Universidade Federal do Agreste de Pernambuco

Departamento de Ciência da Computação

Algoritmo e Estrutura de Dados II

Professor Igor Medeiros Vanderlei

Estudo de grafos para mineração e exploração de cavernas com Gold Diggers

Luís Guilherme Pontes Melquiades

Lucas Branco Alves de Melo

Erik César de Queiroz Brasil

Agreste - PE

April 13, 2023

1 Resumo

Gold Diggers é um jogo de exploração que coloca o jogador no papel de um minerador que se perdeu em uma caverna e precisa encontrar o caminho de volta à superfície. A caverna é representada como um grafo, o que permite a aplicação de algoritmos para a geração de novas cavernas e a definição de objetivos e desafios. O jogador deve coletar recursos ao longo do caminho para obter pontos, mas também deve evitar morrer por cansaço. Com mecânicas de quebra-cabeças e estratégia, Gold Diggers desafia o jogador a encontrar o caminho mais eficiente para chegar à saída. O jogo se destaca pela sua abordagem inovadora de explorar o ambiente subterrâneo e pela sua jogabilidade desafiadora com interface textual. Além de também ter uma identidade ambiental única, ao menos para jogos modernos.

2 Introdução

Gold Diggers consiste em sobreviver numa caverna gélida e escura. O objetivo é buscar a saída antes de sucumbir ao cansaço gerado por locomover-se num ambiente claustrofóbico e nocivo. O jogador buscará a saída da caverna enquanto encontra itens que pode vender para misteriosos mercadores que encontram-se nas mais variadas câmaras da caverna ou um misterioso cristal que o ajudará a encontrar a saída. O jogador incia em uma câmara da caverna e pode mover-se para outras câmaras que estejam conectadas por túneis com a câmara atual, gastando energia para isso, assim que chegar a câmara de saída, o jogador vence o jogo.

O jogo em questão apresenta uma particularidade que o destaca dos demais, visto que, ao invés de recorrer a recursos audiovisuais, ele se utiliza exclusivamente de texto para apresentar as descrições das câmaras, dos itens e, inclusive, para compor a própria arte do jogo. A representação da caverna como um grafo, no qual cada câmara é um vértice e cada túnel é uma aresta, permite a aplicação de algoritmos para a geração de novas cavernas, a definição de caminhos para o usuário e a criação de objetivos e desafios.

3 Fundamentação Teórica

3.1 Grafos

Muitos problemas cotidianos podem ser formulados em termos de objetos e das conexões entre eles. Por exemplo, os dados sobre as rotas de ônibus intermunicipais do Brasil ou a análise de conexões entre os componentes de um circuito. Para trabalhar com esses objetos, desta forma, é necessário ter informações sobre as conexões (rotas de ônibus ou cabeamento dos componentes) entre os objetos (cidades ou os próprios componentes em si) [2]. Os grafos representam os objetos e suas conexões de forma puramente estruturada, ou seja, exploram um objeto e quais objetos este primeiro se relaciona de alguma forma [8].

Diferentemente das estruturas lineares e até mesmo das não lineares, como árvores, nas quais os objetivos quase sempre são armazenar informações e estudar o método mais eficiente possível para recuperar ou alterar tais informações, no modelo dos grafos, o foco se dá em responder perguntas relacionadas aos objetos e o que eles podem representar quando em conjunto [8]. Aplicações constantes de grafos são algoritmos de inteligência artificial [3] ou até mesmo estruturas de dados mais simples, como listas ou árvores [8], que seriam basicamente grafos que precisam seguir uma estrutura como pré-requisito para se adequarem melhor ao seu tipo de tarefa específica, um grafo simplificado em outras palavras.

Formalizando as informações, um grafo é uma coleção de vértices e arestas. Os vértices são os objetos que possuem nomes e diversas outras propriedades. As arestas são as conexões entre os vértices. Um grafo é definido pelo conjunto formado por todos os vértices e arestas. Por definição matemática, não há ordem para os elementos, A; B; C = B; A; C = C; B; A... Todas as possíveis permutações formadas por esses três elementos formam exatamente o mesmo conjunto [2]. Estamos tratando de conjuntos e não de sequências ou listas. Este conceito é herdado da matemática.

O caminho entre dois vértices quaisquer, X e Y, é uma lista de vértices na qual os vértices sucessivos são conectados por arestas. Note que neste exemplo já foi explicado uma situação na qual se usa uma lista e não um conjunto. Um caminho se trata de uma lista, uma sequência [8].

Conceitos:

- Um grafo é dito conectado se para cada vértice existir um caminho para todos os outros vértices [8]. Caso contrário, é um grafo não conec-

tado, formado por vários componentes distintos, os quais são grafos conectados entre si.

- Um caminho simples em um grafo é aquele em que nenhum vértice se repete. Por exemplo, BAFEGAC é um caminho simples de B até C, considerando cada letra como um vértice [8].

- Um caminho cíclico é um caminho simples, exceto pelo fato de que o primeiro e o último vértice devem ser o mesmo, formando um ciclo [8]. Por exemplo, AFEGA é um ciclo que sai de A e termina em A.

Propriedades:

- O número de arestas em um grafo pode variar entre 0 e (1/2)V(V-1) [8], onde V é o número de vértices. Para um grafo com V = n elementos, o número total de conexões possíveis seria N*(N-1), mas cada aresta é contada duas vezes, uma para cada vértice que ela conecta, sendo necessário dividir o resultado por 2 para obter o número máximo de arestas.

  Essa duplicidade deve ser removida apenas no caso de grafos não direcionados. Para grafos direcionados, cada aresta é unidirecional e não há duplicidade em dois caminhos sendo um para ir e outro para voltar.

- Grafos completos são aqueles que possuem o número máximo de arestas, enquanto grafos esparsos são aqueles que possuem uma quantidade pequena de arestas em relação ao máximo [8], geralmente menor que V*(log2V). Por outro lado, grafos densos são aqueles que se aproximam do número máximo de arestas, mas não o atingem, geralmente com mais de V*(log2V) arestas.

Existem duas formas principais de representação matricial: a matriz de adjacência e a matriz de incidência. A matriz de adjacência é uma matriz quadrada de tamanho VxV que descreve as conexões entre os vértices do grafo. Cada célula (i, j) da matriz contém um valor que indica se os vértices i e j estão conectados por uma aresta [8]. Para grafos não direcionados, a matriz é simétrica em relação à diagonal principal.

Já a matriz de incidência é uma matriz de tamanho VxE que descreve a relação entre os vértices e as arestas do grafo. Cada célula (i, j) da matriz contém um valor que indica se o vértice i é um dos extremos da aresta j. Se a

aresta j é direcionada de i para outro vértice, o valor na célula será negativo, enquanto se a aresta j é direcionada para i, o valor será positivo. Se a aresta j não está conectada ao vértice i, o valor na célula será zero [8].

A escolha da representação matricial depende da natureza do problema a ser resolvido e dos algoritmos que serão aplicados. Por exemplo, a matriz de adjacência é útil para determinar a existência de caminhos entre vértices, enquanto a matriz de incidência é útil para determinar a existência de ciclos em um grafo [1].

Além disso, é importante ressaltar que a representação matricial de um grafo pode ser custosa em termos de espaço de memória, especialmente para grafos grandes. Por isso, é comum utilizar estruturas de dados mais eficientes, como listas de adjacência ou matriz de adjacência esparsa, que armazenam apenas as conexões existentes no grafo [5].

A representação por lista de adjacência é uma forma mais compacta de representar grafos esparsos, ou seja, grafos com poucas conexões. Nessa representação, para cada vértice do grafo, uma lista de seus vizinhos é armazenada. Essa lista pode ser implementada de diferentes maneiras, como uma lista ligada, um vetor dinâmico ou uma árvore de busca binária [1].

Para ilustrar, considere um grafo não direcionado com 4 vértices e as seguintes conexões: o vértice 0 está conectado ao vértice 1 e ao vértice 2; o vértice 1 está conectado ao vértice 0, ao vértice 2 e ao vértice 3; o vértice 2 está conectado ao vértice 0, ao vértice 1 e ao vértice 3; e o vértice 3 está conectado ao vértice 1 e ao vértice 2. A representação por lista de adjacência desse grafo seria:

Vértice 0: [1, 2] Vértice 1: [0, 2, 3] Vértice 2: [0, 1, 3] Vértice 3: [1, 2]

Essa representação permite acessar rapidamente todos os vizinhos de um vértice e é útil em muitos algoritmos que exploram a estrutura do grafo, como a busca em largura ou a busca em profundidade [6]. De fato, a lista de adjacência é a representação "favorita" em algoritmos de grafos, sendo casualmente mais utilizada principalmente por sua simplicidade representativa, apesar de existirem modelos mais eficientes [7]. Por fins de simplicidade, o trabalho desenvolvido adotou a representação da lista de adjacência.

3.2 Algoritmo A*

A-estrela é um algoritmo para busca de caminho em grafos, grids, etc. É muito utilizado devido a sua eficiência.

Começando por um nó especifico, o algoritmo procura atingir o nó alvo e determinar qual é a rota de menor custo até aquele nó alvo.

O cálculo principal sobre quais caminhos escolher é definido por 3 variáveis relativas a um nó. $f(n)$, $g(n)$ e $h(n)$. h é a heurística do nó especificado até o nó alvo, isso na prática, o custo caso o caminho fosse uma linha reta. g é o custo de movimentação do nó inicial até o nó especificado. E f é a soma de g e h. Representando o quão próximo aquele nó está do nó alvo.

Para o jogo Gold Diggers. Foi escolhido e utilizado como base o algoritmo A*, porém, alterações tiveram que ser feitas para acomodá-lo às mecânicas do jogo. Ao invés de retornar o melhor caminho para o nó alvo, que no jogo é o nó onde encontra-se a saída da caverna. O algoritmo retorna a distância exata, no caminho de custo mais eficiente, do nó em que o jogador está para o nó onde se localiza a saída da caverna.

O algoritmo recebe o nó atual do jogador, que chamamos de **nó inicial**. Define valores específicos no **nó inicial** em f,g e h para acomodar o algoritmo. Cria duas listas de nós: a **lista aberta** e a **lista fechada**. O **nó inicial** é adicionado à **lista aberta**. E o **nó de saída** é procura pela lista de todos os nós da caverna, essa etapa é necessária em certa parte do algoritmo.

Após essa configuração inicial. É iniciado um ***loop*** que permanecerá ativo até que o **nó de saída** seja encontrado, ou caso perceba-se que não é possível encontrá-lo. Definimos um **nó atual** que será o nó com menor valor de f na **lista aberta**. Removemos o **nó atual** da **lista aberta** e o adicionamos na **lista fechada**. Em seguida, para cada **nó vizinho** do **nó atual** verificamos se está na **lista fechada**. Caso não esteja, nós prosseguimos e verificamos se é o **nó de saída**. Caso seja, retornamos $g(atual) + |d(vizinho, atual)|$, sendo d, a distância do **nó atual** para o **nó vizinho**. Caso não seja o **nó de saída**, calculamos seu g,h e f. Caso o $g(atual) + |d(vizinho, atual|$ for menor que o $g(vizinho)$ ou o **nó vizinho** não estiver contido na **lista aberta**, definimos o g do **nó vizinho** como $g(atual) + |d(vizinho, atual|$. E por último, colocamos o **nó vizinho** na **lista aberta**.

Dessa maneira descobrimos qual é a distância, pelo caminho mais eficiente, do nó escolhido até o nó de saída da caverna. Ou se não é possível chegar nó final, o que é bastante útil na criação aleatória de grafos pois é necessário que o jogo não seja impossível.

4 Materiais e Métodos

Para representar a exploração de mineração e garimpo em uma caverna, foi utilizada a linguagem de programação Java, juntamente com as IDEs Spring Tool Suite 4 e IntelliJ IDEA. Inicialmente, foi realizada uma pesquisa sobre a possibilidade de utilizar o editor de texto Vim para o projeto em questão, porém, constatou-se que seria necessário um extenso trabalho de configuração e instalação de diversos plugins para atender aos requisitos específicos do projeto. Embora houvesse um grande apreço e amor pelo Vim, essa opção foi desconsiderada com pesar, juntamente com a linguagem de programação C. Cabe ressaltar que tal caminho foi considerado como uma possibilidade a ser explorada, porém, em virtude das demandas e especificidades do projeto em questão, optou-se por seguir por outro caminho. É importante frisar que essa decisão foi tomada de forma cuidadosa e respeitosa, levando em consideração a complexidade do projeto e as possibilidades disponíveis para a sua realização.

Foram utilizadas as bibliotecas JACo e Jansi. A primeira foi utilizada como uma maneira fácil, eficiente e completa de reproduzir arquivos de áudio .mp3, que foram inseridos para uma maior imersão do jogador na experiência. A segunda foi utilizada para evitar que emuladores de terminal não compatíveis com sequências de comandos ANSI acabem não representando as saídas do código de maneira adequada.

Foi adotada a representação da caverna em formato de grafo, a qual se mostra útil e prática para a implementação de um sistema de mapeamento e navegação. Cada vértice do grafo corresponde a uma câmara da caverna, enquanto as arestas correspondem a túneis que conectam duas câmaras. Cada câmara é caracterizada por uma série de informações relevantes, como a descrição do local feita para dar imersão ao jogador, o conteúdo que pode ser "garimpado", a representação como coordenada que define onde a câmera está localizada na caverna (informação utilizada internamente no sistema, mas não acessível ao usuário), bem como se é ou não possível sair da caverna através daquele ponto. O conteúdo e a descrição são escolhidos randomicamente, assim tornando mais interessante para o jogador jogar novamente. Além disso, cada câmara também é associada a uma lista de túneis conectados a ela, permitindo a navegação pelo grafo. Para a geração da caverna, foi adotado o algoritmo de Kruskal[4] adaptado, no qual se gerou inicialmente um número x de câmaras que compõem a caverna. Em seguida, foram estabelecidas as conexões entre elas, garantindo que sempre é possível chegar à

saída da caverna. Tal processo se mostrou eficiente e possibilitou a criação de uma variedade de cenários de cavernas distintos e interessantes.

5 Conclusão

Gold Diggers é um jogo singular e inovador que foi desenvolvido em para a disciplina de algoritmos e estrutura de dados II, com foco particular no estudo de grafos e algoritmos de *pathfinding*, notadamente o algoritmo A*. A escolha de representar a caverna como um grafo se mostrou uma decisão muito bem-sucedida, fornecendo uma riqueza de possibilidades e funcionalidades que foram exploradas com sucesso.

Diferente de jogos atuais, Gold Diggers brinca com sentidos humanos e desafia o jogador dessa maneira. Sua mecânica é rápida, porém, deve ser lida aos poucos para evitar confusões. A jogabilidade rápida e intensa é complementada por uma estética cuidadosamente elaborada que visa criar uma sensação de isolamento e desconcerto no jogador. A música, os textos e as artes se combinam de maneira harmoniosa e passam o sentimento de solidão e confusão no jogador. Sendo assim uma experiência coesa e envolvente.

Em última análise, a aplicação desenvolvida demonstrou a enorme utilidade e versatilidade dos sistemas de grafos em diversas áreas, como jogos eletrônicos ou mapeamento de cavernas, abrindo um amplo leque de possibilidades para os estudantes de algoritmos e estrutura de dados II explorarem.

6 Referências

References

[1] behlanmol. *Comparison between Adjacency List and Adjacency Matrix Representation of Graph*. `https://www.geeksforgeeks.org/comparison-between-adjacency-list-and-adjacency-matrix-representation-of-graph/`. Accessado em Março de 2023. 2021.

[2] Polyanna Possani da Costa. "Teoria de Grafos e suas Aplicações". Mestrado Profissional em Matemática Universitária. Rio Claro: Universidade Estadual Paulista "Júlio de Mesquita Filho", 2013, p. 80.

[3] Jim Webber. *Graphs for Artificial Intelligence and Machine Learning*. https://neo4j.com/blog/graphs-for-artificial-intelligence-and-machine-learning/. Acesso em: mar. 2023. 2021.

[4] Jon Kleinberg. *Algorithm Design*. 1st. Boston: Pearson/Addison-Wesley, 2006, pp. 142–151. ISBN: 0-321-29535-8.

[5] Carlos Ant^o*nioRuggieroEtal.*. "On the efficiency of data representation on the modeling and characterization of complex networks". In: *Physica A: Statistical Mechanics and its Applications* 390.11 (2011), pp. 2172–2180. DOI: 10.1016/j.physa.2011.02.011.

[6] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition.* Addison-Wesley, 2011, pp. I–XII, 1–955. ISBN: 978-0-321-57351-3.

[7] Gabriel Valiente. "Adjacency Maps and Efficient Graph Algorithms". In: *Algorithms* 15.2 (2022). ISSN: 1999-4893. DOI: 10.3390/a15020067.

[8] Igor Medeiros Vanderlei. *AED II 28 04 parte 2*. https://www.youtube.com/watch?v=SRWY_kn5ubg. Acesso em: mar. 2023. 2022.