

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística



Guilherme Adenilson de Jesus

Jogo Educacional para o Ensino de Grafos para a Computação

Florianópolis

2025

Guilherme Adenilson de Jesus

**Jogo Educacional para o Ensino de Grafos para a
Computação**

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos necessários para a obtenção do Grau de Bacharel em Ciência da Computação.
Orientador: Prof. Dr. Rafael de Santiago

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

Florianópolis
2025

Ficha Catalográfica

Guilherme Adenilson de Jesus

Jogo Educacional para o Ensino de Grafos para a Computação - Florianópolis,
2025 - 60 p., 30 cm.

Orientador: Prof. Dr. Rafael de Santiago

Grafos; *Game Engine*; Jogos eletrônicos; Aprendizagem

I. Universidade Federal de Santa Catarina. Ciências da Computação. II. Jogo
Educacional para o Ensino de Grafos para a Computação.

Guilherme Adenilson de Jesus

Jogo Educacional para o Ensino de Grafos para a Computação

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Comissão Examinadora

Prof. Dr. Rafael de Santiago
Universidade Federal de Santa Catarina
Orientador

Prof. Dr. Jean Carlo Rossa Hauck
Universidade Federal de Santa Catarina

Prof. Dr. Alvaro Junio Pereira Franco
Universidade Federal de Santa Catarina

Florianópolis, 9 de dezembro de 2025

Resumo

A Teoria dos Grafos é uma área fundamental da Ciência da Computação, com aplicações em problemas como Caminho Euleriano, Caixeiro Viajante e Conectividade. Dada sua importância, a compreensão de seus conceitos é essencial, embora o aprendizado tradicional possa ser desafiador. Jogos eletrônicos surgem como um mecanismo auxiliar para tornar o estudo mais interativo. O presente trabalho tem como objetivo o desenvolvimento de um jogo educacional para o ensino na Computação. O software é implementado na game engine Godot, com uma arquitetura modular que permite a fácil adição de novos conteúdos. O jogo apresenta um editor de grafos interativo e módulos de problemas, como Caminho Mínimo, Árvore Geradora Mínima e Ciclo Euleriano. Como principal contribuição, o projeto implementa um sistema de avaliação de duas etapas: o software valida a resposta objetiva do aluno e, em seguida, utiliza uma LLM para analisar a justificativa discursiva, fornecendo feedback qualitativo sobre o raciocínio. O resultado é uma ferramenta didática funcional que combina interatividade e inteligência artificial para aprimorar o processo de aprendizagem.

Palavras-Chave: Grafos; *Game Engine*; Jogos eletrônicos; Aprendizagem

Abstract

Graph Theory is a fundamental area of Computer Science, with applications in problems such as Eulerian Path, Traveling Salesman Problem, and Connectivity. Given its importance, understanding its concepts is essential, although traditional learning can be challenging. Electronic games emerge as an auxiliary mechanism to make learning more interactive. This work aims to develop an educational game for teaching graph algorithms and theorems. The software is implemented in the Godot game engine, with a modular architecture that allows for the easy addition of new content. The game features an interactive graph editor and problem modules, such as Shortest Path, Minimum Spanning Tree, and Eulerian Cycle. As a main contribution, the project implements a two-stage evaluation system: the software validates the student's objective answer and then uses an LLM (Learning Management Module) to analyze the discursive justification, providing qualitative feedback on the reasoning. The result is a functional didactic tool that combines interactivity and artificial intelligence to enhance the learning process.

Keywords: Graphs; Game Engine; Videogames; Learning

Lista de figuras

Figura 1 – Exemplo de Busca em Largura	15
Figura 2 – Exemplo de Busca em Profundidade	15
Figura 3 – Exemplo de Ciclo Euleriano	16
Figura 4 – Exemplo de Ciclo Hamiltoniano	17
Figura 5 – Exemplo de Coloração	19
Figura 6 – <i>Puzzle 38</i> do Professor Layton and the Curious Village	24
Figura 7 – Grafo equivalente ao <i>puzzle 38</i>	24
Figura 8 – <i>Puzzle 92</i> do Professor Layton and the Curious Village	24
Figura 9 – <i>Puzzle 39</i> do Professor Layton and the Curious Village	25
Figura 10 – Grafo equivalente ao desenho da cartola no <i>puzzle 39</i>	25
Figura 11 – <i>Puzzle 10</i> do Professor Layton and the Pandora Box	26
Figura 12 – Grafo equivalente ao <i>puzzle 10</i>	26
Figura 13 – <i>Puzzle 125</i> do Professor Layton and the Pandora Box	26
Figura 14 – Pandora Box: Puzzle originalmente	27
Figura 15 – Pandora Box: Puzzle com anotações habilitada	27
Figura 16 – Chromagraph: nível usando coloração por vértices	27
Figura 17 – Chromagraph: nível usando coloração por arestas	27
Figura 18 – Nível 6 do Icosian Game	28
Figura 19 – Nível 12 do Icosian Game	28
Figura 20 – JSON com a descrição das questões	32
Figura 21 – Cenário do Menu de Questões	33
Figura 22 – Cenário do Editor de Grafos	34
Figura 23 – Máquina de Estados para Tratamento de Interações ao Editor de Grafo	34
Figura 24 – Arquivo para carregamento de um grafo	35
Figura 25 – Exemplo de Cena de uma Questão	36
Figura 26 – Funções avaliar_resposta e set_ia_feedback	37
Figura 27 – Tela do Menu de Questões	38
Figura 28 – Questão de Ciclo Euleriano	39
Figura 29 – Resultado correto da questão de Ciclo Euleriano	39
Figura 30 – Resultado correto da questão de Ciclo Euleriano	40
Figura 31 – Questão de Completar o Grafo com Ciclo Euleriano	40
Figura 32 – Questão de Completar o Grafo com Ciclo Euleriano com resultado	41
Figura 33 – Questão de Ciclo Hamiltoniano com resultado	41
Figura 34 – Questão de Coloração com resultado	42
Figura 35 – Questão de Ordenação Topológica com resultado	42

Figura 36 – Questão de Ordenação Topológica com resultado	43
Figura 37 – Questão de Árvore Geradora Mínima com resultado	43

Lista de tabelas

Tabela 1 – Comparativo entre os trabalhos relacionados 29

Sumário

1	INTRODUÇÃO	11
1.1	Objetivos	12
1.1.1	Objetivo Geral	12
1.1.2	Objetivos Específicos	12
1.2	Metodologia	12
1.3	Estrutura do Trabalho	13
2	FUNDAMENTAÇÃO TEÓRICA	14
2.1	Problemas Seleccionados de Grafos	14
2.1.1	Busca em Grafos	14
2.1.1.1	Busca em Largura	14
2.1.1.2	Busca em Profundidade	15
2.1.2	Caminhos e Ciclos	15
2.1.2.1	Caminhos e Ciclos Eulerianos	16
2.1.2.2	Caminhos e Ciclos Hamiltonianos	16
2.1.2.3	Problema do Caixeiro Viajante	17
2.1.3	Caminhos de Custo Mínimo	17
2.1.4	Conectividade	17
2.1.4.1	Componentes Fortemente Conexas	17
2.1.4.2	Ordenação Topológica	18
2.1.5	Árvores Geradoras Mínimas	18
2.1.6	Fluxo Máximo	18
2.1.7	Emparelhamento Máximo em Grafos Bipartidos	18
2.1.8	Coloração Própria	19
2.1.9	Caminho Crítico	19
2.2	Jogos e Aprendizagem	19
2.3	<i>Game Engine</i>	20
2.3.1	Escolher uma <i>Game Engine</i>	21
2.3.2	Unity	21
2.3.3	Unreal Engine	21
2.3.4	Godot	22
2.4	Large Language Models (LLM)	22
3	TRABALHOS RELACIONADOS	23
3.1	Professor Layton and the Curious Village	23

3.2	Professor Layton and the Pandora Box	25
3.3	Chromagraph	27
3.4	Icosian Game	27
3.5	Desenvolvimento de Jogos com Godot	28
3.6	Comparativo entre os trabalhos	29
4	ESPECIFICAÇÃO DO JOGO	30
4.1	Requisitos Funcionais (RF)	30
4.2	Requisitos Não Funcionais (RNF)	31
5	DESENVOLVIMENTO	32
5.1	Implementação	32
5.1.1	Menu de Questões	32
5.1.2	Editor de Grafo	33
5.1.2.1	Tratamento de Eventos	34
5.1.2.2	Carregamento de Grafo via arquivo	35
5.1.3	Questões	35
5.1.3.1	Avaliação do Resultado	36
5.1.4	LLM	37
5.2	Resultados	38
5.2.1	Menu das Questões	38
5.2.2	Exemplos de Questões	38
5.2.2.1	Construir grafo com Ciclo Euleriano	39
5.2.2.2	Completar grafo para ter um Ciclo Euleriano	40
5.2.2.3	Ciclo Hamiltoniano	41
5.2.2.4	Coloração	41
5.2.2.5	Ordenação Topológica	42
5.2.2.6	Caminho Mínimo	42
5.2.2.7	Árvore Geradora Mínima	43
5.3	Avaliação	44
6	CONCLUSÕES E TRABALHOS FUTUROS	45
	REFERÊNCIAS BIBLIOGRÁFICAS	46
A	ARTIGO	49
B	CÓDIGO FONTE	60

1 Introdução

A área de grafos é de suma importância para a Ciência da Computação, podendo ser utilizada para diversos propósitos devido ao seu grande nível de abstração. Por exemplo, ela foi fundamental na resolução de problemas clássicos como o Caminho Euleriano, que consiste em encontrar uma rota que passe por todas as arestas de um grafo sem repeti-las; este problema surgiu a partir do desafio das Sete Pontes de Königsberg, proposto por Euler (SANTIAGO, 2024, p. 32). Outro exemplo é o problema do Caixeiro Viajante, que busca a rota mais curta que passa por um conjunto de cidades e retorna ao ponto de partida, sem haver repetições de cidade (CORMEN et al., 2012, p. 883). Além disso, há os Problemas de Travessia, que consistem em sistemas no qual há um certo número de estados possíveis (vértices) que transitam entre eles (arestas), sendo o objetivo encontrar um caminho válido até o estado final a partir do inicial; alguns exemplos são os Potes de Vinho e Fuga dos Ladrões (SANTIAGO, 2024). Nesse contexto, destaca-se a disciplina INE5413–Grafos, oferecida no curso de Ciência da Computação da Universidade Federal de Santa Catarina, cuja ementa contempla tópicos como caminhos e ciclos, coloração, árvores geradoras mínimas, fluxo máximo em redes e emparelhamento em grafos (SANTIAGO, 2025). O presente trabalho dialoga diretamente com esse conteúdo, buscando propor uma abordagem alternativa para o ensino desses conceitos por meio de jogos digitais. Dessa forma, a abrangência e a relevância dos grafos na Ciência da Computação tornam essencial a compreensão de seus conceitos. O uso de jogos digitais e a gamificação como metodologia pode ser uma boa abordagem para estimular a motivação e engajamento dos estudantes sobre o assunto e nas tarefas realizadas (NACIONAL, 2023), além de promover ambientes dinâmicos imersivos e favorecer o desenvolvimento de competências sociais e cognitivas, facilitando a compreensão dos conceitos abordados (VIDENOVIK et al., 2023).

Além disso, nota-se que há uma variedade de jogos que abordam, direta ou indiretamente, conceitos da Teoria de Grafos, como coloração, caminhos mínimos, e ciclos Hamiltonianos e Eulerianos. Jogos como da franquia Professor Layton (Layton Fandom, 2025a; Layton Fandom, 2025b), Chromagraph (RUMPF, 2020a) e Icosian Game (NEAMAR, 2010) demonstram o potencial dos *puzzles* em representar problemas clássicos de maneira intuitiva e interativa, proporcionando ao jogador uma experiência que une raciocínio lógico e entretenimento. Essas evidências reforçam o argumento de que jogos podem ser utilizados como ferramentas de auxílio à compreensão de conteúdos complexos, sendo promissores como suporte ao ensino de Grafos na Ciência da Computação.

Sendo assim, o presente trabalho busca desenvolver um jogo eletrônico como método de ensino complementar, adotando uma abordagem didática e interativa. Em alinha-

mento com a disciplina INE5413–Grafos (SANTIAGO, 2025), oferecida no curso superior em Ciências da Computação na Universidade Federal de Santa Catarina, os módulos contemplam problemas clássicos da Teoria de Grafos, com soluções baseadas em algoritmos e teoremas bem estabelecidos na área de ensino.

1.1 Objetivos

Nesta seção, são apresentados o objetivo geral e objetivos específicos deste trabalho.

1.1.1 Objetivo Geral

Desenvolver um jogo educacional para a demonstração de algoritmos e teoremas da área de Grafos, com dinâmicas que estimulem o usuário a visualizar graficamente e compreender o funcionamento dos mesmos.

1.1.2 Objetivos Específicos

Para atingir o objetivo geral, pretende-se cumprir os seguintes objetivos específicos:

- Selecionar os algoritmos e teoremas de Grafos que estarão inclusos no jogo;
- Escolher uma *Game Engine* adequada para o jogo, a partir das opções encontradas no mercado;
- Especificar o jogo;
- Desenvolver o jogo;
- Divulgar dos resultados do trabalho.

1.2 Metodologia

O presente projeto é descritivo e explicativo, visto que consiste na implementação de um jogo educacional para apresentar algoritmos e teoremas da área de Grafos. O desenvolvimento foi executado seguindo as etapas abaixo:

- Primeiramente, foi realizado um levantamento teórico do estado da arte, para descrever os problemas clássicos de Grafos e suas propriedades e apresentar o uso de jogos como método de aprendizagem;
- Após a conclusão da análise teórica, foram elaboradas as especificações do jogo, além de decidir os algoritmos que serão integrados aos *puzzles* do mesmo;

- Com o jogo definido, foi realizado um comparativo entre *Game Engines* disponíveis e suas características, sendo selecionada a mais adequada (open-source, gratuita e compatível com o sistema operacional Linux) para a implementação.
- Por fim, após o jogo ser devidamente implementado, foram documentados os resultados, a implementação e os diagramas relevantes na monografia.

1.3 Estrutura do Trabalho

O documento está dividido em 6 capítulos e estruturado da seguinte forma:

Capítulo 1: Introduz o projeto e determina os objetivos gerais e específicos do mesmo, além de apresentar a metodologia aplicado durante o desenvolvimento;

Capítulo 2: Aborda os conceitos teóricos necessários para a compreensão do trabalho realizado, focando nos problemas clássicos de Grafos, o impacto dos jogos com meio de aprendizado e o uso de *Game Engines* para desenvolvimento de aplicações;

Capítulo 3: Apresenta os trabalhos correlatos, com foco em jogos e que aplicam problemas de grafos para solução de problemas;

Capítulo 4: É feita especificação do jogo a partir dos requisitos funcionais e não-funcionais;

Capítulo 5: É detalhado o processo de implementação da aplicação e os resultados obtidos;

Capítulo 6: Apresenta as conclusões do trabalho e trabalhos futuros.

2 Fundamentação Teórica

Este capítulo aborda conceitos principais cuja compreensão é necessária para o desenvolvimento e entendimento do trabalho.

2.1 Problemas Seleccionados de Grafos

Nesta seção, são discutidos os principais problemas de grafos presentes na disciplina INE5413-Grafos oferecida no curso superior em Ciências da Computação na Universidade Federal de Santa Catarina.

Esta disciplina traz em sua ementa: grafos e grafos orientados, representação de problemas com grafos, caminhos, ciclos e caminho de custo mínimo, conexidade e alcançabilidade, árvores e árvore de custo mínimo, coloração e planaridade de grafos, grafos hamiltonianos e eulerianos, fluxo máximo em redes, estabilidade e emparelhamento em grafos, problemas de cobertura e de travessia, representações computacionais e complexidade de algoritmos em grafos (SANTIAGO, 2025).

2.1.1 Busca em Grafos

Na presente seção, são apresentados dois dos principais algoritmos de busca: a busca em largura, que percorre o grafo em níveis sucessivos, e a busca em profundidade, que explora caminhos até não conseguir mais antes de retroceder.

2.1.1.1 Busca em Largura

A busca em largura (*Breadth-First Search* - BFS) percorre um grafo a partir de um vértice inicial, explorando primeiro todos os seus vizinhos antes de avançar para os vizinhos desses vértices, seguindo uma lógica em nível. Esse método determina a distância mínima, em número de arestas, entre o vértice inicial e os demais vértices atingíveis do grafo. Além disso, permite a construção de uma árvore de antecessores, que representa os caminhos mínimos dessas distâncias (CORMEN et al., 2012, p. 487).

Na Figura 1, é apresentado um exemplo de grafo que foi aplicado a busca em largura a partir do vértice *A*, tendo sua árvore de antecessores na direita.

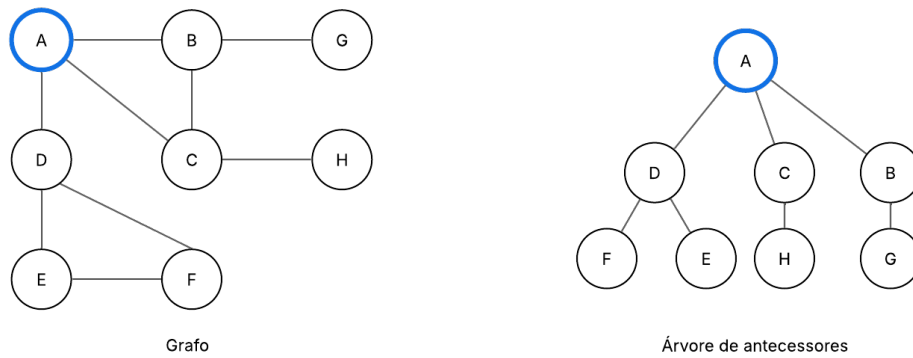


Figura 1 – Exemplo de Busca em Largura

2.1.1.2 Busca em Profundidade

A busca em profundidade (*Depth-First Search* - DFS), diferentemente da busca em largura, é um procedimento que segue por um caminho até que não se haja mais vértices novos. Ao atingir esse limiar, volta para tentar outro caminho, verificando se há algum vértice vizinho que não foi explorado. Esse método também constrói uma árvore de antecessores (CORMEN et al., 2012, p. 494).

A Figura 2 abaixo possui o mesmo grafo apresentado em Busca em Largura, porém nota-se que a sua árvore de antecessores é diferente, mesmo começando do mesmo vértice A.

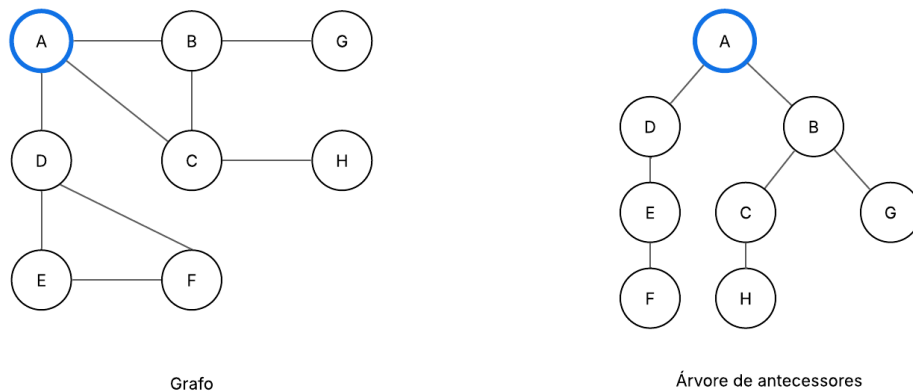


Figura 2 – Exemplo de Busca em Profundidade

2.1.2 Caminhos e Ciclos

Nesta seção, são abordados os conceitos de caminhos e ciclos Eulerianos, que consideram a travessia de arestas, e Hamiltonianos, que se concentra na visita a vértices uma única vez. Também é descrito o problema do Caixeiro Viajante, diretamente ligado aos ciclos Hamiltonianos.

2.1.2.1 Caminhos e Ciclos Eulerianos

Essa classe de problemas envolve grafo não-dirigidos. O caminho euleriano consiste em um trajeto que percorra todas as arestas do grafo, em que nenhuma delas é visitada mais de uma vez. O ciclo euleriano segue essa mesma regra, mas deve finalizar no vértice de origem (SANTIAGO, 2024, p. 32).

Esses problemas podem ser simplificados para um problema de decisão, ou seja, identificar se o grafo possui um caminho ou ciclo euleriano sem determinar um trajeto possível (SANTIAGO, 2024, p. 32).

Na Figura 3 a seguir, há um grafo e o possível ciclo euleriano dele, sendo a ordem das arestas indicadas pelas setas e a numeração, resultando no ciclo $A \rightarrow D \rightarrow C \rightarrow A \rightarrow B \rightarrow E \rightarrow F \rightarrow B \rightarrow C \rightarrow E \rightarrow A$.

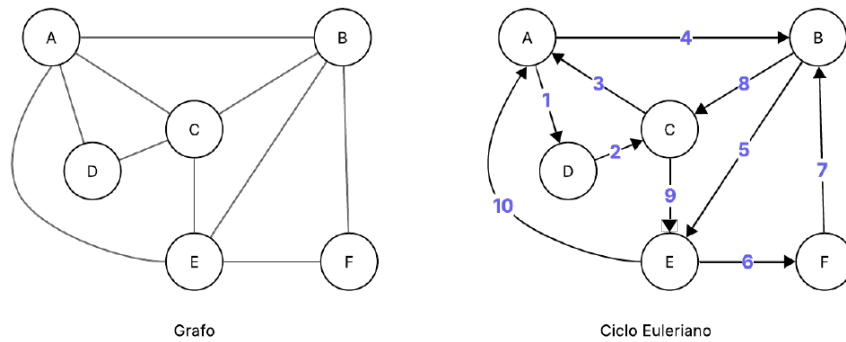


Figura 3 – Exemplo de Ciclo Euleriano

2.1.2.2 Caminhos e Ciclos Hamiltonianos

Diferentemente dos caminhos eulerianos, um caminho hamiltoniano deve percorrer todos os vértices do grafo exatamente uma vez. No caso de um ciclo hamiltoniano, a regra se mantém, mas o trajeto deve terminar no vértice de origem. A identificação desses caminhos e ciclos é um problema computacionalmente complexo, pertencendo à classe de problemas NP-Difíceis (SANTIAGO, 2024, p. 38).

A Figura 4 abaixo possui o mesmo grafo que do exemplo do Ciclo Euleriano, mas agora resultando em Ciclo Hamiltoniano, sendo a ordem das arestas indicadas pelas setas vermelhas, resultando no ciclo $A \rightarrow E \rightarrow F \rightarrow B \rightarrow C \rightarrow D \rightarrow A$.

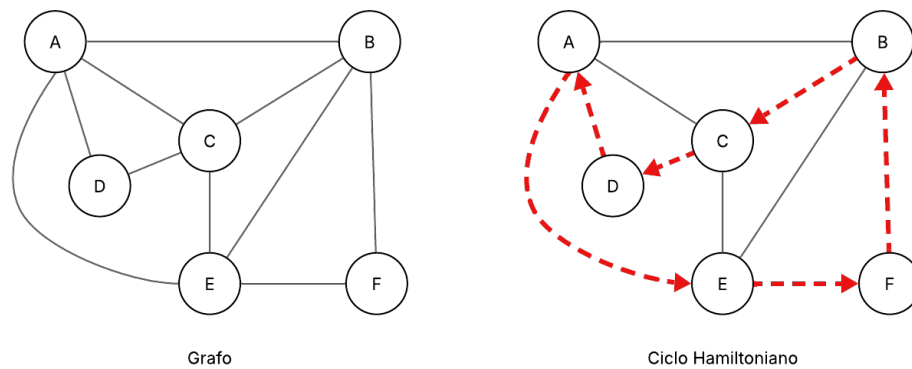


Figura 4 – Exemplo de Ciclo Hamiltoniano

2.1.2.3 Problema do Caixeiro Viajante

No problema do Caixeiro Viajante, o objetivo é, dado um conjunto de cidades e interconexões entre elas, identificar um percurso no qual passa por todas as regiões sem repetições e retorna ao ponto inicial com o menor custo possível. Dessa forma, esse problema está diretamente conectado à busca por um ciclo hamiltoniano de custo mínimo em um grafo, onde os vértices representam as cidades e as arestas, as interconexões (CORMEN et al., 2012, p. 883).

2.1.3 Caminhos de Custo Mínimo

Um caminho de custo mínimo refere-se ao trajeto entre um par de vértices, de forma que a soma dos pesos das arestas seja a menor possível. Esse problema possui algumas variantes dependendo do propósito da busca (CORMEN et al., 2012, p. 526-527).

Nos Caminhos Mínimos de Fonte Única, o objetivo é: dado um grafo ponderado e um vértice inicial, determinar o caminho mais curto para todos os vértices que compõem esse grafo. Já em Caminhos Mínimos para Todos os Pares, busca-se identificar o trajeto mais eficiente entre qualquer par de vértices existentes (CORMEN et al., 2012, p. 527).

2.1.4 Conectividade

Esta seção aborda os conceitos de componentes fortemente conexas em grafos dirigidos, que agrupam vértices mutuamente alcançáveis e mesma propriedade, e a ordenação topológica, que determina uma sequência linear de vértices a partir das relações de precedência.

2.1.4.1 Componentes Fortemente Conexas

Em grafos dirigidos, uma componente fortemente conexa é um subconjunto de vértices no qual, para qualquer vértice da componente, existe um caminho que permite alcançar

qualquer outro vértice dela. Além disso, a componente deve ser maximal, ou seja, todos os vértices do grafo que tenham a mesma propriedade devem pertencer ao conjunto. Dessa forma, todos os vértices da componente estão mutuamente conectados (SANTIAGO, 2024, p. 69).

2.1.4.2 Ordenação Topológica

A Ordenação Topológica é um problema aplicado a grafos dirigidos acíclicos, no qual se busca determinar uma sequência linear dos vértices, garantindo que cada vértice preceda aqueles que dependem dele. Ou seja, dado um grafo G que contém uma aresta $u \rightarrow v$, o u aparecerá antes de v nessa ordenação. Esse procedimento é comumente utilizado para indicar precedência de eventos (CORMEN et al., 2012, p. 501).

2.1.5 Árvores Geradoras Mínimas

A Árvore Geradora Mínima é um conceito que envolve grafos conexos não-dirigidos e ponderados, em que deve-se determinar um subconjunto acíclico de arestas do grafo que forme uma árvore, garantindo que todos os vértices sejam alcançáveis e que a soma dos seus pesos seja a menor possível (CORMEN et al., 2012).

2.1.6 Fluxo Máximo

O problema do fluxo máximo refere-se a um tipo específico de grafo dirigido e ponderado conhecido como rede de fluxo, que possui vértices de origem e destino e o peso das arestas é a capacidade de fluxo que cada uma suporta (PÓLYA; TARJAN; WOODS, 2010, p. 152).

O objetivo é determinar o máximo de fluxo que pode ser transmitido da origem ao destino, sem ultrapassar as capacidades das arestas e garantindo que o fluxo de entrada e saída em cada vértice sejam iguais (PÓLYA; TARJAN; WOODS, 2010, p. 152).

2.1.7 Emparelhamento Máximo em Grafos Bipartidos

O presente problema é aplicado a grafos bipartidos e não-dirigidos, que são um tipo de grafo que possui dois conjuntos disjuntos de vértices, de modo que todas as arestas têm origem em um grupo e destino no outro (SANTIAGO, 2024, p. 93).

Um emparelhamento é um subconjunto de arestas de um grafo, garantindo que cada vértice só esteja conectado a, no máximo, uma aresta. Dessa forma, um emparelhamento máximo tem objetivo de determinar o maior subconjunto possível de arestas independentes (PÓLYA; TARJAN; WOODS, 2010, p. 145).

2.1.8 Coloração Própria

A coloração própria de grafos consiste em atribuir cores aos vértices de maneira que vértices adjacentes tenham cores diferentes. Esse problema possui duas versões: a versão de decisão, que envolve indicar se é possível colorir um grafo com um certo número de cores, e a versão de otimização, que deve-se determinar o mínimo de cores necessárias para realizar a coloração (SANTIAGO, 2024).

Na Figura 5 a seguir, é apresentado um grafo e uma possível combinação de três cores que se enquadram no problema de Coloração Própria. Nota-se que é impossível colorir com menos, visto que o grafo contém pelo menos um triângulo (ciclo de três vértices), formado por A, C e E, por exemplo.

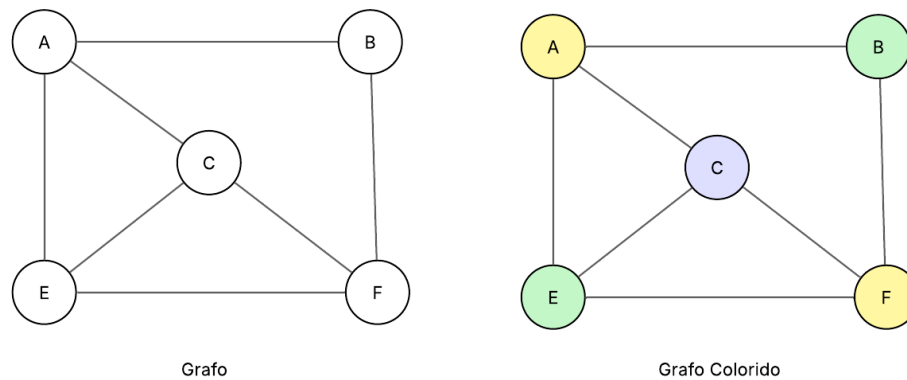


Figura 5 – Exemplo de Coloração

2.1.9 Caminho Crítico

O método do Caminho Crítico é aplicado a grafos dirigidos e ponderados para planejamento de projetos. Nesse problema, arestas representam atividades, os seus pesos indicam o tempo ou recursos para que uma tarefa seja concluída e os vértices correspondem aos possíveis estados do projeto após a conclusão de cada tarefa (SANTIAGO, 2024). Dessa forma, a direção de uma aresta $u \rightarrow v$ indica que, para atingir o estado v , é necessário primeiro estar no estado u e finalizar a atividade correspondente.

2.2 Jogos e Aprendizagem

A constante integração da tecnologia no cotidiano dos estudantes tem impulsionado a busca por abordagens inovadoras que transcendam os métodos tradicionais, visando otimizar o processo de ensino e aprendizagem. Nesse contexto, a integração de elementos dos jogos ao ambiente educacional emerge como uma estratégia promissora, com capacidade para engajar e aprimorar competências cognitivas e sociais dos estudantes.

De acordo com Videnovik et al. (2023), a aprendizagem baseada em jogos tem recebido uma atenção crescente nos últimos anos por ser uma ferramenta poderosa para aprimorar a motivação e o engajamento dos estudantes. Essa abordagem proporciona um aprendizado mais agradável, eficiente e centrado no aluno, promovendo ambientes imersivos e personalizados, nos quais os estudantes podem aprender no seu próprio ritmo. Além disso, essa metodologia tem o potencial para desenvolver competências como resolução de problemas, criatividade, planejamento estratégico e tomada de decisão, além de contribuir positivamente ao desempenho acadêmico.

Sob uma perspectiva semelhante, Johnson et al. (2016) destaca que jogos podem auxiliar no aprimoramento do ensino em diversas áreas, especialmente na Ciência da Computação. Nessa disciplina, o uso de jogos permite que os estudantes apliquem conceitos de forma prática e desenvolvam habilidades como autoeficácia, comunicação e liderança. Além disso, os jogos educacionais são projetados para fornecer *feedback* imediato, o que potencializa a fixação do conteúdo, promover um senso de progressão e contemplar diferentes níveis de dificuldade.

Sob uma outra visão, Nacional (2023) aborda o conceito de gamificação, definido como o processo de integração de elementos e dinâmicas de jogos (como pontuações, níveis e premiações) para contextos não relacionados diretamente a jogos. No âmbito educacional, essa técnica pode motivar o estudante a atingir os objetivos de aprendizado, além de incentivar a colaboração e interação social entre os alunos.

Por fim, por meio da revisão apresentada por Silva, Masaro e Paula (2024), observa-se que a gamificação tem contribuído significativamente para aprimorar o aprendizado no ensino superior. Essa abordagem estimula o foco do aluno sobre as tarefas requisitadas e o proporciona segurança para falhar por ter a possibilidade de tentar novamente, tornando um ambiente mais atrativo e desafiador, mas que reduz a evasão escolar. Além disso, relata-se que ambientes relacionados a tecnologia, como na Programação, por exigir o uso de instrumentos digitais, estão mais propícios para a aplicação da gamificação.

A partir do relatado, nota-se que a inserção de jogos e o uso da gamificação ao contexto educacional influencia no aprendizado dos alunos. Além de promover ambientes dinâmicos, personalizados e interativos, essas abordagens estimulam não apenas o engajamento e a motivação dos estudantes, mas também favorecem o desenvolvimento de competências sociais e cognitivas. Dessa forma, conclui-se que tais estratégias representam não apenas uma inovação metodológica, mas também uma resposta eficaz às demandas contemporâneas da educação.

2.3 *Game Engine*

Uma *Game Engine* é um ambiente de desenvolvimento para jogos, fornecendo a base do que é necessário na maioria dos jogos, como: um motor gráfico para renderização de

modelos e estruturas em tempo real; motor de física para lidar com as interações entre objetos e ambiente, como em detecção de colisões e realização de movimentos; sistema de áudio, permitindo indicações sonoras como ao abrir uma porta ou os passos de um personagem; estrutura de *script* para criação de lógica e comportamento dos elementos do jogo, como, por exemplo, o que deve acontecer ao atingir um inimigo ou ao clicar num botão; entre outras funcionalidades (CHARLEAUX; TOLEDO, 2025). Com esses recursos, os desenvolvedores podem se concentrar nos aspectos exclusivos do seu jogo, otimizando o tempo de desenvolvimento.

2.3.1 Escolher uma *Game Engine*

A decisão de qual *Game Engine* será utilizada no desenvolvimento depende de muitos critérios, como custos de uso, plataformas alvo do produto final, linguagem de programação suportada, curva de aprendizado e a capacidade da ferramenta em atender os requisitos do jogo. Por isso, é importante ter uma visão geral das *Game Engines* disponíveis no mercado e avaliar qual será mais adequado para o projeto.

O estudo comparativo realizado por Dragonfly (2025) entre 93 *Game Engines* analisou os critérios mencionados, além das vantagens, desvantagens e escopo ideal de projeto para cada uma. Dentre as opções avaliadas, destacam-se Unity, Unreal Engine e Godot, sendo as duas primeiras as mais utilizadas do mercado, de acordo com SteamDB (2025), enquanto Godot sobressai entre as opções gratuitas e *open-source*.

2.3.2 Unity

A Unity teve sua primeira versão lançada em junho de 2005, com o objetivo de ser uma *game engine* acessível e para diferentes níveis de conhecimento (MEDIUM, 2023). Os projetos são desenvolvidos na linguagem C#, com suporte a gráficos 2D e 3D (DRAGONFLY, 2025) e abrangendo mais de 20 plataformas, incluindo Linux (UNITY, 2025). Embora seja gratuito, consumidores com maiores receitas necessitam de um plano pago específico (BROMBERG, 2024).

2.3.3 Unreal Engine

A Unreal Engine foi lançada oficialmente em 2000, dois anos após a demonstração de um jogo multijogador Unreal Tournament, desenvolvido pelos próprios criadores na ferramenta (PREVENTIONWORKSVT, 2024). Embora as versões atuais sejam voltadas para gráficos 3D com alto nível de realismo, a *engine* também oferece suporte a projetos 2D. Seu ambiente de desenvolvimento permite a utilização das linguagens C++ e Blueprint (DRAGONFLY, 2025). Assim como a Unity, a Unreal é gratuita, mas impõe um limite

de receita, necessitando pagar 5% de *royalties* sobre o lucro de projetos que atingem um faturamento de 1 milhão de dólares (UNREAL ENGINE, 2025).

2.3.4 Godot

A Godot Engine foi lançada oficialmente em 2014 e se consolidou como uma das ferramentas *open-source* mais populares para o desenvolvimento de jogos. Embora ofereça suporte a gráficos 3D, é amplamente reconhecida por sua *engine* dedicada para jogos 2D (DRAGONFLY, 2025). Diferentemente da Unity e Unreal Engine, é uma ferramenta gratuita, de código aberto, distribuída sob a licença MIT, permitindo uso irrestrito tanto em projetos pessoais quanto comerciais. O ambiente de desenvolvimento suporta múltiplas linguagens, sendo as principais GDScript (linguagem própria da ferramenta), C# e C++; além disso, é possível integrar outras linguagens por meio de extensões. A Godot permite exportação para diversas plataformas, incluindo Windows, Linux, sistemas de dispositivos móveis e um suporte externo para consoles (GODOT FAQ, 2025). A leveza da ferramenta, sua comunidade ativa e o código aberto tornam a Godot uma escolha atrativa para desenvolvedores independentes e projetos educacionais (DRAGONFLY, 2025).

2.4 Large Language Models (LLM)

Os Large Language Models (LLMs) representam sistemas de inteligência artificial sofisticados, preparados para lidar com a linguagem humana. Eles conseguem entender comandos escritos e criar textos lógicos, executando tarefas como resumo, explicação e avaliação de textos (PRADO, 2024).

No contexto da educação, os LLMs têm sido estudados como recursos de assistência, oferecendo *feedback* imediato e colaborando no aprendizado de temas complexos. Estudos mostram que tais modelos podem auxiliar na individualização do ensino, entregando observações detalhadas que incentivam a análise do aluno e impulsionam o desenvolvimento de competências cognitivas (LIMA-LOPES, 2025).

Assim, os LLMs se estabelecem como instrumentos que expandem o contato entre o estudante e a disciplina, viabilizando ações de ensino mais ativas e participativas, sem eliminar técnicas de avaliação já conhecidas, mas funcionando como método complementar.

3 Trabalhos relacionados

Neste capítulo, são apresentados dois jogos relacionados a teoria de grafos, 3.3 e 3.4, encontrados na *web* pelas palavras-chave "graph+theory+games". Além disso, são incluídos dois jogos da franquia Professor Layton, 3.1 e 3.2, por serem conhecidos pelo foco em *puzzles*. Ainda, é apresentado um projeto usando Godot para o desenvolvimento, 3.5, visto que será a *game engine* utilizada no presente trabalho. Por fim, é feito um resumo comparativo entre os trabalhos expostos (3.6).

3.1 Professor Layton and the Curious Village

O jogo “Professor Layton and the Curious Village” foi lançado em 15 de fevereiro de 2007, no Japão, para o Nintendo DS, sendo desenvolvido pela Level-5 e distribuído pela Nintendo. A trama do jogo está em volta de uma fictícia e enigmática vila chamada St. Mystere, onde o investigador Professor Layton e seu discípulo, Luke, viajam para encontrar a *Golden Apple*, uma misteriosa herança de Barão Reinhold, além de desvendar os demais segredos do vilarejo (Layton Fandom, 2025a).

A jogabilidade é focada em *puzzles*, que são encontrados explorando a cidade ou são requisitados pelos habitantes, sendo alguns necessários para o avanço da história. Apresentando cerca de 130 problemas ao longo do jogo, é notável que alguns deles podem ser solucionados utilizando conceitos da teoria de Grafos (STRATEGYWIKI, 2022).

A Figura 6 (LEVEL-5, 2007a) abaixo é referente ao *puzzle* 38, que consiste em um conjunto de ilhas conectadas por pontes, sendo o objetivo indicar uma nova conexão necessária entre duas ilhas de forma que todas sejam alcançadas sem repeti-las. Sob essa perspectiva, é evidente que esse problema pode ser reduzido à inserção de uma aresta a um grafo, de modo que exista um caminho hamiltoniano (Figura 7).



Figura 6 – *Puzzle 38* do Professor Layton and the Curious Village

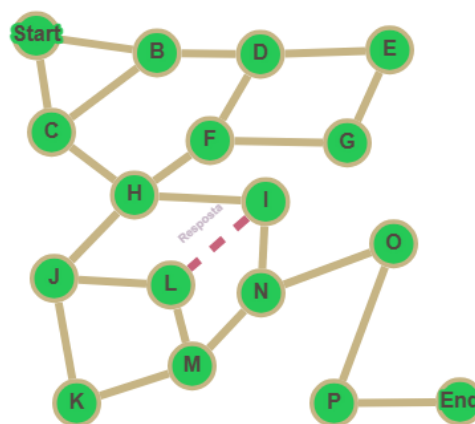


Figura 7 – Grafo equivalente ao *puzzle 38*

Em relação a Figura 8 (LEVEL-5, 2007a), encontra-se a questão 92 do jogo, que deve-se abrir o mínimo de válvulas possível para que a água alcance o balde. Nesse sentido, percebe-se que se equivale a um problema de caminho mínimo, em que as válvulas e o balde representam os vértices e a conexão dos canos, as arestas com custo 1.

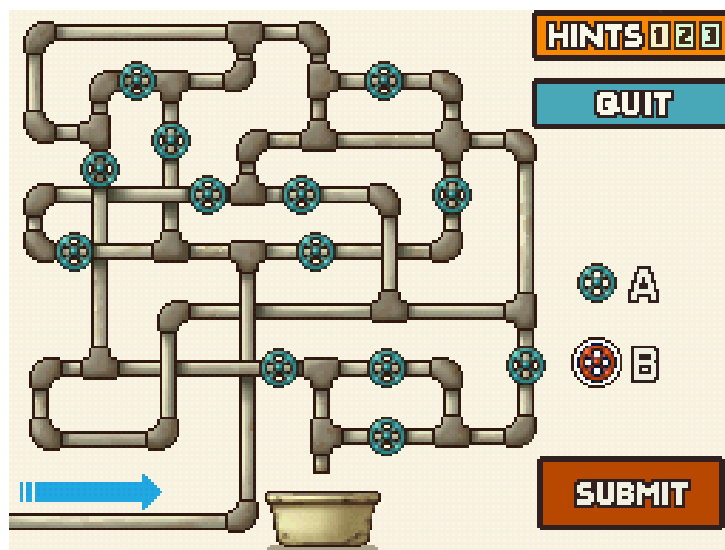


Figura 8 – *Puzzle 92* do Professor Layton and the Curious Village

Por fim, um terceiro exemplo encontrado é o *puzzle 39* (Figura 9), em que deve ser identificado quais dos desenhos apresentados não é possível passar por todo o contorno sem levantar o lápis do papel (LEVEL-5, 2007a). É um problema que pode ser resolvido a partir de uma das propriedades do caminho euleriano: um grafo conexo $G = (V, A)$ admite caminho euleriano se, e somente se, todos os vértices tiverem grau par ou, apenas dois tiverem grau ímpar (PÓLYA; TARJAN; WOODS, 2010, p. 165). Considerando essa propriedade e assumindo que cada cruzamento entre linhas (arestas) do desenho seja um

vértice, é possível determinar que a resposta seria a cartola, que possui 4 vértices de grau ímpar (Figura 10).

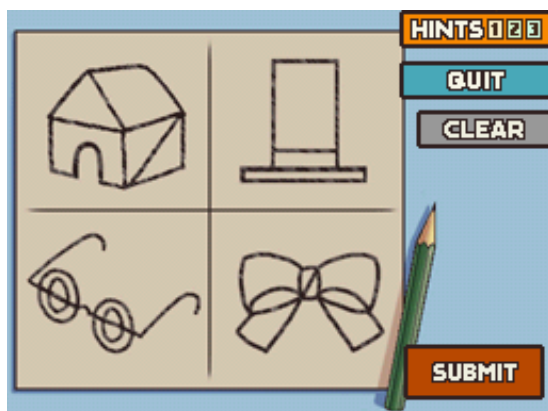


Figura 9 – *Puzzle 39* do Professor Layton and the Curious Village

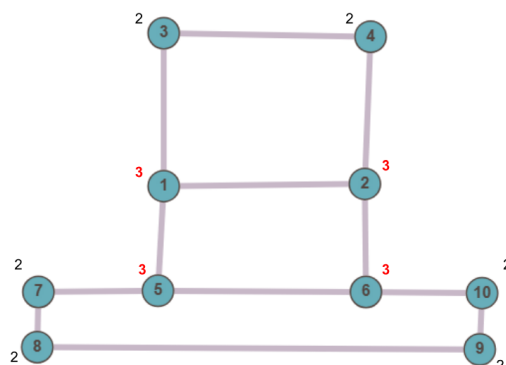


Figura 10 – Grafo equivalente ao desenho da cartola no *puzzle 39*

3.2 Professor Layton and the Pandora Box

“Professor Layton and the Pandora Box” foi o segundo título da franquia, sendo lançado em 29 de novembro de 2007, desenvolvido pela Level-5 para Nintendo DS e publicado pela Nintendo, assim como o primeiro. O enredo acompanha Layton e seu aprendiz, Luke, que estão encarregados de investigar o mistério da *Elysian Box*, uma relíquia com poderes letais para aqueles que decidirem abri-la (Layton Fandom, 2025b).

Assim como no jogo anterior, a jogabilidade é organizada em torno da resolução de *puzzles* que surgem ao longo da história e são encontrados nos cenários pelo jogador. Entre os quase 150 disponíveis (STRATEGYWIKI, 2021), nota-se que alguns podem ser solucionados ao serem reduzidos a problemas da Teoria de Grafos.

A Figura 11 (LEVEL-5, 2007b) representa o décimo problema do jogo, em que o objetivo é determinar o menor número de cores que podem ser usados para pintar o desenho de forma que não exista cores adjacentes iguais. Nesse sentido, percebe-se que esse *puzzle* pode ser resolvido como um problema de coloração de Grafos, representando cada seção do quadro como um vértice e as arestas sendo as divisões entre as seções (Figura 12).



Figura 11 – *Puzzle 10* do Professor Layton and the Pandora Box

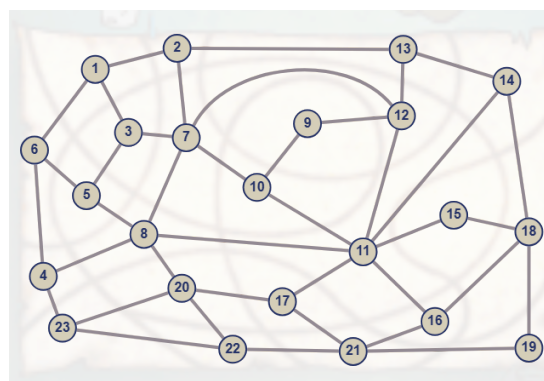


Figura 12 – Grafo equivalente ao *puzzle 10*

Também há o *puzzle 125* na Figura 13 (LEVEL-5, 2007b), em que o alvo é passar por todas as áreas azuis claras sem repetições e no menor trajeto possível, começando no ponto vermelho e finalizando no amarelo. Esse problema é similar ao Caixeiro Viajante, mas com a diferença que não é necessário retornar ao ponto de origem.

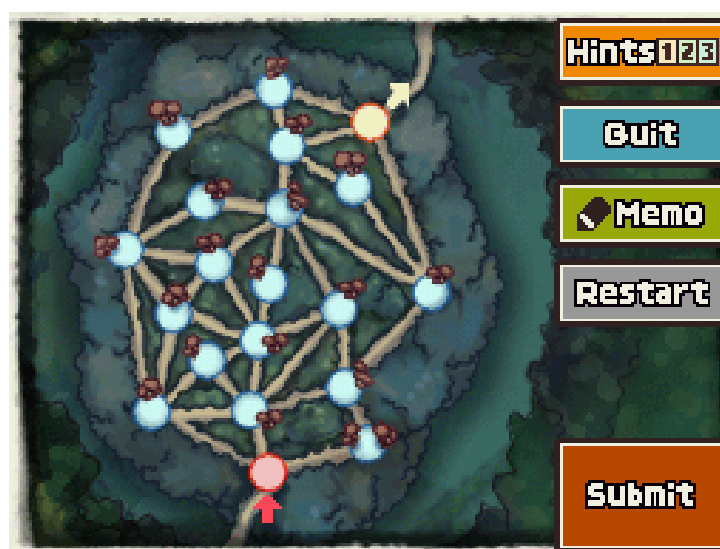


Figura 13 – *Puzzle 125* do Professor Layton and the Pandora Box

Por fim, é importante ressaltar que o jogo dispõe de uma ferramenta interessante para a realização de anotações durante a resolução do *puzzle*, como visto na Figura 14 e 15 (LEVEL-5, 2007b). Essa funcionalidade adiciona uma camada acima à tela do problema, possibilitando o usuário a escrever e desenhar. Isso permite uma melhor visualização gráfica do raciocínio do jogador, facilitando a descoberta da solução.



Figura 14 – Pandora Box: Puzzle originalmente



Figura 15 – Pandora Box: Puzzle com anotações habilitada

3.3 Chromagraph

O jogo *Chromagraph*, desenvolvido por Adam Rumpf na *game engine* *Gamemaker* para a plataforma Windows (RUMPF, 2020a), apresenta diversos *puzzles* referentes a problemas de coloração de grafos, como: Coloring, Edge Coloring, Dominating Set, entre outros (RUMPF, 2020b).

O objetivo de cada nível depende do tipo de problema abordado. Por exemplo, no caso da Figura 16, é necessário colorir os vértices de forma que não haja adjacências com a mesma cor, utilizando o menor número de cores possível (que é indicado pelo jogo). Já na Figura 17, nota-se que agora a coloração é feita nas arestas.

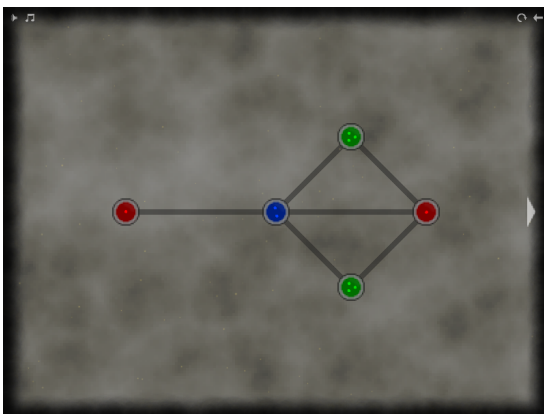


Figura 16 – Chromagraph: nível usando coloração por vértices

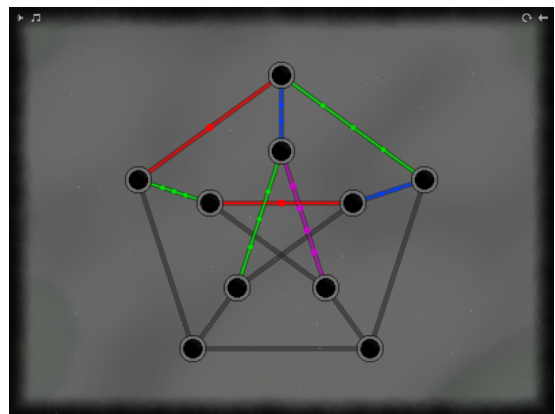


Figura 17 – Chromagraph: nível usando coloração por arestas

3.4 Icosian Game

O *Icosian Game* é um jogo feito em 1857 pelo matemático William Rowan Hamilton, com objetivo de passar por todos os vinte vértices de um dodecaedro, sem repetição, e

retornar ao ponto de partida, ou seja, encontrar um ciclo hamiltoniano (ANTONICK, 2014).

Em 2010, um desenvolvedor, inspirado no jogo original, criou a sua versão digital em Flash, com o mesmo nome. Contudo, esta apresentava algumas variações: são 20 diferentes grafos, sendo que os 9 primeiros exigem em encontrar um caminho euleriano enquanto os demais seguem a lógica do *Icosian* original, buscar um ciclo hamiltoniano (NEAMAR, 2010).

Abaixo, é possível observar dois níveis que são apresentados no jogo. O primeiro (Figura 18) é referente ao sexto problema, em que busca-se um caminho euleriano possível. Já o segundo (Figura 19) corresponde ao *puzzle* 12 do jogo, com o objetivo de determinar um ciclo hamiltoniano.

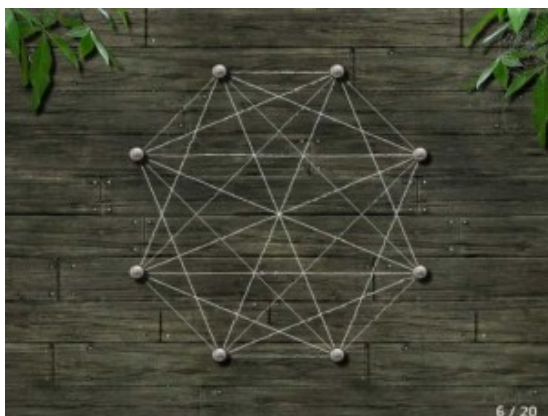


Figura 18 – Nível 6 do Icosian Game

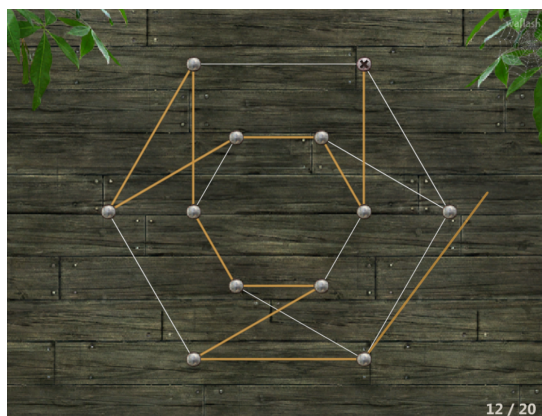


Figura 19 – Nível 12 do Icosian Game

3.5 Desenvolvimento de Jogos com Godot

“Game development using the open-source Godot Game Engine” é o título de uma tese de bacharelado da Universidade de Ciências Aplicadas de Tampere, na Finlândia. O seu objetivo principal é avaliar a capacidade do Godot como ferramenta para desenvolvimento de jogos, descrevendo suas funcionalidades e comparando-a com outras *game engines* existentes no mercado, com foco na Unity e Unreal Engine (SALMELA, 2022).

A tese também compõe de uma parte prática do projeto, na qual foi desenvolvido um jogo chamado Island Assault. Nesta seção, são detalhadas aspectos como: a estrutura do projeto, as decisões de *design* realizadas, a lógica dos *scripts* criados para funcionamento do jogo, os elementos da jogabilidade (controles, inimigos, *layout* das fases, etc), como também a interface do usuário, áudios e persistência dos dados salvos (SALMELA, 2022, p. 27–52).

Por fim, o autor discute sobre o uso do Godot no contexto de jogos independentes, analisando como as suas funcionalidades no 2D e 3D podem atrair ou afastar desenvolvedores da ferramenta. Além disso, é relatada a experiência durante a execução do projeto,

destacando os aspectos positivos, como a criação de novos inimigos e uso do *debugger*, e dificuldades encontradas, como problemas de áudio e organização da interface de usuário. Para concluir, o autor aborda suas expectativas sobre o futuro do Godot, detalhando as melhorias que estão sendo desenvolvidas e novas versões previstas para a ferramenta (SALMELA, 2022, p. 53–60).

3.6 Comparativo entre os trabalhos

A seguir apresenta-se um comparativo entre os trabalhos relacionados descritos nesse capítulo (Tabela 1), focando na cobertura de conceitos de teoria de grafos, abordagens pedagógicas e possíveis limitações.

Trabalho	Tipo	Conceitos de Grafos Cobertos	Pontos Fortes	Limitações
Professor Layton and the Curious Village	Jogo	Caminho e Ciclo Euleriano; caminhos mínimos; redução de desenhos a grafos	Alta capacidade de motivação; puzzles integrados à narrativa incentivam exploração	Não é voltado explicitamente ao ensino formal
Professor Layton and the Pandora Box	Jogo	Coloração; caminhos hamiltonianos; variantes do Caixeiro Viajante	Ferramentas de anotação; variedade de puzzles que estimulam raciocínio visual	Mesmas limitações educacionais formais; pouco controle sobre conceitos algorítmicos
Chromagraph	Jogo	Coloração por vértices e arestas; Dominating Set	Foco direto em problemas formais; níveis com objetivos claros	Plataforma fechada (Windows); foco restrito a classes de problemas de coloração
Icosian Game	Jogo / Puzzle	Ciclos hamiltonianos; caminhos eulerianos	Excelente para exemplificar problemas clássico; níveis progressivos	Pouco acessível; Jogabilidade simples; cobertura limitada
Desenvolvimento de Jogos com Godot	Trabalho acadêmico	—	Fornecer evidências práticas sobre viabilidade do Godot; guia técnico para implementação	Não é um jogo didático por si só; não foca em design instrucional para grafos

Tabela 1 – Comparativo entre os trabalhos relacionados

4 Especificação do Jogo

Esta seção detalha a especificação do *software* desenvolvido. A especificação foi elaborada a partir das reuniões realizadas com o orientador ao longo do projeto e segue um modelo de análise de Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF).

4.1 Requisitos Funcionais (RF)

Os Requisitos Funcionais descrevem as funcionalidades e comportamentos que o sistema deve prover ao usuário.

- **RF1: Menu Principal e Seleção de Módulos:** O sistema deve apresentar uma tela principal que exibe a lista de questões disponíveis. O carregamento da lista deve ser dinâmico, baseado em um JSON, cada exercício possuindo um título, imagem e o arquivo de referência para carregamento.
- **RF2: Editor de Grafo Interativo:** O núcleo do jogo deve ser um editor de grafos que permita ao usuário interagir com a estrutura. O editor deve suportar:
 - Adição e remoção de vértices e arestas.
 - Suporte a grafos dirigidos e não dirigidos.
 - Suporte a grafos ponderados, com edição de peso nas arestas.
- **RF3: Carregamento de Grafos:** O sistema deve ser capaz de carregar definições de grafos (vértices, arestas e pesos) e interface (posição na tela) a partir de arquivos externos em formato JSON.
- **RF4: Módulos de Problemas de Grafos:** O jogo deve conter módulos de questões baseados na disciplina de Grafos, INE5413, do curso superior em Ciências da Computação na Universidade Federal de Santa Catarina (seção 2.1).
- **RF5: Interface de Questão:** Cada módulo deve apresentar uma interface base composta por:
 - Um enunciado claro da questão.
 - Um campo para resposta objetiva.
 - Um campo para resposta discursiva.
 - Um campo para feedback da avaliação.
 - Um botão para retornar ao Menu Principal.

- **RF6: Avaliação de Resposta Objetiva:** O sistema deve validar programaticamente a resposta objetiva do usuário (ex: validar se o grafo construído tem a propriedade exigida ou se os valores colocados estão coerentes com o problema). A lógica de verificação é específica de cada módulo.
- **RF7: Avaliação de Resposta Discursiva (IA):** O sistema deve utilizar uma integração com um Modelo de Linguagem Amplo (LLM) para avaliar a justificativa textual do usuário. O sistema deve construir um *prompt* contendo o enunciado, a resposta do usuário e o grafo, solicitando a IA uma nota e explicação da mesma.

4.2 Requisitos Não Funcionais (RNF)

Os Requisitos Não Funcionais definem os critérios de qualidade e operação do sistema.

- **RNF1: Plataforma de Desenvolvimento:** O jogo deve ser desenvolvido utilizando a *game engine* Godot, em versão 4.5 ou superior.
- **RNF2: Linguagem de Programação:** A lógica do jogo deve ser implementada primariamente em GDScript, linguagem nativa da Godot.
- **RNF3: Compatibilidade:** O sistema deve ser executável em computadores com sistema operacional Linux que suporte a ferramenta Godot.
- **RNF4: Usabilidade:** A interface deve ser clara e intuitiva, separando a área de visualização e interação do grafo da área de texto (enunciado, respostas e feedback).
- **RNF5: Extensibilidade:** A arquitetura deve permitir a fácil adição de novas questões.

5 Desenvolvimento

Este capítulo detalha o processo de implementação do jogo educacional na *game engine* Godot. A seção está dividida em três partes: Implementação, que foca nos aspectos técnicos, arquitetura de cenas e *scripts*; Resultados, que apresenta o produto visual final; e Avaliação, que relata como foi validado da aplicação.

5.1 Implementação

Esta seção descreve a arquitetura técnica do projeto, detalhando como as cenas da Godot, os *scripts* e os arquivos de dados interagem para criar a funcionalidade do jogo.

5.1.1 Menu de Questões

O menu principal é o ponto de entrada do jogo. Para permitir que novas questões sejam adicionadas ao longo do desenvolvimento sem a necessidade de alterar o código do menu, foi implementado um sistema de carregamento dinâmico. Ao iniciar a cena, é lido um arquivo JSON (Figura 20), que define uma lista de questões, onde cada uma especifica o caminho para a cena da questão (`path`), uma imagem de exibição (`image`) e o título (`title`).

```
"questions": [
  {
    "path": "res://questions/question_euleriano.tscn",
    "image": "res://icon.svg",
    "title": "Ciclo Euleriano"
  },
  {
    "path": "res://questions/question_hamiltoniano_completo.tscn",
    "image": "res://svgs/vertex_dragged.svg",
    "title": "Ciclo Hamiltoniano"
  },
  {
    "path": "res://questions/question_euleriano_compleStar.tscn",
    "image": "res://svgs/vertex_selected.svg",
    "title": "Ciclo Euleriano Completar"
  },
]
```

Figura 20 – JSON com a descrição das questões

Na Figura 21 a seguir, é apresentada a árvore da cena que é carregada na inicialização do jogo. Destaca-se o nó `grid_thumbs`, que será o encarregado de apresentar até 6 questões

em uma matriz 2 x 3. Caso haja mais problemas que é suportado, é possível trocar de página pelos botões `btn_prev` e `btn_next`.

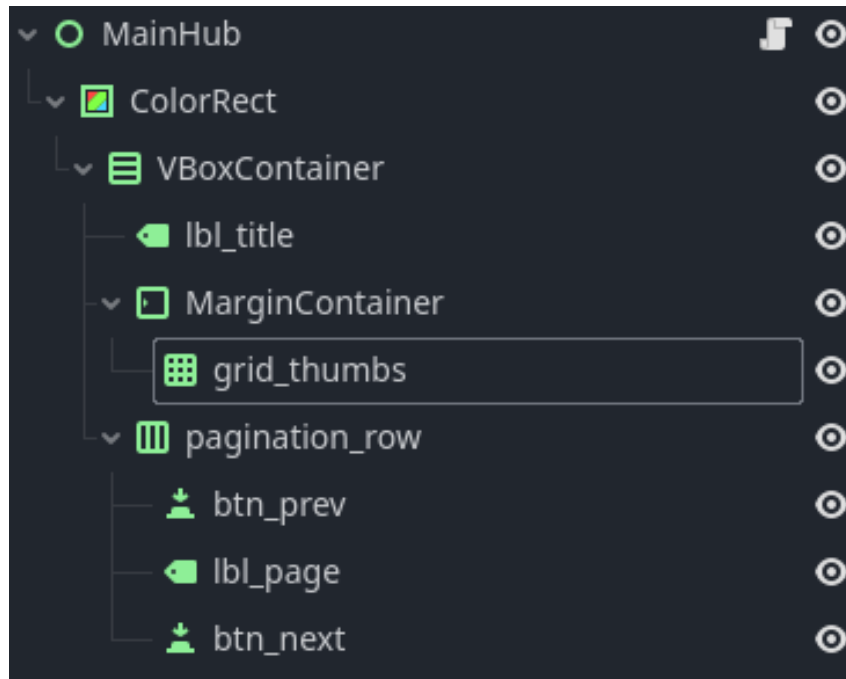


Figura 21 – Cenário do Menu de Questões

5.1.2 Editor de Grafo

Esse componente é responsável pela lógica de interação, manipulação e renderização de grafos. Na Figura 22 a seguir, é apresentada a cena que é instanciada nas questões a serem resolvidas pelo usuário. Dentre os nós observados, destaque-se os `EdgesBatch` e `ArrowsBatch`, responsáveis pela renderização das arestas (e setas, em grafos dirigidos). Também nota-se o nó `ModeOption`, um menu que permite ao aluno alternar entre os modos `ADD_EDIT`, para criação e edição de vértices e arestas, e `DELETE`, para exclusão. Além disso, o `WeightsContainer` e `WeightEditor` são nós para renderização e edição de pesos de grafos ponderados, respectivamente.

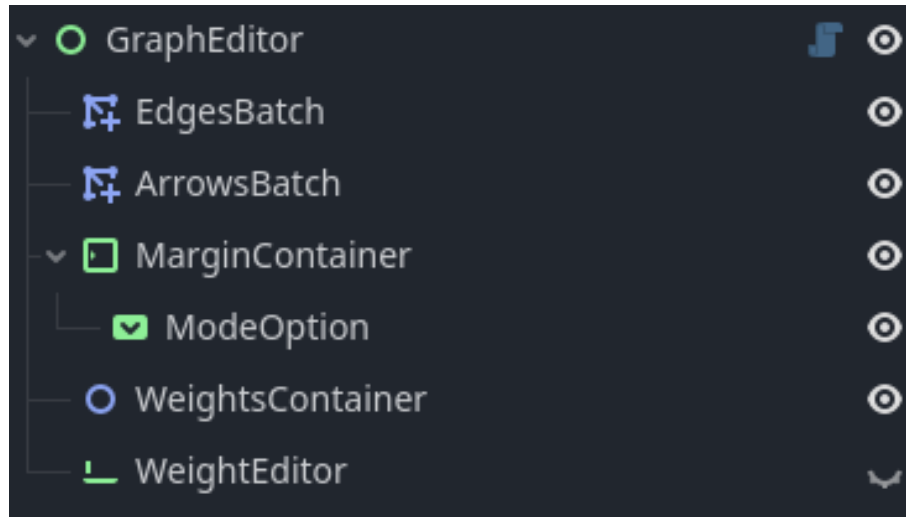


Figura 22 – Cenário do Editor de Grafos

5.1.2.1 Tratamento de Eventos

O comportamento das interações do usuário é gerenciado por uma máquina de estados. Conforme apresentado na Figura 23 abaixo, há quatro estados: IDLE (ocioso), DRAG_VERTEX (mover vértice), CONNECT_EDGE (conectar vértices pela criação de uma aresta) e EDITING_WEIGHT (editar peso de uma aresta). A transição entre eles só ocorre quando está no modo de adição e edição (ADD_EDIT). Contudo, se estiver no modo de deletar (DELETE), será mantido o estado IDLE.

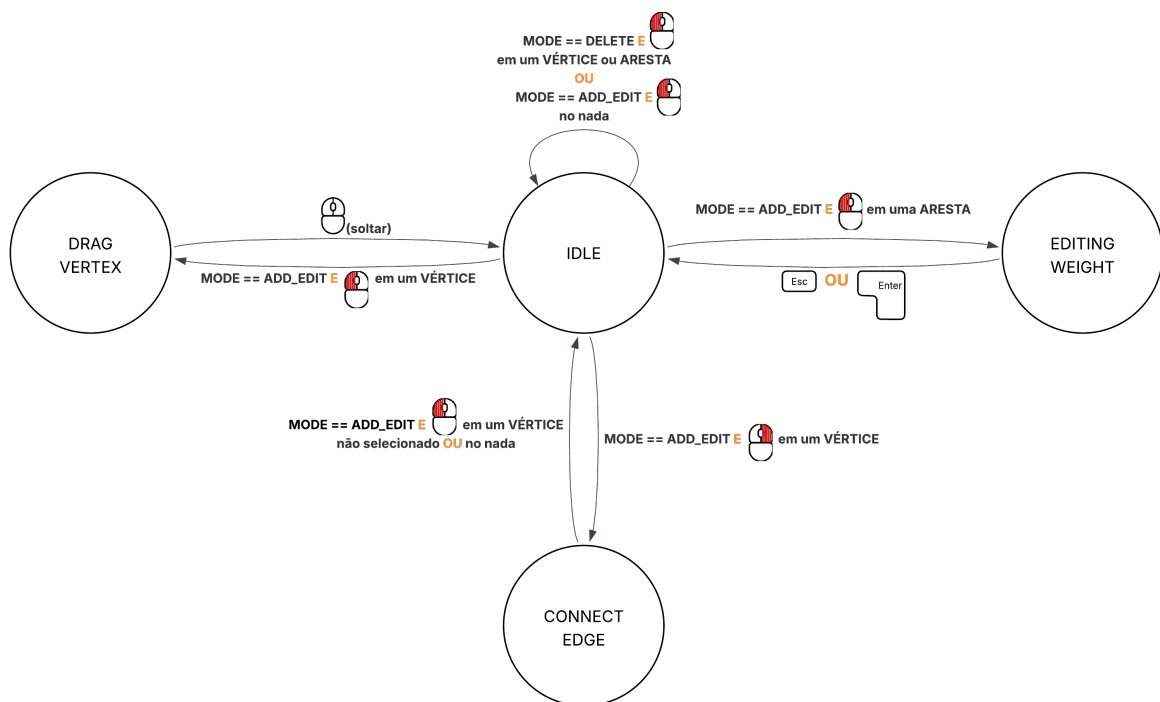


Figura 23 – Máquina de Estados para Tratamento de Interações ao Editor de Grafo

5.1.2.2 Carregamento de Grafo via arquivo

Para facilitar a criação de questões com grafos prontos, é possível carregar via um arquivo JSON. Como visto na Figura 24 abaixo, o documento é separado em três campos: **edges** (lista de arestas com o identificador dos vértices de origem e destino e o peso), **meta** (armazena o se é dirigido ou não, se possui pesos e o *id* do próximo vértice a ser criado) e **vertices**. Esse último apresenta posições *x* e *y* normalizadas entre valores de 0 a 1, tornando-se independente ao espaço real que o editor de grafos ocupa da tela, permitindo uma melhor flexibilidade da estrutura.

```
{
  "edges": [
    {
      "from_id": 1,
      "to_id": 0,
      "weight": 10
    },
    {"...": "..."}
  ],
  "meta": {
    "graph_type": "DIRIGIDO",
    "next_id_vertex": 4,
    "ponderado": true
  },
  "vertices": [
    {
      "id": 0,
      "pos_x": 0.4713541666666667,
      "pos_y": 0.28858024691358
    },
    {
      "id": 1,
      "pos_x": 0.7100694444444444,
      "pos_y": 0.283950617283951
    },
    {"...": "..."}
  ]
}
```

Figura 24 – Arquivo para carregamento de um grafo

5.1.3 Questões

Cada módulo de problema é uma cena independente que herda seu *script* de uma classe base, **Question Abstract**. Esta classe abstrata indica os componentes essenciais da interface que se repetem. A Figura 25 abaixo exemplifica uma das questões desenvolvidas,

sendo a parte selecionada em azul comumente repetida entre as demais. Dessa forma, os problemas geralmente requisitam um **Enunciado**, um campo de **Resposta Discursiva** e **IA/Feedback** e um nó auxiliar para comunicação com a LLM (explicado em 5.1.4).

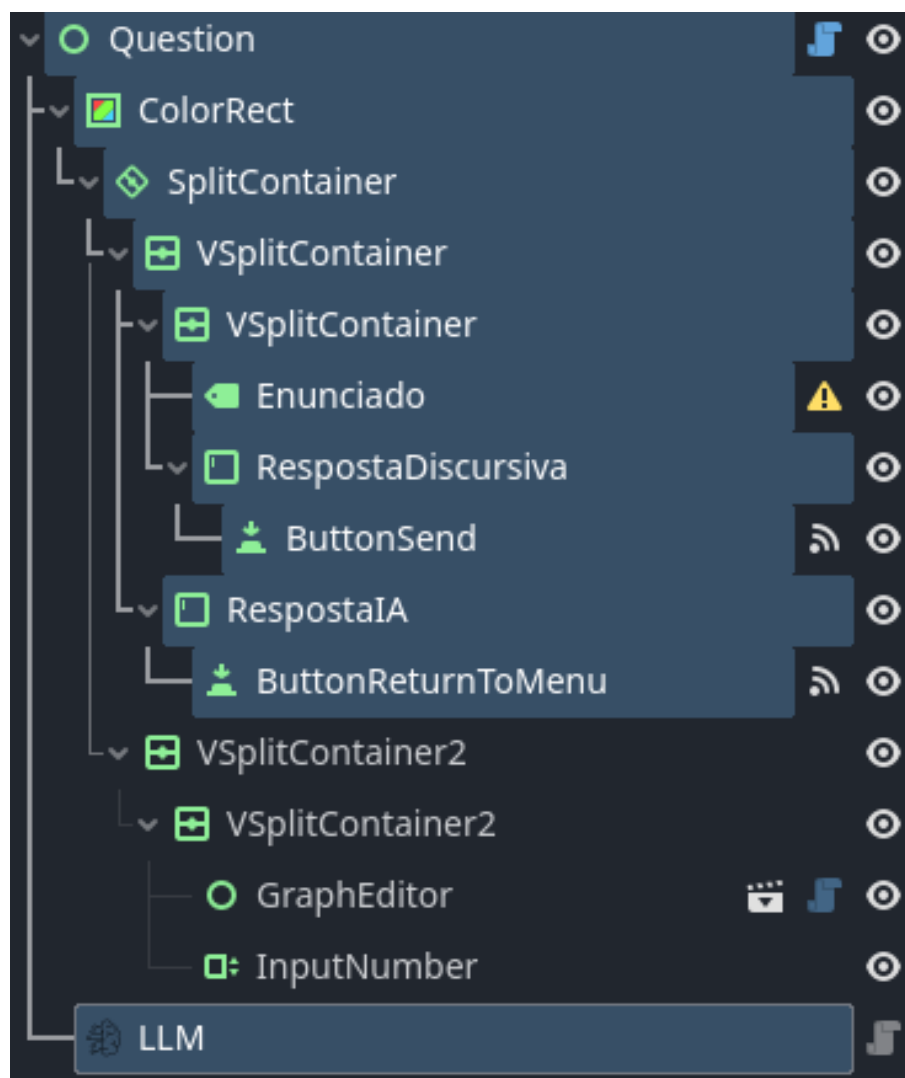


Figura 25 – Exemplo de Cena de uma Questão

5.1.3.1 Avaliação do Resultado

A avaliação da resposta é dividida em duas etapas:

1. **Avaliação Objetiva:** Cada *script* de questão implementa sua própria versão da função `avaliar_resposta()`. Esta função verifica a correção da parte objetiva, seja pelo uso de uma classe auxiliar (`GraphHelper`) que implementa o algoritmo de um problema clássico ou pela verificação direta quando as questões são estáticas.
2. **Avaliação Discursiva:** Caso a resposta objetiva esteja correta, o *script* da questão chama a função `set_ia_feedback()`. Esta função que inicia o processo de avaliação da justificativa textual do aluno usando a LLM.

Na Figura 26 a seguir, há um exemplo de questão para o problema de Árvores Geradoras Mínimas. Nota-se que a avaliação da resposta chama a classe `GraphHelper` para determinar o custo total que a árvore mínima possui e verificando com a resposta do usuário (`sum_weights_mst.value`). Se forem iguais, é indicado no campo de *feedback* enquanto aguarda o retorno da resposta pela LLM (`llm.generate_response()`).

```
func avaliar_resposta():
    GraphHelper.set_vertices_edges(vertices.map(func(x): return x.id), edges

    if sum_weights_mst.value != GraphHelper.kruskal():
        set_ia_feedback("Resposta errada. Analise o grafo com cuidado!")
        return
    else:
        set_ia_feedback("Resposta correta. Analisando feedback...")

    set_ia_feedback()

func set_ia_feedback(text: String = ""):
    if text == "":

        var result = await llm.generate_response("", construct_prompt())
        if result.content:
            respostaIA.text = ""
            for part in result.content:
                respostaIA.text += part.candidates[0].content.parts[0].text
        else:
            respostaIA.text += "\nErro ao se conectar com a LLM. Verifique a

    else:
        respostaIA.text = text
```

Figura 26 – Funções `avaliar_resposta` e `set_ia_feedback`

5.1.4 LLM

A integração com Modelos de Linguagem de Grande Escala (LLM) é realizada por meio do *plugin* “Godot LLM Framework” (GODOT, 2024). O modelo empregado é o `gemini-flash-latest`, com requisições de inferência processadas nos servidores da Google. Quando a avaliação discursiva é solicitada, a função `construct_prompt()` é chamada. Ela constrói dinamicamente um *prompt*, estruturado como um dicionário contendo um *array* de mensagens, cada uma com um *role* (“user” ou “assistant”) e *content*. Este *prompt* inclui:

- Um *prompt* de sistema que define o papel da IA: “*Você é um avaliador de respostas sobre questões de teoria de grafos. Por favor, avalie clareza, correção conceitual e completude da resposta.*”;
- O enunciado da questão;

- A resposta discursiva do aluno.
- O estado atual do grafo, serializado em texto (listas de vértices e arestas).
- O formato esperado da resposta (“Nota: <Nota> \n Justificativa: <Justificativa>”).

5.2 Resultados

Esta seção apresenta os resultados visuais da implementação técnica descrita anteriormente, demonstrando a aparência final e o fluxo de interação do *software* como visto pelo usuário.

5.2.1 Menu das Questões

O resultado do carregamento das questões no menu principal pode ser visto na Figura 27 abaixo. A interface exibe uma grade de 6 cartões, onde cada um representa um módulo de problema disponível para o aluno selecionar. Em caso de haver mais que 6 questões, é possível mudar de página pelos botões inferiores.



Figura 27 – Tela do Menu de Questões

5.2.2 Exemplos de Questões

Ao selecionar um módulo, o usuário é levado à interface da questão. Essa seção apresentará alguns exemplos de problemas desenvolvidos, com possíveis resultados corretos e incorretos.

5.2.2.1 Construir grafo com Ciclo Euleriano

A Figura 28 abaixo ilustra a tela de uma questão de Ciclo Euleriano no estado inicial. É possível reparar que na esquerda tem o enunciado, seguido do campo de resposta discursiva e logo após a área de *feedback*, juntamente com o botão de retorno ao menu. Ainda, na direita é o que costuma alterar de questão a questão. Nesse caso, só apresenta o editor de grafo, pois o objetivo é construir o grafo que possua um ciclo euleriano com 5 vértices e 6 arestas.

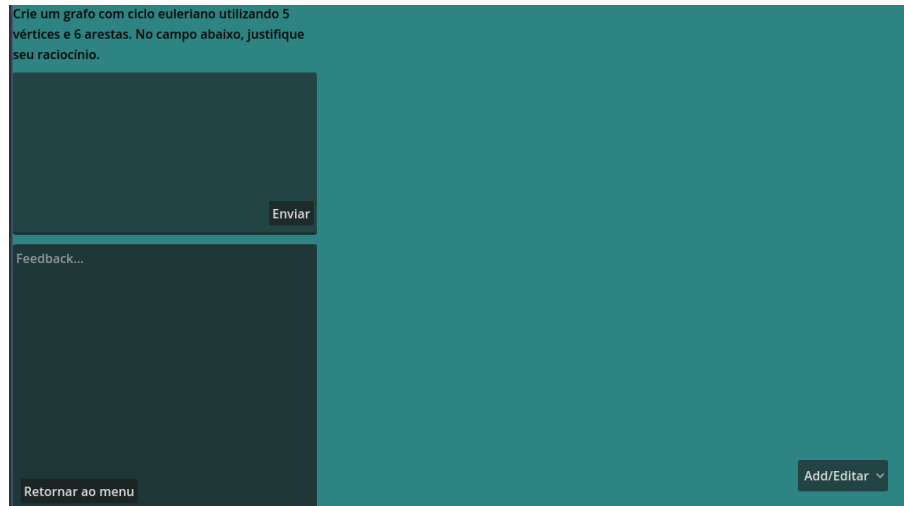


Figura 28 – Questão de Ciclo Euleriano

Na Figura 29 a seguir, há um possível resultado correto para essa questão. Nota-se o grafo construído, a resposta do usuário e o retorno dado pela LLM, respeitando o formato requisitado no *prompt*.

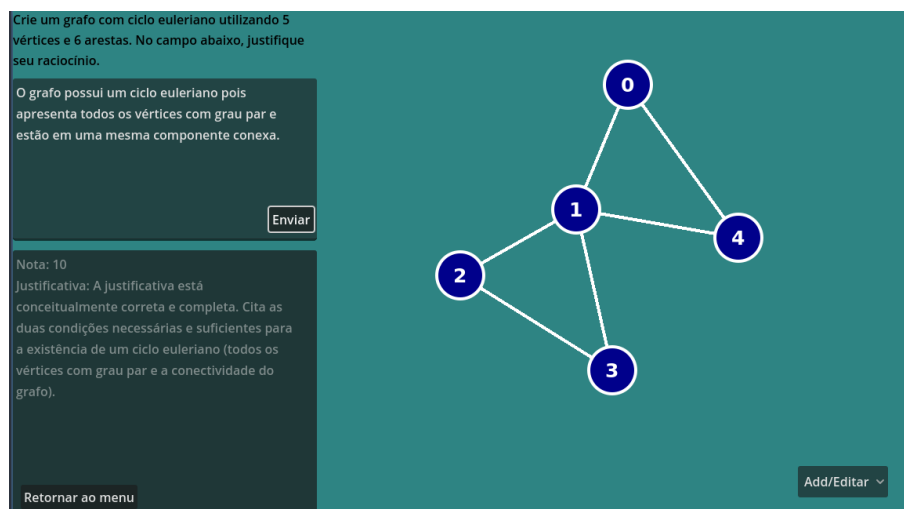


Figura 29 – Resultado correto da questão de Ciclo Euleriano

Agora, na Figura 30 a seguir, há o mesmo grafo construído, mas a justificativa dado pelo usuário foi incompleta. Nota-se que a LLM diminuiu a nota para 3 e levantou as

propriedades esperadas. Dessa forma, o aluno consegue recapitular o importante sobre problema, não apenas na parte prática, mas também no âmbito teórico.

Crie um grafo com ciclo euleriano utilizando 5 vértices e 6 arestas. No campo abaixo, justifique seu raciocínio.

O grafo possui um ciclo euleriano.

Enviar

Nota: 3
Justificativa: A resposta lista corretamente os vértices e arestas do grafo, mas falha em apresentar a justificativa teórica. O raciocínio (que vale 7 pontos) está ausente; deveria citar que um grafo possui um ciclo euleriano se for conexo e se todos os seus vértices tiverem grau par.

Retornar ao menu

Add/Editar

Figura 30 – Resultado correto da questão de Ciclo Euleriano

5.2.2.2 Completar grafo para ter um Ciclo Euleriano

A questão apresentada na Figura 32 abaixo é uma variante da anterior. Nesse caso, parte do grafo já está construído e o objetivo é finalizá-lo de forma que possua o Ciclo Euleriano. O usuário não pode remover qualquer vértice ou aresta que existia inicialmente, apenas o que for adicionado pelo mesmo.

Complete o grafo para que tenha 5 vértices e 6 arestas e possua um ciclo euleriano. No campo abaixo, justifique seu raciocínio.

Enviar

Feedback...

Retornar ao menu

Add/Editar

Figura 31 – Questão de Completar o Grafo com Ciclo Euleriano

Nota-se pelo respondido pelo usuário e retornado pela LLM (Figura 32) que a justificativa esperada é similar ao da questão anterior. O que altera é como ele deve manipular o grafo de forma que atinja o resultado esperado.

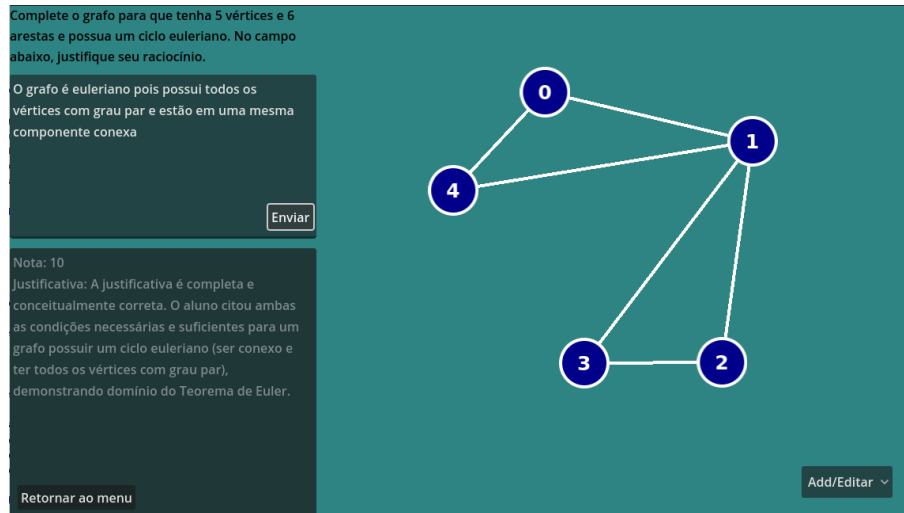


Figura 32 – Questão de Completar o Grafo com Ciclo Euleriano com resultado

5.2.2.3 Ciclo Hamiltoniano

Na questão apresentada (Figura 33), deve-se selecionar entre dois grafos qual possui um ciclo hamiltoniano. Como nota-se pela resposta do usuário e o *feedback* retornado, espera-se que a justificativa seja em torno da propriedade de grafos completos.

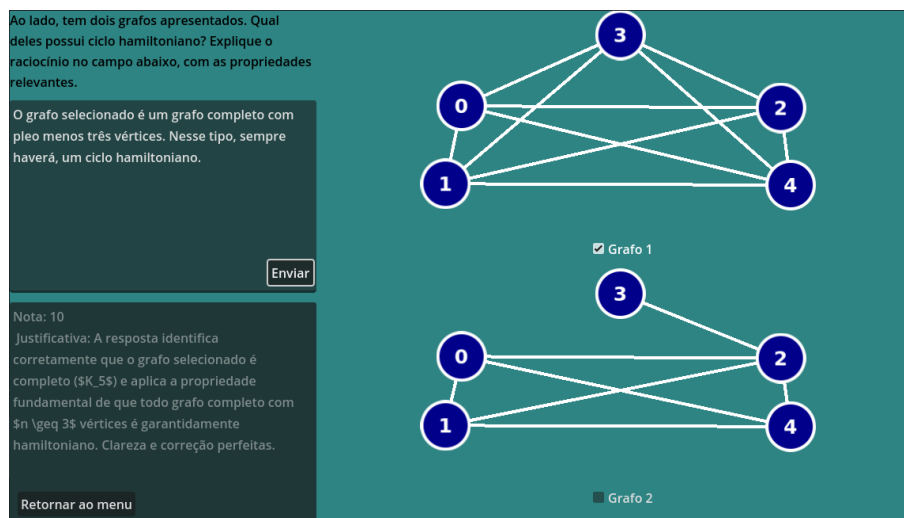


Figura 33 – Questão de Ciclo Hamiltoniano com resultado

5.2.2.4 Coloração

A Figura 34 a seguir é referente a Coloração Própria. Dado um grafo completo, o usuário deve indicar quantas cores são necessárias para conseguir colori-lo. Analisando a resposta discursiva e o retorno da LLM, a resposta estava correta e conseguiu abordar a propriedade esperada para a questão.

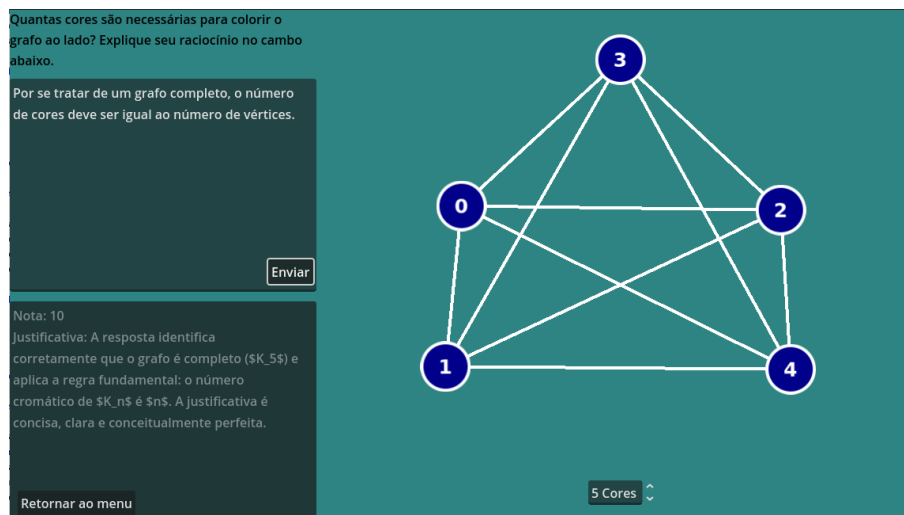


Figura 34 – Questão de Coloração com resultado

5.2.2.5 Ordenação Topológica

Na Figura 35 é possível visualizar uma questão referente à Ordenação Topológica. O objetivo do aluno é dar uma sequência apta para o problema. Na esquerda, destaca-se a nota recebida (9), pois embora as propriedades descritas e o algoritmo estejam corretos, a LLM esperava mais detalhes do seu funcionamento.

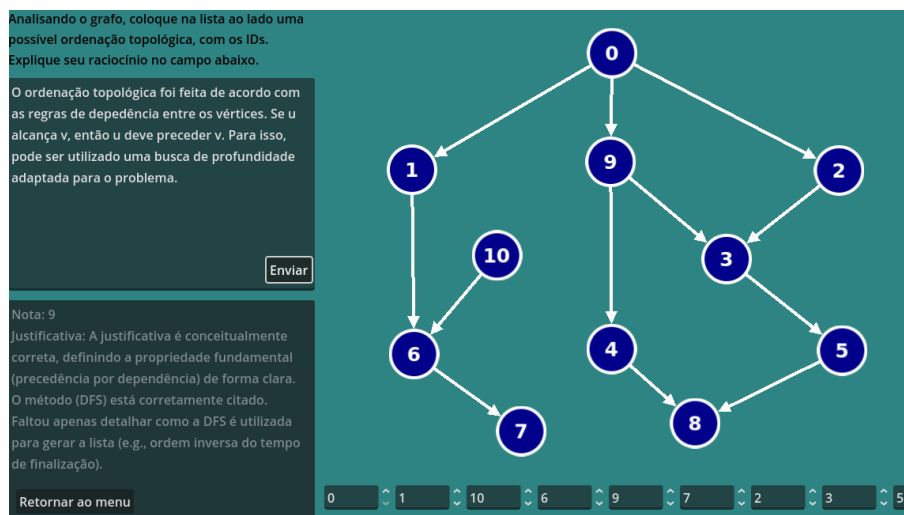


Figura 35 – Questão de Ordenação Topológica com resultado

5.2.2.6 Caminho Mínimo

Nesse exemplo (Figura 36), a questão é baseada no Caminhos de Custo Mínimo. É necessário determinar as distâncias mínimas a partir de um vértice aleatório (no caso, o 3) do grafo apresentado. Destaca-se que a resposta embora esteja correta e fosse possível usar o algoritmo exposto na justificativa, a LLM destacou que para esse grafo o mais eficiente seria outro método, mostrando ao aluno também a importância de decidir a

melhor estratégia para um dado problema. Embora ambos algoritmos retornem o mesmo resultado, o desempenho pode variar dependendo do grafo.

Analisando o grafo não dirigido, coloque na lista ao lado os valores de distância mínima a partir do vértice 3 (se inalcançável, selecione INF). Explique seu raciocínio no campo abaixo

Os valores colocados foram determinados usando o algoritmo de Bellman-Ford, porque ele calcula as menores distâncias a partir de um vértice, atualizando tudo aos poucos com várias passagens com o processo de relaxamento.

Nota: 6
Justificativa: A parte objetiva está correta (3pts). Contudo, a justificativa discursiva é fraca. O algoritmo Bellman-Ford é válido, mas inadequado para este grafo de pesos não negativos, onde Dijkstra é o método padrão e mais eficiente. A explicação apenas descreve o relaxamento, não justificando a escolha algorítmica.

Retornar ao menu

0 1 2 3 4 5 6
10 5 3 0 1 12 14
INF INF INF INF INF INF INF

Figura 36 – Questão de Ordenação Topológica com resultado

5.2.2.7 Árvore Geradora Mínima

Nessa questão (Figura 37), o usuário deve indicar a soma total dos pesos das arestas que formam a Árvore Geradora Mínima. Como indicado na explicação do raciocínio, foi citado o algoritmo utilizado para atingir esse valor, como o mesmo funcionava e as arestas que compõe a árvore. Dessa forma, o *feedback* retornou uma nota alta e uma justificativa elogiando a resposta.

Qual é a soma dos pesos da Árvore Geradora Mínima para o grafo ao lado? Explique seu raciocínio no campo abaixo.

Para encontrar o valor foi usado o algoritmo de Kruskal: ordenou-se as arestas por peso, uniu-se os componentes sem formar ciclos (union-find) e somou-se as arestas escolhidas (3-4, 2-3, 2-1, 0-1, 0-6 e 0-5), obtendo o peso total da AGM.

Nota: 10
Justificativa: Resposta excelente. O raciocínio (Algoritmo de Kruskal) é claro e conceitualmente correto (ordenar e evitar ciclos). A listagem explícita das 6 arestas escolhidas confirma a aplicação correta do método, resultando na soma exata de 18.0.

Retornar ao menu

18

Figura 37 – Questão de Árvore Geradora Mínima com resultado

5.3 Avaliação

A avaliação do *software* desenvolvido foi conduzida de forma contínua durante o processo de implementação, com foco na verificação dos requisitos funcionais e não-funcionais. Foram realizadas validações manuais e testes exploratórios que permitiram identificar falhas e assegurar a consistência da aplicação. O processo de avaliação contemplou os seguintes aspectos:

- **Cumprimento dos Requisitos Funcionais:** Cada módulo implementado foi verificado em relação às funcionalidades descritas na especificação. Por exemplo, no editor de grafos, foram testadas operações de criação, edição e remoção de vértices e arestas, bem como o carregamento de grafos a partir de arquivos externos. Nos módulos de questões, foram comparados os resultados obtidos com valores esperados (como custo da Árvore Geradora Mínima ou distâncias mínimas em Caminhos de Custo Mínimo), garantindo a correção da lógica implementada.
- **Requisitos Não Funcionais:** Foram observados aspectos de desempenho e usabilidade. Testes de estresse foram aplicados ao editor de grafos, simulando interações rápidas e sucessivas, como múltiplas inserções e remoções de elementos, para verificar a estabilidade da renderização e do tratamento de eventos. Também foi avaliada a responsividade da interface e a ausência de travamentos durante o uso.
- **Interação com a LLM:** O fluxo de avaliação discursiva foi validado em diferentes cenários. A função `construct_prompt()` foi testada para verificar a correta estruturação das mensagens enviadas ao modelo, e foram simuladas situações de falha na rede ou respostas inesperadas da API para garantir que o sistema exibisse mensagens adequadas ao usuário. Além disso, foi avaliada a consistência do *feedback* gerado pela LLM em relação ao conteúdo da justificativa fornecida pelo estudante.
- **Fluxo Completo de Avaliação:** Foi analisado o processo integrado de avaliação, desde a entrada da resposta objetiva e discursiva até a apresentação do resultado final ao usuário. Essa verificação assegurou que os diferentes componentes do sistema funcionassem de forma coesa e que o *feedback* fosse apresentado de maneira clara e compreensível.

6 Conclusões e Trabalhos Futuros

O projeto desenvolveu um jogo educacional que integra conceitos e algoritmos de teoria de grafos com mecânicas interativas, atendendo ao objetivo de oferecer um recurso didático complementar ao ensino de Computação.

A implementação atende aos requisitos funcionais e não funcionais definidos, especialmente quanto à extensibilidade das questões e a compatibilidade com Linux. Além disso, o desenvolvimento possibilitou um editor de grafos que permite criar, editar e carregar grafos via JSON, um menu de questões extensível, e uma abordagem híbrida de avaliação (objetiva e discursiva).

A integração com um Modelo de Linguagem Amplo (LLM) provê um *feedback* qualitativo sobre justificativas dos alunos, permitindo um maior aprofundamento dos conceitos e algoritmos para cada problema. Dessa forma, é possível avaliar tanto a solução prática quanto a argumentação teórica do aluno.

O uso da *Game Engine* Godot, uma ferramenta *open-source* e gratuita, permitiu uma melhor visibilidade do funcionamento da aplicação durante o desenvolvimento, o que facilitou a depuração dos problemas de lógica encontrados ao decorrer do projeto. Além disso, a ferramenta possibilita uma compatibilidade futura com outras plataformas como Web, Windows, MacOS e dispositivos móveis.

Por fim, o jogo pode receber melhorias futuras, como a expansão do conjuntos de questões para cobrir problemas de Grafos não abordados na versão atual. Também é possível incluir novos formatos de validação de respostas objetivas para aumentar a interatividade com a aplicação, além de refinar o *prompt* utilizado pela LLM para gerar *feedbacks* mais claros e detalhados. Além disso, pode-se implementar métricas de uso, como tempo gasto por problema, taxa de acertos objetivos e média das notas na justificativa, e avaliar uma possível integração com a plataforma Moodle, permitindo o professor monitorar o desempenho dos alunos e orientar melhorias das questões desenvolvidas.

Referências Bibliográficas

ANTONICK, G. **Hamilton's Icosian Game**. 2014. Acessado em: 24 mai. 2025. Disponível em: <<https://archive.nytimes.com/wordplay.blogs.nytimes.com/2014/10/06/icosian/>>. Citado na página 28.

BROMBERG, M. **A message to our community: Unity is canceling the Runtime Fee**. 2024. Acesso em: 09 jun. 2025. Disponível em: <<https://unity.com/blog/unity-is-canceling-the-runtime-fee>>. Citado na página 21.

CHARLEAUX, L.; TOLEDO, V. **O que é uma engine de jogos? Veja para que serve e quais são seus principais componentes**. 2025. Acessado em: 09 jun. 2025. Disponível em: <<https://tecnoblog.net/responde/o-que-e-uma-engine-de-jogos/>>. Citado na página 21.

CORMEN, T. H. et al. **Algoritmos - Teoria e Prática**. 3. ed. Rio de Janeiro: Elsevier, 2012. ISBN 978-85-352-3699-6. Citado nas páginas 11, 14, 15, 17 e 18.

DRAGONFLY. **Top 93 Game Engines Compared**. 2025. Acessado em: 13 mai. 2025. Disponível em: <<https://www.dragonflydb.io/game-dev/engines>>. Citado nas páginas 21 e 22.

GODOT. **Godot LLM Framework**. 2024. Acesso em: 10 out. 2025. Disponível em: <<https://godotengine.org/asset-library/asset/3282>>. Citado na página 37.

GODOT FAQ. **Frequently asked questions**. 2025. Acesso em: 09 jun. 2025. Disponível em: <<https://docs.godotengine.org/en/stable/about/faq.html>>. Citado na página 22.

JOHNSON, C. et al. Game development for computer science education. In: . [S.l.: s.n.], 2016. p. 23–44. Citado na página 20.

Layton Fandom. **Professor Layton and the Curious Village**. 2025. Acessado em: 04 abr. 2025. Disponível em: <https://layton.fandom.com/wiki/Professor_Layton_and_the_Curious_Village>. Citado nas páginas 11 e 23.

Layton Fandom. **Professor Layton and the Diabolical Box**. 2025. Acessado em: 25 mai. 2025. Disponível em: <https://layton.fandom.com/wiki/Professor_Layton_and_the_Diabolical_Box>. Citado nas páginas 11 e 25.

LEVEL-5. **Professor Layton and the Curious Village**. [S.l.]: Nintendo, 2007. Jogo para Nintendo DS. Citado nas páginas 23 e 24.

LEVEL-5. **Professor Layton and the Pandora Box**. [S.l.]: Nintendo, 2007. Jogo para Nintendo DS. Citado nas páginas 25 e 26.

LIMA-LOPES, R. E. d. Por uma revisão crítica do uso de inteligência artificial na educação. **Revista de Estudos de Cultura**, v. 11, n. 27, p. 39–65, ago. 2025. Disponível em: <<https://periodicos.ufs.br/revec/article/view/22719>>. Citado na página 22.

MEDIUM. **Unity: Development History and the Influence of This Game Engine on the Game Development Industry**. 2023. Acesso em: 23 jun. 2025. Disponível em: <https://medium.com/@wota_mmorpg/unity-development-history-and-the-influence-of-this-game-engine-on-the-game-development-36dc7a7a3b9d>. Citado na página 21.

NACIONAL, R. Gamifying education: Enhancing student engagement and motivation. v. 5, p. 716–729, 04 2023. Citado nas páginas 11 e 20.

NEAMAR. **Icosien**. 2010. Acessado em: 12 mai. 2025. Disponível em: <<https://neamar.fr/Res/Icosien/>>. Citado nas páginas 11 e 28.

PÓLYA, G.; TARJAN, R. E.; WOODS, D. R. Hamiltonian and eulerian paths. In: _____. **Notes on Introductory Combinatorics**. Boston: Birkhäuser Boston, 2010. p. 157–168. ISBN 978-0-8176-4953-1. Disponível em: <https://doi.org/10.1007/978-0-8176-4953-1_13>. Citado nas páginas 18 e 24.

PRADO, M. **Inteligência artificial e os impactos dos grandes modelos de linguagem na educação e na cultura informativa**. 2024. Acesso em: 09 dez. 2025. Disponível em: <<https://jornal.usp.br/artigos/inteligencia-artificial-e-os-impactos-dos-grandes-modelos-de-linguagem-na-educacao-e-na-cultura-informativa/>>. Citado na página 22.

PREVENTIONWORKSVT. **Origin of Unreal Engine?** 2024. Acesso em: 23 jun. 2025. Disponível em: <<https://preventionworksvt.org/blog/origin-of-unreal-engine/>>. Citado na página 21.

RUMPF, A. **Chromagraph**. 2020. Acessado em: 12 mai. 2025. Disponível em: <<https://adam-rumpf.itch.io/chromagraph>>. Citado nas páginas 11 e 27.

RUMPF, A. **Graph Theory Problems**. 2020. Acessado em: 12 mai. 2025. Disponível em: <<https://adam-rumpf.github.io/games/chromagraph/extras.html>>. Citado na página 27.

SALMELA, T. **Game development using the open-source Godot Game Engine**. Bacharelado — Tampereen ammattikorkeakoulu (TAMK), Tampere, Finlândia, 2022. Citado nas páginas 28 e 29.

SANTIAGO, R. de. **Anotações para a Disciplina de Grafos**. 2024. Acessado em: 01 dez. 2024. Disponível em: <<http://inf.ufsc.br/~r.santiago/downloads/INE5413.pdf>>. Citado nas páginas 11, 16, 18 e 19.

SANTIAGO, R. de. **Plano de Ensino da Disciplina: INE5413 - Grafos**. 2025. Acessado em: 10 jun. 2025. Disponível em: <<https://planos.inf.ufsc.br/modulos/planos/visualizar.php?id=4841>>. Citado nas páginas 11, 12 e 14.

SILVA, C. M.; MASARO, R. E.; PAULA, A. V. d. A gamificação como metodologia ativa no processo de ensino-aprendizagem no ensino superior. **Revista Valore**, v. 9, p. e–9014, jul. 2024. Disponível em: <<https://revistavalore.emnuvens.com.br/valore/artic le/view/1341>>. Citado na página 20.

STEAMDB. **What are games built with and what technologies do they use?** 2025. Acessado em: 09 jun. 2025. Disponível em: <<https://steamdb.info/tech/>>. Citado na página 21.

STRATEGYWIKI. **Professor Layton and the Diabolical Box**. 2021. Acessado em: 25 mai. 2025. Disponível em: <https://strategywiki.org/wiki/Professor_Layton_and_the_Diabolical_Box>. Citado na página 25.

STRATEGYWIKI. **Professor Layton and the Curious Village**. 2022. Acessado em: 06 abr. 2025. Disponível em: <https://strategywiki.org/wiki/Professor_Layton_and_the_Curious_Village>. Citado na página 23.

UNITY. **What platforms are supported by Unity?** 2025. Acesso em: 09 jun. 2025. Disponível em: <<https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity>>. Citado na página 21.

UNREAL ENGINE. **Unreal Engine 5**. 2025. Acesso em: 23 jun. 2025. Disponível em: <<https://www.unrealengine.com/en-US/unreal-engine-5>>. Citado na página 22.

VIDENOVİK, M. et al. Game-based learning in computer science education: a scoping literature review. **International Journal of STEM Education**, v. 10, n. 1, p. 54, 2023. ISSN 2196-7822. Disponível em: <<https://doi.org/10.1186/s40594-023-00447-2>>. Citado nas páginas 11 e 20.

APÊNDICE A – ARTIGO

Neste apêndice será apresentado o artigo no formato SBC, referente ao presente projeto.

Jogo Educacional para Grafos no Ensino de Computação

Guilherme Adenilson de Jesus¹, Rafael de Santiago¹

¹Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

guilhermeadenilson16@gmail.com, r.santiago@ufsc.br

Abstract. *Graph Theory is a fundamental area of Computer Science, with applications in problems such as Eulerian Path, Traveling Salesman Problem, and Connectivity. Given its importance, understanding its concepts is essential, although traditional learning can be challenging. Electronic games emerge as an auxiliary mechanism to make learning more interactive. This work aimed to develop an educational game for teaching graph algorithms and theorems using the Godot game engine. The project implements a two-stage evaluation system: validating the objective answer and using an LLM to analyze the discursive justification. The result is a functional didactic tool that combines interactivity and AI to enhance the learning process.*

Resumo. *A Teoria dos Grafos é fundamental na Ciência da Computação, mas seu aprendizado pode ser desafiador devido ao nível de abstração. Jogos eletrônicos surgem como um mecanismo para tornar o estudo mais interativo. O presente trabalho teve como objetivo o desenvolvimento de um jogo educacional para o ensino de algoritmos e teoremas de Grafos, implementado na game engine Godot. O jogo apresenta um editor de grafos interativo e módulos de problemas clássicos. Como principal contribuição, o projeto implementa um sistema de avaliação híbrido: validação algorítmica da resposta objetiva e análise da justificativa discursiva via LLM, fornecendo feedback qualitativo ao aluno.*

1. Introdução

A área de grafos é de suma importância para a Ciência da Computação, sendo fundamental na resolução de problemas clássicos como o Caminho Euleriano, o Caixeiro Viajante e problemas de conectividade [Santiago 2024, Cormen et al. 2012]. Devido ao alto nível de abstração e à complexidade de alguns algoritmos, a compreensão desses conceitos pode ser desafiadora no ensino tradicional.

O uso de jogos digitais e a gamificação mostram-se como abordagens promissoras para estimular a motivação e o engajamento dos estudantes [Nacional 2023]. Além disso, jogos favorecem o desenvolvimento de competências cognitivas e facilitam a visualização de estruturas complexas [Videnovik et al. 2023].

Nesse contexto, este trabalho apresenta o desenvolvimento de um jogo educacional focado em Teoria dos Grafos. O objetivo é fornecer uma ferramenta onde o aluno possa não apenas resolver problemas visualmente, mas também justificar seu raciocínio, recebendo *feedback* automático.

1.1. Objetivos

O objetivo geral deste trabalho foi desenvolver um jogo educacional para a demonstração de algoritmos e teoremas da área de Grafos, com mecânicas que estimulem o usuário a visualizar graficamente e compreender o funcionamento dos mesmos.

Para atingir esse propósito, foram definidos os seguintes objetivos específicos:

- Selecionar os algoritmos e teoremas de Grafos mais relevantes para o ensino introdutório;
- Escolher uma *Game Engine* adequada, priorizando ferramentas *open-source*;
- Especificar e desenvolver o jogo;

2. Fundamentação Teórica

A base teórica deste trabalho sustenta-se em dois pilares principais: os conceitos fundamentais da Teoria dos Grafos e o uso de jogos no processo de aprendizagem.

2.1. Problemas Clássicos de Grafos

A disciplina aborda problemas essenciais como Caminhos e Ciclos Eulerianos, que consistem em percorrer todas as arestas de um grafo sem repetição [Santiago 2024]. Outro tópico central são os Ciclos Hamiltonianos e o Problema do Caixeiro Viajante, que focam na visita única a cada vértice [Cormen et al. 2012]. Além destes, são explorados algoritmos de otimização como Caminho Mínimo (Dijkstra, Bellman-Ford) e Árvore Geradora Mínima (Kruskal, Prim), fundamentais para diversas aplicações em redes e logística.

2.2. Jogos e Aprendizagem

A aprendizagem baseada em jogos é reconhecida por aumentar a motivação intrínseca e o engajamento dos estudantes [Videnovik et al. 2023]. A gamificação, ao aplicar elementos de jogos em contextos educacionais, cria um ambiente seguro para falhas e tentativas, essencial para o aprendizado de lógica e programação [Nacional 2023]. A interatividade proporcionada por jogos digitais permite a visualização dinâmica de algoritmos, facilitando a absorção de conceitos abstratos.

2.3. Game Engines

Uma *Game Engine* é um ambiente de desenvolvimento que fornece a infraestrutura essencial para a criação de jogos, abstraindo tarefas complexas como renderização gráfica, simulação física e gerenciamento de áudio. Com esses recursos, os desenvolvedores podem focar na lógica e no conteúdo exclusivo do jogo, otimizando o processo de produção [Charleaux e Toledo 2025].

A escolha da ferramenta adequada depende de critérios como custo, curva de aprendizado e suporte à plataforma alvo. O mercado atual destaca três opções principais:

- **Unity:** Amplamente utilizada para jogos 2D e 3D, utiliza C# e possui uma vasta comunidade, embora exija licenças pagas para faturamentos elevados [medium 2023].
- **Unreal Engine:** Reconhecida pelo realismo gráfico em 3D e uso de C++ ou Blueprints (programação visual), cobrando *royalties* sobre o lucro do jogo [UNREAL ENGINE 2025].

- **Godot Engine:** Uma ferramenta gratuita e *open-source* (licença MIT) que tem ganhado destaque pela sua leveza e eficiência, especialmente em projetos 2D. Diferente das anteriores, permite uso irrestrito sem custos de licenciamento e possui uma linguagem nativa (GDScript) de fácil aprendizado, além de suporte oficial para Linux [Dragonfly 2025, GODOT FAQ 2025].

2.4. LLM

Os LLMs (Large Language Model) são sistemas de IA avançados capazes de compreender e produzir linguagem humana. Eles realizam tarefas como resumo, explicação e avaliação de textos [Prado 2024]. Na educação, funcionam como ferramentas de apoio com feedback imediato. Eles permitem personalizar o ensino, estimulando análise crítica e competências cognitivas, promovendo uma maior interação entre aluno e disciplina, tornando o aprendizado mais ativo [Lima-Lopes 2025]. Dessa forma, eles atuam como complemento às práticas tradicionais de avaliação, sem substituí-las.

3. Metodologia

Este trabalho caracteriza-se como uma pesquisa aplicada de natureza descritiva e explicativa. O desenvolvimento do jogo seguiu um fluxo estruturado em etapas:

1. **Levantamento Teórico:** Revisão bibliográfica sobre algoritmos de grafos e metodologias de ensino baseadas em jogos, além da análise de trabalhos correlatos como *Chromagraph* e a série *Professor Layton*.
2. **Seleção de Ferramentas:** Análise comparativa de *Game Engines*, resultando na escolha da Godot Engine devido à sua licença *open-source*, leveza e compatibilidade com Linux, ideal para o ambiente acadêmico.
3. **Especificação e Desenvolvimento:** Definição dos requisitos funcionais (editor de grafos, sistema de questões, avaliação) e implementação técnica utilizando a linguagem GDScript. A arquitetura foi projetada para ser modular, permitindo a expansão futura do conteúdo.
4. **Integração com IA:** Implementação de um módulo de avaliação discursiva utilizando um Modelo de Linguagem Amplo (LLM) via API, configurado para atuar como um tutor virtual que analisa as justificativas dos alunos.

4. Trabalhos Relacionados

Diversos jogos comerciais e educativos abordam conceitos de grafos, direta ou indiretamente. A franquia *Professor Layton* apresenta *puzzles* que se reduzem a problemas de caminhos eulerianos e hamiltonianos [Layton Fandom 2025]. O jogo *Chromagraph* foca especificamente em problemas de coloração de grafos [Rumpf 2020]. Já o *Icosian Game*, criado pelo próprio matemático William Hamilton em 1857, é o precursor dos problemas de ciclos hamiltonianos [Neamar 2010].

Embora esses jogos estimulem o raciocínio lógico, eles geralmente carecem de uma formalização pedagógica ou de um mecanismo que avalie se o jogador compreendeu a teoria por trás da solução. Este trabalho busca preencher essa lacuna ao integrar a prática do jogo com a fundamentação teórica exigida em disciplinas de graduação.

5. Desenvolvimento

A implementação foi realizada na *game engine* Godot, utilizando a linguagem GDScript. A arquitetura do projeto foi dividida em componentes modulares para facilitar a manutenção e a expansão.

5.1. Menu de Questões e Carregamento Dinâmico

Para permitir a adição de novas questões sem alterar o código fonte, implementou-se um sistema de carregamento dinâmico. Ao iniciar, o jogo lê um arquivo JSON que define a lista de questões, especificando o caminho da cena (`path`), imagem e título (Figura 1).

```
"questions": [  
  {  
    "path": "res://questions/question_euleriano.tscn",  
    "image": "res://icon.svg",  
    "title": "Ciclo Euleriano"  
  },  
  {  
    "path": "res://questions/question_hamiltoniano_completo.tscn",  
    "image": "res://svgs/vertex_dragged.svg",  
    "title": "Ciclo Hamiltoniano"  
  },  
  {  
    "path": "res://questions/question_euleriano_compleStar.tscn",  
    "image": "res://svgs/vertex_selected.svg",  
    "title": "Ciclo Euleriano Completar"  
  },  
]
```

Figura 1. JSON com a descrição das questões

A interface do menu instancia esses dados em uma grade de cartões selecionáveis, permitindo paginação caso o número de questões exceda o limite da tela.

5.2. Editor de Grafos

O Editor de Grafos é o componente central, responsável pela interação e renderização. Ele utiliza nós especializados como `EdgesBatch` e `ArrowsBatch` para desenhar arestas e setas de forma otimizada (Figura 2).

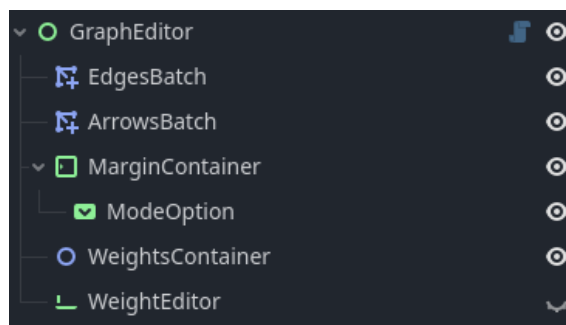


Figura 2. Cenário do Editor de Grafos na Godot

Para gerenciar as interações do usuário (criar, mover, deletar), foi implementada uma máquina de estados finita (Figura 3). Os estados incluem `IDLE` (ocioso),

DRAG_VERTEX (mover vértice), CONNECT_EDGE (criar aresta) e EDITING_WEIGHT (editar peso). A transição entre eles garante que ações conflitantes não ocorram simultaneamente.

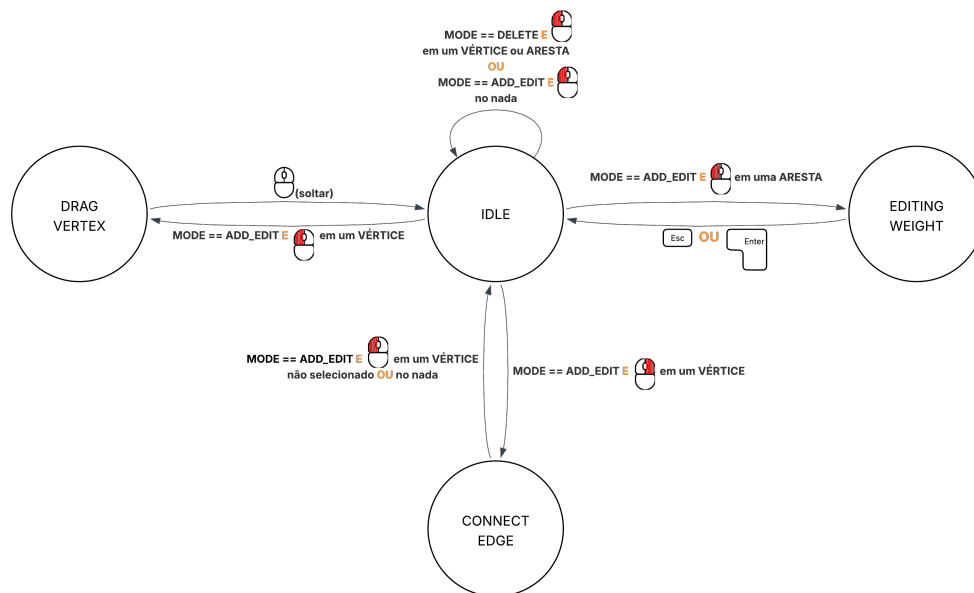


Figura 3. Máquina de Estados do Editor de Grafos

5.3. Sistema de Avaliação e Integração com LLM

Cada questão herda de uma classe base Question Abstract (Figura 4), que padroniza a interface (enunciado, resposta, feedback).

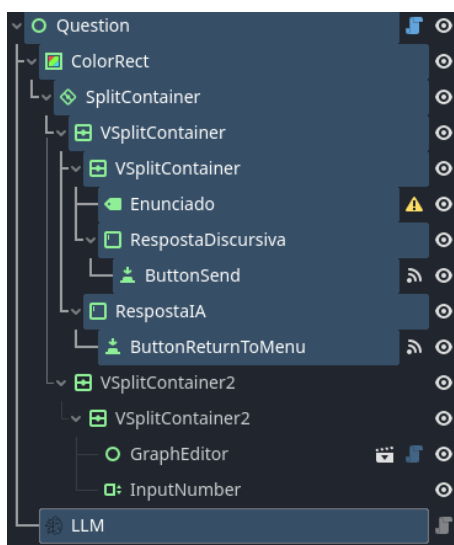


Figura 4. Estrutura de Cena de uma Questão

A avaliação ocorre em duas etapas:

1. **Avaliação Objetiva:** Uma classe auxiliar `GraphHelper` executa algoritmos clássicos sobre o grafo do aluno para validar a resposta matemática.
2. **Avaliação Discursiva:** Se a resposta objetiva for correta, o jogo utiliza o plugin "Godot LLM Framework"[Godot 2024], com comunicação ao modelo `gemini-flash-latest`, processadas nos servidores da Google, para enviar o grafo e a justificativa do aluno para uma IA. O *prompt* construído instrui a IA a avaliar a clareza e a correção conceitual (Figura 5).

```
func avaliar_resposta():
    GraphHelper.set_vertices_edges(vertices.map(func(x): return x.id), edges

    if sum_weights_mst.value != GraphHelper.kruskal():
        set_ia_feedback("Resposta errada. Analise o grafo com cuidado!")
        return
    else:
        set_ia_feedback("Resposta correta. Analisando feedback...")

    set_ia_feedback()

func set_ia_feedback(text: String = ""):
    if text == "":

        var result = await llm.generate_response("", construct_prompt())
        if result.content:
            respostaIA.text = ""
            for part in result.content:
                respostaIA.text += part.candidates[0].content.parts[0].text
        else:
            respostaIA.text += "\nErro ao se conectar com a LLM. Verifique a
    else:
        respostaIA.text = text
```

Figura 5. Código da função de avaliação e chamada da LLM

6. Resultados

Esta seção apresenta os resultados visuais e funcionais da aplicação.

6.1. Menu Principal

O carregamento dinâmico resulta em um menu navegável onde cada módulo é representado visualmente (Figura 6).



Figura 6. Tela do Menu de Questões

6.2. Módulos de Questões

6.2.1. Ciclo Euleriano

O aluno deve construir ou completar um grafo para que possua um Ciclo Euleriano. A Figura 7 demonstra um caso onde o grafo está correto, mas a justificativa foi incompleta. A IA (simulando um professor) reduziu a nota e explicou que faltou citar a necessidade de todos os vértices terem grau par.

Figura 7. Feedback da IA para justificativa incompleta em Ciclo Euleriano

Na variante de completar o grafo (Figura 8), o aluno deve adicionar as arestas faltantes sem remover as existentes.

Complete o grafo para que tenha 5 vértices e 6 arestas e possua um ciclo euleriano. No campo abaixo, justifique seu raciocínio.

O grafo é euleriano pois possui todos os vértices com grau par e estão em uma mesma componente conexa

Enviar

Nota: 10
Justificativa: A justificativa é completa e conceitualmente correta. O aluno citou ambas as condições necessárias e suficientes para um grafo possuir um ciclo euleriano (ser conexo e ter todos os vértices com grau par), demonstrando domínio do Teorema de Euler.

Retornar ao menu

Add/Editar

Figura 8. Questão de completar Ciclo Euleriano com feedback positivo

6.2.2. Ciclo Hamiltoniano e Coloração

Outros módulos abordam Ciclos Hamiltonianos (Figura 9) e Coloração de Grafos (Figura 10), onde o aluno deve identificar propriedades de grafos completos (K_5 , por exemplo).

Do lado, tem dois grafos apresentados. Qual deles possui ciclo hamiltoniano? Explique o raciocínio no campo abaixo, com as propriedades relevantes.

O grafo selecionado é um grafo completo com pelo menos três vértices. Nesse tipo, sempre haverá, um ciclo hamiltoniano.

Enviar

Nota: 10
Justificativa: A resposta identifica corretamente que o grafo selecionado é completo (K_5) e aplica a propriedade fundamental de que todo grafo completo com $n \geq 3$ vértices é garantidamente hamiltoniano. Clarezza e correção perfeitas.

Retornar ao menu

Figura 9. Questão de Ciclo Hamiltoniano

Quantas cores são necessárias para colorir o grafo ao lado? Explique seu raciocínio no campo abaixo.

Por se tratar de um grafo completo, o número de cores deve ser igual ao número de vértices.

Enviar

Nota: 10
Justificativa: A resposta identifica corretamente que o grafo é completo (K_5) e aplica a regra fundamental: o número cromático de K_n é n . A justificativa é concisa, clara e conceitualmente perfeita.

Retornar ao menu

5 Cores

Figura 10. Questão de Coloração (K_5)

6.2.3. Algoritmos de Otimização

Para problemas como Caminho Mínimo (Figura 11) e Árvore Geradora Mínima (Figura 12), o sistema valida se os valores numéricos correspondem à execução correta dos algoritmos de Dijkstra/Bellman-Ford e Kruskal/Prim, respectivamente.

Analisando o grafo não dirigido, coloque na lista ao lado os valores de distância mínima a partir do vértice 3 (se inalcançável, selecione INF). Explique seu raciocínio no campo abaixo.

Os valores colocados foram determinados usando o algoritmo de Bellman-Ford, porque ele calcula as menores distâncias a partir de um vértice, atualizando tudo aos poucos com várias passagens com o processo de relaxamento.

Nota: 6
Justificativa: A parte objetiva está correta (3pts). Contudo, a justificativa discursiva é fraca. O algoritmo Bellman-Ford é válido, mas inadequado para este grafo de pesos não negativos, onde Dijkstra é o método padrão e mais eficiente. A explicação apenas descreve o relaxamento, não justificando a escolha algorítmica.

Retornar ao menu

10 5 3 0 1 12 14

INF INF INF INF INF INF INF

Figura 11. Questão de Caminho Mínimo

Qual é a soma dos pesos da Árvore Geradora Mínima para o grafo ao lado? Explique seu raciocínio no campo abaixo.

Para encontrar o valor foi usado o algoritmo de Kruskal: ordenou-se as arestas por peso, uniu-se os componentes sem formar ciclos (union-find) e somou-se as arestas escolhidas (3-4, 2-3, 2-1, 0-1, 0-6 e 0-5), obtendo o peso total da AGM.

Nota: 10
Justificativa: Resposta excelente. O raciocínio (Algoritmo de Kruskal) é claro e conceitualmente correto (ordenar e evitar ciclos). A listagem explícita das 6 arestas escolhidas confirma a aplicação correta do método, resultando na soma exata de 18.0.

Retornar ao menu

18

Figura 12. Questão de Árvore Geradora Mínima (AGM)

7. Conclusão

Este trabalho apresentou uma ferramenta de apoio ao ensino de grafos que une a interatividade dos jogos com a profundidade da avaliação teórica. A utilização da *Godot Engine* permitiu criar uma aplicação leve e extensível. A integração com Inteligência Artificial mostrou-se promissora para escalar o *feedback* individualizado, simulando a correção de um professor. Trabalhos futuros incluem a expansão para novos tipos de problemas (como Fluxo Máximo) e a integração com plataformas de ensino como o Moodle para acompanhamento de métricas de aprendizado.

Referências

- [Charleaux e Toledo 2025]CHARLEAUX, L.; TOLEDO, V. *O que é uma engine de jogos? Veja para que serve e quais são seus principais componentes*. 2025. Acessado em: 09 jun. 2025. Disponível em: <<https://tecnoblog.net/responde/o-que-e-uma-engine-de-jogos/>>.
- [Cormen et al. 2012]CORMEN, T. H. et al. *Algoritmos - Teoria e Prática*. 3. ed. Rio de Janeiro: Elsevier, 2012. ISBN 978-85-352-3699-6.

- [Dragonfly 2025]DRAGONFLY. *Top 93 Game Engines Compared*. 2025. Acesso em: 13 mai. 2025. Disponível em: <<https://www.dragonflydb.io/game-dev/engines>>.
- [GODOT FAQ 2025]GODOT FAQ. *Frequently asked questions*. 2025. Acesso em: 09 jun. 2025. Disponível em: <<https://docs.godotengine.org/en/stable/about/faq.html>>.
- [Layton Fandom 2025]Layton Fandom. *Professor Layton and the Curious Village*. 2025. Acesso em: 04 abr. 2025. Disponível em: <https://layton.fandom.com/wiki/Professor_Layton_and_the_Curious_Village>.
- [Lima-Lopes 2025]LIMA-LOPES, R. E. d. Por uma revisão crítica do uso de inteligência artificial na educação. *Revista de Estudos de Cultura*, v. 11, n. 27, p. 39–65, ago. 2025. Disponível em: <<https://periodicos.ufs.br/revec/article/view/22719>>.
- [medium 2023]MEDIUM. *Unity: Development History and the Influence of This Game Engine on the Game Development Industry*. 2023. Acesso em: 23 jun. 2025. Disponível em: <https://medium.com/@wotam_morpg/unity-development-history-and-the-influence-of-this-game-engine-on-the-game-development-36dc7a7a3b9d>.
- [Nacional 2023]NACIONAL, R. Gamifying education: Enhancing student engagement and motivation. v. 5, p. 716–729, 04 2023.
- [Neamar 2010]NEAMAR. *Icosien*. 2010. Acesso em: 12 mai. 2025. Disponível em: <<https://neamar.fr/Res/Icosien/>>.
- [Prado 2024]PRADO, M. *Inteligência artificial e os impactos dos grandes modelos de linguagem na educação e na cultura informativa*. 2024. Acesso em: 09 dez. 2025. Disponível em: <<https://jornal.usp.br/artigos/inteligencia-artificial-e-os-impactos-dos-grandes-modelos-de-linguagem-na-educacao-e-na-cultura-informativa/>>.
- [Rumpf 2020]RUMPF, A. *Chromagraph*. 2020. Acesso em: 12 mai. 2025. Disponível em: <<https://adam-rumpf.itch.io/chromagraph>>.
- [Santiago 2024]SANTIAGO, R. de. *Anotações para a Disciplina de Grafos*. 2024. Acesso em: 01 dez. 2024. Disponível em: <<http://inf.ufsc.br/~r.santiago/downloads/INE5413.pdf>>.
- [UNREAL ENGINE 2025]UNREAL ENGINE. *Unreal Engine 5*. 2025. Acesso em: 23 jun. 2025. Disponível em: <<https://www.unrealengine.com/en-US/unreal-engine-5>>.
- [Videnovik et al. 2023]VIDENOVIK, M. et al. Game-based learning in computer science education: a scoping literature review. *International Journal of STEM Education*, v. 10, n. 1, p. 54, 2023. ISSN 2196-7822. Disponível em: <<https://doi.org/10.1186/s40594-023-00447-2>>.

APÊNDICE B – CÓDIGO FONTE

O código fonte desenvolvido para este projeto encontra-se integralmente disponível no repositório do autor no Códigos UFSC, acessível através do seguinte link: <https://codigos.ufsc.br/guilherme.aj/Graph_Game>. Para edição e depuração, recomenda-se o uso de Godot, versão 4.5 ou superior. Para o funcionamento da LLM pelo Gemini, deve ser colocada a API KEY no arquivo `config_api.cfg`.