# Relatório

# do

# primeiro trabalho

# Integrantes

|  |  |
| --- | --- |
| Matrícula | Nome |
| 112031005 | André Alvarado |
| 212031119 | Erick Valadares |
| 212031132 | Daniel Derlando |
| 212031098 | Lucas Tito |
| 113031049 | Matheus Filipe |

# Sumário

1 Introdução

2 API e tecnologias

2.1 Instruções

3 Métodos de busca

3.1 Busca em largura

3.2 Busca A\*

3.3 Busca Gulosa

4 Heurísticas

4.1 HeuristicRoute

4.2 HeuristicWithRadar

5 Resultados

6 Conclusão e observações

# 1 Introdução

O trabalho 1 da disciplina de Inteligência Artificial, lecionada pela professora Aline Paes no primeiro semestre de 2015 requer que seja criada uma aplicação, cujo dado dois pontos (inicial, final), encontre o menor caminho entre eles, em um ambiente de mapa da cidade de Niteroi, Rio de Janeiro, Brasil.

A aplicação se valerá de métodos de busca descritos ao decorrer deste documento. Tais métodos foram escolhidos pelos alunos da disciplina envolvidos neste trabalho.

# 2 API e tecnologias

Linguagem: Java (JDK 8 update 20)

IDE: Netbeans 8.0.1 Full

API de mapa: Google Static Maps

Componente gráfico: Google Maps Java Script API V 3.0

Interface de comunicação de dados: Java Server Page (JSP)

# 2.1 Instruções

1 - Importar o projeto para o Netbeans

2 - Executar o projeto.

3 – buscar no google maps um ponto desejado e copiar sua latitude e longitude.

4 – escrever a latitude e a longitude de um ponto, apartir do passo 3 (fazer isso para o ponto de origem e o ponto destino)

5 – escolher o tipo de método de busca

6 – aguardar o programa carregar o caminho, caso queira fazer uma nova execução basta clicar no botão “voltar” e repetir o passo 3 deste fluxo.

# 3 Métodos de busca

# 3.1 Busca em largura

O método de busca em largura foi representado pelo método java de nome executaBFS.

# 3.2 busca A\*

O método de busca A\* requer uma heurística. Como manda o enunciado do trabalho, foram implementadas duas heurísticas, HeuristicRoute e HeuristicWithRadar, cada uma delas possui um método, executaAStarRoute e executaAStarRadar, respectivamente. As Heurísticas serão melhor definidas na seção 4 deste trabalho.

# 3.3 Busca Gulosa

O método de busca gulosa requer uma heurística. Como manda o enunciado do trabalho, foram implementadas duas heurísticas, HeuristicRoute e HeuristicWithRadar, cada uma delas possui um método, executaGreedyRouteS e executaGreedyRadarS, respectivamente. As Heurísticas serão melhor definidas na seção 4 deste trabalho.

# 4 Heurísticas

# 4.1 HeuristicRoute

Esta heurística utiliza a distância entre o nó atual e o nó destino, atribuindo à heurística do nó atual o valor resultante da fórmula de distância.

# 4.2 HeuristicWithRadar

Esta heurística verifica se o nó atual está em uma lista de ruas da cidade de Niterói que possuem radar. Estas ruas, por possuírem radar, limita a velocidade de tráfego, aumentando a probabilidade da rua possuir um fluxo lento. Se o nó estiver na rua, é acrescida a heurística do nó a HeuristicRoute e a distância máxima entre o ponto inicial e o ponto destino, caso contrário será atribuída a heurística do nó o valor da HeuristicRoute. No primeiro caso o valor seria superior a heurística dos demais nós, o que faria com que ele não fosse escolhido e sim os nós que tem como heurística somente a distância.

# 5 Resultados

|  |  |  |  |
| --- | --- | --- | --- |
| Método | C1 | C2 | C3 |
| Busca em largura | tn(461) n(26) + | tn(5866) n(114) + | tn(679) n(32) + |
| A\* com HeuristicRoute | tn(73) n(27) + | tn(647) n(115) + | tn(72) n(32) + |
| A\* com HeuristicWithRadar | tn(65) n(35) + | tn(647) n(115) + | tn(94) n(32) + |
| GreedySearch com HeuristicRoute | tn(54) n(45) + | tn(186) n(131) + | tn(58) n(32) + |
| GreedySearch com HeuristicWithRadar | Tn(26457) n(35) + | tn(187) n(131) + | tn(113) n(56) + |

Legenda:

C1: ponto de partida(-22.953147, -43.083200) e ponto destino(-22.952984, -43.084578) [possui 2 radares no caminho e uma rua alternativa sem radar, algoritmo escolhe rua alternativa]

C2: ponto de partida(-22.938878, -43.071167) e ponto destino(-22.930412, -43.073323) [não possui radares no caminho]

C3: ponto de partida(-22.952615, -43.089559) e ponto destino(-22.952388, -43.090959) [possui 2 radares e sem rua alternativa sem radar, o algoritmo escolhe rua com radar de menor custo]

N: quantidade de nós do caminho mínimo

tn: quantidade total de nós visitados

\*: ponto destino não encontrado, mas a solução é aproximada

X: não encontrou resultado satisfatório

+: encontrou solução e a solução é a esperada

# 6 Conclusão e Observações

A busca em largura, como esperado é o método que mais utiliza nós e como nossa aplicação realiza solicitações a internet, o método é o mais demorado, a não ser, que as informações fiquem guardadas em cache. Os métodos A\* e guloso são mais eficientes, dado que percorrem menos nós, a diferença entre eles será o caminho escolhido.

Nosso grupo tentou de início implementar o simulated anilling, porém não obtivemos sucesso na definição dos estados, o código não foi apagado do projeto, entretanto para questão de nota e tempo, preferimos implementar o algoritmo guloso.

Uma observação importante é que nossa aplicação é genérica o suficiente para aceitar quais quer dois pontos, independentemente de estarem na cidade de Niteroi ou não, porém quanto maior a distância, maior será o processamento de nós.