Implementação de Estruturas de Dados em um Jogo de Exploração 2D

Ana Paula Xavier[1], Carlos P. da Silva[1], Cíntia Seixas Fontes[1], Susie M. Farias[1], Tiago M. Santos[1]

[1] Instituto de Ciência, Tecnologia e Inovação (ICTI)
Universidade Federal do Amazonas (UFAM) – Itacoatiara – AM – Brasil

ana.paula@ufam.edu.br

Abstract. *This report presents the development of a 2D exploration game using an AVL Tree to manage the player's inventory and a Graph structure to represent the game map. The project demonstrates how data structures influence system performance, gameplay mechanics, and algorithmic efficiency. The implementation allows practical experimentation with search algorithms, balanced trees, and adjacency lists.*

Resumo. *Este relatório apresenta o desenvolvimento de um jogo de exploração 2D utilizando uma Árvore AVL para gerenciar o inventário do jogador e um Grafo para representar o mapa. O projeto demonstra como estruturas de dados influenciam o desempenho do sistema, a dinâmica do jogo e a eficiência dos algoritmos aplicados. A implementação permite experimentação prática com algoritmos de busca, árvores balanceadas e listas de adjacência.*

1. Introdução

O desenvolvimento de jogos eletrônicos requer estruturas de dados eficientes para gerenciar diferentes aspectos do gameplay. Neste trabalho, implementamos duas estruturas principais: uma Árvore AVL para organizar o inventário do jogador e um Grafo para representar o mapa de exploração, seguindo princípios fundamentais de Estruturas de Dados.

2. Objetivo

O sistema tem como finalidade criar um jogo de exploração 2D que possibilite a aplicação prática dos conceitos estudados na disciplina de Algoritmos e Estruturas de Dados II. O mapa do jogo é representado por um grafo, enquanto o inventário do jogador utiliza uma Árvore AVL. O jogo permite percorrer um labirinto, coletar itens, e usar uma chave para alcançar a saída, integrando operações como buscas, balanceamento e recuperação de caminhos.

3. Arquitetura do Sistema

- A classe `Node` representa tanto os nós da Árvore AVL quanto os vértices internos do grafo.
- Uma Árvore AVL pode conter múltiplos nós, assim como um Grafo pode possuir múltiplos vértices.
- Os métodos retornam tipos específicos: booleanos, listas de caminhos ou valores nulos.

4. Análise Detalhada das Estruturas

4.1. Árvore AVL para Inventário

A Árvore AVL, conforme apresentada em [1], garante operações de inserção, busca e remoção em tempo $O(\log n)$, permitindo um gerenciamento eficiente dos itens do inventário. As principais características incluem:

- Balanceamento automático após cada operação;
- Ordenação natural dos itens por chave;
- Suporte a valores duplicados com atualização.

4.2. Grafo para Representação do Mapa

O grafo não direcionado e não ponderado segue referências de [3] e representa o mapa a partir de:

- Vértices representando salas do labirinto;
- Arestas conectando salas adjacentes;
- Implementação por lista de adjacência.

5. Complexidade Computacional

Tabela 1. Análise de complexidade das operações

Operação	Árvore AVL	Grafo
Inserção	$O(\log n)$	$O(1)$
Busca	$O(\log n)$	$O(V + A)$
Remoção	$O(\log n)$	$O(1)$
BFS	–	$O(V + A)$

6. Principais Funções

6.1. Geração do Mapa

A função `generate_graph()` cria um mapa 15×15, gera adjacências e remove caminhos para formar um labirinto. A complexidade é $O(V + A)$.

6.2. Verificação de Solubilidade

A função `is_solvable()` utiliza BFS para garantir que exista um caminho entre entrada e saída, também com complexidade $O(V + A)$.

6.3. Movimentação do Jogador

A função `move(direction)` atualiza posição, verifica colisões e registra histórico com custo $O(1)$.

6.4. Inventário – AVL

A função `insert(item)` insere itens mantendo o balanceamento em tempo $O(\log n)$.

6.5. Coleta Otimizada de Itens

A função `get_collection_path()` usa BFS repetidamente com complexidade $O(k(V + A))$.

6.6. Caminho Mais Curto

A função `bfs(start, end)` retorna o percurso ideal com custo $O(V + A)$.

6.7. Sistema de Salvamento

A função `save_game()` armazena posição e inventário, percorrendo a AVL ($O(n)$).

7. Fluxo de Execução do Sistema

O jogo inicia com a geração automática do mapa. O jogador se movimenta, coleta itens, verifica interações e pode solicitar caminhos otimizados, além de salvar/carregar o progresso. O jogo encerra quando o jogador alcança a sala final com a chave.

8. Captura de Telas

Execução do arquivo `main.py`

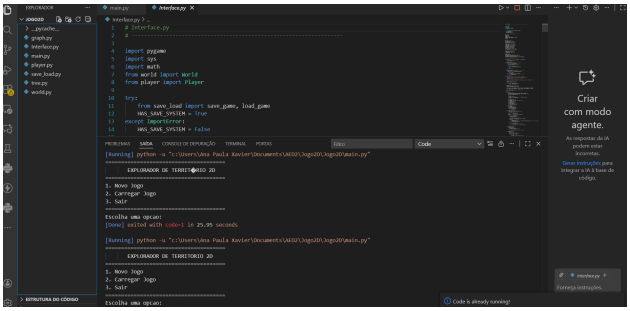

Figura 1. Execução do arquivo `main.py`.

Execução do arquivo `interface.py`

A imagem retrata a criação das salas:

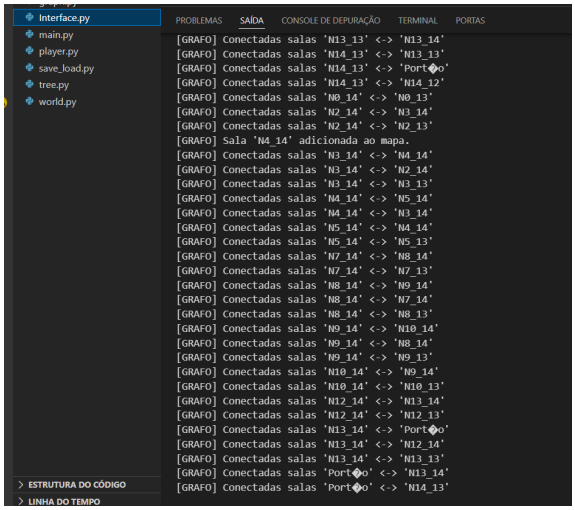

Figura 2. Salas geradas pelo sistema.

A interface gráfica do jogo Explorador de Território 2D é exibida na Figura a seguir.

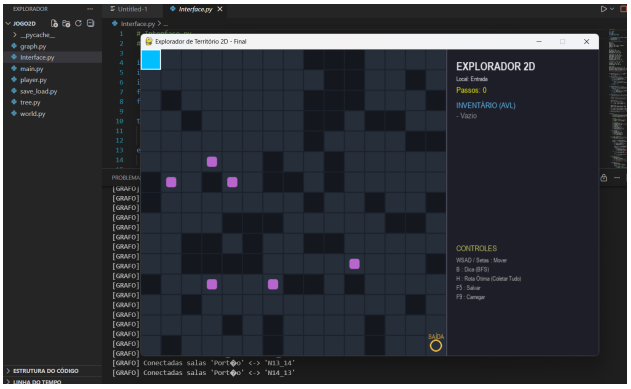

Figura 3. Tela principal do jogo explorador 2D.

A seguir, é apresentada a tela exibida quando o jogador vence a partida:

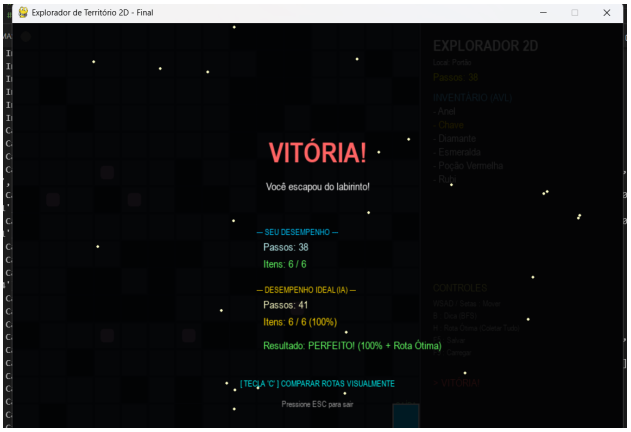

Figura 4. Tela de vitória do jogador.

9. Conclusões e Dificuldades

O projeto permitiu consolidar conceitos de AED2, explorando grafos, árvores balanceadas e algoritmos de busca em um ambiente real. Entre as principais dificuldades enfrentadas, destacam-se:

- Integração dos algoritmos de busca com a interface gráfica;
- Geração automática do mapa garantindo solubilidade;
- Implementação completa da AVL com balanceamento.

O sistema final cumpriu os objetivos, sendo funcional e didático ao demonstrar a aplicabilidade prática das estruturas de dados.

10. Repositório

O código-fonte está disponível em:
`https://github.com/SusieLaureen/AEDII`

Referências

[1] Adelson-Velsky, G.; Landis, E. (1962). *An algorithm for the organization of information*. Soviet Mathematics Doklady.

[2] Cormen, T.; Leiserson, C.; Rivest, R.; Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

[3] Skiena, S. (2010). *The Algorithm Design Manual*. Springer.

[4] Phind (2025). Geração automática de diagramas.