Inteligência Artificial (1001336)

Prof. Dr. Murilo Naldi

Agente Bombeiro – Trabalho 1

Grupo :

- Guilherme Locca Salomão (758569)
- João Victor Mendes Freire (758943)
- Luís Felipe Corrêa Ortolan (759375)

Modelagem do Estado e Ambiente

Primeiramente, foram separados os objetos mutáveis dos imutáveis, de forma que os mutáveis fizessem parte do estado, e os imutáveis do ambiente.

Assim, foi definido o estado como uma lista com as informações do bombeiro, os lugares onde existem Extintores, e os lugares onde existem Incêndios.

`Estado = [Bombeiro, Extintores, Incêndios]`

`Estado = [ [X, Y, Cargas], [[`X_1`, `Y_1`]...[`X_n`, `Y_n`]], [[`X_1`, `Y_1`]...[`X_m`, `Y_m`]] ]`

No exemplo de estado acima, **Cargas** diz respeito ao número de cargas que o bombeiro possui em dado momento, **n** é o número de extintores e **m** é o número de incêndios. Cada par `[`X_i`, `Y_i`]` representa a posição do extintor ou do incêndio **i**. X e Y indicam a posição do agente Bombeiro.

Com o estado definido, passou-se a modelar os objetos imutáveis, ou seja, as dimensões do prédio, paredes, entulhos e escadas. As regras dessa parte estão situadas em arquivos diferentes do das regras gerais, nos arquivos ambX.pl, onde X é o número do ambiente.

A seguinte regra, `dentro_predio()`, é verdadeira quando uma coordenada (X,Y) (representada na forma de uma lista em prolog) está dentro do prédio do problema, ou seja, não ultrapassa os limites das paredes. No exemplo abaixo, o prédio tem tamanho 10x5, portanto é verdadeira se $X \in [1, 10]$ e $Y \in [1, 5]$.

```
% Definindo tamanho do prédio 10x5
dentro_predio([X, Y|_]) :- X > 0, Y > 0, X < 11, Y < 6.
```

A próxima regra, `ocupado_com()`, é usada para indicar o que certa coordenada contém dentre as constantes `parede`, `escada` ou `entulho`. Por exemplo:

```
% Mapeando parede no ambiente
ocupado_com([3, 3], parede).

% Mapeando entulho no ambiente
ocupado_com([4, 2], entulho).

% Mapeando escada[1] no ambiente
ocupado_com([5, 1], escada).
```

Se não existe uma regra `ocupado_com()` para dada coordenada, e essa coordenada não existe na lista de incêndios nem na de extintores, então a posição é considerada vazia.

[1]É apenas a parte de baixo, para descer é preciso verificar se existe uma escada em `[X, Y-1]`).

Transições de Estado

Todas as funções mencionadas daqui para frente estão presentes no arquivo `projeto.pl`, que não depende de qual ambiente o problema está situado (mas precisa ter incluído um arquivo `ambX.pl` antes para executar a busca corretamente).

Foram identificadas 5 ações que alteram o estado do Bombeiro: movimento horizontal, saltos, movimento vertical, apagar incêndio e pegar extintor.

Movimento Horizontal acontece se o Bombeiro continua dentro do prédio, não existe uma parede, entulho ou fogo na posição desejada. Ele pode acessar uma posição que contém extintor e escada. Acabou-se dividindo as regras em movimento horizontal à esquerda e à direita. Apenas o X do bombeiro é atualizado nessa transição (para X1).

```
% Movimento horizontal à direita
s([[X, Y|Carga], Extintores, Incendios],
 [[X1, Y|Carga], Extintores, Incendios]) :-
    X1 is X + 1,
    dentro_predio([X1, Y]),
    not(ocupado_com([X1, Y], parede)),
    not(ocupado_com([X1, Y], entulho)),
    not(pertence([X1, Y], Incendios)).

% Movimento horizontal à esquerda
s([[X, Y|Carga], Extintores, Incendios],
 [[X1, Y|Carga], Extintores, Incendios]) :-
    X1 is X - 1,
    dentro_predio([X1, Y]),
    not(ocupado_com([X1, Y], parede)),
    not(ocupado_com([X1, Y], entulho)),
    not(pertence([X1, Y], Incendios)).
```

Saltos acontecem se o Bombeiro continua dentro do prédio, existe entulho na posição adjacente (X2), não existe nenhum objeto (paredes, entulhos, escadas, incêndios e extintores) na posição destino (X1), adjacente ao entulho.

```
% Salto à direita
s([[X, Y|Carga], Extintores, Incendios], [[X1, Y|Carga], Extintores,
Incendios]) :-
    X1 is X + 2, X2 is X + 1, Y1 is Y - 1,
    dentro_predio([X1, Y]),
    ocupado_com([X2, Y], entulho),
    not(ocupado_com([X1, Y], _)),
    not(pertence([X1, Y], Incendios)),
    not(pertence([X1, Y], Extintores)),
    not(ocupado_com([X1, Y1], escada)).
```

```
% Salto à esquerda
s([[X, Y|Carga], Extintores, Incendios], [[X1, Y|Carga], Extintores,
Incendios]) :-
    X1 is X - 2, X2 is X - 1, Y1 is Y - 1,
    dentro_predio([X1, Y]),
    ocupado_com([X2, Y], entulho),
    not(ocupado_com([X1, Y], _)),
    not(pertence([X1, Y], Incendios)),
    not(pertence([X1, Y], Extintores)),
    not(ocupado_com([X1, Y1], escada)).
```

Essas regras de transição alteram o X do bombeiro para X1, que é X ± 2 (dependendo da direção do pulo), e usam X2 para verificar a presença do entulho.

Movimento Vertical acontece se o Bombeiro continua dentro do prédio, existe uma escada na posição (em [X, Y] caso vá subir e em [X, Y-1] caso vá descer). Acabou-se dividindo as regras em movimento vertical para cima e para baixo. Apenas o Y do bombeiro é atualizado nessa transição (para Y1).

```
% Movimento vertical para cima
s([[X, Y|Carga], Extintores, Incendios],
 [[X, Y1|Carga], Extintores, Incendios]) :-
    Y1 is Y + 1,
    dentro_predio([X, Y1]),
    ocupado_com([X, Y], escada).

% Movimento vertical para baixo
s([[X, Y|Carga], Extintores, Incendios],
 [[X, Y1|Carga], Extintores, Incendios]) :-
    Y1 is Y - 1,
    dentro_predio([X, Y1]),
    ocupado_com([X, Y1], escada).
```

O bombeiro **Pega Extintor** se a carga está vazia (= 0) e existe um extintor na posição atual do Bombeiro. Então é preciso remover aquele extintor pego da lista de extintores e alterar a carga para 2.

```
% Pega extintor
s([[X, Y, Carga|Cauda], Extintores,  Incendios],
  [[X, Y,Carga1|Cauda], Extintores1, Incendios]) :-
    Carga == 0,
    pertence([X, Y], Extintores),
    retirar_elemento([X, Y], Extintores, Extintores1),
    Carga1 is Carga + 2.
```

Apaga Incêndio acontece se o Bombeiro tem carga (> 0), e existe fogo na posição adjacente à do Bombeiro ([X1, Y], onde X1 é X ± 1). Se houver, é necessário remover àquela posição da lista de incêndios e decrementar a carga atual.

```
% Apaga Incêndio à direita
s([[X, Y, Carga|Cauda], Extintores, Incendios],
 [[X, Y, Carga1|Cauda], Extintores, Incendios1]) :-
    Carga > 0, X1 is X + 1,
    pertence([X1, Y], Incendios),
    retirar_elemento([X1, Y], Incendios, Incendios1),
    Carga1 is Carga - 1.

% Apaga Incêndio à esquerda
s([[X, Y, Carga|Cauda], Extintores, Incendios],
 [[X, Y, Carga1|Cauda], Extintores, Incendios1]) :-
    Carga > 0, X1 is X - 1,
    pertence([X1, Y], Incendios),
    retirar_elemento([X1, Y], Incendios, Incendios1),
    Carga1 is Carga - 1.
```

Por fim, definiu-se o estado **`meta()`**, que no caso do problema é qualquer estado no qual a lista de incêndios seja vazia ([ ] em prolog).

```
% Definindo meta(Estado)
% Estado é meta se lista de incendios == []
meta([_, _, []]).
```

Funções Auxiliares

Foram criadas algumas funções auxiliares para facilitar a manipulação de listas no prolog. Exceto a função `limpa_sol()`, as demais foram retiradas dos materiais de aula do Prof. Dr. Murilo Naldi.

```
% Verifica se um elemento existe numa lista.
pertence(Elem, [Elem|_]).
pertence(Elem, [_| Cauda]) :- pertence(Elem, Cauda).

% Retira um elemento de uma lista.
retirar_elemento(Elem, [Elem|Cauda], Cauda).
retirar_elemento(Elem, [Elem1|Cauda], [Elem1|Cauda1]) :-
      retirar_elemento(Elem, Cauda, Cauda1).

% Concatena duas listas
concatena([], L, L).
concatena([Cab|Cauda], L2, [Cab|Resultado]) :-
      concatena(Cauda, L2, Resultado).

% Conta o número de elementos numa lista.
conta([], 0).
conta([_|Cauda], N) :- conta(Cauda, N1), N is N1 + 1.

% Inverte a ordem dos elementos de uma lista
inverte([], []).
inverte([Elem|Cauda], Inv) :-
      inverte(Cauda, Cauda1),
      concatena(Cauda1, [Elem], Inv).

% Função que deixa somente o caminho do bombeiro e o histórico de cargas na
resposta final
limpa_sol([], []).
limpa_sol([[Elem|_]|Cauda], [Elem|Cauda1]) :- limpa_sol(Cauda, Cauda1).
```

Funções de Busca

Foram utilizadas tanto a função de Busca em Largura quanto a Busca em Profundidade passadas pelo professor, havendo apenas o uso da função `inverte()` para deixar na ordem "cronológica" das ações do bombeiro, e a `limpa_sol()` para deixar a visualização mais simples.

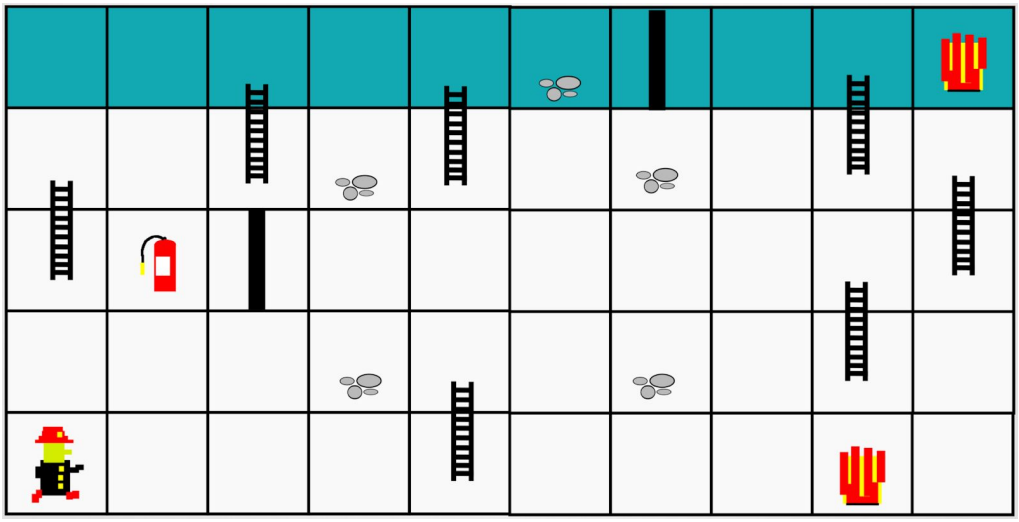
```
% --- BFS ---
% Solução por busca em largura (bl)
solucao_bl(Inicial, SolucaoInv) :-
    bl([[Inicial]], Solucao),
    limpa_sol(Solucao, Solucao1),
    inverte(Solucao1, SolucaoInv).

% 1. Se o primeiro estado de F for meta, então o retorna com o caminho
bl([[Estado|Caminho]|_], [Estado|Caminho]) :- meta(Estado).

% 2. Falha ao encontrar a meta, então estende o primeiro estado até seus
% sucessores e os coloca no final da lista de fronteira
bl([Primeiro|Outros], Solucao) :-
    estende(Primeiro, Sucessores),
    concatena(Outros, Sucessores, NovaFronteira),
    bl(NovaFronteira, Solucao).

% 2.1. Método que faz a extensão do caminho até os nós filhos do estado
estende([Estado|Caminho], ListaSucessores):-
    bagof(
        [Sucessor, Estado|Caminho],
        (s(Estado, Sucessor), not(pertence(Sucessor,[Estado|Caminho]))),
        ListaSucessores
    ), !.

% 2.2. Se o estado não tiver sucessor, falha e não procura mais (corte)
estende( _ ,[]).
```

```
% --- DFS ---
% Solução por busca em profundidade (bp)
solucao_bp(Inicial, SolucaoInv) :-
    bp([], Inicial, Solucao),
    limpa_sol(Solucao, Solucao1),
    inverte(Solucao1, SolucaoInv).

% 1. Encontra a meta
bp(Caminho, Estado, [Estado|Caminho]) :- meta(Estado).

% 2. Senão, coloca o no caminho e continua a busca
bp(Caminho, Estado, Solucao) :-
    s(Estado, Sucessor),
    not(pertence(Sucessor, Caminho)),
    bp([Estado|Caminho], Sucessor, Solucao).
```

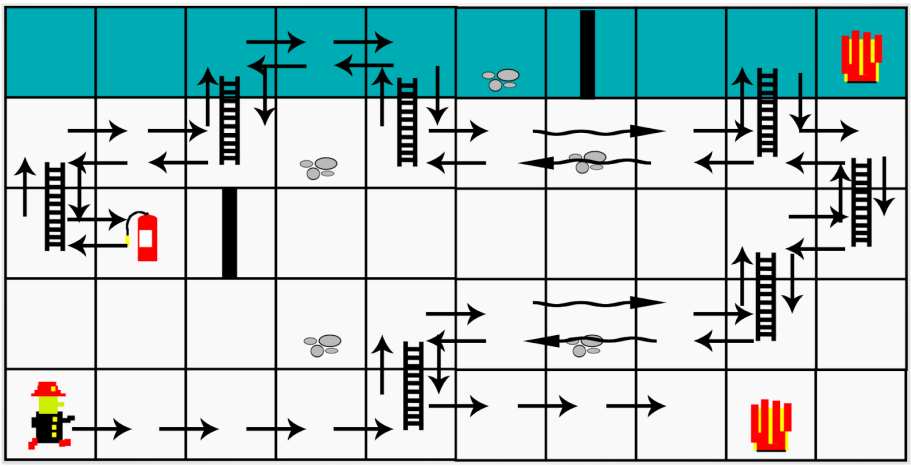
Alguns exemplos e saídas

Casos de teste passados pelo Professor:

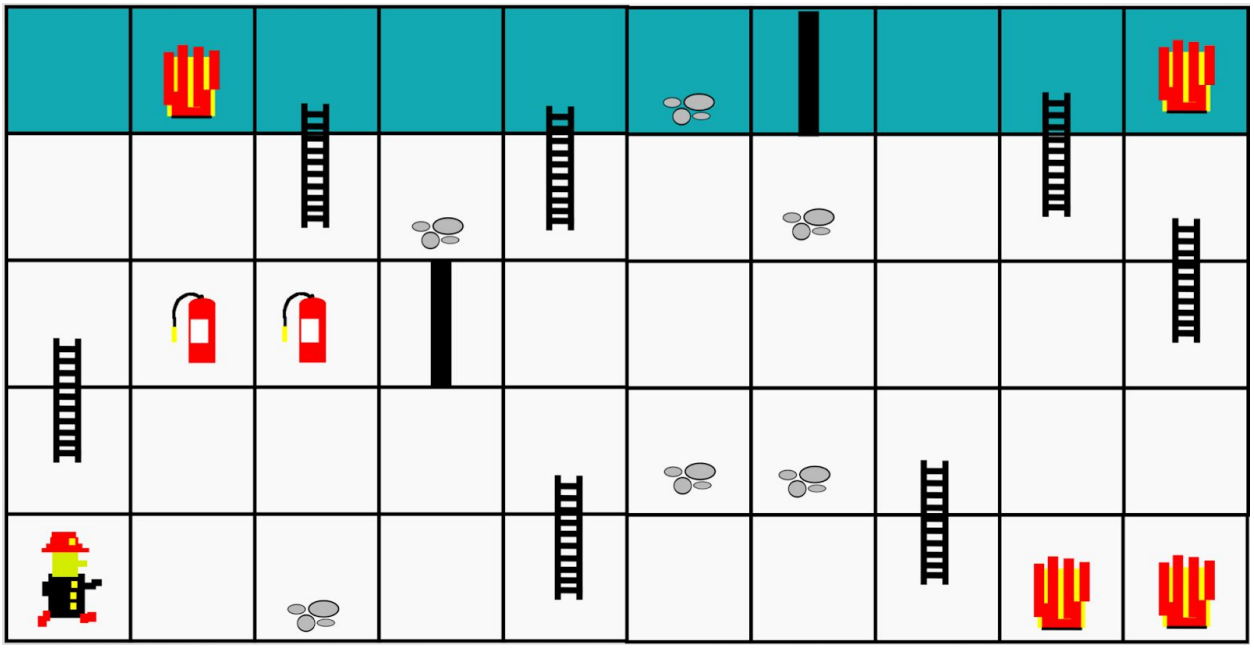
Ambiente 1: modelagem completa do ambiente no arquivo `amb1.pl`



```
Entrada: solucao_bl([ [1, 1, 0], [ [2, 3] ], [ [9, 1], [10, 5] ] ], S).

Saída: S = [
[1, 1, 0], [2, 1, 0], [3, 1, 0], [4, 1, 0], [5, 1, 0], [5, 2, 0], [6, 2, 0],
[8, 2, 0], [9, 2, 0], [9, 3, 0], [10,3, 0], [10,4, 0], [9, 4, 0], [8, 4, 0],
[6, 4, 0], [5, 4, 0], [5, 5, 0], [4, 5, 0], [3, 5, 0], [3, 4, 0], [2, 4, 0],
[1, 4, 0], [1, 3, 0], [2, 3, 0], [2, 3, 2], [1, 3, 2], [1, 4, 2], [2, 4, 2],
[3, 4, 2], [3, 5, 2], [4, 5, 2], [5, 5, 2], [5, 4, 2], [6, 4, 2], [8, 4, 2],
[9, 4, 2], [9, 5, 2], [9, 5, 1], [9, 4, 1], [10,4, 1], [10,3, 1], [9, 3, 1],
[9, 2, 1], [8, 2, 1], [6, 2, 1], [5, 2, 1], [5, 1, 1], [6, 1, 1], [7, 1, 1],
[8, 1, 1], [8, 1, 0]].
```

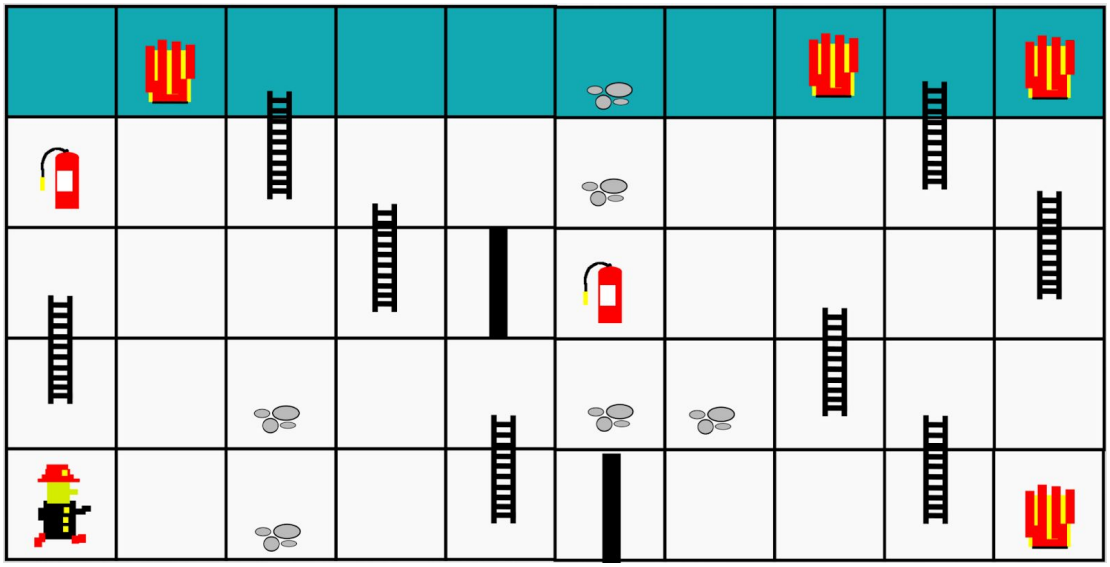


Ambiente 2: modelagem completa do ambiente no arquivo `amb2.pl`

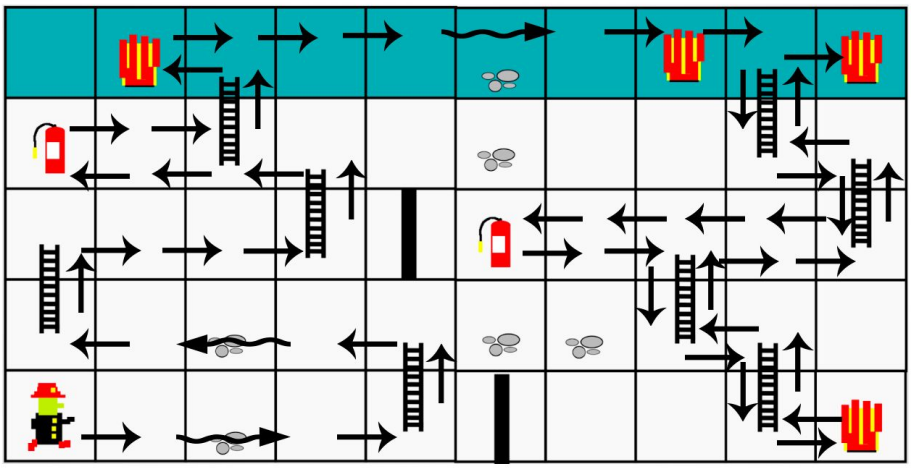


```
Entrada: solucao_bl([ [1, 1, 0], [ [2, 3], [3, 3] ], [ [9, 1], [10, 1], [2, 5], [10, 5] ] ], S).

Saída: false.
```

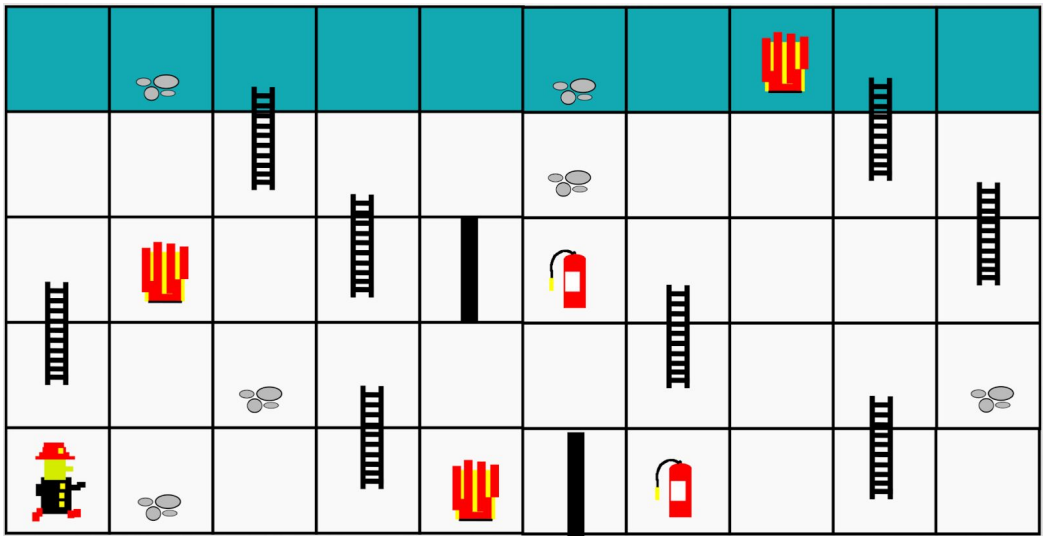
Ambiente 3: modelagem completa do ambiente no arquivo `amb3.pl`



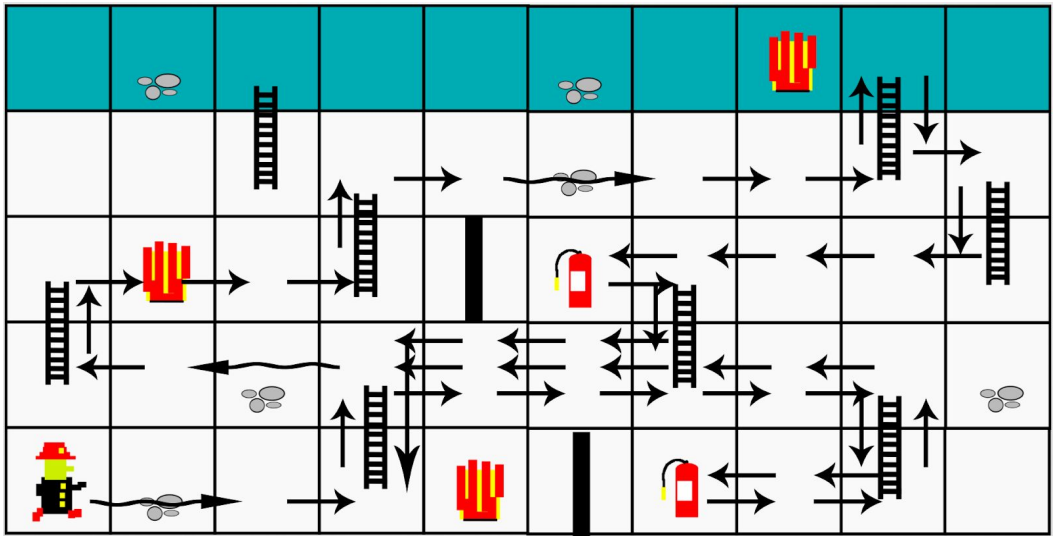
```
Entrada: solucao_bl([ [1, 1, 0], [ [6, 3], [1, 4] ],[ [10, 1], [2, 5], [8, 5],
[10, 5] ] ], S).

Saída: S = [
[1, 1, 0], [2, 1, 0], [4, 1, 0], [5, 1, 0], [5, 2, 0], [4, 2, 0], [2, 2, 0],
[1, 2, 0], [1, 3, 0], [2, 3, 0], [3, 3, 0], [4, 3, 0], [4, 4, 0], [3, 4, 0],
[2, 4, 0], [1, 4, 0], [1, 4, 2], [2, 4, 2], [3, 4, 2], [3, 5, 2], [3, 5, 1],
[4, 5, 1], [5, 5, 1], [7, 5, 1], [7, 5, 0], [8, 5, 0], [9, 5, 0], [9, 4, 0],
[10,4, 0], [10,3, 0], [9, 3, 0], [8, 3, 0], [7, 3, 0], [6, 3, 0], [6, 3, 2],
[7, 3, 2], [8, 3, 2], [8, 2, 2], [9, 2, 2], [9, 1, 2], [9, 1, 1], [9, 2, 1],
[8, 2, 1], [8, 3, 1], [9, 3, 1], [10,3, 1], [10,4, 1], [9, 4, 1], [9, 5, 1],
[9, 5, 0]].
```



Ambiente 4: modelagem completa do ambiente no arquivo `amb4.pl`

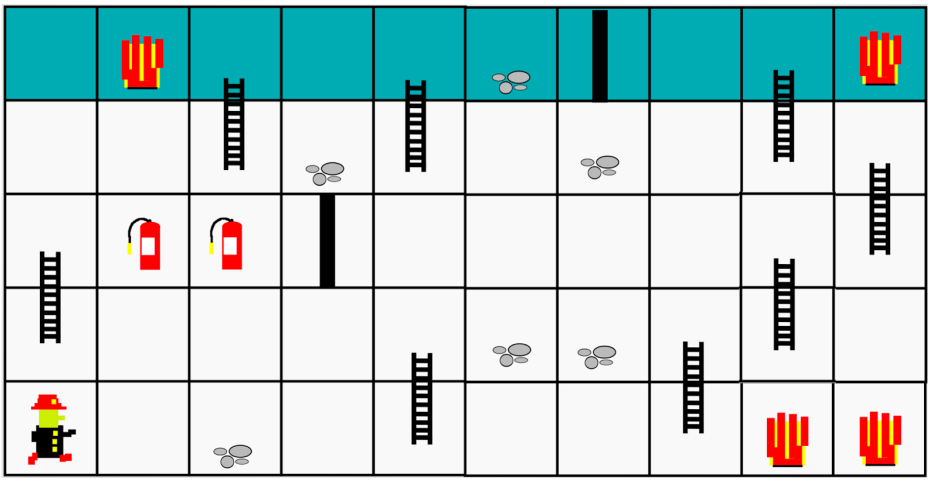


```
Entrada: solucao_bl([ [1, 1, 0], [ [7, 1], [6, 3] ], [ [5, 1], [2, 3], [8, 5]
] ], S).

Saída: S = [
[1, 1, 0], [3, 1, 0], [4, 1, 0], [4, 2, 0], [5, 2, 0], [6, 2, 0], [7, 2, 0],
[8, 2, 0], [9, 2, 0], [9, 1, 0], [8, 1, 0], [7, 1, 0], [7, 1, 2], [8, 1, 2],
[9, 1, 2], [9, 2, 2], [8, 2, 2], [7, 2, 2], [6, 2, 2], [5, 2, 2], [4, 2, 2],
[2, 2, 2], [1, 2, 2], [1, 3, 2], [1, 3, 1], [2, 3, 1], [3, 3, 1], [4, 3, 1],
[4, 4, 1], [5, 4, 1], [7, 4, 1], [8, 4, 1], [9, 4, 1], [9, 5, 1], [9, 5, 0],
[9, 4, 0], [10,4, 0], [10,3, 0], [9, 3, 0], [8, 3, 0], [7, 3, 0], [6, 3, 0],
[6, 3, 2], [7, 3, 2], [7, 2, 2], [6, 2, 2], [5, 2, 2], [4, 2, 2], [4, 1, 2],
[4, 1, 1]].
```

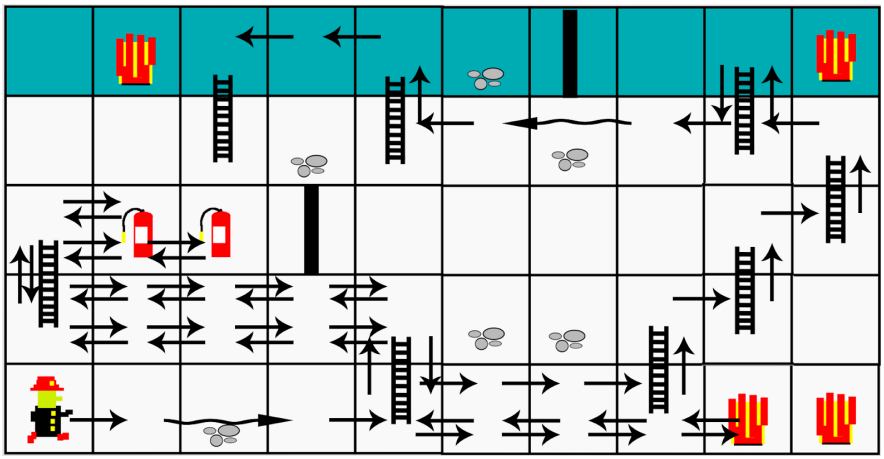


Casos de teste proprios:

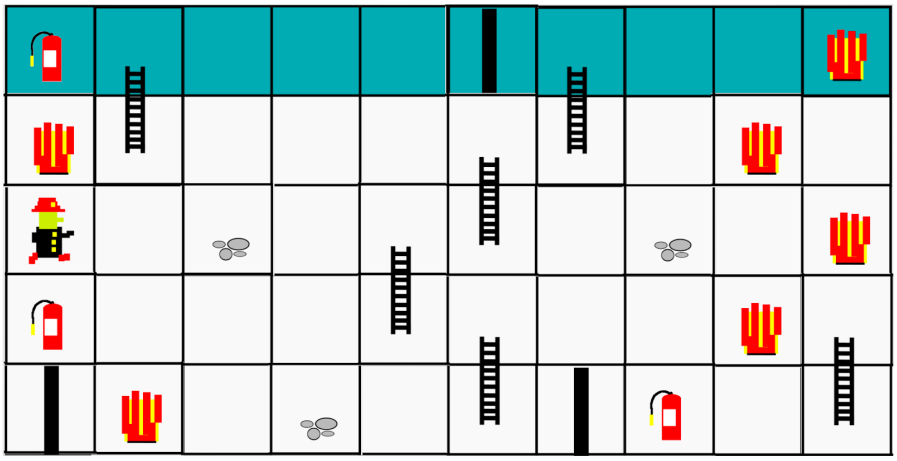
Ambiente 5: modelagem completa do ambiente no arquivo `amb5.pl`, é o ambiente 2 com uma escada a mais (que possibilita solução).



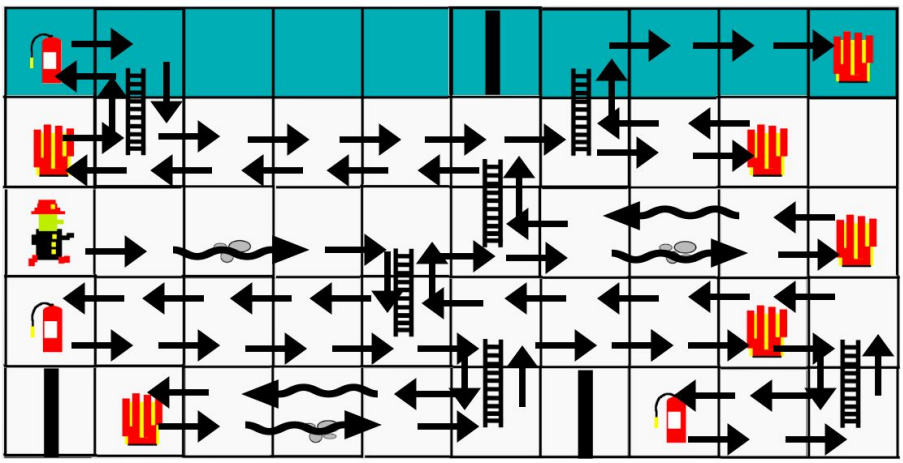
```
Entrada: solucao_bl([ [1, 1, 0], [ [2, 3], [3, 3] ], [ [9, 1], [10, 1], [2,
5], [10, 5] ] ], S).

Saída: S = [
[1, 1, 0], [2, 1, 0], [4, 1, 0], [5, 1, 0], [5, 2, 0], [4, 2, 0], [3, 2, 0],
[2, 2, 0], [1, 2, 0], [1, 3, 0], [2, 3, 0], [3, 3, 0], [3, 3, 2], [2, 3, 2],
[1, 3, 2], [1, 2, 2], [2, 2, 2], [3, 2, 2], [4, 2, 2], [5, 2, 2], [5, 1, 2],
[6, 1, 2], [7, 1, 2], [8, 1, 2], [8, 1, 1], [9, 1, 1], [9, 1, 0], [8, 1, 0],
[7, 1, 0], [6, 1, 0], [5, 1, 0], [5, 2, 0], [4, 2, 0], [3, 2, 0], [2, 2, 0],
[1, 2, 0], [1, 3, 0], [2, 3, 0], [2, 3, 2], [1, 3, 2], [1, 2, 2], [2, 2, 2],
[3, 2, 2], [4, 2, 2], [5, 2, 2], [5, 1, 2], [6, 1, 2], [7, 1, 2], [8, 1, 2],
[8, 2, 2], [9, 2, 2], [9, 3, 2], [10,3, 2], [10,4, 2], [9, 4, 2], [9, 5, 2],
[9, 5, 1], [9, 4, 1], [8, 4, 1], [6, 4, 1], [5, 4, 1], [5, 5, 1], [4, 5, 1],
[3, 5, 1], [3, 5, 0]].
```



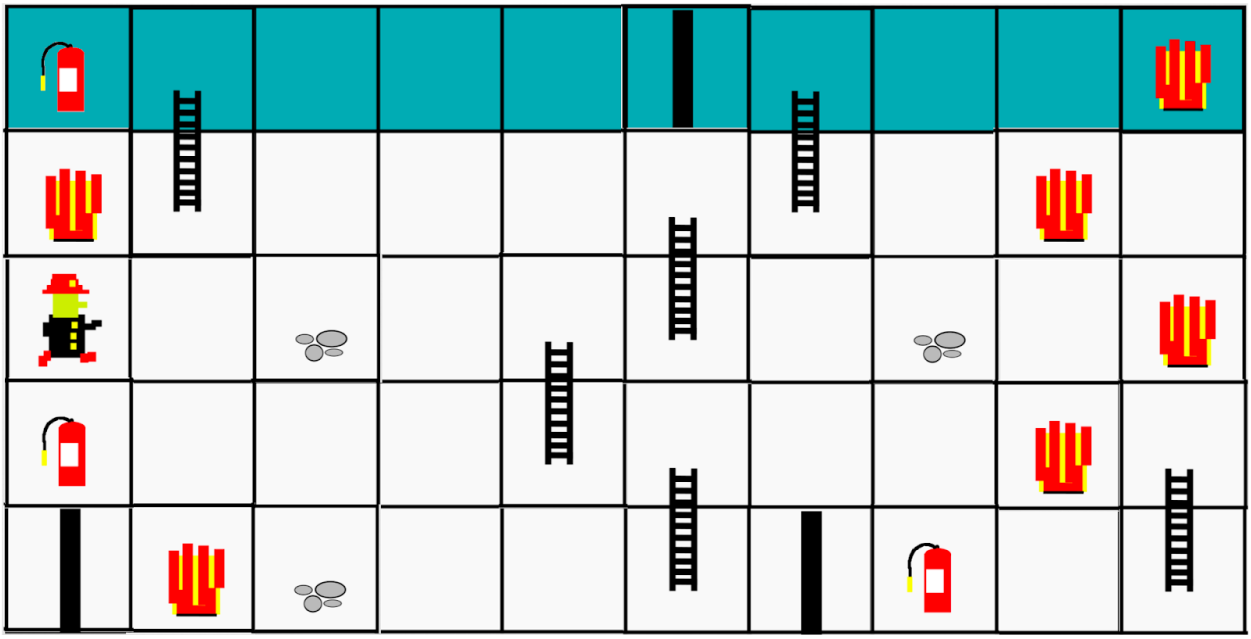
Ambiente 6: modelagem completa do ambiente no arquivo `amb6.pl`



```
Entrada: solucao_bl([ [1, 3, 0], [ [8, 1], [1, 2], [1, 5] ], [ [2, 1], [9, 2],
[10, 3], [1, 4], [9, 4], [10, 5] ] ], S).

Saída: S = [
[1, 3, 0], [2, 3, 0], [4, 3, 0], [5, 3, 0], [5, 2, 0], [4, 2, 0], [3, 2, 0],
[2, 2, 0], [1, 2, 0], [1, 2, 2], [2, 2, 2], [3, 2, 2], [4, 2, 2], [5, 2, 2],
[6, 2, 2], [6, 1, 2], [5, 1, 2], [3, 1, 2], [3, 1, 1], [5, 1, 1], [6, 1, 1],
[6, 2, 1], [7, 2, 1], [8, 2, 1], [8, 2, 0], [9, 2, 0], [10,2, 0], [10,1, 0],
[9, 1, 0], [8, 1, 0], [8, 1, 2], [9, 1, 2], [10,1, 2], [10,2, 2], [9, 2, 2],
[8, 2, 2], [7, 2, 2], [6, 2, 2], [5, 2, 2], [5, 3, 2], [6, 3, 2], [7, 3, 2],
[9, 3, 2], [9, 3, 1], [7, 3, 1], [6, 3, 1], [6, 4, 1], [5, 4, 1], [4, 4, 1],
[3, 4, 1], [2, 4, 1], [2, 4, 0], [2, 5, 0], [1, 5, 0], [1, 5, 2], [2, 5, 2],
[2, 4, 2], [3, 4, 2], [4, 4, 2], [5, 4, 2], [6, 4, 2], [7, 4, 2], [8, 4, 2],
[8, 4, 1], [7, 4, 1], [7, 5, 1], [8, 5, 1], [9, 5, 1], [9, 5, 0]].
```

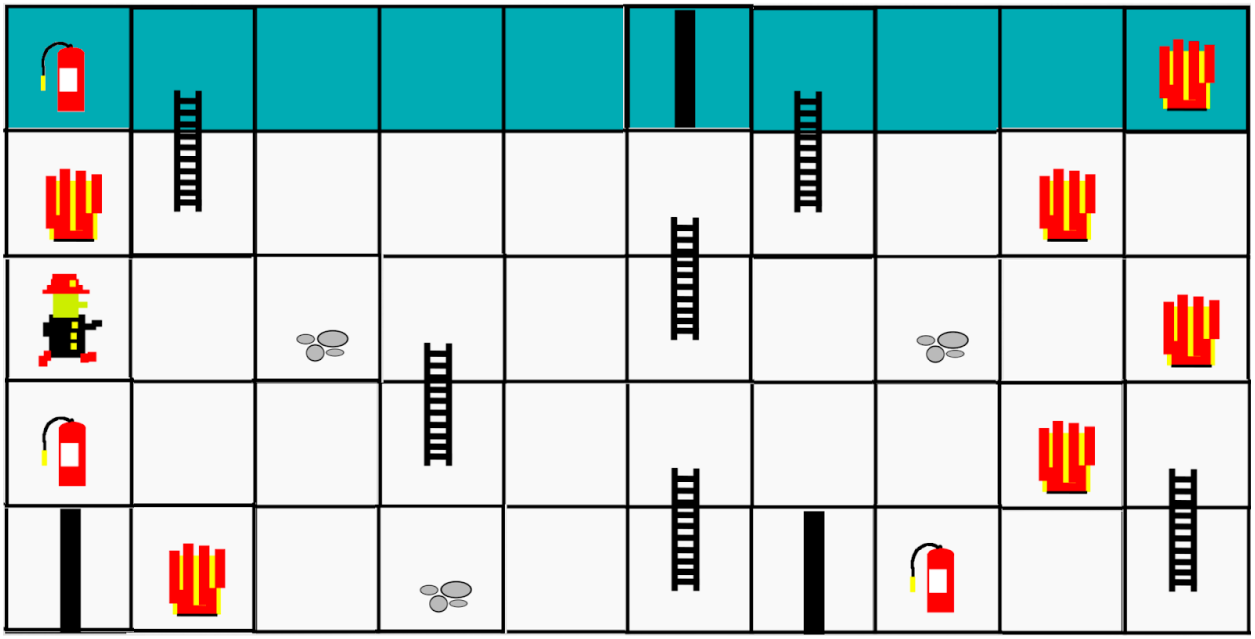


Ambiente 7: modelagem completa do ambiente no arquivo `amb7.pl`, ambiente 6 porém com um entulho que impede o salto.



```
Entrada: solucao_bl([ [1, 3, 0], [ [8, 1], [1, 2], [1, 5] ], [ [2, 1], [9, 2], [10, 3], [1, 4], [9, 4], [10, 5] ] ], S).

Saída: false.
```

Ambiente 8: modelagem completa do ambiente no arquivo `amb8.pl`, ambiente 6 porém com uma escada que impede o salto.



```
Entrada: solucao_bl([ [1, 3, 0], [ [8, 1], [1, 2], [1, 5] ], [ [2, 1], [9, 2], [10, 3], [1, 4], [9, 4], [10, 5] ] ], S).

Saída: false.
```

Bibliografia

As funções auxiliares de manipulação de listas em prolog foram retiradas dos slides de aula do Prof. Murilo Naldi, bem como as funções de Busca em Largura e em Profundidade (com pequenas alterações no retorno da solução final).

As Imagens dos casos de teste 1 à 4 foram retiradas dos slides de aula do Prof. Murilo Naldi, partes dessas imagens foram usadas para montar os casos 5 à 8.