Universidade Federal de Viçosa
Campus Rio Paranaíba

SIN 323 - Inteligência Artificial
Projeto Donkey Kong

Professor Matheus Haddad

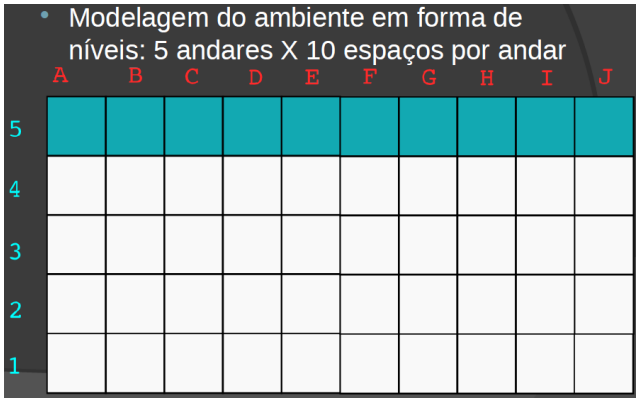
Alexandre Chain - 3924

Sumário

1. Introdução

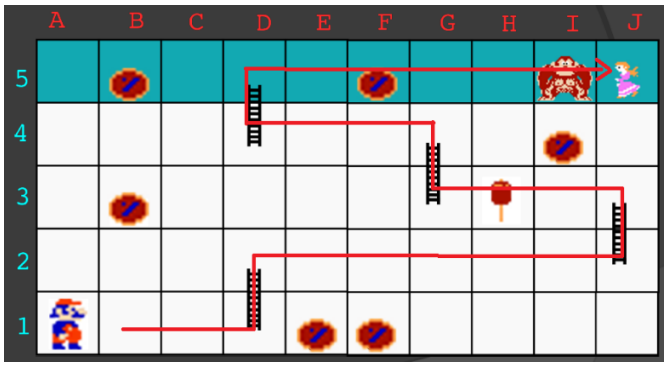
Com o intuito de aplicar os conhecimentos adquiridos nas aulas, foi desenvolvido um projeto que tem como objetivo implementar em Prolog, um ambiente, objetivo (Princesa Peach) e um protagonista (Super Mario) de modo que esses são, respectivamente:

- Ambiente:



X e Y que representam as colunas e linhas, respectivamente, limitando a área do mapa.

- Objetivo



Protagonista super mario que tem como meta percorrer o mapa em busca do martelo localizado em alguma região do mapa, após encontra-lo deverá continuar sua meta em busca do DK (chefão do mapa) ao encontra-lo deverá derrotar e chegar ao objetivo que é a princesa. Vale salientar que durante a buscas obstáculos poderão aparecer no caminho afim de dificultar o trajeto até o objetivo do Mario, tais como: Barris, Paredes e escadas espalhados pelo mapa.

Abaixo, segue o desenvolvimento do raciocínio e a lógica:

Ambientação: definido uma limitação do cenário, de modo que o personagem só poderá percorrer dentro desse limite.

Movimentação: o mario só poderá percorrer dentro das dependências do ambiente e, movimentar-se horizontalmente, onde essa movimentação depende das disposições das barreiras (sequência de dois barris, limite do ambiente e quando o mario encontra um barril seguido de escada), e verticalmente, apenas movimentando por colunas que possuem escadas.

Desse modo, como se pode ver no seguinte tópico, a base de conhecimento tem todas os fatos e regras necessárias para que o protagonista execute o seu objetivo de modo admissível, através do método de busca. Esse método de busca sempre procura, na aplicação, o menor caminho até a meta estabelecida.

2. Base de Conhecimento

%CENARIO 1

```
dk([9,5]).
objetivo([10,5]).
martelo([8,3]).

escada([4,1]).
escada([4,4]).
escada([7,3]).
escada([10,2]).

barril([2,3]).
barril([2,5]).
barril([2,3]).
barril([5,1]).
barril([6,5]).
barril([9,4]).

parede([-1,-1]). // Valores negativos porque no cenário 1 não possui parede.
```

%CENARIO 2

```
dk([9,5]).
objetivo([10,5]).
martelo([10,1]).

escada([1,2]).
escada([4,1]).
escada([4,4]).
escada([7,3]).
escada([9,1]).
escada([10,2]).

barril([3,3]).
barril([5,3]).
barril([6,5]).
barril([7,2]).
barril([8,4]).

parede([3,5]).
parede([7,1]).
```

%CENARIO 3

```
dk([9,5]).
objetivo([10,5]).
martelo([1,5]).

escada([1,3]).
escada([3,2]).
escada([3,4]).
escada([8,4]).
escada([9,1]).
escada([10,3]).

barril([6,3]).
barril([6,5]).
barril([7,2]).
barril([7,4]).
barril([8,3]).

parede([4,4]).
parede([7,5]).
```

%CENARIO 4

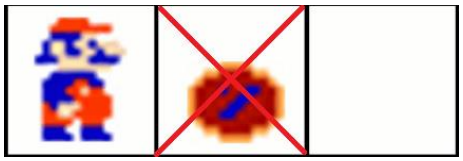
```
dk([9,5]).
objetivo([10,5]).
martelo([2,3]).

escada([1,3]).
escada([3,4]).
escada([5,1]).
escada([5,4]).
escada([8,4]).
escada([9,2]).
escada([10,3]).


barril([4,2]).
barril([4,4]).
barril([6,5]).
barril([7,2]).

parede([3,3]).
parede([7,5]).
```

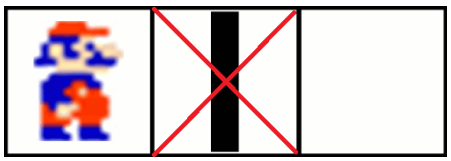
```
%Definicao de Estados


s([X,Y],[X1,Y]):- X<10 ,X1 is X+1,not(parede([X1,Y])),not(barril([X1,Y])).
s([X,Y],[X1,Y]):- X>1 ,X1 is X-1,not(parede([X1,Y])),not(barril([X1,Y])).

s([X,Y],[X1,Y]):- X<9 ,X1 is X+2,X2 is X+1,Y1 is Y-
1,barril([X2,Y]),not(parede([X1,Y])),not(barril([X1,Y])),not(escada([X1,Y])),not(escada([X1,Y1])
), not(dk([X1,Y])).

s([X,Y],[X1,Y]):- X>2 ,X1 is X-2,X2 is X-1,Y1 is Y-
1,barril([X2,Y]),not(parede([X1,Y])),not(barril([X1,Y])),not(escada([X1,Y])),not(escada([X1,Y1])
), not(dk([X1,Y])).

s([X,Y],[X,Y1]):- Y<5 ,Y1 is Y+1,escada([X,Y]),not(barril([X,Y1])).
s([X,Y],[X,Y1]):- Y>1 ,Y1 is Y-1,escada([X,Y1]),not(barril([X,Y1])).


tem_martelo([Cabeca|_]):-martelo(Cabeca).
tem_martelo([_|Cauda]):-tem_martelo(Cauda).


iniciar(X,Y) :- bo([[[X,Y]]],Solucao), reverse(Solucao,Solucao2),write(['Caminho:',Solucao2]).
iniciar(X,Y) :- bm([[[X,Y]]],[Cabeca|Cauda]),bo([[Cabeca]],Solucao),
concatena(Solucao,Cauda,Solucao1),reverse(Solucao1,Solucao2),write(['Caminho:',Solucao
2]).

bm([[Estado|Caminho]|_],[Estado|Caminho]) :- martelo(Estado).
bm([Primeiro|Outros], Solucao) :- estende2(Primeiro,Sucessores),
concatena(Outros,Sucessores,NovaFronteira), bm(NovaFronteira,Solucao).

bo([[Estado|Caminho]|_],[Estado|Caminho]) :- objetivo(Estado).
bo([Primeiro|Outros], Solucao) :- estende1(Primeiro,Sucessores),
concatena(Outros,Sucessores,NovaFronteira), bo(NovaFronteira,Solucao).

estende1([Estado|Caminho],ListaSucessores):-
        bagof(
                [Sucessor,Estado|Caminho],
                (s(Estado,Sucessor),not((pertence(Sucessor,[Estado|Caminho])))),
                ListaSucessores),!.
estende1( _ ,[]).

estende2([Estado|Caminho],ListaSucessores):-
        bagof(
                [Sucessor,Estado|Caminho],

        (s(Estado,Sucessor),not((pertence(Sucessor,[Estado|Caminho]));(dk(Sucessor),(not(te
m_martelo([Estado|Caminho]))))))),
                ListaSucessores),!.
estende2( _ ,[]).
```

```
pertence(X,[X|_]).
pertence(X,[_|Cauda]) :- pertence(X,Cauda).

concatena([],L,L).
concatena([Cabeca|Cauda],L,[Cabeca|Resultado]):- concatena(Cauda,L,Resultado).
```

3. Explicação

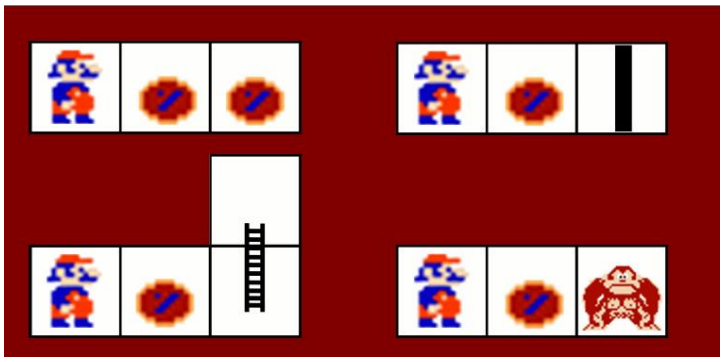
Segue abaixo a explicação das principais regras e fatos presentes no projeto:





Movimentação livre em X : sujeito a paredes e barril, o personagem anda livremente se não possuir obstaculos como barril e parede, tanto pela direita ou pela esquerda.

```
s([X,Y],[X1,Y]):- X<10 ,X1 is X+1,not(parede([X1,Y])),not(barril([X1,Y])).
s([X,Y],[X1,Y]):- X>1 ,X1 is X-1,not(parede([X1, Y])),not(barril([X1,Y])).
```



Movimentação em X de salto de barris: sujeito a restricoes nos saltos caso possua dois barris sucessivos, barril seguido de parede, barril seguido de escada ou barril seguido do Donkey Kong indo para direita.

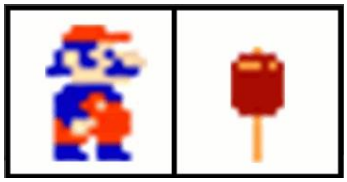
s([X,Y],[X1,Y]):- X<9 ,X1 is X+2,X2 is X+1,Y1 is Y-1,barril([X2,Y]),not(parede([X1,Y])),not(barril([X1,Y])),not(escada([X1,Y])),not(escada([X1,Y1])), not(dk([X1,Y])).

Movimentação em X de salto de barris: sujeito a restricoes nos saltos caso possua dois barris sucessivos, barril seguido de parede, barril seguido de escada ou barril seguido do Donkey Kong indo para esquerda.

s([X,Y],[X1,Y]):- X>2 ,X1 is X-2,X2 is X-1,Y1 is Y-1,barril([X2,Y]),not(parede([X1,Y])),not(barril([X1,Y])),not(escada([X1,Y])),not(escada([X1,Y1])), not(dk([X1,Y])).

Movimentacao em Y pelas escadas: bloqueado quando tem barril na ponta tanto na subida ou descida das escadas.

s([X,Y],[X,Y1]):- Y<5 ,Y1 is Y+1,escada([X,Y]),not(barril([X,Y1])).
s([X,Y],[X,Y1]):- Y>1 ,Y1 is Y-1,escada([X,Y1]),not(barril([X,Y1])).



Verifica se tem martelo para enfrentar DK

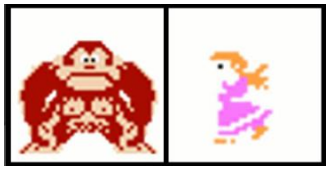
tem_martelo([Cabeca|_]):-martelo(Cabeca).
tem_martelo([_|Cauda]):-tem_martelo(Cauda).

O usuário inicia a função iniciar(X,Y) onde deverá passar a localização do Mario. Após isso é iniciado a busca pelo método “bo” e “bm”, que são adaptações da busca em largura, assegurando o menor caminho na busca cega para a resolução do problema; onde “bm” faz a busca do ponto inicial até a obtenção do martelo, e “bo” continua a busca até a princesa.

iniciar(X,Y) :- bm([[[X,Y]]],[Cabeca|Cauda]),bo([[Cabeca]],Solucao), concatena(Solucao,Cauda,Solucao1),reverse(Solucao1,Solucao2),write(['Caminho:',Solucao2]).

- A função **bm([[[X,Y]]],[Cabeca|Cauda])** irá finalizar quando **Cabeca** estiver com a posição do martelo e **Cauda** estiver com o caminho percorrido.
-A função **bo([[Cabeca]],Solucao)** irá iniciar com a posição inicial no martelo, e finalizará quando estiver no Objetivo, armazenando em **Solucao** o caminho percorrido.
- A função **concatena(Solucao,Cauda,Solucao1)** irá concatenar a **Cauda** com **Solucao,** unindo o caminho [nicial -> martelo] com [martelo->Objetivo].
- A função **reverse(Solucao1,Solucao2)** serve para deixar o resultado de **concatena** na ordem que foi percorrida para apresentação final, e armazena-lo em Solucao2.
- E por fim **write(['Caminho:',Solucao2])** printa na tela o resultado encontrado.

Busca do martelo e objetivo



```
bm([[Estado|Caminho]|_],[Estado|Caminho]) :- martelo(Estado).
bm([Primeiro|Outros], Solucao) :- estende2(Primeiro,Sucessores),
concatena(Outros,Sucessores,NovaFronteira), bm(NovaFronteira,Solucao).

bo([[Estado|Caminho]|_],[Estado|Caminho]) :- objetivo(Estado).
bo([Primeiro|Outros], Solucao) :- estende1(Primeiro,Sucessores),
concatena(Outros,Sucessores,NovaFronteira), bo(NovaFronteira,Solucao).
```

As funções **bo** e **bm** recebem como parâmetros uma lista com a posição inicial do agente, a lista de fronteira atual dividida em cabeça e cauda e as posições já exploradas.
Esta função opera recursivamente, de modo que retornará a posição atual do agente com todo o caminho traçado até aquela posição, caso ela tenha atingido o Objetivo (Posição do **Martelo** ou **Princesa**).
O passo recursivo, ocorre quando o **Estado** não unifica martelo(**Estado**) ou objetivo(**Estado**) fazendo com que a função estenda este estado a seus Sucessores pela função "**estende1**" e "**estende2**", concatene o final da lista de fronteira com a função "concatena" e continue com a execução do algoritmo, onde os próximos estados da lista de fronteira serão explorados.

```
estende1([Estado|Caminho],ListaSucessores):-
        bagof(
                [Sucessor,Estado|Caminho],
                (s(Estado,Sucessor),not((pertence(Sucessor,[Estado|Caminho])))),
                ListaSucessores),!.
estende1( _ ,[]).

estende2([Estado|Caminho],ListaSucessores):-
        bagof(
                [Sucessor,Estado|Caminho],

        (s(Estado,Sucessor),not((pertence(Sucessor,[Estado|Caminho]));(dk(Sucessor),(
not(tem_martelo([Estado|Caminho])))))),
                ListaSucessores),!.
estende2( _ ,[]).
```

Na função "estende1", o goal de bagof garante principalmente 2 condições:

```
s(Estado,Sucessor),not((pertence(Sucessor,[Estado|Caminho])))),
```

1. Que o movimento seja válido, garantido por s()
2. Que o caminho não gere ciclos inválidos, retornando por onde já passou.

Já na função “estende2”, o goal de bagof garante além das 2 condições, uma terceira:

s(Estado,Sucessor),not((pertence(Sucessor,[Estado|Caminho]));(dk(Sucessor),(not(tem_martelo([Estado|Caminho])))))

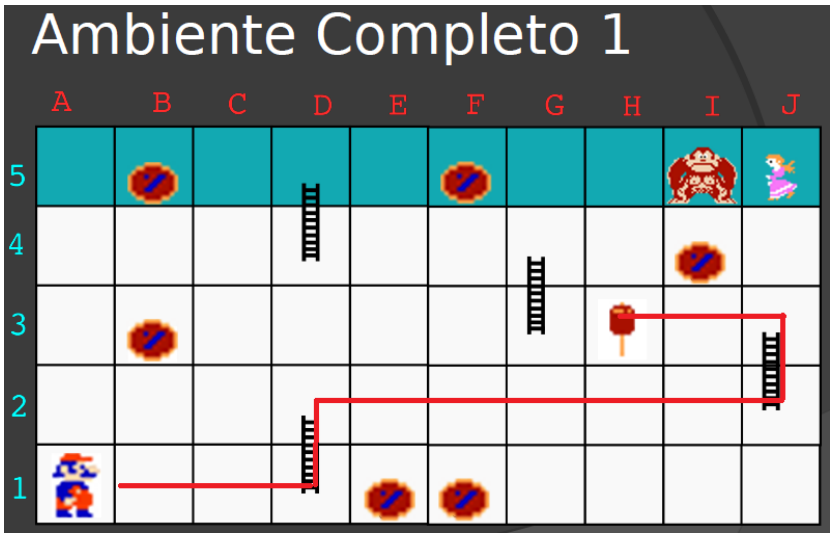
3. Que Mario não chegue ao DonkeyKong se não tiver passado pelo martelo em seu caminho.

pertence(X,[X|_]).
pertence(X,[_|Cauda]) :- pertence(X,Cauda).

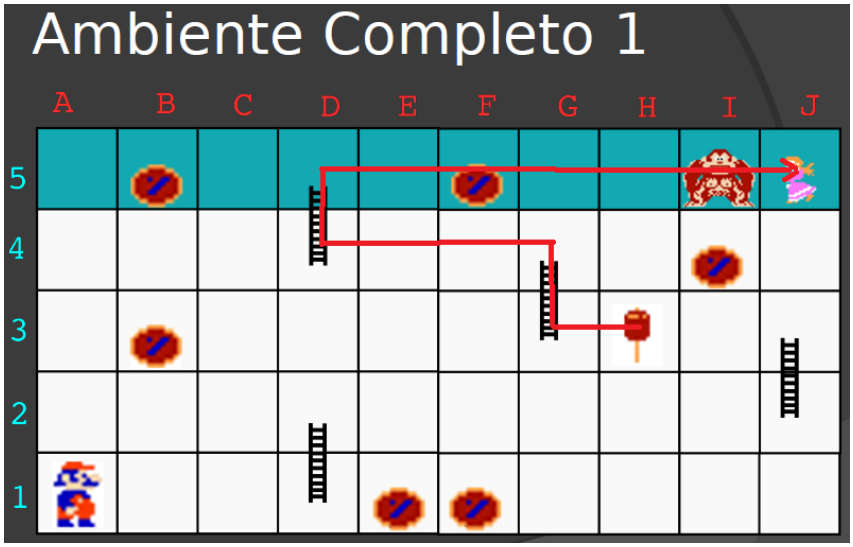
concatena([],L,L).
concatena([Cabeca|Cauda],L,[Cabeca|Resultado]):- concatena(Cauda,L,Resultado).

4. Exemplificação

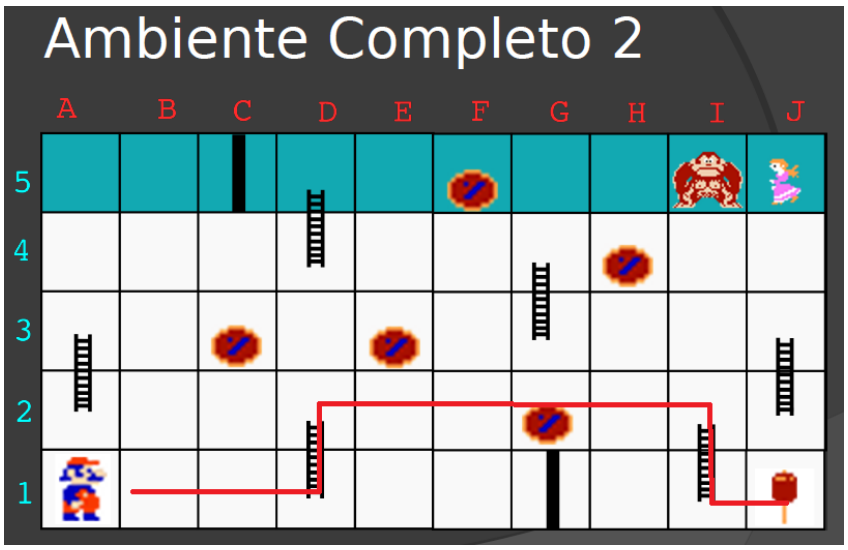
Definição de cenário de exemplo:



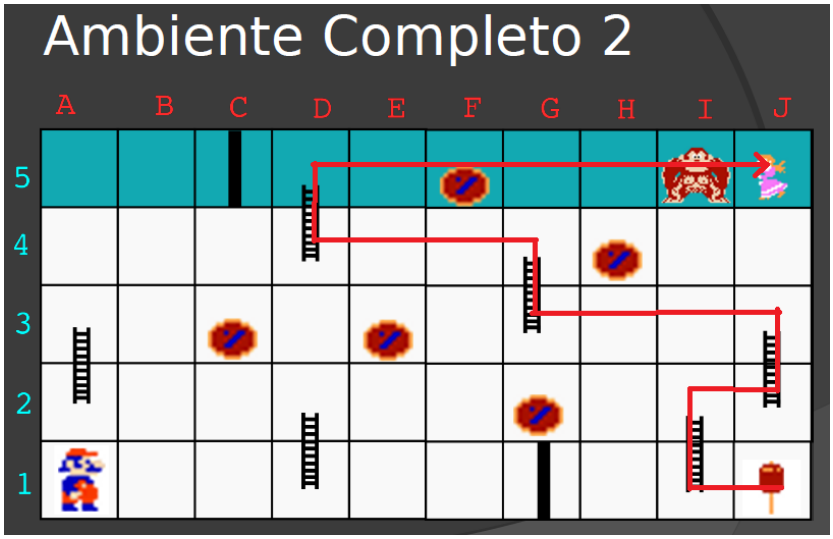

```
?- iniciar(1,1).
[Caminho=[[1,1],[2,1],[3,1],[4,1],[4,2],[5,2],[6,2],[7,2],[8,2],[9,2],[10,2],[10,3],[9,3],
[8,3],[7,3],[7,4],[6,4],[5,4],[4,4],[4,5],[5,5],[7,5],[8,5],[9,5],[10,5]]]
true
```



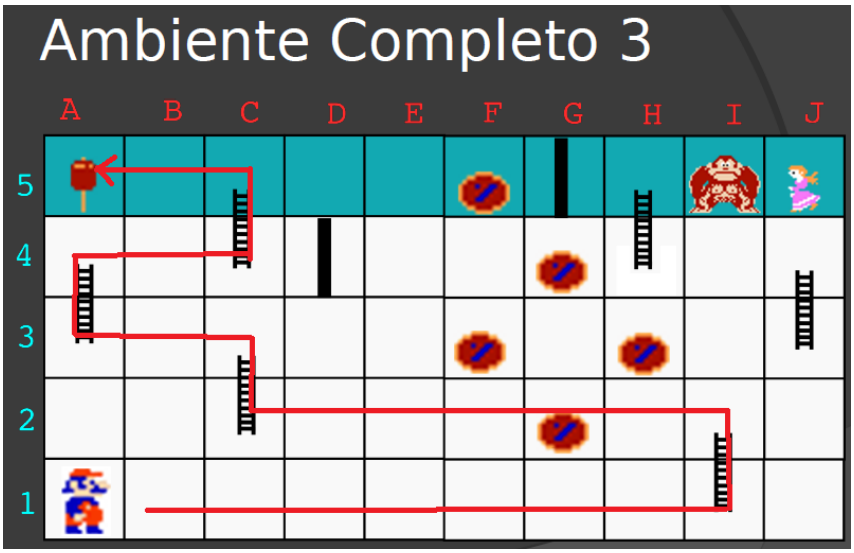

```
?- iniciar(1,1).
[Caminho:,[[1,1],[2,1],[3,1],[4,1],[4,2],[5,2],[6,2],[7,2],[8,2],[9,2],[10,2],[10,3],[9,3],
[8,3],[7,3],[7,4],[6,4],[5,4],[4,4],[4,5],[5,5],[7,5],[8,5],[9,5],[10,5]]]
true
```



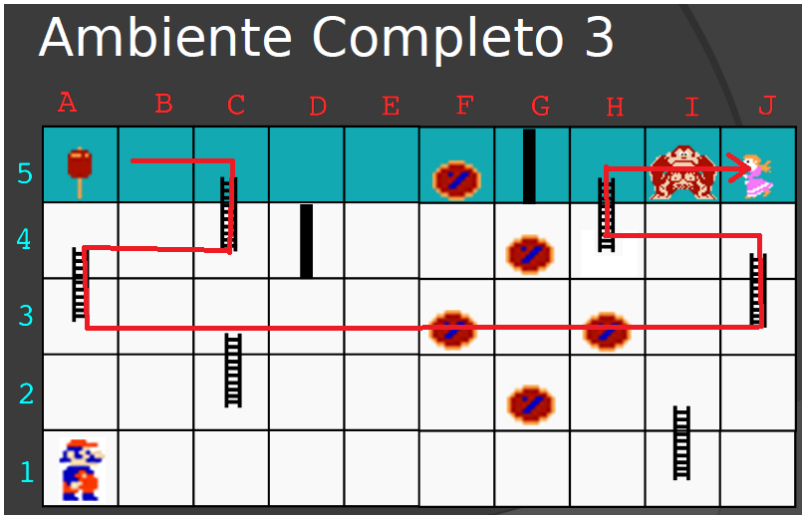

```
?- iniciar(1,1).
[Caminho:,[[1,1],[2,1],[3,1],[4,1],[4,2],[5,2],[6,2],[8,2],[9,2],[9,1],[10,1],[9,1],[9,2],[
10,2],[10,3],[9,3],[8,3],[7,3],[7,4],[6,4],[5,4],[4,4],[4,5],[5,5],[7,5],[8,5],[9,5],[10,5]
]]
true
```



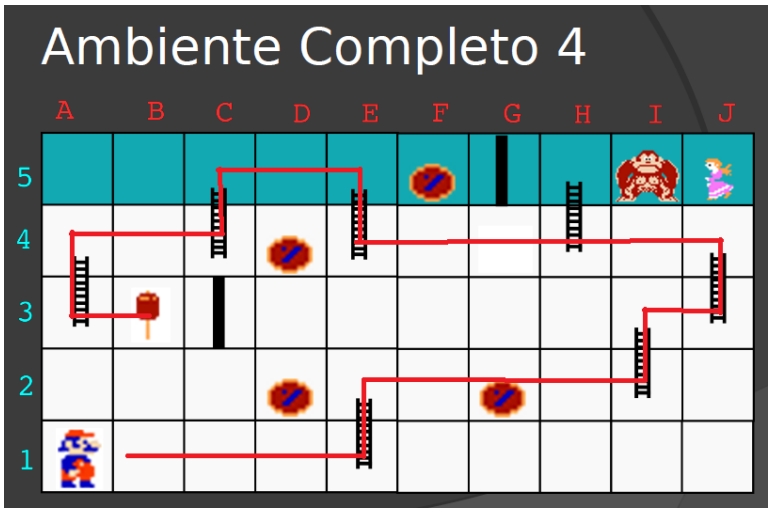

```
?- iniciar(1,1).
[Caminho:,[[1,1],[2,1],[3,1],[4,1],[4,2],[5,2],[6,2],[8,2],[9,2],[9,1],[10,1],[9,1],[9,2],[
10,2],[10,3],[9,3],[8,3],[7,3],[7,4],[6,4],[5,4],[4,4],[4,5],[5,5],[7,5],[8,5],[9,5],[10,5]
]]
true
```



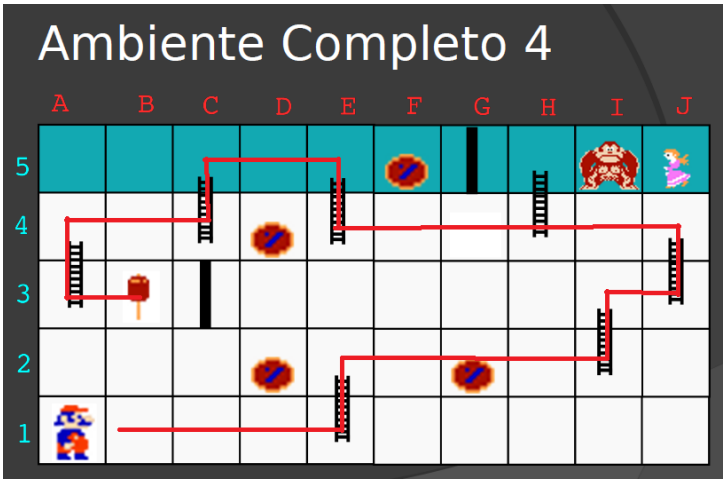

```
?- iniciar(1,1).
[Caminho:,[[1,1],[2,1],[3,1],[4,1],[5,1],[6,1],[7,1],[8,1],[9,1],[9,2],[8,2],[6,2],[5,2],[4,2],[3,2]
[3,3],[2,3],[1,3],[1,4],[2,4],[3,4],[3,5],[2,5],[1,5],[2,5],[3,5],[3,4],[2,4],[1,4],[1,3],[2,3],[3,3]
,[4,3],[5,3],[7,3],[9,3],[10,3],[10,4],[9,4],[8,4],[8,5],[9,5],[10,5]]]
true .

?-
```




```
?- iniciar(1,1).
[Caminho:,[[1,1],[2,1],[3,1],[4,1],[5,1],[6,1],[7,1],[8,1],[9,1],[9,2],[8,2],[6,2],[5,2],[4,2],[3,2],
[3,3],[2,3],[1,3],[1,4],[2,4],[3,4],[3,5],[2,5],[1,5],[2,5],[3,5],[3,4],[2,4],[1,4],[1,3],[2,3],[3,3]
,[4,3],[5,3],[7,3],[9,3],[10,3],[10,4],[9,4],[8,4],[8,5],[9,5],[10,5]]]
true .

?-
```




```
?- iniciar(1,1).
[Caminho:,[[1,1],[2,1],[3,1],[4,1],[5,1],[5,2],[6,2],[8,2],[9,2],[9,3],[10,3],[10,4],[9,4],
[8,4],[7,4],[6,4],[5,4],[5,5],[4,5],[3,5],[3,4],[2,4],[1,4],[1,3],[2,3],[1,3],[1,4],[2,4],[
3,4],[3,5],[4,5],[5,5],[5,4],[6,4],[7,4],[8,4],[8,5],[9,5],[10,5]]]
true
```

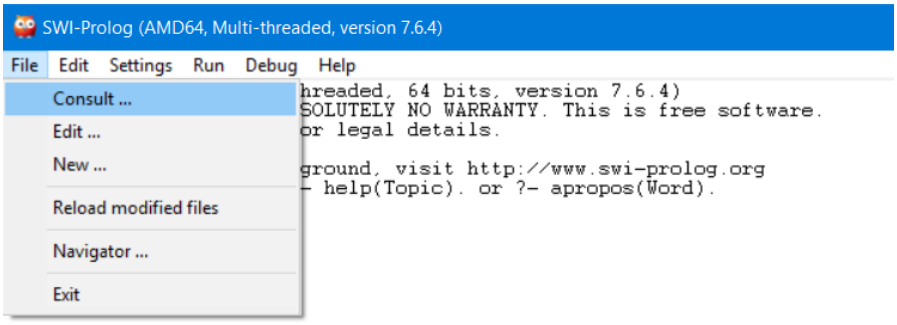


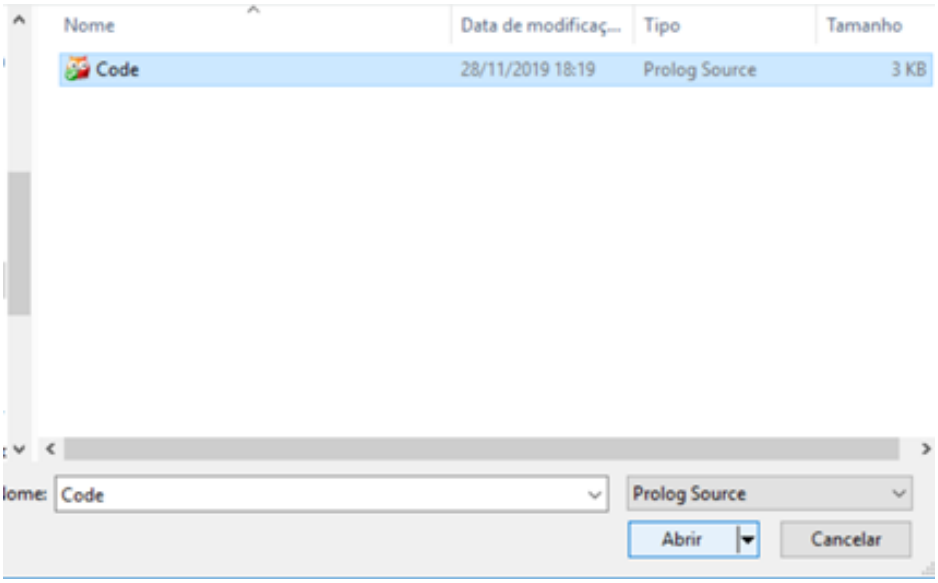

```
?- iniciar(1,1).
[Caminho:,[[1,1],[2,1],[3,1],[4,1],[5,1],[5,2],[6,2],[8,2],[9,2],[9,3],[10,3],[10,4],[9,4],
[8,4],[7,4],[6,4],[5,4],[5,5],[4,5],[3,5],[3,4],[2,4],[1,4],[1,3],[2,3],[1,3],[1,4],[2,4],[
3,4],[3,5],[4,5],[5,5],[5,4],[6,4],[7,4],[8,4],[8,5],[9,5],[10,5]]]
true
```

5 Manual de execução

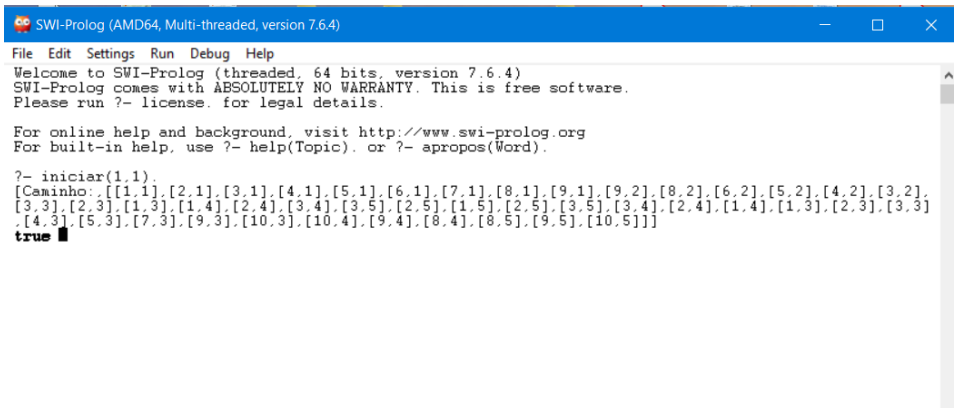
Para a execução do trabalho, é necessário ter instalado em sua máquina, o Prolog, essa ferramenta possui distribuição para vários sistemas operacionais, que pode ser baixado no seguinte link: https://www.swi-prolog.org/Download.html

Desse modo, adicione a nossa base de conhecimento em um arquivo com a extensão **.pl**, e realiza uma consulta dessa base, agora o Prolog possui todo o conhecimento necessário para realizar as operações, então, execute o seguinte comando para que a aplicação retorne o caminho completo realizado pelo protagonista. Abra o Prolog em seguida na sessão “File” clique na opção Consult...





Em seguida uma nova janela ira abrir, vá até o diretório onde encontra o código do Arquivo .pl e selecione a opção abrir como apresenta na seguinte imagem.



Após abrir o arquivo, o prolog estará pronto para o uso. Agora é só usar o comando: iniciar(1,1). para executar o código e realizar a busca.