ANÁLISE DA APLICAÇÃO DO ALGORITMO A* NA RESOLUÇÃO DO PROBLEMA DE PATHFINDING

KAMON F. MESQUITA, MARCO AURÉLIO D. AGNESE, RAFAEL S. MENDANHA

Av. Universitária, n.º 1488 - quadra 86 - bloco A - 3º piso - Setor Leste Universitário
Goiânia - Goiás - CEP: 74605-010
E-mails: kmon_fm@hotmail.com, mdallag@outlook.com, rafaelsmendanha@gmail.com

Abstract: This article aims to present a study about search of optimal paths using the A star algorithm. To achieve this goal, it was formulated a simple two-dimensional problem, which goal is to find the best route from one point to another, dodging the obstacles present in the scenario. The resolution used the A star with some procedures to reduce the complexity of the problem such as the simplification of the search area and limitation of movement of the agent who want to find the best possible path. Some limitations and observations about this algorithm are also presented.

Keywords: A star, search, optimization, heuristic, *pathfinding*.

Resumo: O presente artigo tem como objetivo apresentar um estudo sobre busca de caminhos ótimos com utilização do algoritmo A*. Para atingir este objetivo, foi formulado um problema simples bidimensional, cujo objetivo é encontrar o melhor caminho de um ponto a outro, desviando-se dos obstáculos presentes no cenário. Na resolução utilizou-se o A* com alguns procedimentos para diminuir a complexidade do problema, como a simplificação da área de busca e limitação dos movimentos do agente que deseja procurar o melhor caminho possível. Também são apresentadas algumas limitações e observações acerca deste algoritmo**.**

Palavras-chave: A estrela, search, otimização, heurística, *pathfinding*.

1 Introdução

O problema que será discutido nesse trabalho será o de encontrar um caminho mínimo entre de um ponto inicial até um ponto final num dado ambiente evitando obstáculos. Esse problema é conhecido como pathfinding.

Após a determinação do caminho através do pathfinding é utilizado um algoritmo trivial de movimento para deslocar o objeto sobre a trajetória (Patel, 2014).

O algoritmo que será utilizado no pathfiding é conhecido como A* (A-estrela). É um algoritmo de busca pela melhor escolha, como será explicado na seção 3.2. Foi descrito pela primeira vez em 1968 por Peter Hart, Nils Nilsson e Bertram Raphael com o nome A, passando a se chamar A* após ser utilizado com uma heurística para atingir um comportamento ótimo. Trata-se de uma extensão do Algoritmo de Dijkstra que utiliza uma heurística.

Na seção 2 será descrito o problema a ser resolvido, na seção 3 serão apresentado os algorítimos genéticos. Na seção 4 será descrito o funcionamento do algoritmo A*. Na seção 5 é descrito algumas das limitações deste algoritmo e, em seguida, o modo que o algoritmo foi implementado é exposto. Depois, é discutido os resultados da análise. E por último, são feitas as considerações finais acerca do trabalho.

2 Descrição do Problema

O problema que será abordado é composto por um ambiente, um agente, origem, destino, obstáculos e um caminho conforme disposto na figura 1. O agente é o gato, a origem é a posição do gato, o destino é a posição do osso, os obstáculos são os quadrados hachurados.

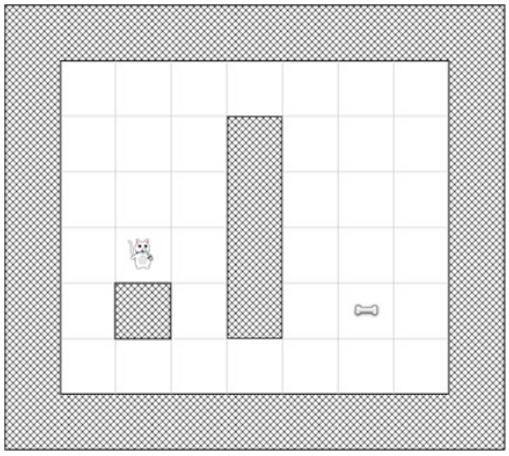


Figura 1. Uma visão dos elementos que constituem o problema a ser resolvido. Fonte: Introduction to A* Pathfinding[1]

O ambiente é uma grade bidimensional formada por quadrados. Estes quadrados podem ser passáveis sem obstáculo ou não passáveis – com e sem obstáculos, respectivamente. Por sua vez, o agente é a entidade que pode se mover sobre o ambiente, percorrendo um caminho, que ser inicia no ponto A, a origem, e termina no ponto B, o destino.

O agente pode ser deslocar no ambiente através de quatro tipos de movimentos, para cima, para baixo, para a direita e para a esquerda. Não é permitido movimentos nas diagonais.

[1]Disponível em: <http://www.raywenderlich.com/4946/introduction-to-a-pathfinding> Acesso: 25/04/2014.

O caminho é uma sequência de quadrados passáveis que se inicia com o quadrado de origem e termina com o quadrado destino. Além disso, dois quadrados sucessivos da sequência devem ser adjacentes na grade bidimensional, isto é, seja A e B quadrados, dizemos que B é adjacente a A se e somente se B está imediatamente em cima, em baixo, à esquerda ou à direita de A. O algoritmo A* será utilizado para determinar o caminho, caso este exista.

Os pontos de origem, destino e obstáculos são pré-determinados ao acaso. O agente não possui um sensor de colisão, mas conhece exatamente onde estão os obstáculos e o ponto destino, apenas não conhece um caminho válido para chegar ao destino..

3 Algoritmos Genéticos

Algoritmos Genéticos (AGs) representam uma família de algoritmos da Computação Evolucionária fundamentados nas teorias de Mendel e Lamarck. que utilizam transições probabilísticas, e não regras determinísticas. Empregam, também, os conceitos de reprodução a partir de uma população inicial ou possíveis soluções para gerar novas soluções (descendentes).

A representação dos indivíduos é feita por uma sequência de caracteres binários chamados de genótipos. A reprodução ocorre principalmente por crossover (cruzamento) entre um ou mais indivíduos selecionados. Uma máscara determina a sequência de caracteres dos descendentes a partir da escolha aleatória de um ou vários pontos do genótipo dos pais, sendo que dois indivíduos complementares são gerados.

Admite-se também a reprodução por mutação, onde cada indivíduo e bit do genótipo tem certa probabilidade de alterar seu material genético. Cada iteração é chamada de geração e tem como objetivo avaliar os indivíduos da população atual, selecionar os que participarão da próxima geração, recombiná-los ou mutá-los para formar uma nova população.

Apesar de não haver provas formais que comprovem a utilidade destes algoritmos em problemas complexos, na prática, o seu uso gera bons resultados em diversas situações. Contudo, este tipo de algoritmo não garante um ótimo global e são designados principalmente para problemas de otimização que não podem ser solucionados em tempo hábil por outros tipos de algoritmos. Durante a proposta deste projeto, foi cogitada a utilização de AGs, porém estes não asseguram que o caminho encontrado seja o menor possível (ótimo global). Já o A* não só era capaz de resolver o problema proposto como também garantir um ótimo global.

4 Algoritmo A*

Esta seção descreverá inicialmente uma forma simplificada do algoritmo A* de maneira intuitiva. Em seguida será descrito de forma mais rigorosa e precisa.

4.1 Introdução ao Algoritmo

Nesta seção, será descrito a lógica passo a passo de uma busca de caminhos utilizando o algoritmo A*. Consideremos um problema simples, em que temos um agente que deseja percorrer o menor caminho possível para encontrar um objeto – neste caso, um osso de brinquedo – em um ambiente com obstáculos que devem ser desviados.

Primeiramente, é preciso simplificar a área de busca para facilitar a resolução do problema. Este processo depende do ambiente gráfico empregado. No problema citado, é conveniente utilizar uma grade bidimensional composta por quadrados, dispostos em uma matriz 7x6.

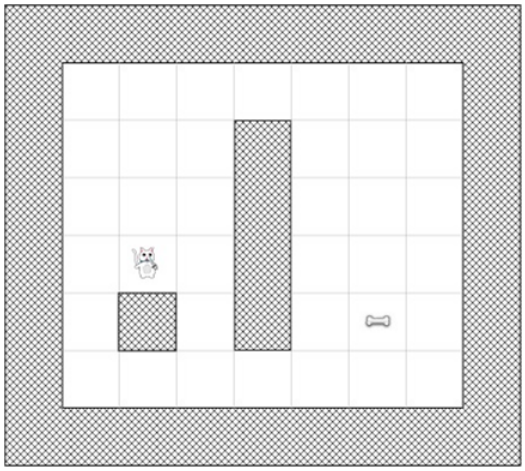


Figura 2. Formulação do problema com área de busca simplificada. Fonte: Introduction to A* Pathfinding[1].

Será permitido locomover-se entre quadrados vizinhos que não estejam nas diagonais da posição atual. Esta movimentação será realizada de um nó para outro, onde nó refere-se ao ponto central de um quadrado.

Agora, pode-se concentrar na determinação do caminho mais curto. Para isto, será preciso utilizar duas listas: uma para armazenar os nós que podem ser considerados para comporem o trajeto do menor caminho, chamada de lista aberta e outra para guardar os nós que não serão mais considerados nesta procura, chamada de lista fechada.

Quando a entidade inicia sua trajetória, o nó em que se encontra (nó de partida) será adicionado na lista fechada. Em seguida, serão adicionados os nós adjacentes ao de partida que não são intransponíveis (obstáculos) na lista aberta. Porém, é preciso determinar qual destes nós pertence ao menor caminho. Esta situação será resolvida pelo algoritmo A*, o qual define um peso para cada nó da lista aberta.

Esta função normalmente é chamada de F e é composta pela soma G + H, onde o primeiro fator refere-se ao custo de movimentação para se mover do ponto de início até o nó considerado e o segundo é o custo de movimentação estimado para mover deste nó até o destino final. Este último fator normalmente é chamado de heurística, porque não se sabe ao certo o valor deste custo, sendo apenas uma estimativa. Custo de movimentação neste caso é simplesmente uma quantidade de quadrados.

Para calcular G de um nó, se obtém o valor deste parâmetro do nó pai e lhe adiciona 1. Caso o nó pai seja o de partida, o valor de G será 1. O cálculo do H não é tão trivial, uma vez que se trata de uma estimativa.

Existem vários métodos para calcular o valor de H, mas este artigo irá explicar apenas um deles, chamado de método de Manhattan -- Manhattan distance method, Manhattan length ou city block distance. Consiste em calcular o número total de quadrados movidos horizontalmente e verticalmente para alcançar o de destino a partir do nó atual, sem realizar movimentações na diagonal e ignorando os obstáculos que possam estar neste caminho.

Para achar o melhor caminho até o nó de destino, pegue o quadrado da lista aberta com menor valor de H e remova-o da lista aberta e adicione-o à lista fechada. Para cada quadrado adjacente a este que não esteja nas diagonais deve ser verificado se ele está na lista fechada. Em caso afirmativo, ignore-o. Caso contrário, verifique se o quadrado está na lista aberta. Se não estiver, adicione-o e calcule H. Caso contrário, verifique se o quadrado apresenta o menor F quando percorremos o caminho gerado atual para chegar nele. Se for, atualize sua pontuação e a do seu pai também.

Para entender melhor este processo, verifiquemos os procedimentos para obter um trecho do caminho no problema citado. Em cada quadrado, o valor de F estará no canto esquerdo superior, o valor de G no canto esquerdo inferior e o valor de H no canto direito inferior.

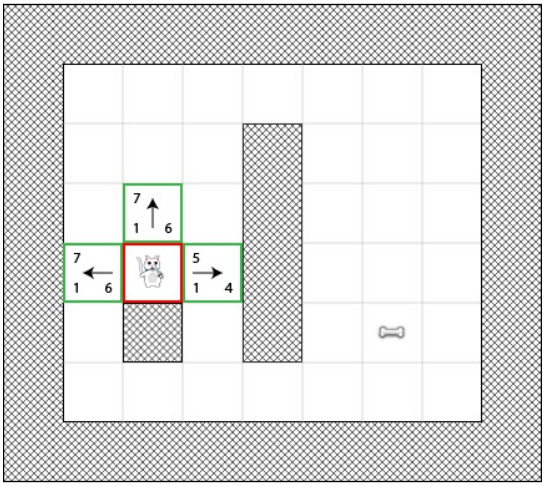


Figura 3. Início da busca. Fonte: Introduction to A* Pathfinding[1].

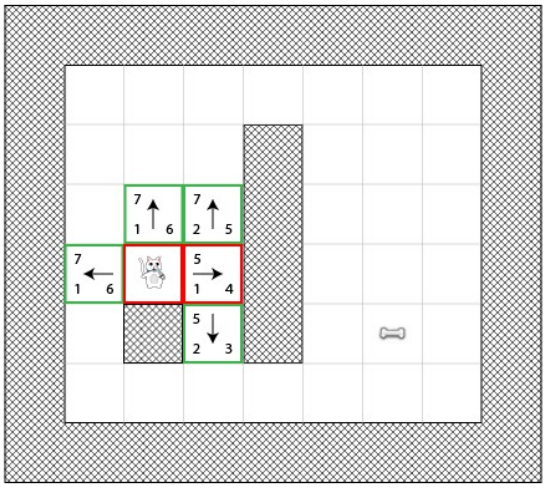


Figura 4. Escolha do quadrado de menor F. Fonte: Introduction to A* Pathfinding[1].

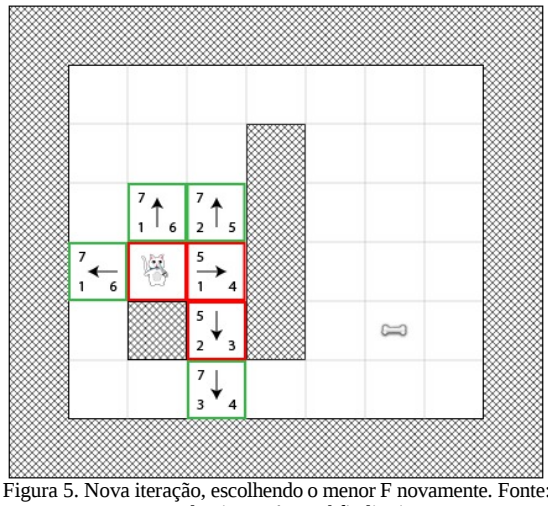


Figura 5. Nova iteração, escolhendo o menor F novamente. Fonte: Introduction to A* Pathfinding[1].

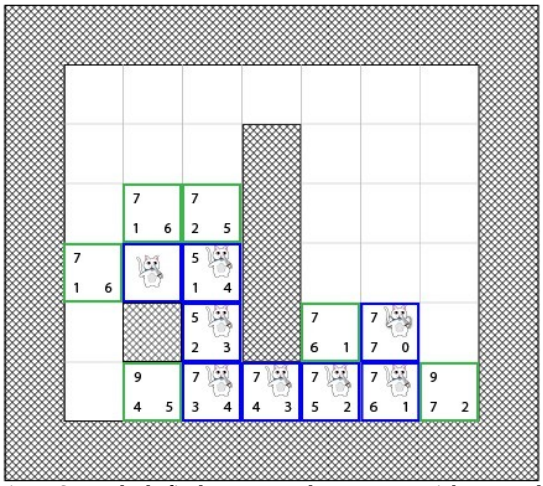


Figura 6. Resultado final, apresentando o menor caminho em azul. Fonte: Introduction to A* Pathfinding[1].

4.2 Formalização do Algoritmo

No item anterior o algoritmo A* foi descrito de forma intuitiva para facilitar o entendimento de seu funcionamento. Nesta seção o mesmo será descrito de forma mais precisa e genérica.

O ambiente foi particionado com quadrados, no entanto, pode dividido em qualquer outra forma geométrica (Lester, 2004). Cada parte da área de busca recebe o nome de nó. E a forma que esses nós se relacionam pode ser representados por um grafo. A busca é feita a partir da expansão dos nós do grafo. A maneira pela qual são determinados os nós a serem expandidos é conhecida como Estratégia de Busca.

As estratégias de busca podem ser com ou sem informação. Na primeira, o algoritmo de busca não recebe nenhuma informação além da definição do problema. Já na segunda, o algoritmo possui uma indicação de onde pode estar a solução. Possui uma função F que é utilizada como base para se expandir os nós, ela mede o quanto se está distante do objetivo. Por isso é essa busca é chamada de busca pela melhor escolha, escolhe (ou tenta escolher) o melhor nó para expandir. Como Russel & Norvig (2004) destacam, esse tipo de busca não sabe exatamente qual é a melhor escolha, mas o nó que será expandido é o que "parece" melhor de acordo com F(n), n representa o nó.

Existem vários algoritmos de busca pela melhor escolha como a busca gulosa e o A*. Uma importante característica dessa classe é que utiliza uma função heurística representada por H. H(n) é uma estimativa do custo de se mover do nó n até um nó objetivo. Será abordado apenas o A*.

O algoritmo A* avalia os nós através da função heurística H(n) combinada com a função G(n):

$$F(n) = H(n) + G(n)$$

a função G(n) mede o custo para se deslocar do nó n até o nó inicial. De acordo com Russel & Norvig (2004), a função F(n) representa a estimativa do caminho de menor custo que passa através do nó n. O algoritmo A* sempre escolhe o nó adjacente com menor valor de F.

O pseudocódigo para o algoritmo A* pode ser representado como:

1. Faça P o nó inicial.
2. Calcule o valor de G, H e F para o nó P.
3. Adicione P na lista aberta.
4. Faça B o nó da lista aberta com menor valor de F(n).
5. Mova B para a lista fechada.
    1. Se B é o nó destino, então o caminho foi encontrado.
    2. Se a lista aberta está vazia, então não existe caminho possível.
6. Para todo nó C adjacente a B faça:
    1. Se o nó C está na lista fechada ou não é passável o ignore
    2. Calcule o valor de G, H e F para o nó C.
    3. Se o nó C não está na lista aberta o adicione.
    4. Senão:
        1. Se o caminho passando por B tiver menor custo então faça C filho de B e atribua os novos valores G, H e F.
        2. Senão, não faça nada.
7. Repita o passo 4.

4.3 Heurística

Em relação à admissibilidade do algoritmo A*, observa-se que este obtém a solução ótima se: a quantidade de nós vizinhos for infinita, os arcos não possuem custos negativos e utilizar de heurísticas admissíveis. Uma heurística é declarada admissível caso não superestime o custo total do caminho, embora sejam obtidas a partir da simplificação do problema proposto, envolvendo, assim, menos restrições que o problema inicial.

A partir desse conceito, optou-se por utilizar e comparar duas outras heurísticas, além do Método Manhattan, para a implementação do A*. Todas elas são admissíveis, embora sejam adequadas para situações específicas e diferentes do problema proposto. São elas: Atalho Diagonal e Distância Euclidiana. Obviamente existem outros métodos para calcular H, mas estes são os mais comumente usados. No geral, quanto mais próximo da estimativa é a distância restante real ao longo do caminho para a meta, mais rapidamente A* vai encontrar esse objetivo.

Na heurística denominada Atalho Diagonal ou Diagonal Shortcut, o valor H calculado é equilibrado em relação ao valor G, sendo então sua principal vantagem, pois é capaz de considerar plenamente pequenos modificadores como uma penalidade de viragem ou a influência de um modificador de mapa. No caso proposto, não foram considerados movimentos na diagonal, o que exigiria que um valor ou peso diferente fosse atribuído a G. Ou seja, o método exigiria que a implementação fosse modificada para acomodar movimentos na diagonal. Empregou-se, portanto, tal heurística apenas com o propósito de comparação.

O algoritmo para o Diagonal Shortcut consiste em calcular a diferença entre as distâncias na vertical e horizontal, ou seja, x e y, entre o nó atual e o nó de destino. Em seguida, utilizando a abordagem de Patrick Lester (2004), o método retorna como resultado da heurística o menor valor entre as duas distâncias, somado do valor resultante da diferença entre o valor x e y do nó caso a distância y seja a menor - ou vice-versa, caso a menor distância seja a de x. No entanto, como o método busca beneficiar o movimento na diagonal, atribui-se pesos para os elementos da soma. Foi considerado que a primeira parcela seria multiplicada por 14, enquanto a segunda seria aumentada em 10 vezes.

Quando as unidades podem se movimentar em qualquer ângulo, é comum utilizar a heurística Distância Euclidiana ou Euclidian Distance, que é um algoritmo menor que os anteriores. Ele calcula a soma dos quadrados da diferença entre x e y do nó atual e nó de destino, e retorna a raiz quadrada desse valor multiplicado pelo peso, que no caso foi estabe-

lecido como valor 10. Existe também a variação deste algoritmo, denominada squared ou quadrada, que é similar ao convencional, porém ineficaz na resolução de problemas com cenários côncavos.

Assim como no Atalho Diagonal, o método da Distância Euclidiana é implementado e testado apenas para ser verificado frente a outros algoritmos, principalmente para avaliar a performance da heurística de Manhattan – apresentada durante parte deste texto, juntamente com o A*, por ser a mais usada e com melhor desempenho no caso proposto. Vale lembrar que seu método retorna a soma da diferença entre os caminhos x e y do nó atual até o nó de destino, multiplicado pelo peso, que é, novamente, 10. Assim, além de ser o mais utilizado na implementação do A*, ele é também o mais simples.

5 Limitações do *Pathfinding* e do Algoritmo A*

Dependendo da complexidade da heurística adotada para o algoritmo A*, principalmente em casos de mapas muito elaborados como os que constituem a maioria dos jogos atuais, o programa demandará alto custo de processamento, o que pode causar atrasos ou "congelamentos" de imagem até que o melhor caminho seja encontrado. (Cain, 2002) O problema é agravado quando é necessário percorrer uma grande quantidade de nós ou no qual o algoritmo realiza buscas para múltiplos agentes, sobretudo quando se deseja deslocá-los de um extremo a outro no mapa.

No entanto, sua principal limitação, ainda sem solução definitiva, é a inclusão de objetos dinâmicos. Embora existam procedimentos para impedir a sobrecarga de processamento, o algoritmo A* não demonstra desempenho satisfatório. Isso acontece porque, ao encontrar um objeto no cálculo do caminho, o algoritmo bloqueia-o, ao considerar que não é possível percorrer aquele nó, o que atrapalhará o reconhecimento real do caminho menor. No caso, o algoritmo continuaria normalmente examinando outros nós e não encontraria a melhor solução. Para evitar essa excesso de consumo de processamento, reduz-se o número de objetos dinâmicos, o que limita o potencial do programa, ou emprega-se redes neurais, à custo de complexidade de implementação.

Como solução alternativa para o algoritmo A*, é comumente utilizado o algoritmo D*, que requer menos linhas de código para ser implementado e é mais leve. O algoritmo permite que os custos de nós possam ser alterados à medida que o agente se move através do mapa e apresenta uma abordagem para modificar as estimativas de custo em tempo real. (Stentz, 1994) A desvantagem dessa abordagem é, novamente, o tempo de CPU, logo ele força um limite sobre os objetos dinâmicos que podem ser introduzidos no ambiente. Vale ressaltar que a variação do algoritmo denominada IDA* também apresenta os mesmos prejuízos, embora utilize menos memória.

Como se pretende implementar o algoritmo para *pathfinding*, deve-se levar em consideração as limitações do ambiente proposto. O pré-processamento é o principal empecilho, pois é necessário simular os caminhos possíveis, ao processar os nós que serão percorridos. No *pathfinding* também não é possível produzir movimentos críveis e realistas dos agentes; assim, incapaz de lidar com mundos dinâmicos. Constata-se, então, que há um dilema entre velocidade – implementação de menor quantidade de nós para melhorar a pesquisa – e a realidade dos movimentos – uso de maior quantidade nós. Os problemas listados devem-se principalmente à introdução de objetos dinâmicos em mapas estáticos.

Enquanto o algoritmo A* geralmente é considerado como o melhor algoritmo de *pathfinding*, existe um algoritmo alternativo denominado algoritmo de Dijkstra. O algoritmo assemelha-se ao A*, porém não utiliza heurística – H é sempre nulo. Sua principal aplicação consiste no desconhecimento da localização do alvo ou destino, porque a procura é realizada se expandindo igualmente para fora em todas as direções, explorando uma área muito maior do mapa antes do objetivo ser encontrado. Utilizar o A* neste caso requer que ele seja aplicado repetidamente, a fim de achar todas as distâncias dos alvos, o que não é eficiente. Entretanto, o algoritmo de Dijkstra é considerado mais lento que o A*, logo deve ser utilizado na hipótese descrita.

6 Implementação

O Algoritmo A* foi implementado utilizando a linguagem de programação Java. Essa linguagem foi escolhida por dois principais motivos. Primeiro, é a linguagem na qual os componentes do grupo possuem maior conhecimento. Segundo, ela é multiplataforma, e assim facilitou o desenvolvimento do projeto, pois foi usado tanto em plataforma Windows quanto em Linux.

Para implementar o algoritmo foi necessário três estruturas de dados principais: uma para representar o tabuleiro (ou mapa), uma para guardar os nós que estão na lista aberta e outra para os nós que estão na lista fechada. Todas elas foram implementadas utilizando o tipo ArrayList de Java. Esse tipo permite que novos elementos sejam adicionados ou retirados sem a necessidade se mudar explicitamente o tamanho do vetor.

Para representar a lista aberta e fechada foi utilizado apenas um ArrayList para cada lista. Já para representar a estrutura do tabuleiro foi necessário criar um vetor dimensionável para representar as linhas, e em cada linha foi criado uma outra estrutura de dados, também do mesmo tipo, para representar as colunas. Assim, para controlar uma posiçao (x, y) do tabuleiro bastava invocar o método para obter a linha e, em seguida, obter a coluna a partir do método get. Esse ArrayList guardou outra estrutura de dados chamada de nó.

Um nó guarda sua posição (x, y) do tabuleiro; guarda se o nó é passável ou se é um obstáculo. Um nó também guarda as referências para seus nós vizinhos em um ArrayList de nós, essa estrutura é varrida na execução do algoritmo A*. Também guarda uma referência para o nó pai. É criada uma relação

pai-filho entre os nós à medida que o algoritmo é executado. Ela é importante para formar o caminho, bastando-se para isso, ao se encontrar o nó destino, ir recuperando quem é o pai do nó destino, quem é o pai do pai do nó destino e assim sucessivamente, até que se chegue a origem.

Existe dentro do nó um estado para origem, destino, nó passável ou obstáculo para representar cada posição do tabuleiro. Foi utilizado números inteiros para representar os estados. Zero significa passável, um significa obstáculo, dois significa origem e três significa destino. A estrutura de que representa o tabuleiro em forma de nós é formada se passando como argumento para uma função uma matriz de inteiros utilizado a codificação de inteiro 1, 2, 3 e 4.

Durante a fase de análise foi utilizada um tabuleiro de 40 linhas e 40 colunas por conveniência de tempo de processamento. Foi criado uma função para inserir a quantidade y de obstáculos à uma matriz qualquer. O algoritmo de análise insere passo a passo uma quantidade preestabelecida de obstáculos, executa o A*, grava os resultados e repete o processo até que não exista caminho possível na matriz gerada. Os resultados gravados são a quantidade de nós explorados para cada heurística.

Cada matriz gerada é gravada em uma planilha eletrônica, nomeada de “Dados”, no diretório raiz.. Também é gerado um arquivo, também como planilha eletrônica, que contém a quantidade de nós explorados quando se varia a quantidade de obstáculos para cada heurística analisada.

Além disso, foi criado um algoritmo para desenhar graficamente o estado da matriz para facilitar a análise do comportamento do A* na exploração dos nós e na compreensão do trajeto encontrado.

7 Análise e Discussão dos Resultados

Nessa seção será feita uma comparação entre o desempenho das heurísticas abordadas no trabalho quando é gerado obstáculos em posições aleatórias. Para tal propósito foi fixado o tamanho da matriz, a posição de origem e de destino para estudar apenas a influência da quantidade de obstáculos no desempenho das heurísticas.

O gráfico da figura 7 mostra a relação entre a quantidade de nós explorados para cada uma das três heurísticas. A partir da análise, conclui-se que a heurística que demostrou o melhor resultado de acordo com o gráfico foi a de Manhattan.

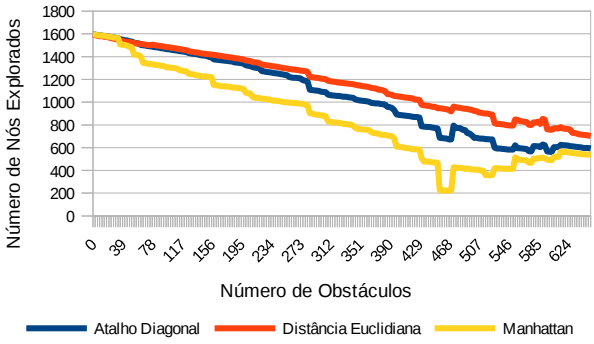


Figura 7. Gráfico comparando números de nós explorados versus a quantidade de obstáculos para cada heurística.

Observando o gráfico percebe-se que o número de nós explorados é quase igual ao número total de nós do tabuleiro que seria 1600 (40x40). Inicialmente foi questionado sobre a eficiência da implementação do A*, por causa da grande quantidade de nós explorados até para o tabuleiro sem nenhum obstáculo. A figura 8 representa a quantidade de nós explorados (azul claro) pelo Método Manhattan:

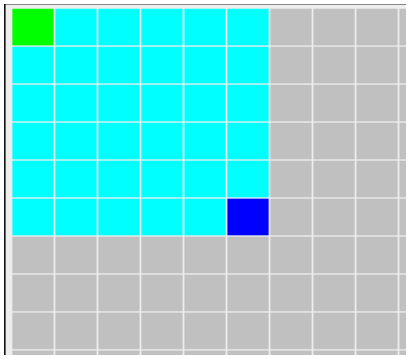


Figura 8. Configuração da exploração de nós utilizando o método Manhattan.

Revisando Patel A. (2014), se descobriu que o problema não está na implementação do A*. Segundo esse artigo, o problema pode ocorrer sempre que se tem vários caminhos com mesmo custo. E esse problema é acentuado pela diferença do deslocamento de um nó para outro nó adjacente ter sempre tem o mesmo custo. Por poder explorar vários caminhos de igual custo, o A* pode expandir nós desnecessários, pois o importante é apenas explorar os nós para se descobrir apenas um caminho de menor custo, e não todos os outros caminhos possíveis.

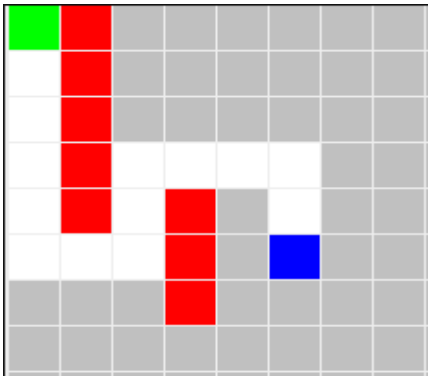
Calculando o valor F dos quadrados explorados na figura 8 descobriu-se que todos tinham valor igual a 100. Assim, pode acontecer do A* explorar uma grande quantidade de nós, como foi o caso.

Uma das soluções propostas por Patel A. (2014) para melhorar o desempenho do A* nesse caso é criar um critérios de desempate para os vários caminhos possíveis. Esse critério de desempate é um valor muito pequeno somado ao valor dos F’s que este-

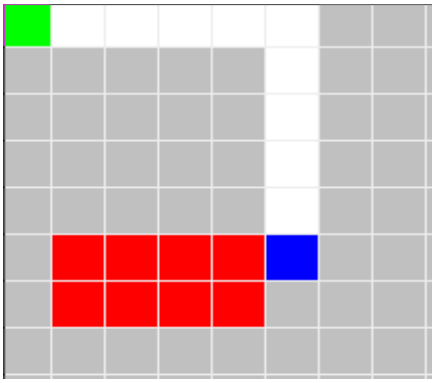
jam empatados para os fazerem diferir um dos outros. Para mais detalhes, consultar Patel A. (2014).

Vale destacar que esse empate não atrapalha no desempenho do A* em relação a encontrar o menor caminho, ele apenas pode levar o algoritmo a explorar uma quantidade de nós muito maior a necessária.

É difícil explicar o comportamento das heurísticas baseados em quantidades aleatórias de obstáculos, pois a dificuldade para se encontrar um trajeto possível e mínimo está diretamente relacionado com a posição dos blocos e não apenas com a quantidade. Para tanto, há um exemplo na figura 9.



(a)

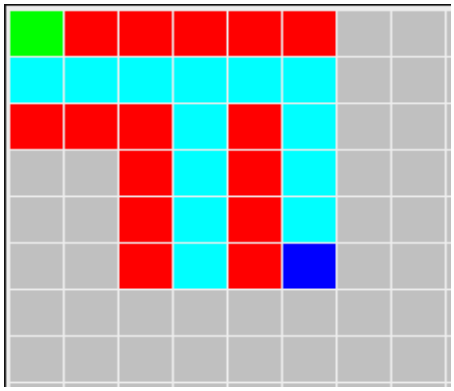


(b)

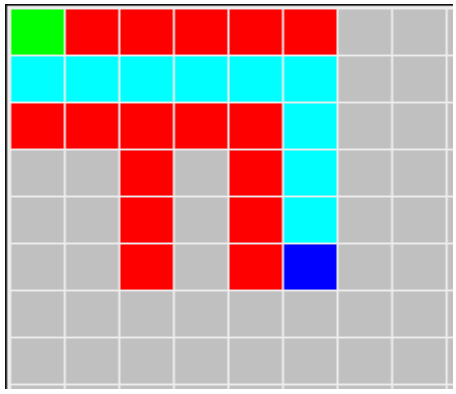
Figura 9. Mostra que a posição dos obstáculos pode causar mais efeito do que simplesmente a sua quantidade. Ambos casos, possuem 8 obstáculos, porém o trajeto (a) é maior que o trajeto (b) (de branco)

Claramente se pode notar que apesar de terem a mesma quantidade de blocos, o trajeto gerado em (b) é bem mais trabalhoso (longo) que o gerado em (a), logo se pode inferir que a posição dos blocos é fundamental para a análise. Apesar desse fato, notou-se alguns padrões de comportamento no gráfico.

A medida que subia incrementalmente a quantidade de obstáculos, reduziu-se a quantidade de nós explorados. Foram encontradas duas principais explicações para esse comportamento. Ao aumentar a quantidade de obstáculos, a quantidade de área que se pode explorar foi diminuída, assim a quantidade de nós explorados decresce.



(a)

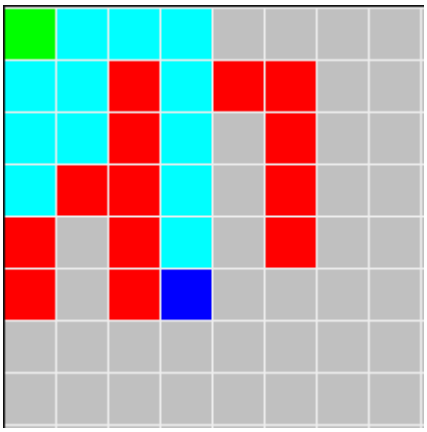


(b)

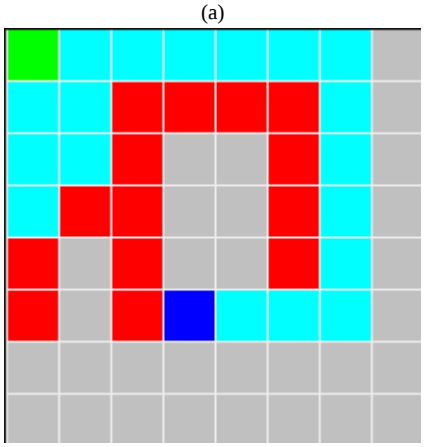
Figura 10. A área destacada expressa uma região similar a uma rua sem saída. Na figura (b) é inserido um novo obstáculo.

Na figura 10 (a), o mapa foi configurado com obstáculos semelhante a uma “rua sem saída” (parte da região azul claro da quarta coluna). Porém o algoritmo só vai descobrir que não se tem saída ao se explorar a área. Na figura (b) é inserido um novo obstáculo (quarta coluna, terceira linha). Com a nova configuração, o algoritmo não vai mais explorar aquela região sem saída, diminuindo assim a quantidade de blocos explorados.

E por fim, outro comportamento observado no gráfico da figura 7 foi o crescimento da quantidade de nós explorados. Uma explicação encontrada foi que inserindo mais obstáculos, tornou-se mais difícil chegar ao destino, forçando o algoritmo a explorar uma área maior para encontrar o caminho, desde modo cresce a quantidade de nós explorados. Ao se inserir um novo obstáculo na figura 11 (a) chegou-se a figura 11 (b). O novo obstáculo inserido fez o algoritmo explorar (de azul claro) uma área maior do que explorou em (a).



(a)



(b)

Figura 11. Com a obstrução de um trajeto possível na parte (a) o algoritmo é forçado a procurar por outra opção com trajeto mais longo.

8 Conclusão

Para o problema de pathfinding proposto, concluiu-se que, primeiramente, a utilização de Algoritmos Genéticos (AGs) não era necessária, uma vez que estes não fornecem o menor caminho possível. O custo da implementação dos AGs também foi uma limitação prevista, assim, visto que o algoritmo A* resolveria o pathfinding, não era necessário empregá-los neste tipo de problema.

Seguindo o mesmo raciocínio, agora para a escolha da heurística adequada, observou-se que, de fato, o Método Manhattan é ideal, embora apresentasse desempenho semelhante aos outros métodos analisados em questão de tempo de processamento, pois as matrizes utilizadas foram no máximo 40x40, assim não houve grande diferença em questão de processamento. No entanto, mostrou que é capaz de explorar menos nós para se encontrar um caminho, portanto fundamental para soluções para problemas complexos, como encontrado em problemas de navegação de robôs autônomos.

A partir do estudo do A*, pode-se apurar suas limitações e manipular o problema para que ele não utilize blocos dinâmicos, que causariam um baixo desempenho devido ao alto custo de processamento do algoritmo clássico, inviabilizando seu uso. A partir disso, foram verificadas suas condições de admissibilidade para que seu funcionamento encontre uma solução ótima, isto é, garantir a determinação do caminho de menor custo como Dijkstra, mas tentando explorar uma quantidade mínima de nós como a busca pela melhor escolha.

Para próximos trabalhos, poderia ser estudado uma estrutura eficiente para comportar mapas utilizados em jogos comerciais ou outros tipos de problema de se determinar o caminho que permitissem movimento em qualquer direção. E para tais problemas seria útil também implementar formas de diminuir a quantidade de nós explorados, se houvesse vários caminhos com mesmo custo. Outra possibilidade seria a implementação de situações em que o destino muda de posição ou até mesmo em casos em que se faça uso de técnicas de Inteligência Artificial para otimizar o desempenho do A*.

Referências Bibliográficas

Cain, T. (2002), Practical Optimizations for A*, AI Game Programming Wisdom, Charles River Media.

Lester, P (2004). Pathfinding for Beginnners. Disponível em: <http://www.policyalmanac.org/games/aStarTutorial_port.htm>. Acesso em: 17/04/2014.

Patel A. (2014). Amit's A* Pages: Pathfinding. Disponível em: <http://theory.stanford.edu/~amitp/GameProgramming/>. Acesso em: 24/04/2014.

Stentz, A. (1994). Optimal and Efficient Path Planning for Partiallyknown Environments. In proceedings of the IEEE International Conference on Robotics and Automation.

Russel, S; Norvig, P (2004). Inteligência Artificial, 2º edição. Editora Campus.

Wenderlich, R. Introduction to A* Pathfinding. Disponível em: <*http://www.raywenderlich.com/4946/introduction-to-a-pathfinding*>. Acesso em: 23/04/2014.