JAEB: Avaliando um editor Java acessível para estudantes com deficiência visual em uma disciplina de programação orientada a objetos

Luis Gustavo Araujo[1], **Edinaelson Silva Dos Santos**[1], **Rulian de Jesus Cruz**[1], **Ideilton Alves Freire Leal**[1], **Andressa Mota da Silva Santos**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia da Bahia - Campus Jacobina (IFBA)

{rulian.cruz, luis_araujo, ideiltonleal}@ifba.edu.br,
{edinaelsonsilvadossantos, dhee.andressa}@gmail.com

Abstract. *Learning to program can be a challenging task due to the need for abstraction, logic, and knowledge of the programming language. Historically, programming courses have had high dropout and failure rates. This difficulty can be exacerbated to visually impaired students, given the challenges of using development environments and the limitations of screen readers. This paper presents an accessible development environment for programming in Java, designed based on the needs of blind students at an educational institution. The results indicate the student's familiarity and motivation in using the tool, a positive evaluation of the developed features, and suggestions that emerged during the testing session.*

Resumo. *Aprender a programar pode ser uma tarefa difícil, em virtude da necessidade de abstração, lógica e conhecimento sobre a linguagem de programação. Historicamente, essas disciplinas apresentam altas taxas de evasão e reprovação. Essa dificuldade pode ser potencializada para estudantes com deficiência visual, tendo em vista os desafios no uso de ambientes de desenvolvimento e as limitações dos leitores de tela nesse contexto. Diante disso, este artigo apresenta um ambiente de desenvolvimento acessível para programação na linguagem Java, idealizado a partir das necessidades de estudantes com deficiencia visual de uma instituição de ensino. Os resultados indicam a familiaridade e a motivação do estudante no uso da ferramenta, a avaliação positiva das funcionalidades desenvolvidas e as sugestões surgidas após a sessão de testes.*

1. Introdução

Aprender a programar pode ser uma tarefa difícil para estudantes de Computação. Historicamente, essas disciplinas apresentam altas taxas de reprovação e abandono. Essas dificuldades podem ser potencializadas para estudantes com deficiência visual, tendo em vista o número reduzido de ferramentas acessíveis e as limitações dos leitores de tela (e.g., NVDA, DOSVOX), como, por exemplo, a não leitura de mensagens de erro e a não identificação das linhas e locais de modificação, sendo necessário, em muitos casos, um "olho amigo" para indicar os erros e as correções. Essa limitação torna o estudante com deficiência visual dependente de outras pessoas para auxiliá-lo no processo de aprendizagem em programação [dos Santos et al. 2025].

Durante o curso da disciplina de Linguagem de Programação II, do *Blind Review*, que possui um estudante com deficiência visual matriculado no semestre de 2025.1, foram percebidas as limitações dos leitores de tela na utilização de IDEs em Java (*e.g.*, Visual Studio Code, NetBeans e Eclipse). A partir dos relatos do estudante e das observações dos professores, foram realizadas buscas por ferramentas acessíveis que atendessem às necessidades mapeadas. Embora existam trabalhos visam avaliar o uso de leitores de tela [Gonçalves et al. 2020], o uso de IA como assistente [dos Santos et al. 2025] ou elencar listas de diretrizes [Albusays et al. 2017], percebeu-se que a quantidade de ferramentas acessíveis para programação ainda é limitada.

Esses trabalhos relatam problemas com o leitor de tela quanto a avisos de compilação ou erro, navegação pelo número de linhas ou identificação do local onde deve ser realizada uma modificação no código, identificação de níveis dentro de estruturas aninhadas, leitura de caracteres especiais, suporte à função de autocompletar, busca de entidades específicas no código, entre outros.

Apesar de trabalhos anteriores abordarem este problema, ainda há inúmeros desafios práticos relacionados à acessibilidade de softwares e à adaptação de materiais didáticos acessíveis para pessoas com deficiência visual em ambientes de desenvolvimento integrado (IDEs) [Zen and Tavares 2023]. Assim, o objetivo deste trabalho é apresentar a ferramenta *Java Accessible Editor for Blind* (JAEB), desenvolvida a partir das observações de professores em uma turma de programação, dos relatos de um estudante com deficiência visual e de achados na literatura. Objetiva-se, ainda, iniciar o processo de avaliação e discussão dos resultados encontrados. O JAEB surge como uma iniciativa relevante, alinhada à ação pedagógica inclusiva, ao propor um ambiente acessível que busca romper barreiras enfrentadas por estudantes com deficiência visual no processo de aprendizagem da programação orientada a objetos.

Essa proposta dialoga com os princípios da Lei de Diretrizes e Bases da Educação Nacional (Lei nº 9.394/1996), que prevê, em seu artigo 59, a necessidade de adaptações curriculares e pedagógicas para atender às especificidades dos estudantes público da educação especial, assegurando-lhes condições de aprendizagem equitativas [BRASIL 1996]. O presente trabalho está organizado em sete seções: a segunda apresenta a fundamentação teórica sobre acessibilidade, lei de inclusão e assuntos correlatos; a terceira apresenta os trabalhos relacionados; a metodologia é descrita na quarta seção; a quinta seção apresenta a ferramenta JAEB, sua arquitetura e funcionalidades; os resultados são apresentados na seção seis e, por fim, as considerações finais são discutidas na seção sete.

2. Fundamentação Teórica

A acessibilidade no contexto educacional tem sido cada vez mais debatida no campo da inclusão, especialmente no que se refere ao acesso de estudantes com deficiência visual às interfaces computacionais, aos recursos de Tecnologia Assistiva e às tecnologias digitais da informação e comunicação [Bruno and Nascimento 2019]. Partindo desse pressuposto, a Lei nº 13.146/2015, conhecida como a Lei Brasileira de Inclusão (LBI), assegura que a acessibilidade é um direito constitucional, sendo parte integrante de um conjunto de ações essenciais para o exercício pleno da cidadania por pessoas com deficiência. O conceito de acessibilidade envolve a possibilidade e condição de alcance para utilização

com autonomia de informação e comunicação, inclusive seus sistemas e tecnologias por pessoa com deficiência ou com mobilidade reduzida [BRASIL 2015].

Sendo assim, assegurar acessibilidade implica remover barreiras físicas, comunicacionais, tecnológicas e atitudinais que limitam a participação plena de pessoas com deficiência nos diversos espaços sociais, incluindo os ambientes educacionais mediados por tecnologias digitais [Sassaki 2019]. No campo do ensino de computação para pessoas com deficiência visual, isso significa garantir o acesso e a interação com diferentes tipos de linguagens de programação, bem como o desenvolvimento de tecnologias assistivas que colaborem para o acesso ao conhecimento da área.

Sobre essa questão, a LBI, em seu Art. 78, enfatiza a necessidade de se estimular o desenvolvimento, a pesquisa, a inovação, assim como a difusão de tecnologias voltadas para ampliar o acesso da pessoa com deficiência às tecnologias da informação e comunicação. Com base nesse princípio legal, compreende-se que o presente estudo busca contribuir, ainda que de forma inicial, com a área da educação – em especial no ensino de computação – ao apresentar alternativas que favoreçam a acessibilidade de estudantes com deficiência visual no aprendizado da programação orientada a objetos.

3. Trabalhos Relacionados

O trabalho de Luque e colaboradores destaca que a maioria das ferramentas de modelagem de software não é acessível, dificultando o aprendizado de estudantes com deficiência visual e também sua inserção na indústria. A partir de uma revisão sistemática e de uma experiência prática no ensino de UML – uma linguagem de modelagem visual – para alunos com deficiência visual, foram definidos 14 requisitos funcionais com base nas atividades comuns no ensino de UML e nas necessidades de inclusão desses estudantes em ambientes virtuais de aprendizagem [Luque et al. 2014]. A análise revelou que nenhuma das soluções existentes atende a todos os requisitos identificados. Soluções como AWMo e CCml demonstraram-se promissoras por combinarem interfaces gráficas e textuais, mas ainda são insuficientes para uma inclusão plena e abrangente. Dessa forma, os autores concluem que há uma lacuna no desenvolvimento de ferramentas acessíveis e inclusivas para estudantes com deficiência visual.

Já o trabalho de Albusays e colaboradores relata que Ambientes de Desenvolvimento Integrado (IDEs) têm um papel importante no trabalho de muitos desenvolvedores de software, mas, infelizmente, utilizam muitas informações visuais que são difíceis de serem transmitidas pelos leitores de tela atuais, o que se torna uma barreira para programadores com deficiência visual. Os autores conduziram um estudo exploratório sobre a navegação em código por desenvolvedores com deficiência visual, observando 28 programadores utilizando suas ferramentas de codificação [Albusays et al. 2017]. Neste estudo, os participantes enfrentaram muitas dificuldades de navegação ao usar softwares de codificação com tecnologias assistivas, demonstrando insatisfação com o uso intenso de abstrações visuais, o que compromete a acessibilidade. Esses relatos apontam uma oportunidade para melhorar a acessibilidade de IDEs para desenvolvedores com deficiência visual, especialmente no que tange à navegação em código.

Outro trabalho investigou desafios semelhantes, explorando o uso de ferramentas baseadas em Inteligência Artificial (IA), como o ChatGPT e o GitHub Copilot. A

pesquisa investigou os impactos dessas tecnologias na acessibilidade, produtividade e autonomia de programadores com deficiência visual, com base em uma metodologia prática que comparou o desempenho dos participantes em atividades com e sem o suporte da IA. Os resultados mostraram que, embora as ferramentas de IA tenham potencial para corrigir erros rapidamente e aumentar a confiança dos estudantes, ainda existem barreiras significativas na integração com leitores de tela, dificultando a plena acessibilidade das sugestões geradas automaticamente. Uma melhoria na integração entre leitores de tela e ferramentas de IA pode aumentar o potencial dessa proposta [dos Santos et al. 2025].

Seo e Rogge abordam a lacuna da falta de acessibilidade de ambientes de desenvolvimento integrado (IDEs) para pessoas cegas, na qual um desenvolvedor cego e uma engenheira da equipe do Visual Studio Code (VSCode) trabalharam juntos para melhorar a usabilidade do editor para usuários de leitores de tela. Embora o VSCode já contasse com características que o tornavam mais acessível do que outros IDEs, os autores demonstram que acessibilidade técnica não equivale à usabilidade prática para programadores com deficiência visual. A partir dessa constatação, os autores propõem uma abordagem em que o usuário cego atua não apenas como testador, mas como especialista ativo no processo de desenvolvimento e melhoria do produto [Seo and Rogge 2023].

4. Metodologia

Esta pesquisa parte das limitações encontradas no uso de leitores de tela e editores apropriados para pessoas com deficiência visual em uma disciplina de programação em Java. Sendo assim, configura-se como uma pesquisa-ação, considerando que esse tipo de abordagem envolve projetos nos quais a prática busca transformar as próprias práticas [Tripp 2005]. Diante disso, buscou-se compreender as limitações no processo de ensino-aprendizagem de programação quanto à acessibilidade. A investigação teve como ponto de partida as observações dos professores e os relatos do estudante com deficiência visual na turma de programação II do *Blind Review*. Em seguida, foi realizada uma pesquisa bibliográfica para identificar as ferramentas já existentes e suas limitações.

4.1. Participantes

O estudo contou com a participação de um estudante com deficiência visual, matriculado no curso técnico em Informática do *Blind Review*, na modalidade subsequente. Atualmente, o estudante está cursando a disciplina Lógica de Programação II.

4.2. Pesquisa bibliográfica

Para a pesquisa bibliográfica utilizou-se a seguinte *string*: “ensino de programação” AND “IDE” AND “java” AND (“cegos” OR “deficiência visual”), no motor de busca Google Scholar. Foram encontrados 23 trabalhos publicados entre os anos de 2021 a 2025, sendo que apenas cinco trabalhos foram artigos publicados em revistas ou congressos, destes apenas um focava no estudo do uso de IDEs por pessoas cegas. Diante do número baixo de artigos, utilizou-se a técnica de *Backward Snowballing*, que consiste na busca recursiva de referências presentes nos artigos previamente identificados segundo critérios prévios [Wohlin 2014]. Para seleção, utilizou-se como critério tratar sobre avaliação, uso ou desenvolvimento de ferramentas acessíveis para programação, além de sistematizar funcionalidades necessárias para garantir a acessibilidade.

4.3. Mapeamento das limitações

A etapa seguinte constituiu-se em mapear as limitações relatadas nos artigos revisados. As limitações foram organizadas em uma tabela e agrupadas conforme a similaridade. Estas limitações foram cruzadas com as observações dos professores e relatos dos estudantes por meio de entrevista semi-estruturada, gerando uma lista de requisitos funcionais. Durante a entrevista, foram feitas as seguintes perguntas: i) Quais suas principais dificuldades no editor Java? ii) Quais suas principais dificuldades do uso do editor com o leitor de tela? iii) Sobre mensagem de erro, como isso funciona na tarefa de programar? iv) Sobre a sua navegação no código, como você avalia a ajuda do leitor de tela? v) Você tem alguma recomendação sobre criação de novas funcionalidades para uma IDE acessível?

4.4. Desenvolvimento e Avaliação do MVP

Diante da lista de requisitos, desenvolveu-se um MVP (*Minimum Viable Product*) utilizando tecnologia web, para garantir a interoperabilidade da ferramenta. O MVP foi utilizado por um estudante cego da instituição de ensino em sessões de teste com observação [Creswell and Creswell 2021]. Durante as sessões a ferramenta foi disponibilizada e foi solicitado que o estudante participante desenvolvesse atividades que visavam criar classes, atributos e métodos. Foram avaliados quatro aspectos: i) navegação; ii) comandos/atalhos; iii) narração; iv) mensagem de erro e dicas. Durante os testes, foi solicitado ainda que o estudante utilizasse as funcionalidades disponíveis, como pesquisar e autocompletar. Durante as sessões foram coletados os dados para analise e melhoria da ferramenta.

4.5. Coleta e Análise de Dados

Os dados foram coletados por meio de entrevistas gravadas e transcritas, bem como por observações feitas pelos professores. Inicialmente usou-se uma abordagem exploratória, visando compreender as limitações das ferramentas. Após essa etapa e com base nos achados da literatura, a análise passou a focar nos tópicos mencionados na Subseção 4.4. Os dados foram analisados qualitativamente, com o objetivo de identificar aspectos positivos e negativos que necessitavam de adaptações. Utilizou-se a codificação aberta para a geração livre de códigos e, posteriormente, a codificação axial, baseada nos tópicos elencados. Por fim, foram gerados memorandos com o intuito de aprofundar e sistematizar a análise [Creswell and Creswell 2021]. Os dados permitiram compreender os impactos do uso da ferramenta e as funcionalidades requeridas.

5. JAEB – Java Accessible Editor for Blind

O *Java Accessible Editor for Blind* (JAEB) é um editor Java baseado em tecnologia web, desenvolvido a partir das limitações observadas na prática docente nas disciplinas de programação, bem como daquelas relatadas na literatura. A escolha por um ambiente web deve-se à sua interoperabilidade, considerando que o *Blind Review* possui laboratórios com diferentes sistemas operacionais (Windows, Ubuntu e Zorin). A Figura 1 apresenta uma visão geral da arquitetura do JAEB (blindreview.com).

No frontend, o JAEB utiliza o framework ace.js[1], que viabiliza a implementação do **editor**. O ace.js, por padrão, não possui funcionalidades de acessibilidade, como

[1]Editor ace.js – `https://ace.c9.io`

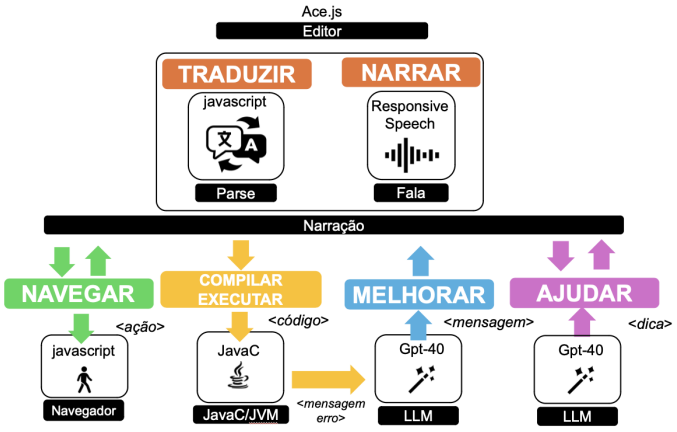


Figura 1. Arquitetura do JAEB

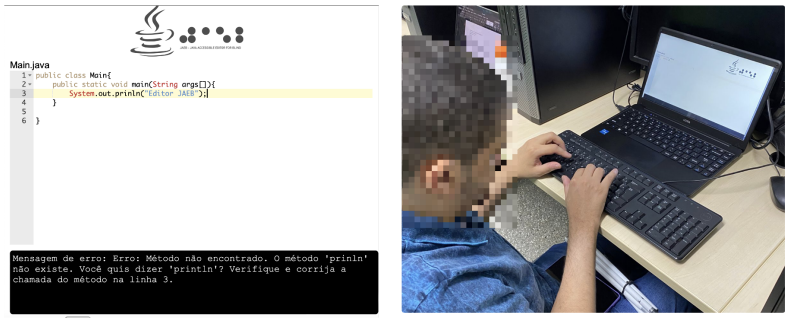
leitura de código e opções de navegação. Considerando que as limitações identificadas estavam relacionadas a dificuldades com softwares leitores de tela, foi implementado um sistema de **narração**, utilizando o framework Responsive Voice[2], que se baseia na API Web Speech. A narração realiza a leitura dos atalhos e ações, do texto do código e da navegação. Considerando uma limitação específica, foi implementado um *parser* que permite a leitura correta de caracteres especiais: ele converte símbolos em seus nomes por extenso, tornando-os legíveis. O *parser* também converte letras maiúsculas para que possam ser corretamente identificadas durante a leitura.

O JAEB **compila** e **executa** o código por meio de chamadas de linha de comando, realizadas no backend desenvolvido em PHP, inspirado na ferramenta PEEF [Araujo et al. 2021]. Em relação à **navegação**, foram implementadas funcionalidades como leitura de código, leitura de linha, indicação de erro, execução, marcação de início e fim de linha, leitura de caracteres na posição atual do cursor, sinalização de nova linha e de caracteres apagados.

Um problema identificado durante as observações e revisão de literatura foi a dificuldade de compreensão das mensagens de erro. Assim, seguindo a abordagem da versão mais recente do PEEF, utilizamos a API da OpenAI, modelo gpt-4o turbo, para gerar mensagens de erro **melhoradas** para o usuário. Essas mensagens são lidas pelo sistema de narração. Para construção do *prompt*, foi utilizada a técnica de *few-shot*, incorporando o código do estudante, a mensagem de erro, o comando e exemplos. O objetivo foi produzir mensagens diretas, simples e com indicação do trecho de código e da linha correspondente.

Além da mensagem aprimorada, foi implementado um sistema de **dicas** de próximo passo, inspirado em trabalho sobre geração de dicas por meio de *Large Language Model* (LLM) [Araujo et al. 2025]. Para esse recurso, o professor deve cadastrar o enunciado da questão e os passos necessários para sua resolução. O *prompt* que solicita a dica inclui o código atual, a descrição da atividade e os passos definidos, solicitando ao LLM a sugestão do próximo passo a ser executado pelo estudante. A tabela do site de apoio[3] apresenta a lista de limitações mapeadas por meio de observações, relatos e revisão

[2]Responsive Voice – `https://responsivevoice.org`
[3]JAEB - https://sites.google.com/view/jaeb-editor/artigoerbase



a) Emissão de mensagem de erro no JAEB

b) Estudante com deficiência visual utilizando o JAEB

Figura 2. Registro da sessão de teste do JAEB. Do lado esquerdo, a interface com código em Java. Do lado direito, um registro do usuário utilizando o JAEB.

da literatura, conforme identificado nos seguintes trabalhos: [Albusays et al. 2017, Gonçalves et al. 2020, dos Santos et al. 2025]. Na tabela, é possível visualizar em qual estudo cada limitação foi relatada, se foi implementada.

A versão atual do JAEB já contempla 14 funcionalidades mapeadas e desenvolvidas, uma em desenvolvimento (ID = 17), uma fora do escopo da linguagem Java (ID = 15) e quatro listadas como trabalhos futuros (IDs = 7, 14,18 e 19). A avaliação foi realizada neste estágio, uma vez que a versão atual pode ser considerada um MVP. A limitação 1 foi resolvida com o uso de um *parser* em JavaScript, que converte os sinais em texto escrito por extenso e adiciona a palavra "maiúscula" após letras maiúsculas, forçando sua leitura pelos leitores de tela. A limitação de ID = 3 foi atendida com a funcionalidade de leitura da linha atual (onde o cursor está posicionado) de forma completa.

Foi implementado um atalho para execução do código que, quando acionado, emite um aviso sonoro indicando o início da execução. As mensagens de erro (IDs = 6, 8 e 10) foram tratadas por meio da alteração da saída padrão, permitindo o envio das informações ao front-end e posterior narração. Toda mensagem de erro ou saída gerada pelo código é lida pelo sistema. As mensagens de erro são aprimoradas com o uso da API da OpenAI, utilizando o mesmo modelo das dicas. Por meio da técnica de *few-shot learning*, que pode ser entendido como a capacidade de generalização de um modelo por meio de uma amostra de exemplos [Parnami and Lee 2022]. Assim, garante-se que o retorno indique de forma clara a linha de modificação necessária, atendendo à limitação de ID 13.

6. Resultados e Discussões

Esta seção relata os resultados quanto à recepção, aspectos positivos e pontos de melhoria.

6.1. Recepção inicial e familiarização

Logo no início da navegação com o JAEB, o estudante manifestou **entusiasmo** ao reconhecer a voz do sistema de narração, afirmando com alegria que se tratava da voz do Google, fato que indica **familiaridade**. Apesar de inicialmente estranhar a nova ferramenta, devido à necessidade de aprender novos comandos, o estudante afirmou posteriormente que ele poderia aprender as novas funcionalidades com o tempo.

Porém, percebe-se que os atalhos similares aos de outras ferramentas potencializam a aprendizagem da ferramenta e a sua adoção. No início, o professor precisou relembrar os comandos; no entanto, com o uso das funcionalidades, o participante assimilou os comandos.

6.2. Funcionalidades bem avaliadas

Diversas funcionalidades foram destacadas positivamente. A leitura de **símbolos especiais**, como chaves, parênteses e ponto e vírgula, foi considerada satisfatória. O estudante também elogiou a capacidade da ferramenta ao tratar símbolos e espaços, mencionando que o NVDA não realiza tais leituras de forma clara. Ao receber uma **mensagem de erro melhorada**, gerada pelo LLM, indicando a ausência de ponto e vírgula ao final da linha, o estudante expressou entusiasmo: "*Gostei porque ele disse o que faz, e disse a linha também. Ele disse: faltou o ponto e vírgula...*" (Participante). O estudante estava acostumado a mensagens de erro padrão, que são, por vezes, enigmáticas. Nesta ocasião, a ferramenta emitiu a seguinte mensagem: *Mensagem de erro: Esperado um ponto e vírgula (';') na linha 3. Por favor, verifique a sintaxe.*

O auxílio à **navegação**, com a indicação de início da linha e final da linha, também foi pontuado pelo estudante como uma boa funcionalidade. Outro aspecto mencionado positivamente é a indicação da **letra maiúscula**, em especial para a linguagem Java, que é *case-sensitive*.

6.3. Sugestões de melhoria adotadas

O participante propôs melhorias relevantes para a navegação e acessibilidade que foram adotadas e analisadas em sessões posteriores. A leitura do caractere apagado, além da mensagem que sinaliza que algo foi apagado, foi adicionada. O participante relatou: "*Mencionar apagado é bom, porque no NVDA não sei o que está apagando, por isso tenho que ter uma pessoa do lado.*". Embora a ferramenta tenha a distinção do minúsculo e maiúsculo, uma funcionalidade sugerida e adotada foi o alerta ao pressionar `Caps lock` para indicar seu estado (Ativado/Desativado). O sistema de navegação foi melhorado com a inclusão da narração para as teclas *Home* (início da linha), *End* (fim da linha) e `Ctrl + Home` (início da primeira linha do programa).

A sessão demonstrou que, embora algumas funcionalidades tenham sido desenvolvidas. Um exemplo foi observado com o atalho `Ctrl + F`, em que o sistema deve indicar por voz: "Procurando". Outro comando que necessitou de narração foi a indicação de "nova linha" e o número da linha ao pressionar `Enter`. Ambas sugestões foram adicionadas.

As funcionalidades foram pensadas com o uso do *Alt* + número, tal como indicou o participante no levantamento dos requisitos. No entanto, após testes, foi solicitada a inclusão de comandos de teclado para facilitar a navegação. Após pressionar `Alt + 1`, o JAEB narra o menu, possibilitando o uso de setas para navegar entre opções. Nota-se que essa forma minimiza a sobrecarga cognitiva para memorizar as opções. Ao mesmo tempo, percebe-se que é uma herança do modo como os leitores de tela leem os menus visuais. Por fim, outra sugestão de melhoria foi na funcionalidade do auto-completar. O estudante solicita que seja lida uma opção por vez e, caso ele pare de digitar, a palavra seja completada. Apenas esta sugestão não foi desenvolvida e testada.

6.4. Impacto da ferramenta e perspectivas futuras

Percebeu-se que a interação com o JAEB provocou no estudante reflexões sobre sua própria trajetória na programação e sobre o potencial da ferramenta para seu futuro. Ele destacou que, com a acessibilidade garantida, se sente mais autônomo e capaz de criar seus próprios projetos: "*Do jeito que está acessível, eu posso até fazer uma coisa legal, que é criar um menu em Java. Talvez eu possa usar para criar Add-Ons.*" (Participante).

Ainda, demonstrou preocupação com o acesso contínuo à ferramenta fora do ambiente institucional: "*Talvez com essa ferramenta eu saiba onde eu esteja. Então, só se você liberar essa ferramenta para o resto da vida.*" (Participante). Foi possível perceber que a ferramenta despertou no estudante motivação, tendo em vista que ele mencionou que iria solicitar o uso do notebook para atividades domiciliares. Até então, embora o notebook estivesse disponível, o estudante não tinha despertado o interesse. Após a sessão, foi relatado pelo coordenador do CAPNE (Coordenação de Atendimento à Pessoa com Necessidades Específicas) que o estudante entrou em contato para relatar a sua experiência com o uso do JAEB, sinalizando o empenho do professor na criação de uma ferramenta acessível, além de demonstrar motivação para continuar o curso.

7. Conclusão

Este trabalho investigou as limitações enfrentadas por estudantes com deficiência visual no uso de editores de programação, a partir de uma análise da literatura e de relatos de estudantes e docentes. A partir das demandas identificadas, foram desenvolvidas e integradas funcionalidades ao editor JAEB, cuja avaliação foi conduzida por meio de sessões de teste com usuários com deficiência visual. Os resultados indicam que a familiaridade com ferramentas previamente utilizadas representa um fator relevante para a adoção de novas soluções tecnológicas. Além disso, observou-se que as funcionalidades de navegação implementadas contribuíram para ampliar a autonomia do estudante nas tarefas de codificação e depuração. Recursos como a leitura de caracteres especiais, espaços e letras maiúsculas mostraram-se fundamentais para a acessibilidade do editor.

Destaca-se ainda a importância das mensagens de erro aprimoradas, que se apresentam como uma funcionalidade promissora no processo de ensino-aprendizagem de programação, embora seu impacto efetivo mereça investigação mais aprofundada. Como trabalhos futuros, propõe-se o desenvolvimento de novas funcionalidades para atender às limitações adicionais identificadas, bem como a realização de testes com um número ampliado de participantes, a fim de validar e refinar as soluções propostas.

Referências

Albusays, K., Ludi, S., and Huenerfauth, M. (2017). Interviews and observation of blind software developers at work to understand code navigation challenges. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 91–100.

Araujo, L. G., Araujo, K., Reis, E., da Silva, M. O., and Pinheiro, A. (2025). Geração de dicas de próximo passo utilizando large language models (llms). In *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, pages 42–43. SBC.

Araujo, L. G. J., Bittencourt, R. A., and Chavez, C. v. F. G. (2021). Python enhanced error feedback: Uma ide online de apoio ao processo de ensino-aprendizagem em

programação. In *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, pages 326–333. SBC.

BRASIL (1996). Lei nº 9.394, de 20 de dezembro de 1996. estabelece as diretrizes e bases da educação nacional. https://www.planalto.gov.br/ccivil_03/leis/l9394.htm. Accessado 09 Jun 2025.

BRASIL (2015). Institui a lei brasileira de inclusão da pessoa com deficiência (estatuto da pessoa com deficiência). http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2015/lei/l13146.htm. Accessado 09 Jun 2025.

Bruno, M. M. G. and Nascimento, R. A. L. d. (2019). Política de acessibilidade: o que dizem as pessoas com deficiência visual. *Educação e Realidade*, 44(1):e84848.

Creswell, J. W. and Creswell, J. D. (2021). *Projeto de pesquisa-: Métodos qualitativo, quantitativo e misto*. Penso Editora.

dos Santos, N. S., de Oliveira Santana, D., and Pereira, C. P. (2025). Inteligência artificial e acessibilidade: Uma experiência de inclusão para programadores cegos em ambientes de desenvolvimento. In *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, pages 502–515. SBC.

Gonçalves, R. S., Santana, R. S., Neto, F. A., Benevides, S. C., and dos Santos, N. S. (2020). Análise dos desafios para programar sem enxergar: estudo de caso na disciplina linguagem de programação 1. In *Simpósio Brasileiro de Sistemas de Informação (SBSI)*, pages 17–20. SBC.

Luque, L., Brandão, L. O., Tori, R., and Brandão, A. A. (2014). Are you seeing this? what is available and how can we include blind students in virtual uml learning activities. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 25, page 204.

Parnami, A. and Lee, M. (2022). Learning from few examples: A summary of approaches to few-shot learning. *arXiv preprint arXiv:2203.04291*.

Sassaki, R. K. (2019). As sete dimensões da acessibilidade. *São Paulo: Lavratus Prodeo*.

Seo, J. and Rogge, M. (2023). Coding non-visually in visual studio code: collaboration towards accessible development environment for blind programmers. In *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 1–9.

Tripp, D. (2005). Pesquisa-ação: uma introdução metodológica. *Educação e pesquisa*, 31:443–466.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10.

Zen, E. and Tavares, T. A. (2023). Estratégias de acessibilidade em ides para estudantes com deficiência visual. In *Congresso Brasileiro de Informática na Educação (CBIE)*, pages 223–228. SBC.