

SMALG PLATFORM – UMA PLATAFORMA EDUCACIONAL

Adriner Maranhão de Andrade, Dalton Solano dos Reis – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

adrinera@furb.br, dalton@furb.br

Resumo: Este artigo detalha o desenvolvimento de uma ferramenta para auxiliar no ensino de Ciência da Computação, em disciplinas como algoritmos e estruturas de dados, através de recursos de visualização, codificação escrita e produção de material de ensino. Para validar a aplicabilidade da ferramenta foram realizadas oficinas com alunos do segundo semestre de Ciência da Computação da FURB e testes com dois professores, também da área de Ciência da Computação da FURB, sendo um responsável pela disciplina de Algoritmos e Estrutura de Dados e outro responsável pela disciplina de Introdução à Programação. Com base nestes testes, a ferramenta demonstrou que possui potencial para auxiliar no aprendizado de estudantes ao mesmo tempo que proporciona aos professores uma maneira de produzir conteúdo de ensino conforme as suas necessidades. A visualização e a interação por meio da codificação demonstraram ser componentes importantes neste processo.

Palavras-chave: Ciência da computação. Plataforma. Ferramenta. Ensino. Educação.

1 INTRODUÇÃO

A partir de uma perspectiva educacional, a Ciência da Computação pode-se demonstrar desafiadora, resultando em um difícil aprendizado, principalmente para aqueles que estão iniciando na área (QIAN; LEHMAN, 2017). Ela possui altos índices de desistência entre os estudantes, sendo que a maioria dos abandonos ocorre nos primeiros dois anos de estudos (GIANNAKOS *et al.*, 2017). Ainda segundo Giannakos *et al.* (2017), dentre as causas levantadas estavam a baixa qualidade de ensino, além da grade rigorosa e demandas densas. Segundo Qian e Lehman (2017, p. 5, tradução nossa) “problemas na compreensão conceitual de programação dos alunos podem levar a profundos e significantes equívocos relacionados ao modelo mental de execução de código e de sistemas de computadores”.

A utilização de ferramentas visuais que ilustram os conceitos de programação e o funcionamento do código têm sido úteis para facilitar neste processo educacional, auxiliando no entendimento dos estudantes (QIAN; LEHMAN, 2017, p. 12). Isso porque na Ciência da Computação, de acordo com Fouh *et al.* (2012, p. 96, tradução nossa) “muitos dos seus conceitos centrais referem-se a abstrações. Não se pode ver ou tocar um algoritmo ou uma estrutura de dados [...]”.

O uso de tipos abstratos, como interfaces, juntamente com o uso de testes automatizados é uma estratégia a se oferecer para a formulação de uma ferramenta auxiliar ao ensino. Interfaces são consideradas tipos de dados abstratos com uma especificação formal de comportamento, permitindo uma estrutura desacoplada de sua implementação (STEIMANN; MAYER, 2005). Esta estrutura pode então ser utilizada no desenvolvimento de testes, sendo estes responsáveis pela definição de um plano de execução e validação de um código que ainda não possui implementação real, conceito conhecido como *test-first* (NAIK; TRIPATHY, 2008; FUCCI *et al.*, 2017).

Diante do exposto, o objetivo desse trabalho foi desenvolver uma plataforma que servirá como uma ferramenta auxiliar no ensino da Ciência da Computação, em disciplinas como Algoritmos e Estrutura de Dados, utilizando-se de recursos de visualização, codificação escrita e produção de material de ensino. Os objetivos específicos são: possibilitar ao professor a produção de conteúdo de ensino por meio da abstração; criar uma *engine* que gerará a partir do código desenvolvido uma representação visual da execução; definir uma estrutura formal que funcionará como um Software Development Kit (SDK) para utilização da ferramenta.

Para avaliar o conceito e a aplicabilidade da ferramenta, foi realizado um teste de usabilidade monitorado com um usuário, oficinas com quatro estudantes do segundo semestre de Ciência da Computação e apresentações para dois professores de Ciência da Computação da FURB, responsáveis pelas disciplinas de Algoritmos e Estrutura de Dados e Introdução à Programação.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresentará a base teórica envolvida no desenvolvimento da ferramenta. Na primeira seção serão tratados os conceitos e conteúdos bibliográficos que fundamentam o trabalho. Na segunda seção serão apresentados os trabalhos correlatos com o desenvolvido neste artigo.

2.1 CONCEITOS

Nesta seção serão descritos os assuntos que fundamentam o desenvolvimento da plataforma: dificuldades no ensino da Ciência da Computação; o uso da visualização no ensino; abstração na programação e interfaces; e conceitos relacionados a utilização de testes.

2.1.1 Dificuldades no ensino da Ciência da computação

Todo estudante de Ciência da Computação deve aprender os conceitos básicos de Ciência da Computação e Programação, mas em muitos casos isso pode ser desafiante, principalmente para alunos que estão nos anos iniciais e nunca tiveram um contato com a área (DEBABI; BENSEBAA, 2016). Um outro levantamento mostrou uma correlação entre o esforço do aluno e a desistência, destacando que alunos que aplicavam um bom índice de esforço, e consequentemente persistência, eram os que permaneciam com os estudos (GIANNAKOS *et al.*, 2016).

Muitos fatores podem contribuir para dificultar o aprendizado e consequentemente o ensino, dentre eles está a formulação de modelos mentais imprecisos por parte do estudante (QIAN; LEHMAN, 2017). Ainda de acordo com Qian e Lehman (2017, p. 5), o estudante pode enfrentar dificuldades até mesmo no entendimento de estruturas fundamentais como condicionais e estruturas de repetições, não compreendendo perfeitamente os critérios para execução ou o funcionamento. Ambas as estruturas são básicas e utilizadas para a construção de algoritmos no modelo procedural. Outra questão importante, é que os estudantes apresentam dificuldades em como desenvolver a solução para um problema e em como criar situações hipotéticas e excepcionais, para então encontrar situações de erro (DEBABI; BENSEBAA, 2016, p. 129).

Esse problema se estende também para conceitos não materializáveis onde não existem analogias diretas na realidade. É possível dizer que o primeiro passo para se aprender programação é partindo de um pensamento humanizado, relacionado com a vida real, para então se adentrar aos conceitos da programação (DEBABI; BENSEBAA, 2016).

Diante dessa realidade, o ensino na área de Algoritmos e Estruturas de Dados já tem recebido atenção. Um estudo de Danielsiek *et al.* (2012), aponta que alunos do primeiro ano da Universidade Técnica de Dortmund demonstraram equívocos na disciplina durante a realização de exames. Dentre os resultados, estava o algoritmo de ordenação *heapsort* com acerto médio de 43%. Realizaram então um levantamento contendo temas como as estruturas de dados *heap* e *binary tree* (árvore binária). Ao formular testes e apresentarem aos alunos, identificaram novamente outros equívocos, caracterizados, por exemplo, por confusões entre as definições *binary tree* com *search tree* (árvore de busca) e *heap* (DANIELSIEK, 2012).

2.1.2 O uso da visualização no ensino

Muitos dos conteúdos centrais da área de Ciência de Computação estão relacionados com abstrações, contribuindo para certas dificuldades no aprendizado. Algumas estratégias então, têm sido adotadas para facilitar neste processo, entre elas o recurso da visualização.

Boas visualizações de algoritmos trazem algoritmos a vida graficamente representando os seus vários estados e animando a transição entre eles. Eles ilustram estruturas de dados em um caminho natural e abstrato ao invés de focar em endereços de memória e chamadas de funções (FOUH *et al.* 2012, p. 96, tradução nossa).

De acordo com Sorva *et al.* (2013, p. 209, tradução nossa), “uma visualização de como um programa de computador é executado pode mostrar o que está normalmente escondido, implícito e automático em algo focal, explícito e controlável”. Ilustrações podem ser realizadas por professores em quadros ou softwares de desenho durante as aulas, mas isso normalmente toma tempo limitando a quantidade de cenários que podem ser cobertos com a visualização. Sendo assim, têm se buscado através de ferramentas de visualização a automatização ou semi-automatização desse processo (SORVA *et al.*, 2013, p. 209 - 210).

Estudos experimentais a respeito da eficiência desse método sugerem que alunos que se engajam com a ferramenta, possuindo um tipo de atividade mais ativa, apresentam melhores resultados do que os estudantes que utilizam a visualização somente passivamente (YUAN *et al.*, 2010). Complementando, Yuan *et al.* (2010) afirma que não interessa então o quão bem a ferramenta foi desenhada, ela não terá uma efetividade tão grande se não proporcionar a possibilidade de um engajamento ativo do estudante. O modo que o estudante interage com a visualização é mais importante do que as técnicas de visualização utilizadas (SORVA *et al.*, 2013). O conceito de interação não se resume a botões e cliques, mas em ações que o estudante pode realizar alterando o comportamento da visualização.

2.1.3 Abstração na programação e interfaces

Conforme Ganascia (2015, p. 28, tradução nossa) afirma, “a abstração desempenhou um papel crucial na computação”. No princípio, seu propósito foi criar uma generalização descritiva dos dados de um programa de modo que pudessem ser trabalhados sem se preocupar em qual máquina seriam executados, ainda que cada máquina tivesse as suas próprias particularidades (GANASCIA, 2015). Uma demonstração da importância da abstração no desenvolvimento da área de computação é o fato que, de acordo com Ganascia (2015, p. 28, tradução nossa), “computadores são compostos por diversos dispositivos diferentes conectados de uma maneira tão intrínseca e complexa que seria impossível isolá-los na mente humana para representar seu comportamento”. A abstração surge então como forma de encapsulamento, ocultando informações específicas e disponibilizando informações genéricas (COLBURN; SHUTE, 2007).

Partindo deste princípio, no meio da programação existem as definições de tipos de dados abstratos. Tipos de dados, como números e vetores, são definições de estruturas e de operadores de como um dado será manipulado. A inclusão da abstração em um tipo caracteriza a possibilidade de uma definição estrutural e comportamental sem a necessidade de referenciar uma implementação real (COLBURN; SHUTE, 2007; GANASCIA, 2015).

Dentro do grupo de tipos de dados abstratos encontram-se as *interfaces*, que são especificações formais de comportamento, indicando quais operações podem ser realizadas para um determinado tipo. Apesar de sua estrutura estar desacoplada de sua implementação, toda implementação realizada em cima de uma interface deve seguir suas especificações comportamentais (STEIMANN; MAYER, 2005). De acordo com Steimann e Mayer (2005, p.79), os operadores são normalmente restringidos a métodos, mas conceitualmente é possível a definição de atributos.

2.1.4 Conceitos relacionados a utilização de testes

A utilização de testes vai além de somente garantir a execução de um programa sem erros. Naik e Tripathy (2008, p. 7-8) destacam dois conceitos existentes em um teste: a *validação* e a *verificação*. A validação consiste na confirmação de que um determinado código atende ao seu propósito. A verificação consiste em determinar se o código satisfaz as especificações definidas antes do início de seu desenvolvimento, como estruturas de código (NAIK; TRIPATHY, 2008).

Existem diferentes abordagens para se desenvolver testes, dentre elas, está o desenvolvimento do teste antes do código a ser construído, conceito conhecido como *test-first* (FUCCI *et al.*, 2017, p. 597). Com este conceito, de acordo com Sommerville (2011, p. 47), “implicitamente se define uma interface e uma especificação de comportamento para uma funcionalidade a ser desenvolvida”. Essa afirmação reforça a possibilidade de implementação de testes através de tipos de dados abstratos. Em casos no qual já existe a definição de uma interface explícita, conseqüentemente, ocorre apenas a especificação e validação de um comportamento.

2.2 TRABALHOS CORRELATOS

Nesta seção serão abordados três trabalhos que podem ser utilizados para auxiliar no aprendizado de Ciência Computação. O primeiro trabalho, de Halim *et al.* (2012) descrito no Quadro 1, se trata de uma ferramenta web que disponibiliza um ambiente unificado e interativo para visualização de algoritmos. O segundo trabalho, de Klopfer *et al.* (2009) descrito no Quadro 2, se trata de uma ferramenta de simulação que apresenta uma representação visual através da programação em blocos. O terceiro trabalho, de Kopsch (2016) descrito no Quadro 3, se trata de uma ferramenta web adaptativa para auxiliar no aprendizado de programação.

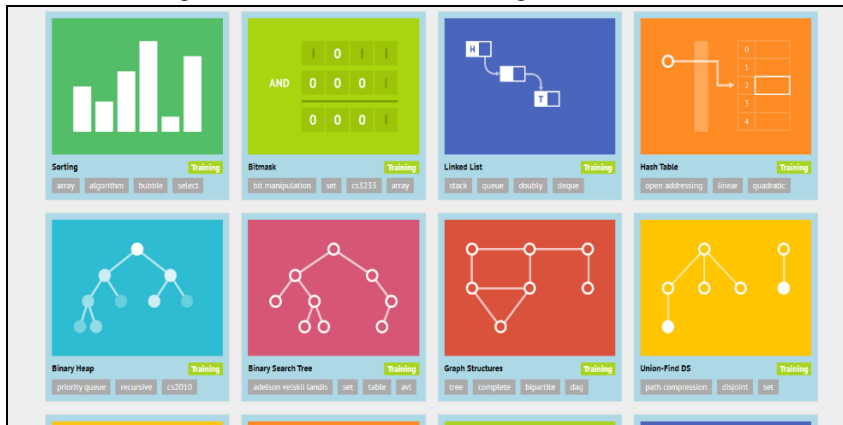
Quadro 1 – Visualgo

Referência	Halim <i>et al.</i> (2012)
Objetivos	Disponibilizar o aprendizado autodidático do aluno de vários algoritmos clássicos e não clássicos da área.
Principais funcionalidades	Acompanhamento passo a passo de execução; visualização gráfica; plataforma unificada com diversos algoritmos diferentes.
Ferramentas de desenvolvimento	HTML5, Canvas e JavaScript.
Resultados e conclusões	A ferramenta foi disponibilizada para dois grupos de usuários e aplicado um questionário. No primeiro grupo, composto por estudantes que participam de programação competitiva (24 respostas), 54,1% responderam que a ferramenta ajudou a entender melhor o algoritmo enquanto 45,8% responderam neutro (já conheciam o algoritmo, preferem análises a visualizações, ...). No segundo grupo, composto por estudantes que participam de aulas de aceleração de algoritmos e estruturas de dados (7 respostas), 71% responderam que a ferramenta ajudou a entender o algoritmo enquanto 28,5% responderam neutro. Nenhum dos alunos responderam que a ferramenta não ajudou a compreender melhor o algoritmo.

Fonte: elaborado pelo autor.

A ferramenta propõe um ambiente unificado, disponibilizando alguns algoritmos para padronizar o método de ensino. Desse modo, diminui-se a necessidade do estudante de adaptar-se às particularidades de cada ferramenta utilizada para este propósito (HALIM *et al*, 2012). A Figura 1 demonstra a listagem dos algoritmos disponibilizados na plataforma, agrupados por módulos.

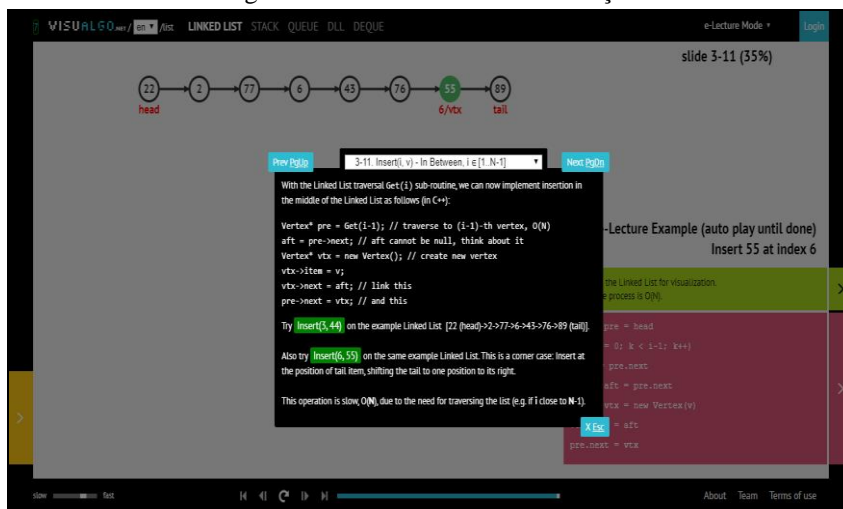
Figura 1 – Tela inicial com a listagem dos módulos



Fonte: digitalizado pelo autor a partir de Halim *et al*. (2012).

Ao selecionar um módulo para estudo, é iniciado por padrão o modo assistido, que dá ao usuário explicações sobre o algoritmo em questão, como o seu conceito e a lógica de seu funcionamento. A execução neste modo é de caráter progressivo intercalando entre explicações sobre o algoritmo, partes conceituais e execuções de pseudocódigos passo a passo, tendo no canto superior direito uma indicação do progresso efetuado naquele conteúdo, conforme ilustrado na Figura 2. O foco da ilustração é mostrar a organização dos elementos. É possível também utilizar o modo exploratório, em que o cenário fica livre para alterações a partir de ações pré-definidas e específicas para cada algoritmo, sem a possibilidade de qualquer tipo de codificação.

Figura 2 – Modo assistido de execução



Fonte: digitalizado pelo autor a partir de Halim *et al*. (2012).

Quadro 2 – Starlogo TNG

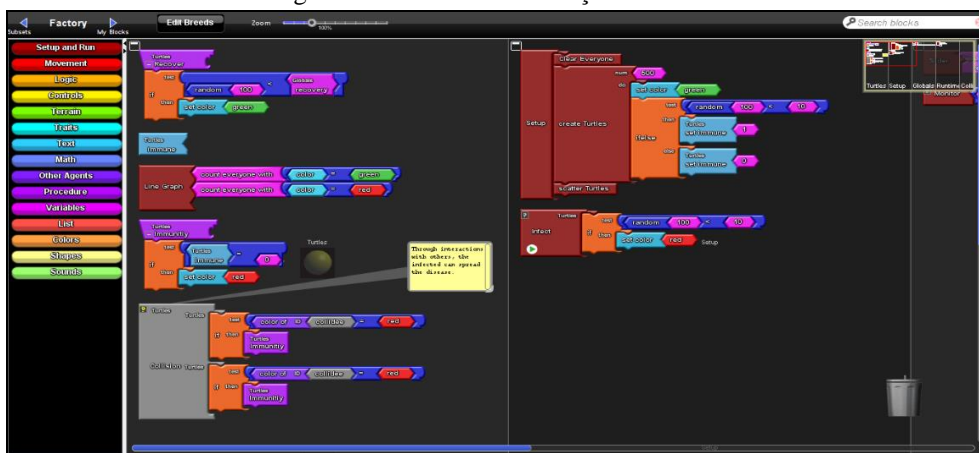
Referência	Klopper <i>et al</i> . (2009)
Objetivos	Permitir alunos e professores do ensino médio modelar sistemas descentralizados através da programação baseada em agentes.
Principais funcionalidades	Criação de cenários gráficos; flexibilidade para formular cenários; representação visual da execução; codificação em bloco.
Ferramentas de desenvolvimento	Não informado.
Resultados e conclusões	Foram realizados experimentos com alunos do ensino fundamental aplicando a metodologia de simulações e jogos. Eram apresentados cenários com uma descrição e um problema a ser resolvido, de modo que a interação com a ferramenta e a coleta de informações por meio da execução e visualização ajudassem o aluno a encontrar a solução. Após a aplicação dos experimentos, de acordo com Klopper <i>et al</i> . (2009, p.93, tradução nossa), foi concluído que “O

uso de jogos, simulações e programação proporcionam o potencial de promover o entendimento do estudante, [...] e engajando estudantes em ciência, tecnologia, engenharia e conteúdo matemático”.

Fonte: elaborado pelo autor.

A ferramenta para aprendizado de programação criada por Klopfer *et al.* (2009), foi desenvolvida para plataforma desktop, e disponibiliza um ambiente interativo ao estudante, possibilitando a construção de sistemas baseados em agentes através de blocos lógicos e visuais, que simplificam a codificação. O método de ensino consiste na criação de um “ciclo de simulação” que combina a engenharia de projeto com o método científico, sendo que a etapa de engenharia consiste no planejamento e construção do cenário, tendo como característica científica a observação e coleta de dados do resultado da execução, gerando novas questões que resultam em alterações no cenário reiniciando o ciclo de aprendizado (KLOPFER *et al.*, 2009). A Figura 3 ilustra a tela de codificação da ferramenta.

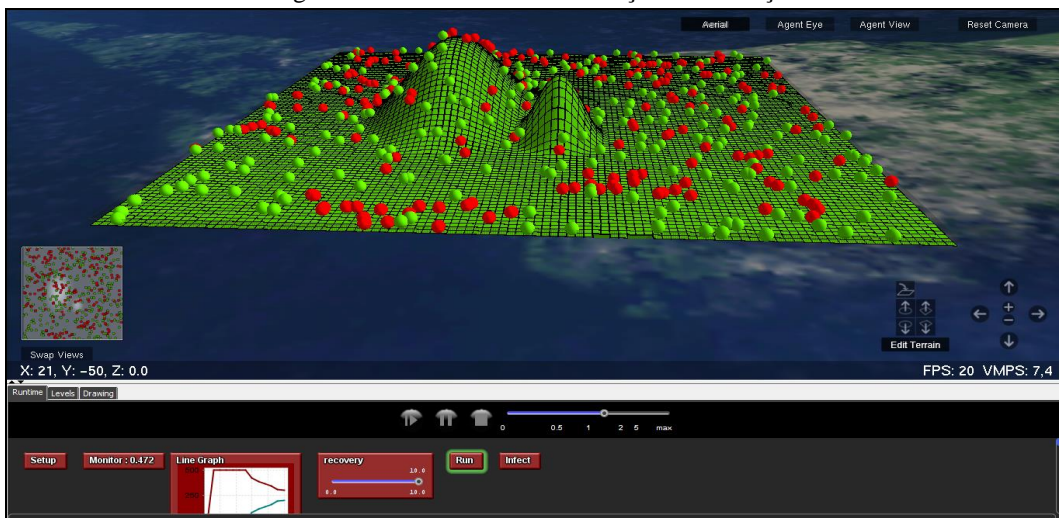
Figura 3 – Janela de codificação de blocos



Fonte: digitalizado pelo autor a partir de Klopfer *et al.* (2009).

Todo o código programado será então representado visualmente de modo que o estudante possa compreender e coletar informações a respeito do que foi implementado. A possibilidade do estudante expressar e explorar os seus próprios entendimentos é algo não proporcionado por visualizações estáticas (KLOPFER *et al.*, 2009). O caráter livre e interativo da ferramenta possibilita a formulação de problemas que se adequem a momentos situacionais no ensino. Ao detectar uma determinada necessidade, professores podem, por exemplo, sugerir desafios aos seus alunos e auxiliá-los neste processo. A Figura 4 demonstra o ambiente de visualização da execução.

Figura 4 – Ambiente de visualização da execução



Fonte: digitalizado pelo autor a partir de Klopfer *et al.* (2009).

Quadro 3 – Furbot Web

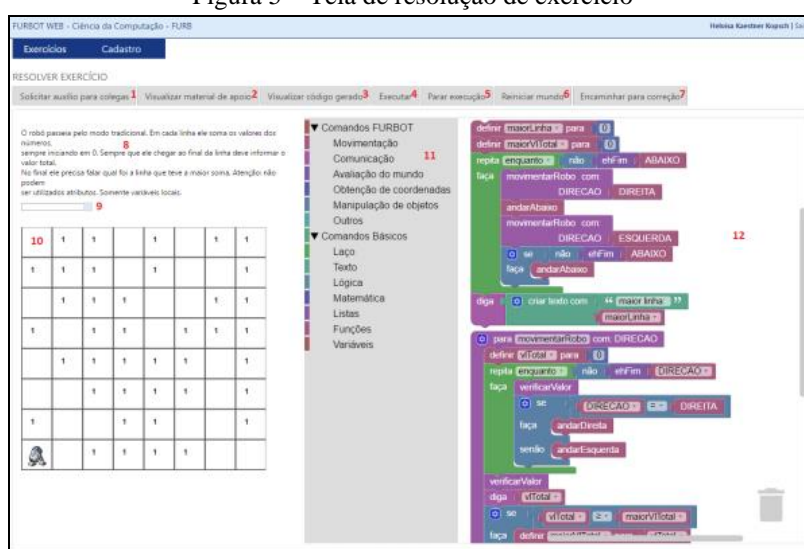
Referência	Kopsch (2016)
Objetivos	Desenvolver uma plataforma web adaptativa para o ensino de programação, tendo como base um <i>framework</i> para representações visuais e aprendizado lógico, chamado FURBOT.
Principais funcionalidades	Cadastro de turmas; gerenciamento de exercícios com suporte a correção do professor; gerenciamento de perguntas; programação em blocos; representação visual da execução.

Ferramentas de desenvolvimento	HTML5, Canvas, Css 3, JavaScript, C#, SQL Server.
Resultados e conclusões	O objetivo de criar uma plataforma web adaptativa foi atendido. A disponibilidade web da aplicação tornou mais fácil a adesão dos usuários e simplificou a experiência. O contato com a programação em blocos é atrativo para estudantes que ainda não trabalharam com a programação escrita, mas pode deixar de ser interessante para aqueles que já a praticaram. Sendo assim, a ferramenta se demonstrou mais apropriada para estudantes que estão no estágio inicial de programação. A possibilidade de realizar programação escrita proporcionaria que a ferramenta atingisse um público com conhecimentos mais avançados.

Fonte: elaborado pelo autor.

A ferramenta foi construída em cima de uma adaptação web de um *framework* chamado FURBOT, utilizado com o objetivo de auxiliar na programação lógica, através da programação de um robô que interage com objetos programáveis dentro de um mundo bidimensional (KOPSCH, 2016; MATTOS *et al.*, 2008). A Figura 5 demonstra a tela de resolução de exercícios, na qual o robô é programado.

Figura 5 – Tela de resolução de exercício



Fonte: Kopsch (2016, p. 83).

Em uma ferramenta não personalizável, o aluno é dependente do método de ensino aplicado a outros alunos, não tendo a possibilidade de produzir material de maneira que atenda às suas próprias habilidades cognitivas e seus interesses. Criar um ensino personalizado, e conseqüentemente uma plataforma adaptativa, é possibilitar que o ensino se adeque às características e especificidades de cada indivíduo (KOPSCH, 2016, p. 18 - 19). Sendo assim, a ferramenta possibilita realizar o gerenciamento dos exercícios que serão aplicados, permitindo a criação de novos exercícios e o seu monitoramento. O ensino personalizado também proporciona ao estudante uma participação mais ativa no desenvolvimento do seu conhecimento. O professor entra então como uma figura de apoio para tirar dúvidas específicas que surgem neste processo (KOPSCH, 2016, p. 18).

3 DESCRIÇÃO

Este capítulo apresenta os detalhes de especificação e implementação da ferramenta desenvolvida. O conteúdo é dividido em duas seções. A primeira seção apresenta uma visão conceitual da ferramenta, explicando os principais conceitos, os objetivos das principais partes da ferramenta e demonstrando o produto do trabalho. A segunda seção trará detalhes da implementação e de como a ferramenta foi construída, tal como as tecnologias utilizadas. Os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) realizados no desenvolvimento do trabalho estão disponíveis no APÊNDICE A.

3.1 VISÃO CONCEITUAL DA FERRAMENTA

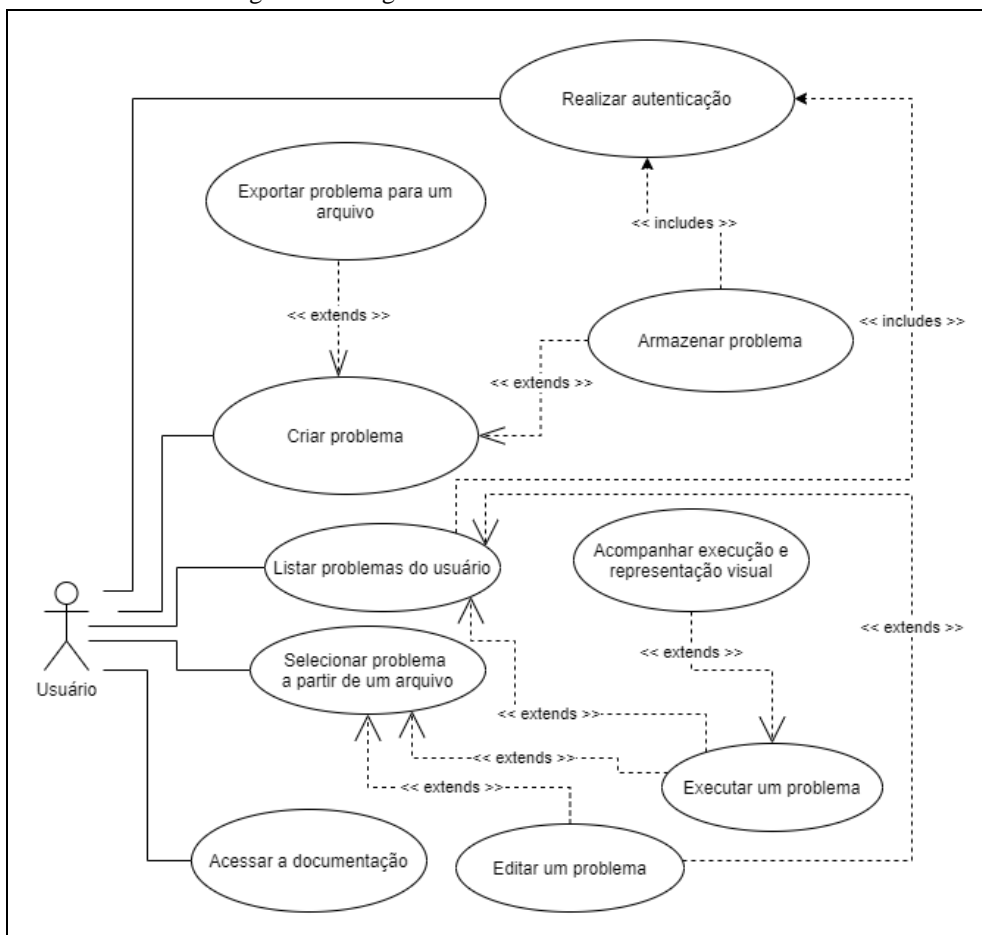
A ferramenta tem como objetivo ser uma plataforma educacional para aprendizagem de programação através da produção de material de ensino e da visualização de algoritmos. Desse modo, não se busca a substituição de uma figura educadora em um aprendizado autônomo do estudante. O foco se torna a potencialização do ensino ao oferecer uma extensão para o trabalho do educador, que atuará como um instrutor (ANDRADE, 2020). A geração do conteúdo se dá principalmente através dos **problemas**, que podem ser entendidos de maneira superficial como atividades compostas por um enunciado e uma solução. Uma descrição mais completa dos elementos de um problema será realizada posteriormente. Em relação a interação dos usuários com a ferramenta, existem dois tipos de papéis, o

educador e o estudante. O educador realiza a criação de problemas e a aplicação deles com os estudantes. Os estudantes por sua vez irão consumir o conteúdo produzido pelo educador, executando o problema com o objetivo de aprimorar o seu aprendizado. A execução de um problema se dá resumidamente pela codificação e reprodução de sua solução. Sendo assim, o educador é o produtor de conteúdo, o estudante o consumidor e a ferramenta o meio. A habilidade de gerenciamento de problemas e o controle passado para o educador define a habilidade de produzir conteúdo de ensino. Os outros recursos disponibilizados como a codificação escrita e a visualização definem o processo de aprendizado.

Considerando que a ferramenta não possui nenhum tipo de permissionamento, o caráter pode ser colaborativo. Professores e alunos são usuários que possuem acesso aos mesmos recursos da ferramenta. Os problemas criados podem ser exportados para um arquivo que será posteriormente utilizado para realizar a execução. Neste cenário, o gerenciamento dos problemas fica a cargo do próprio usuário. Para disponibilizar uma maneira de armazenamento centralizada, foi implementada uma integração com o Github. Ao autenticar na ferramenta usando o seu *login* do Github, um repositório com o nome *_smalg_platform_problems* é criado, onde seus problemas serão armazenados. A partir da autenticação com o Github, o usuário passa a poder executar problemas de outros usuários, listando diretamente do Github. Qualquer usuário pode criar os seus problemas ou executar problemas de outros usuários. O papel de cada usuário é definido pela maneira com que ele interage com a ferramenta. Um professor, por exemplo, na maioria dos casos estará atrelado a uma função de educador pois sua atividade mais comum será o gerenciamento de problemas.

Além disso, foi criado um material de documentação, disponibilizado por Andrade (2020), que pode ser acessado a qualquer momento enquanto o usuário estiver utilizando a ferramenta para instruí-lo em possíveis dúvidas. Dentro do material de documentação, são abordadas questões como: o conceito da plataforma, qual a sua finalidade, criação de problemas com exemplos passo a passo, linguagem utilizada, como funciona a execução de um problema, elementos de visualização, entre outros. O APÊNDICE B demonstra um trecho da documentação disponibilizada. A Figura 6 demonstra os casos de uso de um usuário na ferramenta.

Figura 6 – Diagrama de casos de uso da ferramenta

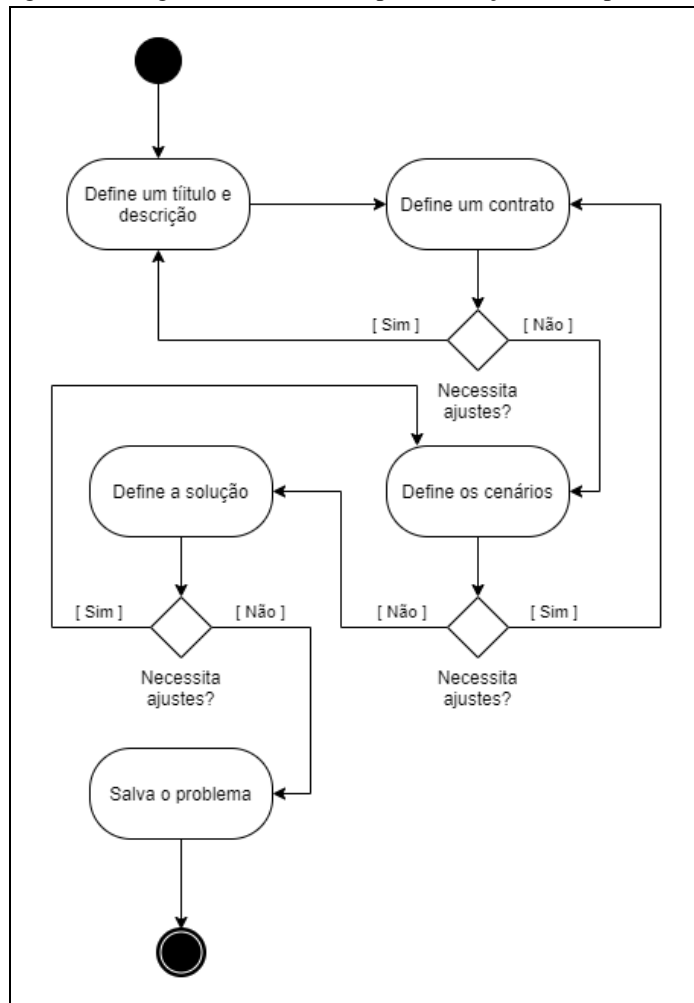


Fonte: elaborada pelo autor.

Duas ações macros norteiam a experiência do usuário, a criação de um problema e a execução de um problema. A criação de um problema é composta por quatro elementos principais. O primeiro é a definição de um **título e descrição**. Esse tem como objetivo definir um nome e contextualizar os conceitos que serão aplicados no problema, tal

como o que deverá ser desenvolvido pelo estudante. A descrição é feita com um editor de texto rico, permitindo a adição de links, imagens e vídeos, caso o educador queira adicionar materiais complementares de ensino. A ideia é que o aluno leia a descrição e entenda o que está sendo proposto. O segundo elemento é a definição de uma abstração/interface, que é tratada na ferramenta com o termo **contrato**. Em um contrato é definido o nome da classe, os campos e os métodos, com as suas respectivas documentações. É a partir do contrato definido que é gerada a interface para ser utilizada no IntelliSense (sugestão de código) do editor de código, e gerada a classe abstrata de implementação para o estudante no momento de execução do problema. O terceiro elemento é a definição dos **cenários**. A ideia do cenário é que ele funcione como pequenos trechos de execução, segmentados em pequenas partes que juntos validem o todo. Essa abordagem é inspirada em testes unitários, em que cada cenário de teste valida uma unidade do sistema, resultando na validação do sistema por completo. Essa abordagem tem algumas vantagens. A primeira é que o estudante trabalhará com unidades lógicas menores, tornando a visualização e execução mais simplificada. Um cenário muito extenso poderá sobrecarregar o entendimento com informações a respeito de uma execução que poderia ser feita em partes. A segunda é o modelo de implementação do estudante, que pode passar a ser incremental. Os cenários se tornam então pequenos objetivos que o estudante pode concluir, coletando resultados e o incentivando para a conclusão de um objetivo maior, que é a resolução do problema. Um cenário é composto por um nome, descrição e o código de execução. A descrição do cenário é importante para dizer ao estudante o que será executado, como será executado e dar dicas se necessário. O código de execução será responsável por definir a execução do cenário, utilizando a interface gerada pelo contrato e validando a implementação realizada pelo estudante. O último elemento é a definição da **solução**. Ela servirá para o educador validar os cenários criados, visualizar o resultado e proporcionar ao estudante uma resposta caso ele não consiga resolver o problema. Nesse último caso, o uso da solução e a apresentação da visualização em si já permite auxiliar no processo de aprendizado. O processo de criação de um problema é especificado na Figura 7.

Figura 7 – Diagrama de atividades para a criação de um problema.



Fonte: elaborada pelo autor.

Para a implementação de um cenário, são disponibilizadas três variáveis: uma variável da abstração da classe criada a partir do contrato que conterá a implementação do estudante, um objeto chamado `context` e outro objeto chamado `assertion`. Exemplificando, se um contrato for definido com o nome `ListaDinamica`, a variável a ser disponibilizada no cenário será `listaDinamica`, em *camel case*, e seguindo a definição dos campos e métodos

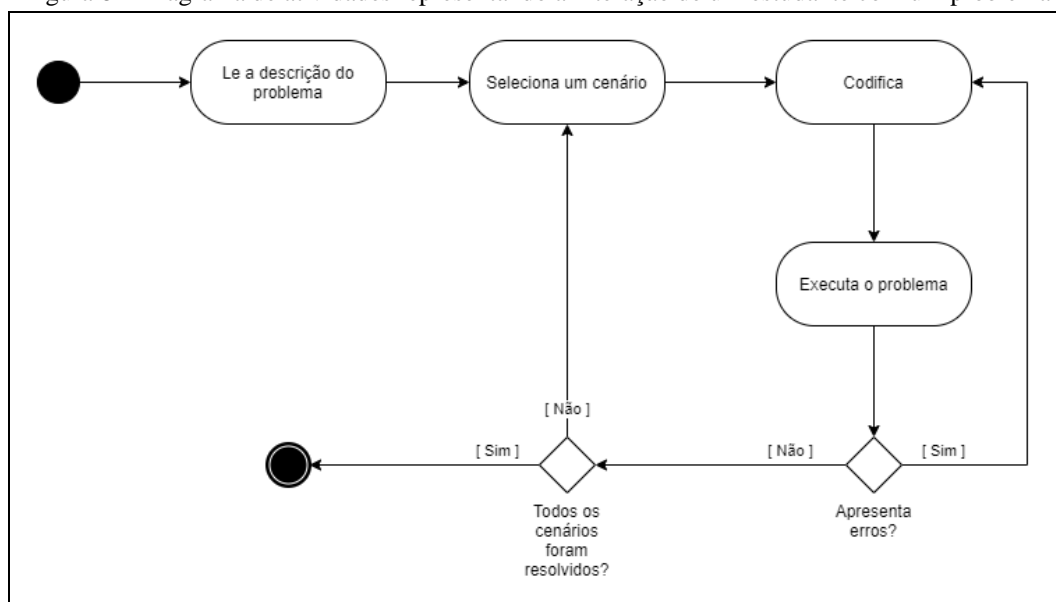
realizada na etapa de definição do contrato. O objeto `context` por sua vez, é o contexto de execução, sendo responsável por criar as estruturas internas chamadas *objeto* e *container*, obter os *objetos* e *containers* existentes ou limpar elementos criados pelo próprio contexto de execução. Os *objetos* e *containers* são os elementos básicos, junto às primitivas, que são utilizados na plataforma. O objeto `assertion` tem a responsabilidade de disponibilizar métodos para realizar a assertiva da execução e validar o cenário. O APÊNDICE C ilustra o processo completo da criação de um problema.

O contexto de execução é uma peça importante na execução de um problema. Como dito anteriormente, a partir dele são criados os elementos básicos para serem trabalhados em código: os *objetos* e *containers*. Os *objetos* são estruturas que funcionam através de chave/valor. Como exemplo, em um problema de uma lista encadeada, eles seriam utilizados para representar os nós. Para criar um objeto basta chamar `context.newObject(): SmalgObject`, que será retornada uma estrutura chamada `SmalgObject`. Ela terá os métodos: `set(c: string, v: any): void`, sendo `c` a chave e `v` o valor; e `get(c: string): any`, sendo `c` a chave e o retorno o valor. Os *containers* são estruturas que funcionam através de indexação e um tamanho pré-fixado. Em outras palavras, eles se comportam como um vetor. Para criar um container basta chamar `context.newContainer(t: int): SmalgContainer`, sendo `t` o tamanho do container, que será retornada uma estrutura chamada `SmalgContainer`. Ela terá os métodos: `set(i: int, v: any)`, sendo `i` o índice e `v` o valor; `container.get(i: int): any`, sendo `i` o índice e o retorno o valor; e `size(): int`, sendo o retorno o tamanho. Tanto o educador quanto o estudante deverão utilizar essas estruturas para criar e solucionar os problemas. O contexto de execução e objeto de assertivas, exemplo de um código de um cenário e um exemplo de uma solução para um problema Bubble Sort, são apresentados no APÊNDICE D.

A linguagem de programação que é utilizada para codificação dos cenários e da solução do problema é JavaScript. Porém, devem ser utilizados os tipos `SmalgObject` e `SmalgContainer`, ao invés de objetos puros e vetores disponibilizado pela linguagem, pois eles serão desconsiderados na visualização. Desse modo, pode-se dizer que a ferramenta disponibiliza um SDK que deve ser utilizado na criação dos problemas.

Para execução do problema, o estudante deverá selecionar um problema previamente criado. Ao abrir o problema ele verá a descrição e um espaço onde ele implementará a solução e visualizará a representação visual. A solução, como dito anteriormente, se dá pela implementação do contrato definido. Para iniciar a implementação ele precisará escolher um cenário, no qual o código que ele implementou será executado. Como a implementação é única, o cenário apenas definirá como será a execução. Como o objetivo é promover uma evolução incremental, a qualquer momento o estudante pode executar o código, coletar um feedback da execução a partir da execução dos cenários e da representação visual e retomar o processo de codificação. O processo termina quando todos os cenários forem atendidos. A Figura 8 demonstra o processo de execução de um problema por parte do estudante.

Figura 8 – Diagrama de atividades representando a interação de um estudante com um problema



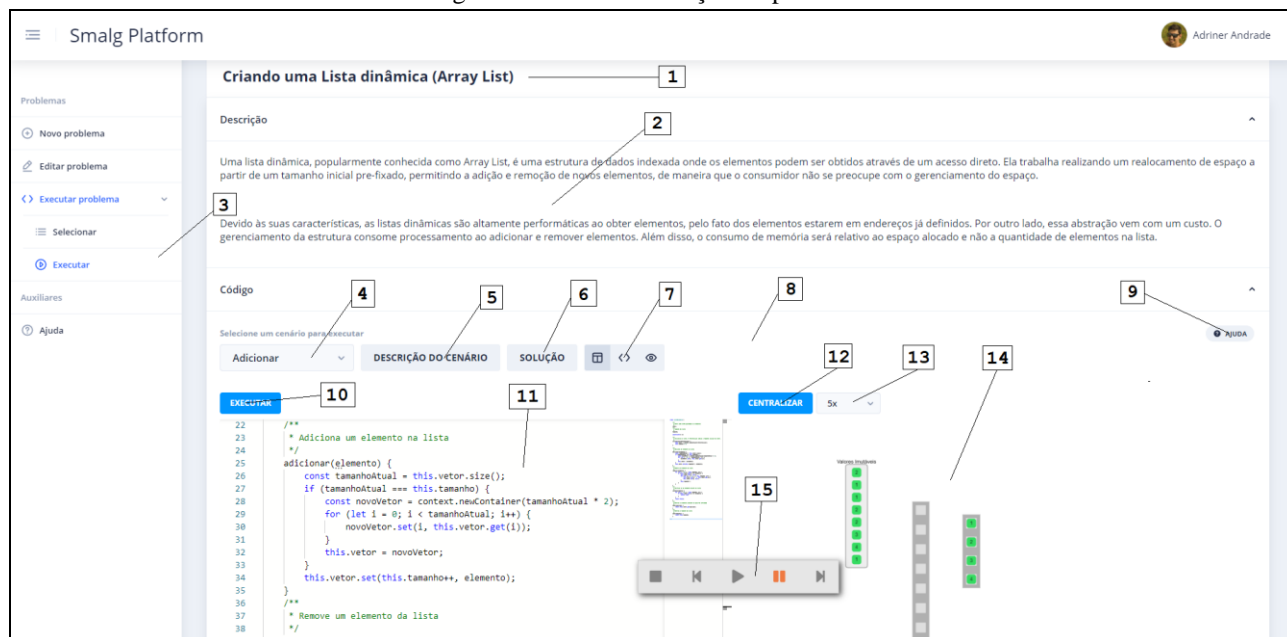
Fonte: elaborada pelo autor.

Toda representação visual é gerada em cima da utilização do contexto de execução e dos elementos criados através dele. Além da criação dos elementos, é tratada a relação que se tem entre eles: por cópia e por referência. Tipos complexos como objetos e containers, são tratados com referência, enquanto tipos primitivos como *boolean* e *number*, são tratados por cópia. O APÊNDICE E detalha os elementos que são apresentados visualmente. Uma característica importante é que os elementos visuais são interativos, permitindo movimentá-los durante a execução. A velocidade da

representação visual pode ser controlada pelo usuário. A execução da visualização também pode ser controlada, permitindo ações como pausar, executar apenas a próxima ação ou retroceder.

A Figura 9 ilustra a tela de execução de um problema, na qual está sendo executado um problema para a criação de uma lista dinâmica ou *array list*. Este problema é um dos exemplos presentes na documentação que instruem a criação de um problema passo a passo. Também é disponibilizado na documentação um arquivo compactado contendo quatro problemas padrões: Criando uma Lista dinâmica (Array List), Criando uma Lista Encadeada, Invertendo os elementos de um container e Ordenando através do método bolha (Bubble Sort). Em todas as páginas existe um componente de ajuda para direcionar para a documentação e um item no menu lateral que permite o acesso ao conteúdo raiz da documentação. O Quadro 4 apresenta uma descrição dos elementos apresentados na tela.

Figura 9 – Tela de execução do problema



Fonte: elaborada pelo autor.

Quadro 4 – Descrição dos elementos da tela de execução do problema

Identificador	Descrição
1	Título do problema.
2	Descrição do problema. O objetivo da descrição é explicar o assunto do problema, disponibilizar materiais auxiliares para o entendimento e descrever o que o estudante deverá fazer para completar o objetivo.
3	Menu lateral de acesso à tela de execução.
4	Componente de seleção do cenário.
5	Botão para exibir a descrição do cenário selecionado.
6	Botão para exibir a solução do problema.
7	Botões de perspectivas, para controlar a visão do código e da visualização.
8	Espaço/painel de código do problema para resolução do problema.
9	Botão de ajuda que realiza a abertura da documentação, contextualizada na execução de problemas.
10	Botão que inicia a execução do código.
11	Editor de código com a classe gerada por meio do contrato para a codificação do problema.
12	Botão para realizar a centralização da animação.
13	Componente para controlar a velocidade da animação.
14	Espaço de visualização da execução do algoritmo.
15	Componente para controle da execução da visualização, permitindo parar, retroceder, resumir, pausar e executar a próxima ação.

Fonte: elaborado pelo autor.

3.2 IMPLEMENTAÇÃO

A ferramenta foi desenvolvida para a plataforma web para os navegadores Microsoft Edge v87.0.664.41, Google Chrome v87.0.4280.66 e Mozilla Firefox v83.0, não tendo foco em disponibilidade para plataformas móveis. O processamento é realizado *client side*. Para gerenciamento dos problemas, foi disponibilizada a opção de se trabalhar

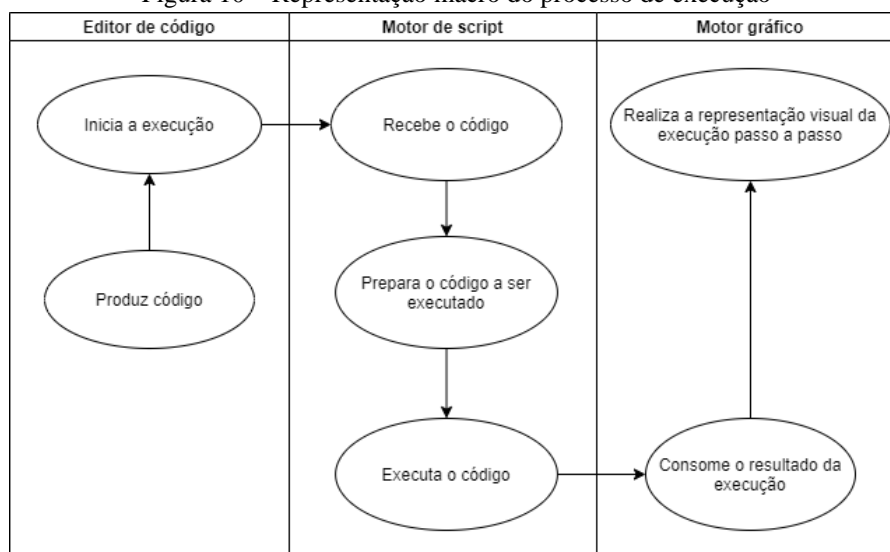
importando e exportando arquivos. Como alternativa, foi implementada uma integração com o Github através de autenticação OAuth2.0, sua Application Programming Interface (API) e sua *library* de desenvolvimento para chamadas Rest, a Octokit Rest (GITHUB, 2020).

Para desenvolvimento do *frontend* foi utilizado o framework Angular 10+, tendo como base o projeto Ngx Admin (AKVEO, 2019a), construído em cima da biblioteca de componentes visuais Nebular (AKVEO, 2019b). O componente utilizado para realizar a codificação do cenário e da solução do problema foi o Monaco Editor (MICROSOFT, 2020a). Ele é construído em cima do motor do Visual Studio Code, com suporte a *text highlight* e *autocomplete* para múltiplas linguagens. Vale ressaltar que ele é extensível a ponto de definir uma própria estrutura de linguagem a partir de uma sintaxe customizada. Para a representação visual foi utilizada a biblioteca Cytoscape.js (CONSORTIUM, 2020). Ela é uma biblioteca utilizada para a representação de grafos e neste trabalho teve seu uso adaptado para representar os elementos visuais e os seus relacionamentos. Como opção de editor de texto rico foi utilizada a biblioteca QuillJS (QUILL, 2020). Ela oferece suporte à adição de vídeos, links e até mesmo imagens. Especificamente na questão das imagens foi necessária a criação de um *handler* customizado que adiciona imagens a partir de um link. Isso porque por padrão é aceito somente imagens no formato Base64, o que acredita-se que poderia afetar a performance e o armazenamento dos problemas, tendo em vista que um link possui um tamanho muito menor do que uma imagem em Base64.

A primeira etapa da implementação foi reunir as tecnologias acima citadas e fazê-las funcionar em um único projeto. O Angular é um *framework* que incorpora Typescript (MICROSOFT, 2020b) e abstrai diversas funcionalidades, trabalhando com conceitos como o *bind*, no qual os valores dos elementos visuais (*input*, *select*, botões, ...) são gerenciados automaticamente em atributos do código. Já bibliotecas como Monaco Editor e Cytoscape.js são disponibilizadas em Javascript puro, sendo necessário um encapsulamento, inicialização e gerenciamento manual das suas funcionalidades.

Em relação a estrutura da ferramenta, visando a abstração e a possibilidade de substituição/incorporação de novas tecnologias em cima das existentes, o processo de execução de um problema foi dividido em três componentes principais: o editor de código, o motor de script e o motor gráfico. A primeira etapa é a produção do código. Com a definição do contrato do problema, o editor de código é inicializado com uma classe contendo o nome, os métodos e os campos definidos, estando pendente somente as implementações. Assim, fica a cargo do estudante preencher cada método com a sua respectiva implementação. Quando a execução do problema é acionada (a implementação não precisa estar completa), o código produzido pelo estudante é capturado e repassado para o motor de *script*. Esse motor tem como responsabilidade preparar o código para a execução e então executá-lo. O resultado dessa execução será uma validação a partir da execução do cenário escolhido, verificando se houve erros na execução, e uma estrutura que será consumida pelo motor gráfico. O motor gráfico por sua vez realiza a renderização visual da execução disponibilizando a execução passo a passo e permitindo um controle de velocidade. A Figura 10 apresenta uma representação macro do processo de execução.

Figura 10 – Representação macro do processo de execução

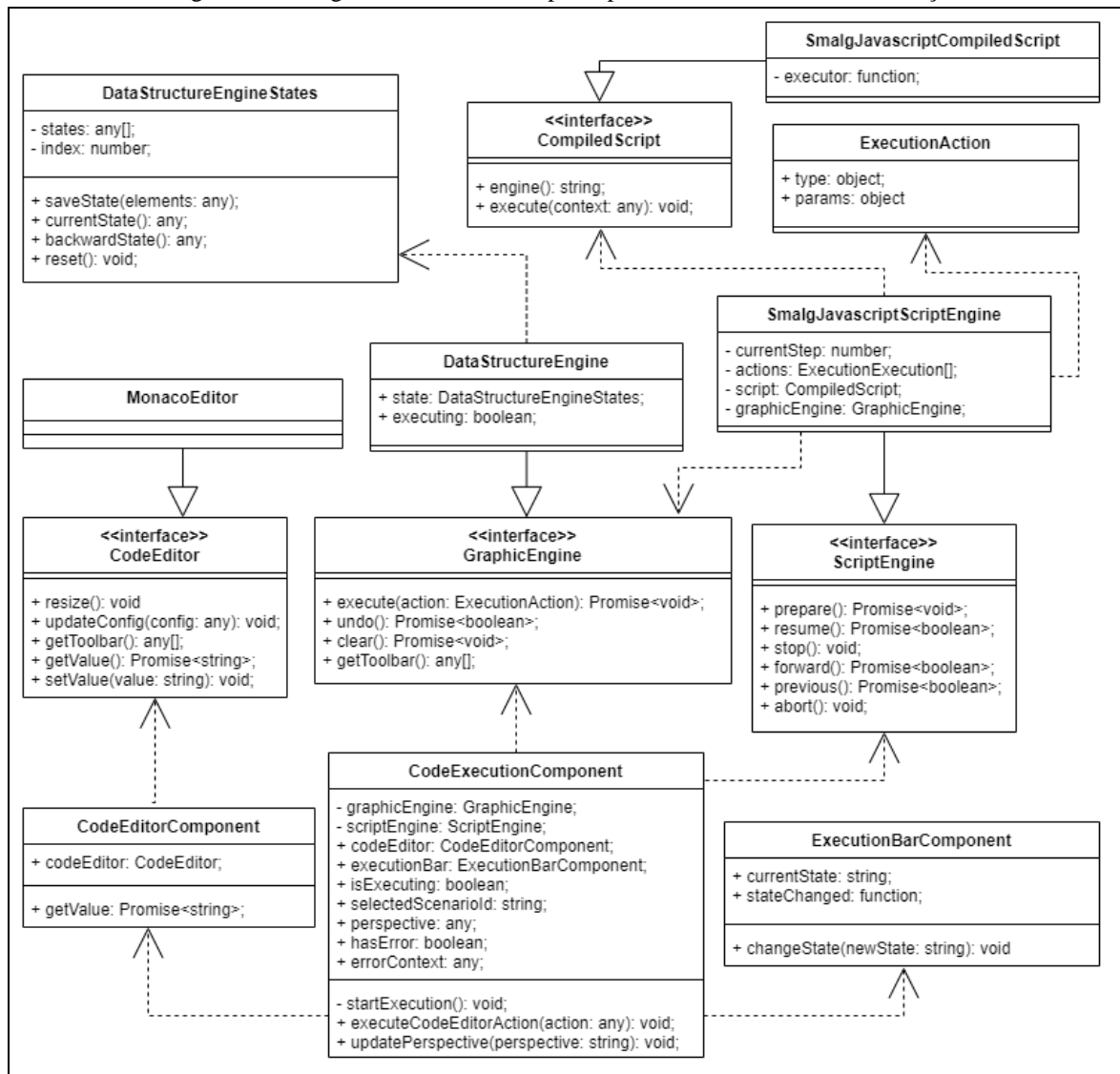


Fonte: elaborada pelo autor.

A Figura 11 apresenta um digrama de classes com as principais classes utilizadas na execução. A ferramenta possui muitas classes que representam componentes visuais ou que são secundárias e por não apresentarem relevância para o entendimento do funcionamento, não serão apresentadas. Para cada um dos componentes principais apresentados anteriormente (editor, motor de script e motor gráfico) foram criadas interfaces (CodeEditor, ScriptEngine e

GraphicEngine), sendo as suas implementações disponibilizadas por um *provider*. O *provider* é uma estrutura que recebendo uma chave, é retornado o componente associado àquela chave. O editor de código, por exemplo, tem duas variações: uma para criação do cenário e outra para a execução do problema, cada uma com sua chave (*smalg-javascript-assertion* e *smalg-javascript-execution*, respectivamente). O objetivo principal é que, tendo a ferramenta o comportamento de plataforma, ela seja extensível a ponto de permitir adicionar um novo editor de código, um novo motor de script ou novo motor gráfico. Isso significa que a arquitetura facilita serem adicionados novas estratégias de execução, novos modelos de visualização e novas configurações no editor que venham satisfazer situações específicas. Um exemplo, seria a criação de um novo tipo de execução para atender uma linguagem funcional ou uma visualização focada especificamente em grafos. A classe *DataStructureEngine* é a responsável por realizar a representação visual, e recebe esse nome somente porque seus elementos visuais representam objetos e vetores. Ela consome a classe *DataStructureEngineStates*, responsável por salvar os estados gerados durante a execução. Essa tarefa é importante para que seja possível retroceder durante a execução. A classe *CodeExecutionComponent* é o componente responsável pela tela de execução. A interface *CompiledScript* tem como objetivo oferecer uma abstração a nível de linguagem de maneira que seja possível reaproveitar o motor de script ou visualização existente. Como exemplo, é possível citar a criação de uma linguagem portugalol, que seria compilada para um formato entendível pela *SmalgJavaScriptScriptEngine*, a implementação atual do motor de script.

Figura 11 – Diagrama de classes das principais classes do núcleo de execução

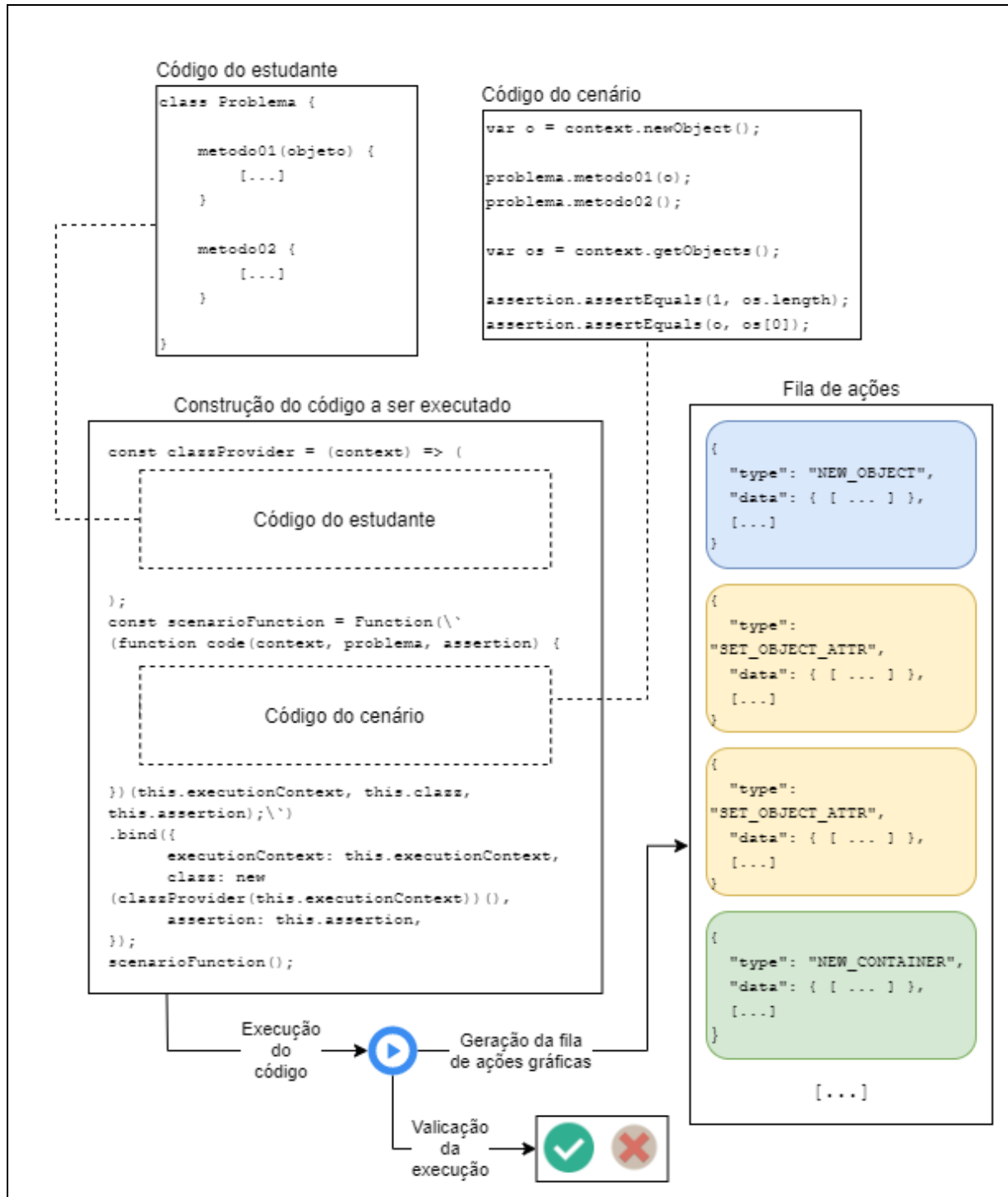


Fonte: elaborada pelo autor.

O processo de execução do motor de script, em detalhe, inicia criando uma função de execução, que encapsula o código do cenário junto com o código do estudante, e a prepara para receber um contexto de execução e de assertivas. Todo o processo é realizado através da criação de funções JavaScript via o construtor *Function*, que permite criar uma função a partir de uma *string*. O resultado da criação da função é abstraído em um *CompiledScript* que ao ser executado recebe o contexto de execução e assertivas. Internamente, tais valores são injetados na função de execução

por meio dos métodos `bind` e `apply`, nativos do JavaScript. O resultado dessa execução é a validação indicando se houve erros durante a execução do código e uma fila de ações que será consumida posteriormente pelo motor gráfico. A fila de ações será gerada até o momento que houver o erro na execução justamente para que o estudante tenha a representação visual ainda que sua implementação não esteja finalizada. A criação de uma ação é realizada por meio da interação que se tem no contexto de execução e nos seus objetos gerenciados, funcionando como um *tracking*. No caso do *objeto* e do *container*, por exemplo, seria em cima dos métodos `get` e `set`. A Figura 12 ilustra o processo descrito.

Figura 12 – Criação da função de execução e a geração da fila de ações



Fonte: elaborada pelo autor.

O Quadro 5 mostra o código que realiza a criação da função de execução. A classe `SmalgJavascriptFormatter` é responsável por montar a *string* que será utilizada para a criação da função.

Quadro 5 – Código responsável por realizar a criação da função de execução

```
class SmalgJavascriptCompiledScript implements CompiledScript {  
  
    private executor: Function;  
  
    constructor(contract: ClassContract, problemScenario: ProblemScenario, code:  
string) {  
        this.executor = new Function(SmalgJavascriptFormatter.format(contract,  
problemScenario, code));  
    }  
  
    engine(): string {  
        return 'smalg-javascript';  
    }  
  
    execute(context: any): void {  
        this.executor.apply({ executionContext: context, assertion: new  
SmalgJavascriptAssertion() });  
    }  
}  
  
export class SmalgJavascriptCompiler implements ScriptCompiler {  
  
    compile(contract: ClassContract, problemScenario: ProblemScenario, code: string):  
CompiledScript {  
        return new SmalgJavascriptCompiledScript(contract, problemScenario, code);  
    }  
}
```

Fonte: elaborado pelo autor.

O Quadro 6 mostra o código do contexto de execução. Pode-se observar que na criação de cada estrutura, como `SmalgObject` e `SmalgContainer`, é repassada a fila de ações, para que sejam adicionadas as suas respectivas ações de criação e as posteriores ações de interação que irão ocorrer com elas.

Quadro 6 – Código disponibilizado pelo contexto de execução

```
export class SmalgJavascriptContext {  
  
    private elements: { [key: string]: SmalgType } = {};  
  
    constructor(private actions: ExecutionAction[]) {}  
  
    newObject() {  
        return this.createNewElement(() => new SmalgObject(this.actions));  
    }  
  
    newContainer(size: number) {  
        return this.createNewElement(() => new SmalgContainer({ size }, this.actions));  
    }  
  
    [...]  
  
    private createNewElement(elementProvider: () => SmalgType): SmalgType {  
        const elem = elementProvider();  
        this.elements[elem.__getId__()] = elem;  
        return elem;  
    }  
}
```

Fonte: elaborado pelo autor.

O Quadro 7 apresenta o código do método `set` da classe `SmalgContainer` para demonstrar a atribuição de um item em um container e a adição de uma ação na fila de ações. É possível observar que ocorre a tratativa de tipos, para identificar quando é um tipo primitivo e assim criar um elemento chamado `SmalgPrimitive`, utilizado para representar um valor de tipo primitivo na aplicação. O método `__reference__` está presente na classe `SmalgType`, abstração para os tipos `SmalgObject`, `SmalgContainer` e `SmalgPrimitive`, e é através dele que a atribuição por

cópia ou por referência é resolvida. Dentro da classe `SmalgType` existem outros métodos importantes também, como o `equals` e o `toString`, apesar de estes serem mais utilizados para as assertivas.

Quadro 7 – Método `set` da classe `SmalgContainer`

```
set(index: number, value?: SmalgType | string | boolean | number) {
  this.validateIndex(index);

  if (isPrimitive(value)) {
    value = new SmalgPrimitive(value, this.actions);
  }

  value = value?._reference_();
  this.actions.push({
    type: DataStructureAction.SET_CONTAINER_SLOT,
    params: { id: this.__getId_(), index, value: value?._getId_() },
  });
  this.container[index] = value;
}
```

Fonte: elaborado pelo autor.

Após a fila de ações ser gerada, a representação visual pode ser iniciada. O gerenciador de execução será responsável por ditar o ritmo da execução, respeitando as ações realizadas pelo usuário, como resumir a execução automática, pausar a execução automática, retroceder ou executar o próximo passo. Sempre que um novo passo for executado, ele irá consumir a ação corrente na fila e enviar para o motor gráfico de modo assíncrono. O motor gráfico ao receber a ação realiza a interpretação e a renderização. Para cada ação existe um código específico que realiza a representação visual correspondente. O Quadro 8 mostra o código responsável por gerenciar uma ação de criação de um *container* para fins de demonstração. Neste caso em questão, ocorre a obtenção do tamanho do container e do id, o posicionamento inicial e a criação dos elementos filhos que representam uma posição no vetor, chamados de `slots`. Em outros casos, como a atribuição de um elemento a um objeto ou *container*, ocorrerá maiores tratativas para saber qual o tipo dos elementos que estão sendo atribuídos e como será representada essa relação. Tipos primitivos são tratados como filhos dos `slots`, enquanto tipos complexos são com uma ligação. No caso da passagem de valores nulos, por exemplo, deve haver a remoção do elemento visual ou a exclusão de uma ligação. Após o motor gráfico terminar de realizar a renderização de uma ação ele notifica o gerenciador de execução para que ele prossiga com o próximo passo. Essa notificação é realizada por meio do recurso `async` e `await` do Javascript e de `Promises`. É importante ressaltar que ao invés de serem eliminados os elementos da fila, é utilizado um indexador que aponta para a ação corrente. A cada passo da execução é salvo o estado gráfico atual (um *snapshot*), em uma estrutura contendo todos os estados gráficos já criados. É a partir dessa estrutura que será recuperado um estado gráfico ao retroceder a execução.

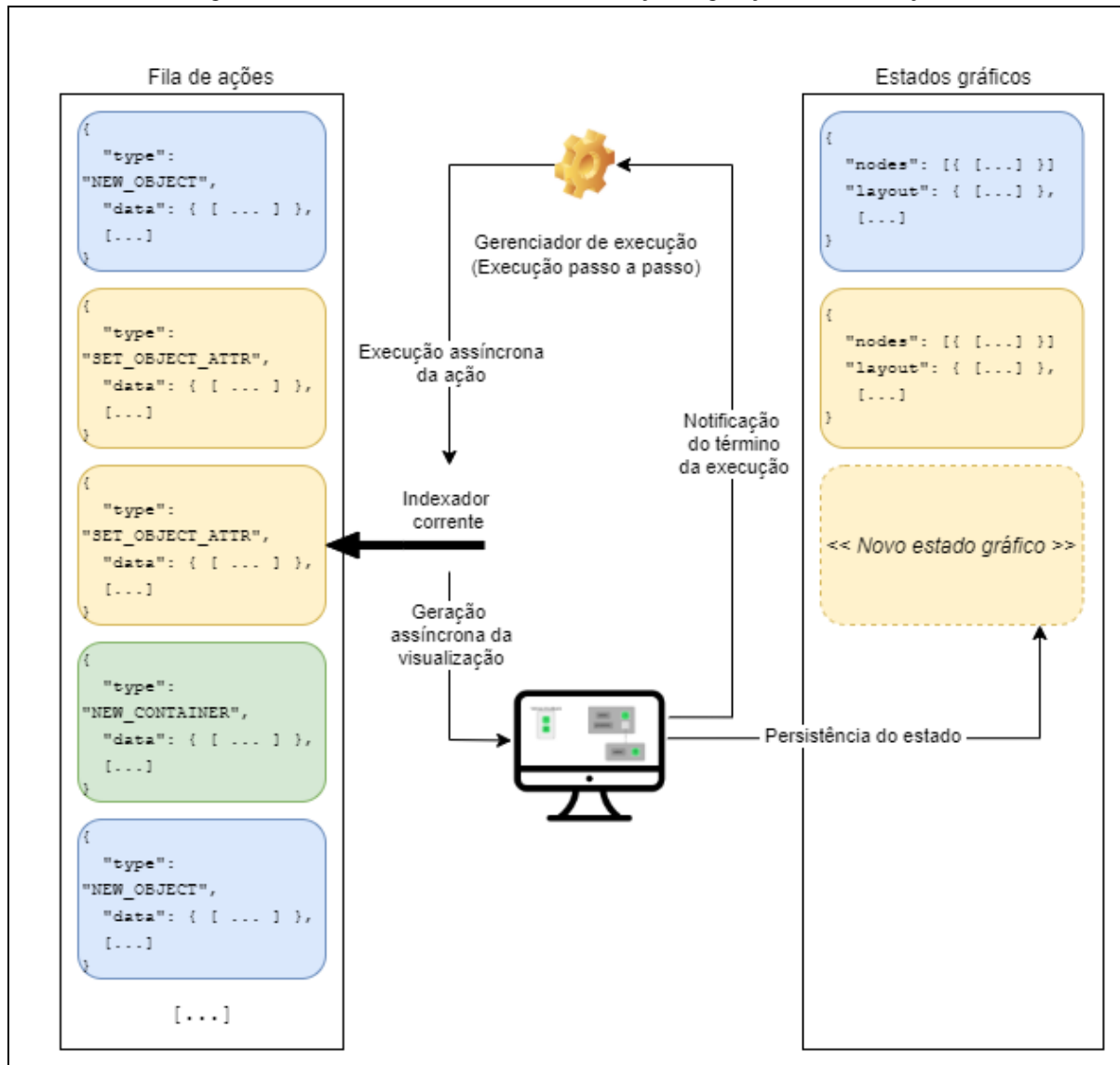
Quadro 8 – Código responsável por gerenciar uma ação de criação de um *container*

```
async handle(cytoscape: any, action: ExecutionAction): Promise<void> {
  const { id, size } = action.params;
  const containerElement = await $add(cytoscape, {
    data: {
      id,
      type: ElementTypes.CONTAINER,
    },
  });
  await this.layoutHandler.setInitialPosition(containerElement);
  for (let i = 0; i < size; i++) {
    const slotElement = await $add(cytoscape, {
      data: {
        id: `${id}_${i}`,
        type: 'container_slot',
        index: i,
      },
      classes: ['slot'],
    });
    await this.layoutHandler.moveToContainer(containerElement, slotElement);
    slotElement.move({ parent: id });
  }
  await this.layoutHandler.run(containerElement);
}
```

Fonte: elaborado pelo autor.

A Figura 13 mostra uma representação do processo de geração da representação visual descrito. Pode-se observar o consumo da fila de ações, a existência do indexador, a execução assíncrona e o salvamento do *snapshot* do estado gráfico a cada execução.

Figura 13 – Processo de consumo da fila de ações e geração da visualização



Fonte: elaborada pelo autor.

4 RESULTADOS E DISCUSSÕES

Este capítulo apresenta os resultados obtidos com a ferramenta por meio de testes. A primeira seção contém considerações do autor a respeito da implementação e do uso da ferramenta, sendo seguido por um teste de usabilidade com um usuário externo. A segunda seção descreve a aplicação de testes com alunos do segundo semestre de Ciência da Computação da FURB. A terceira seção descreve a aplicação de testes com professores de Ciência da Computação da FURB, sendo um responsável pela disciplina de Algoritmos e Estrutura de Dados e outro responsável por Introdução à Programação. A quarta seção discute uma comparação do resultado com os trabalhos correlatos.

4.1 CONSIDERAÇÕES DO AUTOR E TESTE DE USABILIDADE

Durante a implementação e uso da ferramenta pelo autor foram notadas algumas limitações existentes, detalhadas a seguir. A utilização direta da função construtora `Function` do JavaScript tem como vantagem a utilização do próprio interpretador da linguagem. Como desvantagem, não se tem o mesmo controle sobre o código que se teria com a criação de um interpretador próprio. Sendo assim, uma maneira de aprimorar é através da criação de uma linguagem educacional, como `portugol`, que transpile o código desenvolvido para o padrão de linguagem adotado atualmente pela plataforma (JavaScript e os objetos `context` e `assertion`). O objetivo é oferecer um maior controle sobre o código e criar uma linguagem amigável, ao mesmo tempo que é reaproveitado todo o processo de representação visual e execução passo a passo já desenvolvido. A partir dessa implementação, seria mais simples construir o acompanhamento passo a passo, não apenas visual, mas também em cima do código que está sendo executado. A cada expressão detectada, por exemplo, pode haver a criação de um *breakpoint*, que indicará em qual linha especificamente o

código está sendo executado. O acompanhamento em cima do código inclusive foi uma funcionalidade ressaltada pelos professores. Além disso, facilitaria a formulação das mensagens de erro ao usuário final. Utilizar diretamente o interpretador do JavaScript não permite flexibilizar erros de sintaxe, nem realizar customizações em *stacktraces*.

A primeira etapa de validação da plataforma com outros usuários foi realizada com um usuário já experiente em programação, cursando os semestres finais de bacharelado em Ciência da Computação na universidade Unisul, sendo monitorado pelo autor, mas sem nenhuma instrução. A ideia foi coletar pontos de usabilidade que poderiam ser melhorados a partir de um primeiro contato de um usuário, apesar da plataforma não ter sido construída com o objetivo de oferecer uma experiência exclusivamente autodidata. A partir deste teste, foram identificados erros gerados por outros fluxos de uso, sendo posteriormente corrigidos, além de ajustes em componentes para direcionar melhor o usuário em tela, sem que fosse necessário a todo momento recorrer a documentação. Dentre as ações realizadas estão: adicionada uma indicação no componente de código que deixe claro onde o código deve ser programado; enriquecimento de alguns pontos da documentação com *gifs*; melhor instrução de onde a solução deve ser cadastrada na criação de um problema; e melhorias nos questionários de testes a serem aplicados, com um roteiro mais direcionado. Com este usuário especificamente não foi coletado nenhum tipo de informação por meio de questionários, pois a finalidade era testar e refinar o processo para posterior aplicação com alunos e professores.

4.2 TESTES DE UTILIZAÇÃO COM ALUNOS

A segunda etapa de validação da ferramenta, foi aplicada com quatro estudantes de Ciência da Computação na FURB, todos cursando o segundo semestre. O método de aplicação consistiu em oficinas na qual o autor exercia o papel de professor, explicando a ferramenta e os problemas que seriam aplicados, instruindo e respondendo possíveis dúvidas. O objetivo era que o estudante conseguisse através da ferramenta implementar e resolver o problema. Posteriormente, foi aplicado um questionário coletando a visão dos estudantes a respeito da ferramenta e da contribuição que ela proporcionou ao aprendizado. Os problemas aplicados foram: Ordenando através do método bolha (Bubble Sort), Criando uma Lista Encadeada e Criando uma Lista dinâmica (Array List), todos criados pelo autor. Uma observação importante a ser feita, é a de que o problema da lista dinâmica foi o último aplicado nas oficinas e tratado de maneira um pouco mais superficial em relação aos outros problemas por causa do tempo disponível dos participantes. O Quadro 9 apresenta as perguntas aplicadas e as respostas coletadas pelos estudantes. O questionário completo com todas as opções de respostas disponíveis pode ser encontrado no APÊNDICE F.

Quadro 9 – Resultado dos testes aplicados com estudantes

Pergunta	Respostas
Tendo em vista o problema Bubble Sort, você considera que a ferramenta te auxiliou a entender o conceito envolvido?	Sim, muito. – 75% (3/4). Sim, um pouco. – 25% (1/4).
Tendo em vista o problema da lista encadeada, você considera que a ferramenta te auxiliou a entender o conceito envolvido?	Sim, muito. – 75% (3/4). Sim, um pouco. – 25% (1/4).
Tendo em vista o problema lista dinâmica (Array List), você considera que a ferramenta te auxiliou a entender o conceito envolvido?	Sim, muito. – 25% (1/4). Sim, um pouco. – 75% (3/4).
Como você avalia o grau de dificuldade para utilizar a ferramenta?	Não tive dificuldades. – 25% (1/4). Baixo. – 75% (3/4).
Como você descreveria sua curva de aprendizado para utilizar a ferramenta?	Curta, um dia. – 100% (4/4).
Como você avalia a documentação disponibilizada?	Muito boa. – 25% (1/4). Boa. – 50% (2/4). Regular. – 25% (1/4).
No geral, acredito que a ferramenta:	Foi útil para o meu aprendizado. – 100% (4/4).

Fonte: elaborado pelo autor.

No questionário aplicado, também foi disponibilizado um espaço para considerações. Em relação ao problema Bubble Sort um estudante afirma que “com o sistema de animação foi possível ver a representação da ordenação ocorrendo, o que auxilia no entendimento tanto desse problema como dos outros também.”. Em relação ao problema de lista encadeada um estudante afirma que “a representação visual ajudou muito na compreensão do problema e a implementar a solução”.

4.3 TESTES DE UTILIZAÇÃO COM PROFESSORES

A última etapa de validação foi a aplicação da ferramenta com professores de Ciência da Computação da FURB, um responsável pela disciplina de Algoritmos e Estrutura de Dados e o outro responsável pela disciplina de Introdução a Programação. Através de uma chamada on-line com compartilhamento de tela a ferramenta foi

apresentada, sendo explicado os seus conceitos e o seu propósito, demonstrando casos de uso. Após a apresentação foi dado um período de aproximadamente uma semana para que os professores realizassem a experimentação da ferramenta e então respondessem um questionário detalhado no Quadro 10, com base em suas perspectivas. O questionário completo com todas as opções de respostas disponíveis pode ser encontrado no APÊNDICE F.

Quadro 10 – Resultado dos testes aplicados com os professores

Pergunta	Respostas
Como você avalia a flexibilidade para criação de problemas?	Muito boa. – 100% (2/2).
Como você avalia a facilidade para a criação de problemas, considerando a flexibilidade proposta?	Muito boa. – 50% (1/2). Regular, não tão fácil nem tão complexo. – 50% (1/2).
Como você avalia o potencial da plataforma para ensinar de modo autodidata para os alunos?	Bom, porém o aluno terá algumas dificuldades. – 50% (1/2). Regular, o aluno terá dificuldades medianas para usar. – 50% (1/2).
Como você avalia o potencial da plataforma para ser utilizada como ferramenta auxiliar ao professor no ensino?	Muito bom. – 100% (2/2).
Você utilizaria a ferramenta aplicando diretamente com os alunos?	Sim. Acredito que a interação dos alunos pode agregar ao aprendizado. – 50% (1/2) Não, mas montaria meus próprios problemas e apresentaria a visualização para auxiliar na explicação. – 50% (1/2).
Se você for aplicar diretamente com os alunos, como você avalia o nível de acompanhamento e instrução que deverá ser realizado por parte do professor?	Razoável. – 100% (2/2).
Como você classifica a visualização apresentada?	Muito boa. – 50% (1/2). Boa. O conceito pode ser entendido, mas tem alguns pontos que poderiam melhorar. – 50% (1/2).
Como você classifica a curva de aprendizado para conseguir utilizar a ferramenta, por parte do professor?	Curta, um dia. – 50% (1/2). Média, três dias. – 50% (1/2).
No geral, considero a ferramenta como uma possibilidade de ensino:	Muito boa. – 50% (1/2). Boa. – 50% (1/2).
Como você avalia a qualidade da documentação da plataforma?	Muito boa – 100% (2/2).

Fonte: elaborado pelo autor.

No espaço aberto para considerações, um dos professores comentou que: “A visualização dos dados primitivos na animação num primeiro momento me confundiu um pouco, mas depois de olhar a documentação e executar algumas vezes a visualização consegui entender.”. A visualização dos dados primitivos comentada, se refere ao elemento visual que representa os valores primitivos sendo trafegados por fora dos objetos e containers (APÊNDICE E).

Em uma das apresentações realizadas foi sugerido ter um *console* no qual o usuário interage com a animação em tempo de execução. Essa proposta busca aumentar a interatividade e coleta de informações por parte do estudante. Uma outra dificuldade encontrada, foi realizar a organização automática dos elementos visuais, para diminuir a quantidade de interações e tornar a visualização mais ágil em tela. A biblioteca *Cytoscape* possui algoritmos que realizam esse processo, mas eles não apresentam um bom comportamento quando se cria uma hierarquia entre elementos (pais e filhos) e se trabalha com elementos desconexos, algo crucial para o desenvolvimento deste trabalho. De todo modo, ela permite que sejam criados os próprios algoritmos de organização.

4.4 COMPARATIVO COM OS TRABALHOS CORRELATOS

A ferramenta agregou diferentes aspectos de cada um dos trabalhos apresentados. A execução do algoritmo e representação visual é algo que está presente em todos os correlatos. Em relação ao trabalho de Halim *et al.* (2012), foi incorporada a ideia de um ambiente unificado, sendo a ferramenta extensível justamente com o objetivo de ser evoluída e se transformar em um único meio de linguagem de ensino para o estudante. Por outro lado, apesar do ambiente unificado e dos módulos disponíveis, o conteúdo disponibilizado não é flexível para o professor formular seu próprio material. Além disso, o foco foi prover um aprendizado autodidático, sendo que este trabalho foca na ferramenta como auxiliar ao ensino e não substituível para a figura de um professor.

Em relação ao trabalho de Klopfer *et al.* (2009), foi incorporado o aspecto do aprendizado incremental, no qual o estudante produz um conteúdo, o executa, coleta os resultados dessa execução e aprimora o processo trabalhando em

cima do conteúdo produzido, formando uma espécie de ciclo. Isso aumenta o nível de interatividade e a agilidade para a coleta de feedback. A codificação na ferramenta, no entanto, é realizada por meio de JavaScript, enquanto no trabalho correlato é por meio de blocos.

Em relação ao trabalho de Kopsch (2016), a principal característica que se destaca é a flexibilidade para desenvolver o material de aprendizado. Além de todo o processo de execução dos problemas, a possibilidade do professor criar os seus próprios problemas proporciona uma habilidade de gerenciar seu conteúdo de ensino. Por outro lado, assim como o trabalho de Klopfer *et al.* (2009), a programação é em blocos.

5 CONCLUSÕES

De acordo com os resultados obtidos, é notado um potencial da ferramenta para auxiliar no ensino de Ciência da Computação, em disciplinas como Algoritmos e Estrutura de Dados e Programação. Todos os estudantes responderam que a ferramenta foi útil para o seu aprendizado. Nos problemas relacionados a Bubble Sort e Lista Encadeada, por exemplo, 75% (3/4) afirmaram que houve uma grande contribuição para o aprendizado. No problema da lista dinâmica (Array List), aplicado de maneira mais superficial, 75% (3/4) afirmaram que houve uma contribuição ainda que não tão significativa. Essa característica aponta indícios de que a maneira que o problema é construído e aplicado tem um impacto significativo no resultado. Ou seja, a ferramenta de fato se comportaria como um mecanismo auxiliar e não autodidata. Ambos os professores destacam que os alunos terão dificuldades ao tentarem utilizar a ferramenta sozinhos. Inclusive apontam que mesmo aplicando em aula deverá haver uma instrução razoável aos estudantes para auxiliá-los.

A programação escrita demonstrou que pode ser dificultosa para alunos nas fases muito iniciais. Isso porque é necessária uma familiaridade com JavaScript, trabalhando com conceitos que vão além da etapa inicial como: vetores, objetos e *loops*. Apesar disso, a ferramenta flexibiliza a sua aplicação em aula de maneira que o próprio professor pode montar o seu problema e apresentar aos alunos apenas a representação visual, como uma espécie explicação conceitual. Um dos professores inclusive apontou que não aplicaria a ferramenta diretamente com os alunos, mas que utilizaria do recurso visual para auxiliar a explicação. A maneira que a ferramenta trabalhou com a abstração proporcionou tal liberdade. Ambos os professores ressaltaram que a flexibilidade oferecida é muito boa. Apesar disso, um dos professores afirma que o processo possui uma dificuldade regular, não sendo tão fácil, nem tão complexa. É entendível considerando que a criação do problema envolve a compreensão dos conceitos e recursos disponíveis, e a programação de cenários.

O recurso de visualização, acompanhado pela execução passo a passo, demonstrou ter forte contribuição no aprendizado. Nas duas declarações espontâneas obtidas pelos estudantes, foi exaltado o recurso da representação visual e sua contribuição para o entendimento do problema. Porém, alguns ajustes como organizar os elementos automaticamente e aprimorar o container de elementos primitivos ajudará ainda mais o processo. Outro ponto, é que durante a criação de um problema, a disponibilização de um editor rico na descrição do problema proporcionou ao educador a liberdade de incorporar na descrição de um problema, materiais educativos de apoio. Em relação a usabilidade, existe um período de adaptação. No caso dos estudantes, não é demorado, necessitando conhecer apenas a linguagem JavaScript e entender o funcionamento e a dinâmica da ferramenta. Para os professores, por sua vez, esse tempo pode ser um pouco maior, de aproximadamente três dias, considerando que além da execução existe todo o processo de construção de problemas.

A documentação disponibilizada também apresentou um papel essencial no impulsionamento da ferramenta durante a aplicação dos testes. Por parte dos alunos, o impacto foi pequeno, pois o nível de complexidade é menor. Mas no caso dos professores, onde existem mais funcionalidades e conceitos a serem abordados, a disponibilização de exemplos demonstrando o processo de criação de um problema com *gifs*, a explicação da visualização e dicas na utilização apresentaram um papel importante para que o resultado se tornasse utilizável.

Sendo assim, de um modo geral, é possível afirmar que a ferramenta atendeu a boa parte dos objetivos propostos, proporcionando uma representação visual, execução passo a passo, interatividade por meio da codificação e possibilidade de produção de material de ensino. Contudo, em relação ao objetivo principal, existem pontos a serem melhorados para oferecer uma melhor experiência de ensino aos professores e alunos, conforme destacado durante o trabalho. Além disso, é necessária a aplicação da ferramenta com uma quantidade maior de usuários, preferencialmente em cenários reais, como durante aulas de Algoritmos e Estruturas de Dados. A amostra coletada é pequena e consegue apenas destacar indícios e um potencial de utilização da ferramenta, que serão melhormente embasados com uma coleta de dados maior. Como sugestão de trabalhos de extensão é proposto: criação de um interpretador próprio que transpile para o modelo de linguagem atual; criação de uma linguagem portuol em cima do interpretador criado; execução passo a passo em cima do código sendo executado, além da representação visual; implementação de um algoritmo para organização automática dos elementos visuais; e implementação de um console que permita interação do usuário em tempo de execução.

REFERÊNCIAS

- AKVEO. **Ngx Admin**. [S.l.], [2019a?]. Disponível em: <https://github.com/akveo/ngx-admin>. Acesso em: 24 nov. 2020.
- AKVEO. **Nebular**: Customizable UI Kit, Auth & Security. [S.l.], [2019b?]. Disponível em: <https://github.com/akveo/ngx-admin>. Acesso em: 24 nov. 2020.
- ANDRADE, Adriner M. de. **Smalg Platform**. Plataforma auxiliar no processo educacional de programação. [S.l.], 2020. Disponível em: <https://adrinerandrade.github.io/smalg-platform>. Acesso em: 20 nov. 2020.
- COLBURN, Timothy; SHUTE, Gary. Abstraction in Computer Science. **Minds and Machines**, [S.l.], v. 17, n. 2, p. 169–184, jun. 2007.
- CONSORTIUM, Cytoscape. **Cytoscape.js**. [S.l.], [2020?]. Disponível em: <https://js.cytoscape.org/>. Acesso em: 24 nov. 2020.
- DANIELSIEK, Holger *et al.* Detecting and Understanding Students' Misconceptions Related to Algorithms and Data Structures. In: 43RD ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 12. 2012, Raleigh, NC, USA. **Proceedings...**, New York, NY, USA, p. 21-26. fev. 2012.
- DEBABI, W.; BENSEBAA, T. Using Serious Game to Enhance Learning and Teaching Algorithmic. **Journal of e-Learning and Knowledge Society**, [S.l.], v. 12, n. 2, p. 127–140, mai. 2016.
- FOUH, Eric *et al.* The Role of Visualization in Computer Science Education. **Computers in the Schools**, [S.l.], v. 29, n. 1-2, p. 95-117, 18 abr. 2012. Disponível em: <http://people.cs.vt.edu/~shaffer/CS6604/Papers/AVpedagogypost.pdf>. Acesso em: 12 abr. 2020.
- FUCCI, Davide *et al.* A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last?. **IEEE Transactions On Software Engineering**, [S.l.], v. 43, n. 7, p. 597-614, jul. 2017.
- GANASCIA, Jean-Gabriel. Abstraction of levels of abstraction. **Journal of Experimental & Theoretical Artificial Intelligence**, [S.l.], v. 27, n. 1, p. 23-35, ago. 2015.
- GIANNAKOS, Michail N. *et al.* Investigating Factors Influencing Students' Intention to Dropout Computer Science Studies. In: ACM CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 16. 2016, Arequipa Peru. **Proceedings...**, New York, NY, USA, p. 198–203, 1 jul. 2016.
- GIANNAKOS, Michail N. *et al.* Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. **Education and Information Technologies**, [S.l.], v. 22, p. 2365–2382, set. 2017.
- GITHUB. **Octokit Rest**. [S.l.], [2020?]. Disponível em: <https://github.com/octokit/rest.js>. Acesso em: 24 nov. 2020.
- HALIM, Steven *et al.* Learning Algorithms with Unified and Interactive Web-Based Visualization. **Olympiads in Informatics**, Singapore, v. 6, p. 53-68. 2012. Disponível em: <http://www.ioinformatics.org/oi/pdf/INFOL099.pdf>. Acesso em: 5 abr. 2020.
- KLOPFER, Eric *et al.* The Simulation Cycle: Combining Games, Simulations, Engineering and Science Using StarLogo TNG. **E-Learning and Digital Media**, [S.l.], v. 6, n. 1, p. 71-96, 1 jan. 2009. Disponível em: <https://journals.sagepub.com/doi/pdf/10.2304/elea.2009.6.1.71>. Acesso em: 6 abr. 2020.
- KOPSCH, Heloisa Kaestner. **FURBOT-WEB**: uma plataforma adaptativa para o ensino de programação. 2016. 119 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Regional de Blumenau, Blumenau, 2016. Disponível em: http://dsc.inf.furb.br/arquivos/tccs/monografias/2016_1_heloisa-kaestner-kopsch_monografia.pdf. Acesso em: 13 nov. 2020.
- MATTOS, Mauro M. *et al.* **Apostila de OO e FURBOT**. Blumenau, 2008. Disponível em: http://www.inf.furb.br/poo/furbot/files/Apostila_FURBOT.pdf. Acesso em: 13 nov. 2020.
- MICROSOFT. **Monaco Editor**. [S.l.], 2020a. Disponível em: <https://microsoft.github.io/monaco-editor>. Acesso em: 24 nov. 2020.
- MICROSOFT. **TypeScript**. [S.l.], 2020b. Disponível em: <https://www.typescriptlang.org/>. Acesso em: 24 nov. 2020.
- NAIK, Kshirasagar; TRIPATHY, Priyadarshi. **Software Testing and Quality Assurance**: Theory and Practice. Hoboken, New Jersey: Wiley & Sons, Inc., 2008. 616 p. ISBN 978-0-471-78911-6.
- QIAN, Yizhou; LEHMAN, James. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. **ACM Transactions on Computing Education**. New York, NY, USA, v. 18, n. 1, p. 5, 1 out. 2017.
- QUILL. **Quill**. 2020. Disponível em: <https://quilljs.com/>. Acesso em: 20 nov. 2020.
- STEIMANN, Friedrich; MAYER, Philip. Patterns of Interface-Based Programming. **Journal Of Object Technology**. [S.l.], v. 4, n. 5, p. 75-94. ago. 2005.
- SOMMERVILLE, Ian. **Engenharia de Software**. Tradução de Kalinka Oliveira e Ivan Bosnic. 9º. ed. São Paulo: Pearson Education do Brasil, 2011. ISBN 978-85-7936-108-1.
- SORVA, Juha *et al.* Students' ways of experiencing visual program simulation. **Computer Science Education**, [S.l.], v. 23, p. 207-238, set. 2013.
- YUAN, Xiaohong *et al.* Visualization Tools for Teaching Computer Security. **ACM Transactions on Computing Education**. New York, Ny, USA, v. 9, n. 4, p. 1-28. jan. 2010.

APÊNDICE A – REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS DA FERRAMENTA

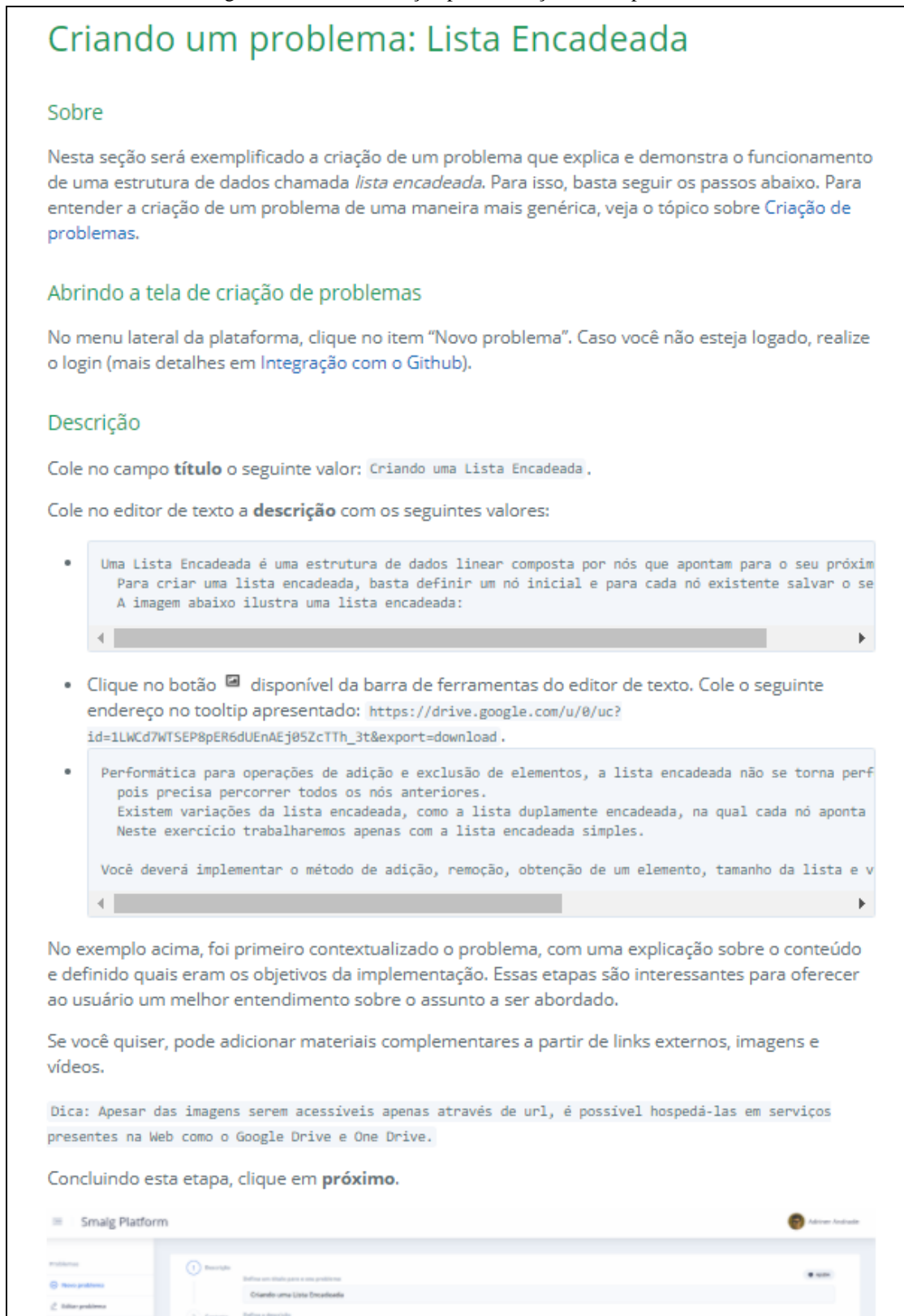
A ferramenta desenvolvida deve:

- a) possibilitar a criação e edição de problemas, composto por título, descrição, contrato, cenários e uma solução (RF);
- b) possibilitar a execução dos problemas (RF);
- c) disponibilizar um componente para ser utilizado pelo estudante na implementação do código (RF);
- d) criar um motor para a conversão do código em representação visual (RF);
- e) apresentar uma representação visual passo a passo da execução de uma implementação de uma interface (RF);
- f) disponibilizar modelos prontos com a implementação dos algoritmos de Array List, Linked List e Bubble Sort (RF);
- g) possibilitar a autenticação do usuário (RF);
- h) ser disponibilizado na web (RNF);
- i) utilizar o editor de código Monaco como componente de programação dentro da ferramenta web (RNF);
- j) utilizar a ferramenta Cytoscape para a representação visual (RNF);
- k) realizar autenticação com o Github (RNF);
- l) permitir o armazenamento dos problemas no Github (RNF);
- m) ter a interface web construída com o framework Angular (RNF).

APÊNDICE B – ILUSTRAÇÃO DA DOCUMENTAÇÃO DISPONIBILIZADA

Este apêndice ilustra a documentação disponibilizada. A Figura 14 demonstra uma página que tem como objetivo instruir o professor passo a passo na criação de um problema para implementação de uma lista encadeada.

Figura 14 – Documentação para a criação de um problema



Criando um problema: Lista Encadeada

Sobre

Nesta seção será exemplificado a criação de um problema que explica e demonstra o funcionamento de uma estrutura de dados chamada *lista encadeada*. Para isso, basta seguir os passos abaixo. Para entender a criação de um problema de uma maneira mais genérica, veja o tópico sobre [Criação de problemas](#).


Abrindo a tela de criação de problemas

No menu lateral da plataforma, clique no item “Novo problema”. Caso você não esteja logado, realize o login (mais detalhes em [Integração com o Github](#)).

Descrição

Cole no campo **título** o seguinte valor: `Criando uma Lista Encadeada`.

Cole no editor de texto a **descrição** com os seguintes valores:

- Uma Lista Encadeada é uma estrutura de dados linear composta por nós que apontam para o seu próximo. Para criar uma lista encadeada, basta definir um nó inicial e para cada nó existente salvar o seu endereço. A imagem abaixo ilustra uma lista encadeada:
- Clique no botão  disponível da barra de ferramentas do editor de texto. Cole o seguinte endereço no tooltip apresentado: `https://drive.google.com/u/0/uc?id=1LwCd7WTSEP8pER6dUEnAEj05ZcTTh_3t&export=download`.
- Perfomática para operações de adição e exclusão de elementos, a lista encadeada não se torna perfeita pois precisa percorrer todos os nós anteriores. Existem variações da lista encadeada, como a lista duplamente encadeada, na qual cada nó aponta para o próximo e para o anterior. Neste exercício trabalharemos apenas com a lista encadeada simples. Você deverá implementar o método de adição, remoção, obtenção de um elemento, tamanho da lista e validação de existência de um elemento.

No exemplo acima, foi primeiro contextualizado o problema, com uma explicação sobre o conteúdo e definido quais eram os objetivos da implementação. Essas etapas são interessantes para oferecer ao usuário um melhor entendimento sobre o assunto a ser abordado.

Se você quiser, pode adicionar materiais complementares a partir de links externos, imagens e vídeos.

Dica: Apesar das imagens serem acessíveis apenas através de url, é possível hospedá-las em serviços presentes na Web como o Google Drive e One Drive.

Concluindo esta etapa, clique em **próximo**.

Smaig Platform Administrar Perfil

Problemas

- Novo problema
- Editar problema

1. Descrição

Defina um título para o seu problema

Criando uma Lista Encadeada

2. Exemplos

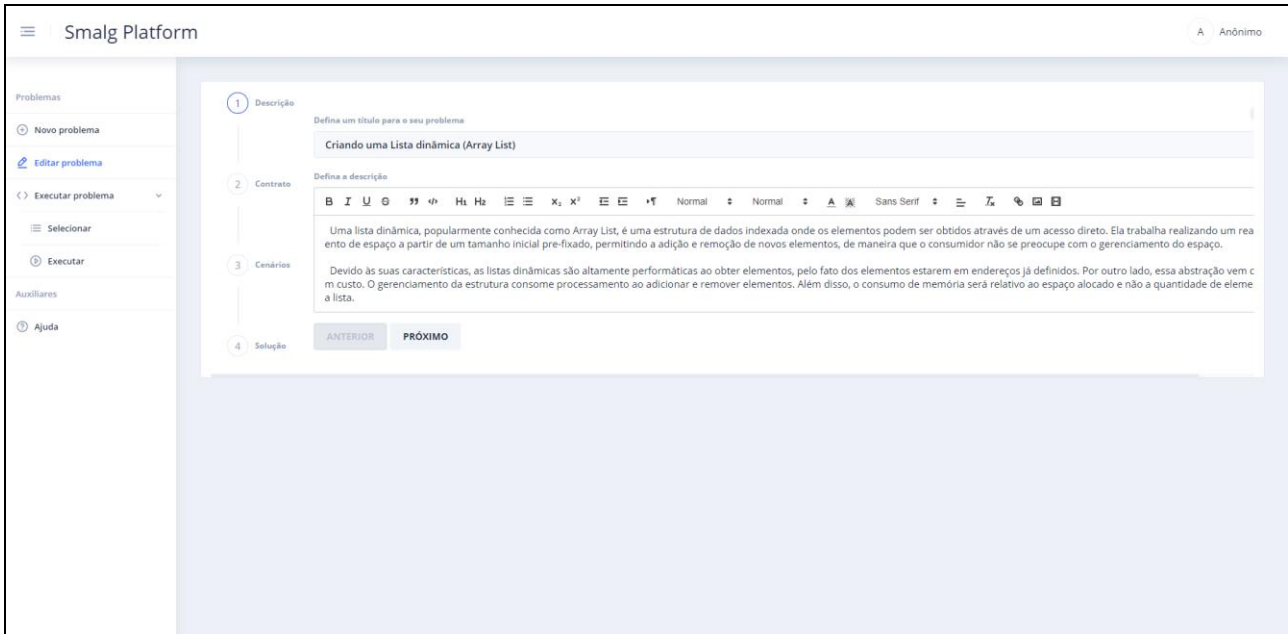
Defina exemplos

Fonte: elaborada pelo autor.

APÊNDICE C – ILUSTRAÇÃO DA CRIAÇÃO DE UM PROBLEMA

Este apêndice ilustra todas as etapas para a criação de um problema de maneira visual. Em todas as etapas, existe um botão de ajuda que redireciona para a documentação de criação de problemas. A Figura 15 ilustra a tela de descrição onde é informado o título e a descrição.

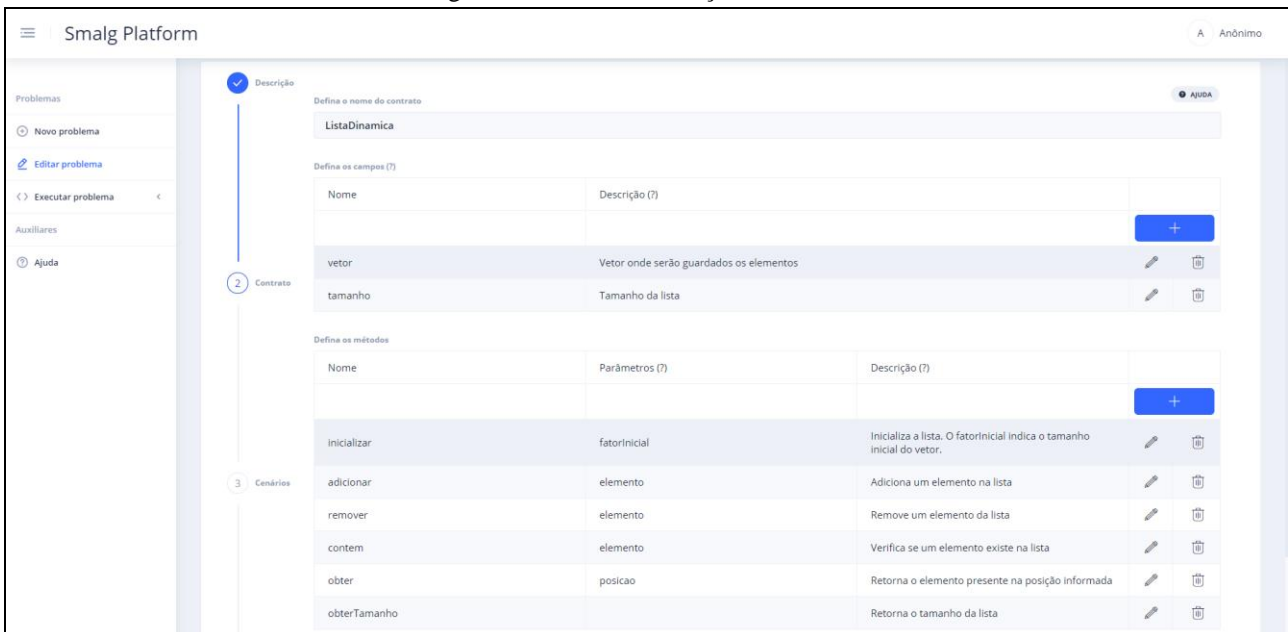
Figura 15 – Tela de descrição do problema



Fonte: elaborada pelo autor.

A Figura 16 ilustra a tela de definição do contrato no qual o professor deverá seguir. Nela é definida o nome do contrato, os nomes dos campos e o nome dos métodos.

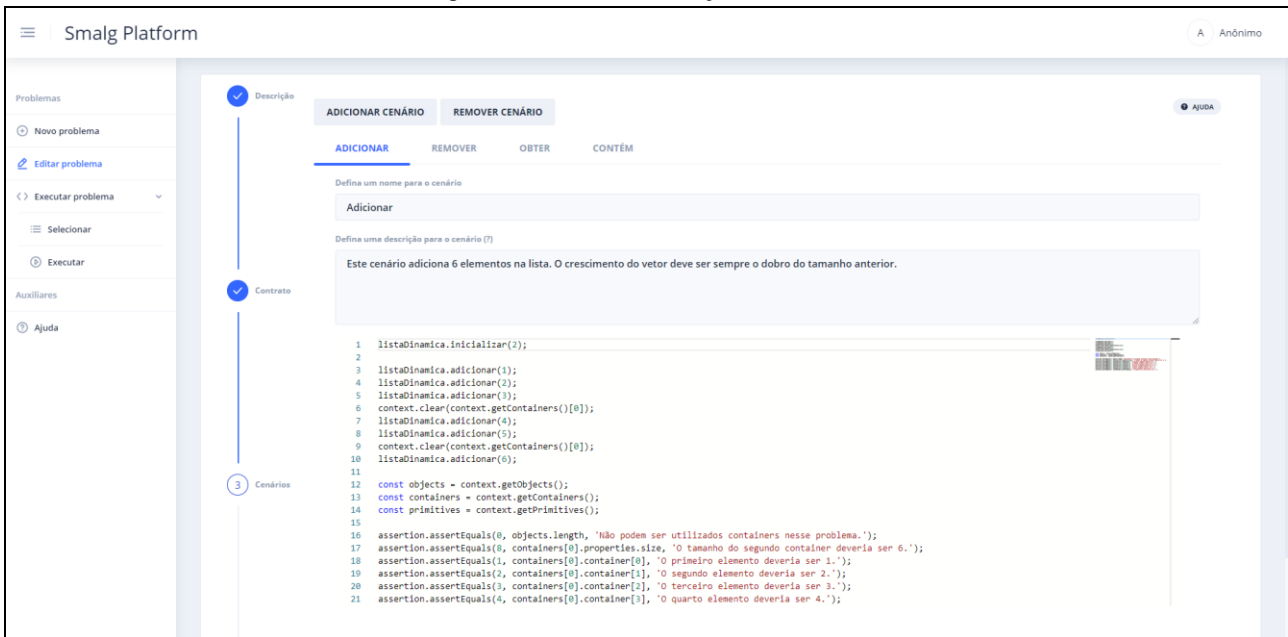
Figura 16 – Tela de definição de contrato



Fonte: elaborada pelo autor.

A Figura 17 por sua vez, ilustra a tela de definição de cenários. Vários cenários podem ser definidos, cada um deles com seu código de execução e suas assertivas.

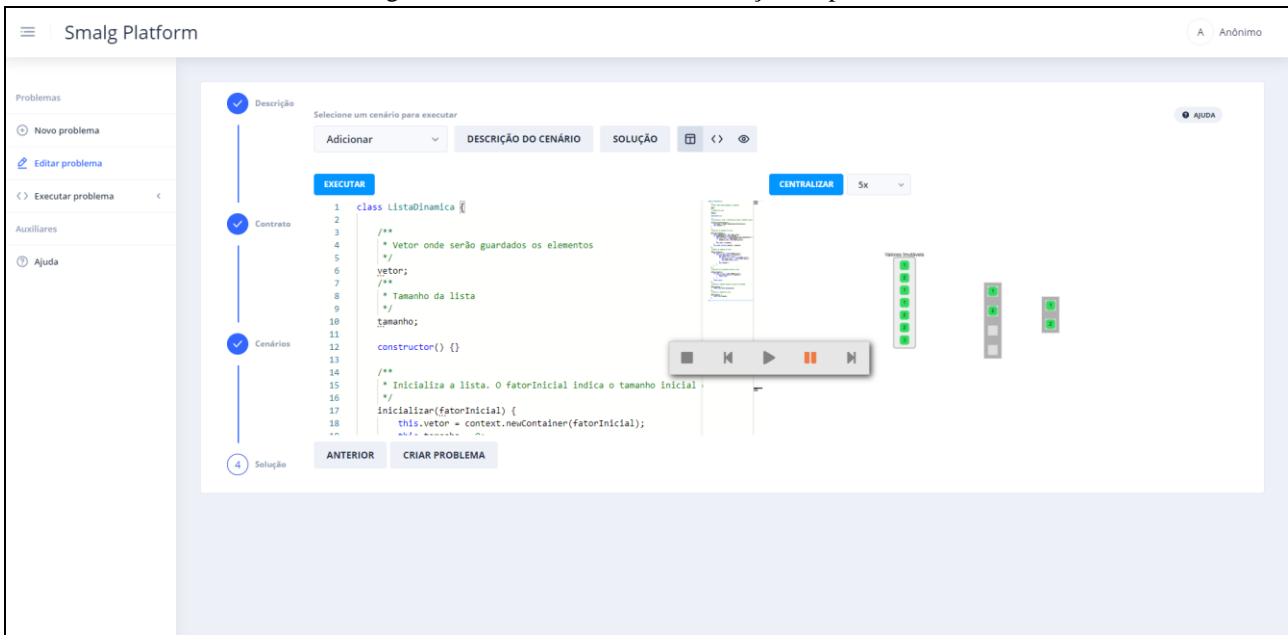
Figura 17 – Tela de definição de cenários



Fonte: elaborada pelo autor.

A Figura 18 ilustra a tela de definição da solução do problema. A partir dela, é possível testar os cenários que foram criados, e se preciso, voltar para realizar ajustes.

Figura 18 – Tela de cadastro da solução do problema



Fonte: elaborada pelo autor.

APÊNDICE D – OBJETOS DE CONTEXTO DE EXECUÇÃO E ASSERTIVAS

Este apêndice detalha os objetos de contexto de execução (utilizado na criação e execução de problemas) e de assertivas (utilizado na criação do problema). Além disso, demonstra um exemplo de código de cenário válido. O Quadro 11 detalha os métodos disponibilizados no objeto `context`. O Quadro 12 detalha os métodos disponibilizados no objeto `assertion`. O Quadro 13 demonstra um código válido para um cenário de uma lista dinâmica (popularmente conhecida, *array list*). O Quadro 14 demonstra um exemplo para a solução para um problema Bubble Sort.

Quadro 11 – Métodos disponibilizados pelo contexto de execução

Método	Descrição
<code>context.newObject();</code>	Cria um objeto.
<code>context.newContainer(t: int);</code>	Cria um container, sendo <code>t</code> o tamanho do container.
<code>context.getObjects();</code>	Obtém todos os objetos criados.
<code>context.getContainers();</code>	Obtém todos os containers criados.
<code>context.getPrimitives();</code>	Obtém todas as primitivas criadas.
<code>context.clear(e: any);</code>	Limpa um elemento do contexto, sendo <code>e</code> o elemento.

Fonte: elaborado pelo autor.

Quadro 12 – Métodos disponibilizados pelo objeto de assertivas

Método	Descrição
<code>assertion.assertEquals(e1: any, e2: any, m: string);</code>	Verifica se dois elementos são iguais, sendo <code>v1</code> o primeiro elemento, <code>v2</code> o segundo elemento e <code>m</code> a mensagem de erro.
<code>assertion.assertTrue(e: any, m: string);</code>	Verifica se o elemento passado é verdadeiro, sendo <code>e</code> o elemento e <code>m</code> a mensagem de erro.
<code>assertion.assertFalse(e: any, m: string);</code>	Verifica se o elemento passado é falso, sendo <code>e</code> o elemento e <code>m</code> a mensagem de erro.
<code>assertion.fail(m: string);</code>	Força o erro na execução, sendo <code>m</code> a mensagem de erro.

Fonte: elaborado pelo autor.

Quadro 13 – Exemplo de um cenário para um problema de lista dinâmica

<pre> listaDinamica.inicializar(2); listaDinamica.adicionar(1); listaDinamica.adicionar(2); listaDinamica.adicionar(3); context.clear(context.getContainers()[0]); listaDinamica.adicionar(4); listaDinamica.adicionar(5); context.clear(context.getContainers()[0]); listaDinamica.adicionar(6); const objects = context.getObjects(); const containers = context.getContainers(); const primitives = context.getPrimitives(); assertion.assertEquals(0, objects.length, 'Não podem ser utilizados containers nesse problema.');</pre>
--

Fonte: elaborado pelo autor.

Quadro 14 – Exemplo de solução para um problema Bubble Sort

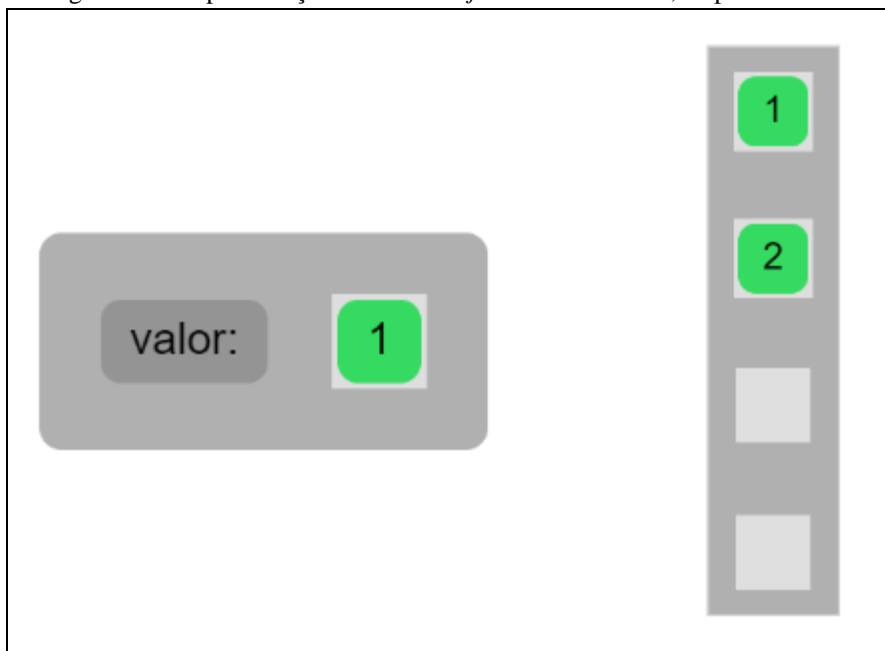
```
class BubbleSort {  
  
  constructor() {}  
  
  ordenar(vetor) {  
    // Variável que otimiza a ordenação parando-a  
    // caso não tenha mais itens a serem ordenados  
    let trocou = false;  
  
    let ultimo_indice = 0;  
  
    do {  
      trocou = false;  
      for (let i = vetor.size() - 2; i >= ultimo_indice; i--) {  
        const elemento_1 = vetor.get(i + 1);  
        const elemento_2 = vetor.get(i);  
        if (elemento_1 < elemento_2) {  
          vetor.set(i + 1, elemento_2);  
          vetor.set(i, elemento_1);  
          trocou = true;  
        }  
      }  
      ultimo_indice++;  
    } while (trocou);  
  }  
}
```

Fonte: elaborado pelo autor.

APÊNDICE E – ELEMENTOS APRESENTADOS NA VISUALIZAÇÃO

Este apêndice detalha os elementos que são utilizados na representação visual da execução do código. A Figura 19 ilustra os elementos *objeto* e *container*, respectivamente. A Figura 20 ilustra o elemento onde são colocados os tipos primitivos criados. A Figura 21 ilustra um relacionamento por referência entre dois objetos.

Figura 19 – Representação visual do *objeto* e do *container*, respectivamente



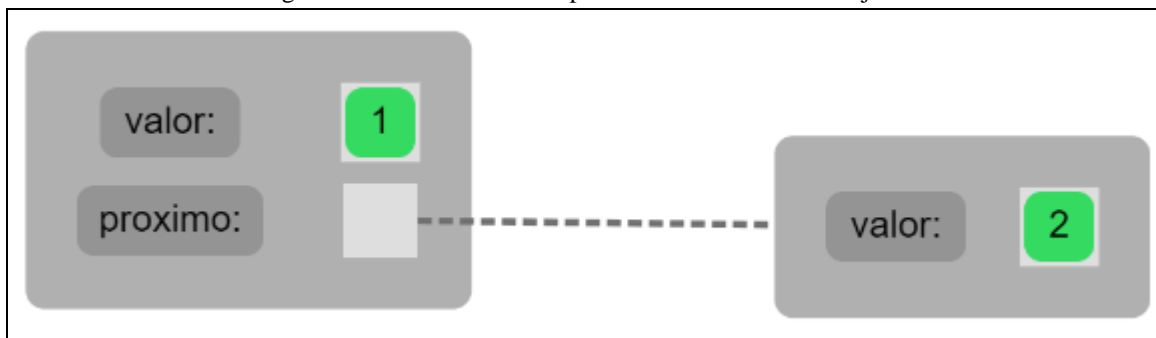
Fonte: elaborada pelo autor.

Figura 20 – Elemento onde ficam os tipos primitivos (valores imutáveis)



Fonte: elaborada pelo autor.

Figura 21 – Relacionamento por referência entre dois objetos



Fonte: elaborada pelo autor.

APÊNDICE F – QUESTIONÁRIOS DE TESTES COMPLETOS

Este apêndice tem como objetivo demonstrar os questionários aplicados nos alunos e professores por completo. O Quadro 15 e o Quadro 16 apresentam as perguntas aplicadas e as respostas coletadas em detalhe.

Quadro 15 – Questionário completo aplicado com os alunos

Pergunta	Respostas disponíveis
Tendo em vista o problema Bubble Sort, você considera que a ferramenta te auxiliou a entender o conceito envolvido?	Sim, muito. – 75% (3/4). Sim, um pouco. – 25% (1/4). Não, pois já compreendia. – 0% (0/4). Não, continuo sem compreender. – 0% (0/4)
Tendo em vista o problema da lista encadeada, você considera que a ferramenta te auxiliou a entender o conceito envolvido?	Sim, muito. – 75% (3/4). Sim, um pouco. – 25% (1/4). Não, pois já compreendia. – 0% (0/4). Não, continuo sem compreender. – 0% (0/4)
Tendo em vista o problema lista dinâmica (Array List), você considera que a ferramenta te auxiliou a entender o conceito envolvido?	Sim, muito. – 25% (1/4). Sim, um pouco. – 75% (3/4). Não, pois já compreendia. – 0% (0/4). Não, continuo sem compreender. – 0% (0/4)
Como você avalia o grau de dificuldade para utilizar a ferramenta?	Não tive dificuldades. – 25% (1/4). Baixo. – 75% (3/4). Médio. – 0% (0/4). Alto. – 0% (0/4). Muito alto. – 0% (0/4).
Como você descreveria sua curva de aprendizado para utilizar a ferramenta?	Curta, um dia. – 100% (4/4). Média, três dias. – 0% (0/4). Longa, mais de três dias. – 0% (0/4). Não sei dizer. – 0% (0/4).
Como você avalia a documentação disponibilizada?	Muito boa. – 25% (1/4). Boa. – 50% (2/4). Regular. – 25% (1/4). Ruim. – 0% (0/4). Péssima. – 0% (0/4). Não cheguei a ler. – 0% (0/4).
No geral, acredito que a ferramenta:	Foi útil para o meu aprendizado. – 100% (4/4). Não contribuiu em nada para o meu aprendizado. – 0% (0/4).

Fonte: elaborado pelo autor.

Quadro 16 – Resultado dos testes aplicados com os professores

Pergunta	Respostas
Como você avalia a flexibilidade para criação de problemas?	Muito boa. – 100% (2/2). Boa, porém poderia ser um pouco mais flexível. – 0% (0/2). Regular, o modelo é um pouco engessado. – 0% (0/2). Ruim, não me ofereceu muita flexibilidade. – 0% (0/2). Péssima, não ofereceu flexibilidade nenhuma. – 0% (0/2).
Como você avalia a facilidade para a criação de problemas, considerando a flexibilidade proposta?	Muito boa. – 50% (1/2). Boa, porém poderia ser um pouco mais fácil. – 0% (0/2). Regular, não tão fácil nem tão complexo. – 50% (1/2). Ruim, é um processo um pouco complexo. – 0% (0/2). Péssima, extremamente difícil. – 0% (0/2).
Como você avalia o potencial da plataforma para ensinar de modo autodidata para os alunos?	Muito bom. – 0% (0/2). Bom, porém o aluno terá algumas dificuldades. – 50% (1/2). Regular, o aluno terá dificuldades medianas para usar. – 50% (1/2). Ruim, o aluno terá várias dificuldades para usar. – 0% (0/2). Péssimo, o aluno não terá condições de usar. – 0% (0/2).
Como você avalia o potencial da plataforma para ser utilizada como ferramenta auxiliar ao professor no ensino?	Muito bom. – 100% (2/2). Bom. – 0% (0/2). Regular. – 0% (0/2). Ruim. – 0% (0/2). Péssimo. – 0% (0/2).
Você utilizaria a ferramenta aplicando	Sim. Acredito que a interação dos alunos pode agregar ao

diretamente com os alunos?	aprendizado. – 50% (1/2) Não, mas montaria meus próprios problemas e apresentaria a visualização para auxiliar na explicação. – 50% (1/2). Não utilizaria a ferramenta. – 0% (0/2).
Se você for aplicar diretamente com os alunos, como você avalia o nível de acompanhamento e instrução que deverá ser realizado por parte do professor?	Nenhum. – 0% (0/2). Quase nenhum. – 0% (0/2). Razoável. – 100% (2/2). Um acompanhamento integral. – 0% (0/2). Não utilizaria com os alunos. – 0% (0/2).
Como você classifica a visualização apresentada?	Muito boa. – 50% (1/2). Boa. O conceito pode ser entendido, mas tem alguns pontos que poderiam melhorar. – 50% (1/2). Regular. – 0% (0/2). Ruim. – 0% (0/2). Péssima. – 0% (0/2).
Como você classifica a curva de aprendizado para conseguir utilizar a ferramenta, por parte do professor?	Curta, um dia. – 50% (1/2). Média, três dias. – 50% (1/2). Relativamente longa, uma semana. – 0% (0/2). Longa, mais de uma semana. – 0% (0/2).
No geral, considero a ferramenta como uma possibilidade de ensino:	Muito boa. – 50% (1/2). Boa. – 50% (1/2). Regular. – 0% (0/2). Ruim. – 0% (0/2). Péssima. – 0% (0/2).
Como você avalia a qualidade da documentação da plataforma?	Muito boa – 100% (2/2). Boa – 0% (0/2). Regular – 0% (0/2). Ruim – 0% (0/2). Péssima – 0% (0/2).

Fonte: elaborado pelo autor.