Aplicação da Teoria de Grafos em Jogos Virtuais

João Pedro Jerônimo Thalis Ambrosim Falqueto
Bruno Ferreira Salvi

November 2024

Sumário

1 Introdução

Desde os anos 20, a Teoria dos Grafos vem sido amplamente utilizada em diversas áreas da ciência, computação e matemática, como, por exemplo: elaboração de rotas, problemas de combinatória, machine learning, entre muitas outras utilizações. Este documento trás mais um destes usos para a Teoria dos Grafos, a aplicação em **Jogos Virtuais**. Não só trazer a ideia **teórica** destas aplicações, como exemplos e aplicações computacionais.

Alguns códigos são grandes e não seria prático colocá-los em sua integridade dentro do documento. Em virtude disto, certos programas serão colocados apenas em partes, visando uma exemplificação, porém todos os algoritmos descritos podem ser encontrados neste repositório do github:

`https://github.com/Os-Estudiosos/graphs-theory-in-games`

2 Pathfinding em jogos 2D

Dado o mapa de um jogo, um algoritmo de pathfinding são sequências de comandos responsáveis por achar o menor caminho de um ponto A até um ponto B no mapa, levando em conta a obstrução do caminho por obstáculos e, dependendo do algoritmo, engloba outras variáveis. São muito utilizados em jogos virtuais em situações onde um inimigo tem que chegar até o jogador.

Visando a integração da teoria de grafos em algoritmos de pathfinding, algumas definições, não rigorosas, relacionadas aos jogos 2D tem de ser estabelecidas:

Jogo 2D é um gênero de jogo caracterizado pela ausência de uma profundidade, onde a movimentação do jogador é restringida ao espaço bidimensional (Cima, baixo, esquerda e direita).

Definição 2.1 (Entidade)**.** *Uma* Entidade *é todo e qualquer objeto com comportamento próprio dentro de um jogo*

Exemplo: *O jogador, um inimigo, um chefe, uma parede, uma armadilha, etc.*

Definição 2.2 (Tile)**.** *Um* Tile *é um quadrado $x \times y$ no plano cartesiano $\mathbb{R}^2$ com um conjunto* C *associado, tudo na área delimitada pelo quadrado é* ***elemento*** *desse cojunto*

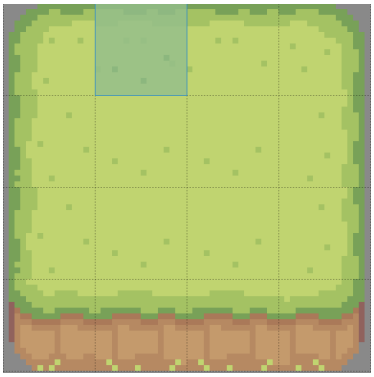
Definição 2.3 ($\psi(T)$)**.** *$\psi(T) = C$ é a função que recebe como parâmetro um Tile* T *e retorna o conjunto dos elementos dentro da área do Tile (*C*)*

Definição 2.4 (Obstáculo)**.** *Um* Obstáculo *é uma entidade tal que, se outra entidade quer chegar de um ponto x a um ponto y, a mesma tem que desviar do* obstáculo

Exemplo: *Uma parede, uma rocha, uma árvore*

Definição 2.5 (Mapa 2D)**.** *Dado um conjunto de m.n tiles, de tal forma que cada tile pode ser empilhado a formar um retângulo M, chamamos M de* Mapa 2D

Notações: $(T_{ij})_{n\times m}$, $\bigcup_{j=1}^{m}\bigcup_{i=1}^{n} T_{ij}$
Exemplo:



Com estes conceitos, podemos estudar como associar teoria dos grafos com as definições antes estabelecidas

2.1 Transformando um Mapa 2D em um Grafo

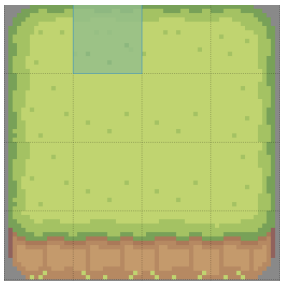
Definição 2.6 (Grafo Associado)**.** *Dado um Mapa 2D* $M = (T_{ij})_{m\times n}$*, podemos criar um grafo* $G(V, E)$*, onde* V *é o conjunto de vértices e* E *o conjunto de arestas, de tal forma que cada vértice* v *é representado por um Tile* T_{ij} *e os vizinhos desse vértice são:*

$\rightarrow v = T_{ij} \implies N(v) = \{T_{sw}/s = i-1, i+1 \wedge w = j-1, j+1\}$

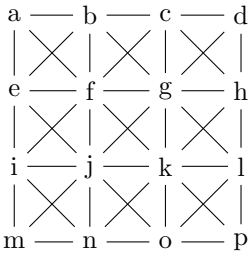
Chamamos esse grafo de Grafo Associado a M

Estas arestas mostram para uma *Entidade* arbitrária, para quais tiles ela pode seguir a partir do tile em que ela está contida

Exemplo Visual:



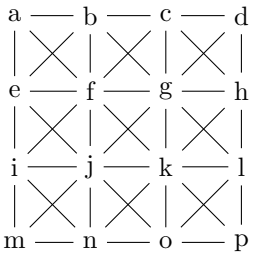
$\implies$



2.2 Adicionando Entidades no Mapa

Dado um Mapa 2D $(T_{ij})_{m\times n}$, se quisermos adicionar uma entidade no mapa, associamos-a com o conjunto de um Tile arbitrário, veja o exemplo:

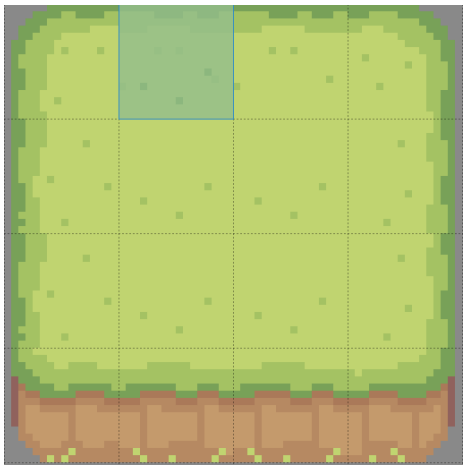
$\Longrightarrow$



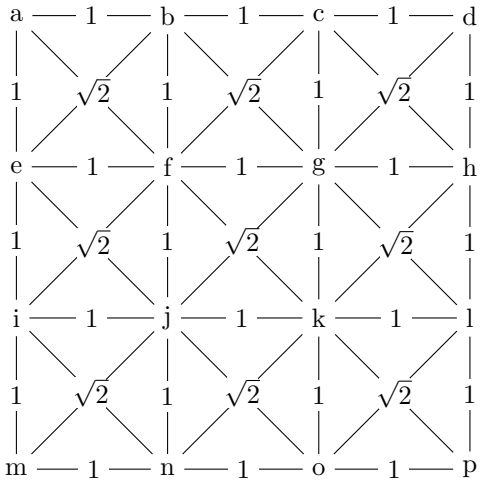
Veja que o Tile representado pelo vértice g tem uma entidade *Cerca* em sua área, logo, podemos dizer que $\psi(T_{2,3}) = \{Cerca\}$

2.3 Aplicando pesos nas arestas

Para utilizar algoritmos de pathfinding, temos que definir, antes de tudo, a dificuldade de uma entidade se locomover de um tile para outro, para isso, vamos atribuir pesos nas arestas, estes indicando a distância entre o *centro* de um tile até o outro.



$\Longrightarrow$



2.4 Primeiro Algoritmo: Dijkstra

Dado um Mapa 2D M e seu grafo associado $G(V, E)$, vamos dizer que existem duas entidades neste mapa, P (Player) e E (Enemy), queremos achar o menor caminho para levar a entidade E até P, um possível algoritmo para isso, é o algoritmo de **Dijkstra**

Lema 2.7. *O algoritmo de Dijkstra encontra o menor caminho entre dois vértices de um grafo*

Demonstração. Página 407, Discrete Mathematics 8ª edição por Richard Johnsonbaugh[1]

Proposição 2.8. *O algoritmo de Dijkstra encontra a menor distância e o menor caminho entre duas entidades*

Demonstração. Dado um mapa $M = (T_{ij})_{m \times n}$, seu grafo associado $G(V, E)$ e duas entidades I e P, contidas em dois tiles diferentes de M, pegamos os dois vértices associados aos tiles que contém as entidades I e P e aplicamos o algoritmo de *dijkstra*. Sabemos pelo lema *2.7* que dijkstra nos retorna o menor caminho/distância entre dois pontos no grafo, consequentemente, ele vai nos retornar a menor distância e caminho entre os Tiles

Há um problema neste algoritmo, dado um Grafo $G(V, E)$ com n vértices, a complexidade do algoritmo é de $\mathcal{O}(n^2)$, logo, se há um Mapa M com $m \times n$ tiles, o grafo associado a M tem $m \times n$ vértices, ou seja, a complexidade do algoritmo será de $\mathcal{O}(m^2 \times n^2)$, o que pode causar problemas de peformance se o mapa M tem muitos tiles!

É para resolver este problema, que há a aplicação de outro algoritmo!

2.5 Segundo Algoritmo: A* Search Algorithm

O Algoritmo A* também encontra o menor caminho dentro de um grafo com pesos. (Para uma explicação completa, confira *O algoritmo A*: Um guia completo*[2]).

Resumindo o funcionamento do algoritmo, ele toma como base 3 funções:

- $g(v)$: Função que indica a distância total do nó inicial até o nó analisado;
- $h(v)$: Função que estima a distância ideal do nó atual até o ponto final (Chamada de *função eurística*). Esta varia de contexto a contexto onde o grafo é aplicado;
- $f(v)$: Função-chave na tomada de decisão do algoritmo, definida por $f(v) = g(v) + h(v)$

A mais importante neste artigo é a *função eurística*. Tendo em vista que ela estima a distância do nó atual até o final. Ao analisar a aplicação deste algoritmo no grafo de um mapa M, lembre-se que M é um subconjunto do $\mathbb{R}^2$, pode-se aplicar, como função eurística do algoritmo, uma medida de distância, como distância euclidiana

Mas ele resolve o problema de complexidade? Sim!

Lema 2.9. *A complexidade em **Big-O** do algoritmo de busca A* aplicado em um grafo $G(V, E)$ é, no pior caso de perfomance, $\mathcal{O}(|E| \times log(|V|))$*

[1] Richard Johnsonbaugh: Discrete Mathematics, 2023.

[2] Rajesh Kumar: O algoritmo A-star: Um guia completo, Acessado em 19 de novembro de 2024, 2024, URL: `https://www.datacamp.com/pt/tutorial/a-star-algorithm`.

Demonstração. Veja o artigo *Time Complexity of A with Approximate Heuristics onMultiple-Solution Search Spaces*[3]

Veja este exemplo de aplicação do algoritmo A

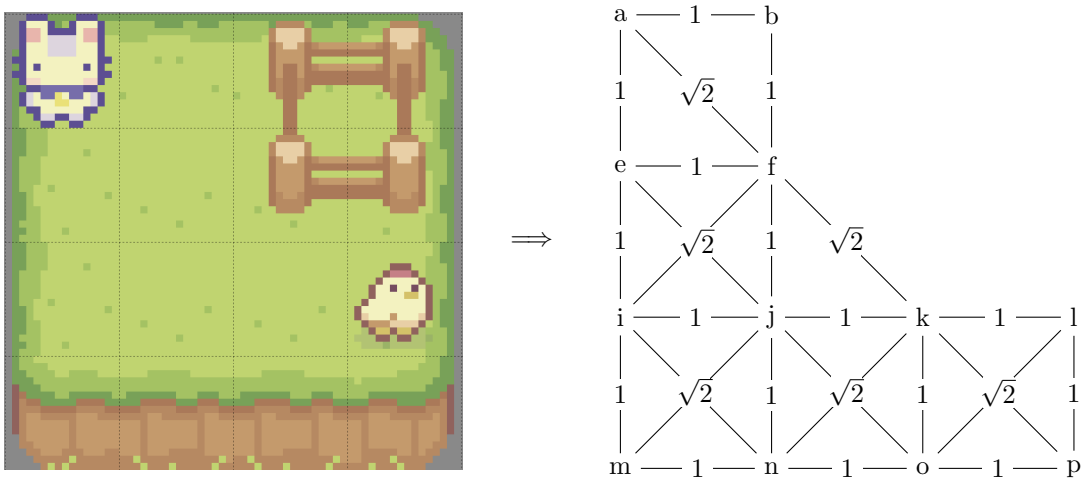
2.6 Algumas outras ideias de aplicação

Anteriormente, os grafos foram utilizados apenas para calcular o caminho mais curto de um tile até outro, porém, algumas outras ideias podem ser elaboradas

2.6.1 Encontrando caminhos entre obstáculos

Dado um Mapa 2D, ele, com certeza, não estará vazio, pode haver muitos obstáculos entre duas entidades, o que pode ser feito para obter um resultado efetivo dos algoritmos anteriormente apresentados? Uma solução simples, eficaz e fácil, é a remoção dos vértices associados aos tiles T_{ij} tais que $\exists O \in \psi(T_{ij})$ com O sendo um obstáculo qualquer

Exemplo

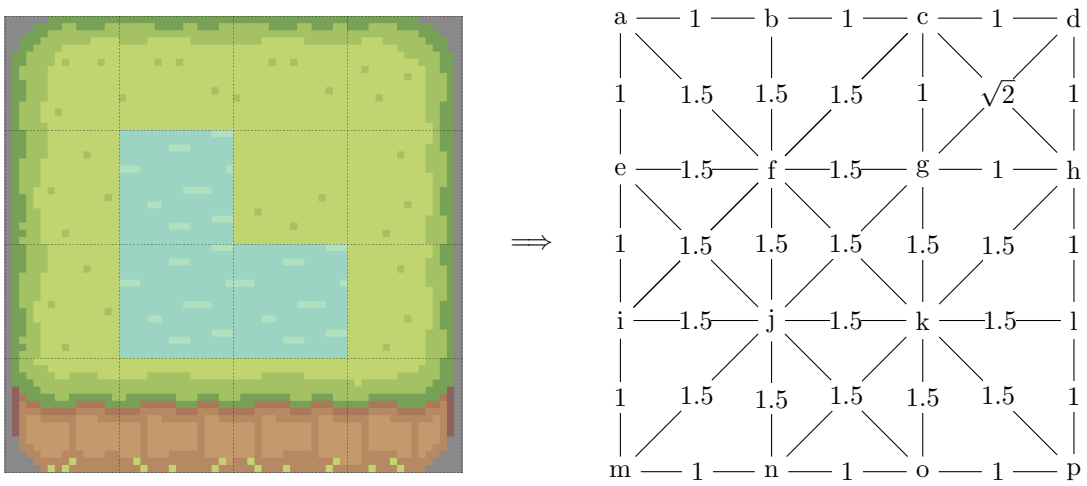


2.6.2 Levando em consideração o terreno do mapa

Dependendo do escopo do jogo feito, o mapa pode conter vários tipos de terreno, como água, lama, rochas, etc. E, dependendo da mecânica do jogo, cada um desses pode influenciar na dificuldade de ir para o próximo tile. Tendo isso em mente, um meio de aplicar este conceito, é mudar o peso de uma aresta $\{v_i, v_j\}$ de acordo com o *tipo do tile* associado a v_i ou v_j

[3]Hang Dinh et al.: The Time Complexity of A-star with Approximate Heuristics on Multiple-Solution Search Spaces, in: Journal of Artificial Intelligence Research 1 (2012), pp. 1–45, URL: `https://www.jair.org/index.php/jair/article/view/10793`.

Exemplo



Nota: Algumas arestas foram removidas para melhor visualização

2.7 Exemplos Computacionais

Todos os códigos citados estão dentro do repositório mencionado na introdução do documento, especificamente, dentro da pasta **pathfinding**

2.7.1 Inimigo desviando de paredes com Dijkstra e A

No arquivo *dijkstra_pathfinding.py*, há um exemplo de código com um MAPA 2D 500x500, onde há um inimigo (vermelho) que percebe o jogador (azul), e desvia das paredes para chegar no objetivo (paredes cinza). A seguir, um código que utiliza dijkstra para achar o caminho

2.7.2 Inimigo desviando de paredes usando A

O arquivo *a_star_pathfinding.py*, possui um exemplo parecido com o anterior, com diferença que utiliza do algoritmo A

2.7.3 Decidindo com base no terreno

No arquivo *terrain_pathfinding.py*, há um exemplo de código que mostra a tomada de decisão do algoritmo do pathfinding baseado no terreno, onde os tiles azuis são água, e os cinzas são lamas, todos diminuem a velocidade das entidades que ficam em cima deles.

3 Crafting

Crafting é o processo de criação de novos itens em um jogo, seja em duas ou três dimensões, a partir da combinação de outros itens ou materiais pré-existentes. Esse mecanismo segue regras específicas de receitas, onde itens básicos ou intermediários são transformados em itens mais avançados.

Para entender melhor o conceito de crafting otimizado usando a teoria dos grafos, apresentamos as seguintes definições.

Definição 3.1 (Item)**.** *Qualquer objeto ou recurso presente no jogo, utilizado como material para crafting, consumido durante ações, ou interagido pelo jogador. Itens podem ser básicos (como materiais primários) ou produtos finais mais complexos.*

3.1 Modelagem do crafting com grafos

Seja G um grafo $G = (V, E)$, onde V representa os itens e E as arestas direcionadas que conectam os itens de entrada aos itens criados, onde cada aresta direcionada indica que o item de origem é necessário para criar o item de destino. Um exemplo de uma estrutura de crafting é representada pelo grafo abaixo:

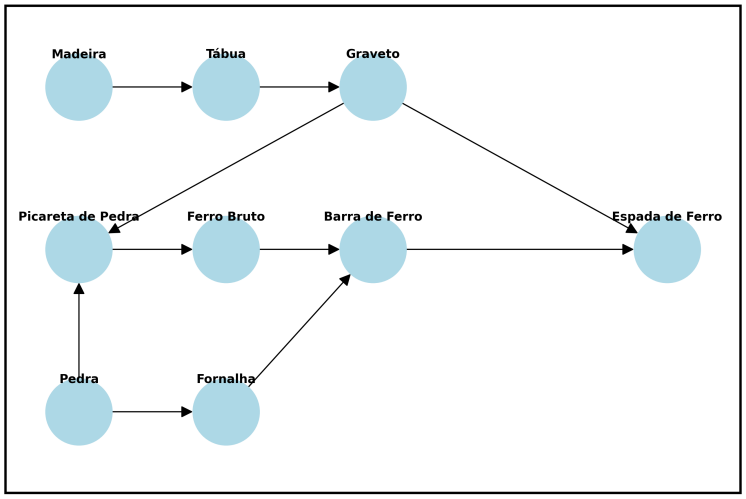


Figure 1: Grafo representando o sistema de crafting.

3.2 Propriedades dos Grafos de Crafting

Para entender melhor as características dos grafos que modelam sistemas de crafting em jogos, apresentamos algumas propriedades fundamentais, acompanhadas de suas respectivas demonstrações.

3.2.1 Grafos de Crafting são DAGs

Proposição 3.2. *Seja $G = (V, E)$ o grafo representando um sistema de crafting. Então G é um **DAG** (grafo direcionado acíclico).*

Demonstração. Por construção, cada aresta direcionada em G representa uma dependência de crafting, onde um item (vértice de origem) é usado como entrada para criar outro item (vértice de destino).

1. Direcionalidade: As arestas têm direção definida, pois representam relações unidirecionais de dependência no crafting (um item só pode ser produzido a partir de outros e nunca o contrário simultaneamente). Logo, G é um grafo direcionado.

2. Ausência de ciclos: Se G possuísse um ciclo, isso implicaria que um item dependeria direta ou indiretamente de si mesmo para ser criado. Por exemplo, $A \to B \to A$, o que é ilógico em sistemas de crafting bem definidos. Portanto, G é acíclico.

Assim, G é um grafo direcionado acíclico (DAG).

3.2.2 Existência de Vértices Fonte e Sumidouro

Proposição 3.3. *Todo grafo de crafting possui pelo menos um **vértice fonte** (vértice sem arestas de entrada) e pelo menos um **vértice sumidouro** (vértice sem arestas de saída).*

Demonstração. 1. Vértices fonte: Um vértice sem arestas de entrada representa um item que não depende de nenhum outro para existir. Esses vértices correspondem aos materiais básicos do sistema de crafting, como "Madeira" ou "Pedra". Se G não tivesse vértices fonte, todos os itens dependeriam de outros para serem criados, formando um ciclo, o que contradiz a propriedade de que G é acíclico (conforme demonstrado anteriormente). Logo, G possui pelo menos um vértice fonte.

2. Vértices sumidouro: Um vértice sem arestas de saída representa um item que não é utilizado para criar outros itens. Esses vértices correspondem aos produtos finais do sistema de crafting, como "Espada de Ferro". Se G não tivesse vértices sumidouro, todos os itens precisariam ser usados para produzir outro, e não haveria produtos finais, o que não faz sentido no contexto de crafting. Assim, G possui pelo menos um vértice sumidouro.

Portanto, todo grafo de crafting possui pelo menos um vértice fonte e pelo menos um vértice sumidouro.

3.3 Exemplo Computacional

O exemplo computacional apresentado encontra os itens que requerem a maior quantidade de recursos para serem craftados, utilizando o algoritmo DFS aplicado em um grafo direcionado acíclico (DAG). A dificuldade de crafting é definida pelo número total de itens únicos (diretos e indiretos) necessários para produzir cada item, garantindo que itens compartilhados entre diferentes caminhos sejam contabilizados apenas uma vez.

No caso do exemplo dado para crafting, a saída retorna que o item que precisa da maior quantidade de itens para ser craftado é a Espada de ferro, com um total de 8 itens necessários.

O código citado está dentro do repositório, especificamente na pasta "crafting".

4 Geração Procedural de Mapas com Grafos

A geração procedural de mapas é uma técnica amplamente utilizada em jogos para criar layouts variados de forma automática, aumentando a rejogabilidade e o dinamismo do jogo. Essa técnica pode ser diretamente modelada utilizando a teoria dos grafos, representando os mapas como grafos direcionados ou não direcionados, dependendo da estrutura do mundo do jogo.

4.1 Representação de Mapas como Grafos

Um mapa gerado proceduralmente pode ser representado como um grafo $G = (V, E)$, onde V é o conjunto de vértices, representando salas, regiões ou áreas navegáveis no mapa, e E, o conjunto de arestas, representando conexões, como corredores ou passagens entre as regiões. No geral, todo mapa procedural possui fonte e sumidouro, significando, respectivamente, a entrada/começo do mapa e o saída/fim do mapa[4].

Além disso, para uma maior diversificação, podem ser adicionados pesos nas arestas, representando a dificuldade de passar de um cômodo(vértice) para o outro. Abaixo, representamos um exemplo genérico de mapa procedural[5], feito sobre as bibliotecas do Pygame e Networkx[6]:

[4] Vazgriz YT: Procedurally Generated 3D Dungeons, Acessado em 21 de novembro de 2024, 2024, URL: `https://www.youtube.com/watch?v=rBY2Dzej03A`.

[5] Graham Cox: Procedural Generation of Computer Game Maps, in: Baeldung CS 2024, URL: `https://www.baeldung.com/cs/gameplay-maps-procedural-generation`.

[6] Wellington Barbosa: Introduction to Graphs in Python with networkx, 2024, URL: `https://www.youtube.com/playlist?list=PLxKmxm_IWVlQPUrPJOQrstBg6Cit-aCQT`.

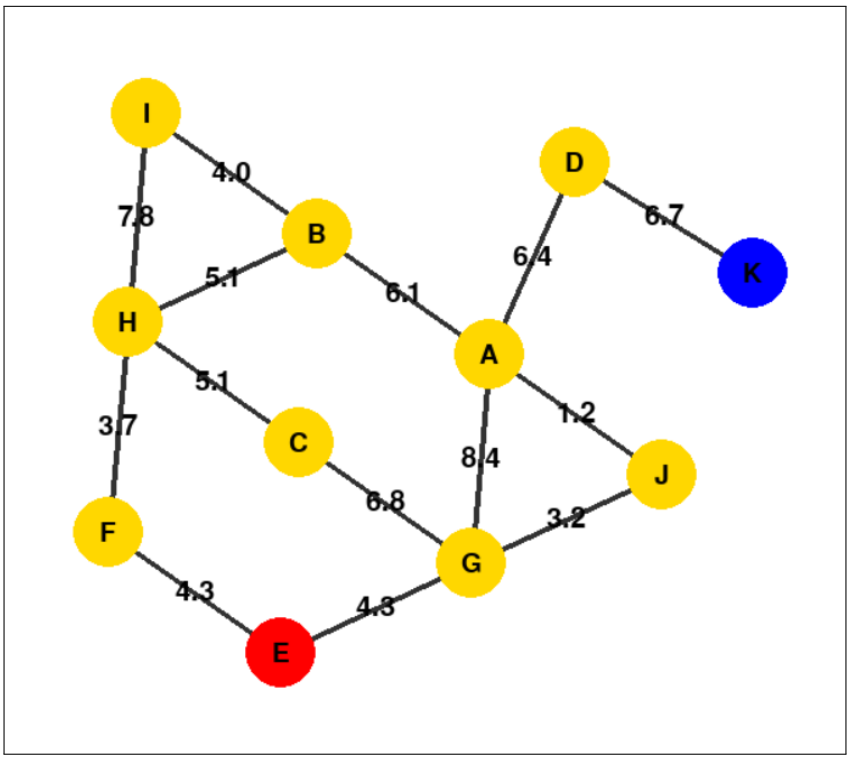


Figure 2: Grafo representando a criação aleatória de um mapa procedural, atribuindo peso às arestas.

4.2 Propriedades de Mapas Procedurais

Para compreender as principais características dos grafos utilizados na criação de mapas procedurais, destacamos algumas propriedades essenciais, acompanhadas de suas respectivas explicações e demonstrações.

4.2.1 Conectividade

Proposição 4.1. *Em um mapa procedural representado como um grafo* $G = (V, E)$*,* G *é sempre conexo.*

Demonstração. Para que um mapa seja jogável, todas as salas ou regiões devem ser acessíveis. Isso significa que, para qualquer par de vértices $u, v \in V$, existe pelo menos um caminho em G conectando u a v.

Nos algoritmos de geração procedural, a conectividade é garantida por construção:

- Inicialmente, os vértices são adicionados ao grafo.

- Em seguida, as arestas são inseridas para conectar os vértices, garantindo que nenhum deles fique isolado. Normalmente, isso é feito conectando cada novo

vértice a pelo menos um vértice já existente.

Mesmo em casos onde conexões adicionais são feitas (como a criação de ciclos), a conectividade geral do grafo é mantida. Como resultado, G é conexo por definição.

4.2.2 Grau dos Vértices

Proposição 4.2. *O grau médio dos vértices em um grafo $G = (V, E)$, que representa um mapa procedural, é dado por grau médio $= \frac{2|E|}{|V|}$, onde $|V|$ é o número de vértices e $|E|$ é o número de arestas.*

Demonstração. O grau de um vértice $v \in V$ é definido como o número de arestas incidentes a ele, e, no contexto de mapas de jogos, é útil para saber se existe algum ciclo, como será provado num teorema abaixo. Para um grafo $G = (V, E)$, a soma dos graus de todos os vértices $v \in V$ é igual ao dobro do número total de arestas $|E|$, conforme a seguinte relação:

$$\sum_{v \in V} \deg(v) = 2|E|,$$

onde $\deg(v)$ representa o grau de v.

O grau médio dos vértices é então dado pela soma dos graus dividida pelo número total de vértices $|V|$:

$$\text{grau médio} = \frac{\sum_{v \in V} \deg(v)}{|V|}.$$

Substituindo a relação acima:

$$\text{grau médio} = \frac{2|E|}{|V|}.$$

Portanto, o grau médio dos vértices depende diretamente do número de arestas $|E|$ e do número de vértices $|V|$, e é sempre bem definido para grafos finitos.

4.2.3 Número de Ciclos em um Grafo Conexo

Um ciclo, no contexto de mapas, é útil porque oferece ao jogador rotas alternativas para alcançar uma mesma sala (vértice) por diferentes caminhos. Isso adiciona dinamismo ao mapa, proporcionando maior liberdade de exploração e opções estratégicas para o jogador. Além disso, ciclos podem enriquecer a jogabilidade ao permitir atalhos, esconder segredos, ou oferecer rotas com níveis de dificuldade variados.

Proposição 4.3. *Seja $G = (V, E)$ um grafo conexo, onde $|V|$ é o número de vértices e $|E|$ é o número de arestas. O número de ciclos em G é dado por:*

$$C = |E| - |V| + 1.$$

Demonstração. A prova será feita em duas etapas:

1. **Caso Base (Árvores):** Uma árvore é um grafo conexo e acíclico. Por definição, uma árvore com $|V|$ vértices possui $|E| = |V|-1$ arestas. Substituindo na fórmula:

$$C = |E| - |V| + 1 = (|V| - 1) - |V| + 1 = 0.$$

Assim, $C = 0$, o que confirma que árvores não possuem ciclos.

2. **Caso Geral (Grafos Conexos com Ciclos):** Para qualquer grafo conexo G, a adição de x arestas além de uma árvore (que já possui $|V| - 1$ arestas) forma exatamente x ciclos. Seja $x = |E| - (|V| - 1)$, então:

$$C = x = |E| - |V| + 1.$$

Essa expressão é válida para qualquer grafo conexo, pois cada aresta extra forma exatamente um novo ciclo.

Portanto, mostramos que a fórmula $C = |E| - |V| + 1$ é válida para todos os grafos conexos.

4.2.4 Limitação na quantidade de arestas

No contexto de mapas em jogos, a existência de arestas paralelas entre dois vértices não é razoável, pois cada aresta deve representar um caminho único entre duas salas ou regiões, e é ilógico existir duas arestas, advindas da mesma sala, ligando-se ao mesmo vértice.[7] Por motivo semelhante, também é inviável a existência de laços em um grafo de mapa procedural. Essa característica permite criar um limite inferior e um limite superior na quantidade de arestas de cada mapa.

Para garantir que o grafo seja conexo, o número mínimo de arestas necessárias é $n - 1$, que corresponde a uma árvore, a estrutura mais simples que conecta todos os vértices sem formar ciclos. Por outro lado, como o grafo é simples, o número máximo de arestas possíveis é $\frac{n(n-1)}{2}$, que corresponde a um grafo completo, no qual todos os vértices estão conectados entre si.

Além disso, cada aresta adicionada após atingir $n - 1$ arestas introduz exatamente um ciclo no grafo. Isso ocorre porque, ao conectar dois vértices que já estavam indiretamente conectados, a nova aresta fecha um caminho, formando um ciclo.

Assim, em um grafo de mapa procedural, o número de arestas varia entre $n - 1$ e $\frac{n(n-1)}{2}$, dependendo da quantidade de ciclos presentes no grafo.

Teorema 4.4. *Existe um ciclo no mapa se e somente se o grau médio dos vértices é maior que 2*

Demonstração. $\implies$) Se existe um ciclo, então o grau médio dos vértices é maior que 2.

[7] Jasonde1aat: Procedural Generation: Graph Rewriting, Acessado em 21 de novembro de 2024, 2022, URL: `https://www.lexaloffle.com/bbs/?tid=50076`.

Em um grafo conexo $G = (V, E)$, o número de ciclos é dado por $C = |E| - |V| + 1$. Se $C > 0$, então $|E| > |V| - 1$. A fórmula do grau médio é:

$$\text{grau médio} = \frac{2|E|}{|V|}.$$

Substituímos a condição $|E| > |V| - 1$ na fórmula do grau médio:

$$\frac{2|E|}{|V|} > \frac{2(|V| - 1)}{|V|}.$$

Simplificando a fração do lado direito:

$$\frac{2(|V| - 1)}{|V|} = 2 - \frac{2}{|V|}.$$

Assim, $\frac{2|E|}{|V|} > 2 - \frac{2}{|V|}$. Como $|V| \geq 2$, temos $\frac{2}{|V|} > 0$. Logo, $\frac{2|E|}{|V|} > 2$, e o grau médio é maior que 2.

$\Longleftarrow$) Se o grau médio dos vértices é maior que 2, então existe um ciclo.

Da fórmula do grau médio, multiplicando ambos os lados por $|V|$:

$$2|E| > 2|V|.$$

Dividindo por 2:

$$|E| > |V|.$$

Em um grafo conexo e sem ciclos (uma árvore), temos $|E| = |V| - 1$. A condição $|E| > |V| - 1$ implica que o grafo não é uma árvore, pois possui pelo menos uma aresta extra.

Cada aresta extra em um grafo conexo cria um ciclo (como mostrado na fórmula $C = |E| - |V| + 1$). Logo, $C > 0$, o que significa que existe pelo menos um ciclo.

4.3 Exemplo Computacional

No contexto de mapas procedurais, um algoritmo útil é o que gera o próprio mapa proceduralmente, indicando o possível caminho mais curto e uma diferença mínima nas arestas de entrada. Por isso, apresentamos um código que gera um mapa aleatoriamente, promovendo equilíbrio na jogabilidade e adaptabilidade em diferentes cenários.

O código citado está dentro do repositório, em particular na pasta "proceduralmaps".

5 Conclusão

Neste trabalho, exploramos como a Teoria dos Grafos pode ser aplicada de forma prática e inovadora em diferentes aspectos dos jogos virtuais, incluindo algoritmos de pathfinding, sistemas de crafting e geração procedural de mapas.

Esses conceitos demonstram como a modelagem matemática, quando associada à computação, pode resolver problemas complexos e otimizar diversas mecânicas fundamentais no desenvolvimento de jogos.

Os algoritmos discutidos, como Dijkstra, A* e DFS, ilustraram como os grafos podem ser utilizados para calcular o menor caminho em mapas bidimensionais, considerando obstáculos e outras variáveis, enquanto a modelagem de crafting mostrou como dependências entre recursos podem ser representadas de maneira lógica e eficiente através de DAGs. Além disso, a geração procedural de mapas reafirma o potencial dos grafos em criar ambientes dinâmicos e imersivos.

Com isso, fica evidente que o uso da Teoria dos Grafos está presente de diversas formas em jogos virtuais e que, além disso, só enriquece a jogabilidade como também amplia as possibilidades criativas e técnicas no design de jogos.

Referências

Barbosa, Wellington: Introduction to Graphs in Python with networkx, 2024, URL: https://www.youtube.com/playlist?list=PLxKmxm_IWVlQPUrPJOQrstBg6CitaCQT.

Cox, Graham: Procedural Generation of Computer Game Maps, in: Baeldung CS 2024, URL: https://www.baeldung.com/cs/gameplay-maps-procedural-generation.

Dinh, Hang et al.: The Time Complexity of A-star with Approximate Heuristics on Multiple-Solution Search Spaces, in: Journal of Artificial Intelligence Research 1 (2012), pp. 1–45, URL: https://www.jair.org/index.php/jair/article/view/10793.

Jasonde1aat: Procedural Generation: Graph Rewriting, Acessado em 21 de novembro de 2024, 2022, URL: https://www.lexaloffle.com/bbs/?tid=50076.

Johnsonbaugh, Richard: Discrete Mathematics, 2023.

Kumar, Rajesh: O algoritmo A-star: Um guia completo, Acessado em 19 de novembro de 2024, 2024, URL: https://www.datacamp.com/pt/tutorial/a-star-algorithm.

YT, Vazgriz: Procedurally Generated 3D Dungeons, Acessado em 21 de novembro de 2024, 2024, URL: https://www.youtube.com/watch?v=rBY2Dzej03A.