

## Investigando a Geração de Desafios Semelhantes e Exemplos Resolvidos em Jogos de Programação

Alan de Oliveira Santana<sup>1</sup>, Thiago Reis da Silva<sup>2</sup>, Eduardo Henrique da Silva Aranha<sup>1</sup>

<sup>1</sup>Departamento de Informática e Matemática Aplicada (DIMAp)  
Universidade Federal do Rio Grande do Norte – UFRN

<sup>2</sup>Instituto Federal de Educação, Ciência e Tecnologia do Maranhão (IFMA)  
alandeoliveirasantana@gmail.com, thiago.reis@ifma.edu.br, eduardoaranha@dimap.ufrn.br

**Abstract.** *Programming games engage students in playful challenges that require computational thinking and programming-related skills. Procedural Content Generation (PCG) enables the automatic creation of game levels and scenarios while also supporting instructional features, such as worked examples and the generation of similar challenges after incorrect answers. This paper investigates these two features through an online controlled experiment involving 51 participants using a  $2 \times 2$  factorial design. Engagement and performance metrics were analyzed across the experimental treatments, and the lessons learned from the study are discussed. Although the observed trends suggest potential benefits under some conditions, statistical tests found no significant differences at  $\alpha = 5\%$ . We also discuss methodological factors that may have limited the detection of the investigated effects.*

**Resumo.** *Jogos de programação engajam estudantes em desafios lúdicos que demandam raciocínio computacional e habilidades relacionadas à programação. A Geração Procedural de Conteúdo (Procedural Content Generation – PCG) possibilita a criação automática de níveis e cenários, além de apoiar funcionalidades pedagógicas, como a apresentação de exemplos resolvidos e a geração de desafios semelhantes após erros dos estudantes. Este artigo investiga essas duas funcionalidades por meio de um experimento controlado online com 51 participantes, utilizando um delineamento fatorial 2x2. Foram analisadas métricas de engajamento e desempenho entre os diferentes tratamentos, bem como as lições aprendidas durante o estudo. Embora as tendências observadas indiquem potenciais benefícios em algumas condições, os testes estatísticos não identificaram diferenças significativas para  $\alpha = 5\%$ . Também são discutidos fatores metodológicos que podem ter limitado a detecção dos efeitos investigados.*

### 1. Introdução

Jogos de programação têm sido amplamente utilizados como uma estratégia para apoiar o desenvolvimento do pensamento computacional e da aprendizagem de programação, especialmente em contextos introdutórios. Esses ambientes buscam combinar desafios lúdicos com atividades de resolução de problemas, permitindo que estudantes pratiquem conceitos computacionais de forma mais motivadora e interativa

[Lindberg et al. 2018, Falkner et al. 2019, Mannila et al. 2014]. Entre os diversos formatos existentes, destacam-se os jogos baseados em desafios e quebra-cabeças, nos quais os participantes devem aplicar conceitos de programação para atingir objetivos específicos.

Compreender e aprender programação, entretanto, permanece um desafio para muitos estudantes. Conceitos abstratos, a necessidade de coordenar múltiplos elementos simultaneamente e a resolução estruturada de problemas podem impor elevada carga cognitiva, especialmente em disciplinas introdutórias. Além disso, a aprendizagem de programação costuma demandar prática constante e feedback oportuno para que os estudantes consigam identificar erros, compreender estratégias de resolução e consolidar novos conhecimentos [Sweller 2011, Margulieux et al. 2020b, McLaren and Isotani 2011a]. Esses desafios motivam a investigação de mecanismos de apoio instrucional capazes de tornar a aprendizagem mais eficiente sem comprometer a autonomia dos estudantes.

Com a crescente adoção desses ambientes, surge a necessidade de investigar estratégias capazes de apoiar o processo de aprendizagem e reduzir dificuldades encontradas pelos estudantes. Entre essas estratégias estão os exemplos resolvidos (*Worked Examples*), amplamente utilizados como mecanismo de apoio instrucional, e a Geração Procedural de Conteúdo (*Procedural Content Generation* – PCG), que possibilita a criação automática de desafios, níveis e cenários de jogo [Hafis et al. 2019, Margulieux et al. 2020a]. Em teoria, ambas as abordagens podem contribuir para a aprendizagem: exemplos resolvidos podem auxiliar estudantes durante a resolução de problemas, enquanto a PCG pode gerar novas oportunidades de prática adaptadas ao contexto do jogador.

Apesar desse potencial, a avaliação empírica dessas estratégias em jogos de programação apresenta desafios metodológicos importantes. Características como o nível de conhecimento prévio dos participantes, a dificuldade das tarefas propostas e a forma como os problemas são apresentados podem influenciar significativamente os resultados observados. Em determinados contextos, esses fatores podem reduzir a necessidade de suporte instrucional, dificultando a identificação dos efeitos produzidos pelas intervenções avaliadas.

Neste contexto, este trabalho investiga duas funcionalidades apoiadas por PCG em jogos de programação: (i) a geração de desafios semelhantes após respostas incorretas e (ii) o fornecimento de exemplos resolvidos como mecanismo de apoio ao estudante. Para isso, foi conduzido um experimento controlado online com 51 participantes, utilizando um delineamento fatorial 2x2.

Embora os resultados não tenham apontado diferenças estatisticamente significativas, a análise identificou fatores experimentais que podem obscurecer os efeitos de intervenções pedagógicas. Assim, este trabalho contribui para o entendimento metodológico sobre mecanismos de apoio instrucional, oferecendo subsídios para o planejamento de futuros experimentos com estratégias adaptativas em jogos educacionais.

## **2. Trabalhos Relacionados**

### **2.1. Jogos de Programação**

Os jogos de programação têm sido amplamente utilizados para apoiar o desenvolvimento do pensamento computacional e da programação introdutória, como evidenci-

ado pela popularização de plataformas como Code.org e pelo levantamento apresentado por Lindberg et al. [Lindberg et al. 2018]. Esses ambientes procuram combinar desafios computacionais com elementos lúdicos para aumentar o engajamento dos estudantes [Silva et al. 2024].

Estudos anteriores investigaram preferências dos usuários [Harms et al. 2016], impactos nas atitudes em relação à programação [Lambić et al. 2020] e diferentes estratégias de suporte instrucional [Zhi et al. 2018]. Em conjunto, esses estudos indicam que fatores como diversão, desafio, valor percebido e mecanismos de apoio influenciam diretamente a experiência e o desempenho dos participantes.

Mais recentemente, pesquisas passaram a explorar mecanismos de apoio à aprendizagem, incluindo textos explicativos, perguntas, exemplos resolvidos e códigos defeituosos (*buggy code*) [Zhi et al. 2018]. Em particular, Zhi et al. observaram que esses mecanismos podem reduzir o tempo necessário para resolver desafios e favorecer a aprendizagem, motivando investigações sobre sua efetividade em diferentes contextos educacionais.

## 2.2. Geração Procedural de Conteúdo (PCG)

A Geração Procedural de Conteúdo (*Procedural Content Generation* – PCG) refere-se ao uso de procedimentos computacionais para gerar conteúdo automaticamente durante a execução de um sistema. Neste trabalho, a PCG foi empregada para gerar dinamicamente desafios e exemplos resolvidos.

Na área de jogos digitais, a PCG é amplamente utilizada para criar cenários, níveis, desafios e quebra-cabeças, aumentando a diversidade das experiências e reduzindo o esforço de desenvolvimento [Sandhu and McCoy 2019, De Kegel and Haahr 2020].

Em contextos educacionais, a PCG também possibilita a geração de atividades conceitualmente equivalentes, porém estruturalmente distintas, ampliando oportunidades de prática e apoiando experiências adaptativas [Silva and D’Emery 2024]. Essas características tornam essa abordagem especialmente adequada para jogos educacionais voltados ao ensino de programação.

## 2.3. Exemplos Resolvidos

Exemplos Resolvidos (*Worked Examples* – WE) são recursos instrucionais que apresentam soluções completas para problemas, permitindo que estudantes observem estratégias de resolução antes de realizar atividades semelhantes [McLaren and Isotani 2011b]. Na Educação em Computação, esses recursos têm sido empregados para apoiar a aprendizagem de programação e a resolução de problemas relacionados [Rahman and Du Boulay 2010].

Em jogos de programação, exemplos corretos e exemplos contendo erros foram utilizados como mecanismos de apoio instrucional [Zhi et al. 2018, Muldner et al. 2022]. Os resultados indicam que esses recursos podem reduzir o tempo necessário para resolver desafios, embora seus efeitos sobre a aprendizagem nem sempre sejam estatisticamente significativos [Zhi et al. 2018]. Esse comportamento é frequentemente explicado pela Teoria da Carga Cognitiva, segundo a qual exemplos resolvidos reduzem a carga cognitiva extrínseca durante a resolução inicial dos problemas [Sweller 2011].

Estudos recentes investigam a geração automática de exemplos resolvidos utilizando modelos de linguagem [Jury et al. 2024, Hassany et al. 2024]. Além disso, sistemas tutores inteligentes e ambientes adaptativos exploram sua personalização de acordo com as características dos estudantes [Santana and Aranha 2019, Silva et al. 2019, Lui et al. 2008]. Apesar desses avanços, ainda são limitadas as evidências sobre o uso combinado de exemplos resolvidos e PCG em jogos adaptativos de programação, lacuna que este trabalho busca investigar.

### 3. Metodologia

Esta seção descreve o delineamento experimental adotado, seguindo recomendações para condução e relato de experimentos em Computação [Ko et al. 2015, Jedlitschka et al. 2008].

#### 3.1. Questões de Pesquisa

Este estudo investiga o uso de Geração Procedural de Conteúdo (PCG) para a criação de desafios semelhantes e exemplos resolvidos em situações nas quais estudantes apresentam dúvidas ou respostas incorretas durante a interação com um jogo de programação. Para isso, foram definidas as seguintes questões de pesquisa:

**RQ1:** Qual o impacto da geração de desafios semelhantes após respostas incorretas em jogos de programação?

**RQ2:** Qual o impacto do uso de exemplos resolvidos como mecanismo de apoio à resolução de desafios em jogos de programação?

As respostas para essas questões são investigadas por meio da análise do comportamento dos estudantes ao interagir com diferentes versões do jogo, contendo ou não os recursos gerados por PCG.

#### 3.2. Delineamento Experimental

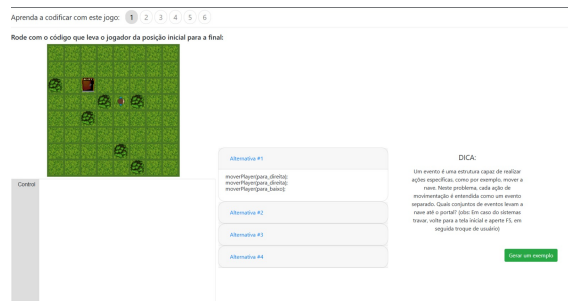
O experimento investiga dois fatores: (i) a geração de desafios semelhantes após respostas incorretas (Gen) e (ii) a disponibilização de exemplos resolvidos (*Worked Examples* – WE). Dessa forma, foi adotado um delineamento fatorial  $2 \times 2$  [Montgomery 2020], no qual cada fator possui dois níveis (presente ou ausente).

A combinação desses fatores resulta em quatro tratamentos distintos. O tratamento **g0w0** corresponde à versão base do jogo, sem geração de desafios semelhantes e sem exemplos resolvidos. O tratamento **g0w1** disponibiliza apenas exemplos resolvidos aos participantes. O tratamento **g1w0** disponibiliza apenas a geração de desafios semelhantes após respostas incorretas. Por fim, o tratamento **g1w1** combina ambas as funcionalidades investigadas.

#### 3.3. Jogo Desenvolvido

Para o experimento foi desenvolvido um jogo de programação composto por quatro fases (sequência, variáveis, condicionais e repetição), cada uma contendo seis desafios do tipo quebra-cabeça. Diferentemente de ambientes tradicionais, os participantes selecionavam uma entre quatro soluções candidatas, cuja execução era imediatamente apresentada no cenário do jogo.

A Figura 1 apresenta a interface principal do jogo, composta pelo cenário do desafio, alternativas de solução e uma área de apoio contendo dicas relacionadas ao conceito abordado.



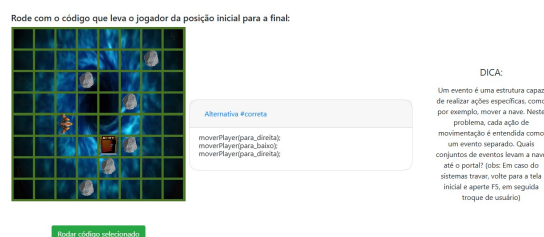
**Figura 1. Interface de um nível do jogo utilizado no experimento.**

Ao concluir corretamente um desafio, o participante avançava para o nível seguinte. Em caso de erro, podia tentar novamente ou, conforme o tratamento experimental, receber um novo desafio semelhante gerado por PCG ou consultar um exemplo resolvido.

Desafios e exemplos resolvidos eram gerados por mecanismos de PCG baseados em templates parametrizados para cada conceito de programação. A partir do template, o sistema construía automaticamente uma solução válida utilizando uma gramática específica, que servia de base para gerar as alternativas apresentadas ao estudante.

Durante a geração, o algoritmo verificava restrições estruturais para garantir desafios solucionáveis e pedagogicamente equivalentes, controlando a dificuldade por parâmetros do template, como a quantidade de passos necessária para concluir o desafio.

A Figura 2 ilustra a apresentação de um exemplo resolvido para um desafio conceitualmente semelhante ao problema atual.



**Figura 2. Exemplo resolvido exibido pelo sistema.**

A adoção de alternativas pré-definidas teve como objetivo minimizar dificuldades associadas à escrita de código e concentrar a experiência de aprendizagem em conceitos de pensamento computacional, como sequenciamento, variáveis, condicionais e estruturas de repetição. Essa escolha permitiu que os participantes focassem na lógica necessária para resolver os desafios, independentemente de conhecimentos específicos de sintaxe. Contudo, por enfatizar o reconhecimento de soluções em vez da construção de respostas, essa característica deve ser considerada ao interpretar os resultados do experimento.

### **3.4. Procedimentos Experimentais**

Os participantes foram recrutados por meio de disciplinas vinculadas a um programa de formação técnica em pensamento computacional. A participação foi voluntária, mediante consentimento eletrônico ao iniciar o sistema, sem qualquer impacto nas avaliações acadêmicas. Durante o período de execução do experimento, o programa possuía 371 estudantes ativos. Os participantes já possuíam contato prévio com conceitos fundamentais de programação, como sequenciamento de comandos, variáveis, estruturas condicionais e estruturas de repetição. A maioria encontrava-se na faixa etária entre 22 e 25 anos.

O jogo permaneceu disponível durante uma semana, podendo ser acessado livremente a partir das residências dos participantes ou dos laboratórios da instituição. Após responder ao formulário de consentimento e ao questionário demográfico, os estudantes tiveram acesso ao jogo. Embora os desafios estivessem organizados em quatro fases sequenciais, não havia obrigatoriedade de seguir uma ordem específica, nem limite de tempo para utilização do sistema. Todas as interações realizadas foram registradas automaticamente em arquivos de log.

A distribuição dos participantes entre os tratamentos foi realizada por meio de um Delineamento Inteiramente Casualizado (DIC) [Montgomery 2020]. Para manter grupos aproximadamente equilibrados, a cada novo participante era selecionado aleatoriamente um dos tratamentos ainda menos representados. Não foi realizado treinamento específico antes da utilização do sistema, uma vez que os participantes já possuíam experiência prévia com conceitos introdutórios de programação e ambientes digitais semelhantes. Durante a execução do estudo, os estudantes tiveram acesso a suporte online para esclarecimento de dúvidas técnicas. A principal tarefa consistia em concluir o maior número possível de desafios disponíveis no jogo.

#### **3.4.1. Variáveis Coletadas**

Foram coletadas, a partir dos registros de interação do jogo, as seguintes métricas: identificador da sessão, quantidade de fases concluídas, quantidade de níveis concluídos, número de tentativas até a obtenção da resposta correta, tempo total de utilização do jogo e avaliações positivas e negativas atribuídas aos níveis.

## **4. Resultados e Discussão**

Esta seção apresenta os resultados obtidos no experimento, contemplando métricas relacionadas ao engajamento dos participantes e ao desempenho na resolução dos desafios do jogo. Por fim, é apresentada a discussão sobre os resultados apresentados, sintetizando os achados do estudo.

### **4.1. Resultados**

#### **4.1.1. Visão Geral dos Dados**

O jogo de programação permaneceu disponível aos participantes durante sete dias. Ao todo, 90 estudantes acessaram o sistema e foram distribuídos aleatoriamente entre os quatro tratamentos experimentais.

Após a análise dos registros de interação, foram removidas as sessões de participantes que apenas navegaram pela interface do jogo sem iniciar ou concluir qualquer nível. Após esse processo de filtragem, permaneceram 51 participantes com sessões válidas para análise.

A Tabela 1 apresenta a distribuição dos participantes entre os tratamentos, variando entre 11 e 16 participantes por grupo. A tabela também resume as principais métricas utilizadas neste estudo, incluindo o tempo mediano de utilização do jogo, o percentual mediano de níveis jogados e os percentuais medianos de respostas corretas e incorretas.

| Trat. | N  | Tempo | Níveis | Acertos | Erros |
|-------|----|-------|--------|---------|-------|
| g0w0  | 13 | 8,6   | 25%    | 89%     | 11%   |
| g0w1  | 11 | 11,8  | 25%    | 100%    | 0%    |
| g1w0  | 11 | 17,3  | 38%    | 90%     | 10%   |
| g1w1  | 16 | 10,0  | 29%    | 98%     | 2%    |

**Tabela 1. Resumo das métricas observadas por tratamento.**

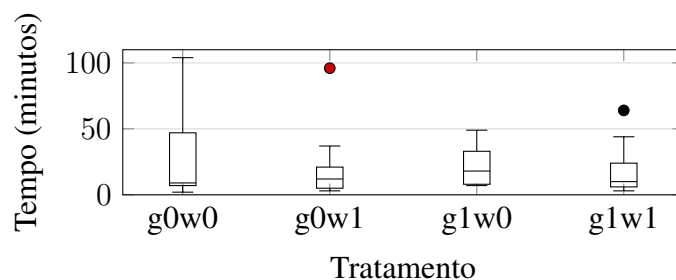
Observa-se que os grupos apresentaram tamanhos relativamente equilibrados e valores iniciais semelhantes para as métricas analisadas.

#### 4.1.2. Engajamento dos Alunos

**Tempo Total** Esta métrica representa o tempo total gasto por cada participante durante sua interação com o jogo. Considerando todos os tratamentos, a mediana geral foi de aproximadamente 10,4 minutos.

A Figura 3 apresenta a distribuição do tempo total de utilização do jogo para cada tratamento experimental. Observa-se elevada variabilidade entre os participantes, incluindo a presença de valores extremos em alguns grupos. O tratamento g1w0 apresentou a maior mediana de tempo de utilização, enquanto os demais grupos apresentaram valores medianos entre 9 e 12 minutos.

Apesar dessas diferenças, as distribuições observadas apresentam ampla sobreposição, dificultando a identificação de efeitos claramente associados aos fatores investigados. Esse comportamento sugere que a variabilidade individual dos participantes teve influência relevante sobre essa métrica.



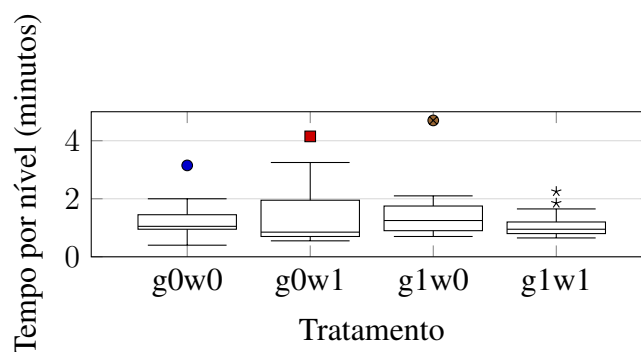
**Figura 3. Tempo (em minutos) gasto pelos estudantes durante o jogo**

**Tempo por nível** Embora o tempo total de utilização do jogo forneça uma medida geral de engajamento, essa métrica pode ser influenciada pela quantidade de níveis jogados por cada participante. Para reduzir esse efeito, foi calculada uma medida normalizada correspondente ao tempo total gasto no jogo dividido pelo número de níveis jogados.

A Figura 4 apresenta a distribuição dessa métrica para os quatro tratamentos investigados. Em comparação com o tempo total de utilização, observa-se menor dispersão dos dados, embora ainda existam alguns valores extremos. As distribuições apresentam considerável sobreposição, indicando diferenças limitadas entre os grupos.

As medianas observadas sugerem diferenças modestas entre os tratamentos. Os grupos com exemplos resolvidos apresentaram tempos medianos inferiores aos grupos sem esse recurso. Considerando os efeitos principais, a presença de exemplos resolvidos esteve associada a uma redução aproximada de 20% no tempo mediano por nível, enquanto a geração de desafios semelhantes esteve associada a um aumento aproximado de 3%.

Apesar dessas diferenças, a elevada sobreposição entre as distribuições sugere que os efeitos observados foram pequenos quando comparados à variabilidade dos dados.

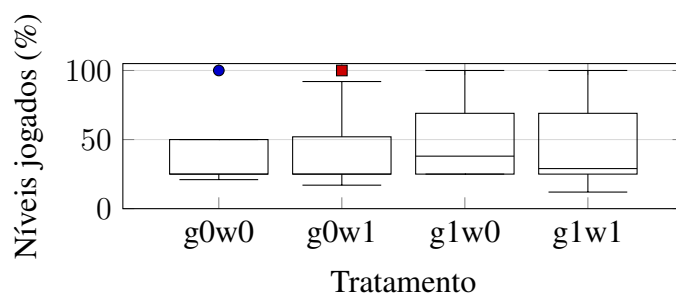


**Figura 4. Variabilidade do tempo total, em minutos, dividido pelo número de níveis jogados**

**Percentual de Níveis Jogados** O percentual de níveis jogados foi utilizado como uma medida de cobertura da atividade, representando o quanto cada participante explorou o conteúdo disponível no jogo.

A Figura 5 apresenta a distribuição dessa métrica para os quatro tratamentos investigados. Observa-se elevada variabilidade entre os participantes, com alguns estudantes explorando apenas uma pequena parcela do jogo, enquanto outros completaram praticamente todo o conteúdo disponível.

As medianas observadas sugerem um efeito positivo da geração de desafios semelhantes sobre a cobertura dos níveis. Os tratamentos com essa funcionalidade apresentaram medianas superiores às observadas nos tratamentos sem geração de novos desafios após erros. Em contrapartida, a presença de exemplos resolvidos esteve associada a uma redução da cobertura observada, particularmente quando combinada à funcionalidade de geração de desafios semelhantes.



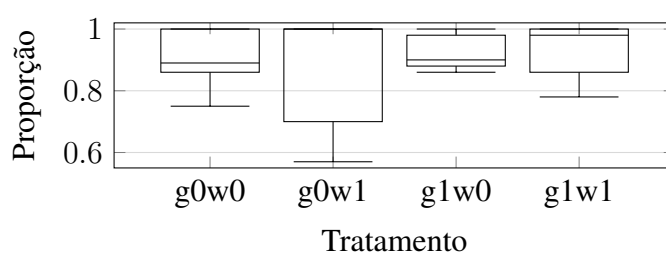
**Figura 5. Percentual de níveis jogados no jogo**

Embora essas diferenças sejam visíveis nas medianas dos grupos, a elevada variabilidade dos dados observada na Figura 5 dificulta a atribuição dessas diferenças exclusivamente aos tratamentos experimentais.

#### 4.1.3. Desempenho dos Participantes

O percentual de respostas corretas foi utilizado como medida de desempenho dos participantes. Conforme apresentado na Tabela 1, a mediana geral de acertos foi de aproximadamente 95%, indicando que a maioria dos estudantes conseguiu resolver corretamente grande parte dos desafios disponibilizados.

A Figura 6 apresenta a distribuição do percentual de respostas corretas para os quatro tratamentos investigados. Observa-se que todos os grupos apresentaram elevadas taxas de acerto, com medianas próximas ou superiores a 90%. Em diversos casos, os participantes atingiram desempenho máximo, obtendo 100% de respostas corretas.



**Figura 6. Distribuição das avaliações positivas por tratamento**

Os resultados sugerem diferenças reduzidas entre os tratamentos. A presença da funcionalidade de geração de desafios semelhantes esteve associada a um aumento aproximado de 1% na mediana do percentual de acertos. Já a disponibilização de exemplos resolvidos esteve associada a uma mediana de desempenho próxima de 100% de respostas corretas.

Entretanto, a elevada taxa de acerto observada em todos os grupos indica uma reduzida variabilidade nessa métrica, dificultando a identificação de diferenças mais expressivas entre os tratamentos.

## 4.2. Discussão

Os resultados apresentados indicam tendências distintas para os fatores investigados. Entretanto, o teste de Kruskal–Wallis não identificou diferenças estatisticamente significativas entre os tratamentos ( $p > 0,05$ ), não sendo possível rejeitar a hipótese nula de igualdade entre os grupos.

### 4.2.1. RQ1 – A geração de desafios semelhantes após respostas incorretas influencia o engajamento e o desempenho dos estudantes?

Os tratamentos que utilizaram geração de desafios semelhantes apresentaram maiores percentuais medianos de níveis jogados, sugerindo maior persistência dos participantes durante a interação com o jogo. Esse comportamento indica que substituir um desafio respondido incorretamente por outro conceitualmente equivalente pode favorecer a continuidade da atividade ao oferecer novas oportunidades de prática. Entretanto, a elevada variabilidade entre os participantes e a ausência de significância estatística impedem atribuir esse efeito exclusivamente à funcionalidade investigada.

Em relação ao desempenho, os percentuais de respostas corretas permaneceram elevados em todos os tratamentos, sem diferenças relevantes associadas à geração de desafios semelhantes. Assim, os resultados sugerem que essa funcionalidade pode contribuir mais para o engajamento do que para ganhos imediatos de desempenho.

Embora existam poucos estudos sobre geração de desafios semelhantes em jogos de programação, esse comportamento é compatível com a literatura sobre PCG em jogos educacionais, que destaca a geração automática de desafios como estratégia para ampliar oportunidades de prática e manter o engajamento dos estudantes [De Kegel and Haahr 2020, Silva and D’Emery 2024]. No presente estudo, entretanto, esse potencial refletiu-se apenas em tendências descritivas, sem evidências estatísticas de melhoria no desempenho.

### 4.2.2. RQ2 – O uso de exemplos resolvidos influencia o engajamento e o desempenho dos estudantes?

Os tratamentos que disponibilizavam exemplos resolvidos apresentaram menor tempo mediano por nível e maiores percentuais de respostas corretas, sugerindo que esse mecanismo de apoio pode tornar a resolução dos desafios mais eficiente. Em particular, os tratamentos com exemplos resolvidos apresentaram medianas de acerto próximas de 100%.

Entretanto, essas diferenças não foram estatisticamente significativas. Esse resultado difere parcialmente dos achados de Zhi et al. [Zhi et al. 2018], que observaram redução no tempo de resolução e melhorias decorrentes do uso de mecanismos de apoio instrucional em jogos de programação. Em ambos os estudos, contudo, os recursos de apoio mostraram potencial para favorecer a resolução dos desafios. No presente trabalho, esse benefício manifestou-se apenas como uma tendência descritiva, possivelmente em função do elevado conhecimento prévio dos participantes e do efeito teto observado.

De forma semelhante, Rahman et al. [Rahman and Du Boulay 2010] destacam que exemplos resolvidos favorecem a compreensão de problemas semelhantes ao fornecer modelos explícitos de resolução. Os resultados obtidos corroboram essa perspectiva, embora indiquem que a efetividade desse mecanismo depende das características da tarefa e do perfil dos estudantes.

#### 4.2.3. Lições Aprendidas

A ausência de diferenças estatisticamente significativas pode ser parcialmente explicada pelas características do contexto experimental.

Um primeiro aspecto refere-se ao elevado desempenho dos participantes. A mediana geral de acertos foi de aproximadamente 95

Outro aspecto refere-se ao formato da atividade. Como os participantes selecionavam soluções candidatas em vez de construí-las, a atividade privilegiou o reconhecimento, reduzindo barreiras sintáticas e a carga cognitiva. Segundo a Teoria da Carga Cognitiva (*Cognitive Load Theory*) [Sweller 2011], recursos instrucionais, como exemplos resolvidos, tendem a produzir maiores benefícios em tarefas que exigem maior esforço cognitivo e construção ativa de soluções. Isso pode explicar por que os exemplos resolvidos apresentaram apenas tendências de melhoria, sem diferenças estatisticamente significativas.

Além disso, os participantes já possuíam experiência prévia com conceitos introdutórios de programação, reduzindo a necessidade de suporte adicional durante a resolução dos desafios. Somado ao número reduzido de participantes por tratamento (11 a 16 estudantes), esse aspecto pode ter limitado o poder estatístico das análises e dificultado a identificação de diferenças entre os tratamentos.

Em conjunto, esses fatores sugerem que o contexto experimental não foi o mais adequado para evidenciar os efeitos das funcionalidades investigadas. Para professores e desenvolvedores, os resultados indicam que exemplos resolvidos tendem a produzir maiores benefícios em desafios mais complexos, atividades baseadas na construção de soluções e com estudantes menos experientes em programação.

Mais do que evidenciar a ausência de diferenças estatisticamente significativas, este estudo demonstra que características do delineamento experimental podem mascarar os efeitos de intervenções pedagógicas. Assim, fatores como conhecimento prévio elevado, tarefas baseadas em reconhecimento de soluções e baixa variabilidade do desempenho devem ser considerados no planejamento de futuras avaliações empíricas.

### 5. Ameaças à Validade

A aprendizagem foi avaliada por meio de métricas de desempenho obtidas durante a interação com o jogo, em um contexto baseado na seleção de alternativas. Esse formato privilegia o reconhecimento de soluções em vez da construção ativa de respostas. O engajamento foi estimado por métricas como tempo de utilização, cobertura dos níveis e avaliações positivas e negativas, as quais podem não capturar integralmente aspectos motivacionais ou afetivos da experiência dos participantes.

O experimento foi conduzido em ambiente online, com participação voluntária e sem controle das condições de execução. Dessa forma, fatores externos, como distrações,

interrupções, diferenças de dispositivos ou condições de acesso, podem ter introduzido variabilidade não controlada nos resultados observados. Além disso, por não haver supervisão presencial durante a realização da atividade, não é possível descartar completamente a consulta a materiais externos ou a interação entre participantes, fatores que podem ter influenciado o desempenho registrado.

Os participantes pertenciam a um programa de formação técnica específico e possuíam experiência prévia em lógica de programação e programação estruturada. Consequentemente, os resultados obtidos podem não ser diretamente generalizáveis para estudantes iniciantes, alunos da educação básica ou outros contextos educacionais.

O número de participantes por tratamento (entre 11 e 16) e o elevado desempenho inicial observado reduziram o poder estatístico das análises, dificultando a detecção de diferenças significativas entre os tratamentos. Além disso, como os participantes podiam percorrer sucessivamente diferentes fases do jogo em uma mesma sessão, não se pode descartar a ocorrência de fadiga ao longo da atividade, o que pode ter influenciado métricas como tempo de resolução e continuidade da interação. A baixa frequência de erros durante a resolução dos desafios também limitou análises mais aprofundadas relacionadas ao número de tentativas necessárias para atingir uma resposta correta.

## **6. Uso de Inteligência Artificial Generativa**

Em conformidade com as diretrizes de integridade acadêmica, declara-se que ferramentas de Inteligência Artificial Generativa foram utilizadas exclusivamente para apoio no aprimoramento estrutural, revisão gramatical e adequação estilística do texto. Todo o conteúdo científico, análises de dados, interpretações e redação final foram conduzidos e validados integralmente pelos autores.

## **7. Conclusões**

Este trabalho investigou o uso de duas funcionalidades pedagógicas apoiadas por Geração Procedural de Conteúdo (PCG) em um jogo de programação: a geração de desafios semelhantes após respostas incorretas e a disponibilização de exemplos resolvidos como mecanismo de apoio à aprendizagem. Para isso, foi conduzido um experimento controlado online com delineamento fatorial  $2 \times 2$  envolvendo 51 participantes.

Os resultados sugerem que a geração de desafios semelhantes pode favorecer a continuidade da interação com o jogo, enquanto os exemplos resolvidos apresentaram tendência de redução do tempo por nível e aumento das taxas de acerto. Entretanto, nenhuma dessas diferenças apresentou significância estatística.

Como principal contribuição, este estudo evidencia que a avaliação de mecanismos de apoio instrucional em jogos de programação é fortemente influenciada pelas características do contexto experimental. Aspectos como conhecimento prévio dos participantes, tarefas baseadas em reconhecimento de soluções e baixa variabilidade do desempenho podem dificultar a identificação dos efeitos produzidos por essas intervenções.

Como trabalhos futuros, pretende-se replicar o estudo utilizando conceitos mais avançados de programação, tarefas baseadas na construção de soluções e amostras mais heterogêneas, buscando ampliar a sensibilidade experimental para avaliar os efeitos de desafios semelhantes e exemplos resolvidos.

## Referências

- De Kegel, B. and Haahr, M. (2020). Procedural puzzle generation: A survey. *IEEE Transactions on Games*, 12(1):21–40.
- Falkner, K., Sentance, S., Vivian, R., Barksdale, S., Busuttil, L., Cole, E., Liebe, C., Maiorana, F., McGill, M. M., and Quille, K. (2019). An international study piloting the measuring teacher enacted computing curriculum (metrecc) instrument. In *ITiCSE-WGR '19: Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, pages 111–142.
- Hafis, M., Tolle, H., and Supianto, A. A. (2019). A literature review of empirical evidence on procedural content generation in game-related implementation. *Journal of Information Technology and Computer Science*.
- Harms, K. J., Balzuweit, E., Chen, J., and Kelleher, C. (2016). Learning programming from tutorials and code puzzles: Children’s perceptions of value. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 59–67.
- Hassany, M., Brusilovsky, P., Ke, J., Akhuseyinoglu, K., and Narayanan, A. B. L. (2024). Human-ai co-creation of worked examples for programming classes. *arXiv preprint arXiv:2402.16235*.
- Jedlitschka, A., Ciolkowski, M., and Pfahl, D. (2008). Reporting experiments in software engineering. In *Guide to Advanced Empirical Software Engineering*, pages 201–228. Springer London, London.
- Jury, B., Lorusso, A., Leinonen, J., Denny, P., and Luxton-Reilly, A. (2024). Evaluating llm-generated worked examples in an introductory programming course. In *Proceedings of the 26th Australasian computing education conference*, pages 77–86.
- Ko, A. J., Latoza, T. D., and Burnett, M. M. (2015). A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering*, 20(1):110–141.
- Lambić, D., Dorić, B., and Ivakić, S. (2020). Investigating the effect of the use of code.org on younger elementary school students’ attitudes towards programming. *Behaviour & Information Technology*, pages 1–12.
- Lindberg, R. S. N., Laine, T. H., and Haaranen, L. (2018). Gamifying programming education in k-12: A review of programming curricula in seven countries and programming games. *British Journal of Educational Technology*.
- Lui, A., Cheung, Y., and Li, S. (2008). Leveraging students’ programming laboratory work as worked examples. *ACM SIGCSE Bulletin*, 40:69–73.
- Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., and Settle, A. (2014). Computational thinking in k-9 education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR '14)*.
- Margulieux, L. E. et al. (2020a). Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, 7(1).

- Margulieux, L. E., Morrison, B. B., and Decker, A. (2020b). Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, 7(1):19.
- McLaren, B. M. and Isotani, S. (2011a). When is it best to learn with all worked examples? In *International conference on artificial intelligence in education*, pages 222–229. Springer.
- McLaren, B. M. and Isotani, S. (2011b). When is it best to learn with all worked examples? In *Proceedings of the 15th International Conference on Artificial Intelligence in Education (AIED '11)*, pages 222–229.
- Montgomery, D. C. (2020). *Design and Analysis of Experiments*. John Wiley & Sons, Inc., Hoboken, NJ, USA.
- Muldner, K., Jennings, J., and Chiarelli, V. (2022). A review of worked examples in programming activities. *ACM Transactions on Computing Education*, 23(1):1–35.
- Rahman, S. and Du Boulay, B. (2010). Learning programming via worked examples.
- Sandhu, A. and McCoy, J. (2019). A framework for integrating architectural design patterns into pcg. In *Proceedings of the 14th International Conference on the Foundations of Digital Games (FDG '19)*.
- Santana, A. and Aranha, E. (2019). An approach to generate virtual tutors for game programming classes. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*.
- Silva, B. A. and D'Emery, R. A. (2024). Desenvolvimento e avaliação do jogo sério educacional coding hunter: um ambiente para prática de pensamento computacional. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 791–806. SBC.
- Silva, G., Pessoa, J. O., da Magatti, I. N., Gonçalves, A. C., Garcia, K. R., Brandão, A. L., and Vittori, K. (2024). Newbot: Jogo educativo para o ensino do pensamento computacional. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 3127–3138. SBC.
- Silva, T., Santana, A., and Aranha, E. (2019). Investigating video classes formats for teaching digital game programming in high school. In *Brazilian Symposium on Computers in Education (SBIE)*, page 753.
- Sweller, J. (2011). Cognitive load theory and e-learning. In *International Conference on Artificial Intelligence in Education*, pages 5–6. Springer.
- Zhi, R., Lytle, N., and Price, T. W. (2018). Exploring instructional support design in an educational game for k-12 computing education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*, pages 747–752.