PBPC
ISSN 2674-9432
Qualis A3 CAPES 2021-2024
DOI - Crossref
Latindex
Indexado no Google Acadêmico

LOGIC RUSH: UM JOGO EDUCACIONAL 2D COM APRENDIZAGEM ADAPTATIVA ASSISTIDA POR INTELIGÊNCIA ARTIFICIAL PARA O ENSINO DE LÓGICA DE PROGRAMAÇÃO

Deivid dos Santos Brito, Alfredo Nazareno Pereira Boente, Ricardo Marciano dos Santos, Vinícius Marques da Silva Ferreira, Eduardo Luis Pareto, José Mauro Baptista Bianchi, Miguel Gabriel Prazeres de Carvalho, George Coelho Kleinau Vasconcelos, Antonio Jorge Borges dos Santos, Juan Gabriel Pires Boente

https://doi.org/10.36557/2674-9432.2026v5n5p473-493
Artigo recebido em 13 de Maio e publicado em 13 de Julho de 2026

ARTIGO ORIGINAL

RESUMO

Este artigo apresenta o desenvolvimento e a avaliação do Logic Rush, um jogo digital educacional 2D com motor adaptativo inteligente voltado ao ensino de lógica de programação assistido por Inteligência Artificial. Desenvolvido na plataforma Godot 4 com GDScript, o sistema evoluiu de uma progressão estática para um Ambiente de Aprendizagem Adaptativo, capaz de gerar questões dinamicamente por meio da Gemini API, classificá-las por tema e nível de dificuldade, recomendar conteúdos com base no desempenho do jogador e ajustar automaticamente a complexidade dos desafios por meio de um algoritmo de Dificuldade Adaptativa Dinâmica (DDA). A proposta integra mecânicas de corrida automática, obstáculos, *buffs*, sistema de vidas, chefes e *quizzes* de múltipla escolha com interpretação de código e previsão de saída, distribuídos em três fases progressivas. A pesquisa, de natureza aplicada e exploratória, compreendeu o desenvolvimento do protótipo e sua avaliação inicial por meio de escala Likert e métricas de uso. Os resultados demonstram a viabilidade técnica do sistema e evidenciam percepções positivas quanto à usabilidade, ao engajamento e ao potencial de aprendizagem entre estudantes do primeiro semestre de Análise e Desenvolvimento de Sistemas sob a disciplina de lógica de programação.

Palavras-chave: Jogos Educacionais, Gamificação, Inteligência Artificial, Aprendizagem Adaptativa, Lógica de Programação.

LOGIC RUSH: A 2D EDUCATIONAL GAME WITH ARTIFICIAL INTELLIGENCE ASSISTED ADAPTIVE LEARNING FOR TEACHING PROGRAMMING LOGIC

ABSTRACT

This article presents the development and evaluation of Logic Rush, a 2D educational digital game featuring an intelligent adaptive engine designed to support the teaching of programming logic through Artificial Intelligence. Developed using the Godot 4 engine and GDScript, the system evolved from a static progression model into an Adaptive Learning Environment capable of dynamically generating questions through the Gemini API, classifying them according to topic and difficulty level, recommending learning content based on player performance, and automatically adjusting challenge complexity using a Dynamic Difficulty Adjustment (DDA) algorithm. The proposed solution integrates auto-run gameplay mechanics, obstacles, buffs, a life system, boss battles, and multiple-choice quizzes involving code interpretation and output prediction, distributed across three progressive stages. The applied and exploratory research encompassed both the prototype development and its initial evaluation using a Likert scale and usage metrics. The results demonstrate the technical feasibility of the system and reveal positive perceptions regarding usability, engagement, and learning potential among first-semester students enrolled in an Analysis and Systems Development program taking the programming logic course.

Keywords: Educational Games, Gamification, Artificial Intelligence, Adaptive Learning, Programming Logic.

Instituição afiliada – FACULDADE DE EDUCAÇÃO TECNOLÓGICA DO ESTADO DO RIO DE JANEIRO (FAETERJ-Rio).
Autores correspondentes*:*
Deivid dos Santos Brito
Https://orcid.org/0009-0005-4765-3601
E-mail: deivid.cote@gmail.com
Graduando em Análise e Desenvolvimento de Sistemas

Alfredo Nazareno Pereira Boente
https://orcid.org/0000-0002-2718-4917
E-mail: boente@nce.ufrj.br
Doutor em Ciências da Engenharia de Produção

Ricardo Marciano dos Santos
https://orcid.org/0000-0002-9031-1608
E-mail: rms221070@gmail.com
Doutor em História das Ciências e das Técnicas e Epistemologia

Vinícius Marques da Silva Ferreira
https://orcid.org/0000-0003-3664-3510
E-mail: profvmarques@gmail.com
Doutor em Ciências da Engenharia de Produção

Eduardo Luis Pareto
https://orcid.org/0009-0001-9854-6663
E-mail: epareto@hcte.ufrj.br
Doutorando em História das Ciências e das Técnicas e Epistemologia

José Mauro Baptista Bianchi
https://orcid.org/0009-0007-2689-411X
E-mail: bianchi@hcte.ufrj.br
Doutorando em História das Ciências e das Técnicas e Epistemologia

Miguel Gabriel Prazeres de Carvalho
https://orcid.org/0009-0000-6691-9990
E-mail: mgpc10@gmail.com
Mestre em Informática

George Coelho Kleinau Vasconcelos
https://orcid.org/0009-0004-3301-502X
E-mail: gvasconcelos@hcte.ufrj.br
Mestrando em História das Ciências e das Técnicas e Epistemologia

Antonio Jorge Borges dos Santos
https://orcid.org/0009-0008-7584-6218
E-mail: asantos@hcte.ufrj.br
Mestrando em História das Ciências e das Técnicas e Epistemologia

Juan Gabriel Pires Boente
https://orcid.org/0009-0001-8722-6306
E-mail: juangabrielpires@gmail.com
Graduado em Tecnologia em Marketing Digital

1 INTRODUÇÃO

O aprendizado inicial de programação é um dos principais desafios enfrentados por estudantes ingressantes em cursos da área de computação. Conceitos como variáveis, estruturas condicionais, laços de repetição, interpretação de código e resolução de problemas exigem não apenas memorização de sintaxe, mas também o desenvolvimento do raciocínio lógico e da capacidade de abstração. Para muitos estudantes, esse processo pode se tornar difícil quando apresentado apenas por meio de aulas expositivas, exercícios tradicionais ou exemplos descontextualizados.

Diante desse cenário, diferentes estratégias têm sido investigadas com o objetivo de tornar o aprendizado de programação mais acessível, interativo e motivador. Entre essas estratégias, destacam-se os jogos digitais educacionais, os jogos sérios e a gamificação. Os jogos sérios são projetados com propósitos que vão além do entretenimento, podendo ser utilizados em contextos de ensino, treinamento e desenvolvimento de habilidades. No campo da programação, Miljanovic e Bradbury revisaram a literatura sobre jogos sérios voltados ao ensino de programação, analisando conteúdos educacionais e avaliações relacionadas a 49 jogos, com foco em fundamentos de programação, leitura e escrita de programas e aspectos de avaliação dessas propostas (MILJANOVIC; BRADBURY, 2018).

A gamificação, por sua vez, está relacionada ao uso de elementos característicos de jogos em contextos que não são necessariamente jogos completos, como sistemas de pontuação, níveis, recompensas, rankings, desafios e *feedbacks*. No ensino de lógica de programação, Castro e Santos realizaram uma revisão sistemática sobre o uso da gamificação em cursos superiores de computação, buscando compreender como ferramentas gamificadas têm sido aplicadas nessas disciplinas, quais aspectos pedagógicos são utilizados e quais métodos de avaliação aparecem nos estudos analisados (CASTRO; SANTOS, 2023).

Nesse contexto, este artigo apresenta o desenvolvimento do Logic Rush, um jogo digital educacional 2D assistido por Inteligência Artificial voltado ao apoio no aprendizado introdutório de lógica de programação. O jogo combina mecânicas de corrida automática, obstáculos, buffs, sistema de vida, chefes e perguntas de *quiz* em formato de múltipla escolha, previsão de saída e debugging com interpretação de código. Diferentemente de um jogo infinito, o Logic Rush possui estrutura finita, composta por três fases, possuindo um “Chefe” por fase, com progressão gradual de dificuldade. O jogador avança pelas fases, enfrenta desafios e responde perguntas para derrotar os “Chefes”, recebendo *feedback* imediato a partir de seus acertos e erros.

Considerando esse cenário, este trabalho parte da seguinte questão de pesquisa: De que maneira um jogo digital educacional 2D, assistido por Inteligência Artificial, baseado em mecânicas de corrida e desafios de *quiz*, pode apoiar a prática de conceitos introdutórios de lógica de programação? A partir dessa questão, propõe-se o desenvolvimento do Logic Rush, um protótipo de jogo educacional voltado ao aprendizado introdutório de lógica de programação.

O objetivo geral deste artigo é desenvolver e analisar um protótipo de jogo digital educacional 2D que integre mecânicas de corrida, progressão por fases e desafios de múltipla escolha relacionados à lógica de programação. Para alcançar esse objetivo, o

trabalho envolve o estudo de conceitos relacionados a jogos educacionais, jogos sérios, gamificação e aprendizado de programação; o planejamento das mecânicas principais do jogo; a implementação do protótipo no Godot 4 com GDScript; a organização de um banco de perguntas por níveis de dificuldade; e a realização de uma avaliação exploratória com usuários, considerando aspectos de usabilidade, engajamento, percepção de aprendizagem e análise dos erros por categoria de conteúdo.

A justificativa para o desenvolvimento do Logic Rush está na possibilidade de utilizar o jogo como uma ferramenta complementar para a prática de conteúdos iniciais de programação. Ao integrar perguntas de lógica a uma experiência interativa, com fases, desafios, *feedback* imediato e acompanhamento de desempenho, busca-se oferecer ao estudante uma forma mais dinâmica de revisar conceitos básicos, sem substituir as práticas tradicionais de ensino.

2 METODOLOGIA

Este trabalho caracteriza-se como uma pesquisa aplicada, de caráter exploratório, com abordagem qualitativa e quantitativa. A pesquisa é aplicada por propor o desenvolvimento de um protótipo de jogo digital educacional voltado a uma necessidade prática: apoiar o aprendizado introdutório de lógica de programação. É exploratória por analisar, em uma versão inicial, o potencial do protótipo quanto à usabilidade, ao engajamento e à percepção de aprendizagem dos usuários.

O desenvolvimento do protótipo ocorreu a partir da definição das mecânicas principais do jogo, da implementação no motor Godot 4, da elaboração do banco de perguntas e da organização de uma avaliação exploratória com usuários. O jogo foi implementado em GDScript e estruturado em cenas e scripts responsáveis pelo menu, jogador, obstáculos, *buffs*, *bosses*, sistema de *quiz*, pontuação e salvamento local.

O Logic Rush foi projetado como um jogo 2D de corrida automática, com progressão finita em três fases, assistido por Inteligência Artificial. Durante a corrida, o jogador deve desviar de obstáculos posicionados no chão ou no alto e coletar *buffs* que podem recuperar vida, aumentar temporariamente a velocidade ou conceder escudo. Ao atingir a distância definida para cada fase, a corrida é interrompida e o jogador enfrenta um “Chefe” por meio de perguntas de múltipla escolha sobre lógica de programação.

As perguntas foram organizadas em arquivos separados por nível de dificuldade, permitindo associar cada fase a um conjunto de questões compatível com sua progressão, com assistência de algoritmos de Inteligência Artificial. O conteúdo das perguntas contempla conceitos básicos de lógica de programação, interpretação de trechos de código e previsão de saída. Optou-se por utilizar questões objetivas de múltipla escolha, em vez de entrada livre de código, para manter o escopo técnico do protótipo viável e evitar a necessidade de integração com compiladores, interpretadores ou mecanismos complexos de validação automática.

A avaliação proposta possui caráter exploratório e será aplicada com um grupo reduzido de usuários. Após utilizar o protótipo, os participantes responderão a um questionário voltado a aspectos como facilidade de uso, clareza dos controles, compreensão das perguntas, interesse pelo jogo, percepção de aprendizagem e

sugestões de melhoria. Além do questionário, serão observados dados gerados durante a experiência, como pontuação, fase máxima alcançada, número de respostas corretas, número de respostas incorretas e precisão.

Também será realizada a análise dos erros por categoria de conteúdo, com o objetivo de identificar quais temas apresentaram maior dificuldade para os participantes. As categorias poderão incluir variáveis, estruturas condicionais, laços de repetição, operadores lógicos, interpretação de código e previsão de saída. Essa análise poderá ser representada por meio de gráficos, contribuindo para a discussão dos resultados e para o aprimoramento futuro do banco de perguntas.

3 RESULTADOS e DISCUSSÃO

3.1 Implementação do Protótipo

O ponto de entrada do Logic Rush é o menu principal, que funciona como um hub central de navegação entre os módulos do jogo, conforme ilustra a Figuira 1. A partir dessa tela, o jogador pode iniciar uma nova sessão, consultar a interface de tutorial, acessar o placar de pontuações local e visualizar o sistema de conquistas, além de identificar de forma imediata se o lote diário de questões geradas pela Adaptive Learning System já se encontra disponível em cache. Essa centralização reduz a fragmentação da experiência de usuário e antecipa, ainda antes do início da corrida, os elementos de gamificação e suporte que serão detalhados nas subseções seguintes.

Figura 1 - Menu principal do Logic Rush.

Fonte: Autores, 2026.

Como resultado direto do desenvolvimento, obteve-se o protótipo funcional do Logic Rush, construído no motor Godot 4 utilizando a linguagem GDScript. O sistema principal foi estruturado em torno da mecânica de corrida automática 2D (*auto-runner*), segmentado em três fases finitas e com dificuldade progressiva.

O ambiente de jogo foi programado para instanciar dinamicamente obstáculos e *buffs* (cura, velocidade e escudo). A progressão de cada fase é atrelada à distância percorrida; ao atingir a métrica pré-definida, a mecânica de corrida é interrompida, dando lugar ao sistema de quiz.

Neste sistema, instanciado visualmente como uma batalha contra um "Chefe", o jogador interage com perguntas objetivas divididas em três níveis de dificuldade, vinculadas diretamente à fase em curso. O registro contínuo do desempenho, incluindo pontuação máxima, fase alcançada, acertos e erros, foi implementado por meio de um sistema de salvamento local (*save data*), compondo o ranking individual do jogador.

A principal contribuição técnica deste trabalho reside na implementação do Motor Adaptativo Inteligente, estruturado em dois scripts principais. O script desafios_gemini.gd atua como camada de dados e persistência: gerencia o estado local das questões (incluindo os campos *category*, *difficulty* e *answered*), normaliza as respostas recebidas da Gemini API para o formato interno do jogo e executa a função precisa_mais_perguntas, que verifica continuamente se o pool de questões não respondidas do nível atual caiu abaixo de um limite de segurança (cinco questões válidas). Quando esse limiar é atingido, a função sincronizar_nivel é disparada assincronamente em segundo plano, solicitando novas questões à API antes que o jogador alcance o "Chefe" da fase. Esse mecanismo de recarga assíncrona garante a fluidez da experiência do usuário mesmo em condições de conexão intermitente, reduzindo chamadas de API desnecessárias e mantendo um cache local válido por até três dias.

O script game.gd atua como controlador do ciclo de jogo e é responsável pelos dois algoritmos centrais do sistema adaptativo. O Algoritmo de Recomendação Pedagógica opera por heurística de baixo aproveitamento: ao selecionar a próxima questão, o motor prioriza:

(1) categorias ainda não respondidas pelo jogador, para introdução de conteúdo inédito;

(2) categorias com a menor taxa de acerto atual, para reforço das fragilidades;

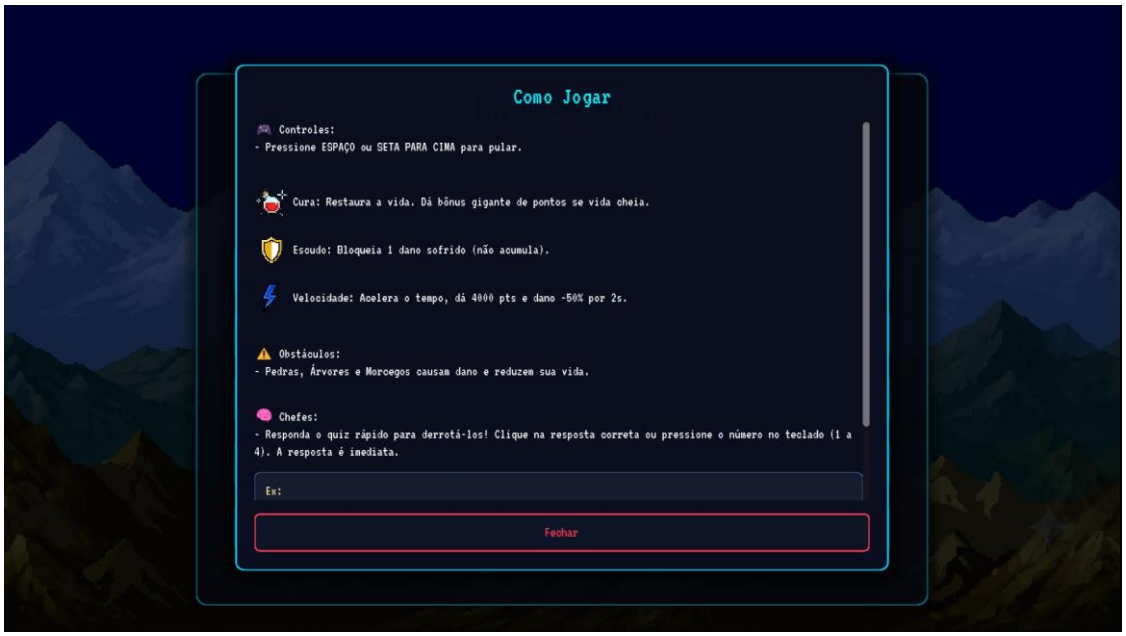
(3) como critério de desempate, questões com sub-dificuldade mais próxima do estado atual do DDA.

O Algoritmo de Dificuldade Adaptativa Dinâmica (DDA) inicia o quiz na sub-dificuldade 1 (Fácil) da fase em curso. Dois acertos consecutivos promovem o nível (+1) e zeram o contador de combo; qualquer erro regride imediatamente o nível (-1), com limite mínimo de 1. As variáveis de estado de dificuldade, categoria e taxa de acerto são registradas em logs analíticos de depuração pedagógica (*Educational Data Mining*) e permanecem ocultas da interface do jogador, mitigando a pressão psicológica associada à visibilidade do desempenho em tempo real.

Em paralelo ao motor adaptativo, o protótipo recebeu uma camada de refinamento de usabilidade voltada à autonomia do jogador. Foi implementada uma interface modular de tutorial ("Como Jogar"), acessível a partir do menu principal e sobreposta ao jogo sem interromper o estado da sessão em andamento. Essa tela concentra, de forma centralizada, a descrição dos controles, a função de cada *buff* (Cura, Escudo e

Velocidade), a natureza dos obstáculos que causam dano e o funcionamento do *quiz* dos chefes. Tecnicamente, o componente foi construído sobre um contêiner de rolagem (*scroll container*) independente do fluxo principal do jogo, permitindo que o conteúdo instrucional seja consultado sob demanda a qualquer momento. Trata-se de uma decisão puramente de design de usabilidade: ao externalizar as regras do jogo para uma camada de consulta opcional, reduz-se a sobrecarga cognitiva inicial do jogador, sem impor a leitura obrigatória de instruções antes do início da experiência propriamente dita, conforme ilustra a Figura 2.

Figura 2 - Tela de tutorial “Como Jogar”.



Fonte: Autores, 2026.

Complementarmente, foi implementado um sistema de conquistas (*achievements*) instanciado dinamicamente ao final de cada sessão de jogo. O script responsável monitora, ao longo da partida, eventos-gatilho específicos, como sofrer *Game Over* com um *buff* determinado ativo, ou concluir a jornada completa e, ao término da execução, verifica quais condições foram satisfeitas. Quando uma condição é atendida, o sistema instancia programaticamente um componente de notificação não bloqueante (*toast*) sobreposto às telas de conclusão de fim de jogo e de vitória, exibindo o ícone, o título e a descrição textual da conquista desbloqueada.

O estado de progresso do jogador, isto é, o registro de quais conquistas já foram obtidas, é persistido localmente em conjunto com os demais dados de *save*, de modo que uma conquista já desbloqueada não seja notificada novamente em sessões subsequentes, preservando o caráter de novidade do reforço.

3.2 Análise das Escolhas de Design

Para o Logis Rush foi adotada uma arquitetura híbrida de persistência totalmente apoiada por otimizações de chamadas de API, proporcionando uma melhoria significativa a experiência do jogador.

Uma das principais contribuições de engenharia de software implementadas no protótipo funcional foi a estruturação de um *cache* global de renovação diária baseado no *Firebase Firestore*.

O servidor intermediário *Express* (desenvolvido em TypeScript) atua como um *proxy* de segurança que encapsula as chaves de API do Gemini e gerencia a inteligência de persistência sob o seguinte fluxo de execução:

(1) **Requisição de Perguntas** (/api/generate-questions): Quando o cliente Godot solicita um novo pacote de questões para um determinado nível, o servidor *Express* gera uma chave identificadora única no *Firestore* baseada no nível e na data atual (ex: 1_2026-06-29).

(2) **Verificação de Cache Compartilhado** (*Firestore Hit*): O servidor realiza uma busca síncrona no Firestore para verificar se aquele identificador já existe. Se outro estudante em qualquer lugar do mundo já jogou esse mesmo nível no dia atual, o pacote de perguntas é retornado imediatamente a partir do banco de dados global.

(3) ***Fallback* e Geração Dinâmica** (*Gemini Miss*): Caso a chave não seja encontrada (indicando que trata-se do primeiro acesso do dia a esse nível), o servidor faz a chamada assíncrona ao modelo Gemini, normaliza e estrutura o JSON das novas questões, e grava esse novo pacote no *Firestore* antes de retorná-lo ao jogador.

(4) **Sincronização e *Fallback* Local**: O cliente Godot recebe o pacote normalizado e o salva localmente (desafios_cache.json). Se o servidor estiver indisponível ou *offline*, o cliente utiliza de forma transparente seu histórico local ou arquivos de perguntas nativos (*offline fallback*).

A arquitetura do Logic Rush materializa conceitos discutidos na literatura de *Game-Based Learning*. Diferente de abordagens tradicionais que utilizam plataformas de codificação estáticas, a adoção do gênero *runner* garante um nível basal de engajamento através da ação contínua (VIDENOVIK et al., 2023). A pausa na corrida durante o enfrentamento do “Chefe” é uma mudança intencional de ritmo.

No *game design*, a corrida representa o progresso contínuo, enquanto o “Chefe” é o clímax do desafio. Pedagogicamente, essa parada transforma o “Chefe” em um 'portal de conhecimento' (*knowledge gate*). Ela sinaliza ao jogador que a exigência de reflexos rápidos dará lugar a uma avaliação de lógica, onde ele testa seus conhecimentos acerca da disciplina de lógica de programação, na qual avançar de fase passa a depender da sua compreensão do código.

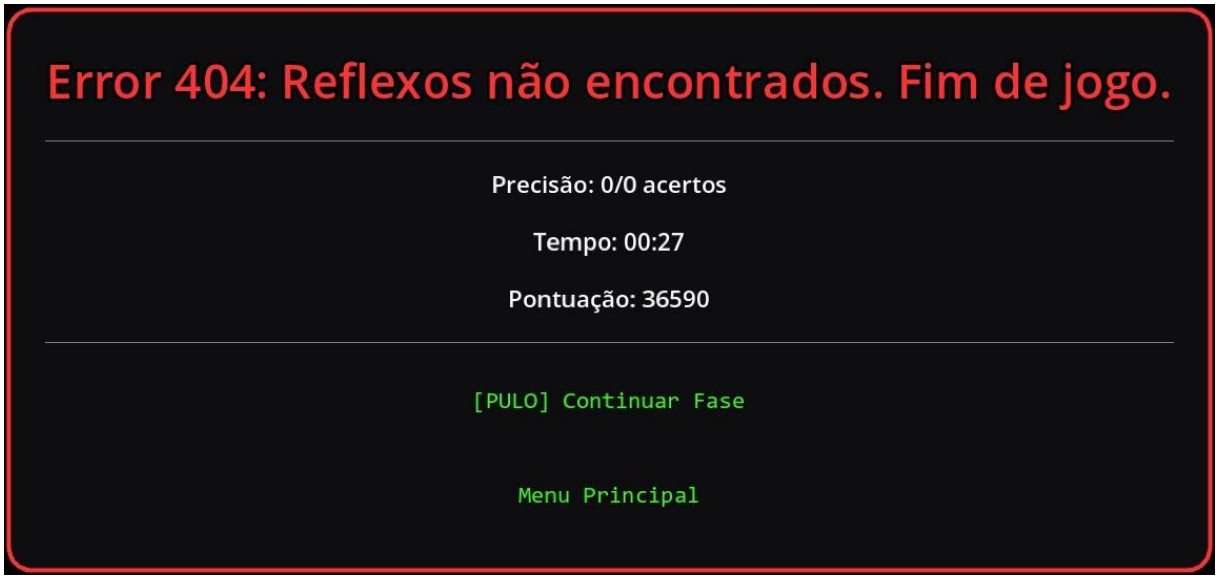
Além disso, o sistema de *buffs* e punições implementado reflete as diretrizes de Mao et al. (2024) sobre o papel do *feedback* imediato. O erro no *quiz* penaliza a vida do avatar instantaneamente, sem encerrar o processo de aprendizagem de forma abrupta, enquanto o acerto causa dano ao “Chefe”, gamificando o sentimento de autoeficácia e progresso. Essa dinâmica de micro-recompensas converge com os achados de Zhao et al. (2022), evidenciando que a associação entre mecânicas ágeis e

avaliação contínua pode mitigar a desmotivação frequentemente associada ao aprendizado introdutório de lógica.

Para operacionalizar essa tensão produtiva e manter o engajamento, as mecânicas de punição e tratamento de falhas foram refinadas. Durante o embate contra os “Chefes”, a penalidade por uma resposta incorreta consistia em um recuo fixo de 100 metros. Para garantir que essa métrica se mantivesse relevante e proporcional em fases mais avançadas, que possuem extensões totais maiores, implementou-se um recuo dinâmico calculado a uma taxa de 15% do tamanho do estágio atual. De forma complementar, para os cenários de falha total onde o personagem perde toda a vida (*Game Over*), foi desenvolvido o recurso de "Continuar Fase Atual". Essa mecânica anti-frustração restaura o HP do jogador e descarta apenas a pontuação e distância do nível vigente, dispensando a necessidade de reiniciar o jogo desde o Nível 1. Isso impede explorações (*score farming*), mas garante que o custo da falha seja balanceado, preservando a sensação de progresso da jornada.

A Figura 3 ilustra a tela que indica o final do jogo, dando opção ao usuário jogador continuar na fase atual.

Figura 3 - Tela de fim de jogo, com opção de continuar fase atual.



Fonte: Autores, 2026.

Essas recompensas foram materializadas na *engine* por meio de micro-interações de resposta imediata, essenciais para o reforço positivo. Ao acertar uma questão de lógica, o sistema instancia programaticamente emissões de partículas douradas e um breve clarão translúcido na tela (*screen flash*), fornecendo responsividade instantânea antes mesmo da conclusão das animações de ataque do personagem, conforme ilustrado na Figura 4.

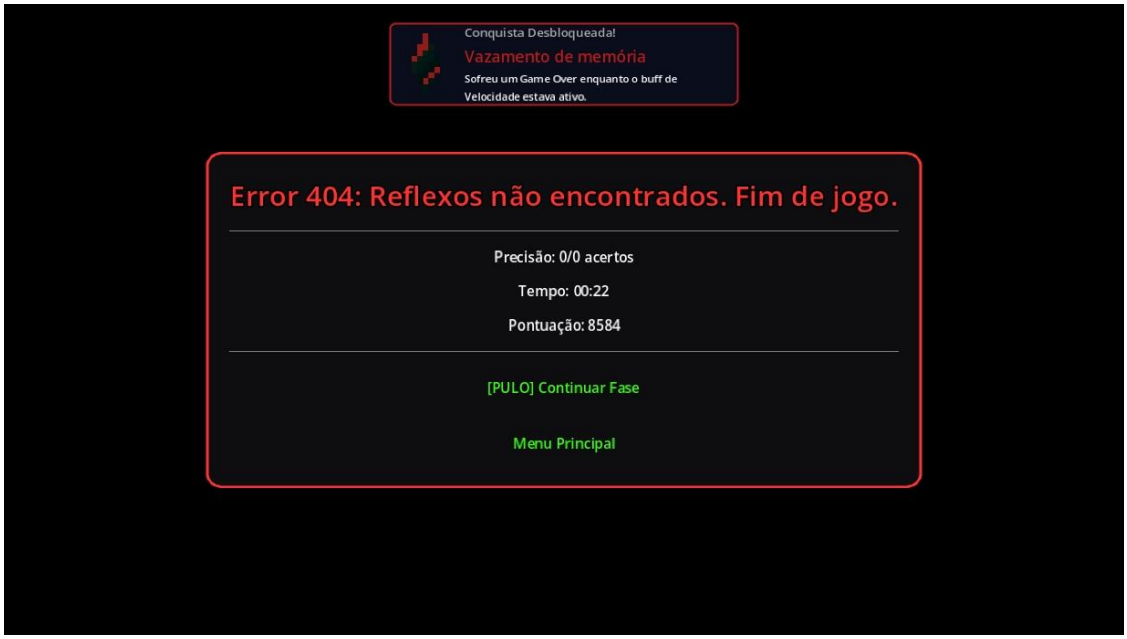
Figura 4 - Tela de fim de jogo, com opção de continuar fase atual.

Fonte: Autores, 2026.

Por outro lado, o tratamento estético da derrota (*Game Over*) foi aprimorado para suavizar a transição abrupta da morte do avatar. Desenvolveu-se um algoritmo de captura da textura da *Viewport* atual e interpolação paralela que simula o esmagamento horizontal e vertical característico de um monitor CRT sendo desligado. Essa transição isola visualmente a subsequente tela vermelha de erro de sistema, conferindo polimento nostálgico e minimizando a quebra de imersão no momento da falha crítica.

É precisamente nesse instante de vulnerabilidade emocional que o sistema de conquistas descrito na fundamentação teórica (Seção 1.3) é acionado: a notificação de uma conquista como "Vazamento de memória" surge sobreposta à tela de erro, redirecionando a atenção do jogador de um evento puramente negativo (a derrota) para um evento de reconhecimento lúdico. Esse deslocamento de foco opera como uma ressignificação da falha, na qual o fracasso deixa de ser apenas uma penalidade e passa a ser também uma oportunidade narrativa, atenuando a carga de frustração associada ao *Game Over*, conforme ilustra a Figura 5.

Figura 5 - Notificação de conquista sobreposta à tela de Game Over.



Fonte: Autores, 2026.

O mesmo mecanismo de notificação de conquistas é reaproveitado na tela de Vitória, exibida ao término bem-sucedido da jornada completa do jogo. Nesse contexto, porém, a função pedagógica do toast se inverte: em vez de ressignificar uma falha, ele atua consolidando a sensação de competência do jogador, reforçando publicamente, ainda que apenas para si mesmo, a conclusão de um percurso de aprendizagem completo. Essa dualidade de propósito (mitigar a frustração na derrota e reforçar a competência na vitória) ilustra como um mesmo componente de interface pode ser reaproveitado com funções psicológicas distintas, a depender do contexto em que é instanciado, conforme ilustra a Figura 6.

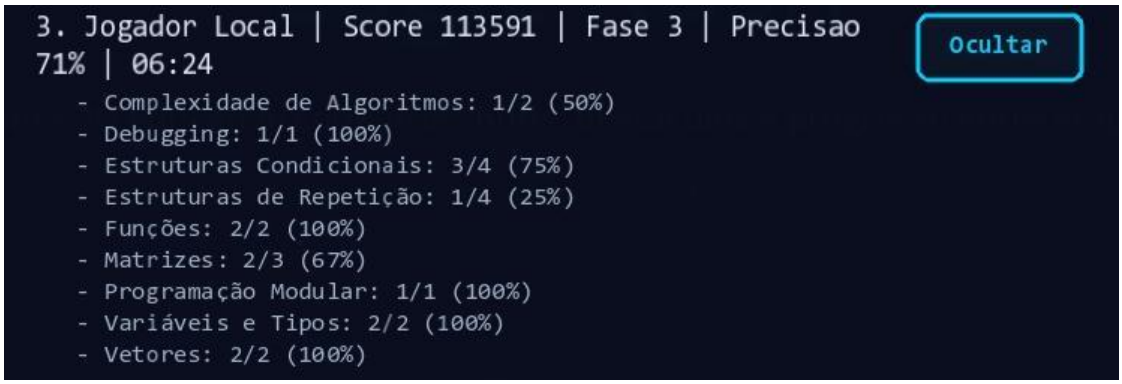
A transição do modelo estático, em que as questões eram selecionadas de pools locais fixos sem consideração do histórico do jogador, para o modelo dinâmico adaptativo representa uma evolução qualitativa no diagnóstico pedagógico possível pelo jogo.

Figura 6 - Notificação de conquista sobreposta à tela de Vitória.

Fonte: Autores, 2026.

No modelo anterior, a única informação disponível era a pontuação global e a fase máxima alcançada. Já no modelo atual, o sistema registra continuamente o desempenho por categoria e sub-dificuldade, permitindo identificar, por exemplo, que um estudante tem alta taxa de acerto em Variáveis mas apresenta dificuldade persistente em Laços de Repetição, informação que seria invisível em uma avaliação de pontuação agregada. Essa granularidade de dados se alinha ao campo do *Educational Data Mining* (EDM), que propõe o uso de dados gerados em ambientes de aprendizagem digital para apoiar decisões pedagógicas e identificar padrões de dificuldade em escala. A ilustração da Figura 7 mostra a pontuação detalhada do jogador.

Figura 7 - Pontuação detalhada.



Fonte: Autores, 2026.

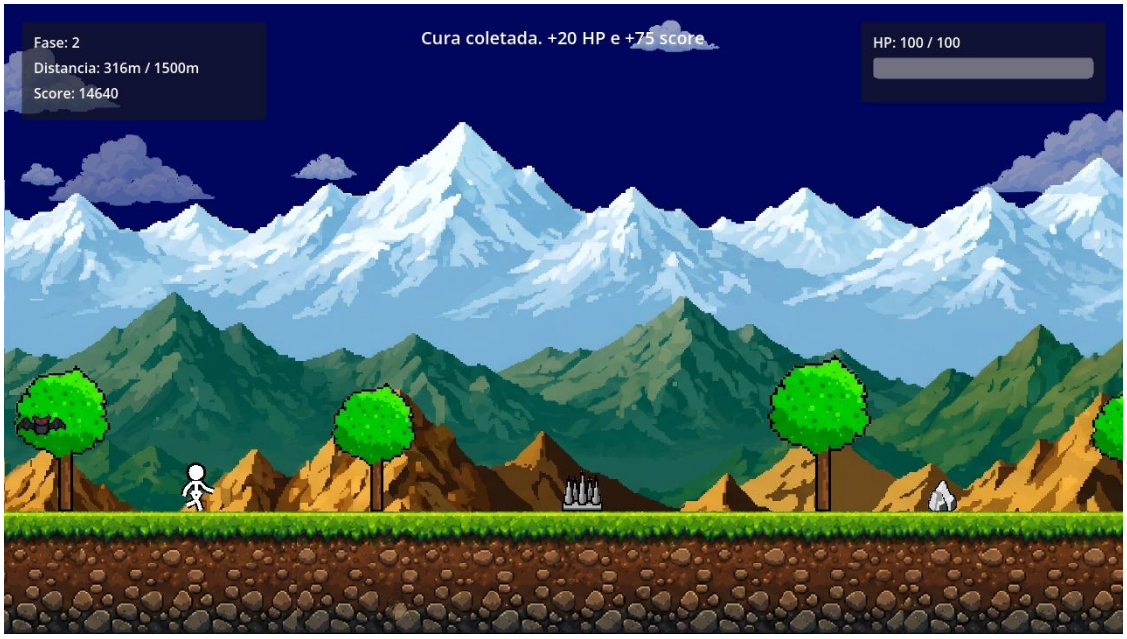
Essa evolução qualitativa no diagnóstico pedagógico foi ampliada pelo aprimoramento da granularidade do rastreamento de dados (*Educational Data Mining*). O sistema passou a quantificar de forma exata a precisão cognitiva do jogador (taxa de acertos e

erros por estágio) e integrou um temporizador ao ciclo principal do jogo.

Como uma decisão deliberada de design, o cronômetro opera de forma oculta (*background*) durante a corrida e a resolução das questões, sendo exibido ao jogador exclusivamente nas telas de conclusão (vitória, falha e placar final). A não exibição do relógio em tempo real evita a sobrecarga de estresse visual e a ansiedade de desempenho durante a interpretação de código, característica comum em alunos iniciantes. Simultaneamente, a coleta desse tempo de conclusão permite que o sistema analise retrospectivamente não apenas a assertividade da resposta, mas a fluência e o tempo de deliberação do discente.

A Figura 8, ilustra as informações completas disponibilizadas na tela do jogo, fase, distância, *score*, cura coletada e *plus* de *score*, que visam auxiliar o jogador durante a sua experiência.

Figura 8 - Tela do Jogo com Informações Complementares.



Fonte: Autores, 2026.

Uma escolha de design deliberada nessa evolução merece destaque: os dados internos de categoria e sub-dificuldade são intencionalmente ocultados da interface do jogador. Essa decisão fundamenta-se em evidências da literatura sobre ansiedade de desempenho em ambientes de avaliação: a visibilidade explícita do nível de dificuldade do estudante pode gerar pressão psicológica que compromete o engajamento, especialmente em iniciantes. Ao manter esses dados no plano dos *logs* analíticos, acessíveis ao docente e ao pesquisador, mas transparentes para o jogador, o Logic Rush preserva a ludicidade da experiência enquanto coleta dados de valor diagnóstico. Esse dualismo entre experiência do jogador e dados do pesquisador é uma característica que diferencia o Logic Rush de plataformas de exercícios gamificados convencionais, que frequentemente exibem métricas de desempenho de forma explícita e potencialmente intimidante.

A geração dinâmica de questões via Gemini API também implica uma discussão sobre

a validade do conteúdo. Para mitigar o risco de questões com erros conceituais geradas eventualmente pela Inteligência Artificial, o sistema foi projetado para que o ementário fornecido ao modelo seja configurável pelo docente, e as questões geradas passem por normalização estrutural antes de serem incluídas no pool ativo. Essa arquitetura posiciona o Logic Rush como uma ferramenta de apoio ao docente, não como um substituto autônomo da avaliação formal, o que é coerente com o escopo declarado do trabalho: uma ferramenta complementar de revisão e prática, não de substituição do processo de ensino tradicional.

A transição do modelo estático e local para o sistema de cache centralizado em nuvem gerou repercussões importantes para a viabilidade do jogo no ambiente escolar:

(1) **Redução de Custo de Operação da IA**: Ao compartilhar as perguntas geradas entre todos os alunos do mesmo nível no mesmo dia, o custo operacional de consultas à API do Gemini foi reduzido de forma exponencial. Se uma turma de 40 alunos acessar o jogo, em vez de 40 chamadas completas de geração (com dezenas de *tokens* de entrada e saída por lote), o sistema consome apenas 1 chamada do Gemini, e os outros 39 acessos ocorrem via leitura rápida no *Firestore*.

(2) **Tempo de Resposta** (Latência): O tempo de geração de um pacote de 15 perguntas via LLM pode variar de 3 a 7 segundos devido ao processamento semântico do modelo. O tempo de leitura de um documento estruturado no *Firestore* é tipicamente inferior a 150 milissegundos. Portanto, o cache global garante uma experiência de usuário fluida e sem travamentos para a ampla maioria dos jogadores.

(3) **Segurança de API e Credenciais**: Toda a lógica de comunicação com o *Firebase* e com o Gemini ocorre estritamente do lado do servidor (*backend proxy*). Isso impede que as credenciais do *Firebase* e as chaves privadas de API sejam expostas ou sujeitas a engenharia reversa a partir do código do cliente compilado em Godot GDScript.

3.3 Posicionamento do Logic Rush em relação a trabalhos semelhantes

A literatura sobre jogos educacionais para programação apresenta uma variedade de abordagens, cada uma com características e focos distintos. Ambientes baseados em blocos visuais, como os utilizados em plataformas de ensino de programação para crianças, reduzem a barreira sintática, mas tendem a se distanciar das linguagens textuais utilizadas no ensino superior. Plataformas web gamificadas, como as que utilizam sistemas de pontos e *rankings* sobre exercícios de código, aproximam-se mais do contexto acadêmico, porém costumam carecer de uma narrativa de jogo que sustente o engajamento em sessões mais longas. *Serious games* tradicionais, por sua vez, frequentemente exigem escrita e execução ativa de código, o que eleva a complexidade técnica de desenvolvimento e pode limitar o público-alvo. Por outro lado, ambientes colaborativos, embora promissores para o desenvolvimento de competências sociais, não se adequam necessariamente à prática individual e autodirigida de conceitos introdutórios.

O Logic Rush ocupa um espaço distinto nesse panorama. Ao adotar o gênero *runner* 2D como estrutura principal, o jogo oferece engajamento baseado em ação contínua sem exigir codificação ativa. A progressão por fases com “Chefes” funciona como estrutura avaliativa integrada à narrativa do jogo, diferentemente de plataformas que

separam o conteúdo educacional da mecânica principal.

No jogo, os *quizzes* de múltipla escolha, previsão de saída e *debugging* com interpretação de código, envolvendo interpretação de trechos de código e previsão de saída, promovem uma forma ativa de processamento conceitual que vai além da simples memorização de sintaxe. Esse posicionamento torna o Logic Rush especialmente adequado como ferramenta complementar de revisão e prática em disciplinas introdutórias, ao oferecer uma experiência lúdica estruturada em torno dos conteúdos curriculares típicos dessas disciplinas.

3.4 Avaliação com usuários

A avaliação do protótipo foi conduzida de forma exploratória, visando identificar indícios de usabilidade, engajamento e percepção de aprendizagem. A amostra contou com a participação de 9 voluntários, estudantes de cursos de computação, que testaram uma versão preliminar do jogo (prévia à implementação dos tutoriais e dos refinamentos de UI/UX).

A metodologia utilizou um questionário estruturado com questões em escala Likert e perguntas abertas para coleta de percepções qualitativas sobre a experiência, desses estudantes, como se fossem usuários jogadores.

Os resultados indicaram uma recepção favorável por parte da maioria dos participantes (aproximadamente 2/3 da amostra), que destacaram o potencial do gênero *runner* como suporte ao ensino de lógica. Contudo, a análise qualitativa revelou um desafio pedagógico fundamental: 1/3 dos usuários relatou a memorização de questões devido à limitação do banco de dados estático da versão inicial. Este *insight* foi o catalisador para a evolução técnica do projeto, servindo como base empírica para a integração da Inteligência Artificial generativa (Gemini API).

Além disso, os *feedbacks* coletados apontaram a necessidade de melhorias na interface do usuário (UI) e na explicitação das mecânicas através de tutoriais, pontos que foram priorizados no ciclo de desenvolvimento subsequente.

Por fim, embora realizada com uma amostra reduzida e em estágio inicial, a avaliação forneceu evidências cruciais que validaram a necessidade de transição para um modelo adaptativo, demonstrando a importância do processo iterativo de design na mitigação de problemas de usabilidade e na otimização da experiência de aprendizagem.

4 CONCLUSÃO

Este trabalho apresentou o desenvolvimento e a avaliação do Logic Rush, um jogo digital educacional bidimensional concebido para apoiar o ensino de lógica de programação por meio de mecânicas de gamificação associadas a técnicas de Inteligência Artificial e aprendizagem adaptativa. A proposta evoluiu de um protótipo baseado em progressão estática para um Ambiente de Aprendizagem Adaptativo (*Adaptive Learning Environment*), capaz de personalizar a experiência do estudante a partir da análise contínua de seu desempenho e da geração dinâmica de conteúdos

educacionais.

Os resultados obtidos demonstram que os objetivos estabelecidos para esta pesquisa foram plenamente alcançados. Foi desenvolvido um sistema funcional utilizando o motor gráfico Godot 4 e a linguagem GDScript, incorporando algoritmos de Dificuldade Adaptativa Dinâmica (*Dynamic Difficulty Adjustment* - DDA), recomendação pedagógica baseada em desempenho, geração automática de questões por meio da Gemini API, armazenamento inteligente utilizando *Firebase Firestore* e mecanismos de cache compartilhado que tornam o sistema escalável e economicamente viável para utilização em ambientes educacionais com grande número de estudantes.

Sob a perspectiva pedagógica, verificou-se que a combinação entre desafios lúdicos e avaliações contínuas cria um ambiente capaz de estimular a participação ativa do estudante durante o processo de aprendizagem. Diferentemente de abordagens tradicionais centradas exclusivamente em listas de exercícios ou avaliações periódicas, o Logic Rush integra entretenimento, feedback imediato e acompanhamento individualizado do desempenho, favorecendo a prática constante dos conteúdos relacionados à lógica de programação. Essa integração contribui para reduzir a percepção de dificuldade frequentemente associada às disciplinas introdutórias da computação e amplia o potencial de engajamento dos estudantes.

Outro aspecto relevante evidenciado neste trabalho refere-se ao uso da Inteligência Artificial como elemento de apoio ao processo educativo. A utilização da Gemini API permitiu automatizar a geração de questões inéditas, organizadas por categorias e níveis de dificuldade, reduzindo significativamente o problema da repetição de exercícios identificado na avaliação preliminar do protótipo. Paralelamente, o algoritmo de recomendação pedagógica passou a direcionar o estudante para conteúdos nos quais apresenta maior índice de dificuldade, promovendo uma experiência de aprendizagem personalizada e alinhada aos princípios contemporâneos dos Sistemas Tutores Inteligentes e dos Ambientes Adaptativos de Aprendizagem.

Do ponto de vista computacional, o projeto também apresenta contribuições relevantes. A arquitetura híbrida baseada em cache local e cache compartilhado em nuvem mostrou-se eficiente para minimizar chamadas repetitivas aos modelos de Inteligência Artificial, reduzindo custos operacionais e melhorando o tempo de resposta do sistema. Além disso, a separação entre cliente e servidor preserva a segurança das credenciais das APIs utilizadas, permitindo que o jogo seja distribuído sem exposição de informações sensíveis.

Os resultados obtidos durante a avaliação exploratória também forneceram informações importantes para o aprimoramento do projeto. Embora realizada com uma amostra reduzida de participantes, a pesquisa evidenciou elevada aceitação quanto à proposta do jogo, indicando percepções positivas relacionadas à usabilidade, ao potencial de aprendizagem e ao nível de motivação proporcionado pela experiência gamificada. Da mesma forma, os comentários qualitativos dos participantes permitiram identificar limitações da versão inicial, especialmente relacionadas ao tamanho do banco de questões, aos elementos de interface e à necessidade de tutoriais mais detalhados. Essas observações foram incorporadas ao processo de evolução do sistema, demonstrando a importância do desenvolvimento iterativo em projetos de jogos educacionais.

Outro diferencial observado refere-se ao conjunto de dados pedagógicos produzidos automaticamente durante a interação do jogador. O registro contínuo de acertos, erros, tempo de resposta, categorias de maior dificuldade e evolução do desempenho cria oportunidades para aplicação de técnicas de *Educational Data Mining* e *Learning Analytics*, permitindo que professores obtenham indicadores mais precisos sobre as dificuldades individuais e coletivas da turma. Dessa forma, o Logic Rush não se limita a funcionar como uma ferramenta de prática, mas também pode atuar como instrumento de apoio ao planejamento pedagógico e ao acompanhamento do processo de aprendizagem.

Apesar dos resultados promissores, esta pesquisa apresenta algumas limitações. A avaliação realizada possui caráter exploratório e foi conduzida com um número reduzido de participantes, não sendo suficiente para estabelecer conclusões estatisticamente generalizáveis sobre os impactos do jogo no desempenho acadêmico. Além disso, o sistema ainda trabalha predominantemente com questões objetivas de múltipla escolha, interpretação de código e previsão de saída, não contemplando, nesta versão, a escrita e execução automática de programas pelos estudantes. Da mesma forma, a qualidade das questões geradas pela Inteligência Artificial depende da adequada configuração dos prompts e da supervisão docente, tornando indispensável a validação humana dos conteúdos produzidos.

Como perspectivas para trabalhos futuros, pretende-se ampliar significativamente o escopo da plataforma. Entre as possibilidades destacam-se a implementação de programação prática com compilação automática de código, integração com Ambientes Virtuais de Aprendizagem (AVA), utilização de modelos de Inteligência Artificial especializados em educação, expansão do sistema de recomendações pedagógicas, implementação de dashboards analíticos para professores, adoção de técnicas mais sofisticadas de mineração de dados educacionais e realização de estudos longitudinais envolvendo diferentes instituições de ensino e um número significativamente maior de participantes. Também se pretende investigar o impacto do Logic Rush sobre indicadores objetivos de aprendizagem, retenção de conteúdo, redução da evasão e desenvolvimento do pensamento computacional.

Por fim, conclui-se que o Logic Rush representa uma contribuição relevante para a pesquisa em jogos educacionais, gamificação e aprendizagem adaptativa aplicada ao ensino de programação. Ao integrar mecânicas de jogos digitais, Inteligência Artificial generativa, adaptação dinâmica de dificuldade e análise contínua do desempenho do estudante, o sistema demonstra que é possível construir ambientes de aprendizagem mais personalizados, motivadores e tecnologicamente sustentáveis. Mais do que uma ferramenta complementar para revisão de conteúdos, o Logic Rush evidencia o potencial das tecnologias inteligentes para transformar a forma como estudantes aprendem lógica de programação, oferecendo novas possibilidades para docentes, pesquisadores e instituições interessadas na inovação dos processos de ensino e aprendizagem na área da Computação.

5 REFERÊNCIAS

CASTRO, Maria Beatriz de Oliveira; SANTOS, Viviane Almeida dos. Gamificação como recurso para aprimorar o ensino de lógica de programação em cursos de computação no ensino superior: uma revisão sistemática. Revista Novas Tecnologias na Educação, v. 21, n. 2, 2023.

CSIKSZENTMIHALYI, Mihaly. Flow: The Psychology of Optimal Experience. New York: Harper & Row, 1990.

KAZIMOGLU, Cagin et al. A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. Procedia – Social and Behavioral Sciences, v. 47, p. 1991–1999, 2012. DOI: 10.1016/j.sbspro.2012.06.938.

MAO, Wei et al. The effects of immediate feedback in game-based learning environments on student motivation, engagement, and learning outcomes: A systematic review. Computers & Education, 2024.

MILJANOVIC, Michael A.; BRADBURY, Jeremy S. A Review of Serious Games for Programming. In: Joint International Conference on Serious Games. Springer, 2018. p. 204-216.

VIDENOVIK, Maja et al. Computer science education through game-based learning: A systematic literature review. Education and Information Technologies, 2023. DOI: 10.1007/s10639-022-11449-4.

VYGOTSKY, Lev S. Mind in Society: The Development of Higher Psychological Processes. Cambridge: Harvard University Press, 1978.

ZHAN, Zehui et al. The effectiveness of gamification in programming education: Evidence from a meta-analysis. Computers and Education: Artificial Intelligence, v. 3, 2022, 100096. DOI: 10.1016/j.caeai.2022.100096.

ZHAO, Wanli et al. Game-based learning in programming education: A systematic

review. Education and Information Technologies, 2022.

ZOHAIB, Mohammad. Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review. Advances in Human-Computer Interaction, v. 2018, 2018. DOI: 10.1155/2018/5681652.