l.: s.n.], 2007.

JIN, Q. et al. Ar-maze: a tangible programming tool for children based on ar technology. In: **IDC '18: Proceedings of the 17th ACM Conference on Interaction Design and Children**. [S.l.: s.n.], 2018.

JORNADAS de Atualização em Informática 2006. Campo Grande: Sociedade Brasileira de Computação SBC, 2006.

KEENE, S. E. **Object-Oriented Programming in Common Lisp**. Addison-Wesley, 1989. ISBN 0201175894. Available from Internet: <http://cl-cookbook.sourceforge.net/clos-tutorial/>.

NETO, M. et al. Robótica educacional uma ferramenta para ensino de lógica de programação no ensino fundamental. In: **Anais do XXIV Workshop de Informática na Escola**. Porto Alegre, RS, Brasil: SBC, 2018. p. 315–322. ISSN 0000-0000. Available from Internet: <https://sol.sbc.org.br/index.php/wie/article/view/14343>.

PASTERNAK, E.; FENICHEL, R.; MARSHALL, A. N. Tips for creating a block language with blockly. In: **2017 IEEE Blocks and Beyond Workshop (B&B)**. [S.l.: s.n.], 2017.

PIMENTA, J. M. M. **Temple - Uma linguagem de Programação para o Ensino de Programação**. Dissertation (Mestrado), Évora, 2019.

PINTO-LLORENTE, A. M. et al. Developing computational thinking via the visual programming tool: lego education wedo. In: **TEEM '16: Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality**. [S.l.: s.n.], 2016.

PRESSMAN, R. S. **Engenharia de Software: Uma abordagem profissional**. 8. ed. Porto Alegre: Bookman, 2016. 968 p.

PRICE, T. W. et al. Evaluation of a frame-based programming editor. In: **Proceedings of the 2016 ACM Conference on International Computing Education Research**. New York, NY, USA: Association for Computing Machinery, 2016. (ICER '16), p. 33–42. ISBN 9781450344494.

SARKAR, S. P.; SARKER, B.; HOSSAIN, S. K. A. Cross platform interactive programming learning environment for kids with edutainment and gamification. In: **2016 19th International Conference on Computer and Information Technology (ICCIT)**. [S.l.: s.n.], 2016.

SEBESTA, R. W. **Conceitos de linguagens de programação**. 11. ed. Porto Alegre: Bookman, 2018. 765 p.

SERAJ, M.; AUTEXIER, S.; JANSSEN, J. Beesm, a block-based educational programming tool for end users. In: **NordiCHI '18: Proceedings of the 10th Nordic Conference on Human-Computer Interaction**. [S.l.: s.n.], 2018.

SOUZA, D. et al. Lightbot logicamente: um game lúdico amparado pelo pensamento computacional e a matemática. In: **Anais do XXIV Workshop de Informática na Escola**. Porto Alegre, RS, Brasil: SBC, 2018. p. 61–69. ISSN 0000-0000. Available from Internet: <https://sol.sbc.org.br/index.php/wie/article/view/14317>.

SOUZA, S. S.; CASTRO, T. H. C. Investigação em programação com scratch para crianças: uma revisão sistemática da literatura. In: **Anais dos Workshops do V Congresso Brasileiro de Informática na Educação (CBIE 2016)**. [S.l.: s.n.], 2016.

TERAN, L.; ARAúJO, F.; PIRES, Y. Elis: Uma ferramenta inclusiva para o ensino de lógica de programação aos surdos. In: **Anais do XXV Workshop de Informática na**

Escola. Porto Alegre, RS, Brasil: SBC, 2019. p. 1024–1033. ISSN 0000-0000. Available from Internet: <https://sol.sbc.org.br/index.php/wie/article/view/13252>.

THOMPSON, S. **Haskell: The Craft of Functional Programming**. 2. ed. Addison-Wesley, 1999. 507 p. ISBN 0-201-34275-8. Available from Internet: <https://www.haskell.org/>.

TOLEDO, B. et al. Desenvolvimento de um jogo educacional para o ensino de programaÇÃo bÁsica. In: ____. [S.l.: s.n.], 2020. p. 112–125. ISBN 9786557065334.

TOURETZKY, D. S. **COMMON LISP: A Gentle Introduction to Symbolic Computation**. Redwood City: The Benjamin/Cummings Publishing Company, Inc., 1990. 587 p. Available from Internet: <http://www.inf.ufsc.br/~aldo.vw/func/touretzky/touretzky.pdf>.

VAHLDICK, A.; FARAH, P. A blocks-based serious game to support introductory computer programming in undergraduate education. **Computers in Human Behavior Reports**, 2020.

WATANABE, T. et al. Analyzing viscuit programs crafted by kindergarten children. In: . New York, NY, USA: Association for Computing Machinery, 2020. (ICER '20), p. 238–247. ISBN 9781450370929.

WING, J. M. Computational thinking and thinking about computing. **Philosophical Transactions of the Royal Society**, 2008.

WING, J. M. **Computational thinking benefits society**. New York: Academic Press, 2014.

ZANETTI, H. A. P.; BORGES, M. A. F.; RICARTE, I. L. M. Pensamento computacional no ensino de programação: Uma revisão sistemática da literatura brasileira. In: **Anais do XXVII Simpósio Brasileiro de Informática na Educação (SBIE 2016)**. [S.l.: s.n.], 2016.

ZHANG, K.; ZHANG, K. B. Graph grammars for visual programming. In: ____. **Software Visualization**. Boston, MA: Springer, 2003. (The Springer International Series in Engineering and Computer Science, v. 734).

APÊNDICE A – MATERIAIS PARA APLICAÇÃO DE ATIVIDADES DESPLUGADAS COM PANDORA

Sugestões de atividades:

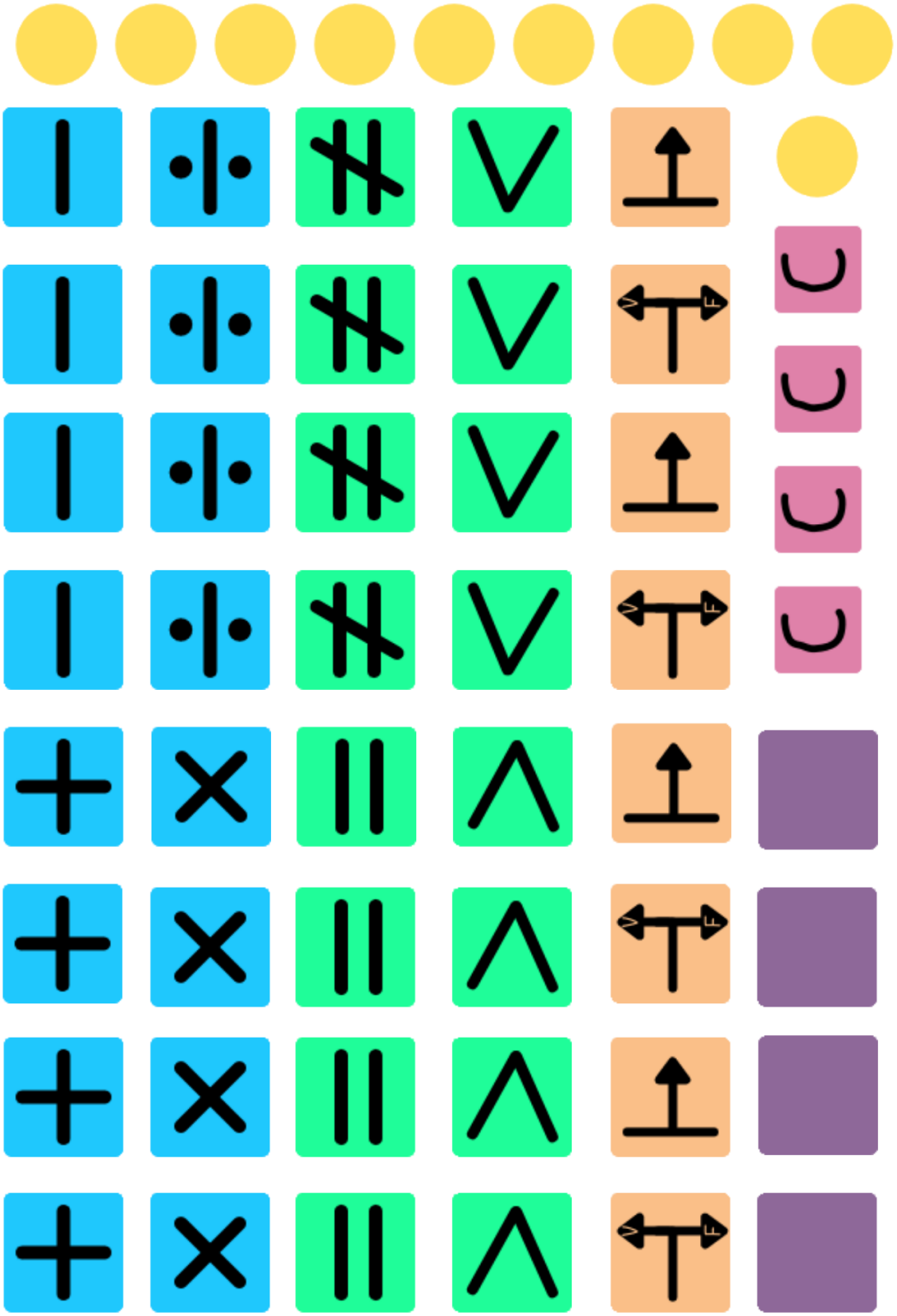
Atividade 1 : Utilizando a linguagem Pandora, crie um programa que calcule quantas horas por semana um aluno estuda programação, considerando que ele estuda todos os dias de segunda a sábado.

Atividade 2 : Utilizando a linguagem Pandora, crie um programa que realize a soma de três valores.

Atividade 3 : Um professor deseja calcular a média das notas de seus alunos. Ele aplicou três provas diferentes durante o semestre. Utilizando a função desenvolvida na Atividade 2, crie um programa Pandora que faça esse cálculo para o professor.

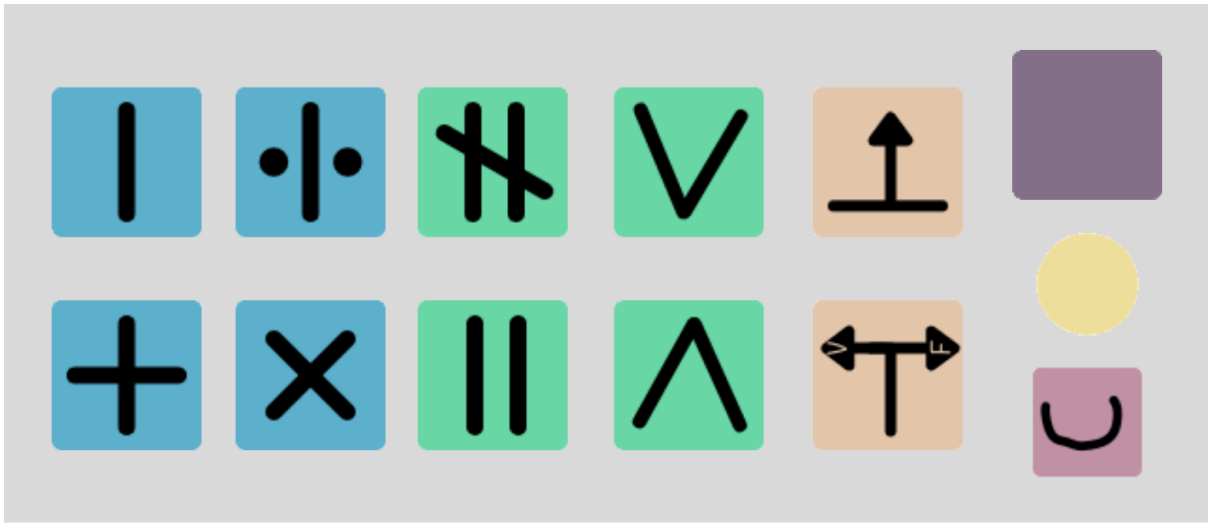
Atividade 4 : Além de saber as médias de seus alunos, o professor também gostaria de saber quais alunos estão aprovados na disciplina. Crie um programa Pandora, que mostre o número 0 quando o aluno está reprovado e o número 1 quando o aluno está aprovado. Considere como aprovado um aluno com notas maiores que 6.

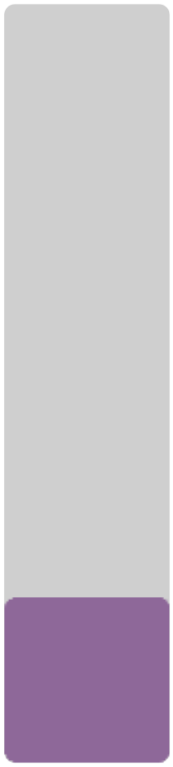
Figura 33 – Blocos Pandora para recortar



Fonte: Autora (2021)

Figura 34 – Quadro para construção de programas





Fonte: Autora (2021)