

SYNTAX: Gamificação do Ensino de Lógica de Programação com um *Serious Game* de Horror em Primeira Pessoa

SYNTAX: Gamifying Programming Logic Education Through a First-Person Horror Serious Game

Filipe Carvalho Matos Galvão¹, Victor Travassos Sarinho¹

¹Laboratório de Entretenimento Digital Aplicado (LEnDA)
Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana, BA – Brasil

filipeglv7@gmail.com, vsarinho@uefs.br

Abstract. Introduction: Introductory programming education faces challenges related to conceptual abstraction and the gap between theory and practice, particularly when beginners cannot easily associate source code with observable effects. **Objective:** This paper presents SYNTAX, an immersive serious game that combines Escape Room and horror mechanics to teach fundamental concepts of computational logic. **Methodology:** The game was developed using the Godot Engine and applies Scaffolding and Computational Thinking principles to transform programming concepts into progressively challenging puzzles inside an immersive first-person environment. **Results:** The prototype features five progressive levels, a diegetic programming interface, and real-time environmental feedback, allowing code modifications to directly affect the virtual world. The implementation demonstrates the technical feasibility and educational potential of the proposed approach and provides a basis for future empirical evaluation with students.

Keywords Serious Games, Programming Education, Computational Thinking, Instructional Design, Godot Engine.

Resumo. Introdução: O ensino introdutório de programação enfrenta dificuldades relacionadas à abstração conceitual e à distância entre teoria e prática, especialmente quando o estudante não associa facilmente código e efeito observável. **Objetivo:** Este artigo apresenta SYNTAX, um jogo sério imersivo que combina mecânicas de Escape Room e horror para ensinar fundamentos de lógica computacional. **Metodologia:** O jogo foi desenvolvido na Godot Engine e utiliza princípios de Scaffolding e Pensamento Computacional para transformar conceitos de programação em desafios progressivos dentro de um ambiente em primeira pessoa. **Resultados:** O protótipo possui cinco níveis progressivos, interface de programação diegética e feedback ambiental em tempo real, permitindo que alterações no código modifiquem diretamente o ambiente virtual. A implementação demonstra a viabilidade técnica e o potencial educacional da abordagem, servindo como base para futuras avaliações empíricas com estudantes.

Palavras-Chave Jogos Sérios, Ensino de Programação, Pensamento Computacional, Design Instrucional, Godot Engine.

1. Introdução

A evasão em cursos de Computação possui causas multifatoriais, sendo a dificuldade de compreender abstrações computacionais um fator recorrente [Watson e Li 2014]. Para estudantes iniciantes, a relação entre uma linha de código e sua execução prática pode permanecer pouco evidente, criando uma lacuna entre representação abstrata e efeito observável [Wing 2006].

A gamificação e os *Serious Games* constituem alternativas para aproximar teoria e prática, mas muitas soluções privilegiam a acessibilidade lúdica ou a profundidade técnica de forma isolada [Michael e Chen 2005]. Dessa forma, experiências em que o aprendizado esteja integrado à própria ação de jogar podem reduzir a separação entre exercício didático e interação com o ambiente virtual.

Este trabalho apresenta SYNTAX, um jogo em primeira pessoa desenvolvido na *Godot Engine*¹ que combina *Escape Room* e horror psicológico para ensinar lógica de programação. O jogador “hackeia” o ambiente por meio de pseudocódigo inspirado em Python e GDScript, trabalhando variáveis, condicionais, laços, decomposição e depuração.

A proposta é voltada a estudantes iniciantes, especialmente do ensino médio/técnico e dos primeiros semestres de Computação. A hipótese de desenvolvimento é que interfaces diegéticas, *Scaffolding* e *feedback* ambiental imediato podem favorecer engajamento, compreensão inicial e percepção de aplicabilidade dos conceitos. Aprendizagem e retenção, entretanto, permanecem dimensões a serem avaliadas empiricamente em trabalhos futuros.

2. Fundamentação Teórica

O Pensamento Computacional, conforme [Wing 2006], envolve decomposição, reconhecimento de padrões, abstração e algoritmos. Em SYNTAX, esses pilares orientam desafios que exigem representar estados, identificar repetições, decompor problemas e formular procedimentos executáveis, evitando que o jogo se limite à memorização de sintaxe.

A progressão adota princípios de *Scaffolding*, com maior suporte nos primeiros contatos com cada conceito e redução gradual das pistas conforme o jogador desenvolve autonomia [Papert 2020, Prensky 2003]. Essa organização busca impedir que a apresentação de uma nova estrutura de programação ocorra simultaneamente a um salto excessivo de dificuldade.

A imersão em jogos digitais pode envolver dimensões espaciais, narrativas e mecânicas [Schell 2008]. SYNTAX enfatiza a dimensão espacial por meio de interfaces diegéticas, ou seja, elementos de interação incorporados ao próprio mundo ficcional. O horror acrescenta tensão e urgência aos ciclos curtos de ação, erro e correção, buscando sustentar o engajamento durante a resolução dos desafios [Cziksentmihalyi 1990].

A principal contribuição de design é posicionar o código como instrumento de intervenção no ambiente: alterações no programa produzem consequências visíveis e imediatas no espaço virtual. Dessa forma, o jogador observa diretamente a relação entre abstração, execução e efeito, em vez de executar exercícios externos ao mundo do jogo.

¹<https://godotengine.org/>

3. Método

SYNTAX foi desenvolvido por prototipação iterativa, com implementação de mecânicas, testes internos de funcionamento e refinamento dos desafios. Foram utilizados *Godot Engine* 4.2 (GDScript), Blender 4.0 para modelagem 3D, Aseprite para elementos visuais da interface e Audacity para edição sonora.

Os desafios utilizam pseudocódigo inspirado em Python/GDScript por apresentar uma sintaxe próxima da leitura natural e reduzir a carga cognitiva inicial. O objetivo não é ensinar uma linguagem específica em profundidade, mas introduzir estruturas fundamentais de lógica de programação que possam ser transferidas posteriormente para linguagens formais.

Tabela 1. Mapeamento pedagógico dos níveis de SYNTAX.

Nível	Mecânica	Pilar	Progressão	Objetivo
1	Variáveis	Abstração	Alteração guiada de valores e observação do estado do ambiente	Compreender estado, mutabilidade e tipos básicos
2	<i>if/else</i>	Algoritmos	Completar condições e combinar operadores	Compreender fluxo condicional e comparações
3	Laços	Padrões	Substituir chamadas repetidas por iteração	Reconhecer padrões e reduzir redundâncias
4	Decomposição	Decomp./abstr.	Coletar e combinar pistas distribuídas no cenário	Sintetizar informações e decompor problemas
5	Depuração	Integração	Corrigir erros sintáticos e lógicos combinados	Desenvolver depuração sistemática

Elaborado pelos autores (2026).

Os cinco níveis seguem dificuldade crescente (Tabela 1), com uma etapa inicial guiada e redução progressiva das pistas dentro de cada conceito. Assim, o estudante pode primeiro compreender a relação entre ação e resultado para, em seguida, resolver variações mais autônomas antes de avançar para a próxima mecânica.

O cenário foi concebido como um “*Room Scale*” intencionalmente restrito, composto por um único quarto onde objetos como lâmpada, ventilador e porta funcionam como elementos ativos controlados por *scripts*. A limitação espacial favorece familiaridade com o ambiente e transforma objetos cotidianos em partes do *puzzle*. Luz verde, amarela e vermelha sinaliza, respectivamente, estados seguro, de alerta e crítico, diminuindo a dependência de instruções textuais extensas.

O ciclo de interação possui quatro etapas: estímulo ambiental negativo, análise no terminal, intervenção por edição do código e consequência imediata no cenário (Figura 1). A comunicação entre *puzzles* e objetos utiliza *Signals* do Godot, seguindo o padrão *Observer*, o que mantém os sistemas desacoplados e facilita a resposta do ambiente às ações do jogador.

4. Protótipo Desenvolvido e Resultados de Implementação

Os resultados referem-se ao protótipo implementado, e não a uma validação empírica de aprendizagem. A interface é integralmente diegética: para acessar o sistema de programação, o jogador se aproxima fisicamente do computador, pressiona a tecla de interação e passa por uma transição de câmera até a IDE simulada (Figura 2). Essa sequência separa os modos de exploração e resolução e reforça a metáfora de que programação é uma ferramenta usada para modificar o mundo do jogo.

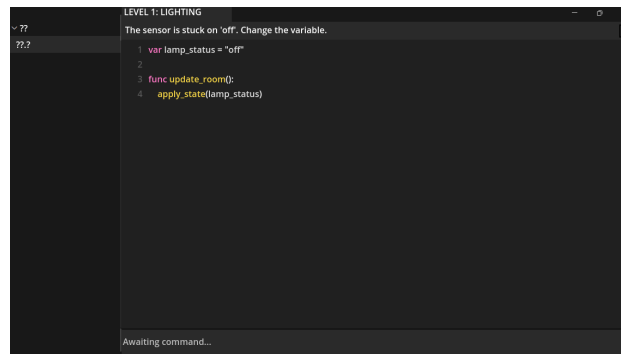


Figura 1. Interface de acesso à IDE simulada.

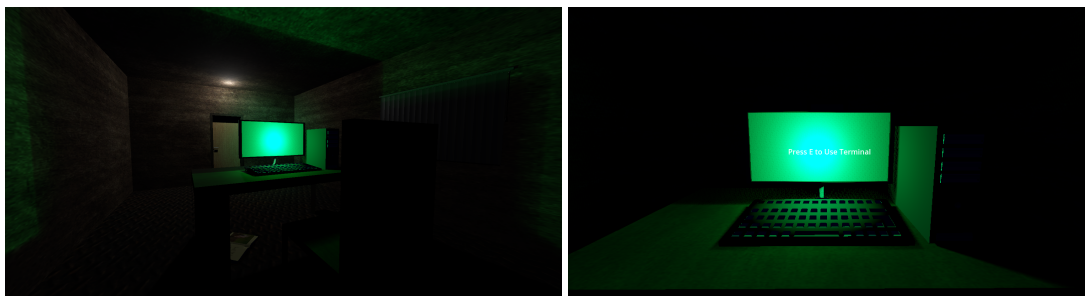


Figura 2. Quarto do jogo e computador utilizado para acessar a interface de programação.

Foram implementados cinco *puzzles*. No primeiro, o jogador restaura a iluminação alterando o valor de uma variável. No segundo, utiliza uma estrutura *if/else* para controlar um sistema de superaquecimento sob pressão temporal. O terceiro exige reconhecer repetição e substituir chamadas redundantes por uma estrutura de laço. No quarto, uma senha deve ser reconstruída a partir de pistas distribuídas pelo cenário, articulando exploração e decomposição. Por fim, o quinto nível combina erros sintáticos e lógicos que precisam ser identificados e corrigidos. Em todos os casos, a solução modifica imediatamente o ambiente, tornando visível o efeito do código executado.

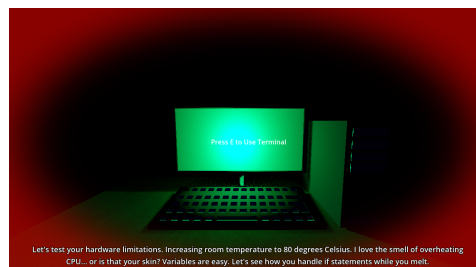


Figura 3. Feedback visual de erro no sistema de temperatura.

Entre as limitações, o protótipo possui escopo intencionalmente restrito: é *single-player*, *offline*, tem duração estimada de 15–20 minutos e cobre apenas conceitos introdutórios de programação. A narrativa da IA antagonista ainda é apresentada por textos estáticos, sem diálogos dinâmicos ou ramificações, e não foi implementado um sistema de salvamento. Além disso, como ainda não houve aplicação com participantes, os resultados não permitem concluir efeitos sobre aprendizagem ou retenção; essas

dimensões permanecem como objetivos de avaliação futura.

5. Considerações Finais

SYNTAX explora a integração entre programação e jogabilidade por meio de interfaces diegéticas, *Scaffolding*, horror e desafios de *Escape Room*. A proposta procura reduzir a distância entre teoria e prática ao fazer com que o avanço no ambiente dependa diretamente da compreensão de estruturas computacionais. Por ainda não ter sido validado com estudantes, os resultados apresentados devem ser interpretados como evidências de viabilidade técnica e potencial de design, e não como comprovação de eficácia pedagógica.

As principais contribuições do trabalho são: a aplicação de interfaces diegéticas ao ensino de código; o uso de *Scaffolding* em mecânicas de horror e *Escape Room*; o mapeamento explícito entre desafios e pilares do Pensamento Computacional; e uma arquitetura de baixo custo baseada em ferramentas *open-source*, com possibilidade de reprodução em outros contextos educacionais.

Como trabalhos futuros, propõem-se a adoção de linguagens amplamente utilizadas, como Python e JavaScript, com interpretador embutido; o desenvolvimento de uma narrativa dinâmica para a IA antagonista; e a realização de estudos empíricos com estudantes iniciantes, incluindo pré-teste, pós-teste e medidas de retenção. Também se pretende comparar a abordagem com métodos tradicionais de Introdução à Programação e investigar mecanismos de adaptação da dificuldade ao desempenho percebido durante a partida.

Referências

- Cziksentmihalyi, M. (1990). Flow—the psychology of optimal experience. 1990.
- Michael, D. R. e Chen, S. L. (2005). *Serious games: Games that educate, train, and inform*. Muska & Lipman/Premier-Trade.
- Papert, S. A. (2020). *Mindstorms: Children, computers, and powerful ideas*. Basic books.
- Prensky, M. (2003). Digital game-based learning. *Computers in entertainment (CIE)*, 1(1):21–21.
- Schell, J. (2008). *The Art of Game Design: A book of lenses*. CRC press.
- Watson, C. e Li, F. W. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 39–44.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.