

Uma imagem com alimentação

Descrição gerada automaticamente

**Relatório de ALGAV**

**3DD \_ Grupo 100**

1191256 André Reis

1191362 Rúben Amorim

1170878 Milene Farias

**Data: 26/11/2023**

# 1 - Representação do conhecimento do domínio

# Base de conhecimento

|  |  |
| --- | --- |
| liga(a,b). | Serve para identificar que existe uma ligação entre 2 edifícios. Os parâmetros são os edifícios. |
| pisos(a,[a1,a2]). | Serve para identificar os pisos de um edifício. O primeiro parâmetro é o edifício específico e o segundo parâmetro uma lista com os pisos. |
| dimensoes(a1, 23, 11). | Serve para indicar as dimensões de cada piso. O primeiro parâmetro é o piso, os outros 2 parâmetros são as dimensões do andar. |
| elevador(a,[a1,a2]). | Serve para identificar os andares que o elevador pode percorrer. O primeiro parâmetro é o elevador em específico e o segundo parâmetro uma lista com os pisos. |
| passagem(a,b,a2,b2). | Serve para indicar a passagem entre 2 edifícios. Os 2 primeiros parâmetros servem para indicar os edifícios da ligação. Os 2 últimos parâmetros são as os pisos que irão ser ligados. |
| sala(apn, a1). | Serve para indicar uma sala. O primeiro parâmetro é o nome da sala e o 2 parâmetro o piso da mesma. |
| coordenadas(apn, a1, 10, 6). | Serve para indicar as coordenadas das portas de cada sala. O primeiro parâmetro indica o nome da sala, o segundo parâmetro o andar da sala e os 2 últimos parâmetros a localização da porta da sala no piso. |
| m(a1,0,0,3). | Serve para indicar o mapa do andar com todas as suas posições. O primeiro parâmetro é o andar, o segundo parâmetro a linha em que se encontra no array de posições, o terceiro parâmetro é a coluna e o quarto parâmetro o valor para dizer se é uma parede, espaço vazio, etc (0 ,1 ,2 ,3) |

# Alterações efetuadas nos algoritmos fornecidos

As alterações efetuadas nos algoritmos fornecidos foram as seguintes:

* Todos os algoritmos foram alterados para incluírem o piso;
* Na criação do graph foi alterado o código para incluir o peso das ligações das células;
* Durante a criação dos graphs, se as células estiverem na vertical ou na horizontal entre si, o peso é de 1 e quando são diagonais é de ;

# 2 - Obtenção da solução ótima para o Planeamento de movimentação entre pisos de edifícios movendo-se por corredores de ligação entre edifícios e elevadores (deve indicar pontos de partida e de chegada que devem ser pontos de acesso a salas/gabinetes/elevadores/corredores externos).

Este é o predicado inicial que irá obter o melhor caminho entre dois pisos.

melhor\_caminho\_pisos(PisoOr,PisoDest,LLigMelhor)

Algoritmo começa por encontrar todos os caminhos entre os dois pisos.

findall(LLig,caminho\_pisos(PisoOr,PisoDest,\_,LLig),LLLig),

O algoritmo invoca este predicado e utiliza os caminhos encontrados pelo predicado anterior, em conjunto com o predicado "conta", para encontrar o percurso com menos elevadores. Se houver igualdade, é considerada a menor utilização de passagens como critério de desempate.

menos\_elevadores(LLLig,LLigMelhor,\_,\_).

menos\_elevadores([LLig|OutrosLLig],LLigR,NElevR,NPassR):-

menos\_elevadores(OutrosLLig,LLigM,NElev,NPass),

conta(LLig,NElev1,NPass1),

(((NElev1<NElev;(NElev1==NElev,NPass1<NPass)),!,NElevR is NElev1, NPassR is NPass1,LLigR=LLig);

(NElevR is NElev,NPassR is NPass,LLigR=LLigM)).

# 3 - Movimentação do robot dentro de um piso de edifício com Primeiro em Profundidade;

Recebe o piso (Piso), o ponto de origem (Orig) e o ponto de destino (Dest). O piso é utilizado no “ligacel” para ir buscar as ligações entre as células desse piso. O algoritmo começa na posição inicial e visita os pontos adjacentes até encontrar o destino, mantendo uma lista dos pontos visitados para evitar ciclos. Quando alcança o destino, o algoritmo devolve o caminho percorrido.

dfs(Piso,Orig,Dest,Cam):-

dfs2(Piso,Orig,Dest,[Orig],Cam).

dfs2(\_,Dest,Dest,LA,Cam):-

reverse(LA,Cam).

dfs2(Piso,Act,Dest,LA,Cam):-

(ligacel(Piso,Act,X,\_); ligacel(Piso,X,Act,\_)),

\+ member(X,LA),

dfs2(Piso,X,Dest,[X|LA],Cam).

# 4 - Geração de todas as soluções do Primeiro em Profundidade e escolha da melhor;

O better\_dfs usa o all\_dfs para encontrar todos os caminhos entre dois pontos do grafo. Depois, utiliza o shortlist para escolher o caminho mais curto entre esses caminhos, considerando o número de elementos em cada um. O shortlist compara os tamanhos dos caminhos encontrados e devolve o caminho mais curto como resultado.

better\_dfs(Piso,Orig,Dest,Cam):-

all\_dfs(Piso,Orig,Dest,LCam), shortlist(LCam,Cam,\_).

shortlist([L],L,N):-!,length(L,N).

shortlist([L|LL],Lm,Nm):-

shortlist(LL,Lm1,Nm1), length(L,NL),

((NL<Nm1,!,Lm=L,Nm is NL); (Lm=Lm1,Nm is Nm1)).

# 5 - Primeiro em Largura

O algoritmo de pesquisa em largura procura um caminho entre dois pontos do grafo. Ao começar na posição inicial, o algoritmo explora os caminhos disponíveis em níveis, expandindo-se gradualmente. Quando atinge o destino, devolve o caminho percorrido até lá.

bfs(Piso,Orig,Dest,Cam):-bfs2(Piso,Dest,[[Orig]],Cam).

bfs2(\_,Dest,[[Dest|T]|\_],Cam):-

reverse([Dest|T],Cam).

bfs2(Piso,Dest,[LA|Outros],Cam):-

LA=[Act|\_],

findall([X|LA],

(Dest\==Act,(ligacel(Piso,Act,X,\_);ligacel(Piso,X,Act,\_)),

\+ member(X,LA)),

Novos),

append(Outros,Novos,Todos),

bfs2(Piso,Dest,Todos,Cam).

# 6 - A\*

Este algoritmo encontra o caminho mais curto entre dois pontos do grafo. O algoritmo usa uma estimativa heurística para explorar os caminhos de forma eficiente, dando prioridade aos que têm menor custo estimado. Quando alcança o destino, devolve o caminho percorrido. O cálculo da estimativa é a distancia entre as duas células.

% calcular a distância euclidiana entre duas células

estimativa(cel(X1, Y1), cel(X2, Y2), Distancia) :-

Distancia is sqrt((X1 - X2)^2 + (Y1 - Y2)^2).

% predicado principal do A\*

aStar(Orig, Dest, Cam, Custo, Piso) :-

aStar2(Piso, Dest, [(\_, 0, [Orig])], Cam, Custo).

% predicado auxiliar para o A\*

aStar2(\_, Dest, [(\_, Custo, [Dest|T])|\_], Cam, Custo) :-

reverse([Dest|T], Cam).

aStar2(Piso, Dest, [(\_, Ca, LA)|Outros], Cam, Custo) :-

LA = [Act|\_],

findall((CEX, CaX, [X|LA]),

(Dest \== Act,

(ligacel(Piso, Act, X,CustoX);ligacel(Piso, X,

Act,CustoX)), \+ member(X, LA),

CaX is CustoX + Ca, estimativa(X, Dest, EstX),

CEX is CaX + EstX),

Novos),

append(Outros, Novos, Todos),

% write('Novos='),write(Novos),nl,

sort(Todos, TodosOrd),

% write('TodosOrd='),write(TodosOrd),nl,

aStar2(Piso, Dest, TodosOrd, Cam, Custo).

# 7 - Consideração de movimentos nas diagonais

Para serem formadas ligações nas diagonais ele tem em consideração o seguinte:

* Diagonal noroeste, verifica se a célula acima e a célula à esquerda estão livres;
* Diagonal nordeste, verifica se a célula acima e a célula à direita estão livres;
* Diagonal sudoeste, verifica se a célula abaixo e a célula à esquerda estão livres;
* Diagonal sudeste, verifica se a célula abaixo e a célula à direita estão livres;

cria\_grafo\_lin(Piso,Col,Lin):-

m(Piso,Lin,Col,0),!,

ColS is Col+1, ColA is Col-1,

LinS is Lin+1,LinA is Lin-1,

((m(Piso,Lin,ColS,0),assertz(ligacel(Piso,cel(Col,Lin),cel(ColS,Lin),**1**))

;true)),

((m(Piso,Lin,ColA,0),assertz(ligacel(Piso,cel(Col,Lin),cel(ColA,Lin),**1**))

;true)),

((m(Piso,LinS,Col,0),assertz(ligacel(Piso,cel(Col,Lin),cel(Col,LinS),**1**))

;true)),

((m(Piso,LinA,Col,0),assertz(ligacel(Piso,cel(Col,Lin),cel(Col,LinA),**1**))

;true)),

((m(Piso,LinS,ColS,0), **m(Piso,LinS,Col,0)**, **m(Piso,Lin,ColS,0)**, assertz(ligacel(Piso,cel(Col,Lin),cel(ColS,LinS),**sqrt(2)**));true)),

((m(Piso,LinA,ColA,0), **m(Piso,LinA,Col,0)**, **m(Piso,Lin,ColA,0)**, assertz(ligacel(Piso,cel(Col,Lin),cel(ColA,LinA),**sqrt(2)**));true)),

((m(Piso,LinA,ColS,0), **m(Piso,LinA,Col,0)**, **m(Piso,Lin,ColS,0)**, assertz(ligacel(Piso,cel(Col,Lin),cel(ColS,LinA),**sqrt(2)**));true)),

((m(Piso,LinS,ColA,0), **m(Piso,Lin,ColA,0)**, **m(Piso,LinS,Col,0)**, assertz(ligacel(Piso,cel(Col,Lin),cel(ColA,LinS),**sqrt(2)**));true)),

Col1 is Col-1,

cria\_grafo\_lin(Piso,Col1,Lin).

# 8 - Integração do ponto 2 com o ponto 3

Este predicado é responsável por integrar o ponto 2 com o ponto 3. Ele recebe o algoritmo que depois irá utilizar para calcular os movimentos do robô nos diferentes pisos. Recebe o ponto de acesso de origem (sala/gabinete, elevador ou passagem) (ElementoOr) e recebe o ponto de acesso de destino (sala/gabinete, elevador ou passagem) (ElementoDest).

O algoritmo começa por determinar o tipo da origem, o tipo do destino e consoante isso obtêm o piso de cada um. Depois ele encontra o melhor caminho entre os dois pisos e guarda numa lista (Caminho). De seguida, adiciona o ponto de origem e ponto de destino a lista. Por fim, ele processa a lista do caminho obtido.

find\_caminho\_entidades(Algorith, ElementoOr, ElementoDest, CaminhoCompleto2, Movimentos, CustoTot) :-

determinar\_tipo\_entidade(ElementoOr, PisoOr),

determinar\_tipo\_entidade(ElementoDest, PisoDest),

find\_caminho(PisoOr, PisoDest, Caminho),

append([ElementoOr|Caminho], [ElementoDest], CaminhoCompleto),

remove\_consecutive\_duplicates(CaminhoCompleto, CaminhoCompleto2),

CustoTot = 0,

processar\_caminho(Algorith,CaminhoCompleto2,Movimentos, CustoTot).

Este predicado percorre a lista dada pelo predicado anterior e percorre-a de forma a “criar” uma “corrente”, por exemplo, para a seguinte lista [a,b,c,d] ele processará da seguinte maneira: a,b; b,c; c,d

Desta forma, permite com que seja calculado os movimentos do robô entre os diferentes pontos de acesso.

processar\_caminho(Algorith, [Elemento1, Elemento2 | Resto], [Cam|CamResto], CustoTot) :-!,

processar\_elementos(Algorith, Elemento1, Elemento2, Cam, Custo),

CustoTotNew is CustoTot + Custo,

processar\_caminho(Algorith, [Elemento2 | Resto], CamResto, CustoTotNew).

**Exemplo com algoritmo A\* entre duas salas do mesmo piso:**

find\_caminho\_entidades(astar,sala(apn),sala(beng), L,Cam, Custo).

L = [sala(apn), sala(beng)],

Cam = [[cel(11, 6), cel(11, 5), cel(11, 4), cel(12, 3)]|\_],

Custo = 3.414213562373095

# Conclusões

Ao considerarmos a aplicação de algoritmos de procura em grafos para determinar o melhor percurso entre pontos de acesso em diferentes pisos de edifícios, conseguimos abordar eficazmente a otimização das rotas de movimentação. A implementação de métodos como o A\*, pesquisa em largura e em profundidade, bem como a consideração de movimentos diagonais, revelou-se crucial para alcançar resultados eficientes.

A integração destes algoritmos permitiu uma solução robusta e eficiente para a deslocação do robô entre diferentes áreas dentro dos edifícios, tendo em conta as restrições de acessibilidade e utilizando trajetórias ótimas para minimizar o tempo e esforço necessários.

Esta abordagem proporcionou a construção de uma base sólida para futuras iterações e aperfeiçoamentos no sistema, especialmente na expansão para casos mais complexos, oferecendo um método versátil e adaptável para a movimentação inteligente dentro dos ambientes representados pelos edifícios.