GERAÇÃO AUTOMATIZADA DE GABARITO E CORREÇÃO DE EXERCÍCIOS EM AMBIENTE FURBOT PARA O ENSINO DO PENSAMENTO COMPUTACIONAL

Francisca Edyr Xavier, Luciana Pereira de Araújo Kohler – Orientadora

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

fexavier@furb.br, lpa@furb.br

Resumo: *Este artigo apresenta uma ampliação da plataforma Furbot para automatizar a geração e a correção de exercícios em ambiente Furbot para o ensino do pensamento computacional. Para o desenvolvimento foi utilizada a última versão disponível do framework Furbot na linguagem Java, além da biblioteca Commons-digester-1.8 para leitura de arquivos XMLs. O algoritmo de Dijkstra foi utilizado para encontrar os caminhos mínimos entre os objetivos do mapa. A combinação de caminhos de menor tamanho é transformada em uma sequência de instruções para executar esse percurso, gerando o gabarito do exercício. A correção compara os resultados obtidos pelo o aluno com o gabarito. Após testes realizados com crianças, teve-se como resultado que elas se sentiram motivadas a buscar melhores soluções nos exercícios por possuir um gabarito com a comparação. Ainda, identificou-se que há uma demora na geração ao colocar muitos objetivos no mapa para gerar o gabarito, sendo está uma limitação do trabalho.*

Palavras-chave*: Pensamento computacional. Furbot. Dijkstra. Menor caminho. Gabarito. Correção de exercicios*

1 INTRODUÇÃO

A computação é um dos fatores responsáveis por acelerar a ocorrência de mudanças em todas as áreas do conhecimento, exigindo novas competências e habilidades cognitivas (RAABE, 2017). Dessa forma, é reconhecido pela a Sociedade Brasileira de Computação (SBC) que é fundamental a inserção da computação na Educação Básica, pois os conhecimentos básicos de computação são de grande relevância para a inserção de indivíduos na sociedade conterrânea, além de que as habilidades e competências adquiridas pelo o pensamento computacional potencializam a capacidade de soluções de problemas (RAABE, 2017).

Com intuito de incentivar o uso de tecnologias em escolas, bem como aplicar o ensino de pensamento computacional, foi desenvolvida a plataforma Furbot utilizando de base o *framework* original do Furbot utilizado desde 2008 para o ensino da disciplina de introdução de computação nos cursos de Ciência da Computação e Sistemas da Informação na Universidade Regional de Blumenau (FURB) (ARAÚJO; MATTOS, 2018). A plataforma foi desenvolvida para disponibilizar um ambiente atrativo e facilitador para o uso na Educação Básica como ferramenta de ensino de programação, que introduz conceitos tais como, programação sequencial, uso de condicionas e laços de repetição. Na plataforma Furbot, os alunos comandam um robô através de um mapa bidimensional, no qual devem desviar os obstáculos e coletar itens para concluir os objetivos propostos pelo enunciado do exercício. Os exercícios do Furbot, assim como a configuração de todo o cenário é realizada em um documento eXtensible Markup Language (XML), os quais podem ser criados por um usuário através de um editor disponível na plataforma.

Ao concluir o objetivo, a plataforma informa para o aluno se completou a missão baseada na quantidade de tesouros coletados. Ainda, ela produz uma pontuação com base no código-fonte produzido. Caso este é igual ao código fonte definido como um gabarito do exercício dentro do XML, o aluno recebe uma bonificação maior. Contudo, há várias formas de resolver um mesmo exercício no Furbot. Na plataforma atual, a produção desse gabarito é feita de forma manual no arquivo XML. Dessa forma, a elaboração de um novo exercício, além do processo de correção, se torna limitada, pois os gabaritos e as verificação das atividades deve ser feitas de forma manual. Esse tempo poderia ser direcionado para alunos que possuem dificuldades para alcançar uma resolução ou para aprimoramento nas dinâmicas de ensino (MATTOS et al, 2017).

A versão ampliada do *framework* e da plataforma Furbot apresentada neste artigo, tem como objetivo automatizar o processo de geração dos gabaritos criados pela ferramenta de geração de mundo, através do uso do algoritmo de menor caminho Dijkstra, gerando a solução ótima para os objetivos do mapa e automatizar o processo de correção dos exercícios. Além disso, tem-se como objetivos específicos auxiliar no desenvolvimento do pensamento computacional, exercitando análise crítica dos alunos, os quais por conta própria podem tentar buscar um melhor resultado e formar alternativas de soluções. Outro objetivo também está em facilitar o uso da plataforma em sala de aula, simplificando o processo de criação e de correção dos exercícios do Furbot.

Assim, esse artigo apresenta como foi feito o processo para efetuar a geração de gabaritos automatizados dos exercícios criados na ferramenta de geração de mundo do Furbot e como o algoritmo de Dijkstra foi utilizado para auxiliar na busca do menor caminho, buscando assim a solução ótima dos mapas. Além disso, efetua a comparação dos gabaritos gerados com as resoluções dos alunos, automatizando o processo de correção. São demonstrados também os testes de performe da ferramenta de geração de gabarito, bem como respostas obtidas por alunos de uma turma de programação em Fubot que utilizou a ferramenta.

Dessa forma, o artigo está dividido nas seguintes seções. A seção 2 apresenta a fundamentação teórica. A seção 3 descreve o desenvolvimento das ferramentas. A seção 4 demonstra os resultados obtidos. Por fim, a seção 5 relata as conclusões dos resultados alcançados em relação aos objetivos definidos.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção apresentas os assuntos que fundamentam este artigo. A subseção 2.1 apresenta alguns conceitos sobre a aplicação da computação na educação básica. A subseção 2.2 descreve os conceitos de grafos e o algoritmo de Dijkstra utilizado para o desenvolvimento da geração automatizada dos gabaritos. A subseção 2.3 apresenta a visão geral dos trabalhos correlatos. Por fim, a subseção 2,4 realiza uma breve descrição da plataforma Furbot utilizada como base para o trabalho apresentado neste artigo.

2.1 COMPUTAÇÃO NA EDUCAÇÃO BÁSICA

Segundo Raabe (2017, p. 1), "A Sociedade Brasileira de Computação (SBC) entende que é fundamental e estratégico para o Brasil que conteúdos de Computação sejam ministrados na Educação Básica". Algumas abordagens veem sendo ressaltadas em vários trabalhos como apontado por Oliveira (2014), que busca reforçar a ideia da aplicação do ensino de computação nos anos iniciais, introduzindo a programação através do Scratch. Outra abordagem apresentada no trabalho de Araújo *et al* (2015) é a aplicação da metodologia de ensino que utiliza o aluno como protagonista, incitando-o a pensar no problema a ser resolvido.

Para que um indivíduo esteja pronto para a sociedade contemporânea, ele deverá estar preparado para entender e tirar proveito dos avanços providos pela relação da Computação com as outras áreas de conhecimento, dessa forma a computação se torna um conhecimento tão importante quanto Matemática, Filosofia, Física e outras Ciências (RAABE, 2017). Os conhecimentos relacionados a Computação podem ser divididos em três eixos: Pensamento computacional; Mundo Digital; e Cultura Digital (RAABE, 2017).

Nesse contexto, Wing (2006) ressalta a importância do pensamento computacional, pois é uma habilidade fundamental para potencializar a capacidade de resolução de problemas de um indivíduo. Para resolver problemas complexos é necessário entender que é possível criar uma representação abstrata para facilitar a construção de soluções, podendo as descrever em passos, possibilitando uma análise crítica do objetivo e buscando soluções melhores (WING, 2006). As gerações automáticas dos gabaritos no *framework* Furbot, permite que os alunos além de desenvolverem suas próprias soluções, possam também observar as soluções ótimas dos gabaritos, exercitando sua análise crítica, assim como aprimorando as outras habilidades do pensamento computacional.

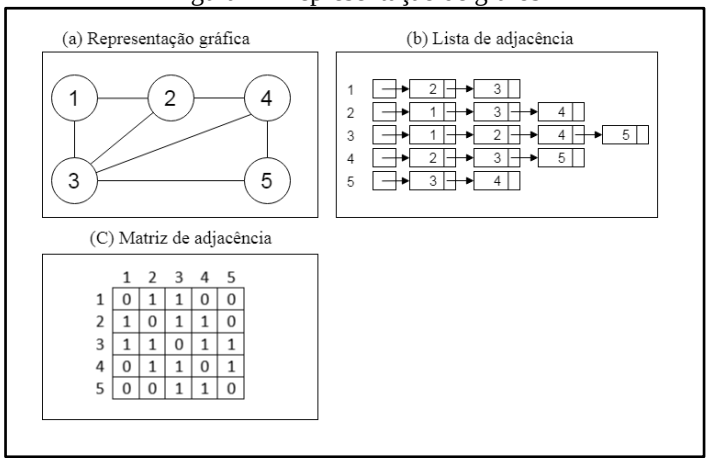
Dentro do ramo de Mundo Digital a codificação é descrita como a compreensão de como ela pode ser descrita e armazenada (RAABE, 2017). Todas as estruturas dos exercícios do Furbot são gravadas em arquivos XML, assim como o gabarito. Os exercícios feitos através da ferramenta de geração do mundo podem ser visualizados em sua forma textual dentro das pastas do jogo.

Por sua vez, a Cultura Digital tem como um de seus pilares a fluência tecnológica (RAABE, 2017). Portanto, vem se utilizado formas lúdicas para introdução de computação para crianças, assim como as abordagens feitas pela Code.org (2018b). Essas abordagens trazem exercícios autoexplicativos que aumentam o grau de dificuldade progressivamente, incluindo novos comandos ou aumentando a complexidade do exercício, além de que traduzem problemas computacionais de forma visual para cativar o aluno utilizando personagens conhecidos. Zanchet (2015) considera a diversão parte da Cultura Digital, pois aumenta o engajamento e a motivação, o que facilita a aprendizagem.

2.2 GRAFOS E O ALGORITMO DE DIJKSTRA

Grafos são estruturas de dados utilizadas na computação para representar um conjunto de elementos e o relacionamento entre eles, de forma que seja possível aplicar soluções de diversos problemas computacionais, dentre deles o problema de caminho mínimo (MENEZES JR *et al.*, 2012). Um grafo `G = (V, E)` configura-se com um conjunto finito não vazio de elementos, chamados de Vértices (`V`) e `E` um conjunto de pares não ordenados de elementos distintos, para grafos não dirigidos, ou de de grafos ordenados, pertencentes à `V` chamados de arestas (CORMEM *et al*., 2002). As arestas `E = (u,v)` representam uma relação de adjacência entre os vértices `u` e v (CORMEM *et al*., 2002). Dentre as formas de representação para um grafo `G = (V, E)` tem-se: Figura 1 (a) representação gráfica, Figura 1 (b) representação da lista de adjacência e Figura 1 (c) representação de uma matriz de adjacência, todas representando o mesmo grafo.

Figura 1 - Representação de grafos



Fonte: elaborada pela autora.

É possível notar pela representação da

Figura 1 (a) que o grafo não possui nenhuma aresta direcionada, nem possui múltiplas arestas ou laços, representando assim um grafo simples e não orientado. A

Figura 1 (b) apresenta a lista de adjacência numeradas de 1 à 5, cada uma com um conjunto composto por `|V|` listas de `adj[u]`, sendo `V` o conjunto de vértices pertencente ao grafo `G`. Cada lista `adj[u]` é formada pelos vértice (ou ponteiro para esses vértices) pertencente aos grafos que possuem relação de adjacência ao vértice o qual a lista de adjacência representa (CORMEM *et al*., 2002). Na Figura 1 (c) é visível a matriz `A =` (a_{ij}) de tamanho `|V| x |V|`, sendo `V` um conjunto de vértices pertencente ao grafo `G`, caso a existência de adjacência `i` com `j`, $(a_{ij}) = 1$, caso contrário é 0 (CORMEM *et al*., 2002). Matrizes de adjacência são estruturas mais simples, mas listas de adjacências ocupam menos para armazenar em casos de grafos esparsos, isso é quando a quantidade de arestas existentes `|E|` for muito inferior a quantidade de arestas máximas possível.

O Algoritmo de Dijkstra é um dos algoritmos utilizados para se encontrar o menor caminho entre dois pontos de um grafo. O Dijkstra é utilizado em grafos com arestas ponderadas com valores não negativos. Seu custo computacional é de `O((m+n)*log n)`, em que `n` é o número de arestas e `m` é o número de vértices. Isto é equivalente a uma busca em largura quando todos os vértices possuem o mesmo valor de atributo (KOCAY e KREHER, 2005). Em vantagem a busca de largura, o uso do algoritmo de Dijkstra permite que caso haja uma mudança na distribuição de valores do grafo o mesmo terá performasse superior a uma busca em largura, o que o torna mais flexível, sendo preferível para cenários que podem haver esse tipo de mudança. Tendo um grafo `G = (V, E)`, se define a função `DIJKSTRA(G, w, s)` em que `w` e `s` são vértices pertencentes `V`, sendo `w` o vértice de destino e `s` o vértice de origem. Esta função pode ser visualizada no Quadro 1.

Quadro 1 - Algoritmo de Dijkstra

```
1  DIJKSTRA(G, w, s)
2        INITIALIZE(G, s)
3        S ← Q
4        Q ← V[G]
5        while Q ≠ ∅
6        do u ← EXTRACT_MIN(Q)
7        S ← S ∪ {u}
8        for cada vértice v ϵ Adj[u]
9              do RELAX(u, v, w)
```

Fonte: adaptado de Cormen *et al*. (2002, p. 471).

A função de inicialização `INITIALIZE(G,s)` invocada na `linha 2` do Quadro 1, atribui os valores iniciais da operação, para o vértice de origem `s` recebe o custo zero e os demais vértices do grafo custo infinito. Na `linha 3`, `S` é iniciado como um conjunto vazio. Ele irá receber o caminho de vértices com o custo mínimo, `Q` (`linha 4`) representa a fila de vértices pertencente a `G` inicializado com todos vértices do grafo. O grafo contém os vértices que ainda não foram visitados. Em seguida, na `linha 5`, enquanto `Q` for diferente de vazio, a função `EXTRACT_MIN(Q)` busca o vértice não percorrido com menor custo representado por `u` (`linha 6`). Na primeira vez que executar, o vértice com menor custo será o `s`, o qual foi inicializado com custo zero. Esse vértice é removido da lista de `Q` e adicionado na lista `S` (`linha 7`). Para cada vértice `v` adjacente a `u` (`linha 8`) é efetuado o processo de relaxamento `RELAX(u,v,w)` (`linha 9`), responsável por atualizar o predecessor e o custo de `v`. Caso o custo de `u` mais o custo de `w` (a distância de `u` para `v`)

seja menor que o custo atual de v, o custo de v será w mais o custo de u e u será atribuído como o predecessor de menor distancia de v.

2.3 TRABALHOS CORRELATOS

Esta seção descreve três trabalhos correlatos ao trabalho apresentado neste artigo. O Quadro 2 aborda uma ferramenta para ensino de programação adaptativa baseada no *framewoek* Furbot (KOPSCH, 2016).

Quadro 2 - Furbot-WEB

Referência	Kopsch (2016)
Objetivos	Ser uma ferramenta de ensino de programação adaptativa web.
Principais funcionalidades	Utiliza de um ambiente de programação em bloco. Recomenda exercícios para os alunos utilizando recomendação com filtragem de conteúdo, com base no conhecimento de cada aluno. Possui um processo de automatização de aplicação de exercícios, em que todos os alunos cadastrados em uma turma têm acesso ao exercício ao mesmo tempo. Possui um ambiente para que os professorem possam cadastrar exercícios e definir as propriedades do mundo, além de possuir um ambiente para os professores efetuarem os processos de correções dos exercícios.
Ferramentas de desenvolvimento	JavaScript e Hyper Text Markup Language (HTML5)
Resultados e conclusões	O ambiente do Furbot-WEB mostrou eficiência em recomendar os exercícios com base no conhecimento do aluno. Ambiente web mostrou certa lentidão ao executar exercícios que utilizam de laços de repetição, o qual aumenta com o número de laços utilizados.

Fonte: elaborado pela autora.

No Quadro 3 é apresentada uma ferramenta para ensino de programação da Code.org (2018a) , o qual possui uma série de exercícios introdutórios.

Quadro 3 - Programação Anna e Elsa

Referência	Code.org (2018a)
Objetivos	Ser uma ferramenta de ensino de programação gratuita que busca expandir o acesso da informática na escola e aumentar a participação de mulheres e minorias.
Principais funcionalidades	Possui 20 exercícios introdutórios de programação fixos que utilizam personagens infantis populares, os quais devem ser resolvidos através do uso de blocos de programação. Após o aluno concluir o objetivo, retorna o desempenho do aluno e o código gerado em JavaScript. Possui um ambiente Web online e um ambiente offline. A personagem fornece dicas e instruções para a resolução da aplicação, incluindo se era possível se resolver com menos comandos.
Ferramentas de desenvolvimento	Informação não foi localizada.
Resultados e conclusões	A programação com Anna e Elsa introduz vários conceitos de programação, tais como programação sequencial, uso de laços de repetições e uso de funções. Oferece a validação automática após a conclusão dos exercícios. Os exercícios disponíveis na aplicação são limitados, mas oferecem instruções direcionadas para cada uma atividade. Como não se tem entrada de código escrito, não é possível testar o código gerado em JavaScript.

Fonte: elaborado pela autora.

O Quadro 4 apresenta um jogo de quebra cabeça social, o qual possui um ambiente próprio em que os usuários elaboram seus próprios quebra cabeças.

Quadro 4 - Robozzle

Referência	Ostrovsky (2009)
Objetivos	Ser um jogo de quebra Cabeça que permita que os membros da comunidade criem e compartilhem seus próprios jogos.
Principais funcionalidades	Permite resolver quebras cabeças, no qual se utiliza comandos representados por símbolos para se mover um robô em forma de seta por um mapa em ladrilhos. O ambiente social disponibiliza um ambiente para elaboração e compartilhamento de quebra cabeças criados pelos os usuários. Ao se concluir um quebra cabeça é possível comparar a quantidades de comandos usados na resposta com a de outros jogadores.
Ferramentas de desenvolvimento	Web SilverLigth, e possui uma versão em beta na linguagem Javascript

Resultados e conclusões	O principal foco do Robozzle é o entretenimento. A ferramenta permite a prática de conceitos de programação tais como programação sequencial, lógica condicional, uso de funções; e programação recursiva. No ambiente de elaboração de quebra cabeças deve-se fornecer uma resolução válida, a qual o deve ser informada de forma manual. O usuário pode então comparar seu desempenho com a quantidade de comandos usados na resposta fornecida pelo o criador do puzzle ou pelos outros usuários que concluíram o mesmo.

Fonte: elaborado pela autora.

2.4 VERSÃO ANTERIOR DA PLATAFORMA FURBOT

O Furbot teve a sua origem em 2008 a partir do desenvolvimento de um *framework* Java. Desde então, esse *framework* vem sendo utilizada nos cursos de Ciências da Computação e de Sistemas de Informação como apoio para o ensino introdutório de programação. A partir deste *framework*, foi desenvolvida uma plataforma para o ensino do pensamento computacional. Essa plataforma do Furbot foi desenvolvida a partir de um projeto de extensão iniciado em 2017, com intuito de ser utilizado em escolas de ensino básico, tendo seu ambiente sendo transformado em uma plataforma de ensino de programação (ARAÚJO; MATTOS, 2018).

Nessa plataforma se introduz os conceitos de programação sequencial, utilização de condicionais e no nível de dificuldade difícil o uso de laços de repetição utilizando características de um jogo. A partir da programação de um robô, o objetivo é fazer com que o mesmo chegue até um destino coletando tesouros, evitando alienígenas e desviando das paredes. O aluno pode escolher um nível de dificuldade e em seguida deve escolher um exercício correspondente a este nível. Então é exibido o enunciado do exercício com a descrição de qual é o objetivo do cenário. O ambiente do jogo é demonstrado na Figura 2.

Figura 2 - Ambiente de Jogo do Furbot

Fonte: elaborada pela autora.

O ambiente do Furbot é formado pelo cenário do mapa, o qual é uma tabela com o tamanho definido pelo exercício, dividido em linhas e colunas. Na Figura 2 é possível observar os tesouros e anjos, sendo os itens coletáveis espalhados no mapa, além dos aliens e paredes que são os obstáculos a serem desviados. Ainda no mapa tem-se o Furbot, personagem programável para percorrer o caminho e alcançar o objetivo do exercício. No lado direito é visível a quantidade de itens atualmente coletados e a pontuação. Na parte inferior da tela se encontram os botões para voltar para a seleção de exercícios, abrir o console para programar, pausar o jogo e visualizar o enunciado do problema, respectivamente.

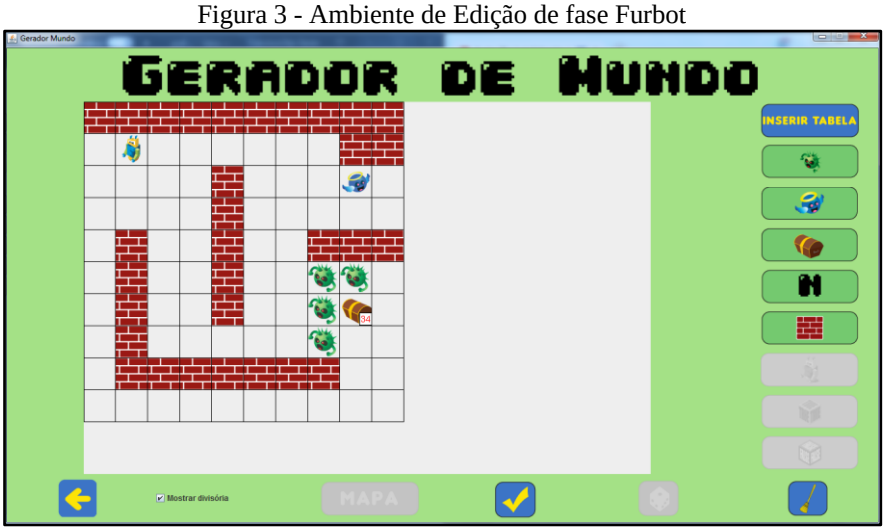
Para resolver o exercício, o aluno deve utilizar comandos para movimentar o robô. No nível fácil é possível movimentar o robô através das setas do teclado. Já no nível médio o aluno deve abrir o console de programação e utilizar comandos escritos para movimentar o robô, sendo `andarDireita()`, `andarEsquerda()`, `andarAcima()` e `andarAbaixo()`. No nível difícil é adicionado os comandos de enquanto para efetuar a programação com laços de repetição.

Após fazer a resolução, o aluno deve apertar o botão executar para validar os comandos e pressionar a barra de espaço para executar as instruções. Ao se movimentar, o robô deixa um rastro de óleo, o qual permite visualizar todo o caminho que ele percorreu no mapa do Furbot. Ao coletar todos os tesouros ele irá considerar o exercício como resolvido

e irá exibir em uma tela os comandos executados e a contagens dos pontos. Caso a quantidade de comandos executados seja a mesma que a identificada nas respostas do exercício, o aluno recebe um bônus na pontuação. Os itens coletados também geram pontos adicionais, já esbarrar em aliens ou na parede faz com que o aluno perca pontos.

O Furbot também possui uma interface visual para editar os mundos, visto na Figura 3, denominada Gerador de Mundo. O gerador permite desenhar os mapas com os obstáculos, itens coletáveis e inimigos, além de escrever o enunciado do exercício. Contudo, as fases criadas não estão integradas de forma dinâmica com a plataforma, para criar novas fases era necessário adicionar de forma manual através do ambiente de desenvolvimento. Dessa forma é necessário recompilar o código para que o mundo seja disponibilizado no jogo. Ainda, não é possível adicionar respostas para os exercícios através do Gerador de Mundo. Logo, para usar o recurso que efetua bonificação aos alunos pela resposta informada é necessário incluir de forma manual todos os nós `respostas` com seus respectivos valores no arquivo XML gerado ao salvar o mundo. Um exemplo deste arquivo pode ser visualizado no Anexo B Quadro 21.

Figura 3 - Ambiente de Edição de fase Furbot



Fonte: elaborado pela autora.

A tela de edição possui um botão para inserir a tabela com o tamanho desejável para o exercício, junto do enunciado dele. Essa tabela é necessária para colocar os itens no mapa. Para adicionar itens é preciso clicar na imagem correspondente que se encontrada ao lado direito da tela de edição e clicar no campo desejável no mapa. Ainda, todo mapa deve possuir apenas um único componente Furbot.

3 FERRAMENTA DESENVOLVIDA

Esta seção apresenta o desenvolvimento da ampliação da plataforma Furbot. Os Requisitos Funcionais (RF) da ferramenta responsável pela geração de gabaritos estão apresentados no Quadro 5. Os RF da ferramenta responsável pela a autoavaliação estão no Quadro 6. Já os Requisitos Não Funcionais (RNF) estão apresentados no Quadro 7. Na subseção 3.1 é apresentado o desenvolvido da Ferramenta de Geração de Gabarito, sendo está a ferramenta que cria os gabaritos a partir dos mapas do gerador do mundo. Na subseção 3.2 é apresentada a Ferramenta de Autoavaliação que retorna a comparação do gabarito com a resolução dos alunos, de modo que eles possam comparar seu código com a melhor resposta possível.

Quadro 5 - Requisitos Funcionais da Ferramenta de Geração de Gabaritos

Requisitos Funcionais da Ferramenta de Geração de Gabaritos
RF01: identificar e classificar itens do mapa do gerador do mundo (Objetivo, Objetivo Opcional Parede)
RF02: encontrar o menor caminho entre o robô e entre os objetivos obrigatórios do mapa
RF03: encontrar caminhos alternativos baseados nos objetivos opcionais
RF04: gerar gabarito com os comandos sequenciais que o robô deve fazer para coletar os objetivos do mapa
RF05: gravar o registro dos gabaritos baseados nos objetivos opcionais no arquivo do exercício

Fonte: elaborado pela autora.

Quadro 6 - Requisitos Funcionais da Ferramenta de Autoavaliação

Requisitos Funcionais da Ferramenta de Autoavaliação
RF01: fazer leitura de registros de gabaritos no arquivo do mundo
RF02: fazer leitura dos objetivos do mapa

RF03: permitir visualização dos códigos escritos pelo o jogador e o gerado pela aplicação
RF04: permitir visualização dos totalizadores de itens coletados na conclusão do jogo
RF05: efetuar comparação do tamanho do código descrito pelo o jogador e o gerado pela a aplicação

Fonte: elaborado pela autora.

Quadro 7 - Requisitos Não-Funcionais da Aplicação

Requisitos Não-Funcionais da Aplicação
RNF01: ser implementado na linguagem Java
RNF02: ser implementado através do ambiente de desenvolvimento Eclipse
RNF03: utilizar o algoritmo de Dijkstra para encontrar a melhor solução para o exercício
RNF04: ser implementado sobre a última versão do *framework* Furbot
RNF05: utilizar a biblioteca Commons-digester-1.8 para leitura do XML dos enunciados e resposta

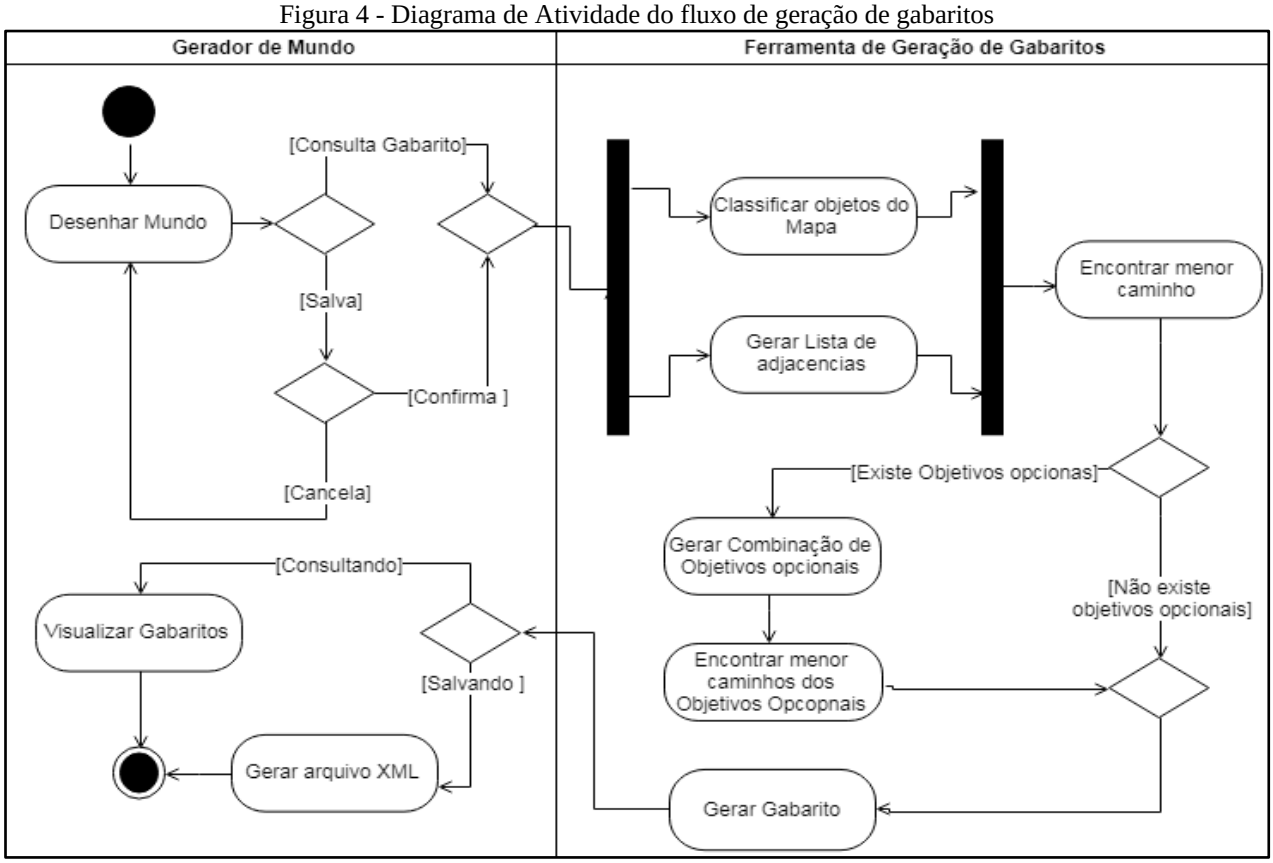
Fonte: elaborado pela autora.

A ampliação da plataforma Furbot, demostrada nesse artigo pode ser dividida em duas partes sendo: a primeira a geração automatizada de gabarito, utilizando do algoritmo de Dijkstra para encontrar o menor caminho, o qual é traduzido para as instruções conhecidas pelo o robô; e a segunda que é a ferramenta de autoavaliação que define o caminho a ser percorrido e traduz em comandos entendíveis para o robô. Essa segunda é responsável por comparar a resolução do aluno com o gabarito e trazer um retorno informativo. Ambas ampliações foram adicionadas à plataforma original do Furbot, o qual é utilizado para o ensino de programação na Educação Básica.

3.1 FERRAMENTA DE GERAÇÃO DE GABARITOS

A Ferramenta de Geração de Gabaritos é responsável por ler o mapa desenhado através da interface visual do Gerador de Mundo da plataforma e gerar um objeto de classe Gabarito, seu diagrama de classe está disponível no Apêndice A Figura 9. O fluxo de geração dos gabaritos é ilustrado na Figura 4. Inicialmente o usuário deve desenhar o mundo do exercício, distribuindo como desejar os objetos no mapa. Duas opções iniciam o processo de geração de gabarito sendo: a opção de consultar o gabarito; e a opção de salvar o mundo. Em seguida, os componentes do mapa atual do gerador de mundo são classificados. Em paralelo é gerada a lista de adjacências. Se o processo for bem-sucedido, o fluxo passa para a etapa para encontrar o menor caminho. Caso existam objetivos opcionais, é gerada uma sequência de combinações de objetivos opcionais para gerar gabaritos opcionais.

Figura 4 - Diagrama de Atividade do fluxo de geração de gabaritos



Fonte: elaborado pela autora.

O processo inicial da geração de gabarito é chamado pela classe `GeradorMundo` a qual estende da classe `JFrame`, abrindo a tela da Figura 5. O usuário deve criar uma tabela para iniciar a criação do mundo. Existe duas formas de criar uma tabela. Primeiramente pode-se inserir uma tabela nova informando as dimensões do mapa. Na versão do Furbot

apresentado neste artigo foi incluída a função para carregar um mapa a partir de um exercício criado anteriormente pela ferramenta de geração de mundo, dessa forma será carregado a tabela do Furbot com todos seus componentes.

Figura 5 - Tela de geração de mundo da versão ampliada da plataforma Furbot

Fonte: elaborado pela autora.

Na tabela, os componentes podem ser adicionados através dos itens `2`, `3`, `4`, `5` e `6` e removidos através da ferramenta de limpar item `10`. Os componentes que podem ser acionados na tabela do mundo são: Alien (`item 2`); Anjo (`item 3`); Tesouro (`item 4`); Número (`item 5`) e Parede (`item 6`). Os botões de abrir (`item 8`) e o botão de consulta gabarito (`item 9`) foram incluídos para a versão ampliada da plataforma apresentada neste artigo. Os números apontados pelo `item 11` são componentes do tipo Número marcados como objetivos do mapa. Na versão ampliada da plataforma Furbot, os componentes de Números podem ser marcados como objetivos. Para isso é necessário clicar sobre o número com o botão esquerdo do mouse, o que os deixara de cor verde indicando que serão considerados como Objetivos.

Ao adicionar componentes do Furbot na tabela, eles são carregados em uma lista de objetos do tipo `ContemItem`, guardando informações de posição na tabela e do tipo do componente. Quando o usuário confirma a gravação do mapa ou quando efetua uma consulta no gabarito, dando início ao fluxo de atividade da Ferramenta de Geração de Gabarito. Um diagrama de classe da ferramenta está disponível no Apêndice B Figura 9. Para isso, é criada uma instância da classe `GabaritoGerenciador` a qual chama o método `getListaVertice()` que irá executar as duas atividades necessárias antes de efetuar a busca do menor caminho. Entre essas atividades está a de classificar os objetos do mapa, separando os componentes em:

a) obstáculos: são componentes que não fazem parte do caminho ou que não devem ser coletados no mundo. São considerados como obstáculos as paredes, os alien e os números não marcados como objetivo;
b) objetivos: são componente que devem ser coletados para poder concluir o exercício, sendo que são obrigatórios. São considerados objetivos os componentes tesouros e números marcados como objetivos;
c) objetivos opcionais: componentes que podem ser coletados para dar pontos extras. É considerado como objetivo opcional o componente anjo;
d) início: ponto de partida do exercício. É considerado como início o componente Furbot.

A partir desses elementos é criada então uma matriz análoga a tabela no qual os componentes da lista `ContemItem` são redistribuídos com base em sua posição e uma matriz auxiliar de vértices. Essas matrizes são percorridas, os objetos classificados como obstáculos são ignorados e para cada outro campo da matriz de `ContemItem`, incluso os campos vazios, são criadas instâncias de vértices (`Vertice.java`) e atribuídos a posição equivalente na matriz auxiliar. Para cada vértice `v` criado na matriz auxiliar `M` será checado se existem vértices adjacentes, sendo a posição do vértice `v` M_{ij}. Checa-se a existência de vértices em $M_{i-1\,j}$ e em $M_{i\,j-1}$, caso existam serão atribuídos como adjacentes a `v` e `v` será atribuído a eles como adjacente, fazendo assim uma atribuição de adjacência simétrica. Todos os vértices são

adicionados na `listaVertices`. Os vértices que contém objetivos são adicionados na `listaObjetivos` e os que contém objetivos opcionais na `listaObjetivosOpcionais`.

Seguindo o fluxo de atividade da Figura 4, em seguida tem-se a atividade de encontrar o menor caminho. Isto é feito pelo método `gerarGabarito()`, disponível no Apêndice B no Quadro 15, correspondente a classe `GabaritoGerenciador`. Esse é o método responsável por retornar uma instância de `Gabarito`. Ele passa a lista de objetivos para método b`uscaMenorCaminho()`. O método `buscamenorCaminho()` é apresentado no Quadro 8.

Quadro 8 - Método buscamenorCaminho()

```
1  private ArrayList<Vertice> buscaMenorCaminho(ArrayList<Vertice> listaObjetivos)
2  throws GabaritoExeception {
3  // Lista de objetivos e origem devem ser passados por parametro futuramente
4          tamanhoMenorcaminho = Integer.MAX_VALUE;
5          Vertice origem = inicio;
6          ArrayList<Vertice> listaObjetivosNaoVisitados = (ArrayList<Vertice>) listaObjetivos.clone();
7          ArrayList<Vertice> menorCaminho = buscaMenorCaminhoObjetivos(origem,
8          listaObjetivosNaoVisitados,
9          new ArrayList<Vertice>(), new ArrayList<ArrayList<Vertice>>(), qtdObjObrigatorio,
10         qtdObjOpcional);
11 return menorCaminho;
12         }
```

Fonte: elaborado pela autora.

Como apresentado no Quadro 8 `linha 6`, a `litaObjetivosNaoVisitados` é duplicada usando a `listaObjetivos` recebido do parâmetro. São inicializados os parâmetros do método recursivo busca de menor caminho nas `linhas 6` até `7`. Então é iniciada a busca da combinação de caminhos que possua o menor tamanho para alcançar todos os objetivos listados, sendo esse o caminho que passa por uma menor quantidade de vértices para alcançar todos os objetivos do mapa. Para isso, passa-se por todos os objetivos da lista, os quais são usados de destino e então é calculado o menor caminho a partir da `origem`. Utiliza-se o método `executa (origem, destino, listaVertices)()`(Quadro 10 `linha 7`) da classe `Dijkstra`. Os passos da execução do algoritmo de Dijkstra em cima de um mapa do Furbot é ilustrado no Apêndice D Figura 13. A implementação do algoritmo de Djkstra permite O caminho retornado pelo algoritmo é representado através da variável `Arraylist<Vertice> caminhoNovo`. Nesse processo foi incluída uma otimização, em que todo menor caminho entre uma origem e destino é incluso na variável `HashMap mapaCaminhos`. Toda vez que é necessário obter um `caminhoNovo` é checado se já existe um caminho para essa origem e destino armazenado, caso exista esse caminho, ele se torna o `caminhoNovo`. O caminho será atualizado com `caminhoDestino` e será formado pelo vértice de `inicio`, o `caminhoAtual` e `caminhoNovo` sem o vértice de `origem`. Desta forma, o valor do `caminhoNovo` será substituído. Se o `caminhoDestino` ainda for menor que todas as combinações de caminhos já obtidas e ainda existir objetivos para serem percorridos na lista, é efetuada a chamada recursiva do método do Quadro 9.

Quadro 9 - Trecho do método `buscaMenorCaminhosObjetivos()` e chamada recursiva

```
1  if (ListaObjetibosNaoVisitadosR.size() > 0) {// Efetua a Busca recursiva
2          buscaMenorCaminhoObjetivos(destino, listaObjetivosNaoVisitadosR, caminhoNovo,
3          listaCaminhos,qtdObj, qtdObjOpcional);
4  } else {
5          listaCaminhos.add(caminhoDestino);
6          tamanhoMenorcaminho = caminhoDestino.size(); //
7          caminhoPercorridos++;
8  }
```

Fonte: elaborado pela autora.

O destino será a nova origem. Dessa forma, será removido o destino atual da `listaObjetivosNaoVisitados` e o `caminhoNovo` será o `caminhoAtual`. Caso tenha terminado de percorrer a lista, o `caminhoDestino` será adicionado na `listaCaminhos` e seu tamanhos será atribuído como o tamanho da lista de menor caminho. Após retornar para a chamada original de `buscaMenorCaminhoObjetivos()`, o caminho com o menor tamanho é retornado para `BuscaMenorCaminho()`(Quadro 8), que por sua vez retorna para o método `gerarGabarito()` disponível no Apêndice B Quadro 15. Com a lista de objetivos e o menor caminho encontrado é criada a instância de `Gabarito`.

Seguindo o fluxo (Figura 4), caso exista objetivos opcionais é efetuada a geração de combinações de objetivos opcionais. Através do método `getVariacoesOpcionais`() são criadas todas as combinações possíveis de conjuntos formados pelos objetivos opcionais. A quantidade de variações geradas será de $\sum_{i=1}^{n} C_{n,i} = \frac{n!}{i!*(n-i)!}$ combinações, sendo `n` a quantidade de gabaritos existentes no mapa. Para cada uma dessa combinações é efetuado o processo de

`buscaMenorCaminho()` no qual a combinação de objetivos opcionais é adicionada à lista de objetivos a serem explorados pelo o algoritmo. Quando existem objetivos opcionais no mapa é necessário efetuar validações, conforme o trecho apresentado no Quadro 10.

Quadro 10 - Trecho do método `buscaMenorCaminhosObjetivos()` e validações de Objetivos Opcionais

```
1  if (validaObjetivosOpcionais(qtdObj, qtdObjOpcional)) {
2          String chave = origem.toString() + destino.toString();
3          ArrayList<Vertice> caminhoNovo = null;
4          if (mapaCaminhos.containsKey(chave)) {    /// otimização, busca tragetórias já conhecidas
4                  caminhoNovo = (ArrayList<Vertice>) mapaCaminhos.get(chave).clone();
5          } else {
6                  try {
7                          caminhoNovo = Dijkstra.executa(origem, destino, AuxlistaVertices);
8                  }catch (GabaritoExeception ga ) {
9                          ignoraCaminho = true;
10                         try {
11                                 caminhoNovo = Dijkstra.executa(origem, destino, listaVertices);
12                         }catch(GabaritoExeception ga2) {
13                                 throw ga2;
14                         }
15                 }
16                 if(!ignoraCaminho) {
17                         mapaCaminhos.put(chave, (ArrayList<Vertice>) caminhoNovo.clone());
18                 }
19 }
```

Fonte: elaborado pela autora.

Durante a geração de gabaritos opcionais deve ser garantido que todos os objetivos opcionais serão coletados antes de coletar o último objetivo obrigatório. Isso é necessário porque o exercício será finalizado quando o Furbot atingir o último objetivo obrigatório do mapa. Para isso, o método `validaObjetivosOpcionais()` checa se a quantidade de objetivos opcionais é maior que zero, caso o destino seja o último objetivo restante, caso seja o caminho é então considerado inválido. Ocorre que essa validação é feita apenas nos `destinos` do caminho para garantir que em nenhum momento o vértice cruze com um objetivo obrigatório no meio do caminho. Se isso acontecesse, o objetivo não seria contabilizado para a validação. Logo, é necessário que caso o `destino` que se deseja alcançar seja um objetivo opcional, a lista de vértices enviada para o método `executa()` do `Dijkstra` é trocada por uma cópia a `listaVerticesParaObjetivosOpcionais`. A lista de vértices utilizada no método `executa()` do `Dijkstra` deve considerar todos objetivos obrigatório como obstáculos. Então todos os vértices contidos na `listaObjetivos` são removidos da cópia da lista de vértices original, o que fará com que os caminhos gerados não cruzem com esses objetivos. Caso não seja possível encontrar um caminho durante a busca do objetivo opcional, será checado se o caminho não está apenas bloqueado por um objetivo obrigatório executando a busca com a lista de vértices originais. Nesse caso, se for, é indicado para ignorar esse caminho. Caso não seja esse motivo, é lançado um erro para o usuário, pois quando não é possível alcançar um objetivo o mapa é considerado inválido, a lista de vértices desse mapa representa uma estrutura de grafo chamado de desconexo o qual não pode ter todos seus vértices alcançados. O resto do processo é executado de forma igual a geração de gabarito para a lista de objetivos. Tenta-se então adicionar os gabaritos opcionais criados na lista de gabaritos opcionais da instância principal de `Gabarito`, verifica-se a existência de gabaritos na lista que tenham a mesma quantidade de objetivos opcionais, caso não exista nenhum com a mesma quantidade é adicionado à lista. Caso exista um gabarito com a mesma quantidade de objetivos opcionais, o que possui menor caminho ficará na lista.

Se o fluxo for acionado ao consultar o gabarito será exibida a `TelaGabaritoConsulta`. Caso a saída desejada seja a gerar o arquivo XML de exercício, a chamada do método `criaXML()` é realizada. Esse método e a classe `Tab` foram adaptadas para armazenar as novas informações necessárias para a utilização do gabarito. Foi adicionada uma estrutura de nós para se armazenar os dados do gabarito (Quadro 11). A classe Gabarito chama o método `getObjetivosOpcional()` que retorna uma `String` com os vértices dos objetivos opcionais estruturadas em `([linha], [coluna])` separando cada um deles com um ponto e vírgula. O Método `getComandos()`, disponível no Apêndice B lê a sequência de vértices do caminho do gabarito e identifica as direções que são necessárias seguir. Em seguida chama-se o comando correspondente a essa direção e os armazena em uma `String` separados por ponto e vírgula.

Quadro 11 - Estrutura XML nó gabarito

```
</gabarito>
<gabarito>
	<objetivos></objetivos>
	<direcao>
		</direcao>
</gabarito>
```

Fonte: elaborado pela autora.

Dentro do XML do exercício há um nó de gabarito para a instância de gabarito gerada e um para cada um dos `gabaritosOpcionais` gerados. O nó `objetivos` armazena a posição dos objetivos opcionais utilizados para gerar os objetivos formatados. O nó de direção armazena a sequência de comandos do gabarito. Foram também necessários incluir novos componentes nó de mundo, `qtdNumerosqtd` para armazenar a quantidades de números do mundo e `NumerosObrigatorios` para armazenar a quantidade de números que são classificados como objetivos. No nó de número foi incluído o objetivo para atribuir um valor de tipo boolean indicando se o número apresentado é um objetivo do exercício ou não (Quadro 21).

Para aumentar o dinamismo da versão atualizada da plataforma Furbot, foi efetuada uma adaptação no código para que permita que os jogos criados pelo gerador de mundo sejam carregados em tempo de execução. Anteriormente seria necessário incluir o exercício na Plataforma e criar um executável. Com essa adaptação, os exercícios criados possuem acesso disponível através de um botão na tela na qual é feita a seleção do nível de dificuldade do exercício.

3.2 FERRAMENTA DE AUTO AVALIAÇÃO

A Ferramenta de Autoavaliação é responsável por retornar à comparação da resolução no momento no qual o jogo finalizar, ou seja, após alcançar todos os objetivos. O diagrama de classe desse processo está disponível no Apêndice A Figura 10. Para incluir a autoavaliação nos exercícios, primeiramente foi necessário adaptar o processo responsável por carregar os exercícios no jogo. No momento que o aluno escolher um exercício para executar, é chamada a classe responsável por ler o arquivo XML. Utiliza-se a biblioteca Commons-digester-1.8 para criar uma instância de exercício (`Exercicio.java`). O método responsável por criar a instância do gabarito está disponível no Quadro 12. Ele pertence a classe `ExercicioFactory`, que recebeu os ajustes para poder ler os novos nós que foram incluídos na estrutura do arquivo XML.

Quadro 12 - Método `addRuleGabarito()`

```
1   private static void addRuleGabarito(Digester d, String pattern) {
2           d.addObjectCreate(pattern, Gabarito.class);
3           d.addBeanPropertySetter((new
4           StringBuilder(String.valueOf(pattern))).append("/objetivoOpcional").toString());
4           d.addBeanPropertySetter((new
5           StringBuilder(String.valueOf(pattern))).append("/comandos").toString());
6           d.addCallMethod((new StringBuilder(String.valueOf(pattern))).append("/objetivos").toString(),
7           "setObjetivoOpcional", 1);
8           d.addCallParam((new StringBuilder(String.valueOf(pattern))).append("/objetivos").toString(), 0);
9           d.addCallMethod((new StringBuilder(String.valueOf(pattern))).append("/direcao").toString(),
10          "setComandos", 1);
11          d.addCallParam((new StringBuilder(String.valueOf(pattern))).append("/direcao").toString(),
12          0);//setObjetivo
13  }
```

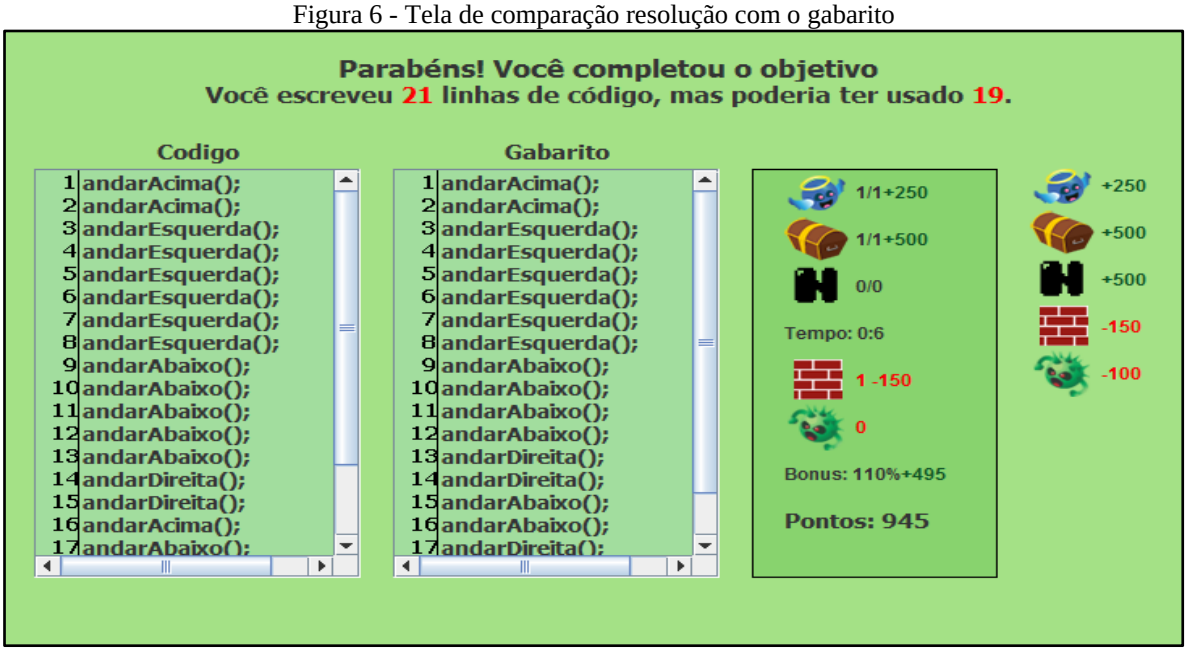
Fonte: elaborado pela autora.

É criada uma instância de gabarito para cada nó de nome gabarito contido no arquivo XML através da chamada do método `addObjectCreate()` na `linha 1` do Quadro 12. A biblioteca que identifica o nó de nome gabarito na *String* passada pelo parâmetro `pattern`. Os métodos da classe `Gabarito` chamados durante a montagem `setObjetivoOpcionais` (Quadro 12 `linha 6` e `7`) e `setComandos`() (Quadro 12 `linha 9` e `10`), atribuem os valores em formato de texto para as instâncias de gabaritos criadas. O método `setObjetivosOpcionais` também recria a lista com os objetivos opcionais do gabarito. Foi incluído na classe de exercício os atributos necessários para armazenar as informações relacionadas a automatização do processo de avaliação de exercício. Esses atributos incluídos foram referentes a quantidade de números de objetivos do mapa e a lista contendo todos os gabaritos dos exercícios. Os gabaritos criados pelo método do Quadro 12 são adicionados à lista de gabaritos da instância de exercício.

Durante a execução do exercício no Furbot são guardadas as informações referentes ao desempenho do aluno, como colisões com obstáculos e itens coletados. O exercício será considerado concluído quando todos os componentes classificados como objetivos do mapa forem recolhidos. A versão apresentada nesse artigo permite definir números como

objetivos do mapa. O processo anteriormente só considerava os componentes tesouros. Após o processo será aberta a tela `TelaGabaritoCompara` (Figura 6) mostrando a avaliação da resolução feita pelo aluno.

Figura 6 - Tela de comparação resolução com o gabarito



Fonte: elaborada pela autora.

A tela de comparação possui uma mensagem de parabenizarão por concluir o exercício e uma mensagem descrevendo a comparação com os comandos contidas no gabarito. Além disso, tem-se um `scrollPanel` a esquerda com os comandos gerado pelo o aluno e ao lado dele um `scrollPanel` com os comandos do gabarito. Ainda, no lado direito há um painel contendo todos os elementos de pontuação discriminando a quantidade respectiva de pontos recebido pelo o aluno e o valor individual de cada elemento.

A decisão de qual gabarito deve ser usado para mostrar na interface é feita de acordo com a quantidade de objetivos opcionais que o aluno coletou durante a resolução do exercício. Caso não existam objetivos opcionais será retornado o gabarito padrão. Existe dois métodos de escolha de gabarito: o primeiro `getEscolheGabarito()`, retorna o gabarito equivalente aos objetivos que foram coletados; já o método `getEscolheGabaritoPorQtd()` usa para a comparação um gabarito que possui a mesma quantidade de objetivos opcionais coletados. Esse método foi desenvolvido por conta da limitação estabelecida na quantidade de gabaritos geradas. A partir do gabarito escolhido é feita a análise responsável por trazer uma mensagem de comparação. A condição para a exibição das mensagens pode ser vista no Quadro 13.

Quadro 13 - Mensagens da comparação da tela de Autoavaliação

Condição	**Mensagem**
Não possui gabarito para o exercício	Você escreveu [quantidade de linhas escritas] de código.
Resolução igual ao gabarito	Você escreveu [quantidade de linhas escritas] de código. Igual ao Gabarito.
Resolução ótima diferente do gabarito	Você escreveu [quantidade de linhas escritas] de código. Equivalente ao Gabarito.
Resolução não ótima	Você escreveu [quantidade de linhas escritas] de código, mas poderia ter usado [quantidade de linhas do gabarito].

Fonte: elaborado pela autora.

4 RESULTADOS

Como demonstrado no Quadro 14 a comparação das características das aplicações em relação aos trabalhos correlatos e a plataforma anterior a ferramenta apresentada por esse artigo. Os trabalhos de Kopsh (2016) e Code.org (2018a) são ferramentas que foram construídas para fins educacionais. Já o trabalho do Ostrovsk (2009) é um jogo que tem como objetivo o entretenimento. Todas as ferramentas têm em comum a utilização de um ambiente para o exercício da lógica de programação.

Quadro 14 - Comparação de correlatos e plataforma anterior

Correlatos / Características	Kopsh (2016)	Code.org (2018a)	Ostrovsk (2009)	MATTOS et al. (2017)	Xavier (2019)
Ferramenta com objetivo educacional	Sim	Sim	Não	sim	sim
Análise automatizada de do exercício em comparação a uma solução ótima	Não	Sim	Não	Parcial	sim
Ambiente de criação de exercício personalizados	Sim	Não	Sim	Sim	sim
Automatização na geração de soluções	Não	Não	Não	Não	sim

Fonte: elaborada pela autora.

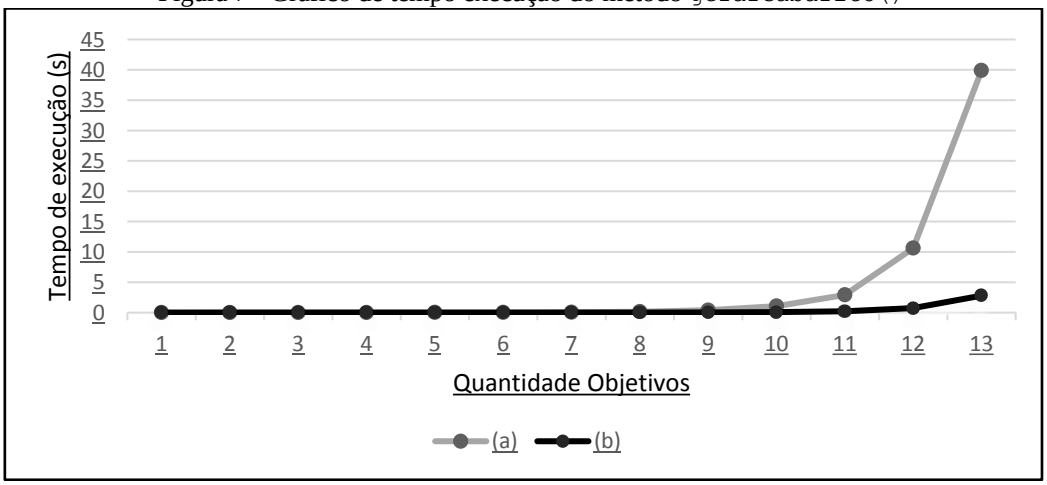
É possível observar que somente a ferramenta pela Code.org (2018a) possui a capacidade de avaliar a resolução elaborada pelo o usuário de forma automatizada, indicando a possibilidade de uma solução melhor e retornando para o usuário o código gerado pelo o usuário. Em relação a versão anterior da plataforma Furbot (MATTOS et al., 2017), ela efetua a comparação com um gabarito que tenha sido informado de forma manual pelo criador do exercício, de modo que essa informação é usada para acréscimo de bônus e não há garantia que seja uma solução ótima. A aplicação apresentada neste artigo também possui essa característica no processo de avaliação. Ela retorna também para o aluno o gabarito que foi efetuado para realizar esse exercício comparando com a quantidade de comandos que ele poderia ter utilizado e apresenta o percentual de bônus pelo acerto. A plataforma também define dentre os gabaritos gerados para o exercício, aquele que corresponde melhor a solução no caso de o aluno atingir objetivos que não são obrigatórios.

No caso das ferramentas de Ostrovsk (2009) e Kopsh (2016), bem como na versão anterior da plataforma Furbot (MATTOS et al., 2017) todas possuem um ambiente de criação de exercício. Ostrovsk (2009) obriga o usuário a fornecer uma solução válida para o exercício criado. A ferramenta de Mattos et al. (2017) permite que seja fornecida uma solução, mas a validade e a qualidade da solução não são verificadas. Com a automatização do processo de validação de solução demostrada nesse artigo, qualquer mapa criado pelo ambiente visual vai primeiramente validar a existência de uma solução possível, além de gerar sempre uma solução ótima para a coleta dos objetivos obrigatórios do mapa. Para casos com objetivos opcionais, há uma solução ótima para cada quantidade diferente de objetivos a serem coletados.

Foram efetuados testes unitários utilizando a ferramenta JUnit 4 na plataforma de desenvolvimento Eclipse. Os testes buscavam saber o tempo de demora que teria a execução responsável por retornar as instâncias de `Gabarito` em relação a certas características possíveis para o mapa.

Foi identificado o tempo e o aumento no tempo na geração de gabaritos com base na quantidade de objetivos existentes no mapa. No exemplo apresentado na Figura 7, o aumento do tempo de execução se torna notável ao uso, quando se entra na faixa de dez objetivos no mapa (serie (b)), já que até então a geração dos gabaritos demora tempo inferior a `u` segundo (Figura 7 serie (a)). Ao adicionar restrição para só executar o Dijkstra, caso o caminho que se deseja percorrer for conhecido, o mesmo é resgatável de uma lista salva em memória.

Figura 7 - Gráfico de tempo execução do método `gerarGabarito()`



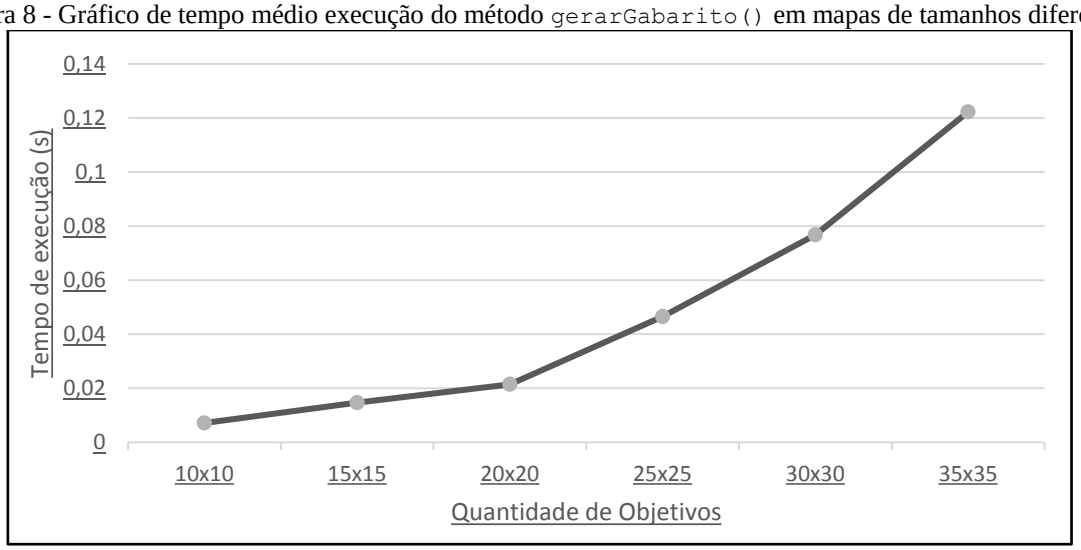
Fonte: elaborado pela autora.

O gráfico apresentado na Figura 7 tem seu valor baseado na média de execução de 100 gabaritos de um mapa com 10 colunas e com 10 linhas, sem nenhum obstáculo presente, com todos os objetivos a uma distância a partir de 10

vértices, sendo o ponto de início do mapa na coluna 1 linha 1. Exemplos desses modelos de testes são apresentados no Apêndice C, Figura 11 e Figura 12. A quantidade de objetivos do mapa varia de 1 até a quantidade não preenchidos do mapa do exercício.

Em outros testes feitos pela a mesma ferramenta JUnit 4, foi efetuado teste com mapas de dimensões diferentes. A partir disso foi identificado o aumento no tempo médio na geração de gabaritos com, com relação aos tamanhos dos mapas dos exercícios criados, como é possível observar na Figura 8.

Figura 8 - Gráfico de tempo médio execução do método `gerarGabarito()` em mapas de tamanhos diferentes



Fonte: elaborado pela autora

O gráfico apresentado na Figura 8 tem seu valor baseado na média de tempo de execução de exercícios de 1 à 14 objetivos e do tempo médio da execução de 100 gabarito de mapas, sem nenhum obstáculo presente, com todos os objetivos a uma distância a partir de 10 vértices, sendo o ponto de início do mapa na coluna 1 linha 1. Exemplos desses modelos podem ser visualizados no Apêndice C, Figura 11 e Figura 12.

A versão atualizada da plataforma foi levada para ser usada em um curso de programação com Furbot na Associação Criança em Primeiro Lugar (ACPL). Na turma estavam presentes 4 crianças, sendo duas de 9 anos de idade e duas de 11 anos. Três delas eram do sexo masculino e uma do feminino. Todos os alunos já tiveram aulas com a versão antiga da plataforma Furbot e todos já haviam programado nas 3 dificuldades disponíveis pela plataforma. Apenas uma dessas crianças já havia usado a ferramenta de geração de mundo da versão anterior da plataforma. Foi então disponibilizado para elas a versão descrita nesse artigo e instruído para que elas criassem seus próprios mapas e depois os resolvessem. Após as práticas foram feitas uma série de perguntas para as crianças.

Quando foi questionado para as crianças se haviam compreendidos a tela, apenas uma delas apontou que não havia ficado claro os pontos, bem como porque chegou nessa pontuação. Quando questionado se foi identificado algum erro, um deles apontou que havia batido na parede e essa batida não foi contabilizada. Ao ser questionado se eles se sentiram incentivados a fazer o menor caminho tendo o gabarito para comparar no final, todos concordaram que sim e um apontou que dessa forma "tem um pouco mais de pressão".

Os comentários das crianças quando questionadas sobre o que acharam da apresentação das repostas foram: "Achei legal, tem formas de ver outras oportunidades"; "Tem a correção, é bom para saber o que errou", "Achei legal, porque mostra as possibilidades que dá para fazer", "Achei legal, porque mostra o que você faz, o tanto de pontos e se fez errado".

Com base nos resultados obtidos pode-se observar que foi possível automatizar o processo de geração de gabarito. Com base nos testes de performance de geração de gabaritos, é possível observar eventualmente se terá uma quantidade limitada de objetivos que se deve usar em um exercício para que os gabaritos sejam gerados em tempo hábil. Referente as respostas obtidas com a turma de crianças, ter acesso ao resultado ótimo motivou as crianças a analisarem seus próprios resultados. Pode-se concluir também que ter a autocorreção do exercício e resultados de forma mais detalhadas foi recebido de forma positiva pelos alunos.

5 CONCLUSÕES

Este artigo apresentou o desenvolvimento de uma ferramenta para automatizar o processo de geração de gabaritos do gerador de mundo do Furbot, através da busca de menor caminho com o auxílio do algoritmo de Dijkstra. Além disso, apresentou uma ferramenta para automatizar a avaliação das resoluções desenvolvidas pelos os alunos, auxiliando o desenvolvimento do pensamento computacional. As ferramentas foram desenvolvidas a partir do ambiente Eclipse na linguagem Java, implementado sobre a última versão do *framework* Furbot, utilizando a biblioteca Commons-digester-1.8 para a leitura dos XML.

A versão ampliada da plataforma Furbot teve como objetivo automatizar o processo de geração dos gabaritos criados pela ferramenta de geração de mundo. Para a implementação foi utilizado o algoritmo de Dijkstra para encontrar o menor caminho entre a origem e os objetivos, os quais foram agrupados através de um processo de busca recursiva. Através disso foi possível encontrar o menor caminho ótimo dentre as combinações de caminhos possíveis. Os caminhos encontrados têm-se então as direções de movimentação identificada. Essas direções são traduzidas em comandos para o Furbot. Outro objetivo era facilitar o uso da plataforma Furbot em sala de aula para o auxílio do desenvolvimento do pensamento computacional, a versão atualizada da plataforma foi então levada para ser usada em um curso de programação com o Furbot, no qual as crianças puderam criar seus próprios exercícios. Com o processo de geração de gabarito e análise da resolução, elas puderam jogar esses exercícios e obter a correção de forma automatizada.

O processo de geração de gabarito permite gerar não só os objetivos principais, mas também considerar itens opcionais que possam ser coletados pelo o aluno durante a execução. Para permitir que o aluno receba a bonificação pela quantidade de itens opcionais coletados, e a bonificação por se conseguir o menor caminho para coletar essas quantidades de itens é gerado um gabarito ótimo para cada quantidade de objetivo opcional que o aluno pode coletar.

A partir dos resultados obtidos é possível observar que o tempo de demora de geração de gabaritos começa a aumentar de forma abrupta ao possuir certa quantidade de componentes em tela, conforme o tamanho de mapa utilizado, levando em conta que o tempo varia dependo da distribuição dos componentes e da existência de obstáculos. Esse tempo impossibilitaria a criação de certos mapas serem feitos em sala de aula em um tempo hábil.

Em comparação com seus correlatos, a versão atualizada da plataforma consegue juntar duas características que eram oferecidas em ferramentas diferentes, oferecendo o processo de correção automatizada, além de permitir um ambiente de elaboração de exercícios, garantindo que o gabarito oferecido pela a ferramenta será uma solução ótima.

Ressaltam-se as limitações identificadas no trabalho o acréscimo abrupto que ocorre no tempo de demora para gerar os gabaritos por conta da relação de tamanho do mapa e quantidade de objetivos, o que se torna necessário estipular limites para conseguir uma solução em tempo hábil. Além do processo que se agrava ao gerar gabaritos com objetivos opcionais, por conta da quantidade de combinações de objetivos que precisam ser gerados e testados.

Por fim, esse trabalho apresentou alterações na plataforma Furbot em busca de automatizar o processo de geração de gabaritos e de verificação de soluções de usuários. Esse processo pode auxiliar no ensino do pensamento computacional. Dentre as possibilidades para extensão, tem-se:

a) otimizar o tempo de geração de gabaritos;
b) efetuar o processo de comparação de gabarito com o código de entrada do consolo ao invés do código de resolução gerado na execução do Furbot;
c) implementar geração de gabaritos com instruções de *loops*;
d) apresenta solução de gabarito de forma visual no mapa.

REFERÊNCIAS

ARAÚJO, Luciana; MATTOS, Mauro. Furbot: plataforma para o ensino-aprendizado de pensamento computacional e programação. In: I CONCURSO LATINOAMERICANO DE TECNOLOGIAS EDUCACIONAIS PARA APRENDIZAGEM (EDUTECH), 1., 2018, São Paulo. 1p.

CODE.ORG. **Programação Anna e Elsa**, [s.l.], 2018a. Disponivel em: https://studio.code.org/s/frozen/stage/1/puzzle/1. Acesso em: 09 set. 2018.

CODE.ORG. **About Us**, [s.l.] 2018b. Disponivel em: https://code.org/about. Acesso em: 09 set. 2018.

CORMEM, Thomas H. et al. **Algoritmos**: teoria e prática. Tradução de Vandeberg D. de Souza. 2. ed. Rio de Janeiro: Elsevier, 2002.

KOCAY, William.; KREHER, Donald L. **Graphs, Algorithms, and Optimization Discrete Mathematics and Its Applications**. [s.l]: Taylor & Francis, 2005.

KOPSCH, Heloisa K. **FURBOT-WEB:** uma plataforma adaptativa para o ensino de programação. 2016. 119 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau. Blumenau.

MATTOS, Mauro M. et al. Aplicação da prática colaborativa no desenvolvimento de um jogo para o ensino da programação. In: SIMPÓSIO BRASILEIRO DE SISTEMAS COLABORATIVOS, 14, 2017. São Paulo. **Anais eletrônicos...** São Paulo: Não paginado.

OLIVEIRA, Milena L. S. de, et al.Ensino de lógica de programação no ensino fundamental utilizando o Scratch: um relato de experiência. In :CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (CSBC), 34, 2014, [Brasilia]. **Anais...** [Brasilia], [s.n],[2014].Não Paginado.

OSTROVSKY, Igor . **My hobby project: a social puzzle game developed in Silverlight** [s.l.]. 2009. Disponrvel em: http://igoro.com/archive/my-hobby-project-a-social-puzzle-game-developed-in-silverlight/. Acesso em: 17 set. 2018.

MENEZES JR, Rômulo C. et al. Aplicação de Algoritmos de Grafos para Gerar e Percorrer Jogos de Labirintos Aleatórios. In: CONGRESSO DE MATEMATICA APLICADA E COMPUTACIONAL, CMAC-Nordeste, 2012, João Pessoa. **Anais eletronicos...** Natal: Não paginado.

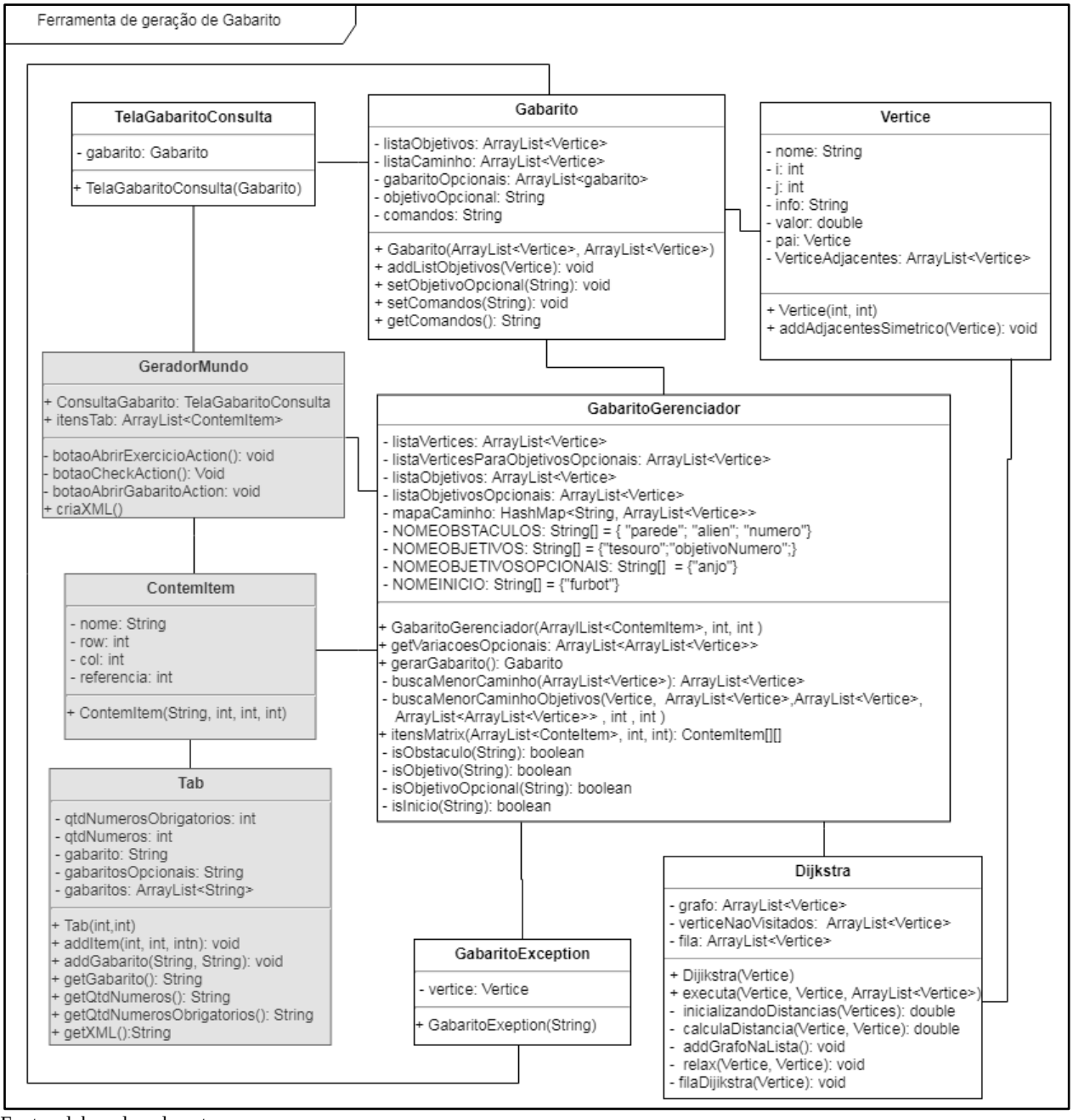
RAABE, André Luís Alice et al. Referenciais de Formação em Computação: Educação básica. Porto Alegre: Sbc, 2017. 9 p.

WING, Jeannette M.. Computational Thinking. Communications Of The Acm: Self managed systems. Nova York, p. 33-35. mar. 2006.

APÊNDICE A – DIAGRAMAS DE ESPECIFICAÇÃO

Na Figura 9 se encontra o diagrama de classe da ferramenta de geração de gabarito. Em cinza estão as classes do *framework* Furbot que foram utilizadas e modificada durante o desenvolvimento implementação demonstrada nesse artigo.

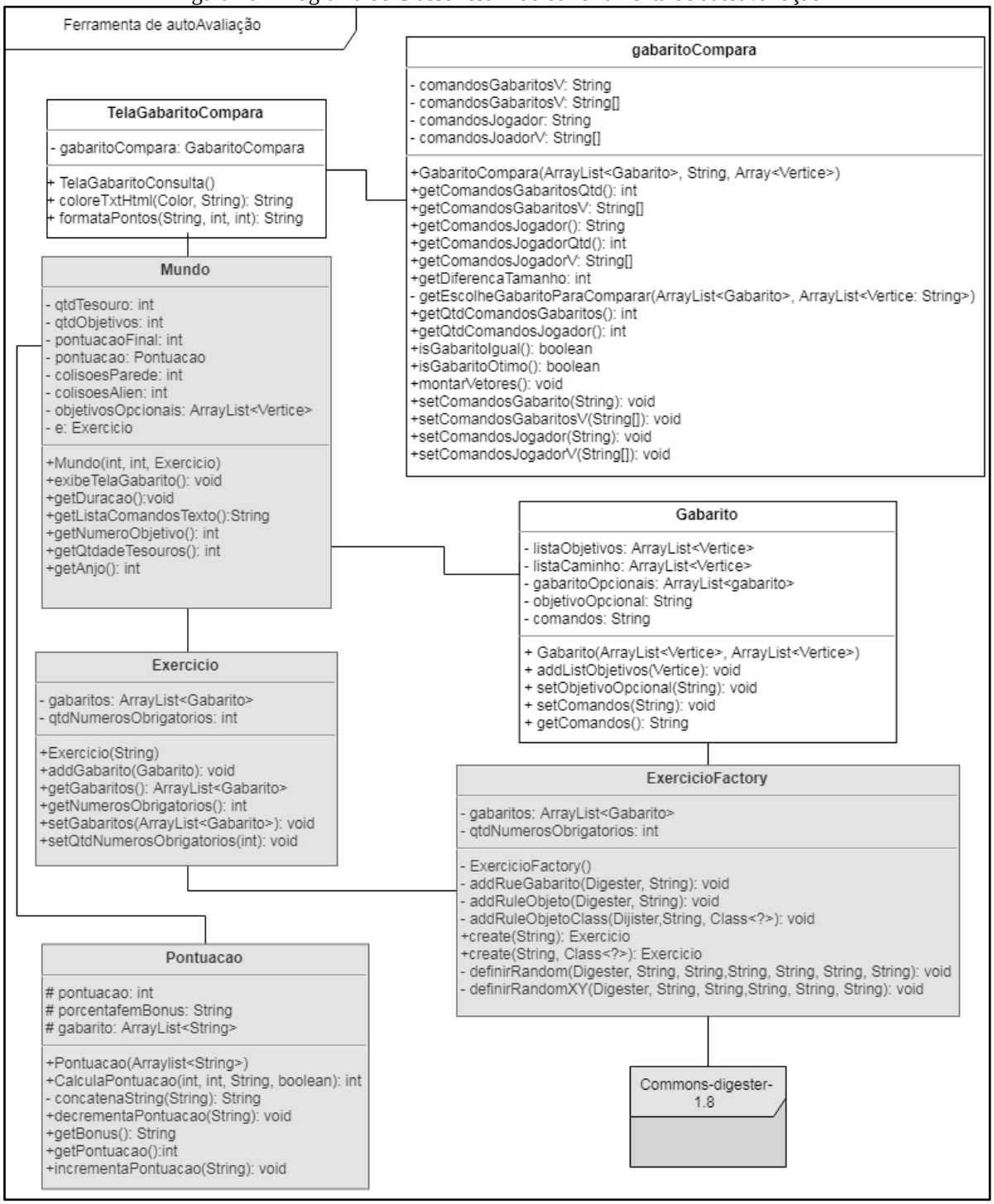
Figura 9 - Diagrama de classe da geração da ferramenta de geração de gabarito



Fonte: elaborado pela autora.

Na Figura 10 está o diagrama de classe da ferramenta de autoavaliação. Em cinza estão as classes do *framework* Furbot, que foram utilizadas e modificada durante o desenvolvimento implementação demonstrada nesse artigo.

Figura 10 - Diagrama de Classe resumido da ferramenta de autoavaliação



Fonte: elaborado pela autora.

APÊNDICE B – DETALHES DE IMPLEMENTAÇÃO

O Quadro 15 apresenta o método `gerarGabarito()` explicado no decorrer desse artigo. O método está comentado para melhor compreensão de seu código-fonte.

Quadro 15 – Método `gerarGabarito()`

```
1  public Gabarito gerarGabarito() throws GabaritoExeception {
2          // Buscar menor caminho para alcançar os vertices com componentes de tipo objetivo
3          ArrayList<Vertice> menorCaminho = BuscaMenorCaminho(listaObjetivos);
4          // Cria a instância de Gabarito e atribui-os a listadeobjetivos e o menor caminho
5          Gabarito gabarito = new Gabarito(listaObjetivos, menorCaminho);
6          // Inicia processo pra gerar gabaritos opcionais
7          if (gerarGabaritosOpcionais) {
8          // estipula quantidade de Objetivos opcionais que será gerado
9                 qtdObjObrigatorio = listaObjetivos.size();
9                 qtdAuxObjObrigatorio = qtdObjObrigatorio;
10         // Gera lista com combinação de objetivos equivalente a quantidade obj. opcionais
11                for (ArrayList<Vertice> opcionais : getVariacoesOpcionaisPorQtd()) {
12                       qtdObjOpcional = opcionais.size();
13                       qtdAuxObjOpcional = qtdObjOpcional;
14                       opcionais.addAll(listaObjetivos);
15                       ArrayList<Vertice> menorCaminhoOpcional =
16                                                  BuscaMenorCaminho(opcionais);
17                       Gabarito gabaritoOpcional = new Gabarito(opcionais,
18                                                         menorCaminhoOpcional);
19                       Gabarito.setListaObjetivosOpcionais(gabarito, gabaritoOpcional);
20                       gabarito.getGabaritosOpcionais().add(gabaritoOpcional);
21
                  }
          }
          return gabarito;
   }
```

Fonte: elaborado pela autora.

O Quadro 16 apresenta o código-fonte do método `buscaMenoeCaminhoObjetivos()` abordado no decorrer deste artigo.

Quadro 16 – Método `buscaMenoeCaminhoObjetivos()`

```
1  private ArrayList<Vertice> buscaMenorCaminhoObjetivos(Vertice origem, ArrayList<Vertice>
2  listaObjetivosNaoVisitados,ArrayList<Vertice> caminhoAtual,ArrayList<ArrayList<Vertice>>
3  listaCaminhos, int qtdAuxObjObrigatorio, int qtdqtdAuxObjOpcionalObjOpcional) throws
4  GabaritoExeception{
5          ArrayList<Vertice> caminhoDestino = null;
6          ArrayList<Vertice> menorCaminho = new ArrayList<>();
7          boolean ignoraCaminho = false;
8
9          for (Vertice destino : listaObjetivosNaoVisitados) {
10                caminhoDestino = new ArrayList<>();
11                int qtdObj = qtdAuxObjObrigatorio;// qtdAuxObjObrigatorio = qtdObj;
12                int qtdObjOpcional = qtdqtdAuxObjOpcionalObjOpcional; // qtdAuxObjOpcional
13                ArrayList<Vertice> AuxlistaVertices = listaVertices;
14                if (destino.getInfo().equals(TipoObjetivo.OBRIGATORIO.toString())) {
15                       qtdObj--;
16                } else {
17                       qtdObjOpcional--;
18                       AuxlistaVertices = (ArrayList<Vertice>)
19                       listaVerticesParaObjetivosOpcionais.clone();
20                }
21                if (validaObjetivosOpcionais(qtdObj, qtdObjOpcional)) {
22                       String chave = origem.toString() + destino.toString();
23                       ArrayList<Vertice> caminhoNovo = null;
24                       if (mapaCaminhos.containsKey(chave)) {
25                       caminhoNovo = (ArrayList<Vertice>)
26                       mapaCaminhos.get(chave).clone();
27                       } else {
28                              try {
29                                     caminhoNovo = Dijkstra.executa(origem,destino,
30                                                                AuxlistaVertices);
31                              }catch (GabaritoExeception ga ) {
32                                     ignoraCaminho = true;
33                                     try {
34                                            caminhoNovo = Dijkstra.executa(origem, destino,
35                                            listaVertices);
36                                     }catch(GabaritoExeception ga2) {
37                                            throw ga2;
```

```
38                                      }
39                                  }
40                                  if(!ignoraCaminho) {
41                                      mapaCaminhos.put(chave, (ArrayList<Vertice>)
42                                              caminhoNovo.clone());
43                                  }
44                              }
45                              if(!ignoraCaminho) {
46                                  ArrayList<Vertice> listaObjetivosNaoVisitadosR =
47                                  (ArrayList<Vertice>) listaObjetivosNaoVisitados.clone();
48                              listaObjetivosNaoVisitadosR.remove(destino);
49                              caminhoDestino.addAll((ArrayList<Vertice>) caminhoAtual.clone());
50                                  if (origem != inicio) {
51                                      caminhoNovo.remove(0);
52                                  }
53                                  caminhoDestino.addAll(caminhoNovo);
54                                  caminhoNovo = caminhoDestino;
55                                  if (caminhoDestino.size() < tamanhoMenorcaminho ||
56                                  ignoraCaminho) {
57                                      if (ListaObjetibosNaoVisitadosR.size() > 0) {
58                                          buscaMenorCaminhoObjetivos(destino,
59                                          listaObjetivosNaoVisitadosR, caminhoNovo,
60                                          listaCaminhos, qtdObj, qtdObjOpcional);
61                                      } else {
62                                          listaCaminhos.add(caminhoDestino);
63                                          tamanhoMenorcaminho caminhoDestino.size();
64                                          caminhoPercorridos++;
65                                      }
66                                  } else {
67                                      caminhoignorados++;
68                                  }
69                              }else{
70                                  caminhoignorados++;
71                              }
72                      } else {
73                              caminhoignorados++;
74                      }
75              }
76              if (inicio == origem) {
77                      for (ArrayList<Vertice> caminho : listaCaminhos) {
78                              if (caminho.size() == tamanhoMenorcaminho) {
79                                  menorCaminho = caminho;
80                                  return menorCaminho;
81                              }
82                      }
83              }
84              return menorCaminho;
85      }
```

Fonte: elaborado pela autora.

O Quadro 17 apresenta o código-fonte do método `getVariacoesOpcionais()`, o Quadro 18 aborda o método `getComandos()` e o Quadro 19 apresenta o método `getVariacoesOpcionais()`. Ambos foram abordados no decorrer deste artigo.

Quadro 17 - Método `getVariacoesOpcionais()`

```
1  public ArrayList<ArrayList<Vertice>> getVariacoesOpcionais(ArrayList<Vertice> listaOpcionais,
2  ArrayList<ArrayList<Vertice>> listaVarObjetivosOpcionais,  ArrayList<Vertice>
3  objetivosOpcionais ) {
4
5          if (listaVarObjetivosOpcionais == null) {
6                  listaVarObjetivosOpcionais = new ArrayList<>();
7          }
8          if (objetivosOpcionais == null) {
9                 objetivosOpcionais = new ArrayList<>();
10         }
11         ArrayList<Vertice> listaOpcionaisR = (ArrayList<Vertice>) listaOpcionais.clone();
12         for(Vertice objetivo:listaOpcionais) {
13                ArrayList<Vertice> objetivosOpcionaisNovo = new ArrayList<>();
14                objetivosOpcionaisNovo.addAll(objetivosOpcionais);
15                objetivosOpcionaisNovo.add(objetivo);
16                listaVarObjetivosOpcionais.add(objetivosOpcionaisNovo);
17                ArrayList<Vertice> objetivosOpcionaisR = (ArrayList<Vertice>)
18                objetivosOpcionaisNovo.clone();
19                listaOpcionaisR.remove(objetivo);
20                getVariacoesOpcionais(listaOpcionaisR,listaVarObjetivosOpcionais,
21         objetivosOpcionaisR);
22         }
23         return listaVarObjetivosOpcionais;
24 }
25
```

Fonte: elaborado pela autora.

Quadro 18 - Método `getComandos()`

```
1  public String getComandos() {
2          if( comandos == null) {
3          String comandos = "";
4          for (int i = 0; i < listaCaminho.size() - 1; i++) {
5                 Direcao direcao = Direcao.identificaDirecao(listaCaminho.get(i),
6                 listaCaminho.get(i + 1));
7                 comandos += direcao.getComando() + "\n";     }
8          this.comandos = comandos;
9          }
10         return comandos;
11         }
```

Fonte: elaborado pela autora.

Quadro 19 - Método `getVariacoesOpcionais()`

```
1  public ArrayList<ArrayList<Vertice>> getVariacoesOpcionais(ArrayList<Vertice> listaOpcionais,
2  ArrayList<ArrayList<Vertice>> listaVarObjetivosOpcionais,  ArrayList<Vertice>
3  objetivosOpcionais ) {
4          if (listaVarObjetivosOpcionais == null) {
5                  listaVarObjetivosOpcionais = new ArrayList<>();
6           }
7          if (objetivosOpcionais == null) {
8                 objetivosOpcionais = new ArrayList<>();
9          }
10         ArrayList<Vertice> listaOpcionaisR = (ArrayList<Vertice>) listaOpcionais.clone();
11         for(Vertice objetivo:listaOpcionais) {
12
13                ArrayList<Vertice> objetivosOpcionaisNovo = new ArrayList<>();
14                objetivosOpcionaisNovo.addAll(objetivosOpcionais);
15                objetivosOpcionaisNovo.add(objetivo);
16                listaVarObjetivosOpcionais.add(objetivosOpcionaisNovo);
17                ArrayList<Vertice> objetivosOpcionaisR = (ArrayList<Vertice>)
18                objetivosOpcionaisNovo.clone();
19                listaOpcionaisR.remove(objetivo);
20                getVariacoesOpcionais(listaOpcionaisR,listaVarObjetivosOpcionais,
21         objetivosOpcionaisR);
22         }
23         return listaVarObjetivosOpcionais;
24 }
25
```

Fonte: elaborado pela autora.

Quadro 20 - Arquivo XML de exercício Furbot

```
1  <?xml version="1.0"?>
2          <furbot>
3                 <fase>
4                        <nivel>Facil / 08</nivel>
5                 </fase>
```

```
6                   <teclado>
7                           <tecla>liberado</tecla>
8                   </teclado>
9                   <enunciado>Faca com que o Furbot recolha todos os tesouros no mapa.</enunciado>
10                  <mundo>
11                          <qtdadeLin>5</qtdadeLin>
12                          <qtdadeCol>5</qtdadeCol>
13                          <tamanhoCel>Grande</tamanhoCel>
14                          <qtdTesouros>1</qtdTesouros>
15                          <qtdAnjos>2</qtdAnjos>
16                          <qtdNumeros>1</qtdNumeros>
17                          <qtdNumerosObrigatorios>0</qtdNumerosObrigatorios>
18                  </mundo>
19                  <gabarito>
20                          <objetivos></objetivos>
21                          <direcao>
22                                  andarAbaixo();
23                                  andarAbaixo();
24                                  andarAbaixo();
25                                  andarDireita();
26                                  andarDireita();
27                                  andarAcima();
28                                  andarAcima();
29                                  </direcao>
30                  </gabarito>
31                  <gabarito>
32                          <objetivos>(2,4);</objetivos>
33                          <direcao>
34                                  andarAbaixo();
35                                  andarAbaixo();
36                                  andarAbaixo();
37                                  andarDireita();
38                                  andarDireita();
39                                  andarAcima();
40                                  andarDireita();
41                                  andarAcima();
42                                  andarDireita();
43                                  andarEsquerda();
44                                  andarEsquerda();
45                                  </direcao>
46                  </gabarito>
47                  <robo>
48                          <x>0</x>
49                          <y>1</y>
50                  </robo>
51                  <objeto class="br.furb.furbot.Parede">
52                          <x>1</x>
53                          <y>3</y>
54                  </objeto>
55                  <objeto class="br.furb.furbot.Parede">
56                          <x>1</x>
57                          <y>2</y>
58                  </objeto>
59                  <objeto class="br.furb.furbot.Anjo">
60                          <x>4</x>
61                          <y>0</y>
62                          <bloqueado>false</bloqueado>
63                  </objeto>
64                  <numero class ="br.furb.furbot.Tesouro">
65                          <x>2</x>
66                          <y>2</y>
67                          <valor>91</valor>
68                          <bloqueado>false</bloqueado>
69                  </numero>
70          </furbot>
```

Fonte: elaborado pela autora.

Quadro 21- Arquivo XML de exercício do Furbot

```
1   <?xml version="1.0"?>
2   <furbot>
3           <fase>
4                   <nivel>Facil / 01</nivel>
5           </fase>
6
```

```
7         <teclado>
8                 <tecla>liberado</tecla>
9         </teclado>
10        <enunciado>Ande para a direita até chegar no tesouro.
11                Temos neste mapa o
12                tesouro e a parede, no tesouro voce pode passar pelo
13                tesouro, mas não
14                pelas paredes.
15        </enunciado>
16        <mundo>
17                <qtdadeLin>12</qtdadeLin>
18                <qtdadeCol>12</qtdadeCol>
19                <tamanhoCel>Grande</tamanhoCel>
20                <qtdTesouros>1</qtdTesouros>
21                <respostas>
22                        <qntComandos>10</qntComandos>
23                        <lerQuantidade>true</lerQuantidade>
24                        <resposta>
25                                andarDireita();
26                                andarDireita();
27                                andarDireita();
28                                andarDireita();
29                                andarDireita();
30                                andarDireita();
31                                andarDireita();
32                                andarDireita();
33                                andarDireita();
34                                andarDireita();
35                        </resposta>
36                </respostas>
37        </mundo>
38        <objeto class="br.furb.furbot.Parede">
39                <x>5</x>
40                <y>11</y>
41        </objeto>
42        <objeto class="br.furb.furbot.Parede">
43                <x>4</x>
44                <y>10</y>
45        </objeto>
46        <objeto class="br.furb.furbot.Parede">
47                <x>3</x>
48                <y>11</y>
49        </objeto>    <objeto class="br.furb.furbot.Parede">
50                <x>1</x>
51                <y>11</y>
52        </objeto>
53        <numero class="br.furb.furbot.Tesouro">
54                <x>11</x>
55                <y>6</y>
56                <valor>0</valor>
57                <bloqueado>false</bloqueado>
58        </numero>
59        <robo>
60                <x>0</x>
61                <y>6</y>
62        </robo>
63  </furbot>
```

Fonte: elaborado pela autora.

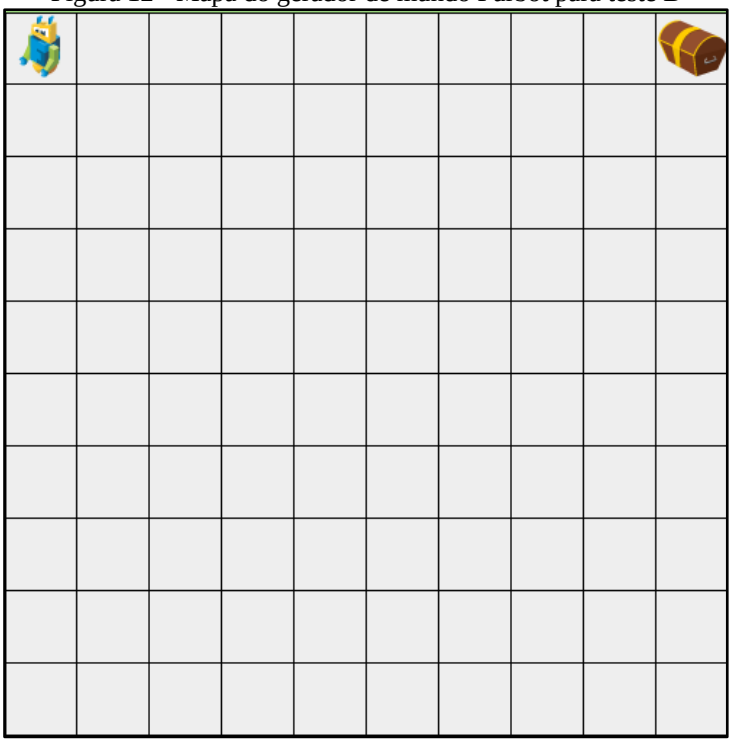
APÊNDICE C – MAPAS UTILIZADOS EM TESTES

A Figura 11e Figura 12 apresentam exemplos de mapas gerados a partir do gerador de mundos do Furbot. Os mesmos são mencionados ao longo do texto.

Figura 11 - Mapa do gerador de Mundo Furbot para teste A

Fonte: elaborada pela autora.

Figura 12 - Mapa do gerador de mundo Furbot para teste B

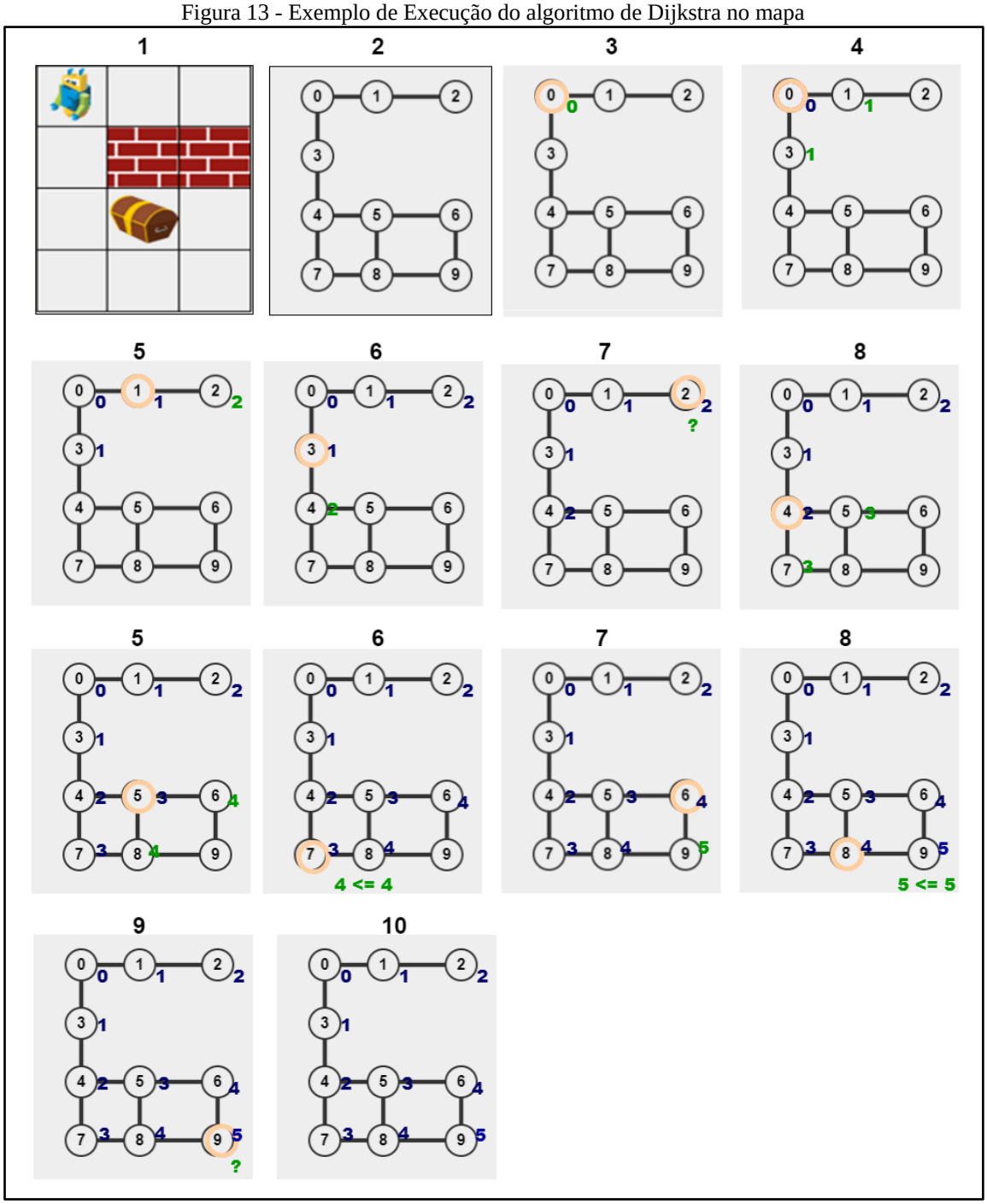


Fonte: elaborada pela autora.

APÊNDICED D – DEMONSTRAÇÃO DE PROCESSOS

Na Figura 13 é demostrada a execução do algoritmo de Dijkstra em cima do grafo gerado a partir de um mapa do Furbot (1). Os vértices são numerados de 1 a 9, sendo que todas as arestas do grafo possuem valor 1. O ponto de início equivale a posição inicial do Furbot. O ponto da execução é demarcado pelo círculo acima do vértice, os números ao lado do vértice representam o valor calculado para chegar até o vértice a partir do ponto de origem.

Figura 13 - Exemplo de Execução do algoritmo de Dijkstra no mapa



Fonte: elaborada pela autora.