UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Ingrid Danielle Vilela Costa

Um Algoritmo Para Geração de Navigation Meshes em Mapas Bidimensionais Homogêneos: uma aplicação no jogo *Dragon Age: Origins*

Recife

2019

Ingrid Danielle Vilela Costa

Um Algoritmo Para Geração de Navigation Meshes em Mapas Bidimensionais Homogêneos: uma aplicação no jogo *Dragon Age: Origins*

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Silvana Bocanegra

Recife

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

C837a Costa, Ingrid Danielle Vilela
Um Algoritmo Para Geração de Navigation Meshes em Mapas Bidimensionais Homogêneos: uma aplicação no jogo Dragon Age: Origins / Ingrid Danielle Vilela Costa. - 2019.
56 f. : il.

Orientadora: Silvana Bocanegra.
Inclui referências.

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, , Recife, 2019.

1. otimização de mapas. 2. algoritmos de busca. 3. caminho mínimo. 4. navegação robótica. I. Bocanegra, Silvana, orient. II. Título

CDD

Ingrid Danielle Vilela Costa

Um Algoritmo Para Geração de Navigation Meshes em Mapas Bidimensionais Homogêneos: uma aplicação no jogo Dragon Age: Origins

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada em: 13 de Dezembro de 2019.

BANCA EXAMINADORA

Silvana Bocanegra
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Marcelo Gama
Departamento de Estatísitca e Informática
Universidade Federal Rural de Pernambuco

À minha família

Agradecimentos

Agradeço primeiramente a Deus pela minha vida e por ter proporcionado todos esses desafios durante essa trajetória acadêmica e me dado sabedoria, paciência e forças para não desistir mesmo quando não enxergava mais esperança.

Agradeço à minha adorável família pela compreensão, incentivo e paciência(na verdade falta dela) quando o assunto era a vida acadêmica. Dedico esse trabalho especialmente ao meu filho, Aziz, minha alegria, que mesmo sem entender a importância do projeto foi a minha maior motivação para seguir, também dedico ao meu querido esposo Arthur, por todas as noites mal dormidas em que passamos horas discutindo o projeto, por toda a paciência e o incentivo para continuar e finalizar mais essa etapa na minha vida. À minha mãe Socorro, só tenho a agradecer e não tem nada nesse mundo que recompense por tudo que ela fez, e faz, por mim e pelo meu filho, todos os tormentos que passamos juntas para que nada atrapalhasse a conclusão tanto do ensino médio quanto da universidade e no final valeu a pena. Agradeço ao meu pai Adilson e a minha irmã Oriana por sempre estarem comigo e dando muita força e apoio para concluir e partir para novos desafios.

Agradeço a professora Silvana por ter me acolhido nessa reta final não só pela orientação, mas o apoio e incentivo para concluir. Aos meus colegas incríveis que fiz na universidade e que hoje são meus amigos de muitas risadas e broncas. Mirela, Tancicleide, Jessica e Rotsen, passamos por muitos obstáculos, nos separamos no meio do curso, mas sempre procurando manter o contato e o velho 'Bora marcar um encontro?' que sempre achavamos um jeito de estar juntos e rir um do outro.

Para finalizar gostaria de agradecer a todos de contribuíram direta e indiretamente para a realização desse trabalho e aproveito para me desculpar pela minha falta de paciência, meu desespero, ansiedade e pelas ausências em conversas e encontros, enfim, tudo isso foi em prol de um bem maior.

“O único modo de escapar da corrupção causada pelo sucesso é continuar trabalhando.”

(Albert Einstein)

Resumo

Nos cenário dos jogos eletrônicos e, mais recentemente, na robótica, agentes autônomos comumente necessitam resolver repetidamente o problema de busca de menor caminho. Esta necessidade eventualmente pode consumir muitos recursos e demanda otimizações para que estas buscas sejam mais eficientes. Tais otimizações podem incluir melhorias nos algoritmos de busca, na representação dos mapas, nas estruturas de dados utilizadas, entre outras. O presente trabalho apresenta uma otimização para algoritmos de busca baseada na redução do tamanho do espaço de busca, através de um algoritmo de geração automática de Navigation Meshes, que são redes de áreas caminháveis de mapas, implicando em redução do espaço de busca e consequentemente melhoria de tempo de processamento. A geração de Navigation Meshes é um problema ainda sem solução estabelecida. Para avaliar a heurística foram solucionados problemas de caminho mínimo em 156 mapas obtidos de um benchmark. Os caminhos foram determinados pelo algoritmo A* e foram comparadas as soluções no mapa original e no mapa otimizado pela heurística. Foi alcançada uma redução de espaço de busca média de 97,42%, com desvio padrão de 0,026 e a busca teve uma redução marginal média no tempo de execução de 46,76%.

Palavras-chave: otimização de mapas, algoritmos de busca, caminho mínimo, navegação robótica .

Abstract

In the field of electronic gaming and more recently in robotics, autonomous agents often need to repeatedly solve the problem of searching for the smallest path. This need can eventually consume a lot of resources and demands optimizations to make these searches more efficient. Such optimizations may include improvements in search algorithms, map representation, data structures used. This work presents an optimization for search algorithms based on the reduction of the search space by means of an automatic Navigation Meshes generation algorithm which are networks of walkable map areas implying in a reduction of the search space and consequently improving the search processing time. The generation of Navigation Meshes is a problem with no consolidated solution. To prove the heuristic, pathfinding problems were solved on 156 benchmark maps. The pathfindings were performmed by the A* algorithm and the solutions were compared between the original maps and the optimized ones. An average search space reduction of 97.42% was achieved, with a standard deviation of 0.026 and the search had an average marginal reduction in execution time of 46.76%

Keywords: map optimization, search algorithms, minimum path, robotic navigation.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

DAO	Dragon Age: Origins
IA	Inteligência Artificial

Sumário

1 Introdução

1.1 Apresentação

Dentro da teoria dos grafos, o problema de busca do caminho mínimo é o problema de se encontrar, dado um par de vértices seja em grafo com função de peso, seja em grafo com pesos iguais, uma sequência de vértices adjacentes entre si em que se minimize o somatório dos pesos das arestas definidas por esta sequência. Este problema está nas mais diversas aplicações, como jogos eletrônicos, simulação, roteamento de tráfego, robótica e exploração de ambientes desconhecidos (CUI; SHI, 2011).

Diversas soluções foram propostas para o problema do caminho mínimo, como os algoritmos de busca em largura, busca em profundidade, algoritmo de Dijkstra, algoritmo de Bellman-Ford (CORMEN et al., 2009). Tais algoritmos não utilizam heurística para determinar a busca e são conhecidos como algoritmos de busca não informada. Algoritmos que utilizam alguma heurística para melhorar a eficiência da busca, são conhecidos como algoritmos de busca informada. O algoritmo A* é o mais utilizado desta categoria. Ele utiliza uma função heurística para estimar o custo (distância) dos vértices remanescentes na busca até o vértice alvo (EDELKAMP; SCHROEDL; KOENIG, 2010)

Dada a crescente demanda de requisições de caminho mínimo, em especial em sistemas multiagentes, como simuladores, jogos eletrônicos e na robótica, surgiu a necessidade de se otimizar, tanto quanto possível, os algoritmos de busca. Tais otimizações podem incluir melhorias nas heurísticas de avaliação de distância, tomando-se proveito das informações específicas da aplicação sobre a qual a implementação está sendo feita, otimizações nas representações dos mapas, novas estruturas de dados e redução de uso de memória (CUI; SHI, 2011).

As otimizações na representação de mapas diminuem drasticamente o espaço de busca, e, por não incluir pontos inacessíveis, fazem com que seja desnecessária a detecção de colisão entre agentes e objetos estáticos na execução dos algoritmos de caminho mínimo (por exemplo, árvores, lava, água, ou outros objetos num ambiente de jogos), o que implica em redução de custo de processamento e uso de memória. As contrapartidas deste tipo de otimização são o custo da execução do pré-processamento do mapa, que pode ser desempenhada apenas uma vez e o armazenamento desta representação simplificada (HALE, 2011). Na área de jogos digitais, as áreas caminháveis otimizadas são conhecidas como *Navmesh* e na robótica como

meadow map.

O presente trabalho apresenta um algoritmo para gerar uma estrutura das áreas caminháveis otimizadas em um mapa bidimensional, criando uma representação sem os obstáculos, com malhas de pontos que delimitam as áreas caminháveis. Assim, espera-se diminuir o espaço de busca e melhorar a performance de processamento e uso de memória na execução dos algoritmos de caminho mínimo.

1.2 Relevância

Otimizações sobre mapas são amplamente usadas, porém ainda não há um método padrão para construção ou comparação de tais otimizações (HALE, 2011). Tais técnicas demandam muito trabalho e são propensas a erros quando feitas manualmente, pois tanto pontos inacessíveis podem ser incluídos como pontos acessíveis podem ser deixados de fora da representação (OLIVA; PELECHANO, 2011), (HALE, 2011) e (PELECHANO; FUENTES, 2016). A geração automática desta rede dispensa o trabalho manual e elimina os problemas de inserção acidental de erros. A geração automática é usada com o compromisso de se deixar de inserir áreas caminháveis nas adjacências de geometrias mais complexas (mapas com polígonos convexos ou com ângulos pequenos) de partes dos mapas, o que diminui a cobertura de áreas que deveriam ser consideradas como caminháveis.

1.3 Objetivos

1.3.1 Objetivo Geral

O presente trabalho tem por objetivo apresentar um algoritmo para gerar uma estrutura que representa as áreas caminháveis otimizadas em mapas bidimensionais discretos, de modo a se garantir que a representação gerada não contenha nem pontos inacessíveis nem exclua pontos acessíveis do mapa. A heurística proposta será validada através da resolução de um conjunto de problemas de mapas oriundos de um jogo comercial de sucesso (Dragon Age), disponíveis no benchmark (STURTEVANT, 2012).

1.3.2 Objetivos Específicos

Para atingir os objetivos apresentados no tópico anterior, é necessário se atingir os seguintes objetivos específicos:

- Desenvolver um algoritmo para otimizar áreas caminháveis em mapas bidimensionais ladrilhados homogêneos, ou seja, mapas com quadrados de mesmo ta-

manho (tiled maps).

- Simular o algoritmo A* no ambiente proposto, com e sem as otimizações geradas no mapa.
- Analisar e comparar o tempo de processamento em ambos os casos.

1.4 Organização do trabalho

Este trabalho está organizado em 5 capítulos, incluindo a introdução. A seguir temos uma breve descrição de cada um dos capítulos que se seguem:

- Capítulo 2: Fundamentação teórica. Apresenta todos os conceitos fundamentais que serviram de orientação para o desenvolvimento do trabalho. Serão abordados os conceitos de algoritmos para o planejamento de rotas e navigation mesh.
- Capítulo 3: Metodologia. Apresenta como foi construído o mapeamento, o desenvolvimento do algoritmo para simplificar o mapa e a aplicação do algoritmo A* para a geração dos caminhos.
- Capítulo 4: Estudo de caso. Apresenta o desempenho do algoritmo desenvolvido sendo executado sobre os mapas do jogo Dragon Age: Origins, a sua comparação com o mapa não-otimizado em termos de processamentos e uso de memória.
- Capítulo 5: Conclusões. Apresenta os objetivos alcançados e os trabalhos futuros.
- Referencial bibliográfico.

2 Fundamentação teórica

Neste capítulo apresentaremos conceitos fundamentais para o entendimento do trabalho como um todo, com fundamentação em trabalhos realizados na área. Foram discutidos aspectos gerais sobre navegação róbotica móvel, incluindo métodos para o mapeamento das áreas navegáveis e algoritmos para o planejamento de rotas de caminho mínimo. Em seguida, foram apresentadas estratégias para otimização do mapeamento das áreas navegáveis e alguns trabalhos que tem sido desenvolvidos nesta área.

2.1 Navegação Róbotica Móvel

No que tange à robótica móvel, navegar consiste em guiar um robô em um dado espaço físico, de um ponto de origem para um ponto de destino. Esta tarefa tem os seguintes subproblemas subjacentes (SOUZA, 2008):

- mapeamento do ambiente (áreas livres e obstáculos), seja advindo de um mapeamento anterior à navegação ou de mapeamento através de sensores;
- localização do robô no espaço;
- o planejamento da rota;
- execução da locomoção em si.

Segundo (SOUZA, 2008), o planejamento normalmente é executado em três etapas: a primeira é a definição do espaço de configuração, ou seja, o espaço de possibilidades de posição, orientação e ângulos de articulação do robô. Em um robô com pernas, por exemplo, em que posição as pernas estão. Para um robô que seja um carro com partes móveis sendo apenas duas rodas, não existem componentes de ângulo de articulação do robô. A segunda etapa consiste em se determinar o conjunto de estados válidos para busca, ou seja, quais são os espaços e orientações possíveis que o robô pode ocupar. A terceira etapa consiste da busca em si, onde se deve determinar a sequência de ações de alto nível que o robô deve executar até atingir a posição desejada.

2.2 Mapeamento do ambiente

O espaço de busca pode ser modelado de diversas formas (RUSSELL; NORVIG, 2009). Comumente, são utilizados esqueletização e decomposição celular (SOUZA, 2008). Esqueletização é a criação de conjuntos de segmentos de retas para delimitar o espaço navegável de um ambiente, buscando ser uma representação minimal (RUSSELL; NORVIG, 2009). Decomposição celular consiste em se dividir o espaço em regiões chamadas de células. Neste tipo de representação, as células são mapeadas como vértices de um grafo e suas adjacências são modeladas como arestas (RUSSELL; NORVIG, 2009). As Seções 2.2.1 e 2.2.2 abordam duas formas de esqueletização e a Seção 2.2.3 trata de decomposição celular aproximada.

2.2.1 Diagramas de Voronoi

Diagramas de Voronoi são uma forma de decomposição para espaços bidimensionais. Dado um conjunto de pontos, chamados de geradores, cada um desses pontos determinará uma célula, chamada de célula de Voronoi. Para cada ponto não gerador do mapa, é determinado qual gerador está mais próximo deste ponto. Todos os pontos que tem um dado gerador como gerador mais próximo compõem uma célula de Voronoi. Estas decomposições também são chamadas de Tesselação de Voronoi, Decomposição Voronoi, ou um Mosaico de Dirichlet (WEIN; BERG; HALPERIN, 2007). Pontos equidistantes a três ou mais geradores são chamados de vértices de Voronoi e as ligações entre estes pontos são chamadas de arestas de Voronoi.

A Figura 1 mostra três exemplos de Diagramas de Voronoi, com dois, três e quatro geradores. A região em azul abarca os pontos mais próximos do gerador desta célula. O mesmo vale para as outras cores e células.

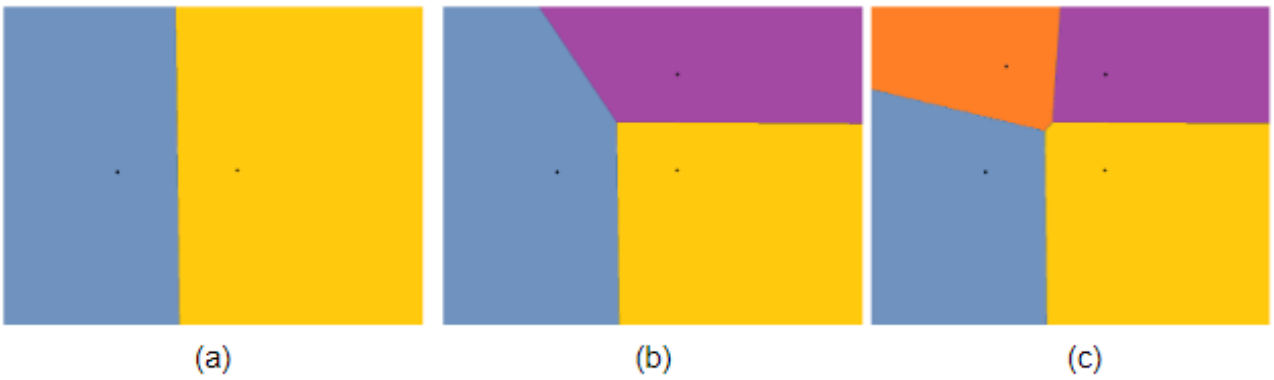

Figura 1 – Diagramas de Voronoi.

Dentre suas inúmeras aplicações, Diagramas de Voronoi são usados para determinar áreas mais seguras de caminhamento de agentes em espaços desconhecidos. Um robô autônomo pode, a cada intervalo de tempo, sondar objetos ao seu redor. A

cada obstáculo encontrado, um gerador é adicionado à representação do mundo usado pelo robô. Os limites de cada célula de Voronoi, também chamados de bissetores, por serem regiões equidistantes dos pontos geradores (e por conseguinte, dos obstáculos encontrados), são trechos propensos a serem mais seguros para os agentes, por diminuírem riscos de colisão(CHOSET; BURDICK, 2000).

Uma forma de se criar Navigation Meshes através do uso de Diagramas de Voronoi é a partir da criação de Mapa de Corredor Explícito, que são caminhos (sequências de pontos) previamente estabelecidos em que fica garantido que não haverá colisão caso o agente se desloque somente entre esses pontos. Formalmente, um Mapa de Corredor Explícito,é um diagrama de Voronoi G = (V, E) onde V e E são seus vértices e arestas de Voronoi (Geraerts, 2010). Além disso, cada aresta armazena o seu custo, para que a busca de caminho possa ser realizada.

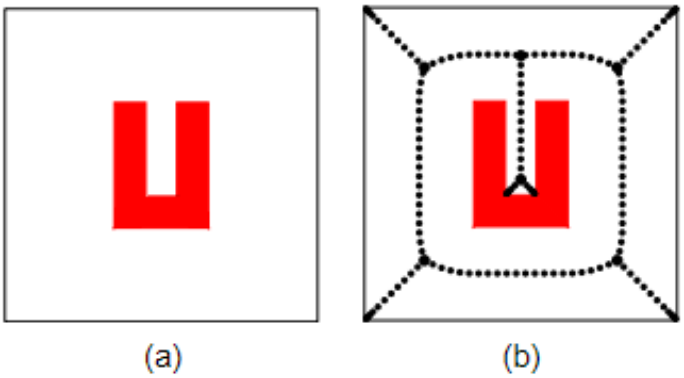

Figura 2 – Mapa de corredor explícito.
(Geraerts, 2010)

2.2.2 Triangulação de Delaunay

A triangulação de Delaunay é um método de divisão de topografias em triângulos de modo a maximizar o ângulo mínimo dos triângulos. A maximização de ângulos mínimos é uma característica desejada porque células com ângulos pequenos deixa menos naturais os movimentos de agentes e faz com que os agentes possam estar em duas células simultaneamente, como mostrado na Figura 3, o que pode dificultar a execução da navegação de robôs.

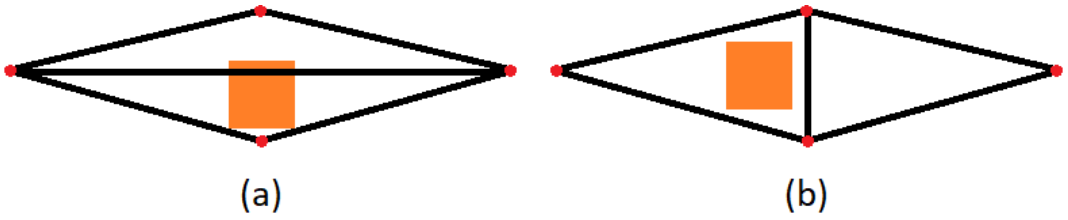

Figura 3 – a) Triangulação sem maximização b) Triangulação com maximização

Uma forma de se gerar uma triangulação de Delaunay é através de flipagem. Um par de triângulos que compartilham um lado determinam um quadrilátero. Neste quadrilátero, o lado que o divide em dois triângulos pode ser flipado (alternado) ou não. O que definirá se haverá a flipagem será a maior ordem lexicográfica de ângulos das duas possibilidades de quadrilátero. Por exemplo, para o quadriláteros da Figura 4 e sua versão flipada, na Figura 5, a ordem lexicográfica da Figura 4 é O^1 = 30, 30, 30, 40, 110, 120 e a ordem lexicográfica da Figura 5 é O^2 = 30, 40,60, 70, 80, 80. No primeiro elemento da ordem, eles são iguais. No segundo elemento, o quadrilátero da Figura 5 tem um ângulo maior. Neste caso, o quadrilátero da Figura 4 deve ser preterido em favor do quadrilátero de Figura 5.

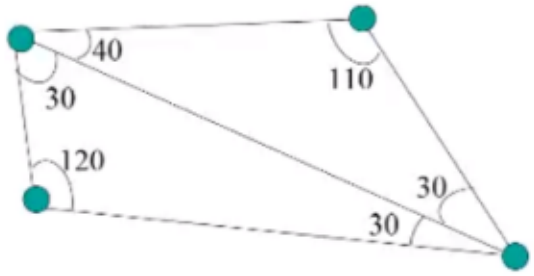

Figura 4 – Exemplo de quadrilátero.
(RODRIGUES, 2017)

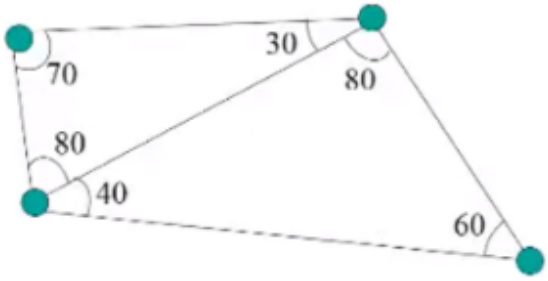

Figura 5 – Contraparte.
(RODRIGUES, 2017)

A Figura 6 apresenta o método de flipagem para uma estrutura um pouco mais complexa. As arestas destacadas são as que foram alternadas em cada passo.

2.2.3 Decomposição celular aproximada

Uma forma de se modelar um mapa é pela decomposição celular aproximada. Nesta abordagem, as regiões do espaço são divididas em um número finito de elementos simples, chamados de células. Para cada célula, caso parte dela contenha regiões não navegáveis, a célula inteira é considerada como obstáculo. Caso não tenha, será uma célula livre. Após as células terem sido determinadas, as células livres que correspondem a áreas adjacentes serão ligadas entre si, criando um grafo de conectividade.

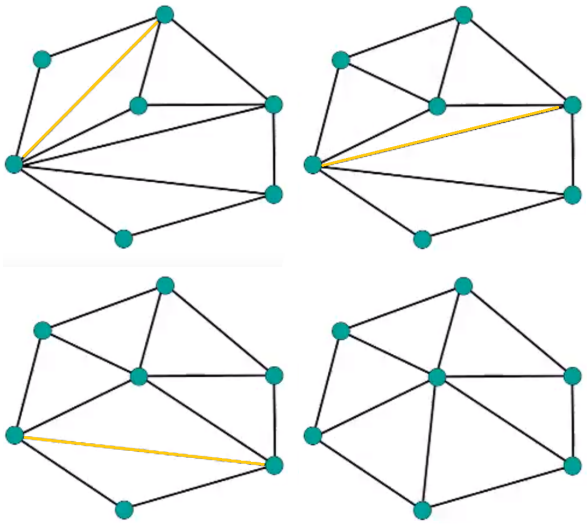

Figura 6 – Flipagem de quadriláteros.
(RODRIGUES, 2017)

As buscas de caminho serão então feitas neste grafo. A Figura 7 ilustra uma decomposição celular aproximada. As regiões em azul correspondem a células em que houve perda de área navegável.

Geralmente, este tipo de representação apresenta perdas de área navegável, mas este problema pode ser atenuado se ajustando, tanto quanto necessário, a granularidade das células, o que implica no compromisso entre maior cobertura contra maior gasto de memória e de processamento.

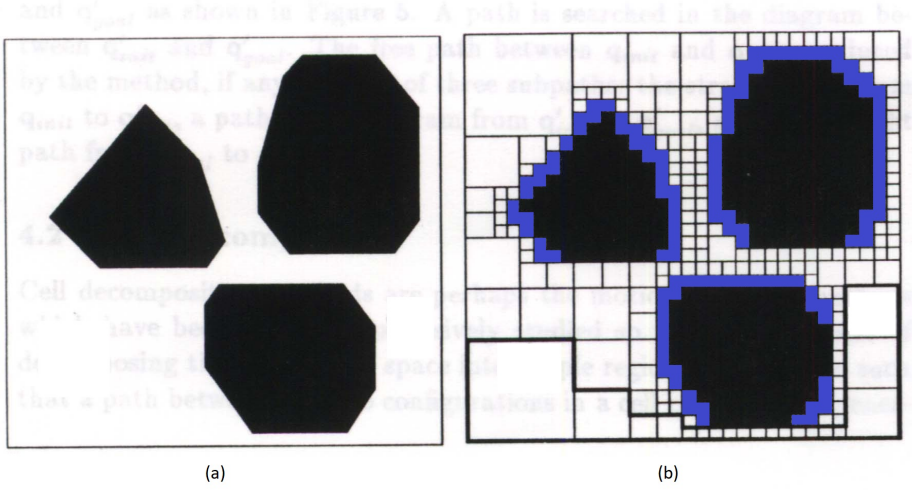

Figura 7 – a) Mapa com polígonos b) Decomposição celular aproximada.
(SOUZA, 2008)

2.3 Algoritmos para o planejamento de rotas

Conceitualmente, um algoritmo é uma sequência de passos finitos que dão uma solução correta para um dado problema. Uma busca em largura resolve o problema de se encontrar um caminho entre dois pontos. Para o problema de se encontrar não um caminho, mas o menor caminho entre dois pontos, a busca em largura já não garante encontrar o menor caminho. Neste caso, o algoritmo de Dijkstra é uma solução algorítmica para o problema.

Infelizmente, muitas vezes uma solução algorítmica pode não ser suficientemente eficiente para determinados problemas. Nestes casos, caso exista, pode-se lançar mão de uma solução heurística, ou seja, uma sequência de passos finitos que dão uma solução próxima para o dado problema. Evidentemente, a definição do que seria "próximo" se dará para a aplicação real da solução e do problema em questão. A seguir, discutiremos o algoritmo de Dijkstra e o algoritmo A*.

2.3.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra resolve o problema de encontrar os caminhos mínimos a partir de um nó de um grafo direcionado com pesos positivos (CORMEN et al., 2009). Neste algoritmo, cada nó mantém uma referência para um outro nó, seu antecessor (ou pai), e o valor do custo para se chegar até este nó. Ou seja, o custo de um nó é o somatório da sequência de valores das arestas do nó inicial até o nó em questão. Durante a inicialização, o antecessor tem seu valor atribuído como nulo e ao seu custo é atribuído infinito, a menos do nó inicial, que tem pai nulo e custo zero.

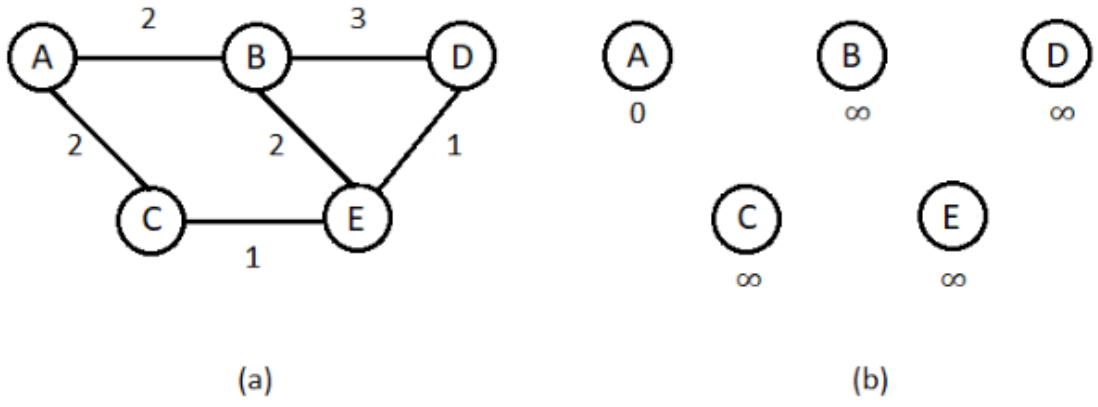

Figura 8 – Exemplo de inicialização do algoritmo de Dijkstra.

Também na inicialização, uma fila de vértices pendentes é criada, com o nó inicial dentro desta fila. Esta fila é ordenada pelo custo do vértice.

A cada iteração do algoritmo, os vizinhos do primeiro nó pendente da fila serão processados.

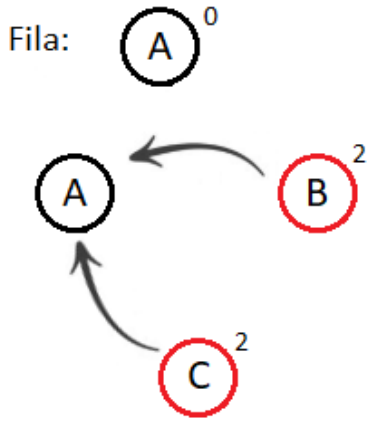

Figura 9 – Exemplo de primeira iteração do algoritmo de Dijkstra.

Cada nó adjacente a este primeiro nó tem seu possível novo custo calculado como sendo o custo do nó atual somado com o custo de sua ligação com o pendente. Caso o possível novo custo seja menor que o custo atual, o custo é atualizado para esse possível novo custo e o pai é atualizado para o primeiro nó da fila. No exemplo, da Figura 9, o custo do nó B passa de infinito para 2 (zero no nó A somado com 2, da aresta AB) e o antecessor de B passa a ser A. A cada iteração, caso o custo atual do nó adjacente seja infinito, além de ter seu pai e custo atualizado, o nó adjacente é adicionado à lista de vértices pendentes. Caso contrário, temos um nó revisitado. Para um nó revisitado, caso o novo valor calculado de custo seja maior que o valor atual do nó adjacente, nada precisa ser feito. Sendo menor, o pai deve ser atualizado para o primeiro nó da fila e o custo deve ser atualizado para o valor calculado nesta iteração. Para o caso de se procurar o caminho entre dois nós, o algoritmo pode ser interrompido quando o nó de destino for alcançado.

A Figura 10 mostra com mais detalhes a execução do algoritmo de Dijkstra. Na Figura 10.1, o nó “A” é adicionado à fila de execução , com custo 0. Este nó altera o pai dos nós “B” e “C” e seus custos. Para os dois nós, o custo será 2.

Passo seguinte, a Figura 10.2 mostra fila com os nós “B” e “C”. Como os seus custos são iguais, qualquer dos nós pode ser tratado primeiro. Neste passo há a adição dos nós “D” e do nó “E”. Ambos terão “B” como pai e seus custos serão 5 e 4, respectivamente.

A Figura 10.3 mostra a fila com os nós “C”, “E”e “D”. Neste passo, 2, o custo do nó “C” é somado com 1, o custo da aresta“C-E”, totalizando 3. Atualmente o custo de “E” é 4. Como o custo para se chegar a “E” passando por “C” é menor do que pelo seu pai atual, o pai de “E” é alterado de “B” para “C” e seu custo é atualizado de 4 para 3.

Finalmente, a Figura 10.4 mostra os nós “E” e “D” na lista. Este passo mostra a atualização do nó “D”. Após a remoção do nó “E”, o nó “D” também será processado, sem alterar qualquer valor.

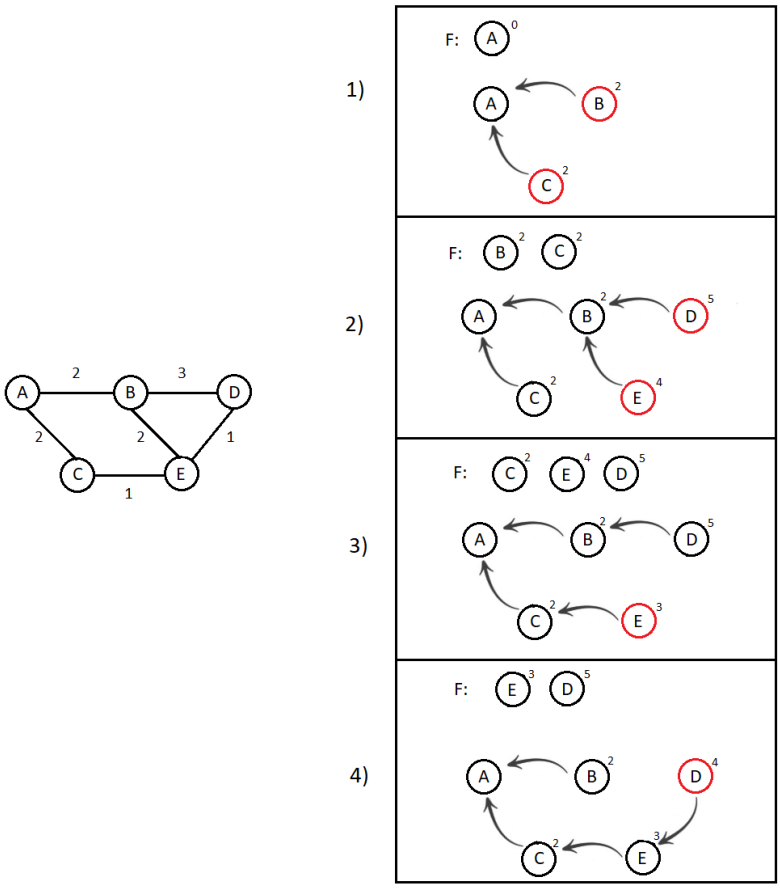

Figura 10 – Execução do algoritmo de Dijkstra.

2.3.2 Algoritmo A*

Para resolver o problema de se encontrar o menor caminho entre um ponto inicial e um ponto final, o algoritmo de Dijkstra visita muitos nós que não são necessários. Buscando otimizar a expansão dos nós, foi desenvolvido o algoritmo A*, que não expande todos os vizinhos e para quando encontra o nó destino (HART; NILSSON; RAPHAEL, 1968).

Para realizar a poda na expansão, a lista de nós pendentes é ordenada não apenas pelo custo de caminhamento, mas pela soma do custo de caminhamento com uma estimativa do custo do nó até o destino, através de uma função heurística que estima qual seria o custo até se chegar ao nó destino. Ou seja, nós que parecem estar mais próximos do nó de destino são tratados com precedência, através da função de ordenação:

$$f(u) = g(u) + h(u) \tag{2.1}$$

Ao que $g(u)$ corresponde ao custo para se chegar do nó inicial até o nó em questão e $h(u)$ corresponde ao valor estimado através da função heurística (EDELKAMP; SCHROEDL; KOENIG, 2010).

2.3.2.1 Pseudo-código - Algoritmo A-Estrela

Este algoritmo leva em consideração um nó inicial (noInicial), um nó destino (noDestino), o grafo G(V,E) que os contêm, a função de ordenação $f(u)$ e o mapeamento pai(u), que corresponde ao nó pai de um dado nó.

1. criar lista de nós que conterá nós visitados, chamada “fechados’
2. criar lista de nós pendentes, chamada de “abertos”
3. para cada nó n de V, $f(n)$ = infinito
4. para cada nó n de V, pai(n) = nulo
5. f(noInicial) = 0
6. abertos.adicionar(noInicial)
7. enquanto (!abertos.vazio()):
    - 7.1. noAtual = o nó de abertos com menor $f(u)$
    - 7.2. fechados.adicionar(noAtual)
    - 7.3. SE (noAtual == noDestino)
        - 7.3.1. retornar o noAtual
    - 7.4. para cada vizinho N de noAtual:
        - 7.4.1. SE(“fechados” não contém N)
            - 7.4.1.1. fCandidato= g(noAtual) + h(N) + custoVertice(noAtual, N)
            - 7.4.1.2. SE(fCandidato < $f(n)$)
                - 7.4.1.2.1. $f(n)$ = fCandidato
                - 7.4.1.2.2. pai(N) = noAtual
            - 7.4.1.3. SE (!abertos.contem(N))
                - 7.4.1.3.1. abertos.adicionar(N))

As Figura 11 e 12 mostram comparações entre a execução do algoritmo de Dijkstra e o A*. A célula em verde escuro representa o ponto inicial e o vermelho representa o destino. Em verde claro estão os nós que estão na fila para serem processados e em azul estão os nós visitados. A Figura 12 mostra mais claramente que a expansão do A* vai sempre na direção da origem para o destino enquanto que a expansão do

Dijkstra, justamente por não ser informada, se espalha indiscriminadamente por todos os lados.

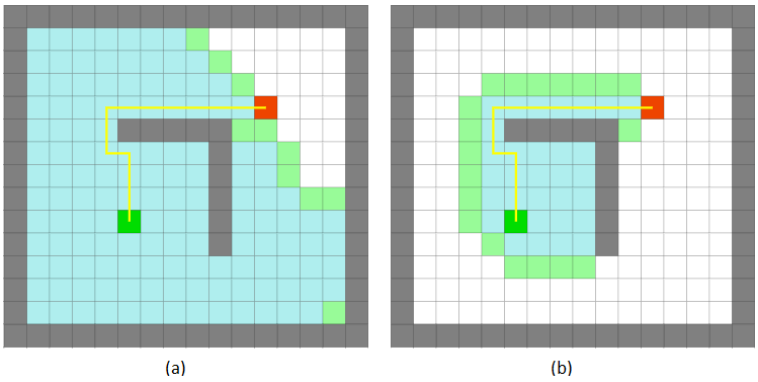

Figura 11 – Comparação entre Dijkstra (Figura a) e A* (Figura b).
(PATHFINDING..., 2017)

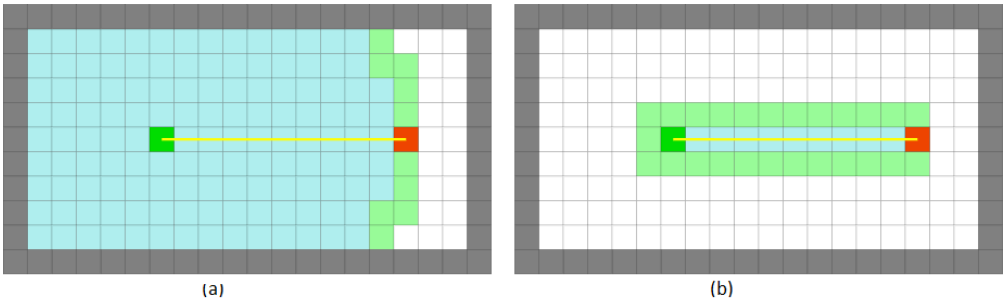

Figura 12 – Comparação mais simplificada entre Dijkstra (Figura a) e A* (Figura b).
(PATHFINDING..., 2017)

2.3.2.2 Métricas para o cálculo de distância

Para a execução do algoritmo A*, é ncessário uma função heurística para estimativa do custo de caminhamento do nó atual até o nó destino. Diferentes formas de cálculos de estimativa podem ser usadas pelo algoritmo. Algumas dessas formas são discutidas a seguir:

- **Distância Euclidiana:** Dados dois pontos, A e B, a distância euclidiana é a distância em linha reta entre estes dois pontos. Esta distância é calculada pela fórmula:

$$d(A,B) = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2} \tag{2.2}$$

  Caso o agente não possa fazer movimentos em ângulos arbitrários, esta distância perde sentido. Outro aspecto que pesa contra o uso desta métrica é o custo de se executar duas multiplicações e uma raiz quadrada.

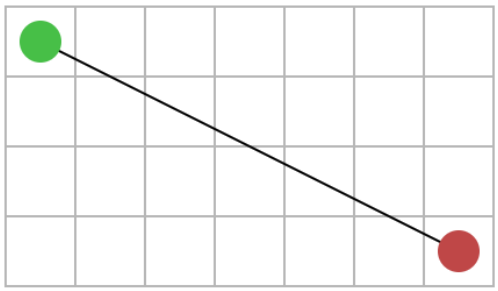

Figura 13 – Distância euclidiana.
(SOUZA, 2008)

- **Distância Manhattan:** Usualmente usamos a distância euclidiana para o cálculo da distância entre dois pontos. No espaço discreto que só permite movimentações horizontais ou verticais, esta distância tem pouco sentido. Neste caso, pode ser usada a distância Manhattan:

$$d(A, B) = |x_A - x_B| + |y_A - y_B| \tag{2.3}$$

  Podemos entender com mais facilidade esta métrica como sendo o caminho feito de carro entre dois pontos de uma cidade em que as quadras são todas quadrados de tamanhos iguais e igualmente espaçados, como na Figura 14. Na distância Manhattan, o valor da distância independe do caminho escolhido. Neste caso, em que é necessário se locomover nas direções horizontal ou na vertical, a distância euclidiana tem pouca utilidade prática.

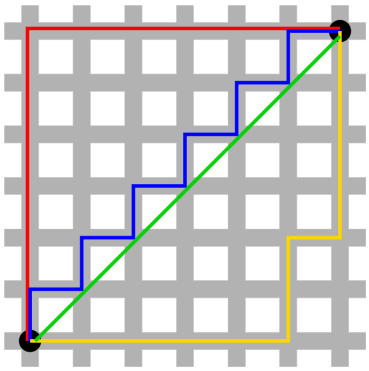

Figura 14 – Comparação distância euclidiana vs distância Manhattan.
(WIKIPEDIA, 2019)

- **Distância diagonal:** Outra métrica de distância, que combina a distância Manhattan e a distância euclidiana, é a distância diagonal. Esta distância considera que o agente pode fazer movimentos em diagonais de 45 graus, como a diagonal de um quadrado. A Figura 15 exemplifica a ideia da distância diagonal. Seu valor é

dado pela fórmula:

$$d(A,B) = ||X_A - X_B| - |Y_A - Y_B|| + \sqrt{2} * min(|X_A - X_B|, |Y_A - Y_B|) \quad (2.4)$$

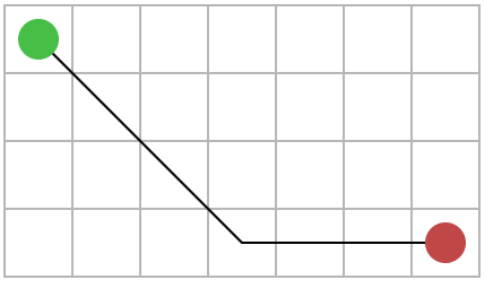

Figura 15 – Distância diagonal.
(SOUZA, 2008)

Como o valor de $\sqrt{2}$ da Equação 2.4 pode ser calculado apenas uma vez e armazenado, do ponto de vista da eficiência, esta métrica tem a vantagem de prescindir do cálculo de raíz quadrada.

Como o presente trabalho trata de mapas bidimensionais ladrilhados homogêneos, para o planejamento da rota foi utilizado o algoritmo A*, com a heurística da distância diagonal para estimativa de custo.

2.4 Otimização na representação dos mapas

Navigation Meshes (NavMeshes) são um recurso utilizado para decompor as áreas caminháveis dentro de um ambiente virtual ou da representação de um ambiente real, resultando em uma representação diminuta da área caminhável do mapa em questão, assim, os algoritmos de caminho mínimo podem ser usados com esta representação de forma otimizada, melhorando sua performance, já que há muito menos pontos para se considerar em seu processamento e não há necessidade de se processar algoritmos de colisão entre os agentes e as áreas inacessíveis do ambiente (SNOOK, 2000). Esse termo tem sido usado pela comunidade de IA em jogos e ganhou popularidade nos anos 2000.

Muitas vezes, essas NavMeshes são criadas manualmente, o que é extremamente trabalhoso e propenso a erros, já que pode-se incluir acidentalmente como área caminhável regiões que são inacessíveis. Também é possível se incidir no problema inverso, que é deixar de se incluir áreas acessíveis na NavMesh (OLIVA; PELECHANO, 2011), (HALE, 2011) e (PELECHANO; FUENTES, 2016). Apesar de ser uma técnica bastante usada na indústria de jogos, não há uma forma canônica de se avaliar ou comparar tais malhas. Cada implementação têm compromissos diferentes, foi testada em ambientes diferentes e com diferentes aplicações em mente (TOLL et al., 2016).

Para exemplificar a proposta, a Figura 16 mostra um mapa do jogo World of WarCraft, com uma possível NavMesh associada.

(a) (b)

Figura 16 – Mapa original e uma possível Nav Mesh.
(CUI; SHI, 2011)

2.5 Trabalhos Relacionados

O seminal artigo “Simplified 3D Movement and Pathfinding Using Navigation Meshes”(SNOOK, 2000), em que as Navigation Meshes são inicialmente propostas, trata da representação de áreas caminháveis em ambientes virtuais tridimensionais, focando inicialmente nas aplicações de *pathfinding* para área de jogos eletrônicos tridimensionais de tempo real, mas que pode ser aplicado diretamente em jogos bidimensionais. Nesta proposição, é considerado o compromisso de se fazer uma representação bidimensional para o *pathfinding*, o que livra os programadores de se fazer algoritmos complexos de colisão tridimensional e os computadores dos custosos cálculos envolvidos nestes algoritmos. Eventualmente, partes dos agentes podem “entrar” nos objetos do jogo, como quando a mão de um personagem próximo de uma parede passa parcialmente através da parede, mas sem propriamente o personagem como um todo atravessar a parede. Outro compromisso que pode ser feito é de se restringir a área caminhável, o que diminui a área coberta, mas impede estas pequenas falhas.

Esta representação bidimensional, em que se coloca intuitivamente o que seria como um carpete poligonal que modela as regiões em que os personagens podem firmar os pés e, por consequência, caminhar, viabiliza o uso de algoritmos bidimensionais, que são bem mais simples de se implementar e de se manter e muito mais eficientes computacionalmente para executar, além de evitar a verificação de colisão com os objetos estáticos do ambiente, o que impacta de forma dramaticamente positiva na performance dos jogos tridimensionais como um todo, já que é requisitado que esta tarefa seja executada em grande volume.

Em (SNOOK, 2000), a área caminhável do mapa é modelada como um conjunto de triângulos, o que possibilita o uso das várias técnicas matemáticas da geometria.

A Figura 17 apresenta um exemplo de terreno tridimensional com a sua MavMesh associada.

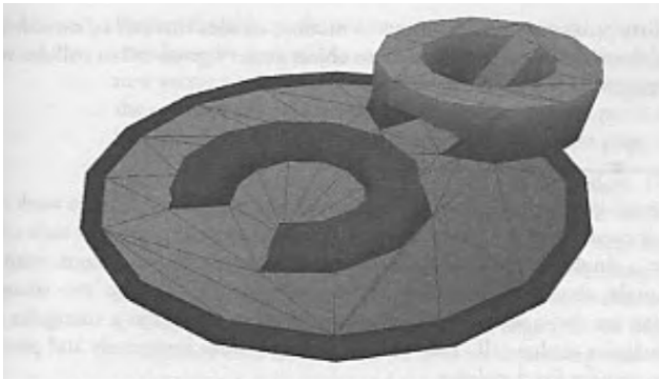

Figura 17 – Mapa 3D com Nav Mesh.
(SNOOK, 2000)

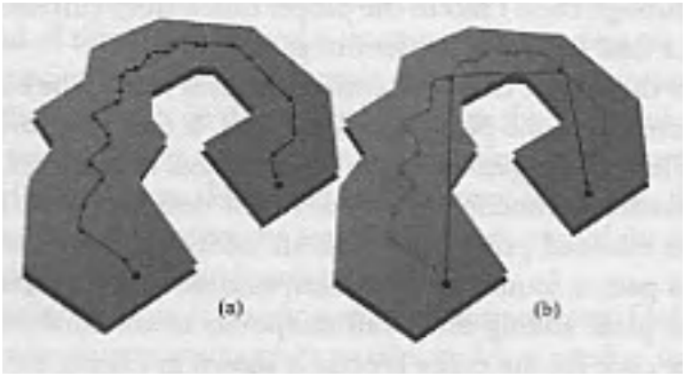

Figura 18 – (a) Caminho gerado por uma NavMesh e (b) Caminho suavizado.
(SNOOK, 2000)

Um problema das NavMeshes abordado no artigo é a qualidade dos caminhos gerados, em que o caminho gerado tanto não é o menor possível quando se considera o mapa inicial, como não parece ser o caminho natural de um agente real, como exemplificado na Figura 17. Ambos os problemas podem ser suavizados posteriormente por um algoritmo de linha de visada (Figura 18). A conclusão dos autores no artigo é a de que, apesar do caminho gerado não ser ótimo, a melhoria de performance dada pela NavMesh compensa a perda, ainda mais se for feita a melhoria de linha de visada.

O artigo "Automatic Generation of Suboptimal NavMeshes" (OLIVA; PELECHANO, 2011) apresenta um algoritmo de geração automática de NavMeshes baseado em divisão da geometria inicial do mapa em polígonos, até que polígonos convexos que cubram toda a área do mapa sejam gerados. Num segundo momento, seja para diminuir a quantidade de nós e aumentar ainda mais o nível de otimização, seja para evitar células com ângulos próximos de zero, o que deixa os movimentos menos naturais e faz com que os agentes possam estar em duas células simultaneamente, os polígonos

podem ser aglutinados. Caso este polígono gerado não seja convexo, ele é redividido, preferencialmente não desfazendo a aglutinação, mas de modo que evite que uma das células desta divisão contenha ângulos próximos a zero. Algumas divisões são feitas usando força bruta até que se encontre uma divisão sem células convexas. Caso não seja encontrada esta divisão, a aglutinação é desfeita.

Ainda segundo este artigo, se aponta que a geração manual de NavMeshes, além dos problemas de inclusão de áreas caminháveis como não caminháveis, pode gerar, através dos algoritmos de busca, caminhos que às vezes não parecem naturais. Também é apontado que os *game engines* que provêm alguma funcionalidade de geração automática de NavMeshes são restritos a certos tipos de mapas ou criam células "mal condicionadas" (com ângulos próximo de zero). São citados os exemplos da *engine* Valve, que cria quadriláteros e não permite uso de todo tipo de geometria, da Unreal Engine, que tende a gerar um grande número de células mal condicionadas caso o mapa apresente uma geometria complexa, que geram caminhos que não parecem naturais, e da *engine* Recast, que gera muitas células desnecessárias, que poderiam ser aglutinadas, o que impacta negativamente na performance dos algoritmos de busca.

A dissertação de doutoramento "A Growth-Based Approach To The Automatic Generation of Navigation Meshes" (HALE, 2011) apresenta uma técnica de crescimento de estruturas chamadas de "regiões de unidade". As regiões de unidade são expandidas radialmente até encontrarem obstáculos. Enquanto não cobrirem o mapa, novas regiões de unidade são inseridas, gerando no final uma NavMesh de alta qualidade (com polígonos de alta ordem, com poucas regiões degeneradas [regiões vazias ou com ângulos próximos a zero] e com alta cobertura final), de acordo com as simulações executadas durante estudos realizadas pelo autor.

Na dissertação também discute-se a questão das relações entre maior granularidade de células gerar maior cobertura do mapa (principalmente em mapas com estruturas geométricas muito complexas) e melhor qualidade do caminho gerado pelos algoritmos de busca (o caminho parecer ser mais natural), com as desvantagens de maior custo de processamento e gasto de memória. Por conta do problema da qualidade do caminho poder ser suavizado com técnicas de linha de visada, o autor conclui que é mais vantajoso se buscar uma menor granularidade e talvez se fazer uma intervenção específica nas áreas não cobertas pela NavMesh gerada automaticamente. A depender do ambiente em questão, ambientes com estruturas mais simples tendem a ter altas coberturas dadas pelos algoritmos de geração automática de NavMeshes.

A dissertação "Planejamento de Trajetória Para Um Robô Móvel Com Duas Rodas Utilizando Um Algoritmo A-estrela Modificado" (SOUZA, 2008) descreve a implementação de um sistema em que um servidor planeja uma rota e um robô a exe-

cuta, remotamente. Este trabalho modela o ambiente usando decomposição celular aproximada e lança mão do algoritmo A-estrela para fazer o planejamento. Sensores embarcados no robô provêem informações sobre a localização do robô, para que o planejamento seja feito considerando a posição real do robô.

Após o planejamento ser realizado, o caminho encontrado é então suavizado removendo nós. O nó em que o robô se encontra não pode ser removido. Para cada outro nó, caso o nó seguinte possa ser atingido diretamente sem se atingir um obstáculo, o nó em questão é removido. Caso contrário, o nó em questão deve ser mantido. Este procedimento é repetido até se atingir o nó de destino. Através de radiofrequência, o caminho que deve ser seguido é passado para o robô, que o executa. Diferente do presente trabalho, é calculado o planejamento da rota utilizando a representação discretizada completa, sem uma otimização sobre o mapa.

Segundo “A Comparison of High-Level Approaches for Speeding Up Pathfinding” (STURTEVANT; GEISBERGER, 2010), a maioria dos jogos eletrônicos desenvolvidos hoje em dia utilizam alguma forma de abstração de alto nível para planejamento de caminho em mapas, como NavMeshes ou waypoints, que são conjuntos de pontos navegáveis pelo agente, semelhantes a um trilho de trem. Este artigo aborda mapas muito grandes de jogos, trazendo o estudo de caso do jogo “Dragon Age: Origins”, em que, para que o *pathfinding* fosse otimizado, o mapa do mundo foi dividido em regiões. Então, um ponto em cada região é escolhido para representar esta região. Caso a origem e destino que se queira encontrar o caminho estejam na mesma região, uma abordagem padrão é executada. Caso os pontos estejam em regiões distintas, a busca é realizada entre o nível de abstração de regiões. Quando o caminho de regiões é encontrado, uma busca padrão é feita entre os pontos de ligação entre as regiões intermediárias encontradas no caminho de alto nível, até que se chegue na região do ponto final, onde a última busca padrão é executada. Para evitar que mais processamento seja realizado, nos casos em que o caminho não seja mais necessário, seja porque o agente parou ou “morreu” ou qualquer outra razão, estas buscas padrão só são executadas no momento em que o agente chega numa das regiões intermediárias. Esta abordagem reduz drasticamente o número de nós que devem ser computados. A Figura 19 mostra um exemplo em um mapa pequeno, 8x8, em que suas células são abstraídas em cinco regiões, com as suas ligações apresentadas.

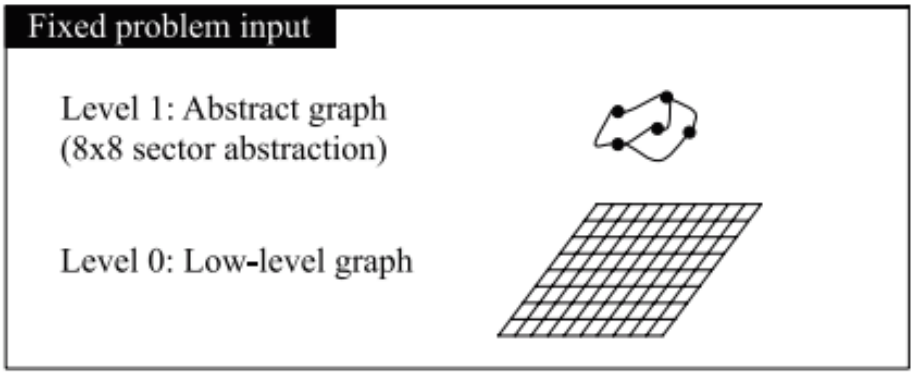

Figura 19 – Mapa dividido em regiões.
(STURTEVANT; GEISBERGER, 2010)

3 Metodologia

Este capítulo aborda como foi desenvolvido o algoritmo para reduzir o espaço de busca em um conjunto de mapas bidimensionais e como o algoritmo proposto foi validado. Após o algoritmo ter sido desenvolvido, foi executado sobre cada um dos 156 mapas do jogo Dragon Age: Origins (BIOWARE, 2009). Foram criadas representações otimizadas para cada mapa original do jogo, e foram utilizados para validação os conjuntos de problemas propostos pelo *Moving IA LAB* (STURTEVANT, 1999). Então, o algoritmo A-estrela foi executado sobre os problemas propostos (pares de pontos iniciais e finais) utilizando o mapa bidimensional do jogo duas vezes, a primeira sobre os mapas originais e então sobre os mapas com a otimização. Os resultados das duas execuções foram então comparados e analisados e estão descritos no Capítulo 4.

3.1 Algoritmo proposto

Como elicitado nos objetivos, o algoritmo trabalha com mapas bidimensionais ladrilhados. Cada célula é considerada com sendo ou área livre ou área inacessível (doravante denominadas simplesmente por “muro”) ou ponto-chave. Inicialmente, todas as células são ou muros ou áreas livres e então o algoritmo encontra as células que têm que ser pontos de navegação, transformando células de áreas livres em pontos de navegação. Os pontos de navegação então são ligados entre si, caso haja um caminho do tipo *city* block entre eles, ou seja, uma sequência de passos em uma direção horizontal (esquerda ou direita) alternados com passos em uma direção vertical (cima ou baixo).

A lista de pontos de navegação então é usada para se descobrir o caminho mínimo entre os pares de pontos propostos, usando o algoritmo A-estrela. Mas antes da execução do algoritmo de caminho mínimo ser feita os pontos inicial e final são inserido, como se fossem pontos de navegação. Então o algoritmo de caminho mínimo pode ser executado.

Para efeito didático, faremos uma descrição intuitiva antes de tecer maiores considerações técnicas sobre o algoritmo. Em um primeiro momento, as áreas livres que ladeiam os muros são consideradas pontos de navegação. Depois, os pontos de navegação que estejam exatamente entre dois pontos de navegação colinearmente deixam de ser pontos de navegação. Finalmente, os pontos de navegação restantes são conectados, caso sejam “visíveis” um para o outro. Os algoritmos de busca que forem operar sobre esta representação do mapa devem então inserir o ponto inicial e final na representação, ou seja, ligar os pontos inicial e final aos pontos de navegação

que estiverem “visíveis” no mapa.

Num exemplo simples, considere o mapa da Figura 20 (a), um mapa de 5 linhas e 10 colunas, onde as áreas inacessíveis são representadas como células pretas. As restantes são áreas acessíveis. Inicialmente, o entorno dos muros (as células em laranja) são considerados pontos de navegação (Figura 20 (b)). Então, os pontos de navegação que estão entre dois pontos de navegação colinearmente são removidos (Figura 20 (c)). Os pontos de navegação então são ligados (Figura 20 (d)). Na figura não é exibido o peso das arestas, mas nessa etapa este peso já é calculado e armazenado. Neste momento, temos o pré-processamento da geração da NavMesh.

Dentro do mesmo exemplo da Figura 20, consideremos que queremos descobrir o caminho mínimo entre os pontos O e D, representados na Figura 20 (e). O algoritmo de busca deve inserir os pontos O e D na malha, conectando-os aos pontos de navegação “visíveis”. A busca é então executada sobre os elementos da Figura 20 (g). Neste exemplo, o espaço de busca foi reduzido de 50 vértices para apenas 6. Há o overhead do pré-processamento, que é feito apenas uma vez para todos os problemas que forem tratados sobre um mapa, e de se encontrar para os pontos de origem e destino quais pontos de navegação lhes são “visíveis”.

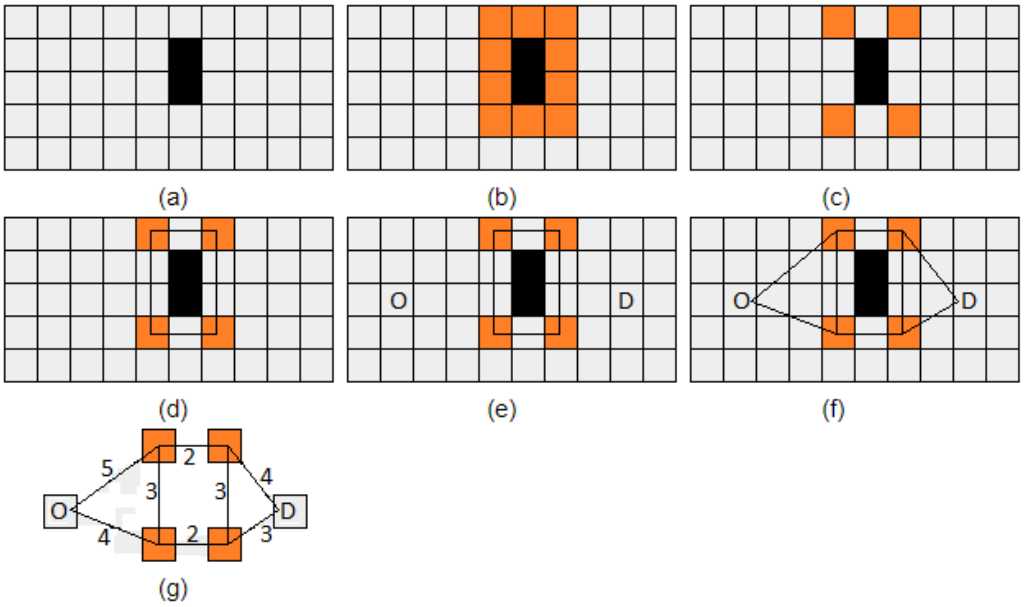

Figura 20 – Geração de Navigation Mesh.

3.1.1 Pseudo-código - algoritmo principal

Consideremos que cada parede tenha os atributos “linha”, “coluna”, correspondentes à linha e à coluna da parede e “tipo”, que pode assumir os valores “PAREDE”, que corresponde a regiões parcialmente ou completamente obstaculizadas, ou “ÁREA_NAVEGÁVEL”, que são as regiões em que a movimentação é livre, ou “PONTO_CHAVE”,

as áreas livres que farão parte do mapa otimizado. As células também possuem os métodos “cima()”, “baixo()”, “esquerda()” e “direita()”, que retornam as células da vizinhança da célula em questão.

1. criar uma lista de células, chamada “pontosChave”
2. para cada célula “p” do tipo “PAREDE” do mapa:

    2.1. para cada célula “c” localizadas nas posições {
    (p.linha-1, p.coluna-1),
    (p.linha-1, p.coluna),
    (p.linha-1, p.coluna+1),
    (p.linha, p.coluna-1),
    (p.linha, p.coluna+1),
    (p.linha+1, p.coluna-1),
    (p.linha+1, p.coluna),
    (p.linha+1, p.coluna+1)
    }:
    2.1.1. SE (c.tipo == “ÁREA_NAVEGÁVEL”)
    2.1.1.1. c.tipo = “PONTO_CHAVE”
    2.1.1.2. pontosChave.adicionar(c)

3. para cada célula “c” em “pontosChave”:

    3.1. SE (
    (c.cima().tipo == “PONTO_CHAVE” AND
    c.baixo().tipo == “PONTO_CHAVE” AND
    c.esqueda().tipo != “PONTO_CHAVE” AND
    c.direita().tipo != “PONTO_CHAVE”)
    OR
    (c.cima().tipo != “PONTO_CHAVE” AND
    c.baixo().tipo != “PONTO_CHAVE” AND
    c.esqueda().tipo == “PONTO_CHAVE” AND
    c.direita().tipo == “PONTO_CHAVE”)
    3.1.1. c.tipo = “ÁREA_NAVEGÁVEL”
    3.1.2. pontosChave.remover(c)

4. para cada célula “c” na lista “pontosChave”, ligar “c” às outras células na lista “pontosChave”

3.1.2 Pseudo código - ligação de pontos de navegação

Para efeito de brevidade, o código a seguir só conterá a implementação de ligações diretamente para cima, diretamente para direita e as ligações que sejam na região superior direita. As demais direções foram implementadas de forma semelhante.

1. para cada célula “c” em “pontosChave”:
    1.1. passosCima = 0
    1.2. cima = c
    1.3. enquanto (cima.tipo != “PAREDE” AND cima.tipo!= “PONTO_CHAVE” AND passosCima != 0)
        1.3.1 SE (passosCima == 0)
            1.3.1.1. direita = cima
            1.3.1.2. passosDireita = 0
        1.3.2 SENÃO
            1.3.2.1. direita = cima.direita()
            1.3.2.2. passosDireita = 1
        1.3.3. enquanto(direita.tipo != “PAREDE” AND direita.tipo!= “PONTO_CHAVE”)
            1.3.3.1. direita = cima.direita()
            1.3.3.2. passosDireita++
        1.3.4. SE (direita.tipo == “PONTO_CHAVE”)
            1.3.4.1. custo = passosCima + passosDireita
            1.3.4.2. c.adicionarVizinho(direita, custo)
            1.3.4.3. direita.adicionarVizinho(c, custo)
        1.3.5. cima = cima.cima()
        1.3.6. passosCima++
        1.3.7. SE(cima.tipo == “PONTO_CHAVE” && passosCima != 0)
            1.3.7.1. custo = passosCima
            1.3.7.2. c.adicionarVizinho(cima, custo)
            1.3.7.3. cima.adicionarVizinho(c, custo)

4 Estudo de caso

Este capítulo aborda como foi feita a validação do algoritmo proposto. Iremos detalhar o ambiente de testes, a aplicação implementada e exporemos os resultados obtidos.

4.1 Ambiente de testes

Como estudo de caso, foram realizadas otimizações sobre o conjunto de mapas do jogo Dragon Age: Origins (BIOWARE, 2009), que foi cedido pela BioWare Corporation para o Moving AI Lab (STURTEVANT, 1999), um laboratório dedicado à pesquisa na área de pathfinding e dirigido pelas University of Alberta e pela University of Denver. Este conjunto compreende 156 mapas do jogo, com tamanhos variando entre 28 x 22 (616 células) e 1104 x 1260 (1.391.040 células).

Para validação, foi utilizado o conjunto de problemas de teste para o mesmo jogo do mesmo banco. Este conjunto de testes compreende, para cada um dos 156 mapas, a pares de início e fim, que variam entre 50 e 7.070 problemas.

Segundo (STURTEVANT, 2012), estes mapas são todos os mapas do jogo, de modo que o estudo de caso se deu sobre todo o conjunto de mapas de um jogo eletrônico comercial de sucesso.

4.1.1 Dragon Age: Origins

Dragon Age: Origins (DAO) é um jogo do tipo RPG desenvolvido pela BioWare Corporations, publicado em 2009. DAO utiliza internamente o formato de grids em seus algoritmos de pathfinding (STURTEVANT, 2012). A BioWare explicitamente cedeu os arquivos dos mapas extraídos, para que pudessem ser utilizados como marcadores de benchmark. Foram extraídos cada um dos 156 mapas que foram utilizados no jogo original e representam um conjunto complexo e variado de mapas de um jogo eletrônico de grande sucesso (STURTEVANT, 2012). Estes arquivos têm na primeira linha o tipo de mapa, sempre ”octile”, indicando que são mapas bidimensionais em que os agentes podem se movimentar na diagonal. Então há a indicação de altura e de largura do mapa e depois a indicação de áreas caminháveis e áreas bloqueadas. Áreas caminháveis são indicadas com o caractere ”.”e áreas bloqueadas como ”T”ou ”@”.

Para cada um dos 156 arquivos dos mapas, o Moving AI Lab criou um arquivo com casos de testes. A primeira linha destes arquivos indica a versão do arquivo (atualmente estando todos os arquivos na versão ”1”), em seguida, cada linha representa um

problema do benchmark. A primeira coluna corresponde ao grupo ao que o problema pertence. Segundo (STURTEVANT, 2012), para dar mais variabilidade ao conjunto de problemas de teste do benchmark, a geração do conjunto de problemas de teste se deu limitando-se a quantidade de problemas cuja solução tem tamanho semelhante. Primeiramente, foram escolhidos, aleatoriamente, pares de pontos do mapa. Em seguida, para cada problema, foi encontrado o caminho ótimo e, baseado no tamanho do caminho encontrado, o problema foi agrupado num grupo de problemas cujas soluções variam de 4 em 4 passos. Ou seja, um conjunto de problemas cuja solução varia entre 1 e 4 passos, outro que varia entre 5 e 8, outro com soluções variando entre 9 e 12 passos e assim por diante. Então, cada grupo foi limitado para ter no máximo 10 problemas. A segunda coluna do arquivo de testes indica qual o mapa sobre o qual o problema se refere. As terceira e quarta colunas indicam, respectivamente, a quantidade de colunas e de linhas do mapa, que, a rigor, são informações redundantes. As duas colunas seguintes correspondem às coordenadas de coluna e linha do ponto inicial do problema e as duas próximas são a coluna e linha do ponto final. A nona e última coluna corresponde ao custo de um menor caminho encontrado. Vale notar que para um dado problema pode haver mais de uma solução ótima.

4.2 Aplicação em Java

Para validar o algoritmo proposto foi desenvolvida uma aplicação utilizando a linguagem Java. Nesta aplicação foram implementados o A* e a otimização proposta. Inicialmente a aplicação carrega os arquivos de mapas do benchmark, então carrega os arquivos de problema e executa a otimização dos mapas, porém preservando os mapas originais. Com os mapas e problemas carregados, os problemas são executados utilizando-se os mapas originais e suas contrapartes otimizadas. Findo o processo, os resultados da execução são armazenados em arquivo.

Para execução do algoritmo proposto foi utilizado um notebook Dell 5458 modificado, contando com um processador 5200U, com dois núcleos de 2.2GHz, com memória ram de 16GB e uma memória de tipo SSD e sistema operacional Windows10.

4.3 Resultados

Nesta seção serão apresentados os resultados obtidos durante a execução do algoritmo.

A Tabela 1 dá uma visão geral dos resultados obtidos. Ao todo, foram otimizados 156 mapas, com uma redução média de espaço de busca de 97,42%, com desvio padrão de 0,026. O tempo para geração das NavMeshes foi de 4.498 milissegundos.

Resumo da otimização dos mapas	
Quantidade de mapas	156
Redução média no espaço de busca	97,42%
Desvio padrão da redução no espaço de busca	0,026
Tempo de geração das NavMeshes (milissegundos)	4.498
Tempo total A* mapa original (milissegundos)	12.545
Tempo marginal A* mapa otimizado (milissegundos)	7.732
Tempo absoluto A* mapa otimizado (milissegundos)	12.230
Redução marginal de tempo usando a heurística	46,76%
Redução absoluta de tempo usando a heurística	2,51%

Tabela 1 – Resumo da Otimização dos mapas

Houve uma redução no tempo de execução das buscas de 12.545 milissegundos para 7.732 milissegundos considerando-se o custo marginal ou para 12.230 milissegundos considerando-se o tempo da geração das NavMeshes. Percentualmente, as reduções de tempo marginais e absolutas foram de 46,76% e 2,52%. Para o cálculo do tempo da execução da busca no mapa otimizado, este tempo tem que ser diluído pela quantidade de execuções de buscas. Num jogo eletrônico é comum que uma grande quantidade de buscas seja executada.

Para visualizar os resultados com mais detalhes foram selecionados 30 mapas entre 156, cujos resultados estão apresentados na Tabela 2. A tabela tem as colunas **Mapa**, que é o nome do arquivo do mapa, **Tempo heurística**, que é o tempo gasto para se gerar o mapa, **Linhas**, **Colunas**, **Células**, que é a quantidade de células do mapa (espaço de busca do mapa original), **Pontos de navegação**, a quantidade de pontos de navegação gerados pelo algoritmo (espaço de busca otimizado) e **Redução**, percentual de redução do espaço de busca alcançado pelo algoritmo.

Vale salientar que para o mapa original, células não navegáveis também fazem parte do espaço de busca, já que cada célula armazena apenas sua posição, não armazenando em si uma lista de seus vizinhos navegáveis, sendo necessário se consultar no mapa para se filtrar se um dado vizinho é uma área navegável ou não. Já no mapa otimizado, os pontos de navegação são ligados se são visíveis entre si e armazenam o custo desta ligação e a sequência de nós intermediários entre dois pontos de navegação (o caminho entre dois dados pontos de navegação).

Tabela 2 – Redução do espaço de busca

Mapa	**Tempo heurística**	**Linhas**	**Colunas**	**Células**	**Pontos de navegação**	**Redução**
arena.map	88	49	49	2401	105	95,627%
den407d.map	4	57	33	1881	110	94,152%
lak101d.map	4	31	30	930	53	94,301%

orz107d.map	4	48	34	1632	81	95,037%
orz201d.map	4	45	47	2115	129	93,901%
orz203d.map	4	35	19	665	35	94,737%
orz601d.map	6	47	107	5029	320	93,637%
orz704d.map	8	46	84	3864	158	95,911%
ost102d.map	4	22	28	616	36	94,156%
oth999d.map	3	32	49	1568	4	99,745%
arena2.map	268	209	281	58729	1043	98,224%
den001d.map	29	80	211	16880	475	97,186%
den011d.map	39	167	247	41249	1414	96,572%
den400d.map	175	268	259	69412	1698	97,554%
den401d.map	50	113	259	29267	1122	96,166%
den500d.map	174	417	288	120096	1718	98,569%
den501d.map	71	338	320	108160	2014	98,138%
orz302d.map	18	145	146	21170	432	97,959%
lak302d.map	26	289	193	55777	1149	97,94%
lak504d.map	56	194	193	37442	2930	92,175%
orz700d.map	1585	1260	1104	1391040	7294	99,476%
orz701d.map	509	722	839	605758	4255	99,298%
orz702d.map	727	939	718	674202	5762	99,145%
orz703d.map	494	502	652	327304	3682	98,875%
orz800d.map	355	844	868	732592	2331	99,682%
orz900d.map	1671	656	1491	978096	7735	99,209%
orz901d.map	368	678	601	407478	3845	99,056%
orz999d.map	419	698	632	441136	1836	99,584%
ost000a.map	624	969	487	471903	10272	97,823%
ost000t.map	1900	971	487	472877	7910	98,327%

As Figuras 21, 22 e 23 apresentam gráficos de barra da redução de 30 mapas selecionados dentre os 156 mapas originais. A Figura 21 apresenta mapas com células variando entre 616 e 5029 células. A Figura 22 tem dados de mapas variando entre 16.880 e 120.096 células. A Figura 23 apresenta mapas com células variando entre 327.304 e 1.391.040 células.

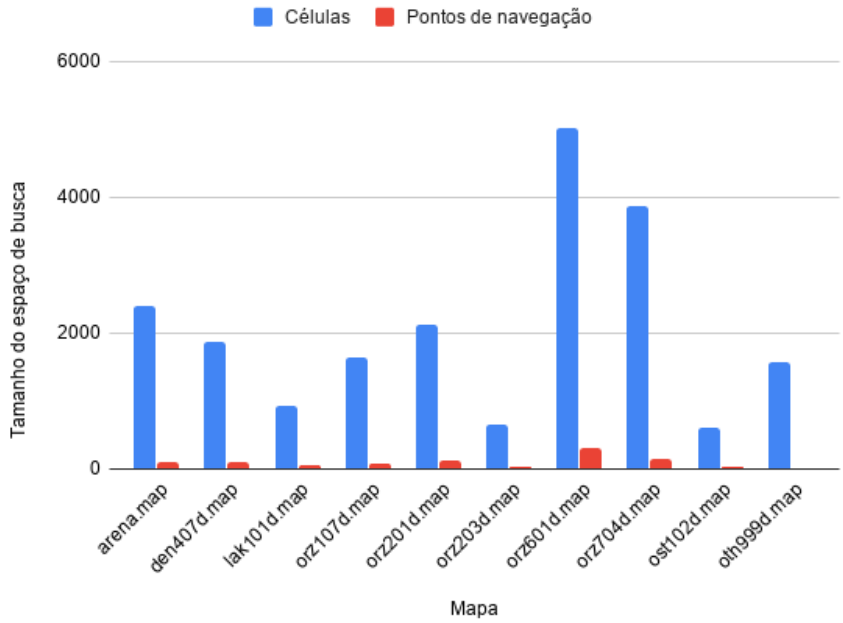

Figura 21 – Tamanho do espaço de busca - Mapas pequenos.

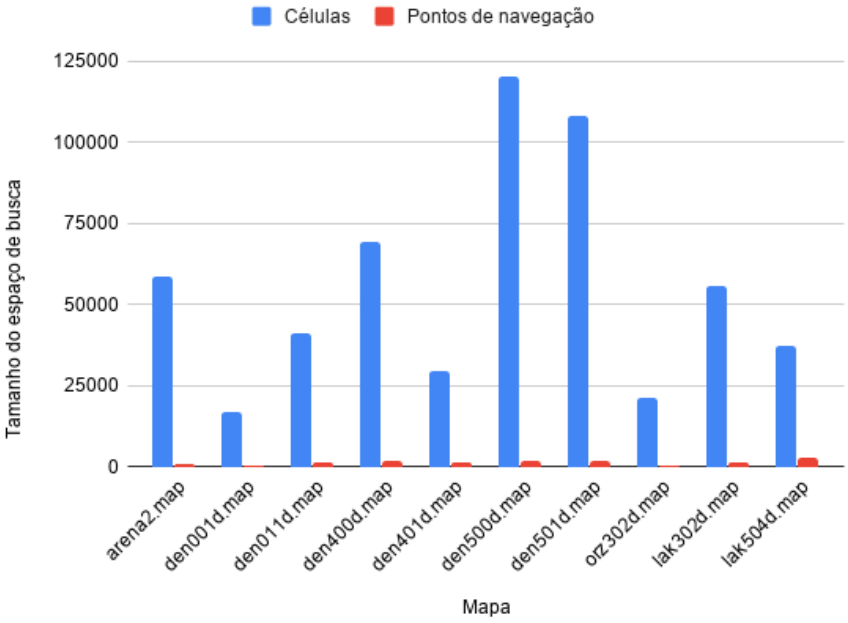

Figura 22 – Tamanho do espaço de busca - Mapas médios.

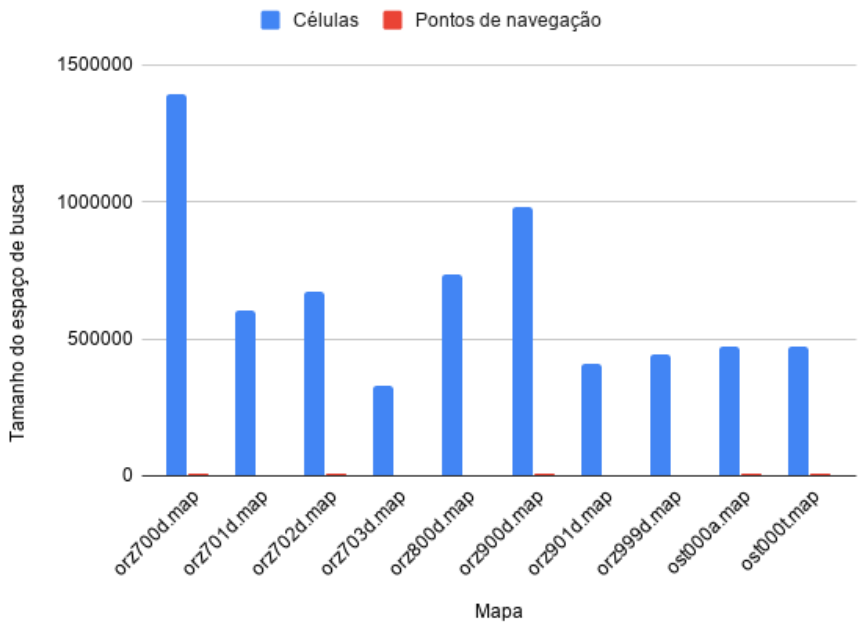

Figura 23 – Tamanho do espaço de busca - Mapas grandes.

A Tabela 3 mostra os resultados de 5 problemas para 6 mapas diferentes (2 mapas pequenos, 2 médios e 2 grandes). As colunas **Mapa**, **Colunas** e **Linhas** representam o nome do mapa, sua largura e altura. As colunas **Início** e **Destino** referem-se às coordenadas dos pontos iniciais e finais. Então é listado o tempo necessário para se gerar o mapa otimizado, na coluna **Tempo heurística**. A coluna **Tempo A* original** mostra o tempo em milisegundos necessário para se executar o A* no mapa original. A coluna **Tempo A* otimizado** mostra o tempo de execução do problema quando se executa a busca no mapa otimizado.

A coluna **Tempo total** é a soma do tempo A* otimizado com a quinta parte do tempo heurística, sendo o tempo total proporcional de execução de cada um dos 5 problemas de cada mapa. A coluna **Ótimo benchmark** tem o custo de um menor caminho do problema. A coluna **Ótimo heurística** mostra o custo do caminho encontrado quando se utiliza o mapa otimizado.

Observando estes resultados, vemos que o mapa ”ost102d.map”, um mapa (28x22) apresenta problemas cujo tempo de execução foram próximos a 1 milissegundo. Em um dos casos, o tempo de execução com o mapa original foi de menos de 1 milissegundo e o tempo de execução com o mapa otimizado foi de 1 milissegundo. O mapa ”orz900d.map”apresentou um problema em que a execução no mapa original foi superior à do mapa otimizado (769 milissegundos com o mapa original contra 796 milissegundos com o mapa otimizado).

É importante salientar que foram considerados apenas 5 caminhos distintos em cada mapa e portanto, o tempo total (Tempo A* + Tempo heurística/5) ainda é alto em alguns problemas.

Tabela 3 – Otimização em caminhos diferentes

Mapa	**Colunas**	**Linhas**	**Início**	**Destino**	**Tempo heurística**	**Tempo A* original**	**Tempo A* otimizado**	**Tempo total**	**Ótimo bench-mark**	**Ótimo heurística**
arena.map	49	49	(21,13) (28,8) (8,25) (6,3) (45,6)	(44,45) (15,44) (45,15) (18,38) (8,26)	86	203 37 37 39 40	8 4 5 5 4	25,2 21,2 22,2 22,2 21,2	41,526 41,384 41,142 40,556 45,284	41,527 41,385 41,142 40,556 45,284
Tempo total						356	26	112		
ost102d.map	28	22	(18,17) (25,15) (23,15) (26,14) (9,6)	(6,17) (8,7) (6,15) (7,11) (26,14)	4	0 1 1 1 0	0 1 0 0 1	0,8 1,8 0,8 0,8 1,8	21,071 20,313 20,899 20,242 20,313	21,071 20,899 20,899 20,243 22,071
Tempo total						3	2	6		
arena2.map	281	209	(97,97) (149,119) (118,110) (155,90) (166,103)	(56,106) (118,86) (82,93) (119,116) (143,67)	315	400 239 149 91 119	192 150 65 75 71	255 213 128 138 134	44,727 45,840 44,213 46,769 45,526	44,728 45,841 44,213 46,77 45,527
Tempo total						998	553	868		

den400d.map	259	268	(249,13) (202,191) (51,71) (73,115) (191,181)	(242,18) (205,201) (60,69) (69,106) (191,171)	159	339 498 422 319 318	132 146 137 138 259	163,8 177,8 168,8 169,8 290,8	9,071 11,242 9,828 10,656 10,000	9,071 11,243 9,828 10,657 10
Tempo total						1896	812	971		
orz700d.map	1104	1260	(219,1145) (25,742) (247,1200) (190,964) (269,1240)	(203,1172) (22,775) (272,1224) (169,991) (239,1232)	1487	640 891 1028 936 705	485 622 628 643 395	782,4 919,4 925,4 940,4 692,4	33,627 34,242 34,941 35,698 33,313	33,627 34,243 34,941 35,698 33,314
Tempo total						4200	2773	4260		
orz900d.map	1491	656	(1208,393) (999,369) (509,443) (484,381) (848,246)	(1194,371) (1025,356) (516,469) (477,353) (876,245)	2447	772 769 1374 1304 873	729 796 693 793 555	1218,4 1285,4 1182,4 1282,4 1044,4	29,556 31,384 28,899 30,899 28,414	29,556 31,385 28,899 30,899 28,414
Tempo total						5092	3566	6013		

As Tabelas 4, 5 e 6 têm o mesmo formato da tabela 2, porém cada linha corresponde a um problema em um mapa diferente. Elas foram divididos em 3 grupos. Respectivamente, mapas pequenos, médios e grandes.

Tabela 4 – Otimização em mapas diferentes (mapas pequenos)

Mapa	Colunas	Linhas	Início	Destino	Tempo heurística	Tempo A* original	Tempo A* otimizado	Tempo total	Ótimo benchmark	Ótimo heurística
arena.map	49	49	(21,13) (28,8) (8,25)	(44,45) (15,44) (45,15)	86	203 37 37	8 4 5	36,667 32,667 33,667	41,526 41,384 41,142	41,527 41,385 41,142
Tempo total						277	17	103,001		
den407d.map	33	57	(17,46) (22,12) (16,53)	(26,18) (26,36) (11,27)	4	1 1 2	1 1 1	2,333 2,333 2,333	31,727 29,656 28,071	31,728 29,657 28,657
Tempo total						4	3	6,999		
lak101d.map	30	31	(7,30) (13,26) (8,8)	(7,9) (7,6) (13,26)	3	1 0 1	1 0 0	2 1 1	21,828 22,485 20,071	21,828 22,485 20,071
Tempo total						2	1	4		
orz107d.map	34	48	(11,26) (7,2) (16,34)	(16,6) (25,12) (12,12)	4	1 1 1	1 1 1	2,333 2,333 2,333	22,071 22,142 23,656	22,071 22,142 23,657
Tempo total						3	3	6,999		
orz201d.map	47	45	(8,4) (15,12) (23,31)	(18,8) (22,24) (26,17)	4	1 1 1	1 2 1	2,333 3,333 2,333	14,000 15,485 15,828	14 15,485 15,828
Tempo total						3	4	7,999		

orz203d.map	19	35	(16,9) (7,13) (10,34)	(8,27) (10,34) (7,15)	3	1 0 1	0 1 0	1 2 1	21,313 22,242 20,242	21,314 22,243 20,243
Tempo total						2	1	4		
orz601d.map	107	47	(74,7) (43,18) (79,23)	(75,18) (49,8) (90,27)	6	4 4 4	2 3 3	4 5 5	15,656 13,656 12,656	15,657 13,657 12,657
Tempo total						12	8	14		
orz704d.map	84	46	(28,33) (52,18) (50,32)	(50,23) (42,40) (52,7)	10	8 8 8	6 7 5	9,333 10,333 8,333	26,142 26,142 25,828	26,142 26,142 25,828
Tempo total						24	18	27,999		
ost102d.map	28	22	(18,17) (25,15) (23,15)	(6,17) (8,7) (6,15)	4	0 1 1	0 1 0	1,333 2,333 1,333	21,071 20,313 20,899	21,071 20,899 20,899
Tempo total						2	1	4,999		
oth999d.map	49	32	(13,28) (12,31) (13,3)	(47,30) (7,0) (37,28)	3	2 2 2	1 0 0	2 1 1	34,828 33,071 34,941	34,828 96,042 34,941
Tempo total						6	1	4		

Tabela 5 – Otimização em mapas diferentes (mapas médios)

Mapa	Colunas	Linhas	Início	Destino	Tempo heurística	Tempo A* original	Tempo A* otimizado	Tempo total	Ótimo benchmark	Ótimo heurística
arena2.map	281	209	(97,97) (149,119) (118,110)	(56,106) (118,86) (82,93)	315	400 239 149	192 150 65	297 255 170	44,727 45,840 44,213	44,728 45,841 44,213
Tempo total						788	407	722		
den001d.map	211	80	(109,48) (46,44) (71,51)	(69,42) (132,37) (132,8)	22	32 26 28	17 17 16	24,333 24,333 23,333	42,485 88,899 89,355	42,485 88,899 89,355
Tempo total						86	50	71,999		
den011d.map	247	167	(152,51) (130,44) (110,89)	(165,69) (126,20) (111,62)	29	76 46 41	20 21 19	29,667 30,667 28,667	24,556 25,656 27,414	24,556 25,657 27,414
Tempo total						163	60	89,001		
den400d.map	259	268	(249,13) (202,191) (51,71)	(242,18) (205,201) (60,69)	159	339 498 422	132 146 137	185 199 190	9,071 11,242 9,828	9,071 11,243 9,828
Tempo total						1259	415	574		
den401d.map	259	113	(204,72) (105,68) (58,50)	(182,72) (85,60) (37,57)	23	31 35 38	15 16 15	22,667 23,667 22,667	22,000 23,313 23,899	22 23,314 23,899
Tempo total						104	46	69,001		

den500d.map	288	417	(103,69) (185,366) (76,127)	(115,77) (179,354) (91,126)	52	82 72 74	27 30 30	44,333 47,333 47,333	15,313 14,485 15,414	15,314 14,485 15,414
Tempo total						228	87	138,999		
den501d.map	320	338	(259,288) (153,280) (294,253)	(267,282) (156,288) (299,245)	50	76 80 75	32 33 32	48,667 49,667 48,667	11,071 9,242 10,071	11,071 9,243 10,071
Tempo total						231	97	147,001		
orz302d.map	146	145	(89,42) (115,55) (119,84)	(105,38) (110,36) (124,65)	15	14 13 13	7 8 6	12 13 11	20,242 21,071 21,071	20,243 21,071 21,071
Tempo total						40	21	36		
lak302d.map	193	289	(191,233) (30,88) (181,81)	(117,259) (10,150) (167,157)	24	28 231 62	13 13 13	21 21 21	84,769 86,769 85,313	84,77 86,77 85,314
Tempo total						321	39	63		
lak504d.map	193	194	(148,104) (160,116) (11,71)	(115,93) (146,78) (43,43)	118	162 217 264	64 66 66	103,333 105,333 105,333	47,213 44,970 46,526	47,213 45,556 47,113
Tempo total						643	196	313,999		

Tabela 6 – Otimização em mapas diferentes (mapas grandes)

Mapa	Colunas	Linhas	Início	Destino	Tempo heurística	Tempo A* original	Tempo A* otimizado	Tempo total	Ótimo benchmark	Ótimo heurística
orz700d.map	1104	1260	(219,1145) (25,742) (247,1200)	(203,1172) (22,775) (272,1224)	1487	640 891 1028	485 622 628	980,667 1117,667 1123,667	33,627 34,242 34,941	33,627 34,243 34,941
Tempo total						2559	1735	3222,001		
orz701d.map	839	722	(199,134) (110,290) (619,529)	(232,130) (125,319) (610,560)	524	347 354 344	182 192 176	356,667 366,667 350,667	34,656 35,213 34,727	34,657 35,213 34,728
Tempo total						1045	550	1074,001		
orz702d.map	718	939	(546,693) (383,296) (397,385)	(572,673) (359,321) (368,398)	753	710 761 830	379 392 371	630 643 622	34,284 34,941 34,384	34,284 36,698 34,385
Tempo total						2301	1142	1895		
orz703d.map	652	502	(83,290) (204,10) (398,291)	(68,315) (215,39) (399,324)	475	448 606 728	484 517 446	642,333 675,333 604,333	31,213 33,556 33,414	31,799 33,556 33,414
Tempo total						1782	1447	1921,999		
orz800d.map	868	844	(569,753) (4,673) (442,171)	(545,743) (12,700) (444,144)	276	200 248 271	202 138 124	294 230 216	28,142 30,313 30,071	28,142 30,314 30,071

Tempo total						719	464	740		
orz900d.map	1491	656	(1208,393) (999,369) (509,443)	(1194,371) (1025,356) (516,469)	2447	772 769 1374	729 796 693	1544,667 1611,667 1508,667	29,556 31,384 28,899	29,556 31,385 28,899
Tempo total						2915	2218	4665,001		
orz901d.map	601	678	(398,244) (119,157) (267,118)	(365,214) (111,110) (248,127)	295	62 297 292	8 200 206	106,333 298,333 304,333	45,426 50,313 51,899	50,698 50,314 51,899
Tempo total						651	414	708,999		
orz999d.map	632	698	(338,527) (384,539) (467,239)	(366,525) (407,527) (453,215)	409	348 263 343	178 227 208	314,333 363,333 344,333	30,485 29,142 29,798	30,485 29,142 30,385
Tempo total						954	613	1021,999		
ost000a.map	487	969	(258,732) (130,405) (17,468)	(274,752) (159,411) (33,444)	768	1530 1161 1649	495 535 591	751 791 847	27,213 31,485 30,627	28,971 31,485 30,627
Tempo total						4340	1621	2389		
ost000t.map	487	971	(205,704) (250,751) (272,604)	(223,690) (271,753) (263,622)	1950	1349 1242 932	654 589 472	1304 1239 1122	23,798 21,828 21,727	26,971 21,828 21,728
Tempo total						3523	1715	3665		

Tabela 7 – Resumo dos resultados de busca (em milissegundos)

	Somatório Tempo A* original	**Somatório Tempo A* otimizado**	**Somatório Tempo total**
Mapas pequenos	335	57	184
Mapas médios	3.863	1.418	2.225
Mapas grandes	20789	11.919	21.303

Analisando as Tabelas 4 e 5, verifica-se que para problemas pequenos e médios, em alguns casos o tempo total (tempo da busca somado com a diluição do tempo da heurística) no mapa otimizado é menor do que o tempo usando o mapa original. Analisando a Tabela 6, percebe-se que a otimização proposta só é vantajosa para o caso em que vários problemas de caminho forem resolvidos no mesmo mapa otimizado, já que a heurística é feita apenas uma vez.

Tomando como exemplo uma caso mais específico, o mapa ”lak105d.map”, por exemplo, teve uma redução de 86,83% no seu espaço de busca, sendo a menor redução alcançada. Contando com 25 células e 31 colunas, suas 775 células foram reduzidas para 102 pontos de navegação, levando poucos 4 milissegundos para otimização deste mapa.

O mapa ”ost101d.map”teve a maior redução, de 99,93%. Partindo de 540.672 células (704 linhas e 768 colunas) para ter apenas 373 pontos de navegação, com tempo de geração de apenas 295 milisegundos. Na média, a redução de espaço de busca foi de 97,42%, com desvio padrão de 0,026.

5 Conclusões

Haja vista a escassez de trabalhos publicados sobre otimização de mapas bidimensionais homogêneos para *pathfinding*, talvez o objetivo mais significativo deste trabalho seja justamente o de contribuir para o desenvolvimento de referências para comparação de outras propostas de otimização de mapas.

Os resultados obtidos com o algoritmo proposto foram promissores quando aplicado ao jogo Dragon Age: Origins. Quanto ao tempo de execução da busca com os mapas otimizados, houve uma redução média de tempo de 46,76%, um resultado bastante expressivo. No entanto, o custo para gerar os mapas otimizados é elevado, principalmente em mapas de maior dimensão. Assim, a aplicação é vantajosa quando diversos problemas precisam ser solucionados no mesmo mapa.

5.1 Trabalhos futuros

Apesar do problema de busca de caminho ter uma solução ótima conhecida, simples e consolidada, o algoritmo de Dijkstra, há muito espaço para otimizações. Algumas otimizações, como o algoritmo A* comprometem o resultado, podendo encontrar soluções subótimas, algumas otimizações mudam a representação dos dados, como a apresentada no presente trabalho.

Este trabalho pode ser melhorado, através de uma análise mais profunda de uso de memória e processamento do computador.

Para se exaurir as possibilidades do algoritmo proposto, ele pode ser conjugado com outras técnicas de otimização de busca. Possivelmente a junção de técnicas trará resultados ainda melhores dos que os obtidos apenas pelo algoritmo. Possíveis problemas podem trazer novos desafios que podem ensejar a novas melhorias na área de busca de caminho.

Visando dar maior divulgação ao algoritmo desenvolvido, cogitamos reescrever este trabalho em formato de artigo, bem como criarmos uma API para uso do código desenvolvido.

Referências

BIOWARE. *DRAGON Age: Origins*. 2009. [CD-ROM]. Disponível em: <http://dragonage.bioware.com/>.

CHOSET, H.; BURDICK, J. Sensor-based exploration: The hierarchical generalized voronoi graph. *The International Journal of Robotics Research*, v. 19, n. 2, p. 96–125, 2000. Disponível em: <https://doi.org/10.1177/02783640022066770>.

CORMEN, T. H. et al. *Introduction to Algorithms, Third Edition*. 3rd. ed. [S.l.]: The MIT Press, 2009. 595–-601 p. ISBN 0262033844, 9780262033848.

CUI, X.; SHI, H. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, International Journal of Computer Science and Network Security (IJCSNS), v. 11, n. 1, p. 125–130, 2011. Disponível em: <http://vuir.vu.edu.au/8868/>.

EDELKAMP, S.; SCHROEDL, S.; KOENIG, S. *Heuristic Search: Theory and Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN 0123725127, 9780123725127.

Geraerts, R. Planning short paths with clearance using explicit corridors. In: *2010 IEEE International Conference on Robotics and Automation*. [S.l.: s.n.], 2010. p. 1997–2004. ISSN 1050-4729.

HALE, D. H. *A Growth-based Approach to the Automatic Generation of Navigation Meshes*. Tese (Doutorado), Charlotte, NC, USA, 2011. AAI3493710.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2), p. 100–107, 1968.

OLIVA, R.; PELECHANO, N. Automatic generation of suboptimal navmeshes. In: . [S.l.: s.n.], 2011. p. 328–339.

PATHFINDING.JS. 2017. Disponível em: <https://qiao.github.io/PathFinding.js/visual/>. Acesso em: 01-06-2019.

PELECHANO, N.; FUENTES, C. Hierarchical path-finding for navigation meshes (hna*). *Comput. Graph.*, Pergamon Press, Inc., Elmsford, NY, USA, v. 59, n. C, p. 68–78, out. 2016. ISSN 0097-8493. Disponível em: <http://dx.doi.org/10.1016/j.cag.2016.05.023>.

RODRIGUES, P. S. *Computação Gráfica: triangulação de Delauney*. 2017. Vídeo-Aula Triangulação de Delauney – Out/2017. Disponível em: <https://www.youtube.com/watch?v=nzayrUTpaEA>. Acesso em: 30-06-2019.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 0136042597, 9780136042594.

SNOOK, G. Simplified 3d movement and pathfinding using navigation meshes. In: DELOURA, M. (Ed.). *Game Programming Gems*. [S.l.]: Charles River Media, 2000. p. 288–304.

SOUZA, S. C. B. de. *PLANEJAMENTO DE TRAJETÓRIA PARA UM ROBÔ MÓVEL COM DUAS RODAS UTILIZANDO UM ALGORITMO A-ESTRELA MODIFICADO*. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2008.

STURTEVANT, N. *Moving AI Lab*. 1999. Disponível em: <https://movingai.com/>.

STURTEVANT, N. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, v. 4, n. 2, p. 144 – 148, 2012. Disponível em: <http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf>.

STURTEVANT, N. R.; GEISBERGER, R. A comparison of high-level approaches for speeding up pathfinding. In: *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, October 11-13, 2010, Stanford, California, USA*. [s.n.], 2010. Disponível em: <http://aaai.org/ocs/index.php/AIIDE/AIIDE10/paper/view/2139>.

TOLL, W. van et al. A comparative study of navigation meshes. In: *Proceedings of the 9th International Conference on Motion in Games*. New York, NY, USA: ACM, 2016. (MIG '16), p. 91–100. ISBN 978-1-4503-4592-7. Disponível em: <http://doi.acm.org/10.1145/2994258.2994262>.

WEIN, R.; BERG, J. P. van den; HALPERIN, D. The visibility–voronoi complex and its applications. *Computational Geometry*, v. 36, n. 1, p. 66 – 87, 2007. ISSN 0925-7721. Special Issue on the 21st European Workshop on Computational Geometry. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0925772106000496>.

WIKIPEDIA. *Taxicab geometry*. 2019. Disponível em: <https://en.wikipedia.org/wiki/Taxicab_geometry>. Acesso em: 30-06-2019.