SARALK - Um Aplicativo em JavaScript para Demonstração Lúdica de Algoritmos de Busca

André Victor Saraiva de Oliveira

Relatório de projeto apresentado à Escola Superior de Tecnologia e Gestão para obtenção do grau de mestre em Informática no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná

Orientadores:

Rui Pedro Lopes

Giuvane Conti

Bragança

2024 - 2025

Dedicatória

Dedico este trabalho às duas mulheres mais importantes da minha vida: minha avó, Catharina Saraiva (*in memoriam*) e a minha mãe, Maria Cristina Saraiva.

Agradecimentos

À medida que concluo este capítulo da minha vida acadêmica, sinto uma profunda gratidão por aqueles que foram essenciais nesta jornada.

Agradeço imensamente a Deus, por me guiar através dos desafios e por me fortalecer nos momentos de dúvida. Sua presença foi uma constante fonte de luz e esperança.

À minha mãe, minha principal torcedora, cujo apoio e fé inabaláveis em mim me incentivaram a superar cada desafio. Mãe, sua força e amor foram fundamentais nesta jornada.

Aos meus orientadores, cuja expertise e orientação foram vitais na execução deste projeto. Seu apoio, paciência e conhecimento não apenas me ajudaram a navegar pelos complexos problemas da informática, mas também a crescer como profissional.

Resumo

Os cursos de Ciência da Computação estão entre os mais procurados na atualidade, porém apresentam elevados índices de evasão, em grande parte devido às dificuldades enfrentadas pelos estudantes na compreensão de determinados conteúdos, especialmente dos algoritmos de busca. Diante desse cenário, torna-se necessário explorar novas metodologias de ensino. Uma das alternativas em ascensão é o uso de jogos digitais, embora ainda existam poucos estudos que avaliem sua eficácia em cursos da área de Computação, e os trabalhos disponíveis concentram-se, em sua maioria, apenas nos fundamentos de programação.

Este trabalho surge como uma proposta para mitigar tais dificuldades, por meio do desenvolvimento do Saralk, uma aplicação web que possibilita a visualização, em tempo real, da execução dos algoritmos de busca Busca em largura (BFS), Busca em profundidade (DFS), Busca Gulosa (GS) e Busca A-estrela (A*). Para isso, foram aplicadas técnicas de ludologia, o modelo ARCS de motivação e princípios de Experiência do Usuário (UX) e Interface do Usuário (UI) voltados a jogos educacionais. O jogo foi concebido de forma a permitir a compreensão teórica dos algoritmos enquanto o estudante os executa na prática, seja por meio de uma interface visual, voltada a iniciantes, ou por uma interface programática, destinada a usuários mais avançados.

O Saralk foi testado com estudantes do ensino superior e também com alunos do ensino básico, demonstrando ganhos significativos na compreensão dos algoritmos. Além de apoiar estudantes e profissionais interessados em algoritmos de busca, este trabalho contribui para o campo acadêmico ao aplicar, na prática, teorias sobre jogos educacionais voltados ao ensino de Computação, ampliando as discussões sobre sua adoção como metodologia ativa de aprendizagem.

Palavras-chave: Saralk, JavaScript, Grafos, Vue.js, Algoritmos de Busca.

Abstract

Computer Science programs are among the most sought after today; however, they also present high dropout rates, largely due to the difficulties students face in understanding certain subjects, especially search algorithms. In this context, it becomes necessary to explore new teaching methodologies. One emerging alternative is the use of digital games, although few studies have evaluated their effectiveness in Computer Science courses, and most of the available work focuses only on introductory programming.

This study proposes to mitigate such difficulties through the development of Saralk, a web application that enables real-time visualization of the execution of search algorithms such as BFS, DFS, GS, and A*. To achieve this, the project applied techniques from ludology, the ARCS motivational model, as well as UX and UI principles tailored to educational games. The game was designed to facilitate both the theoretical understanding of algorithms and their practical execution, either through a visual interface aimed at beginners or a programmatic interface intended for more advanced users.

Saralk was tested with higher education students and also with elementary-level students, demonstrating significant improvements in algorithm comprehension. Beyond supporting students and professionals interested in search algorithms, this work contributes to the academic field by putting into practice theories of educational games applied to Computer Science education, broadening the discussion on their adoption as an active learning methodology.

Keywords: Saralk, JavaScript, Graphs, Vue.js, Search Algorithms.

Conteúdo

Lista de Figuras

Acrônimos

A* Busca A-estrela. v, vii, xi, 2, 3, 4, 6, 10, 23, 24, 31, 33, 34, 35, 36, 37, 53, 56, 58, 71, 73

AI Inteligência Artificial. 23

BFS Busca em largura. v, vii, 1, 2, 3, 4, 6, 23, 24, 25, 27, 28, 30, 34, 37, 53, 56, 71

CSV Valores separados por vírgula. 13

DFS Busca em profundidade. v, vii, 1, 2, 3, 4, 6, 23, 24, 28, 30, 37, 53, 56, 71

GS Busca Gulosa. v, vii, 2, 3, 4, 6, 10, 23, 24, 31, 32, 33, 34, 37, 53, 56, 71, 73

IHC Interação Humano-Computador. 37, 65, 71, 74

IPB Instituto Politécnico de Bragança. xii, 69, 70

JSON Notação de Objeto JavaScript. 67

UI Interface do Usuário. v, vii

UTFPR Universidade Tecnológica Federal do Paraná. 4

UX Experiência do Usuário. v, vii, xi, 46, 47, 56, 71, 74

Capítulo 1

Introdução

Os algoritmos de busca representam uma base fundamental tanto para profissionais quanto para estudantes das áreas tecnológicas, destacando-se pela sua ampla aplicabilidade tanto em contextos teóricos quanto práticos. Contudo, a implementação dessas técnicas em código apresenta-se como um desafio significativo. Esse obstáculo, especialmente evidente no ambiente acadêmico, contribui para um elevado índice de reprovações e desistências, conforme indicado por uma série de estudos conduzidos em âmbito, tanto nacional quanto internacional, pelos autores [1]–[9].

As técnicas de busca na Ciência da Computação são métodos sistemáticos cruciais para encontrar e analisar dados em estruturas complexas ou conjuntos de dados extensos. Essas técnicas variam em sua abordagem e eficácia dependendo do problema em questão. Por exemplo, a DFS explora profundamente antes de retroceder, enquanto a BFS examina todos os vizinhos de um nó antes de avançar. A escolha de uma técnica adequada é vital, pois o impacto na eficiência pode ser grande. Ilustrando a importância da escolha de um método robusto para a solução de problemas de busca, Garey e Johnson [10] observam que, ao empregar, por exemplo, uma técnica de força bruta, que verifica todas as soluções possíveis, um computador que processa uma operação por microssegundo levaria pouco mais de um segundo para resolver um problema envolvendo 10 itens. No entanto, para um problema com 40 itens, por exemplo, esse tempo escalaria para mais de 3855

séculos (utilizando um algoritmo de complexidade 3^n) [10], destacando a necessidade crítica de algoritmos eficientes na resolução de problemas que possuam computacionalmente complexidade de tempo e espaço satisfatórias.

Este trabalho utiliza uma combinação de tecnologias web, JavaScript e Vue.js, proporcionando uma aplicação eficiente que não serve somente para ilustrar, mas também para permitir aos usuários o esclarecimento de algoritmos de busca e sua implementação em uma linguagem de programação, que neste caso, será JavaScript. Esta abordagem visa desmistificar a complexidade destes algoritmos, tornando-os mais acessíveis e compreensíveis para um público mais amplo, promovendo assim uma maior apreciação da tecnologia moderna e suas aplicações práticas.

Este projeto explora os fundamentos dos algoritmos de busca, unindo tanto teoria quanto a prática dessas técnicas, bem como realiza uma comparação direta entre elas, abrindo caminho para uma compreensão mais ampla destes conceitos na Ciência da Computação e da Engenharia de *Software*. Espera-se inspirar a próxima geração de desenvolvedores a abraçar os desafios da codificação de teorias complexas, fortalecendo assim o campo da computação com ferramentas inovadoras e soluções eficazes.

Portanto, o foco deste trabalho é construir uma ponte entre a teoria e a prática, para desmistificar e tornar mais acessíveis os algoritmos de DFS, BFS, GS e A*. Para tal, propõe-se desenvolver um *software* que possibilite a visualização lúdica, dinâmica e interativa do funcionamento interno desses algoritmos.

1.1 Objetivos

1.1.1 Geral

Desenvolver uma aplicação lúdica que execute e demonstre em tempo real o funcionamento dos algoritmos DFS, BFS, GS e A* de forma visual e programática.

1.1.2 Específicos

1. Desenvolver uma interface de usuário que estimule a motivação do usuário a utilizá-la;
2. Permitir que o usuário interaja pela interface do jogo ou de maneira programática;
3. Permitir ao usuário visualizar os cálculos executados em tempo real pelos algoritmos;
4. Suportar os algoritmos de busca DFS, BFS, GS e A*;
5. Esclarecer o funcionamento da teoria por trás do código desenvolvido;
6. Disponibilizar a aplicação desenvolvida para computadores e celulares.

1.2 Contribuições do trabalho

A contribuição efetiva deste trabalho está no desenvolvimento de uma aplicação que visa a elucidação da implementação de técnicas de busca e a possibilidade de visualizar seu funcionamento.

1. Comparações do desempenho entre diferentes algoritmos;
2. Disponibilização do jogo, acessível a estudantes e profissionais através do site `https://saralk.ipb.pt`;
3. Disponibilização do servidor para interação via código através do endpoint `wss://saralk.ipb.pt/api/websocket`.

1.3 Motivação

A compreensão profunda de algoritmos de busca é fundamental na Ciência da Computação, influenciando uma vasta gama de aplicações, desde sistemas de recomendação até a otimização de rotas de transporte. Contudo, a complexidade teórica e a dificuldade de

visualização prática desses algoritmos podem representar um obstáculo significativo para estudantes e profissionais da área [11]. Portanto, para avaliar a necessidade de uma ferramenta educacional mais eficaz, foi conduzida uma pesquisa com 52 estudantes de cursos relacionados à computação da Universidade Tecnológica Federal do Paraná (UTFPR). Os resultados mostrados na Figura 1.1 revelaram que a pesquisa abrangeu alunos de Licenciatura em Informática, Engenharia de Computação e Ciência da Computação.

Figura 1.1: Respostas à pergunta 1 do formulário

Fonte: Autoria própria (2024).

A pesquisa mostrou, conforme a Figura 1.2, que 96,2% dos alunos estudaram algoritmos de busca durante a faculdade, indicando a relevância do tema no currículo acadêmico. No entanto, quando questionados sobre a clareza da abordagem em sala de aula, as respostas foram mistas: 23,1% atribuíram uma nota 5, e 17,3% uma nota 6, em uma escala de 0 a 10, sugerindo que a explicação em sala de aula não é totalmente satisfatória para muitos alunos.

Além disso, um dado preocupante emergiu: 71,2% dos estudantes afirmaram que não conseguiriam aplicar facilmente algoritmos como DFS, BFS, GS e A* em desafios cotidianos de programação (Figura 1.3). Isso reforça a necessidade de métodos de ensino mais eficazes e ferramentas que facilitem a compreensão e aplicação prática desses algoritmos.

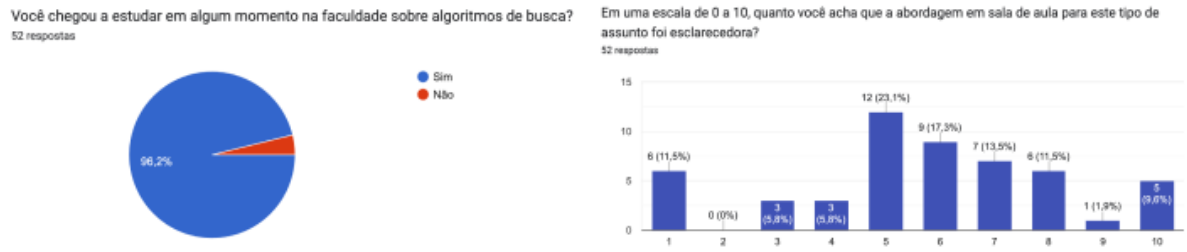


Figura 1.2: Respostas à pergunta 2 e 3 do formulário

Fonte: Autoria própria (2024).

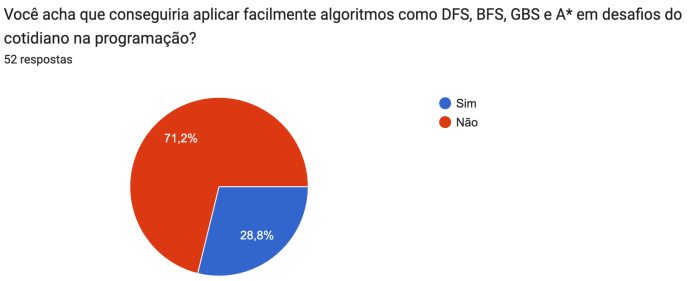


Figura 1.3: Respostas à pergunta 4 do formulário

Fonte: Autoria própria (2024).

Ao desenvolver o Saralk, este trabalho visa se tornar uma ferramenta de apoio para os que desejam aprender mais sobre essa categoria de algoritmos. O jogo contribuirá para uma compreensão mais intuitiva e aprofundada dos conceitos, potencializando o desenvolvimento profissional e acadêmico na área de Ciência da Computação. Este projeto não só responde às necessidades identificadas pela pesquisa, mas também fornecerá uma aplicação inovadora e lúdica para o auxílio no aprendizado de algoritmos de busca.

1.4 Delimitações do trabalho

Este trabalho de conclusão de curso possui as seguintes delimitações:

1. O projeto se limita ao uso de tecnologias específicas para o desenvolvimento da aplicação, sendo elas: (1) JavaScript para a lógica de programação e (2) Vue.js para a construção da interface do usuário;
2. O projeto se limita aos algoritmos DFS, BFS, GS e A*;
3. O projeto se limita ao funcionamento e disponibilização da aplicação através do site `https://saralk.ipb.pt`.

1.5 Público-alvo

O público-alvo deste trabalho consiste em estudantes de graduação dos cursos de Ciência da Computação e áreas afins que enfrentam dificuldades na compreensão de algoritmos de busca.

Capítulo 2

Enquadramento concetual

Este capítulo aborda de forma detalhada os conceitos essenciais envolvidos no desenvolvimento e na implementação do *software* proposto, bem como dos algoritmos de busca. Inicialmente, na Seção 2.1, foi introduzida a estruturação de dados em grafos e árvores no contexto da Ciência da Computação, lançando as bases para compreensão, organização e manipulação de dados complexos. Prosseguindo, na Seção 2.2, mergulha-se no estudo da complexidade computacional, destacando sua importância para a eficiência dos algoritmos, acompanhado de exemplos práticos que ilustram sua aplicação direta. A seção 2.3 a examina as técnicas de busca, enfatizando o funcionamento destas, bem como a relevância dessas estratégias na computação, para a resolução de problemas de *pathfinding*[1]. Complementarmente, a Seção 3.1 apresenta fundamentos de jogos educacionais (ludologia, modelo ARCS e princípios de UX/UI) que orientam o design pedagógico do Saralk, enquanto a Seção 2.4 aborda Interação Humano-Computador, detalhando as modalidades de interação gráfica e programática (via WebSocket) e os critérios de usabilidade adotados.

[1]Processo para encontrar o caminho mais curto entre dois pontos.

2.1 Estrutura de dados

Esta seção apresenta as estruturas de dados que fundamentam o Saralk, iniciando em grafos (orientados, não-orientados e, quando aplicável, ponderados) e sua especialização, as árvores. Em seguida, descreve-se a matriz de adjacência, usada para mapear a grade do tabuleiro para um grafo.

2.1.1 Grafos

Antes do advento dos grafos como uma estrutura de dados fundamental, a computação muitas vezes recorria ao uso de listas para organizar e manipular conjuntos de informações. Embora as listas sejam excelentes para uma variedade de aplicações, como a manutenção de coleções ordenadas de itens, elas apresentam limitações significativas quando se trata de representar relações complexas entre os dados. Sem a capacidade de modelar eficientemente as interconexões entre elementos, resolver problemas de busca e navegação em estruturas de dados altamente interligadas tornava-se um desafio. Foi nesse contexto que os grafos emergiram como uma solução poderosa, superando as limitações das listas ao oferecer uma maneira mais flexível e intuitiva de representar relações [12].

A estrutura de um grafo é basicamente construída com base na conexão entre vértices por arestas ou arcos, veja um exemplo na Figura 2.1. A divisão dos tipos de grafo se manifesta especificamente na ligação entre os vértices, abrindo margem para a existência de grafos orientados e não-orientados. Em um grafo não-orientado, as arestas são bidirecionais por natureza, significando que elas não possuem uma direção específica. Isto é, se existe uma aresta conectando os vértices A e B, então pode-se dizer que A está conectado a B e B está conectado à A indistintamente. Neste tipo de grafo, as arestas são frequentemente representadas por linhas simples ligando dois vértices [12].

Conforme Netto [12], os grafos não-orientados são amplamente utilizados para modelar redes onde a direção da relação entre os vértices é irrelevante ou é mutuamente inclusiva. Exemplos incluem a modelagem de redes sociais, redes de computadores onde as conexões são bidirecionais, e problemas de conectividade, como verificar se existe um caminho entre

Figura 2.1: Arestas e arcos

Fonte: Adaptado de Netto [12].

dois pontos em um mapa.

Grafos orientados, por outro lado, como o da Figura 2.2, possuem arcos que são unidirecionais. Cada arco tem uma direção claramente definida, saindo de um vértice de origem e apontando para um vértice de destino [13]. Isso significa que se existe um arco direcionado de A para B, não necessariamente existe um arco de B para A, a menos que isso seja explicitamente estabelecido por outro arco. Os arcos em grafos dirigidos são frequentemente representados por setas indicando a direção da relação. Estes são utilizados para modelar sistemas no qual a direção das relações entre elementos é crucial. Eles são ideais para representar fluxos, como sistemas de rotas de transporte público (onde a direção é importante), cadeias de suprimentos, diagramas de fluxo de processos, e até mesmo em algoritmos de redes neurais, onde a informação se move em uma direção específica através da rede [12].

Existe ainda a possibilidade de introdução de valores e custos nos arcos ou arestas de um grafo, abrindo uma dimensão adicional de complexidade e utilidade, transformando-os em poderosas ferramentas para modelar e resolver uma vasta gama de problemas práticos. Segundo Netto [12] esses grafos, conhecidos como grafos ponderados ou grafos com pesos, atribuem um valor numérico a cada arco ou arestas (e, em alguns casos, aos próprios arcos ou arestas), representando, por exemplo, o custo, a distância, o tempo, ou qualquer outra métrica relevante para a análise do sistema modelado.

A capacidade de grafos ponderados para representar tais métricas é explorada por uma variedade de algoritmos especializados que buscam resolver problemas como encontrar o

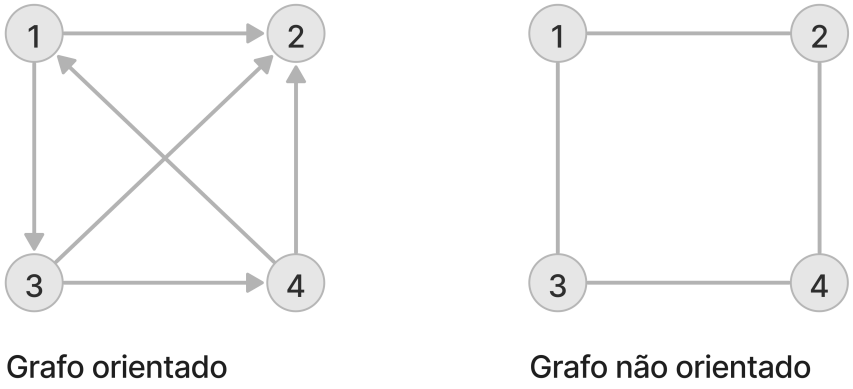


Figura 2.2: Estruturas de grafo

Fonte: Adaptado de Netto [12].

caminho mais curto ou o mais eficiente considerando custos, por exemplo, nos algoritmos GS e A*, que foram investigados nos próximos capítulos e implementados mais a frente [12].

A utilização de grafos ponderados estende a aplicabilidade dos grafos a problemas onde não apenas a estrutura da rede é importante, mas também a magnitude das relações entre seus elementos. Conforme os estudos de Netto [12], os grafos ponderados tornam-se ferramentas indispensáveis em áreas como logística, planejamento urbano, otimização de redes, e muitas outras.

Os grafos são estruturas flexíveis de dados e é o que permitiu a implementação dos algoritmos que serão tratados mais à frente neste trabalho. Foi explorado como estes se aplicam a problemas reais, destacando suas metodologias, complexidades e, especialmente, suas soluções eficazes para desafios de otimização em grafos [12]. A compreensão desses algoritmos em conjunto aos grafos não apenas fornecem percepções valiosas para a resolução de problemas práticos, mas também contribui para o avanço teórico na Ciência da Computação, demonstrando o poder e a versatilidade dos grafos como uma estrutura de modelagem.

2.1.2 Árvores

Devido à arquitetura dos grafos, permite-se criar uma estrutura derivada e igualmente crucial que merece destaque: as árvores. As árvores são uma especialização dos grafos, representam hierarquias e relações de dependência de maneira intuitiva, tornando-se uma ferramenta essencial não apenas na teoria dos grafos, mas em toda a Ciência da Computação [13]. Uma árvore é um grafo conexo que não possui circuitos e, segundo Ziviani et al. [14], possuem a necessidade de considerar todas ou alguma combinação de requisitos, como:

1. Acesso direto e sequencial eficazes;
2. Facilidade de inserção e retirada de registros;
3. Boa taxa de utilização de memória;
4. Utilização de memória primária e secundária.

Uma árvore é uma estrutura de dados fundamental, que organiza elementos de maneira hierárquica, imitando a forma de uma árvore natural invertida, onde se tem uma raiz e ramos que se estendem em várias direções, veja o exemplo abaixo na Figura 2.3. Em termos técnicos, uma árvore é composta por nós, que contêm dados ou valores, e arestas, que conectam esses nós, definindo a relação pai-filho entre eles [13]. A árvore geralmente começa com um nó raiz, de onde todos os outros nós descendem, dividindo-se em níveis e subníveis [15].

Existem vários tipos de árvores de dados, mas todas compartilham entre si, algumas características que são intrínsecas a esta estrutura [15]:

1. **Nó Raiz**: É o primeiro nó, que serve como o ponto de partida da estrutura [16], [17];
2. **Nó Pai**: É um nó que tem um ou mais nós descendentes [18];
3. **Nó Filho**: É um nó que descende de outro nó [19];

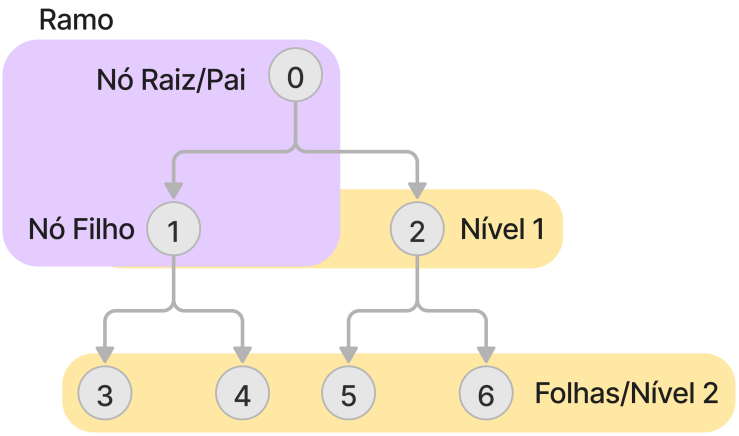


Figura 2.3: Árvore de dados

Fonte: Autoria própria (2024).

4. **Folhas**: São nós sem filhos, representando os pontos terminais da árvore [20];
5. **Aresta**: Ligação que conecta dois nós em uma árvore ou grafo [21];
6. **Fator de ramificação**: Número médio de sucessores que cada nó tem em uma árvore ou grafo [22];
7. **Altura**: É a distância do nó mais distante até a raiz. Em outras palavras, é o número máximo de arestas em qualquer caminho da raiz até uma folha [23];
8. **Profundidade (ou Nível)**: É o número de arestas de um nó específico até a raiz da árvore [24];
9. **Subárvore**: É qualquer nó da árvore, juntamente com seus descendentes, formando uma árvore por si só [25].

As árvores são cruciais para a organização de dados, pois permitem que operações como busca, inserção e remoção sejam executadas de forma eficiente [14].

2.1.3 Representação em algoritmo

A representação algorítmica de grafos e árvores pode ser abordada por meio de uma variedade de métodos. Tais métodos variam desde estruturas de dados simples, como listas armazenadas em arquivos do tipo Valores separados por vírgula (CSV), até matriz de adjacência. Estruturas mais complexas e sofisticadas conseguem armazenar informações adicionais sobre vértices e arestas, enriquecendo o conjunto de dados [26].

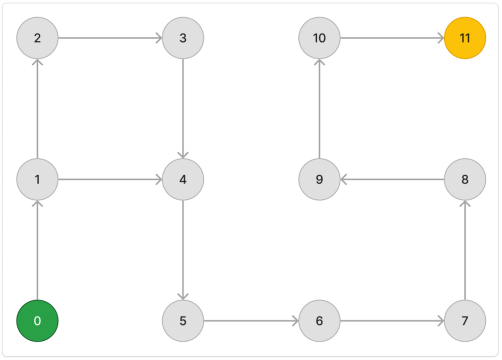
O escopo deste trabalho é delimitado, portanto, não são exploradas exaustivamente todas as possíveis formas de representação dessas estruturas de dados. Em vez disso, é apresentada uma introdução à matriz de adjacência para grafos, que serve como base para o desenvolvimento subsequente da aplicação que é foco deste estudo.

Harary [27] sugere que matriz de adjacência é uma estrutura de dados bidimensional que captura as relações entre os vértices em um grafo. Esta matriz possui dimensões N $\times$ N, onde N representa a quantidade de vértices presentes no grafo. As linhas e colunas da matriz são indexadas conforme os vértices, de modo que o elemento na linha i e coluna j indica o tipo de conexão entre o vértice i e o vértice j [27]. Se os dois vértices estiverem conectados por uma aresta, a célula correspondente contém o valor um (1); caso contrário, contém o valor zero (0), simbolizando a ausência de conexão (veja na Figura 2.4, há um grafo e sua matriz de adjacência correspondente).

Em grafos ponderados, o conceito da matriz é ampliado para refletir os pesos das arestas [27]. Assim, em vez de simplesmente marcar a presença de uma conexão com o valor um, a célula da matriz contém o peso específico da aresta que conecta os dois vértices. Na ausência de uma aresta, o uso de um valor representativo de infinito é comum, indicando não haver caminho direto entre os vértices.

2.2 Complexidade

A análise e compreensão da complexidade dos algoritmos são essenciais no campo da Ciência da Computação, atuando como pilares para o desenvolvimento de soluções eficazes. A expressão do tempo de execução de um algoritmo pode variar significativamente



	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	0	1	0
10	0	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	0	0

Figura 2.4: Exemplo de grafo para matriz de adjacência

Fonte: Autoria própria (2024).

devido a fatores como o processador e estratégias de cache, refletindo a importância de uma abordagem que não dependa das especificidades da máquina [28]. Neste contexto, a notação O e os conceitos de complexidade surgem como ferramentas fundamentais. De acordo com [29] essa notação permite uma avaliação quantitativa do desempenho de um algoritmo que considera tanto o tempo necessário para sua execução (complexidade temporal) quanto a quantidade de recursos de memória utilizados (complexidade espacial), sem depender das características particulares do *hardware*.

De acordo com Chivers e Sleightholme [29] notação O, ou Notação de *Big O*, é um conceito matemático utilizado para descrever o comportamento assintótico das funções, especialmente útil na análise de algoritmos, serve para caracterizar a eficiência de um algoritmo em termos de tempo de execução ou espaço necessário, em função do tamanho da entrada. A notação proporciona uma forma de expressar o crescimento máximo de uma função, ignorando constantes e termos menos significativos [29]. Assim, permite aos desenvolvedores entender como um algoritmo escala e se comporta à medida que o tamanho da entrada aumenta, oferecendo uma base para comparar a eficiência de diferentes algoritmos sob uma perspectiva generalizada, independentemente das particularidades de implementação ou *hardware*.

Esta noção de complexidade não é meramente acadêmica, ela tem implicações diretas e significativas na escolha de algoritmos para resolver problemas específicos, especialmente em um contexto onde os recursos são limitados e a eficiência é crítica. Um algoritmo mais eficiente pode significar a diferença entre uma aplicação que executa em tempo real e outra que falha em responder dentro de um prazo aceitável, ou entre uma operação que utiliza recursos de forma otimizada e outra que excede a capacidade do sistema.

Além disso, a complexidade de um algoritmo afeta diretamente sua capacidade de escalar [29]. À medida que o tamanho do problema aumenta, geralmente, um algoritmo com melhor complexidade temporal ou espacial escala de maneira mais previsível, garantindo que o sistema permaneça responsivo e viável mesmo sob cargas crescentes de trabalho [29].

Por essas razões, a análise da complexidade não é apenas uma etapa crítica no processo de design e seleção de algoritmos, mas também um fator chave na otimização de soluções existentes. Através da compreensão detalhada da complexidade, os desenvolvedores e cientistas da computação podem fazer escolhas informadas que equilibram corretamente a eficiência, a eficácia e o uso de recursos, orientando o desenvolvimento de *software* e sistemas que são robustos, ágeis e escaláveis.

2.2.1 Complexidade de algoritmos

A complexidade de um algoritmo é um conceito fundamental que descreve a eficiência com que ele opera, considerando tanto o tempo quanto os recursos necessários para sua execução. Ela fornece a teoria que permite aos cientistas da computação prever o comportamento de um algoritmo em diferentes condições, oferecendo uma base para comparações objetivas entre diferentes abordagens algorítmicas [28].

Segundo Dasgupta, Papadimitriou e Vazirani [28] entender a complexidade é crucial por várias razões. Primeiro, ela ajuda a identificar os possíveis gargalos de desempenho antes mesmo da implementação do algoritmo, permitindo otimizações proativas. Segundo, em um ambiente de recursos limitados, a escolha de um algoritmo menos complexo pode

significar uma economia significativa de tempo e memória, resultando em sistemas mais rápidos e eficientes. Por último, a análise de complexidade é indispensável para a escalabilidade dos sistemas, assegurando que o aumento no volume de dados ou nas demandas de processamento não degrade inesperadamente o desempenho do sistema.

A complexidade dos algoritmos é comumente medida de duas formas principais: complexidade temporal e complexidade espacial. A complexidade temporal de um algoritmo refere-se à quantidade de tempo que ele leva para ser completado, em função do tamanho da entrada. Esta medida é crucial para entender quão rápido um algoritmo pode resolver um problema. A complexidade temporal é frequentemente expressa utilizando a notação *Big O*, que descreve o limite superior do tempo de execução conforme o tamanho da entrada cresce. Por exemplo, um algoritmo com complexidade temporal O(n) aumentará linearmente o tempo de execução com o aumento do tamanho da entrada [29], veja a Figura 2.5.

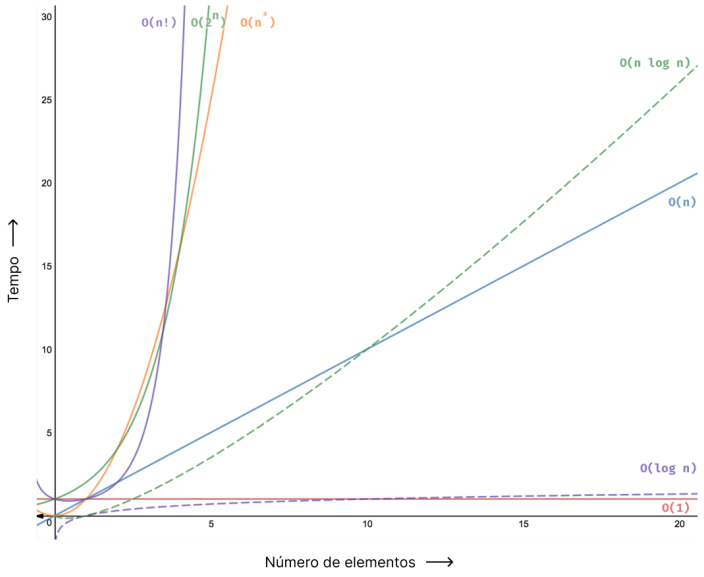


Figura 2.5: *Big O*

Fonte: Adaptado de Mejia [30].

Enquanto a complexidade temporal se concentra no tempo de execução, a complexidade espacial lida com a quantidade de memória que um algoritmo necessita para sua execução. Isso inclui tanto a memória necessária para armazenar as entradas quanto qualquer memória adicional exigida pelo algoritmo para processar essas entradas. Assim como a complexidade temporal, a complexidade espacial também pode ser expressa usando a notação *Big O*, oferecendo uma forma de prever como o uso da memória escala com o tamanho da entrada [28]. Chivers e Sleightholme [29] destaca que um algoritmo com complexidade espacial O(1), por exemplo, usa uma quantidade constante de memória, independentemente do tamanho da entrada.

Ambas as medidas de complexidade, temporal e espacial, desempenham papéis vitais na análise de algoritmos, influenciando diretamente na escolha de soluções algorítmicas para problemas específicos. Ao equilibrar estas duas medidas, os desenvolvedores podem criar algoritmos que não só atendem às expectativas de desempenho, mas também operam nas restrições de recursos disponíveis, garantindo a eficiência geral do sistema.

2.2.2 Análise prática de algoritmos

A análise prática da complexidade dos algoritmos oferece uma visão clara sobre como e por que determinados algoritmos se comportam de maneira diferente sob variadas condições. Ao considerar exemplos de código simples, podemos ilustrar os conceitos de melhor caso, pior caso e caso médio, fundamentais para entender a complexidade de um algoritmo.

Considere o seguinte código como um exemplo genérico de uma busca linear, onde busca-se um elemento específico em uma lista:

Algorithm 1 Busca linear

```
1: procedure BUSCALINEAR(lista, elemento)    ▷ Verifica a presença de elemento em lista
2:     for i ← 0 a tamanho de lista - 1 do
3:         if lista[i] = elemento then
4:             return verdadeiro
5:         end if
6:     end for
7:     return falso
8: end procedure
```

O melhor caso ocorre quando o elemento que estamos buscando é o primeiro elemento da lista. Nesse cenário, o algoritmo encontra o elemento imediatamente, resultando em uma complexidade temporal de O(1), pois a operação é concluída em um número constante de passos, independentemente do tamanho da lista.

O pior caso acontece quando o elemento não está presente na lista ou está na última posição. O algoritmo precisa percorrer toda a lista e comparar cada elemento, resultando em uma complexidade temporal de O(n), onde n é o número de elementos na lista. Isso significa que o tempo necessário para completar a operação aumenta linearmente com o tamanho da lista [29].

O caso médio considera a performance do algoritmo em uma distribuição de entradas típica ou aleatória, isso significaria que o elemento pode estar em qualquer posição da lista. A complexidade temporal, nesse caso, tende a se aproximar de O(n), pois, em média, o algoritmo precisa examinar uma porção substancial da lista antes de encontrar o elemento ou determinar que ele não está presente.

A estrutura de dados escolhida para armazenar os elementos pode impactar significativamente a complexidade dos algoritmos. Por exemplo, se utilizarmos uma lista ordenada ao invés de uma lista comum, podemos aplicar técnicas de busca mais eficientes que reduzem o tempo necessário para encontrar um elemento.

Considere uma lista ordenada e uma busca binária simplificada como alternativa ao exemplo anterior:

Algorithm 2 Busca Binária

```
1: procedure BUSCABINARIA(lista, elemento)   ▷ Realiza busca binária por elemento em lista ordenada
2:     esquerda ← 0
3:     direita ← tamanho de lista - 1
4:     while esquerda ≤ direita do
5:         meio ← ⌊(esquerda + direita)/2⌋
6:         if lista[meio] = elemento then
7:             return verdadeiro
8:         else if lista[meio] < elemento then
9:             esquerda ← meio + 1
10:        else
11:            direita ← meio − 1
12:        end if
13:    end while
14:    return falso
15: end procedure
```

Nesse cenário, a escolha de uma lista ordenada e a aplicação de uma busca binária reduzem a complexidade temporal para O(log n) no caso médio e no pior caso, que significa que o crescimento será logarítmico [29], demonstrando como a estrutura de dados e o algoritmo escolhido podem influenciar diretamente a eficiência da operação, veja abaixo um resumo das principais notações.

Tabela 2.1: Notação Big O e complexidade

Notação	**Nome**
O(1)	Constante
O(log n)	Logarítmico
O(n)	Linear
O(n log n)	Linearítmico
O(n!)	Fatorial
$\boldsymbol{O(n^2)}$	Quadrático

Fonte: Adaptado de Chivers e Sleightholme [29].

2.3 Algoritmos de busca

Algoritmos de busca são técnicas computacionais projetadas para localizar informações ou elementos em grafos, árvores e listas, ordenadas ou não [31], [32]. Esses algoritmos são fundamentais para a Ciência da Computação, ao permitirem a recuperação rápida de dados em vastos conjuntos de informações, otimizando assim o desempenho de sistemas e aplicações. Existem diversos tipos de algoritmos de busca, cada um adequado para situações específicas, pois diferem em complexidade de tempo, complexidade de espaço e objetivo [32].

Na categoria de algoritmos de busca em listas, a busca linear ou busca sequencial é uma abordagem simples que verifica cada elemento sequencialmente até encontrar o alvo, sendo melhor aplicada em listas pequenas ou não ordenadas [33], veja um exemplo da aplicação deste no algoritmo 1. Considere uma lista com 100 itens, onde o valor desejado é o último elemento, nesse caso, como a Figura 2.6 ilustra, o algoritmo de busca linear teria que navegar por todos os 100 elementos para encontrar o valor alvo, ilustrando a natureza ineficiente deste método para listas grandes ou quando o elemento desejado está perto do fim da lista. Pela sua dinâmica, este algoritmo em seu melhor caso, com o

elemento desejado estando na primeira posição, o algoritmo possui complexidade O(1); no caso médio, considerando o elemento desejado sendo o mediano, sua complexidade é de O(n/2); e no pior caso, como este do exemplo, com o elemento sendo o último da lista, sua complexidade é de O(n) [14].

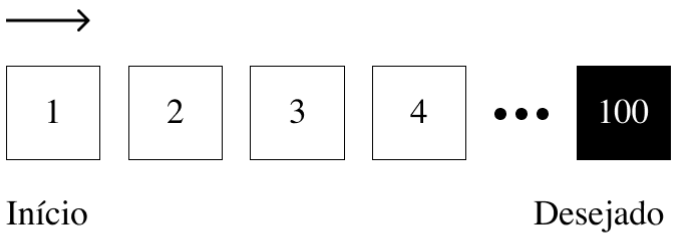


Figura 2.6: Busca linear

Fonte: Autoria própria (2024).

De acordo com Cormen, Leiserson, Rivest et al. [31] a busca binária é uma metodologia diferente de busca, que consiste em separar repetidamente o conjunto de dados, buscando desprezar uma das metades com base na comparação de um elemento inicial ao elemento mediano da lista, sendo melhor empregada em listas ordenadas, confira a aplicação no algoritmo 2. Por exemplo, considere uma lista com 100 itens, onde o elemento desejado é o 69, nesse caso, como a Figura 2.7 ilustra, o algoritmo de busca binária iria comparar o valor desejado com o mediano da lista, neste caso, seria o valor 50, estes dados são analisados e é visto que 50 é menor do que 69, portanto os itens de 1 a 50 são removidos sendo feita uma nova divisão e comparação repetidamente, até que o valor desejado seja encontrado, por causa desta dinâmica, sua complexidade é sempre O(log n) [14].

Após exploradas as nuances dos algoritmos simples de buscas em listas, como a busca linear e a binária, é necessário compreender aplicações em estruturas de dados mais complexas, especificamente os grafos e as árvores. A transição do contexto de listas ordenadas

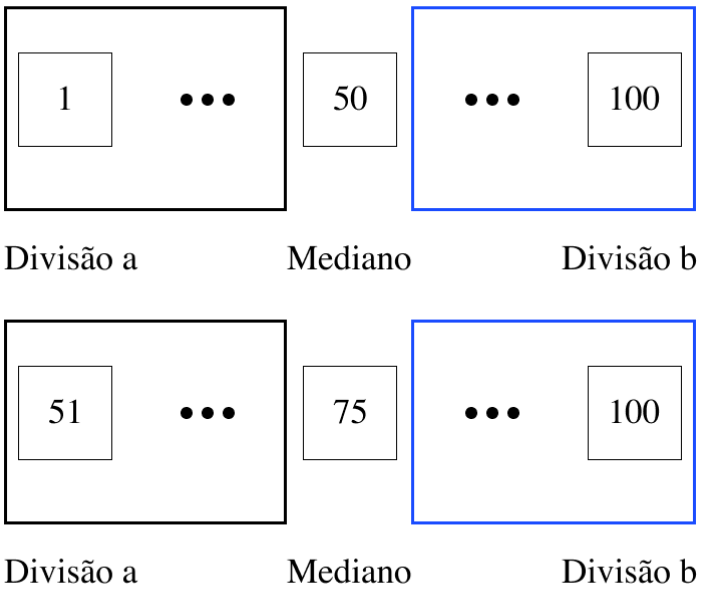


Figura 2.7: Busca binária

Fonte: Autoria própria (2024).

e não ordenadas para o domínio dos grafos e das árvores marca uma evolução significativa e importante para a Ciência da Computação [31], trazendo consigo um conjunto de desafios e oportunidades únicas.

De acordo com Cormen, Leiserson, Rivest et al. [31] algoritmos de busca em grafos representam uma subcategoria vital no universo dos algoritmos de busca, abrangendo técnicas fundamentais como a Busca BFS, DFS, GS, e A*. Essas técnicas são essenciais para a exploração de estruturas de dados complexas e desempenham um papel crucial em uma grande variedade de aplicações computacionais. O entendimento profundo e a implementação eficaz desses algoritmos são fundamentais para resolver problemas que envolvem a determinação de caminhos ótimos ou adequados entre pontos em redes complexas. A escolha do algoritmo de busca mais apropriado para uma determinada aplicação pode ter um impacto significativo na eficiência das operações de pesquisa, influenciando diretamente a velocidade e precisão com que as informações são acessadas e manipuladas em sistemas computacionais.

No contexto do desenvolvimento de sistemas inteligentes, a capacidade de navegar e explorar grafos eficientemente não se limita apenas à busca de caminhos mínimos ou à resolução de puzzles, ela se estende a aplicações mais sofisticadas, como planejamento de rotas em ambientes desconhecidos, otimização de redes de transporte, e até na elaboração de estratégias em jogos de inteligência artificial. A relevância dos algoritmos de busca em grafos no campo da Inteligência Artificial (AI) é evidenciada pela sua aplicabilidade em problemas de *pathfinding*, nos quais agentes autônomos devem encontrar o caminho mais curto ou mais eficiente entre dois pontos, uma situação comum em jogos digitais e sistemas de navegação.

Para compreender a importância desses algoritmos, é crucial analisar não apenas suas características individuais, mas também como eles se comparam entre si em termos de eficiência, complexidade e aplicabilidade. Por exemplo, enquanto o BFS garante a descoberta do caminho mais curto em termos de número de arestas, o DFS pode ser mais eficiente em termos de memória em grafos densos, apesar de não garantir a solução ótima.

Por outro lado, algoritmos informados como o GS e o A* utilizam heurísticas para direcionar a busca, potencialmente alcançando soluções mais rápidas e ótimas comparadas aos métodos não-informados [34].

2.3.1 Algoritmos de busca não-informada

Na esfera dos algoritmos de busca utilizados para navegação e solução de problemas em grafos, existem duas categorias principais: os algoritmos de busca não-informada e os algoritmos de busca informada. Essas categorias divergem fundamentalmente em relação à quantidade e ao tipo de conhecimento prévio que requerem sobre o domínio do problema.

Os algoritmos de busca não-informada, também conhecidos como algoritmos de busca cega, operam sem quaisquer dados adicionais sobre os estados além daqueles fornecidos na definição do problema. Eles não possuem informações sobre o custo ou a qualidade dos caminhos e, por isso, exploram geralmente o espaço de busca de maneira sistemática, mas potencialmente ineficiente. Exemplos típicos de tais algoritmos incluem o BFS e DFS, que são robustos e garantem encontrar uma solução, se ela existir, mas podem não ser ótimos em termos de custo ou distância percorrida (otimalidade).

Busca em largura (BFS)

O algoritmo de Busca em Largura, conhecido como *Breadth-First Search* (BFS), é uma técnica fundamental em Ciência da Computação para explorar grafos ou árvores. Este algoritmo examina sistematicamente todos os vértices de um grafo camada por camada, começando por um vértice de origem, e é comumente utilizado para encontrar o menor caminho em termos de número de arestas entre dois vértices. Devido à sua abordagem uniforme, o BFS é ideal para problemas que requerem uma exploração completa de todos os níveis de um grafo, esta metodologia garante que, se existirem múltiplos caminhos do ponto de origem ao destino, o caminho encontrado pelo BFS será o mais curto em termos do número de arestas atravessadas [34].

Segundo Prolubnikov [35] a invenção do algoritmo BFS é atribuída a Konrad Zuse, que

o descreveu em 1945, embora sua implementação computacional tenha sido popularizada nas décadas de 1950 e 1960 através do trabalho de Edward F. Moore [36], que o usou para resolver problemas de labirinto. Moore propôs o algoritmo como uma maneira de encontrar o caminho mais curto por meio de um labirinto, marcando cada ponto alcançável a partir de um ponto de origem dado. Desde então, o BFS evoluiu para ser um algoritmo amplamente estudado e ensinado em cursos de Ciência da Computação, com aplicações que vão desde a análise de redes até algoritmos de roteamento em informática. Sua evolução é marcada pela expansão em novas áreas e adaptações, como algoritmos que determinam a coloração mínima de grafos ou que verificam a bipartição.

Na busca em largura, os nós com profundidade d são expandidos antes dos nós com profundidade $d+1$, ou seja, os nós são expandidos por ordem de camada, como ilustrado na Figura 2.8, ou seja, primeiro expande-se o nó inicial e em seguida todos os nós alcançados a partir do nó raiz são expandidos, e depois os descendentes dos nós alcançados e assim por diante [37].

As principais vantagens da utilização deste método são as relacionadas à completude e otimalidade, já que o método realiza a verificação de todos os nós da camada 1, depois todos os nós da camada 2 e assim por diante, com isto, é garantido que a solução seja encontrada, se ela houver. Além disto, se o grafo não for ponderado, ou for ponderado com valores constantes, a solução encontrada pode ser considerada uma solução ótima [37].

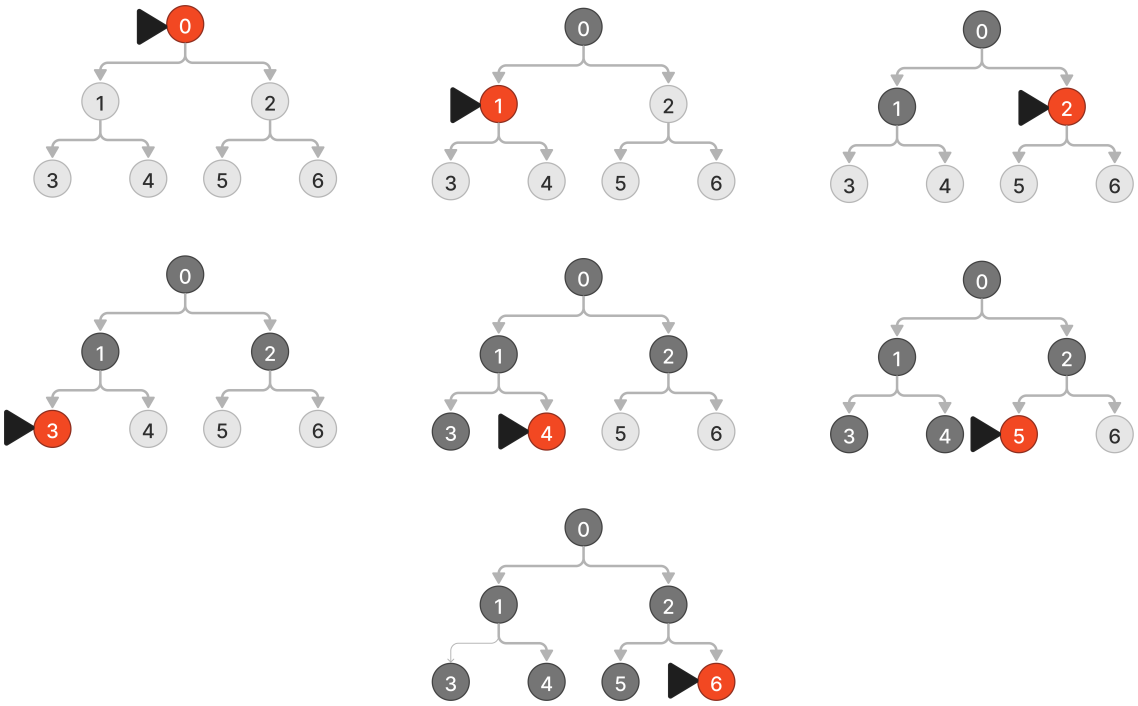


Figura 2.8: Busca em largura

Fonte: Autoria própria (2024).

Algorithm 3 Busca em Largura (BFS)

```
1: procedure BreadthFirstSearch(problem)
2:     node ← new node(state = problem.initialState, pathCost = 0)
3:     if problem.goalTest(node.state) then
4:         return solution(node)
5:     end if
6:     frontier ← new FIFO queue with node
7:     explored ← new empty set
8:     loop
9:         if frontier.isEmpty() then
10:            return failure
11:        end if
12:        node ← frontier.pop()
13:        explored.add(node.state)
14:        for each action in problem.actions(node.state) do
15:            child ← new childNode(problem, node, action)
16:            if child.state not in explored and not frontier.contains(child.state) then
17:                if problem.goalTest(child.state) then
18:                    return solution(child)
19:                end if
20:                frontier.insert(child)
21:            end if
22:        end for
23:    end loop
24: end procedure
```

Fonte: Russell e Norvig [34].

De acordo com Molck et al. [37], as desvantagens do BFS, por sua vez, são decorrentes do custo computacional de tempo e espaço de se visitar camada por camada, a sua complexidade de tempo de $O(b^d)$, onde b é o fator de ramificação e d é a profundidade. Esta complexidade decorre do fato de que cada vértice é inspecionado uma vez e cada aresta é vista uma vez durante o processo de busca, assumindo que o grafo é representado como uma lista de adjacências. A complexidade de espaço para o BFS é $O(V)$, no pior caso, pois o algoritmo precisa manter um registro de todos os vértices visitados, além da fila que pode, em teoria, conter todos os vértices do grafo em um determinado momento, especialmente em grafos densamente conectados.

Busca em profundidade (DFS)

A Busca em Profundidade, em inglês *Depth-First Search* (DFS) é um algoritmo usado para explorar todos os vértices de um grafo ou árvore de maneira exaustiva. Ao contrário da busca em largura, que explora o grafo camada por camada, o DFS visa mergulhar profundamente em um ramo antes de retroceder [34]. Este método é especialmente útil para categorizar componentes conectados e resolver problemas que necessitam de exploração completa, como encontrar configurações possíveis em jogos de tabuleiro, navegando por todas as posições e movimentos até alcançar o objetivo ou um estado terminal.

O conceito do algoritmo DFS não é atribuído a uma única pessoa ou período específico, diferentemente de muitos algoritmos clássicos que têm origens claras. Sua adoção e formulação foram moldadas gradualmente com o desenvolvimento da teoria dos grafos e da Ciência da Computação como disciplinas. Segundo Russell e Norvig [34] a estrutura básica do DFS, explorando cada vértice e aresta uma única vez, reflete uma abordagem natural e intuitiva para percorrer grafos, o que pode ser visto como uma extensão dos procedimentos sistemáticos usados em labirintos e puzzles desde a antiguidade. Ao longo do tempo, com o avanço dos computadores e algoritmos, o DFS foi formalizado e sua eficácia em aplicações específicas como a ordenação topológica e a detecção de ciclos em grafos direcionados foi estabelecida, solidificando sua relevância e versatilidade em problemas computacionais complexos.

Na busca em profundidade, o processo de expansão dos nós, segue um princípio de exploração vertical antes de horizontal. Isso significa que, ao visitar um nó, o algoritmo tenta seguir adiante ao máximo possível ao longo de um ramo, explorando cada filho sucessivo antes de considerar os irmãos. Assim, um nó é expandido assim que é descoberto, e um de seus filhos é imediatamente explorado na sequência [34]. O processo continua até que se alcance um ponto em que não há mais filhos a explorar em um dado ramo, indicando um fundo ou folha do grafo, ou árvore, como apresentado na Figura 2.9. Somente após explorar completamente um caminho, e não havendo mais nós para acessar nesse ramo específico, o algoritmo retrocede para explorar outros irmãos não visitados do último nó visitado. Esse método garante uma abordagem sistemática e profunda antes de mover horizontalmente para outros ramos, assegurando que cada caminho possível seja completamente investigado antes de se prosseguir para alternativas.

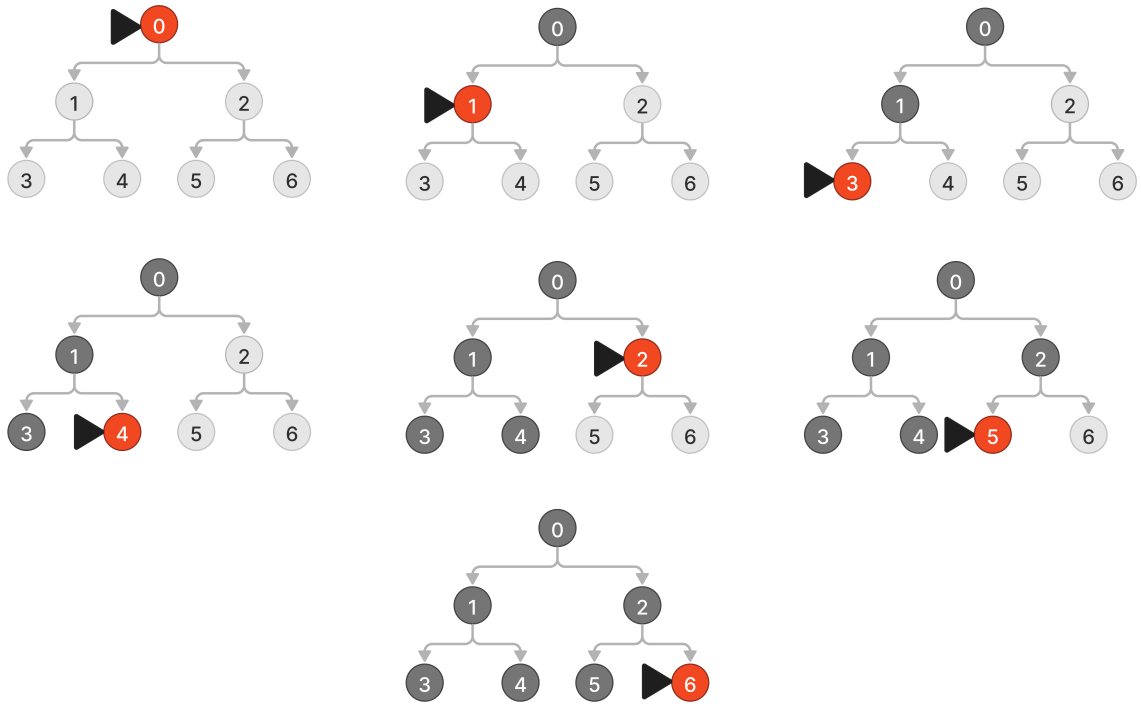


Figura 2.9: Busca em profundidade

Fonte: Autoria própria (2024).

As principais vantagens deste método são, como informado por Molck et al. [37] em relação a critérios de complexidade temporal e espacial, apesar de frequentemente ser mais rápido e ocupar menos espaço em memória, a busca em profundidade possui algumas desvantagens em relação à busca em largura, onde se o problema for grande em relação a sua profundidade, este algoritmo tende a encontrar soluções com custos maiores do que no algoritmo de busca em largura, portanto a busca em profundidade não é ótima e garante a completude em árvores e grafos finitos.

Algorithm 4 Busca em Profundidade (DFS)

```
1: procedure DFS(G, v)
2:     visited[v] ← true                ▷ Marca o vértice v como visitado
3:     processa v                       ▷ Opcional: processa o vértice v
4:     for each u ∈ adj[v] do           ▷ Explora os adjacentes de v
5:         if not visited[u] then
6:             DFS(G, u)                ▷ Chamada recursiva para u
7:         end if
8:     end for
9: end procedure
```

Fonte: Adaptado de Russell e Norvig [34].

A complexidade de tempo do DFS é de $O(b^m)$ onde b é o fator de ramificação e m é a profundidade máxima da árvore [34], porém em problemas com múltiplas soluções ele é superior ao BFS, já que nestes cenários há grande possibilidade de que a solução possa ser encontrada explorando uma pequena parte do espaço. Já a complexidade de espaço é de $O(bm)$, pois é necessário guardar apenas o caminho entre o nó raiz e a folha da árvore Molck et al. [37].

2.3.2 Algoritmos de busca informada

Os algoritmos de busca informada, também chamados de algoritmos heurísticos, empregam conhecimento adicional na forma de heurísticas, estimativas inteligentes que orientam a busca na direção de estados mais promissores, que provavelmente levarão a uma solução mais rapidamente. Isso permite que eles focalizem seus esforços nas partes mais promissoras do espaço de busca, melhorando a eficiência e reduzindo a complexidade temporal e espacial em comparação com as buscas não-informadas. Algoritmos como A* e GS exemplificam essa abordagem, onde a heurística pode representar, por exemplo, a distância direta até o objetivo, permitindo que o algoritmo priorize caminhos que pareçam levar mais diretamente à solução.

Busca gulosa (GS)

A Busca Gulosa, também conhecida como *Greedy Search* (GS), é um algoritmo de busca que segue o princípio de fazer a escolha localmente ótima em cada etapa com a esperança de encontrar o caminho global ótimo até o objetivo [31]. Este algoritmo é amplamente usado em problemas de otimização, onde se busca a solução mais eficiente por meio de escolhas que parecem as melhores no momento.

O uso de estratégias gulosas antecipa os anos 1950, com soluções para árvore geradora mínima já em Borůvka (1926) e Jarník (1930) [38]. O tema ganhou popularidade nas décadas de 1950 e 1960 com os algoritmos de Kruskal (1956) e Prim (1957) para MST e o de Dijkstra (1959) para caminhos mais curtos [39]. Estes algoritmos utilizam estratégias gulosas para resolver problemas que podem ser decompostos em subproblemas, onde escolhas locais ótimas levam a uma solução global ótima. A evolução dos algoritmos gulosos tem sido central para o desenvolvimento de métodos eficientes em teoria dos grafos, otimização combinatória, e outras áreas da pesquisa operacional. Sua adoção em sistemas de tempo real e aplicações onde a velocidade de execução é crítica exemplifica sua utilidade prática e eficácia.

O GS opera tomando a decisão mais benéfica em cada etapa do processo, sem reconsiderar escolhas anteriores, e tampouco escolhas futuras, reduzindo a complexidade computacional mas também implica que a solução encontrada pode não ser a ótima globalmente (exceto em problemas específicos onde a estratégia gulosa é apropriada). A ideia central é sempre escolher o passo que oferece o máximo benefício imediato, conforme o valor heurístico[2], como apresentado na Figura 2.10, onde o GS escolhe os nós com menor custo em arestas [31].

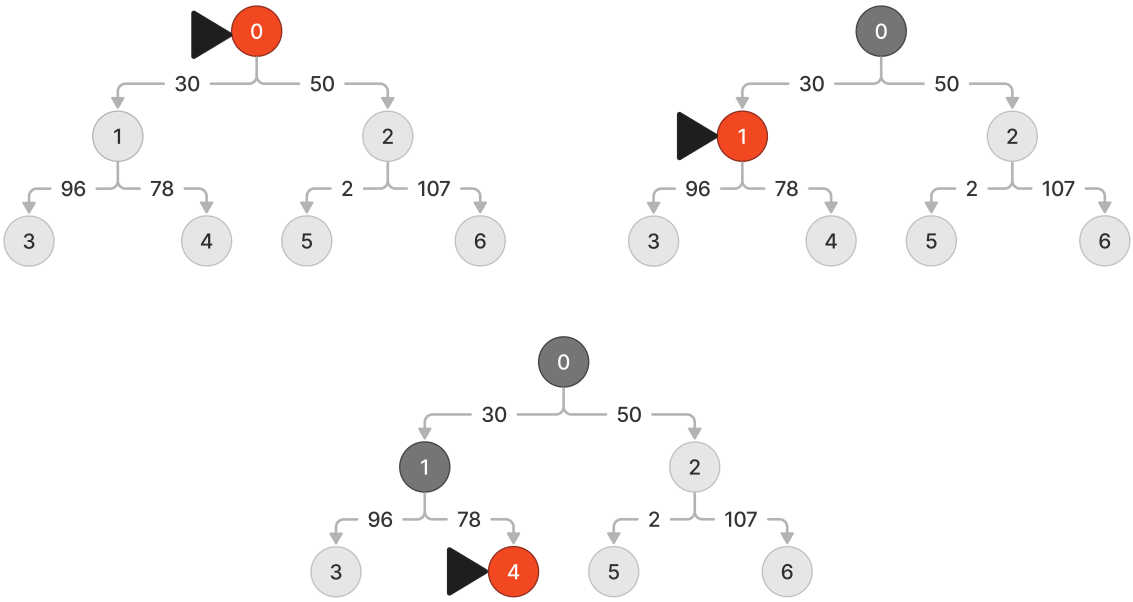


Figura 2.10: Busca gulosa

Fonte: Autoria própria (2024).

O algoritmo GS é conhecido por sua eficiência em tomar decisões que parecem ótimas no momento, visando a obtenção de uma solução globalmente aceitável para problemas de otimização. A complexidade de tempo e espaço deste algoritmo pode variar bastante dependendo do problema específico e da maneira como o algoritmo é implementado. A principal desvantagem é em sua tendência a se assemelhar-se à busca em profundidade, e

[2]Valor estimado usado para guiar a busca por soluções mais eficientes em problemas de otimização.

por isto não é nem ótimo e nem completo [37].

Algorithm 5 Busca Gulosa

```
1: procedure GreedySearch(C, S)
2:     Result ← ∅                          ▷ Inicializa o conjunto do resultado
3:     Available ← C                       ▷ Inicializa os candidatos disponíveis
4:     while Available ≠ ∅ and not GoalReached(Result) do
5:         NextOption ← SELECT-BEST(Available, S)
6:         if IS-FEASIBLE(NextOption, Result) then
7:             Result ← Result ∪ {NextOption}
8:             UPDATE-AVAILABLE(Available, NextOption, S)
9:         else
10:            REMOVE(Available, NextOption)
11:        end if
12:    end while
13:    return Result
14: end procedure
```

Fonte: Adaptado de Cormen, Leiserson, Rivest et al. [31].

Em relação à complexidade de tempo e de espaço, segundo os estudos de Molck et al. [37], este método possui, no pior caso, complexidade de $O(b^m)$ tanto em tempo quanto a espaço, onde m é a profundidade máxima e b o fator de ramificação.

Busca A*

O Algoritmo A* é uma busca informada que se destaca na solução eficiente de problemas de caminho mais curto, oferecendo uma combinação de alguns princípios do GS, como, por exemplo, a utilização de uma heurística, só que com a garantia de completude e otimalidade (quando utilizadas heurísticas eficientes) [37]. Este método é amplamente

utilizado em planejamento de rotas e navegação, otimizando o percurso através da utilização de heurísticas que estimam a distância até o objetivo, reduzindo significativamente o número de explorações necessárias em comparação com algoritmos menos informados como o BFS.

Desenvolvido inicialmente por Peter Hart, Nils Nilsson e Bertram Raphael em 1968 [40]. A principal diferença do A* é sua função de avaliação $f(n) = g(n) + h(n)$, onde $g(n)$ é o custo do caminho desde o início até o nó n, e $h(n)$ é uma heurística que estima o custo de n até o objetivo, como demonstrado pela Figura 2.11. Esta abordagem permite que o A* explore caminhos que parecem mais promissores, reduzindo drasticamente os recursos computacionais necessários e garantindo completude, especialmente em espaços de busca grandes [40].

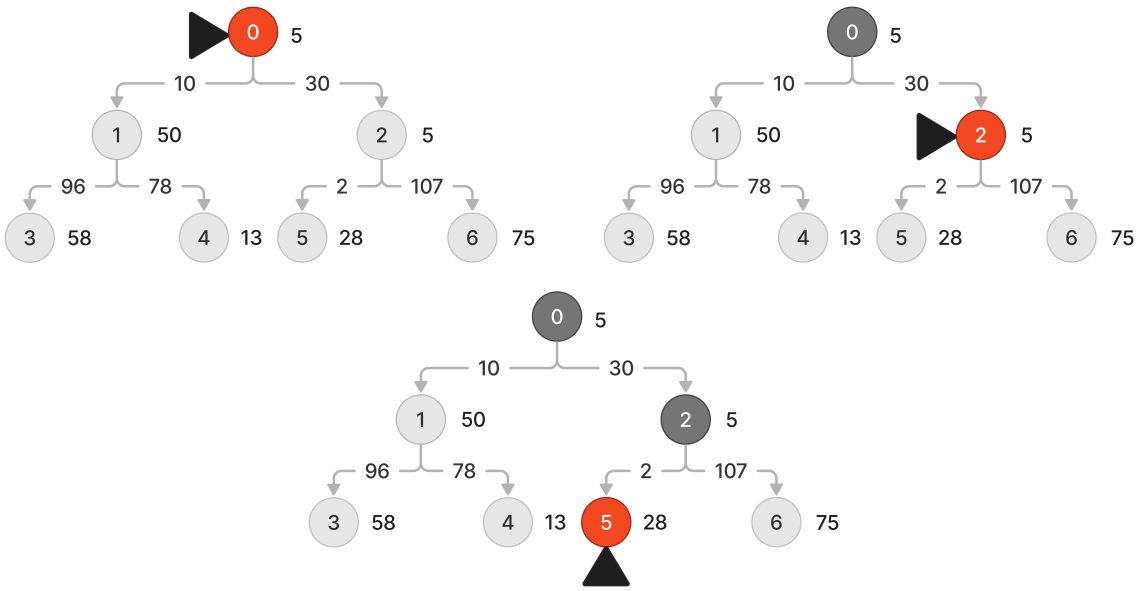


Figura 2.11: Busca A*

Fonte: Autoria própria (2024).

De acordo com Molck et al. [37], ao combinar as características dos métodos GS e BFS, o algoritmo A* se torna razoavelmente eficiente em relação ao tempo gasto na busca, porém a função heurística $f(n)$ precisa ser pensada e programada de maneira inteligente

e eficiente. No entanto, pela maneira como o algoritmo funciona (veja o pseudocódigo 6) este algoritmo armazena todos os nós gerados na memória e a complexidade exponencial resultante pode impedir a utilização do mesmo em diversos cenários [37].

Algorithm 6 Algoritmo A*

```
1:  procedure AStarSearch(graph, start, goal)
2:      openSet ← PriorityQueue({(start, 0)})  ▷ Inicializa a fila de prioridade com o nó inicial
3:      cameFrom ← an empty map                ▷ Guarda o melhor caminho para trás
4:      gScore ← map with default value of Infinity
5:      gScore[start] ← 0                      ▷ Custo do melhor caminho até cada nó
6:      fScore ← map with default value of Infinity
7:      fScore[start] ← h(start)  ▷ Estimativa de custo total do nó inicial até o objetivo
8:      while not openSet.isEmpty() do
9:          current ← openSet.pop()                          ▷ Nó com o menor fScore
10:         if current = goal then
11:             return ReconstructPath(cameFrom, current)
12:         end if
13:         for neighbor ∈ graph.neighbors(current) do
14:             tentative_gScore ← gScore[current] + dist(current, neighbor)
15:             if tentative_gScore < gScore[neighbor] then
16:                 cameFrom[neighbor] ← current
17:                 gScore[neighbor] ← tentative_gScore
18:                 fScore[neighbor] ← gScore[neighbor] + h(neighbor)
19:                 if neighbor not in openSet then
20:                     openSet.add(neighbor, fScore[neighbor])
21:                 end if
22:             end if
23:         end for
```

```
24:     end while
25:     return failure                                  ▷ Caso não encontre o caminho
26: end procedure
27: function ReconstructPath(cameFrom, current)
28:     total_path ← [current]
29:     while current in cameFrom.keys() do
30:         current ← cameFrom[current]
31:         total_path.append(current)
32:     end while
33:     return total_path
34: end function
```

Fonte: Adaptado de Molck et al. [37].

A complexidade de tempo do A* varia conforme a heurística e a estrutura do grafo. No pior caso, especialmente se a heurística não for admissível (não subestimar o custo ao objetivo), o tempo pode degradar para $O(b^d)$, onde b é o fator de ramificação e d é a profundidade do objetivo. Normalmente, com uma boa heurística, o desempenho tende a ser substancialmente melhor que outras abordagens de busca. A complexidade de espaço, por outro lado, é $O(n)$ onde n é o número de nós, pois cada nó pode precisar ser armazenado na memória.

2.3.3 Comparação de algoritmos

Para concluir esta seção, a tabela a seguir sumariza as características de cada algoritmo discutido, com base nos critérios de avaliação adotados.

Tabela 2.2: Características dos algoritmos de busca

Critério	**BFS**	**DFS**	**GS**	**A***
Tempo	b^d	b^m	b^m	b^d
Memória	b^d	$b * m$	b^m	b^d
Ótimo	Sim	Não	Não	Sim
Completo	Sim	Não	Não	Sim

Fonte: Adaptado de Molck et al. [37].

2.4 Interação Humano-Computador

A área de Interação Humano-Computador (IHC) visa compreender e melhorar a forma como os usuários interagem com sistemas computacionais, considerando aspectos cognitivos, ergonômicos, comunicacionais e contextuais [41]. A ascensão tecnológica, associada à popularização de dispositivos e interfaces digitais, torna cada vez mais central o estudo das interações entre humanos e máquinas, sobretudo no desenvolvimento de aplicações educacionais e lúdicas. Para o contexto deste trabalho, as duas interações abaixo se tornam fundamentais.

2.4.1 Interação gráfica

A interação gráfica é uma das formas mais difundidas de interação humano-computador, especialmente em ambientes web. Neste modelo, o usuário utiliza dispositivos como teclado, mouse ou toque para manipular elementos visuais dispostos na interface. Interfaces gráficas bem projetadas favorecem o aprendizado, pois reduzem a carga cognitiva exigida para compreender a estrutura do sistema, facilitando o foco nos objetivos educacionais da aplicação [42].

No caso do Saralk, a escolha pela interação via navegador (*browser-based interaction*)

permitiu ampla acessibilidade, compatibilidade entre plataformas e facilidade de uso, características fundamentais quando o objetivo é atingir estudantes com diferentes níveis de familiaridade com algoritmos de busca.

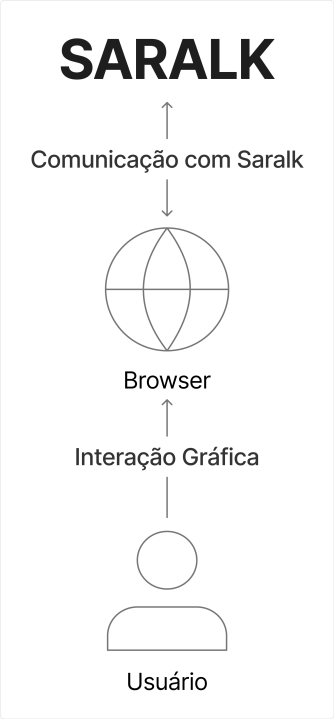


Figura 2.12: Interação gráfica

Fonte: Autoria própria (2025).

A Figura 2.12 exemplifica o modelo de interação do usuário com o Saralk, utilizando a interface gráfica do navegador, permitindo ao usuário executar algoritmos e observar em tempo real o comportamento das buscas.

Dessa forma, a interação gráfica se torna não apenas um meio de controle do sistema, mas também um recurso didático fundamental [42].

2.4.2 Interação programática

A interação programática, neste trabalho foi especialmente projetada para suportar paralelismo, o que permite a execução programática de algoritmos e a manipulação remota

dos elementos visuais da aplicação (veja a Figura 2.13). Essa abordagem é especialmente útil para fins didáticos avançados, simulações automatizadas e integrações externas.

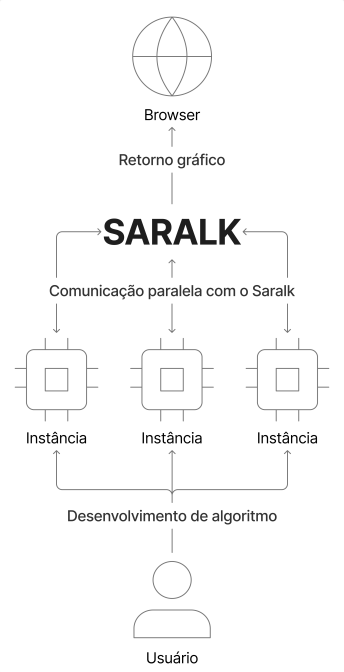


Figura 2.13: Interação programática

Fonte: Autoria própria (2025).

A base dessa interação no contexto deste trabalho é WebSocket, uma tecnologia padronizada pela RFC 6455 que permite estabelecer conexões persistentes e bidirecionais entre cliente e servidor. Diferente das requisições HTTP tradicionais, os WebSockets possibilitam que o servidor envie dados para o cliente sem a necessidade de uma nova requisição, viabilizando atualizações em tempo real e comunicação eficiente com baixa latência [43].

Essa abordagem permite, por exemplo, que comandos de execução de algoritmos sejam enviados diretamente por outros sistemas ou por interfaces alternativas, como terminais de texto ou scripts, e que os resultados sejam refletidos imediatamente na interface gráfica. Isso enriquece a experiência de aprendizagem ao permitir que alunos alternem entre diferentes formas de interação, desenvolvendo também competências em manipulação de

dados e automação.

Capítulo 3

Estado da arte

A aprendizagem baseada em jogos tem vindo a afirmar-se como uma abordagem pedagógica inovadora, que procura conjugar o potencial lúdico dos jogos com objetivos educativos claros. Esta perspetiva, frequentemente associada aos conceitos de game-based learning e de gamificação, destaca-se pela capacidade de promover a motivação intrínseca dos estudantes, fomentar a participação ativa e criar ambientes de aprendizagem imersivos. A literatura evidencia que, ao introduzir desafios progressivos, recompensas e feedback imediato, os jogos podem estimular competências cognitivas, sociais e emocionais, tornando o processo educativo mais envolvente e eficaz.

Na sequência desta abordagem, diversos jogos educativos têm sido desenvolvidos com vista a explorar os benefícios deste paradigma em contextos específicos. Estes jogos variam entre adaptações digitais de conteúdos curriculares, aplicações que simulam cenários práticos de resolução de problemas e experiências interativas orientadas para o desenvolvimento de competências transversais. Ao analisar alguns exemplos representativos, torna-se possível compreender de que forma a integração de mecanismos lúdicos nos processos de ensino-aprendizagem pode contribuir para uma maior eficácia pedagógica, bem como identificar as limitações e os desafios que ainda se colocam à sua implementação.

3.1 Aprendizagem baseada em jogos

Conforme os estudos seguidos por Fernandes [44], desde a antiguidade, os jogos são reconhecidos como instrumentos estimulantes para a aprendizagem. No Brasil, a incorporação dos jogos ao processo educativo se popularizou na década de 80. Com a evolução dos jogos eletrônicos, emergem os jogos digitais, que começam a ser destacados como ferramentas eficazes para o ensino [45].

Segundo Pereira, Fusinato e Neves [46], a construção do conhecimento é interpretada como o desenvolvimento operacional do estudante, isto é, sua habilidade para assimilar os conteúdos fornecidos por meio de materiais didáticos e abordagens metodológicas. Os autores acrescentam que um jogo educativo pode servir como mais um recurso didático disponível para o educador. A principal vantagem de empregar um jogo como material de ensino reside no engajamento do aluno com o desafio proposto pelo jogo, onde o resultado de ganhar ou perder assume significado para ele.

Os jogos digitais têm se tornado uma presença cada vez mais notável no mercado brasileiro, particularmente aqueles de cunho educacional, expandindo sua função para além do mero entretenimento [45]. Apesar de seu potencial evidente, esses jogos ainda encontram barreiras em sua integração no ambiente acadêmico, especialmente nas instituições de ensino superior, inclusive nas que se especializam em tecnologia da informação. No entanto, pesquisas indicam que os jogos digitais possuem um papel significativo no auxílio ao aprendizado de programação e no entendimento de estruturas essenciais no campo da computação [4]. Esta falta de difusão nos contextos universitários pode estar atrelada a diversos fatores, incluindo a resistência cultural a novas metodologias de ensino ou a falta de infraestrutura adequada para implementar tais recursos de maneira efetiva.

Nos últimos anos, a gamificação emergiu como uma tendência relevante em metodologias de ensino, visando engajar estudantes e revisar o conteúdo abordado em sala de aula. Nessa direção, ferramentas como o Kahoot exemplificam a estratégia de aprender jogando e ajudam a reduzir a distância entre conteúdos teóricos e prática em sala [47]. Ainda assim, permanece a questão sobre quais conteúdos da Computação têm sido de fato

contemplados e como os jogos têm tornado visíveis os processos cognitivos subjacentes à resolução de problemas.

Videnovik, Vold, Kiønig et al. [48] apresentam uma revisão sobre jogos educacionais em computação, cobrindo o período de 2017-2021 e sintetizando 113 estudos indexados em IEEE, Springer, Elsevier e PubMed, com protocolo PRISMA-ScR. Os autores observam: (1) crescimento consistente de publicações até 2020; (2) predomínio de investigações no ensino superior e (3) foco marcante em tópicos ligados a programação, especialmente orientação a objetos, ambientes em blocos e desenvolvimento do pensamento computacional. Também destacam que a estratégia pedagógica mais comum é aprender jogando, enquanto abordagens de aprender projetando o jogo aparecem em bem menos trabalhos. Além disso, segundo os autores, não há um jogo ou metodologia padronizada para criação e avaliação de jogos educacionais na área, e faltam estudos em temas além de programação (por exemplo, grafos, redes, segurança).

Em jogos educativos, os estudantes muitas vezes buscam comprovar suas habilidades e conhecimentos adquiridos. No entanto, o medo de cometer erros e ser estigmatizado pode comprometer tanto o processo de aprendizagem quanto o bem-estar psicológico do aluno. Ao enfrentar e superar esses receios, os jogadores podem se tornar mais ativos e comprometidos, beneficiando não apenas seu desempenho escolar, mas também sua atuação na vida diária [46].

3.1.1 Ludologia

A ludologia, também chamada de estudos dos jogos, é uma disciplina focada no estudo de jogos, com um enfoque particular nos videogames [49]. Diferente das abordagens que utilizam a narratologia para analisar jogos, a ludologia examina os jogos através das suas regras e mecânicas, explorando como esses elementos estruturais influenciam a experiência do jogador. O objetivo da ludologia (do Latim ludus [jogo] + logia [estudo]) é entender os jogos como sistemas fechados, onde as interações e os comportamentos no jogo são mais importantes que a história ou narrativa externa [50].

Este campo acadêmico surgiu do debate entre ludologia e narratologia, que discutia se jogos deveriam ser estudados como obras narrativas ou como sistemas de regras. Os ludologistas argumentam que os jogos devem ser vistos através das lentes de suas regras e mecânicas de jogo, pois isso destaca o que é único nos jogos em comparação com outras formas de mídia, como livros e filmes [50].

A ludologia oferece métodos importantes para a análise de jogos e para entender como os jogadores interagem com esses sistemas. Isso envolve estudar como os jogadores aprendem e adaptam estratégias, afetando diretamente suas experiências nos jogos. Por exemplo, a maneira como um jogo premia ou pune comportamentos pode influenciar significativamente o comportamento dos jogadores, similarmente aos conceitos de reforço da psicologia comportamental.

Segundo Gallo [51], os videogames podem também servir como ferramentas educacionais e terapêuticas, auxiliando no desenvolvimento de habilidades cognitivas e motoras. Esta visão apoia o uso de jogos para além do entretenimento, destacando seu potencial em ambientes educacionais e sociais. A ludologia, portanto, não só amplia nossa compreensão dos jogos como fenômenos culturais, mas também auxilia designers a criar jogos que educam e engajam socialmente, promovendo habilidades e mudanças comportamentais positivas.

3.1.2 Modelo ARCS

Segundo Keller [52], a motivação para aprender é essencial em qualquer sistema educacional. Portanto, é crucial que os ambientes de aprendizagem sejam cuidadosamente projetados para estimular um nível adequado de motivação nos estudantes. De acordo com Maehr (1976), a motivação no contexto educacional pode ser descrita como um engajamento voluntário contínuo que leva os estudantes a desejar aprender mais sobre um determinado assunto.

Este conceito, reforçado por Keller e Suzuki [53], destaca a importância de criar ambientes que não apenas informem, mas que também inspirem os alunos a prosseguir seu

Figura 3.1: Fundamentos para geração de motivação para aprender

Fonte: Adaptado de Keller e Suzuki [53].

aprendizado com entusiasmo e interesse. ARCS é um acrônimo para: atenção, relevância, confiança e satisfação, estratégias que, segundo Keller [52] conseguem motivar os alunos na aprendizagem (veja a Figura 3.1).

A atenção dos alunos aos estímulos instrucionais é um aspecto crucial no processo de aprendizagem, e o desafio reside em captar e manter um nível satisfatório dessa atenção [53].

A relevância do conteúdo é fundamental para os alunos perceberem o valor do que está sendo ensinado. Frequentemente, estudantes questionam “Por que tenho que estudar isso?”, e a falta de uma boa resposta indica um problema de relevância que precisa ser endereçado [54].

Adicionalmente, a confiança é uma estratégia essencial que envolve a criação de expectativas positivas nos estudantes. Isso pode ser alcançado ao proporcionar experiências de sucesso que resultem de suas próprias habilidades e esforços. Este fator é decisivo para a persistência dos estudantes no processo educacional [53].

Por fim, a satisfação dos alunos com a experiência de aprendizagem é vital. Isso pode ser promovido mediante recompensas e reconhecimento, além de oferecer oportunidades

para os alunos aplicarem o que aprenderam o mais cedo possível. É importante que os estudantes sintam que seu esforço nos estudos foi apropriado e que houve consistência entre os objetivos, o conteúdo e as avaliações. A sensação de satisfação não apenas melhora a experiência de aprendizagem, mas também motiva os alunos a se engajarem de forma contínua e eficaz no processo educativo [54].

O modelo ARCS é uma estrutura versátil que serve tanto para o desenvolvimento de estratégias motivacionais em novos materiais educacionais quanto para avaliar o nível de motivação em materiais já existentes. Amplamente empregado em diversos estudos, este modelo é utilizado para examinar a motivação dos estudantes ao usarem recursos educativos. Além disso, já foi validado para uso em ambientes interativos e implementado em contextos específicos, como em jogos educacionais, demonstrando sua adaptabilidade e eficácia em diferentes cenários de aprendizado [53].

3.1.3 Experiência do usuário em jogos educacionais

A experiência do usuário, ou em inglês, UX em jogos educacionais é um campo de estudo que abrange uma ampla gama de interações humanas com jogos projetados para fins educacionais. A importância da UX em jogos educacionais reside na capacidade de envolver e motivar os alunos, oferecendo-lhes uma experiência de aprendizagem que é ao mesmo tempo, informativa e agradável [54].

Embora a UX em jogos seja frequentemente debatida, Savi, Von Wangenheim, Ulbricht et al. [54] identificaram apenas quatro modelos para relevantes sobre este tema. Todos esses modelos compartilham a visão de que o conceito de "experiência"é complexo e desafiador para se compreender completamente. Na sua pesquisa, Savi, Von Wangenheim, Ulbricht et al. [54] notaram que, embora não haja consenso sobre os elementos específicos que definem a UX em jogos (Figura 3.2), foi possível identificar e compilar uma lista de elementos recorrentes em pelo menos três dos modelos analisados, sendo eles:

Imersão e Atenção: A imersão é um dos pilares da UX em jogos, especialmente educacionais. Ela se refere à profundidade do envolvimento do jogador no jogo, que muitas

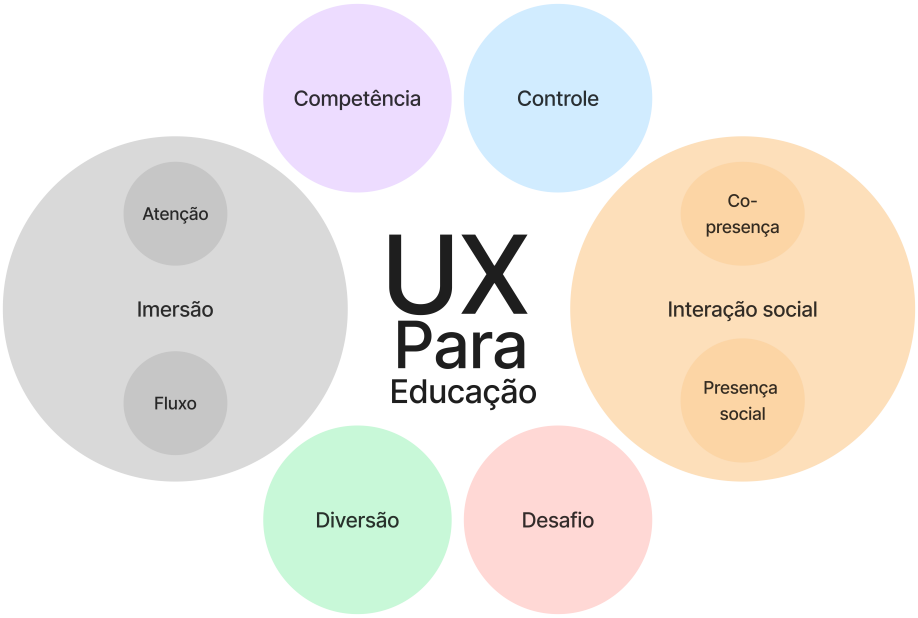


Figura 3.2: Fundamentos de UX para jogos educacionais

Fonte: Adaptado de Savi, Von Wangenheim, Ulbricht et al. [54].

vezes resulta na perda da percepção da realidade e na distorção da noção de tempo. Este alto nível de envolvimento é crucial para o aprendizado, pois quando os alunos estão totalmente imersos, a absorção do conteúdo e o desenvolvimento de habilidades ocorrem de maneira mais natural e eficaz. A atenção, por sua vez, é capturada e sustentada por meio de uma imersão eficaz, garantindo que os alunos permaneçam focados nos objetivos de aprendizagem ao longo de toda a experiência do jogo [54].

Interação Social e Co-presença: A interação social em jogos educacionais desempenha um papel fundamental ao criar um ambiente de aprendizado colaborativo. Conforme Savi, Von Wangenheim, Ulbricht et al. [54], os jogos que incentivam a co-presença e a interação social não só promovem o engajamento com o conteúdo, mas também reforçam habilidades sociais importantes como comunicação, cooperação e competição saudável.

Desafio e Competência: Desafio é outro elemento crucial para a experiência do usuário em jogos educacionais. Um jogo deve oferecer desafios alinhados com o nível de habilidade do jogador, com ajustes e variações de dificuldade que se adaptam conforme

o progresso do aluno. Essa adaptação garante que o jogo permaneça engajador sem causar frustração ou apatia. A sensação de competência que o aluno experimenta ao superar desafios é vital para a motivação e o crescimento contínuo, contribuindo para uma aprendizagem eficaz e gratificante [54].

Diversão e Satisfação: A diversão é um componente essencial que deve estar presente em todos os jogos educacionais. Jogos que são prazerosos e relaxantes podem transformar o aprendizado em uma atividade desejável, aumentando a disposição do aluno para participar e persistir no processo educativo. A satisfação, por sua vez, é alcançada quando os alunos sentem que seus esforços conduzem a resultados significativos e reconhecidos, seja através da conclusão de tarefas difíceis, conquista de objetivos ou superação de desafios [54].

Controle e Autonomia: A sensação de controle é fundamental para a experiência do usuário em jogos educacionais. Os jogadores devem sentir que têm autonomia sobre suas ações no jogo, podendo influenciar o ambiente e o desfecho das situações com suas decisões. Esse controle não só aumenta o engajamento, como também fortalece o sentimento de responsabilidade e a conexão do jogador com o conteúdo educacional.

3.2 Jogos relacionados

Embora o uso de jogos educacionais em faculdades como uma forma de esclarecer conteúdos específicos ainda não seja amplamente adotado, há diversos desenvolvimentos nessa área. Um exemplo é o trabalho de Carvalho e Nery [55], em que foi criado um jogo educacional voltado para o ensino de Engenharia de Produção. Neste jogo, os autores escolheram o ambiente de *Fast Food* para o jogo, por ser um contexto familiar no cotidiano, permitindo a fácil identificação de diversos aspectos da Engenharia de Produção, como Gestão da Qualidade, Ergonomia, Segurança do Trabalho, Gestão de Estoques, Logística, Gestão de Pessoas e Engenharia de Sustentabilidade.

O jogo desenvolvido apresenta um design gráfico que simula um *Fast Food* (veja na Figura 3.3), com metas baseadas na satisfação de clientes e funcionários. A dinâmica do

Figura 3.3: Jogo de *fast food* desenvolvido para área de Engenharia de Produção

Fonte: Adaptado de Carvalho e Nery [55].

jogo envolve a gestão do tempo de entrega dos pedidos, que afeta diretamente a satisfação dos clientes. Isso está ligado ao número de funcionários disponíveis e à eficiência de cada um, que diminui com o aumento do volume de trabalho. Uma barra verde na tela inicial indica o nível de satisfação geral, se ela cair muito, o jogo termina. Os autores destacam que o objetivo principal foi alcançado: simular de maneira simplificada e lúdica um processo produtivo, engajando os alunos efetivamente [55].

O X-MED v1.0, por sua vez, é um jogo educacional de simulação que ensina os conceitos e práticas de medição de *software* no contexto de gerência de projetos, desenvolvido por Wangenheim, Thiry, Kochanski et al. [56]. O principal intuito, de acordo com seu criador, é complementar o ensino tradicional, o jogo simula a definição e execução de um programa de medição, sendo projetado para oferecer aos alunos de pós-graduação e profissionais de Engenharia de *Software* uma maneira prática de aplicar teorias e conceitos aprendidos em sala de aula em um ambiente controlado e interativo.

Na dinâmica do jogo, os jogadores passam por várias telas (veja a Figura 3.4) que replicam as etapas reais de um programa de medição, incluindo a caracterização do contexto, definição de objetivos de medição, e desenvolvimento de planos de coleta e análise de dados. Cada decisão tomada pelo jogador é seguida de *feedback* imediato, permitindo uma

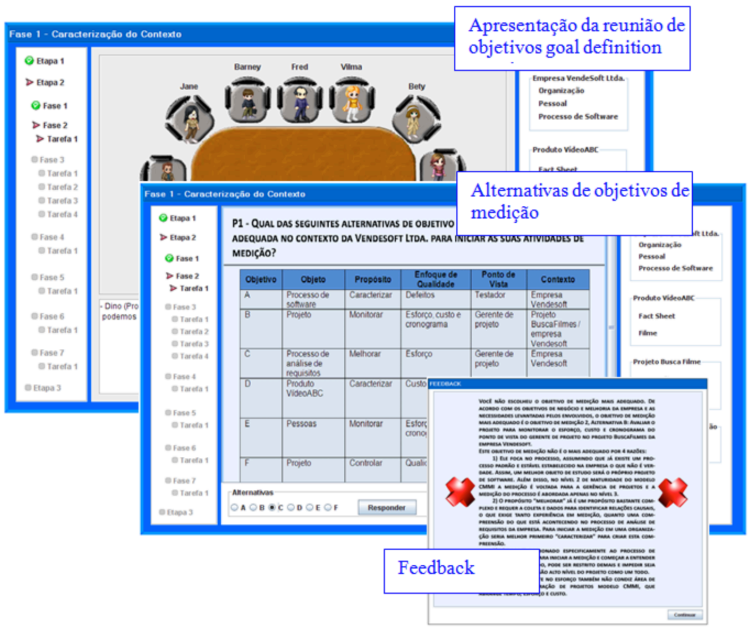


Figura 3.4: X-MED v1.0: Um jogo para ensino de medição de *software*

Fonte: Wangenheim, Thiry, Kochanski et al. [56]

aprendizagem ativa e baseada em resultados. Essas tarefas são apresentadas de maneira sequencial e o jogador precisa escolher entre várias alternativas propostas, aprendendo com os acertos e erros no próprio jogo.

O X-MED v1.0 aborda o problema da falta de capacitação prática em medição de *software*, um conteúdo que, conforme o autor, em sala de aula é abordado somente teoricamente, e que, ainda segundo Wangenheim, Thiry, Kochanski et al. [56] dificulta a aprendizagem de habilidades práticas frequentemente exigidas no mercado de trabalho.

Em outro caso, os resultados de Oliveira Melle, Braga, Pimentel et al. [57] mostraram que uma abordagem revisada da metodologia INTERA no desenvolvimento do jogo educacional no estilo RPG "Antártica"(um jogo projetado para ensinar conceitos de educação científica, utilizando um cenário envolvente de pesquisa na Antártica), foi bastante eficiente e promissora para futuros estudos.

As principais etapas da metodologia INTERA são: (1) contextualização; (2) requisitos; (3) Arquitetura; (4) Desenvolvimento; (5) Testes e Qualidade; (6) Disponibilização; (7)

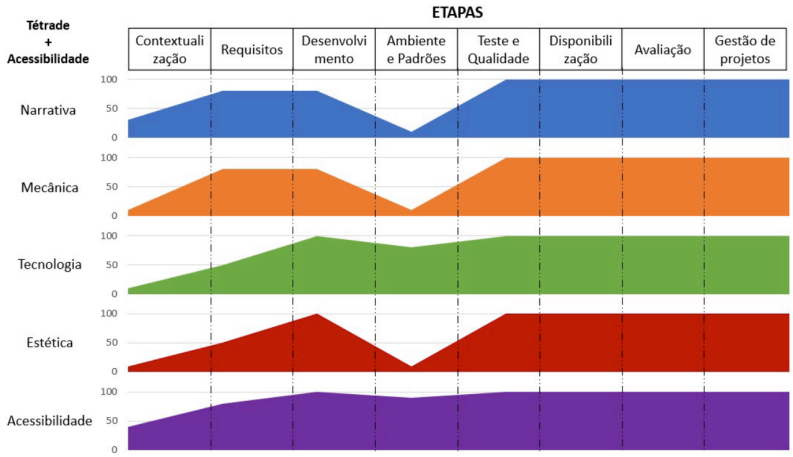


Figura 3.5: Etapas da metodologia INTERA em relação aos elementos da tétrade

Fonte: Oliveira Melle, Braga, Pimentel et al. [57]

Avaliação; e (8) Gestão de projetos. Neste trabalho em questão, o autor fez uma revisão da metodologia INTERA e somou-os com os elementos da tétrade e acessibilidade, sendo eles (1) Narrativa; (2) Mecânica; (3) Tecnologia; (4) Estética; e (5) Acessibilidade.

Ao final, Oliveira Melle, Braga, Pimentel et al. [57] concluiram que a edição repensada do INTERA foi útil e resultou em um bom conjunto de etapas para a criação de um jogo RPG educacional, principalmente por nenhuma etapa ter deixado de ser executada durante alguma fase, como é possível notar pela Figura 3.5, e propõe que estas etapas também podem ser aplicadas em outros tipos de jogos educacionais, buscando resolver o problema da falta de engajamento e aplicação prática no aprendizado por meio de jogos digitais [57].

Dentre os trabalhos encontrados, dois em especial se destacaram pela similaridade dos objetivos em relação ao objetivo geral deste trabalho, disponível na seção 1.1.1. A Última Árvore, de Junior, Cavalheiro e Foss [58], propõe uma metodologia inovadora para fomentar o desenvolvimento do Pensamento Computacional por meio de um jogo educacional. Este jogo é baseado na Gramática de Grafos e é apresentado como um jogo de tabuleiro de estratégia por turnos, veja a Figura 3.6, onde os alunos assumem papéis de animais visando restaurar uma floresta quase totalmente destruída.

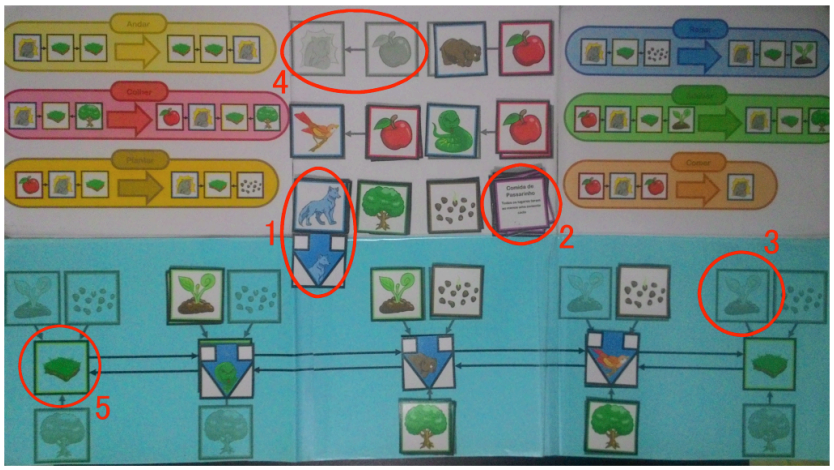


Figura 3.6: A Última Árvore: um jogo para ensino sobre grafos

Fonte: Junior, Cavalheiro e Foss [58]

A escolha da Gramática de Grafos como base para o jogo permite uma representação visual que facilita a abstração de dados e processos, aspectos fundamentais no desenvolvimento do Pensamento Computacional. O jogo enfatiza habilidades críticas como coleta, análise e representação de dados, paralelismo e abstração, promovendo assim uma maneira lúdica e engajadora de ensinar competências computacionais fundamentais.

A implementação do jogo é descrita como "desplugada", o que significa que não são necessários computadores ou outros recursos tecnológicos avançados para a sua execução. Essa característica torna o jogo acessível e fácil de ser implementado em diversos contextos educacionais, incluindo escolas que podem não ter acesso a laboratórios de informática bem equipados [58].

Por último, Alencar, Pessoa e Pires [59] descrevem o desenvolvimento do jogo educacional Doce Sort, criado para auxiliar a compreensão das propriedades de árvores binárias de busca, especialmente destinado a estudantes de graduação em computação. O jogo, inspirado em populares títulos como "Overcooked"e "Candy Crush", é um puzzle ambientado numa doceria e desenvolvido na plataforma Unity (veja a Figura 3.7).

Os autores descrevem a organização em três principais modos de jogo: alocação, inserção e caminho, cada um com quinze fases. Através destes modos, os jogadores enfrentam

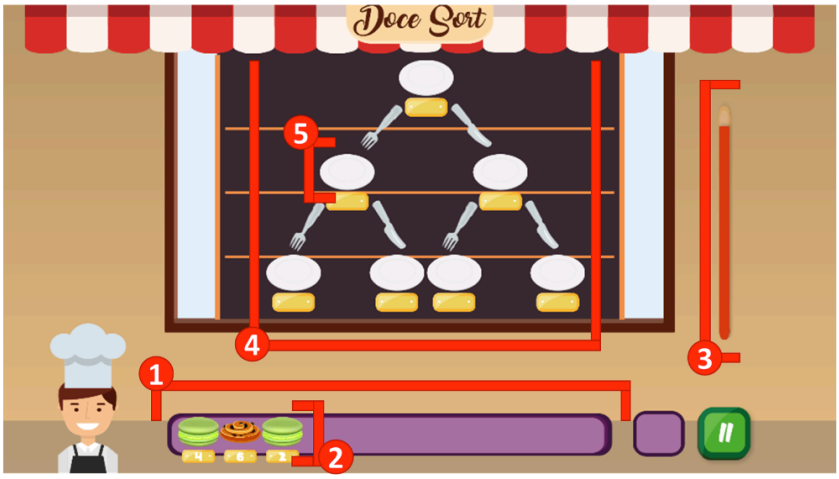


Figura 3.7: Doce Sort: um jogo para exercitar propriedades de árvores binárias de busca

Fonte: Alencar, Pessoa e Pires [59]

desafios que refletem diretamente os conceitos de árvores binárias de busca, onde devem alocar, inserir e navegar por produtos de uma confeitaria de forma que respeitem as regras dessa estrutura de dados. O design e a mecânica de jogo foram cuidadosamente pensados para tornar a aprendizagem desses conceitos computacionais mais acessível e engajante.

Conforme o criador, ao integrar elementos visuais atraentes com desafios estruturados em torno de árvores binárias de busca, o Doce Sort se tornou um valioso recurso pedagógico para facilitar a compreensão de conceitos complexos de computação de uma maneira lúdica e interativa.

Como observado nos trabalhos revisados, apenas um explora algoritmos de busca, porém, ainda sim não explora os algoritmos DFS, BFS, GS e A*, e consequentemente, torna o Saralk flexível para demonstrar tanto algoritmos de busca informados quanto não-informados. Além disso, os trabalhos revisados carecem de um sistema interno de visualização dos cálculos em execução. Outra limitação observada é a ausência de modos automáticos de jogo, oferecendo apenas uma modalidade manual, restringindo ao usuário a possibilidade de estudo passivo.

Capítulo 4

Metodologia

Para a realização deste trabalho e consequentemente, o desenvolvimento do Saralk, elaborou-se um fluxograma das atividades realizadas (Figura 4.1). A metodologia foi estruturada em quatro etapas: (1) levantamento e consolidação do referencial teórico e do estado da arte; (2) design do jogo; (3) desenvolvimento; e (4) validação com o público-alvo.

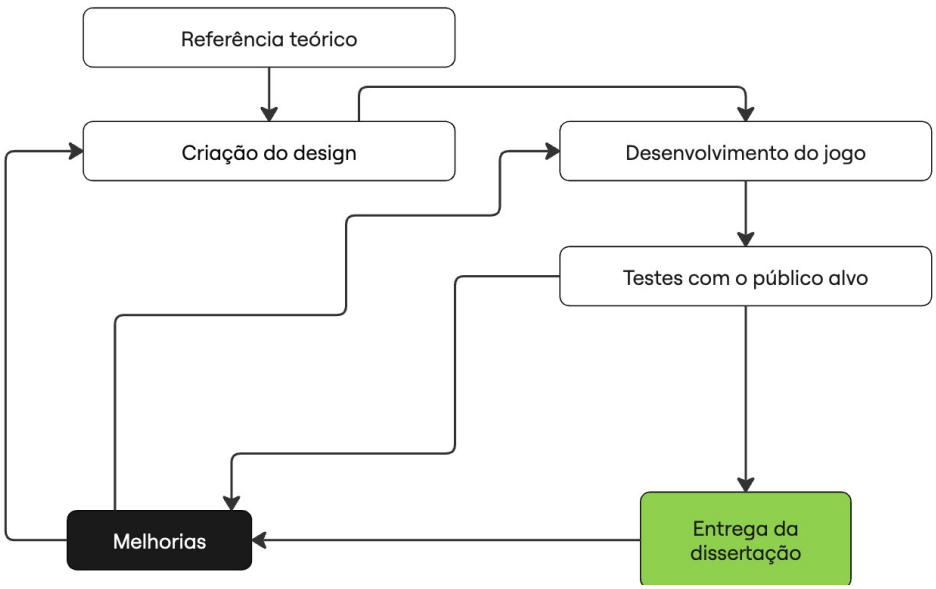


Figura 4.1: Fluxograma de atividades para o desenvolvimento do Saralk

Fonte: Autoria própria (2024).

O referencial teórico desempenhou um papel crucial no embasamento da metodologia.

Nesta fase foi realizado o estudo necessário para a compreensão dos métodos de busca utilizados, como, por exemplo, o estudo de grafos, árvores e conceitos de complexidade (tempo e espaço), além disso, os estudos se estenderam ao próprio exame dos métodos de busca DFS, BFS, GS e o algoritmo A*. Por fim, essa etapa também envolveu uma revisão detalhada da literatura existente, buscando identificar e analisar artigos, trabalhos, paradigmas e técnicas recentes que são pertinentes ao desenvolvimento de jogos educativos.

Estes estudos não apenas fortaleceram a base teórica do projeto, mas também ajudaram a garantir que a abordagem adotada estivesse alinhada com as práticas mais atuais e eficazes no campo. Neste contexto, as obras de Gomes [50], Keller e Suzuki [53] e Savi, Von Wangenheim, Ulbricht et al. [54] ganharam destaque, respectivamente: por sua análise crítica do debate entre ludologia e narratologia; pelo método consistente para criação de jogos educativos e pelas técnicas de UX relevantes para esta categoria de jogos.

4.1 Protótipo

O protótipo (Figura 4.2) foi concebido no Figma[1] com base nos elementos de UX para jogos educacionais identificados na revisão (imersão, competência, controle, interação social, desafio e diversão) [54]. Esses princípios nortearam os componentes centrais do Saralk.

O desenvolvimento do jogo, correspondente à terceira etapa do projeto, utilizou a linguagem de programação JavaScript e o *framework* VueJS para a criação dos componentes visuais, conforme o design definido na etapa anterior. Além disso, este processo adotou técnicas alinhadas ao modelo ARCS, reforçado por Keller e Suzuki [53] e Keller [52], para geração de motivação no aprendizado, aspecto apontado por Keller [52] como essencial em qualquer sistema educacional. Como resultado, o protótipo na Figura 4.2 ilustra os itens do modelo ARCS e UX alcançados pelos componentes-chave do Saralk, sendo estes:

Cronômetro: Fornece um limite de tempo para a conclusão das tarefas. O jogador

[1]Plataforma de design usada para criar interfaces de produtos digitais.

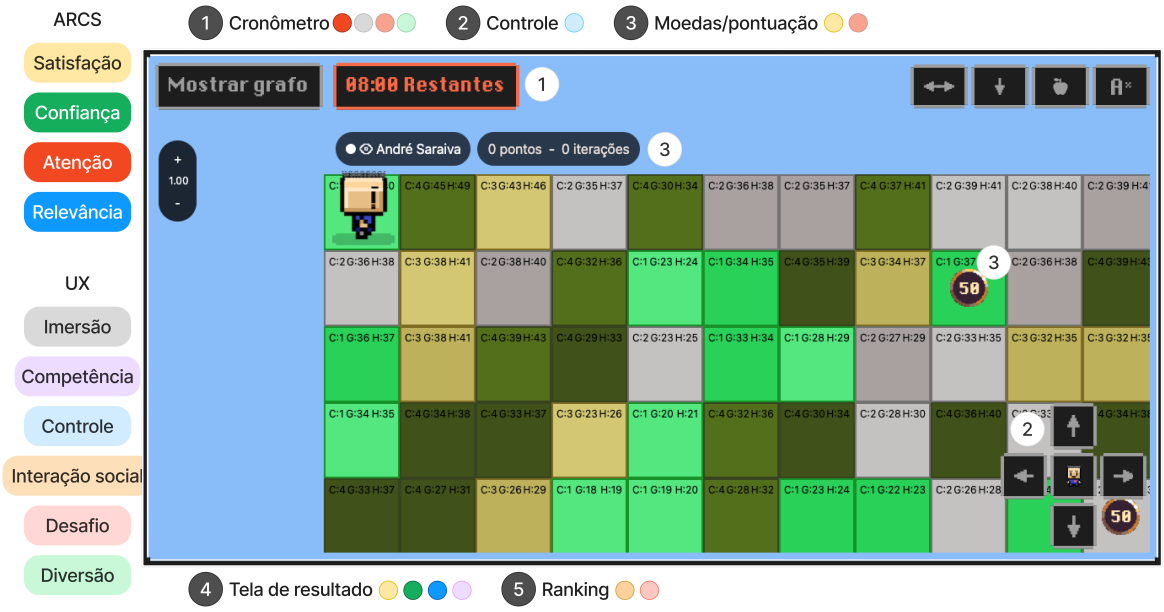


Figura 4.2: Protótipo do Saralk

Fonte: Autoria própria (2024).

tende a prestar atenção ao cronômetro porque ele sinaliza quanto tempo resta para completar as missões, incentivando o foco nos objetivos do jogo. Além disso, o cronômetro contribui para a imersão, pois ao observar a contagem regressiva, o jogador instintivamente se envolve no jogo.

Controle: O controle permite que o jogador interaja diretamente com o ambiente do jogo, reforçando o senso de controle sobre suas ações. A sensação de comandar os movimentos do personagem e interagir com objetos no cenário tende a proporcionar ao jogador sentimentos como autonomia e domínio, fundamentais para mantê-lo engajado (veja na Figura 4.3).

Moedas/Pontuação: A conquista das moedas tem como finalidade proporcionar satisfação imediata, promovendo o incentivo à busca por mais. O desafio se dá pela satisfação do usuário ao coletar mais moedas e atingir pontuações melhores, criando objetivos claros que estimulam a progressão no jogo.

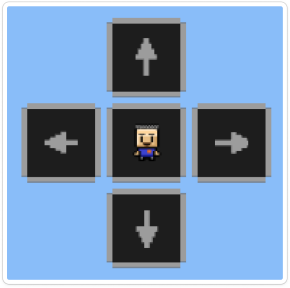


Figura 4.3: Controle do Saralk

Fonte: Autoria própria (2024).

Tela de resultado: Oferece *feedback* imediato sobre o desempenho do jogador, resultando em satisfação, confiança, relevância e competência. A satisfação surge ao mostrar resultados positivos. A confiança é reforçada quando o jogador percebe seu progresso. A relevância é percebida mediante avaliações contextualizadas, e a competência é reforçada ao destacar as conquistas do jogador, mostrando claramente seu crescimento e progresso ao longo do jogo (veja na Figura 4.4).

Ranking: Contribui para a interação social ao incentivar uma competição saudável entre os jogadores e, para o desafio, fornecendo um objetivo adicional de superar os colegas (veja o exemplo de multijogador na Figura 4.5). Ver o próprio nome no topo do ranking ou subir posições deve proporcionar senso de realização e motivação para melhorar constantemente.

O diagrama de atividades representado pela Figura 4.6 resume o fluxo principal do jogo. Após iniciar a sessão, o jogador pode usar o Controle (Figura 4.3) para mover-se manualmente ou selecionar um algoritmo no menu para acionar o modo automático.

No modo automático, o algoritmo selecionado é executado para determinar a direção mais vantajosa, conforme o método escolhido, veja um exemplo de execução do A* (com heurística) na Figura 4.7.

Também é possível visualizar os cálculos que estão sendo considerados para a decisão

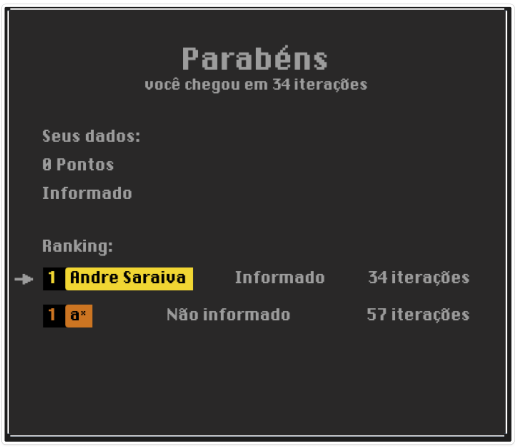


Figura 4.4: Tela de resultado do Saralk

Fonte: Autoria própria (2024).

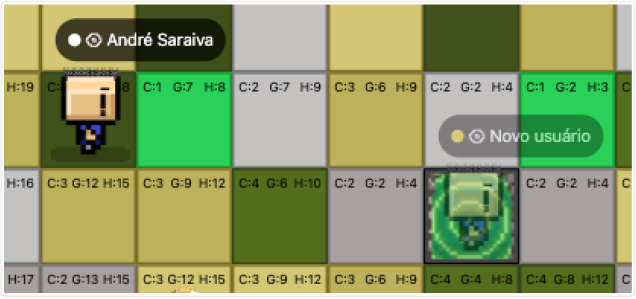


Figura 4.5: Exemplo de multijogador

Fonte: Autoria própria (2024).

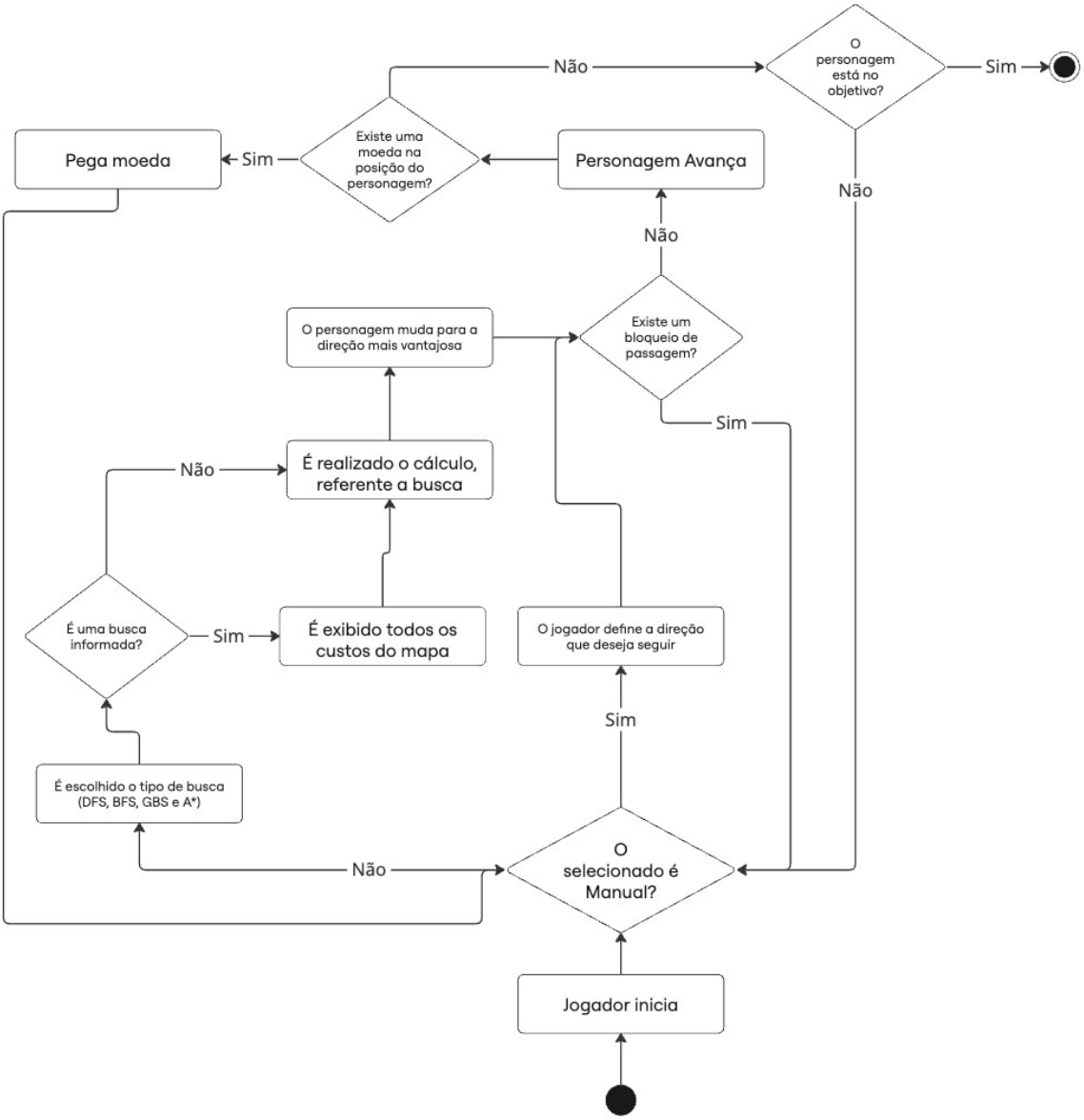


Figura 4.6: Diagrama de atividades do Saralk

Fonte: Autoria própria (2024).

Figura 4.7: Exemplo de funcionamento do Saralk

Fonte: Autoria própria (2024).

dos algoritmos clicando no botão de “+” ao lado do nome do algoritmo, conforme a Figura 4.8.

Figura 4.8: Processo de decisão dos algoritmos do Saralk

Fonte: Autoria própria (2024).

No modo manual, o usuário controla o movimento do personagem pelo tabuleiro, evitando obstáculos e coletando moedas. Ao final do jogo, o desempenho do jogador é comparado ao desempenho dos algoritmos de busca selecionados e a outros jogadores.

A primeira versão do Saralk implementa as seguintes regras: (1) diante de um obstáculo, o personagem não avança; (2) ao coletar uma moeda, a pontuação é atualizada; e (3) a partida termina quando o objetivo é alcançado ou quando o tempo do cronômetro se esgota.

A escolha das tecnologias decorreu diretamente dos requisitos definidos no protótipo, assim, adotou-se uma *stack* web com JavaScript na lógica e Vue.js na camada de interface, solução que favorece reatividade, baixo tempo de resposta e portabilidade entre

dispositivos, em consonância com os objetivos de demonstrar e operar os algoritmos tanto visualmente quanto por código.

4.2 Tecnologias

4.2.1 JavaScript

JavaScript é uma linguagem de programação que foi inicialmente criada para tornar as páginas web mais interativas. A linguagem foi desenvolvida por Brendan Eich em 1995 enquanto trabalhava na empresa Netscape e, inicialmente, foi chamada de ECMAScript, depois a Microsoft criou outra versão da linguagem, formalmente chamada de JScript, porém, é popularmente chamada de JavaScript [60]. Este rápido desenvolvimento foi uma resposta da Netscape, à necessidade de uma linguagem de scripting que pudesse ser embutida diretamente no navegador, para melhorar a experiência do usuário sem a necessidade de plug-ins externos ou reenvios excessivos de páginas para o servidor [60].

A adoção do JavaScript não se restringiu apenas a tornar páginas web mais dinâmicas. Com o tempo, seu papel expandiu-se significativamente à medida que a Internet cresceu. A introdução do Ajax no início dos anos 2000 foi um ponto de inflexão para JavaScript, permitindo que as aplicações web enviassem e recebessem dados de um servidor de forma assíncrona, sem interferir na exibição da página atual. Isso possibilitou o desenvolvimento de aplicações web dinâmicas e responsivas, como mapas online e plataformas de mídia social, que poderiam carregar um conteúdo sem recarregar a página inteira [60].

Hoje, JavaScript é considerado uma das linguagens de programação mais populares do mundo, sendo usado não apenas em navegadores, mas também em servidores, através do Node.js, e em uma variedade de outras aplicações, incluindo desenvolvimento de jogos e aplicativos móveis. A linguagem é suportada por todos os navegadores modernos e é fundamental para o desenvolvimento de aplicações *front-end*. Além disso, *frameworks* e bibliotecas como React, Angular e Vue.js utilizam JavaScript para criar interfaces de usuário interativas e eficientes.

4.2.2 VueJS

VueJS é um *framework* progressivo para JavaScript usado para construir interfaces de usuário e aplicações de página única, em inglês, *Single Page Applications* (SPA). Criado por Evan You em 2014, VueJS ganhou rapidamente popularidade por sua simplicidade e por abordar alguns dos problemas que os desenvolvedores enfrentavam com outros *frameworks* mais pesados, como, por exemplo, o ReactJS, o qual de acordo com Incau [61] é mais lento do que o Vue em todos os cenários, veja a Tabela 4.1. Com sua arquitetura leve e modular, VueJS permite aos desenvolvedores adicionar incrementos ao *framework* somente onde e quando é necessário, tornando-o ideal tanto para adicionar interatividade simples a websites como para construir aplicações complexas.

Tabela 4.1: Comparação entre ReactJS e VueJS

Cenário	**VueJS**	**ReactJS**
Rápido	23ms	63ms
Média	51ms	94ms
Lento	343ms	453ms

Fonte: Adaptado de Incau [61].

Este *framework* é conhecido por seu sistema reativo para manipulação de dados. Isso significa que sempre que o estado da aplicação muda, ele atualiza automaticamente o *Document Object Model* (DOM) para refletir essas mudanças eficientemente. Isso é realizado mediante um sistema de detecção de dependências que rastreia quais componentes precisam ser re-renderizados quando o estado muda, garantindo um desempenho rápido mesmo em aplicações complexas.

Essa base técnica viabiliza visibilidade do estado (nós visitados, custos, pontuação), consistência de rótulos/ícones e incrementos de acessibilidade (fonte, contraste, teclado). Assim, as tecnologias escolhidas não só sustenta o protótipo como prepara, diretamente, os critérios de usabilidade discutidos a seguir.

4.3 Usabilidade

A usabilidade (um dos pilares da IHC) guiou tanto a interface gráfica quanto a estrutura interna de comunicação/extensibilidade. No ambiente visual, é priorizada a navegação simples, botões visíveis e respostas responsivas, permitindo observar, em tempo real, as etapas de execução dos algoritmos, aspecto valioso para perfis visuais e cinestésicos [62].

No modo não-informado, o jogo revela apenas os blocos adjacentes ao jogador, simulando o conhecimento limitado desses algoritmos (Figura 4.9).

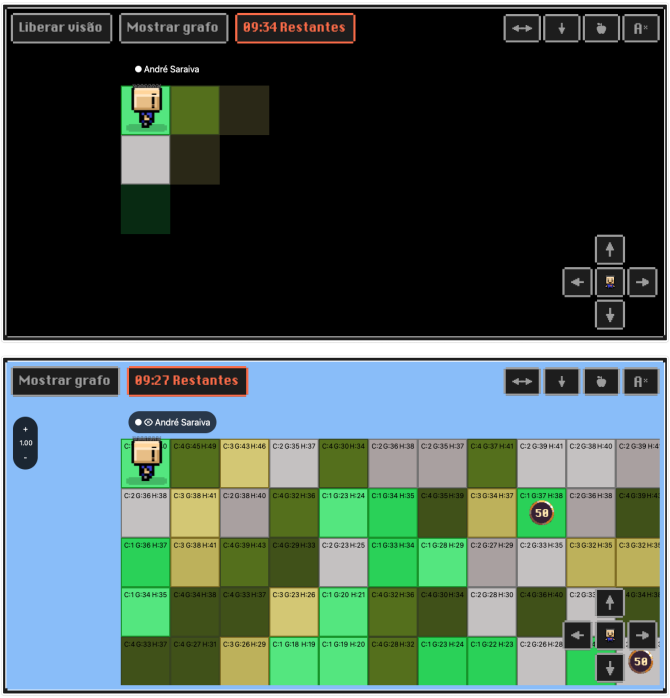


Figura 4.9: Modo não-informado e informado no Saralk

Fonte: Autoria própria (2025).

Ao clicar em “Liberar visão”, exibe-se o mapa completo com os valores C (custo), G (distância até o objetivo) e H (heurística G+C), destacados no exemplo da Figura 4.7.

4.3.1 Visualização do trajeto em grafo

Além da visualização em grade, o Saralk oferece a opção “Mostrar grafo”, que exibe o trajeto percorrido pelo jogador como um grafo direcionado (Figura 4.10). Essa visualização tem o objetivo pedagógico de aproximar o estudante da conversão entre a matriz de adjacência (representação padrão usada no jogo por ser um tabuleiro) e um grafo, reforçando a ponte entre as duas formas de modelar o mesmo problema.

Figura 4.10: Trajeto percorrido em grafo

Fonte: Autoria própria (2025).

Para manter a coerência visual, as cores dos nós no grafo correspondem às cores dos quadrados do tabuleiro. Quando um bloco é visitado mais de uma vez, o nó torna-se circular e recebe o símbolo 🔁, destacando revisitas/loops no caminho. Se o bloco visitado contiver moeda/pontuação, o nó apresenta o símbolo 💰, permitindo identificar rapidamente onde houve coleta de pontos. Os arcos (setas) representam a sequência de movimentos executados ao longo da partida, favorecendo a leitura do percurso e a análise posterior. Em conjunto com os modos não-informado/informado, a visualização em grafo amplia a transparência do processo de busca e facilita a transferência de conhecimento entre a experiência lúdica na grade e a notação clássica de grafos usada em disciplinas de

Estruturas de Dados e IA.

4.3.2 Interação via código

Além da interface gráfica, o Saralk oferece uma camada adicional de usabilidade por meio de sua interface programática baseada em *WebSocket*. Essa abordagem permite que estudantes avancem para um nível mais técnico de interação com o sistema, controlando o ambiente de simulação por meio de comandos diretos enviados via código. A aplicação adota o protocolo *WebSocket* conforme definido na RFC 6455, possibilitando comunicação persistente e bidirecional entre cliente e servidor, característica essencial para cenários educacionais que exigem atualização em tempo real, interatividade contínua e baixo tempo de resposta.

A comunicação entre cliente e servidor é orientada a eventos e ocorre por meio de mensagens padronizadas em formato Notação de Objeto JavaScript (JSON). O cliente pode enviar comandos específicos ao servidor para controlar o movimento do jogador, solicitar informações do tabuleiro ou manter a conexão ativa. Os comandos válidos estão listados na Tabela 4.2.

Tabela 4.2: Comandos enviados do cliente para o servidor via *WebSocket*

Comando	**Descrição**
`top`	Solicita movimentação do jogador para o bloco superior, respeitando regras de travamento
`left`	Solicita movimentação para o bloco à esquerda
`right`	Solicita movimentação para o bloco à direita
`down`	Solicita movimentação para o bloco inferior
`getBoardInfo`	Solicita os dados completos do tabuleiro, incluindo heurísticas, transformando o jogador em "informado"

Fonte: Autoria própria (2025).

Em resposta a essas interações, o servidor envia eventos que notificam os jogadores sobre alterações no estado do jogo, como movimentações, novas conexões, erros ou atualizações do tabuleiro. Esses eventos estão listados na Tabela 4.3.

Tabela 4.3: Eventos enviados do servidor para o cliente via *WebSocket*

Evento	Descrição
`newPlayer`	Notifica que um novo jogador entrou na sala, transmitindo seus dados
`yourPlayer`	Retorna ao jogador recém-conectado seus dados, bloco atual e blocos adjacentes
`gameInfo`	Envia informações gerais do jogo, como sala, jogadores e pontuação
`movePlayer`	Atualiza a posição de um jogador no tabuleiro e retorna os blocos adjacentes
`removePlayer`	Informa que um jogador desconectou-se da sala
`boardInfo`	Resposta ao comando `getBoardInfo`, com todos os dados do tabuleiro
`invalidAction`	Indica que a ação enviada pelo cliente é inválida ou não permitida

Fonte: Autoria própria (2025).

Do ponto de vista pedagógico, a interação via código representa uma oportunidade para que estudantes desenvolvam habilidades avançadas em algoritmos, redes e automação. Ao permitir o envio direto de comandos, criação de agentes personalizados e integração com sistemas externos, o Saralk estimula o pensamento computacional e o desenvolvimento de soluções criativas. A persistência dos dados em banco de dados e a manutenção de estado entre sessões tornam a experiência ainda mais robusta, permitindo que o aluno retome experimentos ou análises com continuidade e profundidade.

Capítulo 5

Resultados e Discussão

A validação pedagógica ocorreu em duas turmas do 4º período de cursos ligados à Computação no IPB (veja a Figura 5.1), ela seguiu mediante a uma exposição teórica de 30 minutos sobre os temas necessários para o entendimento de algoritmos de busca, sendo eles: (1) Estrutura de dados; (2) Grafos e árvores; (3) Big O e complexidade; e (4) algoritmos de dados (algoritmos informados e não-informados). Após a aula teórica os estudantes utilizaram o Saralk (`https://saralk.ipb.pt`) e em seguida responderam a um questionário sobre autopercepção de conhecimento, experiência de uso e indicação.

Para esta etapa, foi criado um formulário online que conteve as seguintes perguntas: (1) Como você avaliaria o seu conhecimento em algoritmos de busca antes da aula de demonstração?; (2) Você sentiu que conseguiu aplicar os conceitos aprendidos na prática dentro do jogo?; (3) Qual dos algoritmos você achou mais fácil de entender com o Saralk?; (4) Após jogar o Saralk, como você avaliaria o seu conhecimento em algoritmos de busca?; (5) Como foi a experiência ao utilizar o Saralk?; (6) Você indicaria o Saralk para outros estudantes que estão aprendendo algoritmos de busca? Por quê?; e (7) Alguma sugestão de melhoria para o jogo ou para a aula?

A amostra totalizou 21 respostas (coleta em 06/05/2025 e 07/05/2025), o que dá lastro estatístico suficiente para observar tendências no grupo e interpretar as demais métricas do questionário. A Figura 5.2 apresenta o perfil geral dos participantes.

Quanto à aplicação prática dos conceitos no próprio jogo, 47,6% responderam “Sim,

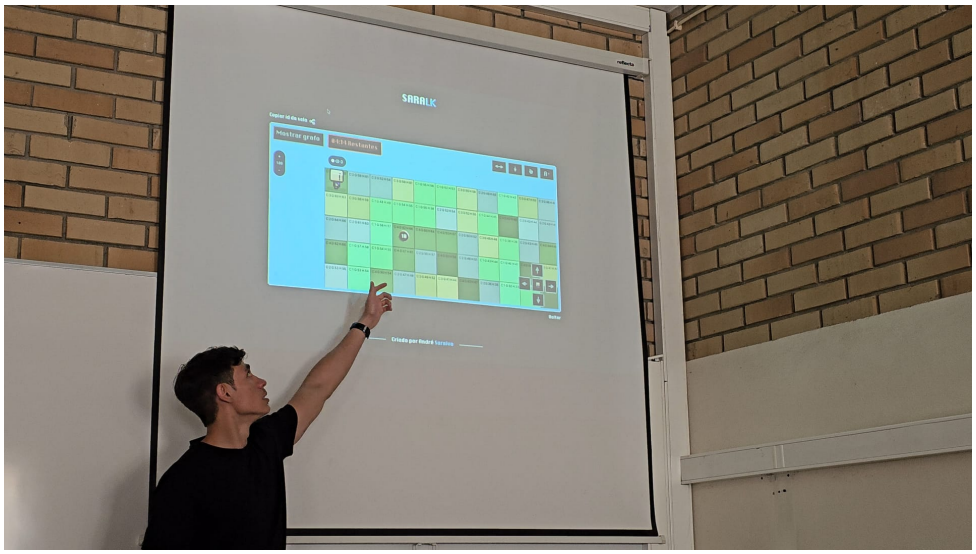


Figura 5.1: Apresentação no IPB - 06/05/2025

Fonte: Autoria própria (2025).

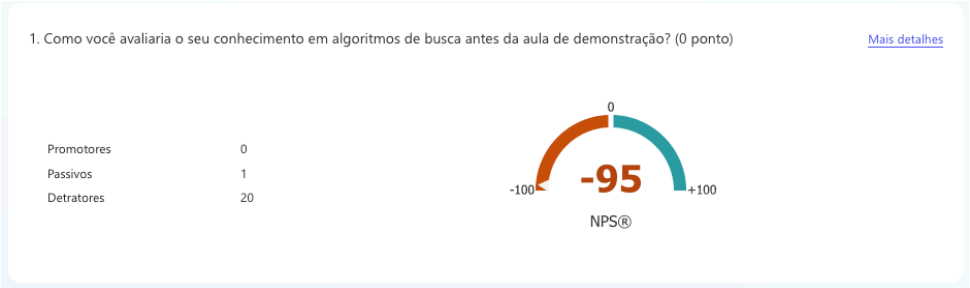


Figura 5.2: Respostas à pergunta 1 do formulário final

Fonte: Autoria própria (2025).

totalmente", 42,9% "Em parte"e 9,5% "Não muito", sugerindo que 90,5% perceberam uma melhora no entendimento da teoria com a prática imediatamente (Figura 5.3).

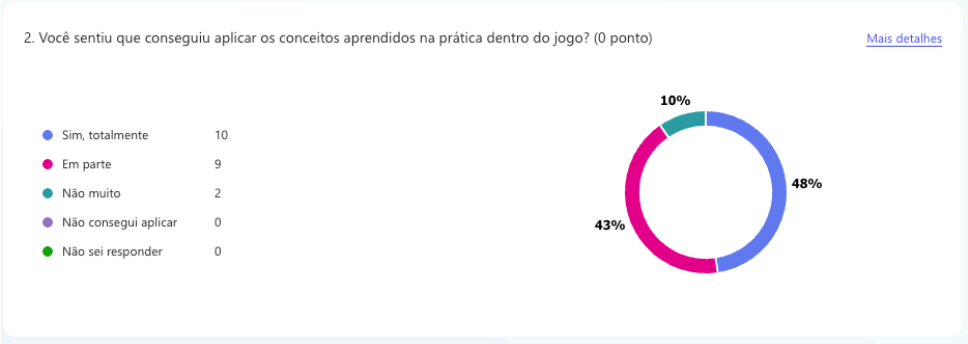


Figura 5.3: Respostas à pergunta 2 do formulário final

Fonte: Autoria própria (2025).

Sobre qual algoritmo ficou mais fácil após a atividade, destacou-se DFS com 47,6%, seguida de GS com 33,3%, BFS e A* obtiveram 9,5% cada (Figura 5.4).

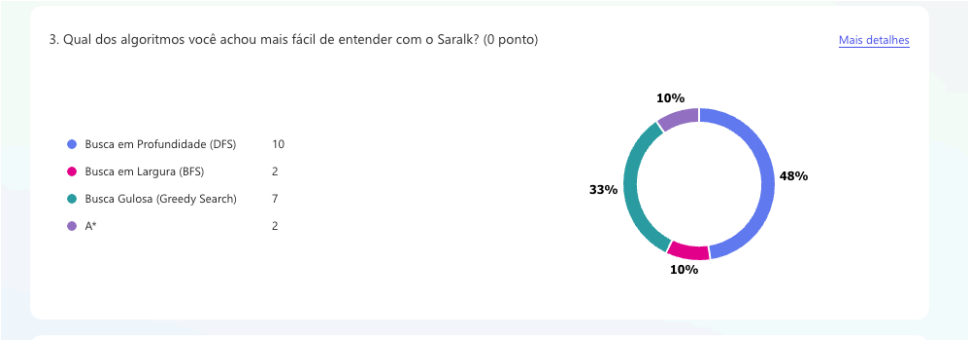


Figura 5.4: Respostas à pergunta 3 do formulário final

Fonte: Autoria própria (2025).

A Figura 5.5 mostra que a autopercepção de conhecimento (escala 0–10) evoluiu de 2,43 (pré) para 6,38 (pós), com ganho médio de 4 pontos. Este patamar sugere melhora substancial após a interação com o Saralk.

A experiência de uso foi altamente positiva, com uma média de 8,4 e NPS® +33 (9 promotores, 10 passivos e 2 detratores), em linha com expectativas de IHC/UX para *feedback* imediato e visibilidade do estado do sistema (Figura 5.6).

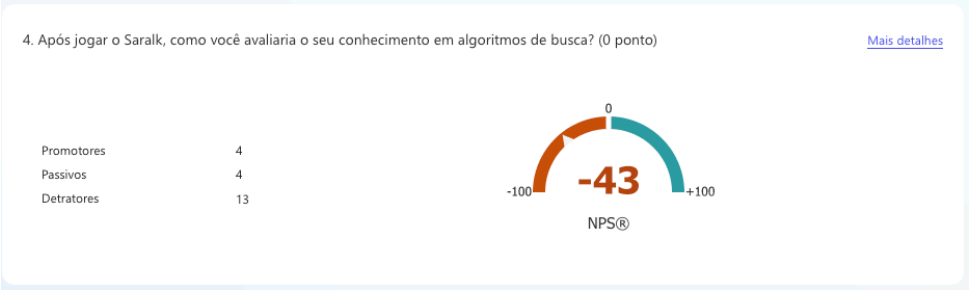


Figura 5.5: Respostas à pergunta 4 do formulário final

Fonte: Autoria própria (2025).

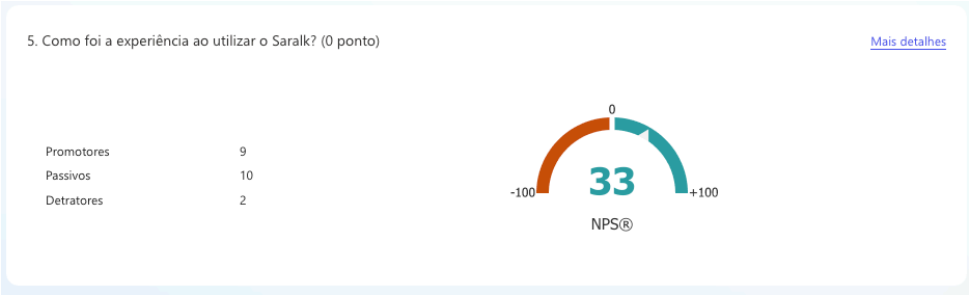


Figura 5.6: Respostas à pergunta 5 do formulário final

Fonte: Autoria própria (2025).

Na Figura 5.7 é mostrada a intenção de recomendação, onde, todos os alunos deram respostas positivas, com respostas como por exemplo: “Sim, porque nos ajudar a perceber melhor sobre busca”, “Sim, é possivel aplicar os diferentes algoritmos no jogo e perceber bem como funcionam.”, “Sim, tem uma boa demonstração prática de como funcionam os algoritmos.”.

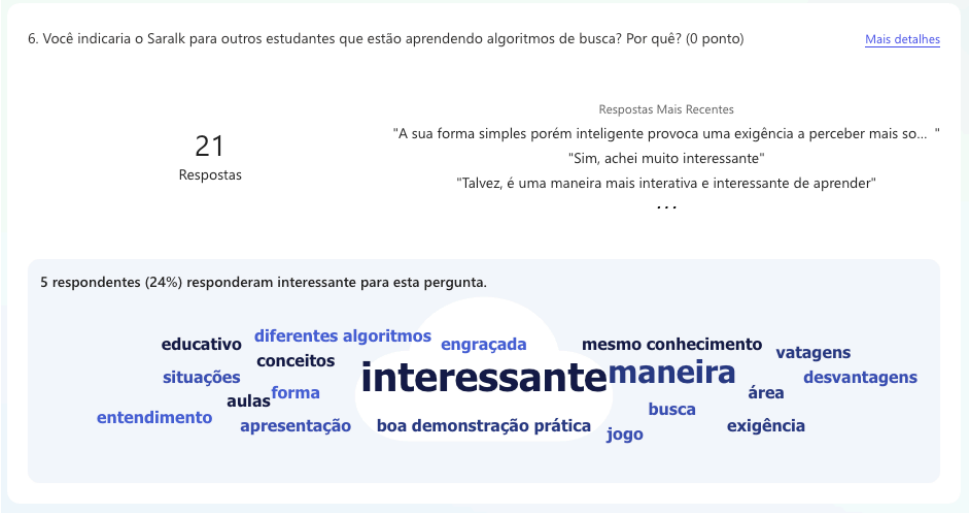


Figura 5.7: Respostas à pergunta 6 do formulário final

Fonte: Autoria própria (2025).

Por fim, nas sugestões de melhoria, cerca de 43% responderam “não”(sem sugestões), ao passo que as demais contribuições se concentraram em quatro frentes recorrentes: (1) fluxo/navegação; (2) legibilidade/feedback visual; e (3) robustez dos algoritmos (especificamente em GS/A*). A Figura 5.8 resume essas ocorrências.

Este resultado destaca a utilidade de ferramentas como o Saralk, especialmente na visualização de algoritmos com comportamento progressivo e exploratório, o que demonstra que representações interativas são particularmente benéficas para o entendimento desses paradigmas de busca.

Para além do ambiente acadêmico formal, o Saralk foi apresentado no evento *Bragança Gaming Takeover* (15 de março de 2025), a um público heterogêneo que incluiu estudantes do ensino fundamental ao superior. Embora se trate de uma ação de extensão com caráter

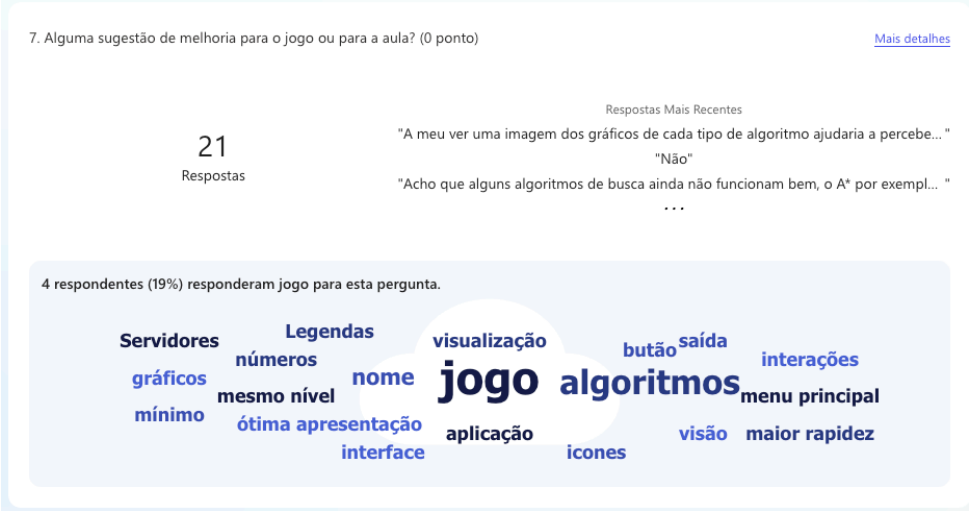


Figura 5.8: Respostas à pergunta 7 do formulário final

Fonte: Autoria própria (2025).

demonstrativo (sem delineamento experimental controlado), as observações qualitativas e as interações foram satisfatórias, verificou-se elevada imersão entre crianças e adolescentes, com rápida apropriação dos controles e curiosidade em testar diferentes estratégias de busca. Tal evidência anedótica é consistente com os fundamentos de UX para jogos educacionais (imersão/atenção, diversão/satisfação) e com o princípio de motivação por *feedback* imediato do modelo ARCS [52], [54]. Para o público universitário, a possibilidade de alternar entre o modo visual e o modo programático (via WebSocket) foi frequentemente destacada como diferencial para conectar teoria, prática e pensamento computacional.

Em síntese, os dados indicam que o Saralk favorece a compreensão declarada dos algoritmos de busca e é bem aceito pelos estudantes, com evidências convergentes entre métricas quantitativas e o racional pedagógico de jogos educacionais e IHC.

Os achados refletem autopercepção em amostra específica e sem delineamento experimental controlado (sem grupo de comparação). Não houve mensuração de desempenho objetivo em prova padronizada.

Capítulo 6

Conclusão

Este trabalho dedicou-se ao desenvolvimento de um *software* interativo, com o objetivo de elucidar e aplicar algoritmos de pesquisa complexos de forma acessível e compreensível. O projeto visou não apenas esclarecer aos estudantes e profissionais iniciantes em Ciência da Computação essas técnicas fundamentais, mas também permitir a exploração e compreensão prática desses métodos através de visualizações interativas e comparações de desempenho.

A metodologia adotada, que combinou o desenvolvimento de uma interface de usuário intuitiva com a implementação de visualizações dinâmicas e ferramentas de análise comparativa, provou ser um sucesso. Este enfoque permitiu aos usuários não apenas entender os princípios teóricos por trás dos algoritmos de pesquisa, mas também visualizar seu funcionamento em tempo real, o que enriqueceu significativamente o processo de aprendizado. Os resultados alcançados, especialmente a capacidade dos usuários de interagir com os algoritmos e observar suas operações passo a passo, destacaram a eficácia da metodologia em tornar conceitos complexos mais acessíveis e compreensíveis.

6.1 Perspectivas futuras

Em síntese, os achados indicam que o Saralk cumpre o objetivo de aproximar teoria e prática em algoritmos de busca, oferecendo um ambiente que combina motivação, visualização transparente e experimentação ativa. As evidências coletadas, ainda que iniciais, somadas à boa aceitação em contexto acadêmico e de extensão, sustentam o potencial do Saralk como ferramenta didática para o ensino de algoritmos, abrindo caminho para estudos mais amplos e para a contínua evolução da plataforma.

Como desdobramentos, propõem-se: (1) estudos controlados com pré/pós-teste objetivo (itens conceituais e procedimentais), grupo de controle e acompanhamento para retenção; (2) expansão do escopo algorítmico para além destes incluídos neste trabalho, algoritmos em grafos ponderados/dinâmicos) e de heurísticas parametrizáveis no A*; (3) mecanismos de acessibilidade ampliada (teclas alternativas, alto contraste, leitores de tela) e internacionalização e (4) avaliações com públicos diversos (educação básica e técnica) para mapear perfis de uso e necessidades específicas.

Por fim, ressalta-se que, mesmo após a conclusão deste trabalho, o software continuará a ser atualizado e mantido periodicamente, de modo a consolidar o Saralk como uma ferramenta estável, acessível e de referência para estudantes e professores.

Bibliografia

[1] M. Rodrigues Jr, «Experiências positivas para o ensino de algoritmos,» *Workshop de Educação em Computação e Informática, Salvador*, 2004.

[2] E. P. Pimentel, V. F. de França, R. V. Noronha e N. Omar, «Avaliação contínua da aprendizagem, das competências e habilidades em programação de computadores,» *Anais do Workshop de Informática na Escola*, vol. 1, n.º 1, pp. 533–544, 2003.

[3] A. A. A. Esmin, «Portugol/Plus: uma ferramenta de apoio ao ensino de lógica de programação baseado no Portugol,» *IV Congresso RIBIE, Brasília*, 1998.

[4] A. J. Mendes, «Software educativo para apoio à aprendizagem de programação,» *Universidade de Coimbra*, 2002.

[5] C. D. Menezes e I. A. M. Nobre, «Um ambiente cooperativo para apoio a cursos de introdução a programação,» *Congresso da Sociedade Brasileira de Computação*, vol. 22, 2002.

[6] J. J. Kaasbøll, «Exploring didactic models for programming,» *NIK 98–Norwegian Computer Science Conference*, pp. 195–203, 1998.

[7] C. Bruce, L. Buckingham, J. Hynd, C. McMahon, M. Roggenkamp e I. Stoodley, «Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university,» *Journal of Information Technology Education: Research*, vol. 3, n.º 1, pp. 145–160, 2004.

[8] R. Lister e J. Leaney, «First year programming: Let all the flowers bloom,» *ACE*, pp. 221–230, 2003.

[9] G. L. White e M. P. Sivitanides, «A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics,» *Journal of information systems education*, vol. 13, n.º 1, p. 59, 2002.

[10] M. R. Garey e D. S. Johnson, *Computers and intractability*. California: Freeman San Francisco, 1979, vol. 174.

[11] A. L. A. Raabe e J. d. Silva, «Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos,» *XIII Workshop de Educação em Computação (WEI'2005)*, vol. 3, n.º 5, 2005.

[12] P. O. B. Netto, *Grafos: teoria, modelos, algoritmos*. São Paulo: Editora Blucher, 2003.

[13] R. Diestel, *Graph theory*. Springer Nature, 2025, vol. 173.

[14] N. Ziviani et al., *Projeto de algoritmos: com implementações em Pascal e C*. São Paulo: Cengage, 2004, vol. 2.

[15] H.-J. Bandelt e A. Dress, «Reconstructing the shape of a tree from observed dissimilarity data,» *Advances in applied mathematics*, vol. 7, n.º 3, pp. 309–343, 1986.

[16] P. E. Black. «Dictionary of Algorithms and Data Structures: root,» National Institute of Standards e Technology (NIST). (), URL: `https://xlinux.nist.gov/dads/HTML/root.html` (acedido em 27/08/2025).

[17] P. E. Black. «Dictionary of Algorithms and Data Structures: tree,» National Institute of Standards e Technology (NIST). (), URL: `https://xlinux.nist.gov/dads/HTML/tree.html` (acedido em 27/08/2025).

[18] P. E. Black. «Dictionary of Algorithms and Data Structures: parent,» National Institute of Standards e Technology (NIST). (), URL: `https://xlinux.nist.gov/dads/HTML/parent.html` (acedido em 27/08/2025).

[19] P. E. Black. «Dictionary of Algorithms and Data Structures: child,» National Institute of Standards e Technology (NIST). (), URL: https://xlinux.nist.gov/dads/HTML/child.html (acedido em 27/08/2025).

[20] P. E. Black. «Dictionary of Algorithms and Data Structures: leaf,» National Institute of Standards e Technology (NIST). (), URL: https://xlinux.nist.gov/dads/HTML/leaf.html (acedido em 27/08/2025).

[21] P. E. Black. «Dictionary of Algorithms and Data Structures: edge,» National Institute of Standards e Technology (NIST). (), URL: https://xlinux.nist.gov/dads/HTML/edge.html (acedido em 27/08/2025).

[22] P. E. Black. «Dictionary of Algorithms and Data Structures: minimax,» National Institute of Standards e Technology (NIST). (), URL: https://xlinux.nist.gov/dads/HTML/minimax.html (acedido em 27/08/2025).

[23] P. E. Black. «Dictionary of Algorithms and Data Structures: height,» National Institute of Standards e Technology (NIST). (), URL: https://xlinux.nist.gov/dads/HTML/height.html (acedido em 27/08/2025).

[24] P. E. Black. «Dictionary of Algorithms and Data Structures: depth,» National Institute of Standards e Technology (NIST). (), URL: https://xlinux.nist.gov/dads/HTML/depth.html (acedido em 27/08/2025).

[25] P. E. Black. «Dictionary of Algorithms and Data Structures: subtree,» National Institute of Standards e Technology (NIST). (), URL: https://xlinux.nist.gov/dads/HTML/subtree.html (acedido em 27/08/2025).

[26] A. C. Gabardo, *Análise de redes sociais: uma visão computacional*. São Paulo: Novatec Editora, 2015.

[27] F. Harary, «The determinant of the adjacency matrix of a graph,» *Siam Review*, vol. 4, n.º 3, pp. 202–210, 1962.

[28] S. Dasgupta, C. Papadimitriou e U. Vazirani, *Algoritmos*. Rio Grande do Sul: AMGH Editora, 2009.

[29] I. Chivers e J. Sleightholme, «An Introduction to Algorithms and the Big O Notation,» em *Introduction to Programming with Fortran: With Coverage of Fortran 90, 95, 2003, 2008 and 77*. Cham: Springer International Publishing, 2015, pp. 359–364, ISBN: 978-3-319-17701-4. DOI: 10.1007/978-3-319-17701-4_23. URL: https://doi.org/10.1007/978-3-319-17701-4_23.

[30] A. Mejia, *8 time complexities that every programmer should know*, https://adrianmejia.com/most-popular-algorithms-time-complexity-every-programmer-should-know-free-online-tutorial-course/, Online; Acessado em 31 de Março de 2024, 2019.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest e C. Stein, *Introduction to algorithms*. Massachusetts: MIT press, 2022.

[32] N. Sultana, S. Paira, S. Chandra e S. S. Alam, «A brief study and analysis of different searching algorithms,» *International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–4, 2017. DOI: 10.1109/ICECCT.2017.8117821.

[33] D. E. Knuth, *The art of computer programming*. Pearson Education, 1997, vol. 3.

[34] S. J. Russell e P. Norvig, *Artificial intelligence: a modern approach*. New York: Pearson, 2016.

[35] A. Prolubnikov, «Finding the connected components of the graph using perturbations of the adjacency matrix,» *arXiv preprint arXiv:2306.16389*, 2023.

[36] E. F. Moore, «The shortest path through a maze,» em *Proc. of the International Symposium on the Theory of Switching*, Harvard University Press, 1959, pp. 285–292.

[37] P. H. R. Molck et al., «Metodos heuristicos em programação de atividades em patios de estocagem,» *Universidade Estadual de Campinas. Campinas*, 2002.

[38] J. Nešetřil, E. Milková e H. Nešetřilová, «Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history,» *Discrete mathematics*, vol. 233, n.º 1-3, pp. 3–36, 2001.

[39] E. W. Dijkstra, «A note on two problems in connexion with graphs,» em *Edsger Wybe Dijkstra: his life, work, and legacy*, 2022, pp. 287–290.

[40] P. E. Hart, N. J. Nilsson e B. Raphael, «A formal basis for the heuristic determination of minimum cost paths,» *IEEE transactions on Systems Science and Cybernetics*, vol. 4, n.º 2, pp. 100–107, 1968.

[41] S. Barbosa e B. Silva, *Interação humano-computador*. Elsevier Brasil, 2010.

[42] A. B. de Sales, M. Serrano e M. Serrano, «Aprendizagem baseada em projetos na disciplina de interação humano-computador,» *Revista Ibérica de Sistemas e Tecnologias de Informação*, n.º 37, pp. 49–64, 2020.

[43] I. Fette e A. Melnikov, «The websocket protocol,» rel. téc., 2011.

[44] S. Fernandes, «Salada de frutas alfabética: jogo educacional para auxiliar a aprendizagem da leitura via método fônico,» B.S. thesis, Universidade Tecnológica Federal do Paraná, 2023.

[45] G. R. d. Carvalho, «A importância dos jogos digitais na educação,» *Universidade Federal FLuminense*, 2018.

[46] R. F. Pereira, P. A. Fusinato e M. C. D. Neves, «Desenvolvendo um jogo de tabuleiro para o ensino de física,» *Anais do VII ENPEC*, pp. 1–12, 2009.

[47] R. Dellos, «Kahoot! A digital game resource for learning,» *International Journal of Instructional technology and distance learning*, vol. 12, n.º 4, pp. 49–52, 2015.

[48] M. Videnovik, T. Vold, L. Kiønig, A. Madevska Bogdanova e V. Trajkovik, «Game-based learning in computer science education: a scoping literature review,» *International Journal of STEM Education*, vol. 10, n.º 1, p. 54, 2023.

[49] G. S. Gomes e D. S. Schmidt, «Videogames sob uma perspectiva bakhtiniana: entrelaçando dialogismo e ludologia,» *Revista Gatilho*, vol. 25, 2023.

[50] R. Gomes, «Narratologia & Ludologia: um novo round,» *VIII Brazilian Symposium on Games and Digital Entertainment Rio de Janeiro*, 2009.

[51] S. N. Gallo, «Ludologia e (m) Videogame,» *Anais do Intercom–XXVI Congresso Brasileiro de Ciências da Comunicação. Belo Horizonte*, 2003.

[52] J. M. Keller, *Motivational design for learning and performance: The ARCS model approach*. New York: Springer Science & Business Media, 2010.

[53] J. Keller e K. Suzuki, «Learner motivation and e-learning design: A multinationally validated process,» *Journal of educational Media*, vol. 29, n.º 3, pp. 229–239, 2004.

[54] R. Savi, C. G. Von Wangenheim, V. Ulbricht e T. Vanzin, «Proposta de um modelo de avaliação de jogos educacionais,» *Revista Novas Tecnologias na Educação*, vol. 8, n.º 3, 2010.

[55] A. L. Carvalho e M. d. S. Nery, «Desenvolvimento de um jogo educacional aplicável a Engenharia de Produção,» *Encontro Nacional de Engenharia de Produção*, vol. 35, 2015.

[56] C. G. von Wangenheim, M. Thiry, D. Kochanski, L. Steil, D. Silva e J. Lino, «Desenvolvimento de um jogo para ensino de medição de software,» *Anais do VIII Simpósio Brasileiro de Qualidade de Software*, pp. 46–60, 2009.

[57] L. F. de Oliveira Melle, J. C. Braga, E. P. Pimentel e S. C. Dotta, «Revisão da Metodologia INTERA e sua Aplicação no Desenvolvimento de um Jogo Educacional do tipo RPG,» *Anais do XXXI Simpósio Brasileiro de Informática na Educação*, pp. 602–611, 2020.

[58] B. Junior, S. Cavalheiro e L. Foss, «A Última Árvore: exercitando o Pensamento Computacional por meio de um jogo educacional baseado em Gramática de Grafos,» *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, vol. 28, n.º 1, p. 735, 2017.

[59] L. Alencar, M. Pessoa e F. Pires, «Um jogo educacional para exercitar propriedades de árvores binárias de busca,» *Anais dos Workshops do IX Congresso Brasileiro de Informática na Educaçao*, pp. 226–231, 2020.

[60] D. Flanagan, *JavaScript: o guia definitivo*. Rio Grande do Sul: Bookman Editora, 2012.

[61] C. Incau, *Vue. js: Construa aplicações incríveis*. São Paulo: Editora Casa do Código, 2017.

[62] L. H. Smith e J. S. Renzulli, «Learning style preferences: A practical approach for classroom teachers,» *Theory into practice*, vol. 23, n.º 1, pp. 44–50, 1984.