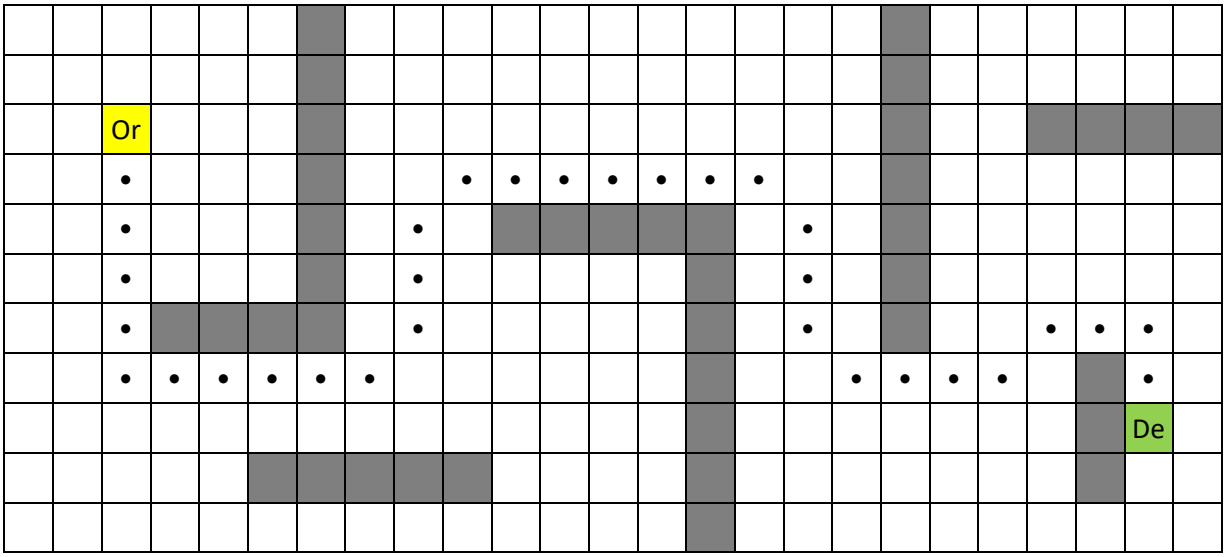
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

PLANEJADOR DE CAMINHOS EM LABIRINTOS

PROFESSOR: ADELARDO ADELINO DANTAS DE MEDEIROS



O objetivo é desenvolver em C++ um programa para determinar o caminho mais curto entre células de origem e destino, dentro de um ambiente descrito por um mapa com obstáculos, utilizando o algoritmo A* e as estruturas de dados da biblioteca STL de C++.

ALGORITMO A*

O algoritmo A* encontra o caminho de menor custo em um grafo no qual a transição de cada nó do grafo para outro nó ao qual ele esteja conectado tem um custo associado. No exemplo do labirinto, cada célula do mapa é um nó do grafo. As células estão conectadas às 8 células vizinhas e o custo de ir de uma célula para outra é a distância entre os centros das células:

- 1, para movimento horizontal ou vertical; e
- $\sqrt{2}$, se o movimento for em diagonal.

O A* mantém um conjunto dos nós já visitados (***Fechado***) e um conjunto dos nós ainda não analisados (***Aberto***). No início, ***Fechado*** está vazio e ***Aberto*** contém apenas o nó de origem.

A cada passo, o A* retira um nó de ***Aberto***, coloca em ***Fechado***, verifica se ele é o destino e, se não for, gera até 8 sucessores, que correspondem às possíveis direções de movimentação. Os sucessores válidos são colocados em ***Aberto***. O algoritmo prossegue até que o destino seja alcançado.

Cada nó tem um custo associado, que é o tamanho do caminho percorrido da origem até ele. Esse custo é denominado de custo passado ***(g)***. Ele é igual ao custo passado do seu antecessor mais o custo da movimentação do antecessor até ele (no caso do labirinto, 1 ou $\sqrt{2}$).

$$g(n_k) = g(n_{k-1}) + \text{custo}(n_{k-1}, n_k)$$

Para garantir que o caminho mais curto seja encontrado, o nó retirado de ***Aberto*** deve ser sempre o de menor custo. Por essa razão, normalmente os nós em ***Aberto*** são mantidos ordenados em ordem crescente de custo e retira-se sempre o primeiro.

Para cada célula do mapa, só pode haver um nó armazenado em ***Aberto*** ou ***Fechado***. Quando um sucessor é gerado, verifica-se se um nó que representa a mesma célula já não existe em ***Aberto*** ou em ***Fechado***. Caso exista, significa que foi encontrado outro caminho para chegar ao mesmo nó. Nesse caso, deve ser mantido apenas o caminho de menor custo:

- Caso o sucessor tenha custo maior que o nó existente, ignora-se o novo sucessor.
- Caso o sucessor tenha custo menor que um nó em ***Fechado***, exclui-se o nó de ***Fechado*** e coloca-se o sucessor em ***Aberto***.
- Caso o sucessor tenha custo menor que um nó em ***Aberto***, exclui-se o nó de ***Aberto*** e coloca-se o sucessor em ***Aberto***.

Ordenando os nós apenas pelo custo passado, o algoritmo A* se torna equivalente ao algoritmo de Dijkstra, que se assemelha a uma busca em largura: são analisados primeiro todos os vizinhos da origem, depois todos os vizinhos dos vizinhos e assim sucessivamente. Isso garante que o caminho mais curto será encontrado primeiro, mas pode ser lento.

Para acelerar a busca, o algoritmo A* ordena os nós pelo custo total ***(f)***, que é a soma do custo passado ***(g)*** com o custo futuro ***(h)***. O custo futuro é baseado em uma estimativa (heurística). O caminho mais curto será encontrado se a heurística ***h*** for admissível, isto é, se o seu valor for sempre menor ou igual do que o custo real para mover do nó até o destino.

```
// Tipo de dado Noh
Noh:
  pos: célula atual (posição)
  ant: célula anterior (antecessor)
  g: custo passado
  h: custo futuro

// Cria o noh inicial
atual.pos ← origem
atual.ant ← void()
atual.g ← 0.0
atual.h ← heurística()

// Inicializa o conjunto Aberto
inserir(atual,Aberto)

// Iteração: repita enquanto houver
// nohs em Aberto e ainda não
// houver encontrado a solução
encontrou_solucao ← false
Repita
| atual ← menor_custo_tot(Aberto)
| inserir(atual,Fechado)
| // Testa se é solução
| Se ( é_destino_final(atual) )
| | encontrou_solucao ← true
| Caso contrário
| | // Gera sucessores de atual
| | Para dir em L,NE,N,NO,O,SO,S,SE
| | | // Testa se pode mover de
| | | // atual na direção dir
| | | Se ( mapa.válido(atual+dir) )
| | | | // Gera novo sucessor:
| | | | suc.pos ← atual.pos+dir
| | | | suc.ant ← atual.pos
| | | | suc.g ← atual.g+custo(dir)
| | | | suc.h ← heurística()
| | | | // Procura suc em Fechado
| | | | old ← procura(suc,Fechado)
| | | | Se (existe(old))
| | | | | // Testa qual o melhor
| | | | | Se (suc < old)
| | | | | | remove(old,Fechado)
| | | | | Caso contrário
| | | | | | descarta(suc)
| | | | Fim Se
| | | | // Procura suc em Aberto
| | | | old ← procura(suc,Aberto)
| | | | Se (existe(old))
| | | | | // Testa qual o melhor
| | | | | Se (suc < old)
| | | | | | remove(old,Aberto)
| | | | | Caso contrário
| | | | | | descarta(suc)
| | | | Fim Se
| | | | // Insere suc em Aberto
| | | | Se não_descartado(suc)
| | | | | inserir(suc,Aberto)
| | | | Fim Se
| | | Fim Se
| | Fim Para
| Fim Se
Enquanto ( Não(encontrou_solucao) E
           Não(vazio(Aberto)) )

// Imprime estado final da busca
exibir(tamanho(Fechado))
exibir(tamanho(Aberto))
// Pode ter terminado porque
// encontrou a solução ou porque
// não há mais nohs a testar
Se ( Não(encontrou_solucao) )
| exibir("Não existe caminho")
Caso contrário
| | Enquanto (atual.ant != void())
| | | exibir(atual)
| | | atual ← procura(atual.ant,
| | |         Fechado)
| | Fim Enquanto
Fim Se
```

POSSÍVEIS HEURÍSTICAS

- Distância Manhattan: usada quando só se move nas 4 direções principais

$$h = \Delta x + \Delta y$$

- Distância diagonal: usada quando se move nas 8 direções vizinhas

$$h = \sqrt{2} \cdot \min(\Delta x, \Delta y) + \text{abs}(\Delta x - \Delta y)$$

- Distância Euclidiana: usada quando todos os movimentos são possíveis

$$h = \sqrt{\Delta x^2 + \Delta y^2}$$