

UFSCar - Universidade Federal de São Carlos
Departamento de Computação

Inteligência Artificial - 2019.2 (1001336)

Murilo Naldi

Trabalho 1

Alcides Mignoso e Silva - 760479

Matheus Victorello - 758606

23/10/2019

1. Objetivo

Este trabalho consiste no desenvolvimento de regras para a linguagem de programação Prolog que sejam capazes de permitir que um agente resolva um problema em um determinado ambiente informado, de forma que o código seja capaz de resolver todos os possíveis cenários.

2. Problema

Para os que não conhecem o problema, o mesmo consiste em um agente bombeiro que tem o objetivo de encontrar extintores e apagar focos de incêndio em um mapa simulado por uma matriz de 5 linhas e 10 colunas.

Para isso, o bombeiro pode subir de linha (andar) utilizando eventuais escadas no mapa. O agente deve resgatar os extintores, que lhe dão, individualmente, 2 cargas para apagar incêndios. O agente só pode resgatar um extintor quando estiver sem cargas, e cada carga pode apagar somente um foco de incêndio.

Entretanto, existem obstáculos no mapa. O mesmo possui espaços que contém paredes divisórias, pelas quais o bombeiro não pode passar. Também existem pilhas de entulho, pelas quais o bombeiro só consegue passar (saltar) caso as duas posições horizontais adjacentes da mesma estejam livres (sem nenhum objeto).

Dado esse cenário, é necessário encontrar o resultado: um caminho o qual o bombeiro deve fazer para apagar os fogos, visando a menor sequência de passos possíveis, utilizando as regras elaboradas e a estratégia de busca às cegas.

3. Resultados

Foi escolhida a busca em largura para efetuar a busca às cegas, pelo motivo de, apesar de ser mais custosa, encontrar sempre o melhor resultado possível.

Basicamente, fora estruturado, através de regras no Prolog, um ambiente no qual o bombeiro pode percorrer, através do backtracking realizado pela linguagem, todos os caminhos possíveis de serem realizados no mapa. Entretanto, como isso seria muito custoso, optamos por parar o programa quando um primeiro resultado fosse encontrado (tendo em vista que existem diversos possíveis resultados ótimos), uma vez que, por conta da utilização da busca em largura, ele já é ótimo.

Para isso, duas versões de código foram elaboradas. Elas podem ser encontradas em <https://github.com/alcidesmig/ia-2019/> e <https://github.com/matheusvictorello/ia-t1-2019>.

Por conta de melhor desempenho, a versão do primeiro link foi a escolhida para ser a versão final.

4. Código

Funções auxiliares para manipulação das tuplas, para a busca em largura e demais dependências foram disponibilizadas pelo professor, e serão apresentadas em breve.

Em relação ao problema em si, diversas estratégias poderiam ser utilizadas para representar o mapa e os objetos do mesmo. Optamos, entretanto, por utilizar tuplas na representação de todos dos mesmos, por conta da facilidade existente na manipulação.

A representação do estado consiste na seguinte tupla:

$(I, J, Fogos, CargasExtintor, Extintores)$

na qual I é a posição da linha atual (começando em 0 e indo até 4, onde 4 é a última linha inferior) e J é a posição da coluna atual (começando de 0 e indo até 9), $CargasExtintor$ é a quantidade de cargas que o bombeiro possui disponíveis para serem utilizadas em um dado momento, $Fogos$ é uma tupla de tuplas que representam as posições onde existem foco de incêndio ainda não apagados dentro do mapa (Exemplo: $((0, 0), (0, 1))$) e $Extintores$ é uma tupla de tuplas que representam as posições onde existem extintores que ainda não foram utilizados (Exemplo: $((1, 0), (1, 1))$).

As pedras, escadas e bloqueios são fixas, não variam de acordo com o estado atual do bombeiro e, portanto, são representadas através de regras fixas no prolog. Exemplo:

$escada(0, 1)..$

$bloqueio(4, 0).$

$pedra(2, 7).$

observação: dizer que existe uma escada em (X, Y) é dizer que existe uma escada que permite ao bombeiro andar da linha X para a linha $X + 1$ na coluna Y .

Tendo em vista o ambiente apresentado, algumas funções auxiliares foram utilizadas:

$limiteI(X) :- X < 5, X \geq 0.$

$limiteJ(X) :- X < 10, X \geq 0.$

que são regras responsáveis por determinar os limites do mapa, para expandir o ambiente basta alterar os valores limitantes das regras.

```

livre(I, J, Fogos) :-
    not(pedra(I, J)),
    not(escada(I - 1, J)),
    not(escada(I, J)),
    not(bloqueio(I, J)),
    not(pertence((I, J), Fogos)).

```

que é a regra responsável por informar se uma determinada posição (I, J) está livre no mapa (não existe nenhum objeto em (I, J)).

```

andarPedra(I, J, Fogos) :-
    not(pedra(I, J))
    ;
    (
        pedra(I, J),
        NJD = I + 1,
        livre(I, NJD, Fogos),
        NJE = I - 1,
        livre(I, NJE, Fogos)
    ).

```

que é a regra responsável por dizer se é possível andar para uma posição (I, J) com uma possível pedra/entulho existente. Pode-se andar, portanto, caso não exista nenhum entulho na posição, ou exista entulho mas o ponto tem as posições adjacentes (I-1 e I+1) livres.

```

andarFogo(I, J, Fogos, CargasExtintor) :-
    (
        not(pertence((I, J), Fogos))
        ;
        (
            pertence((I, J), Fogos),
            CargasExtintor > 0
        )
    ).

```

que é a regra responsável por dizer se é possível andar para uma posição com um possível foco de incêndio (fogo). Pode-se andar, portanto, caso não exista foco na posição, ou exista fogo e o bombeiro possui cargas suficientes para apagá-lo.

Observação: o código das regras acima pode ser integrado diretamente ao código das outras regras (de movimentação), e isso foi feito a priori. Entretanto, com a finalidade de tornar o código mais legível, as regras foram separadas.

Além das regras auxiliares, existem as regras responsáveis por permitir a mudança de estado no problema. São 6 regras: apagar fogo, pegar extintor, movimentar-se a direita, movimentar-se a esquerda, movimentar-se para baixo e movimentar-se para cima. Segue abaixo o código e a explicação de cada um:

```
% Apaga fogo
s(
    (I, J, Fogos, CargasExtintor, Extintores),
    (I, J, NovoFogos, NovoCargasExtintor, Extintores)
) :-
    pertence((I, J), Fogos),
    NovoCargasExtintor is CargasExtintor - 1,
    CargasExtintor > 0,
    remove_elem((I, J), Fogos, NovoFogos).
```

A regra acima é a responsável por permitir que focos de incêndio (fogo) sejam apagados pelo bombeiro. Para apagar o fogo, o bombeiro tem que estar em cima dele, entretanto, a regra *andarFogo* faz com que o bombeiro só possa estar em cima de um fogo caso tenha carga suficiente para apagá-lo, o que torna o *CargasExtintor > 0* da regra desnecessário, porém optamos por mantê-lo visando a legibilidade do código. Quanto ao restante do código dessa regra, é feita a verificação da existência de um fogo na posição (que é a qual o bombeiro se encontra), e caso exista, o bombeiro apaga o mesmo (o remove da lista de fogos).

```
% Pega extintor
s(
    (I, J, Fogos, CargasExtintor, Extintores),
    (I, J, Fogos, NovoCargasExtintor, NovoExtintores)
) :-
    pertence((I, J), Extintores),
    NovoCargasExtintor is 2,
    CargasExtintor == 0,
    remove_elem((I, J), Extintores, NovoExtintores).
```

A regra acima é responsável por permitir que o bombeiro pegue extintores. O bombeiro pode pegar um extintor se a posição na qual ele estiver (I, J) possuir um extintor (*pertence((I, J), Extintores)*), se estiver com as cargas zeradas

(CargasExtintor == 0) e se o novo valor para as cargas for 2. Por fim, é feita a remoção do extintor pego da lista de extintores disponíveis.

Começando com as regras de movimentação, temos:

% Movimentacao horizontal - direita

```
s(  
    (I, J, Fogos, CargasExtintor, Extintores),  
    (I, NovoJ, Fogos, CargasExtintor, Extintores)  
) :-  
    NovoJ is J + 1,  
    limiteJ(NovoJ),  
    not(bloqueio(I, NovoJ)),  
    andarPedra(I, NovoJ, Fogos),  
    andarFogo(I, NovoJ, Fogos, CargasExtintor).
```

% Movimentacao horizontal - esquerda

```
s(  
    (I, J, Fogos, CargasExtintor, Extintores),  
    (I, NovoJ, Fogos, CargasExtintor, Extintores)  
) :-  
    NovoJ is J - 1,  
    limiteJ(NovoJ),  
    not(bloqueio(I, NovoJ)),  
    andarPedra(I, NovoJ, Fogos),  
    andarFogo(I, NovoJ, Fogos, CargasExtintor).
```

as regras acima são as responsáveis pela movimentação horizontal do bombeiro, elas verificam se o novo valor de J (NovoJ) está dentro dos limites do mapa, se não existe bloqueio nesse mesmo valor, e utiliza das regras anteriormente apresentadas para verificar se é possível andar para a posição (I, NovoJ), em relação a existência de focos de incêndio (fogo) ou pedras. Caso tudo seja satisfeito, ocorre a transição para o novo estado com o novo valor de J.

Existem, por fim, mais duas regras:

```
s(  
    (I, J, Fogos, CargasExtintor, Extintores),  
    (NovoI, J, Fogos, CargasExtintor, Extintores)  
) :- NovoI is I + 1,  
    escada(I, J),  
    limiteI(NovoI).
```



```

s(
    (I, J, Fogos, CargasExtintor, Extintores),
    (NovoI, J, Fogos, CargasExtintor, Extintores)
) :- NovoI is I - 1,
    escada(NovoI, J),
    limiteI(NovoI).

```

que são as regras responsáveis por permitir que o bombeiro suba e desça escadas. A primeira é a de descida e a segunda, a de subida. Na primeira é verificado se a posição na qual ele está possui uma escada, e se a nova posição ainda está dentro do mapa, se sim, a nova posição é setada. Já a segunda verifica se a posição acima possui uma escada (conforme segue a representação já apresentada dos objetos), e se possuir, e estiver dentro dos limites do mapa, a movimentação é feita.

É necessário, por fim, definir o estado de meta no início do código. O nosso estado de meta é:

```

meta((_, _, [], _, _)).

```

que representa uma posição I, J qualquer, onde não existem focos de incêndio acesos, e é indiferente a existência de cargas no extintor ou extintores disponíveis.

Existem, também, os predicados disponibilizados pelo professor para a realização da busca às cegas. Não convém ao caso explicar os mesmos, mas seguem os seus códigos abaixo:

```

% Auxiliar ~ Disponibilizado por Murilo Naldi
inverte_lista([], X, X).
inverte_lista([X|Xs], Y, R) :-
    inverte_lista(Xs, Y, [X|R]).
remove_elem(Elem, [Elem|Resto], Resto).
remove_elem(Elem, [Cabeça|Resto], [Cabeça|NovoResto]) :-
    remove_elem(Elem, Resto, NovoResto).
pertence(Elem, [Elem|_]).
pertence(Elem, [_|Cauda]) :-
    pertence(Elem, Cauda).
concatena([ ], L, L).
concatena([Cab|Cauda], L2, [Cab|Resultado]) :-
    concatena(Cauda, L2, Resultado).
solucao_bl(Inicial, Solucao) :-
    bl([[Inicial]], Solucao).
bl([[Estado|Caminho]|_], [Estado|Caminho]) :-

```

```

    meta(Estado).
bl([Caminho|Outros], Solucao) :-
    estende(Caminho, NovoCaminho),
    concatena(Outros, NovoCaminho, CaminhoAnterior),
    bl(CaminhoAnterior, Solucao).
estende([Estado|Caminho], NovoCaminho):-
    bagof([Sucessor, Estado|Caminho], (s(Estado,Sucessor), not(pertence(Sucessor,
[Estado|Caminho]))), NovoCaminho), !.
estende(_, []).
solucao_bp(Inicial, Solucao) :-
    bp([], Inicial, Solucao).
bp(Caminho, Estado, [Estado|Caminho]) :-
    meta(Estado).
bp(Caminho, Estado, Solucao) :-
    s(Estado, Sucessor),
    not(pertence(Sucessor, Caminho)),
    bp([Estado|Caminho], Sucessor, Solucao).

```

Tendo em vista todos os códigos, basta, então, passar o ambiente através da base de dados e do estado inicial. Segue exemplo abaixo.

Na base de dados:

```

escada(0, 2).
escada(0, 4).
escada(0, 8).
escada(1, 0).
escada(1, 9).
escada(2, 8).
escada(3, 4).

pedra(0, 5).
pedra(1, 3).
pedra(1, 6).
pedra(3, 3).
pedra(3, 6).

```

```

bloqueio(0, 6).
bloqueio(2, 2).

```

No Prolog:

```

solucao_bp(bl((4, 0, [(0, 9), (4, 8)], 0, [(2, 1)]), X).

```

5. Representação gráfica com Javascript e Python

Para a representação das ações do bombeiro foi feito um visualizador em Javascript (p5.js), que usa um servidor local em Python (Flask).

O visualizador permite que casos de teste sejam montados interativamente, enviados para o prolog e então animados pelo mesmo. A comunicação entre o Javascript e o Prolog ocorre da seguinte maneira:

1. Um caso é montado no visualizador.
2. O visualizador converte uma matriz com informação sobre cada posição no grid para um dicionário.
3. O dicionário é enviado para o python como um JSON.
4. O Python interpreta o dicionário e monta um arquivo com as informações contidas nele, indicando a posição dos elementos estáticos, focos, extintores, bombeiro e limites.
5. O Python então roda o prolog como um comando no terminal usando uma *pipe* para importar o arquivo base do prolog, o arquivo com as informações e o predicado que acha a solução. Outra pipe é usada para redirecionar a saída do prolog para um arquivo.
6. O Python então lê o arquivo de resposta do Prolog, o converte para uma lista de ações contendo, posição do bombeiro, número de cargas, fogos não apagados, extintores que não foram pegos e a ação em si.
7. Essa lista é enviada como resposta para o Javascript como um JSON.
8. O Javascript faz um parsing da lista e passa a usá-la para animar os elementos do grid.

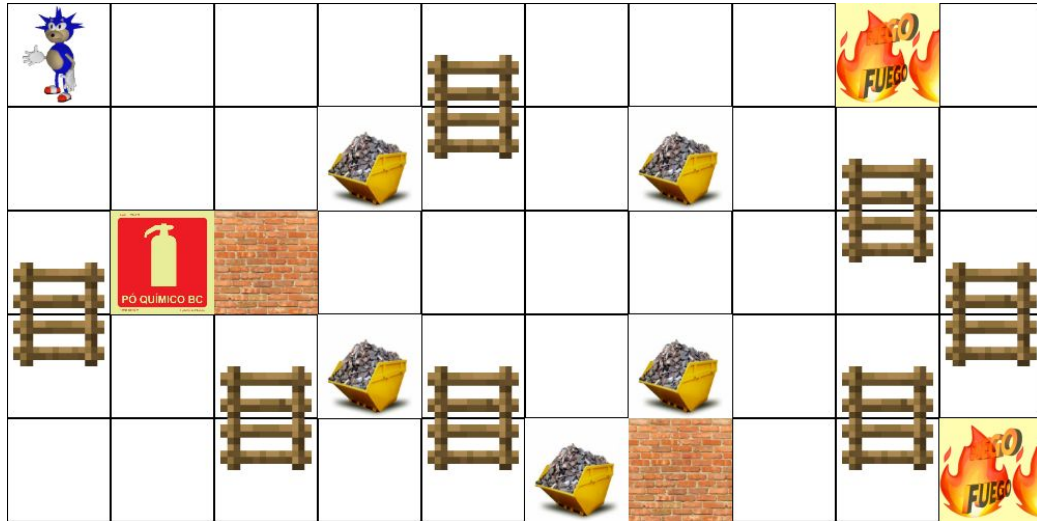
OBS: O código do visualizador não foi incluído já que não é o foco do trabalho, mas pode ser encontrado em: <https://github.com/matheusvictorello/ia-t1-2019>.

6. Casos de teste

Seguem, nas próximas páginas, alguns casos de testes realizados, seguidos do primeiro resultado encontrado pela busca em largura no Prolog.

Cada tupla representa um estado pelo qual o bombeiro vai passar, e segue o formato $(X, Y, Cargas, Fogos, Extintores, Acao)$, explicado anteriormente com o acréscimo de *Acao* sendo a ação realizada pelo agente, alteração dos valores I e J por Y e X, respectivamente (com X sendo a coluna e Y a linha). A representação dos objetos também segue essa mudança. Essas alterações na representação dos resultados foram feitas com a finalidade de tornar o ambiente gráfico compatível com ambos os códigos (<https://github.com/alcidesmig/ia-2019/> e <https://github.com/matheusvictorello/ia-t1-2019>) antes citados.

Caso 1

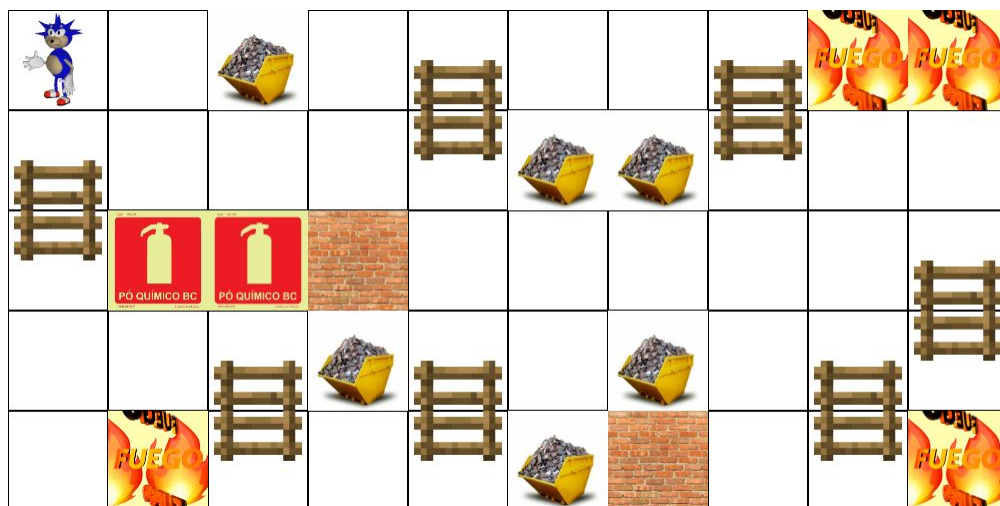


S =

[(0, 0, 0, [[8, 0], [9, 4]], [[1, 2]], "Comeco"),
 (1, 0, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (2, 0, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (3, 0, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (4, 0, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (4, 1, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (5, 1, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (7, 1, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (8, 1, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (8, 2, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (9, 2, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (9, 3, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (8, 3, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (7, 3, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (5, 3, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (4, 3, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (4, 4, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (3, 4, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (2, 4, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (2, 3, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (1, 3, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (0, 3, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (0, 2, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (1, 2, 0, [[8, 0], [9, 4]], [[1, 2]], "Anda"),
 (1, 2, 2, [[8, 0], [9, 4]], [], "Pega"),
 (0, 2, 2, [[8, 0], [9, 4]], [], "Anda"),
 (0, 3, 2, [[8, 0], [9, 4]], [], "Anda"),

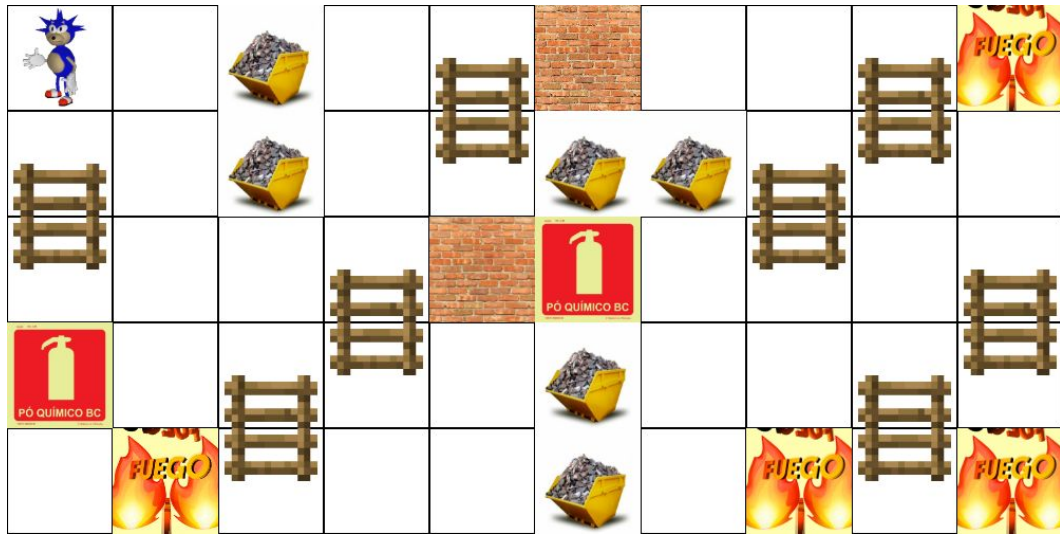
(1, 3, 2, [[8, 0], [9, 4]], [], "Anda"),
 (2, 3, 2, [[8, 0], [9, 4]], [], "Anda"),
 (2, 4, 2, [[8, 0], [9, 4]], [], "Anda"),
 (3, 4, 2, [[8, 0], [9, 4]], [], "Anda"),
 (4, 4, 2, [[8, 0], [9, 4]], [], "Anda"),
 (4, 3, 2, [[8, 0], [9, 4]], [], "Anda"),
 (5, 3, 2, [[8, 0], [9, 4]], [], "Anda"),
 (7, 3, 2, [[8, 0], [9, 4]], [], "Anda"),
 (8, 3, 2, [[8, 0], [9, 4]], [], "Anda"),
 (8, 4, 2, [[8, 0], [9, 4]], [], "Anda"),
 (9, 4, 2, [[8, 0], [9, 4]], [], "Anda"),
 (9, 4, 1, [[8, 0]], [], "Apaga"),
 (8, 4, 1, [[8, 0]], [], "Anda"),
 (8, 3, 1, [[8, 0]], [], "Anda"),
 (9, 3, 1, [[8, 0]], [], "Anda"),
 (9, 2, 1, [[8, 0]], [], "Anda"),
 (8, 2, 1, [[8, 0]], [], "Anda"),
 (8, 1, 1, [[8, 0]], [], "Anda"),
 (7, 1, 1, [[8, 0]], [], "Anda"),
 (5, 1, 1, [[8, 0]], [], "Anda"),
 (4, 1, 1, [[8, 0]], [], "Anda"),
 (4, 0, 1, [[8, 0]], [], "Anda"),
 (5, 0, 1, [[8, 0]], [], "Anda"),
 (6, 0, 1, [[8, 0]], [], "Anda"),
 (7, 0, 1, [[8, 0]], [], "Anda"),
 (8, 0, 1, [[8, 0]], [], "Anda"),
 (8, 0, 0, [], [], "Apaga")]

Caso 2



false.

Caso 3

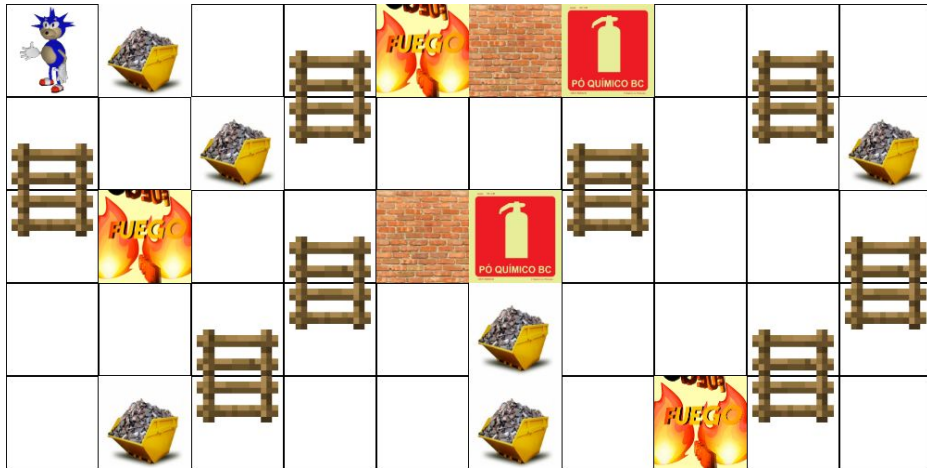

$$S =$$
$$[(0, 0, 0, [[9, 0], [1, 4], [7, 4], [9, 4]], [[0, 3], [5, 2]]],$$

"Comeco"),

(1, 0, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(3, 0, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(4, 0, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(4, 1, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(3, 1, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(1, 1, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(0, 1, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(0, 2, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(1, 2, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(2, 2, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(3, 2, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(3, 3, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(2, 3, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(1, 3, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(0, 3, 0, [9, 0], [1, 4], [7, 4], [9, 4]), [[0, 3], [5, 2]], "Anda"),
(0, 3, 2, [9, 0], [1, 4], [7, 4], [9, 4]), [[5, 2]], "Pega"),
(1, 3, 2, [9, 0], [1, 4], [7, 4], [9, 4]), [[5, 2]], "Anda"),
(2, 3, 2, [9, 0], [1, 4], [7, 4], [9, 4]), [[5, 2]], "Anda"),
(2, 4, 2, [9, 0], [1, 4], [7, 4], [9, 4]), [[5, 2]], "Anda"),
(1, 4, 2, [9, 0], [1, 4], [7, 4], [9, 4]), [[5, 2]], "Anda"),
(1, 4, 1, [9, 0], [7, 4], [9, 4]), [[5, 2]], "Apaga"),
(2, 4, 1, [9, 0], [7, 4], [9, 4]), [[5, 2]], "Anda"),
(3, 4, 1, [9, 0], [7, 4], [9, 4]), [[5, 2]], "Anda"),
(4, 4, 1, [9, 0], [7, 4], [9, 4]), [[5, 2]], "Anda"),
(6, 4, 1, [9, 0], [7, 4], [9, 4]), [[5, 2]], "Anda"),
(7, 4, 1, [9, 0], [7, 4], [9, 4]), [[5, 2]], "Anda"),
(7, 4, 0, [9, 0], [9, 4]), [[5, 2]], "Apaga"),
(8, 4, 0, [9, 0], [9, 4]), [[5, 2]], "Anda"),
(9, 3, 0, [9, 0], [9, 4]), [[5, 2]], "Anda"),
(9, 3, 0, [9, 0], [9, 4]), [[5, 2]], "Anda"),
(9, 2, 0, [9, 0], [9, 4]), [[5, 2]], "Anda"),
(8, 2, 0, [9, 0], [9, 4]), [[5, 2]], "Anda"),
(7, 2, 0, [9, 0], [9, 4]), [[5, 2]], "Anda"),
(6, 2, 0, [9, 0], [9, 4]), [[5, 2]], "Anda"),
(5, 2, 0, [9, 0], [9, 4]), [[5, 2]], "Anda"),

(5, 2, 2, [[9, 0], [9, 4]], [], "Pega"),
 (6, 2, 2, [[9, 0], [9, 4]], [], "Anda"),
 (7, 2, 2, [[9, 0], [9, 4]], [], "Anda"),
 (7, 1, 2, [[9, 0], [9, 4]], [], "Anda"),
 (8, 1, 2, [[9, 0], [9, 4]], [], "Anda"),
 (8, 0, 1, [[9, 0], [9, 4]], [], "Anda"),
 (9, 0, 2, [[9, 0], [9, 4]], [], "Anda"),
 (9, 0, 1, [[9, 4]], [], "Apaga"),
 (8, 0, 1, [[9, 4]], [], "Anda"),
 (8, 1, 1, [[9, 4]], [], "Anda"),
 (7, 1, 1, [[9, 4]], [], "Anda"),
 (7, 2, 1, [[9, 4]], [], "Anda"),
 (8, 2, 1, [[9, 4]], [], "Anda"),
 (9, 2, 1, [[9, 4]], [], "Anda"),
 (9, 3, 1, [[9, 4]], [], "Anda"),
 (8, 3, 1, [[9, 4]], [], "Anda"),
 (8, 4, 1, [[9, 4]], [], "Anda"),
 (9, 4, 1, [[9, 4]], [], "Anda"),
 (9, 4, 0, [], [], "Apaga")]

Caso 4









S =

[(0, 0, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Comeco"),
 (2, 0, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (3, 0, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (3, 1, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (4, 1, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (5, 1, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (6, 1, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (7, 1, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (8, 1, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (8, 0, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (7, 0, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (6, 0, 0, [[4, 0], [1, 2], [7, 4]], [[6, 0], [5, 2]], "Anda"),
 (6, 0, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Pega"),
 (7, 0, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (8, 0, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (8, 1, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (7, 1, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (6, 1, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (5, 1, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (4, 1, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (3, 1, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (1, 1, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (0, 1, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (0, 2, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (1, 2, 2, [[4, 0], [1, 2], [7, 4]], [[5, 2]], "Anda"),
 (1, 2, 1, [[4, 0], [7, 4]], [[5, 2]], "Apaga"),
 (2, 2, 1, [[4, 0], [7, 4]], [[5, 2]], "Anda"),
 (3, 2, 1, [[4, 0], [7, 4]], [[5, 2]], "Anda"),
 (3, 3, 1, [[4, 0], [7, 4]], [[5, 2]], "Anda"),
 (4, 3, 1, [[4, 0], [7, 4]], [[5, 2]], "Anda"),
 (6, 3, 1, [[4, 0], [7, 4]], [[5, 2]], "Anda"),
 (7, 3, 1, [[4, 0], [7, 4]], [[5, 2]], "Anda"),
 (8, 3, 1, [[4, 0], [7, 4]], [[5, 2]], "Anda"),
 (8, 4, 1, [[4, 0], [7, 4]], [[5, 2]], "Anda"),
 (7, 4, 1, [[4, 0], [7, 4]], [[5, 2]], "Anda"),
 (7, 4, 0, [[4, 0]], [[5, 2]], "Apaga"),
 (8, 4, 0, [[4, 0]], [[5, 2]], "Anda"),
 (8, 3, 0, [[4, 0]], [[5, 2]], "Anda"),
 (9, 3, 0, [[4, 0]], [[5, 2]], "Anda"),
 (9, 2, 0, [[4, 0]], [[5, 2]], "Anda"),
 (8, 2, 0, [[4, 0]], [[5, 2]], "Anda"),
 (7, 2, 0, [[4, 0]], [[5, 2]], "Anda"),
 (6, 2, 0, [[4, 0]], [[5, 2]], "Anda"),
 (5, 2, 0, [[4, 0]], [[5, 2]], "Anda"),

(5, 2, 2, [[4, 0]], [], "Pega"),
 (6, 2, 2, [[4, 0]], [], "Anda"),
 (6, 1, 2, [[4, 0]], [], "Anda"),
 (5, 1, 2, [[4, 0]], [], "Anda"),
 (4, 1, 2, [[4, 0]], [], "Anda"),
 (3, 1, 2, [[4, 0]], [], "Anda"),
 (3, 0, 2, [[4, 0]], [], "Anda"),
 (4, 0, 2, [[4, 0]], [], "Anda"),
 (4, 0, 1, [], [], "Apaga")]

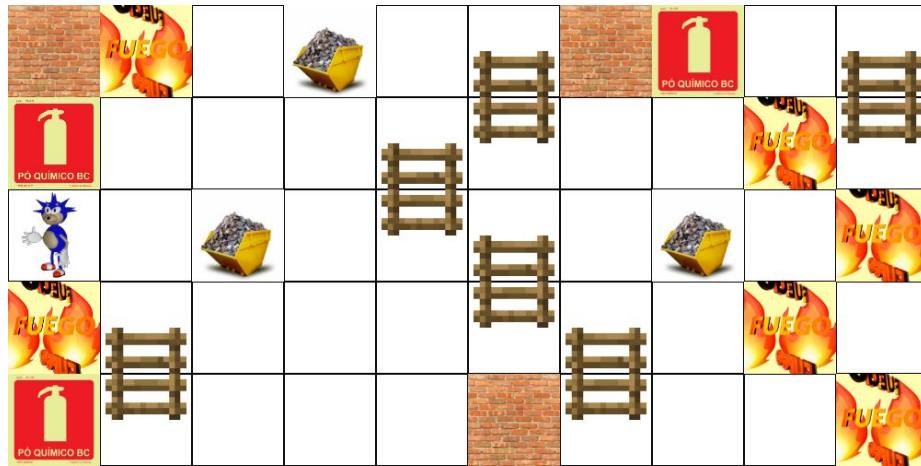
Caso 5

							 FUEGO FUEGO FUEGO
							 FUEGO FUEGO FUEGO
							 PÓ QUÍMICO BC PÓ QUÍMICO BC PÓ QUÍMICO BC

$$S =$$
$$[(0, 0, 0, [[7, 0], [7, 1], [8, 0], [8, 1], [9, 0], [9, 1]], [[7, 2], [8, 2], [9, 2]], \text{"Comeco"}),$$
[illegible][illegible]

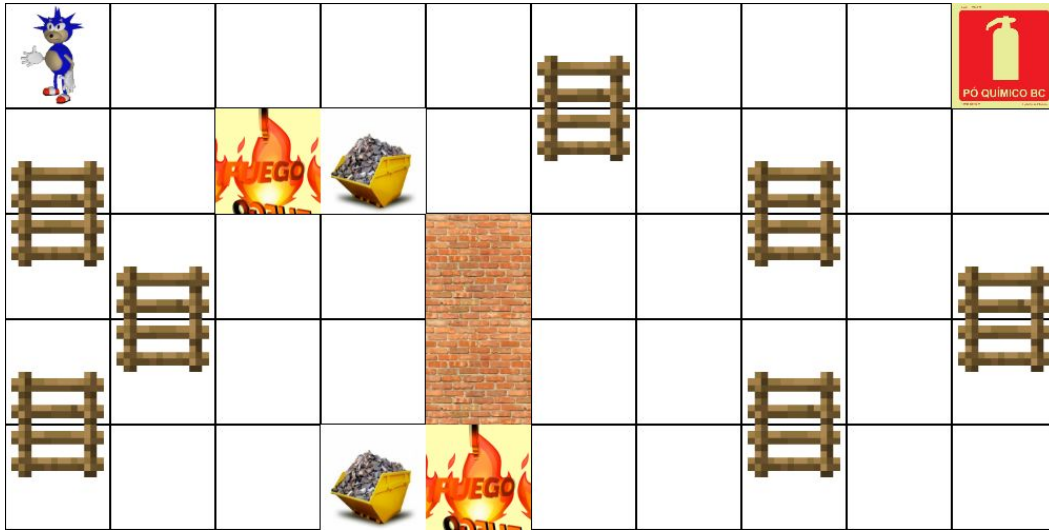
```
(2, 1, 2, [[8, 0], [9, 0]], [], "Anda"),
(2, 0, 2, [[8, 0], [9, 0]], [], "Anda"),
(3, 0, 2, [[8, 0], [9, 0]], [], "Anda"),
(4, 0, 2, [[8, 0], [9, 0]], [], "Anda"),
(5, 0, 2, [[8, 0], [9, 0]], [], "Anda"),
(6, 0, 2, [[8, 0], [9, 0]], [], "Anda"),
(7, 0, 2, [[8, 0], [9, 0]], [], "Anda"),
(8, 0, 2, [[8, 0], [9, 0]], [], "Anda"),
(8, 0, 1, [[9, 0]], [], "Apaga"),
(9, 0, 1, [[9, 0]], [], "Anda"),
(9, 0, 0, [], [], "Apaga")]
```

Caso 6


$$S =$$
[illegible]

(5, 1, 2, [[9, 2], [0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (4, 1, 2, [[9, 2], [0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (4, 2, 2, [[9, 2], [0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (5, 2, 2, [[9, 2], [0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (6, 2, 2, [[9, 2], [0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (8, 2, 2, [[9, 2], [0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (9, 2, 2, [[9, 2], [0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (9, 2, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Apaga"),
 (8, 2, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (6, 2, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (5, 2, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (5, 3, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (4, 3, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (3, 3, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (2, 3, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (1, 3, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (0, 3, 1, [[0, 3], [8, 3], [9, 4]], [[0, 4]], "Anda"),
 (0, 3, 0, [[8, 3], [9, 4]], [[0, 4]], "Apaga"),
 (1, 3, 0, [[8, 3], [9, 4]], [[0, 4]], "Anda"),
 (1, 4, 0, [[8, 3], [9, 4]], [[0, 4]], "Anda"),
 (0, 4, 0, [[8, 3], [9, 4]], [[0, 4]], "Anda"),
 (0, 4, 2, [[8, 3], [9, 4]], [], "Pega"),
 (1, 4, 2, [[8, 3], [9, 4]], [], "Anda"),
 (1, 3, 2, [[8, 3], [9, 4]], [], "Anda"),
 (2, 3, 2, [[8, 3], [9, 4]], [], "Anda"),
 (3, 3, 2, [[8, 3], [9, 4]], [], "Anda"),
 (4, 3, 2, [[8, 3], [9, 4]], [], "Anda"),
 (5, 3, 2, [[8, 3], [9, 4]], [], "Anda"),
 (6, 3, 2, [[8, 3], [9, 4]], [], "Anda"),
 (7, 3, 2, [[8, 3], [9, 4]], [], "Anda"),
 (8, 3, 2, [[8, 3], [9, 4]], [], "Anda"),
 (8, 3, 1, [[9, 4]], [], "Apaga"),
 (7, 3, 1, [[9, 4]], [], "Anda"),
 (6, 3, 1, [[9, 4]], [], "Anda"),
 (6, 4, 1, [[9, 4]], [], "Anda"),
 (4, 4, 1, [[9, 4]], [], "Anda"),
 (8, 4, 1, [[9, 4]], [], "Anda"),
 (9, 4, 1, [[9, 4]], [], "Anda"),
 (9, 4, 0, [], [], "Apaga")]

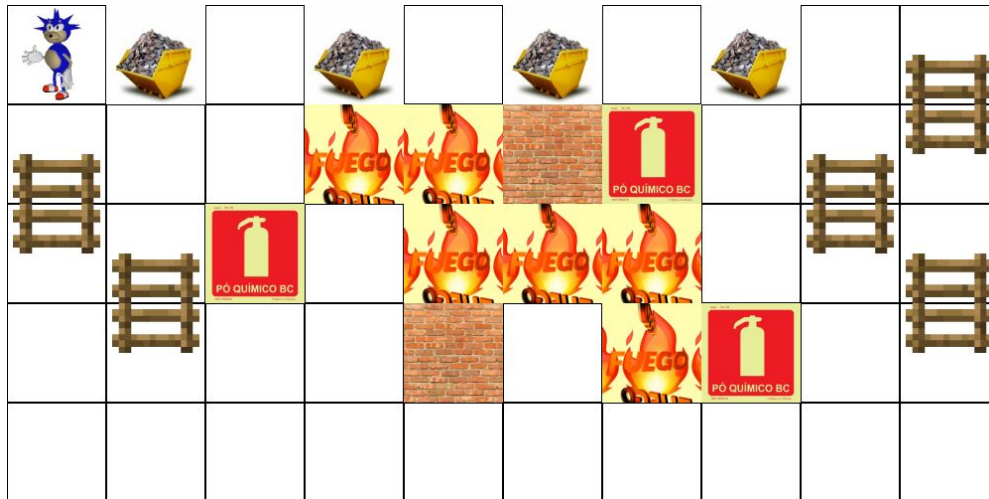
Caso 8



S =




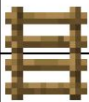



[(0, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Comeco"),
 (1, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Anda"),
 (2, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Anda"),
 (3, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Anda"),
 (4, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Anda"),
 (5, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Anda"),
 (6, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Anda"),
 (7, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Anda"),
 (8, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Anda"),
 (9, 0, 0, [[2, 1], [4, 4]], [[9, 0]], "Anda"),
 (9, 0, 2, [[2, 1], [4, 4]], [], "Pega"),
 (8, 0, 2, [[2, 1], [4, 4]], [], "Anda"),
 (7, 0, 2, [[2, 1], [4, 4]], [], "Anda"),
 (6, 0, 2, [[2, 1], [4, 4]], [], "Anda"),
 (5, 0, 2, [[2, 1], [4, 4]], [], "Anda"),
 (5, 1, 2, [[2, 1], [4, 4]], [], "Anda"),
 (6, 1, 2, [[2, 1], [4, 4]], [], "Anda"),
 (7, 1, 2, [[2, 1], [4, 4]], [], "Anda"),
 (7, 2, 2, [[2, 1], [4, 4]], [], "Anda"),
 (8, 2, 2, [[2, 1], [4, 4]], [], "Anda"),
 (9, 2, 2, [[2, 1], [4, 4]], [], "Anda"),
 (9, 3, 2, [[2, 1], [4, 4]], [], "Anda"),
 (8, 3, 2, [[2, 1], [4, 4]], [], "Anda"),
 (7, 3, 2, [[2, 1], [4, 4]], [], "Anda"),
 (7, 4, 2, [[2, 1], [4, 4]], [], "Anda"),
 (6, 4, 2, [[2, 1], [4, 4]], [], "Anda"),
 (5, 4, 2, [[2, 1], [4, 4]], [], "Anda"),
 (4, 4, 2, [[2, 1], [4, 4]], [], "Anda"),
 (4, 4, 1, [[2, 1]], [], "Apaga"),
 (2, 4, 1, [[2, 1]], [], "Anda"),
 (1, 4, 1, [[2, 1]], [], "Anda"),
 (0, 4, 1, [[2, 1]], [], "Anda"),
 (0, 3, 1, [[2, 1]], [], "Anda"),
 (1, 3, 1, [[2, 1]], [], "Anda"),
 (1, 2, 1, [[2, 1]], [], "Anda"),
 (0, 2, 1, [[2, 1]], [], "Anda"),
 (0, 1, 1, [[2, 1]], [], "Anda"),
 (1, 1, 1, [[2, 1]], [], "Anda"),
 (2, 1, 1, [[2, 1]], [], "Anda"),
 (2, 1, 0, [], [], "Apaga")

Caso 9


$$S =$$
[illegible]

(6, 3, 0, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2], [7, 3]], "Apaga"),
 (7, 3, 0, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2], [7, 3]], "Anda"),
 (7, 3, 2, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2]], "Pega"),
 (8, 3, 2, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2]], "Anda"),
 (9, 3, 2, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2]], "Anda"),
 (9, 2, 2, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2]], "Anda"),
 (8, 2, 2, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2]], "Anda"),
 (7, 2, 2, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2]], "Anda"),
 (6, 2, 2, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2]], "Anda"),
 (5, 2, 2, [[3, 1], [4, 1], [4, 2], [5, 2]], [[2, 2]], "Anda"),
 (5, 2, 1, [[3, 1], [4, 1], [4, 2]], [[2, 2]], "Apaga"),
 (4, 2, 1, [[3, 1], [4, 1], [4, 2]], [[2, 2]], "Anda"),
 (4, 2, 0, [[3, 1], [4, 1]], [[2, 2]], "Apaga"),
 (3, 2, 0, [[3, 1], [4, 1]], [[2, 2]], "Anda"),
 (2, 2, 0, [[3, 1], [4, 1]], [[2, 2]], "Anda"),
 (2, 2, 2, [[3, 1], [4, 1]], [], "Pega"),
 (1, 2, 2, [[3, 1], [4, 1]], [], "Anda"),
 (0, 2, 2, [[3, 1], [4, 1]], [], "Anda"),
 (0, 1, 2, [[3, 1], [4, 1]], [], "Anda"),
 (1, 1, 2, [[3, 1], [4, 1]], [], "Anda"),
 (2, 1, 2, [[3, 1], [4, 1]], [], "Anda"),
 (3, 1, 2, [[3, 1], [4, 1]], [], "Anda"),
 (3, 1, 1, [[4, 1]], [], "Apaga"),
 (4, 1, 1, [[4, 1]], [], "Anda"),
 (4, 1, 0, [], [], "Apaga")]

Caso 10

false.