Henrique Alves (5968), José Eduardo (5964)

Trabalho Final de Inteligência Artificial

Brasil

28 de abril de 2021

Sumário

1 Introdução

Neste projeto, foi implementado um conjunto de regras em Prolog que modela um ambiente do jogo Popeye, onde, o personagem Popeye deve recolher todos os corações presentes no ambiente, em seguida, pegar o espinafre e derrotar o Brutus.

1.1 Busca em Largura

Utilizamos da Busca em Largura para realizar a busca em nossos ambientes, pois, a Busca em Largura sempre encontrará o menor caminho entre o estado-inicial e o objetivo final. Nossa Busca em Largura recebe a posição inicial do problema(posição do Popeye inicialmente), as posições das escadas, garrafas, a posição do Brutus, uma variável de controle para saber se o Popeye pegou o espinafre, as posições dos corações e uma variável que armazena a Solução do objetivo.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Busca em largura                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

busca_em_largura(Inicio, Escadas, Garrafas, Brutus, TemEspinafre, Objetivo, Solucao) :-
    b_l([[Inicio]], Escadas, Garrafas, Brutus, TemEspinafre, Objetivo, Solucao).

b_l([[Estado|Caminho]|_], _, _, _, _, Objetivo, [Estado|Caminho]) :-
    Objetivo == Estado.

b_l([Primeiro|Restante], Escadas, Garrafas, Brutus, TemEspinafre, Objetivo, Solucao) :-
    extende(Primeiro, Sucessores, Escadas, Garrafas, Brutus, TemEspinafre),
    concatenar(Restante, Sucessores, NovaFronteira),
    b_l(NovaFronteira, Escadas, Garrafas, Brutus, TemEspinafre, Objetivo, Solucao).

extende([Estado|Caminho], ListaSucessores, Escadas, Garrafas, Brutus, TemEspinafre) :-
    bagof([Sucessor, Estado|Caminho],
    (prox(Estado, Escadas, Garrafas, Brutus, TemEspinafre, Sucessor),
    \+ pertence(Sucessor, [Estado|Caminho])),ListaSucessores),
    !.
extende( _ ,[], _,_,_,_).
```

Figura 1 – Implementação da busca em largura

1.2 Cenários

Criamos três cenários fixos, um cenário fácil, onde o Popeye tem que buscar três corações antes de pegar o espinafre, um cenário médio, onde o Popeye busca quatro corações e um cenário difícil, onde ele precisa de cinco corações para pegar o espinafre e então derrotar o Brutus. Cada cenário possui escadas e garrafas posicionadas em locais diferentes, onde, os cenários mais difíceis possuem mais obstáculos e os mais fáceis, menos obstáculos. Nas figuras 2, 3 e 4 estão ilustrados nossos cenários criados, onde as posições do Popeye são as posições que podem ser escolhidas para iniciar a busca, os corações,

o espinafre e o Brutus estão posicionados em suas possíveis posições(mais detalhes na próxima seção).

Figura 2 – Cenário fácil

Figura 3 – Cenário médio

Figura 4 – Cenário difícil

1.3 Posições

Em nossos ambientes estáticos, o jogador pode escolher sua posição inicial, desde que seja no primeiro andar e não tenha nenhum obstáculo(escadas ou garrafas) no local escolhido, por exemplo, na figura 4, o jogador não pode escolher as posições 1, 4 e 7, pois, existem obstáculos(escadas e garrafas) que o impedem de serem escolhidas. Os corações são fixos em um andar, porém, a posição dos mesmos em cada andar é aleatória em um determinado intervalo. O espinafre funciona da mesma forma. A posição do Brutus(objetivo final) também é aleatória em um certo intervalo, porém, ele aparece apenas no último andar(5º andar).

1.4 Movimentação

Como definido na descrição do projeto, o Popeye pode mover livremente desde que não haja duas garrafas consecutivas, uma garrafa e o Brutus consecutivamente ou caso ele encontre o Brutus, mas não possui o espinafre. As escadas são utilizadas para movimentar-se de andar para andar, tanto subindo quanto descendo, Popeye também pode passar por uma escada sem precisar subir ou descer de andar. Caso Popeye encontre uma garrafa com os dois quadrados adjacentes vazios, ele poderá saltar sobre a garrafa, passando para o outro lado.

2 Funcionalidades

2.1 Fatos e regras utilizadas

2.1.1 Regras 'reverse' e 'insereFinal'

A regra 'reverse' é utilizada para inverter uma Lista, passando o primeiro elemento para o final da Lista através do uso da regra 'insereFinal' que, basicamente, insere um elemento no final de uma Lista.

```
%insere no final de uma lista(utilizada na regra reverse)
insereFinal(X,[],[X]).
insereFinal(X,[Y|L], [Y|W]):-   insereFinal(X, L, W).

%inverte as posições de uma lista
reverse([], []).
reverse([X], [X]).
reverse([X|R], L):- reverse(R, W), insereFinal(X, W, L).
```

Figura 5 – Fatos e regras 'reverse' e 'insereFinal'

2.1.2 Regra 'pegaPrimeiro'

A regra 'pegaPrimeiro' pega o primeiro elemento de uma Lista, ou seja, a cabeça de uma Lista.

```
%pega o primeiro elemento de uma lista
pegaPrimeiro([Cabeca|_], Cabeca).
```

Figura 6 – Regra 'pegaPrimeiro'

2.1.3 Regra 'pertence'

A regra 'pertence' verifica se um elemento pertence a uma Lista, verificando se o elemento pertence a cabeça ou a cauda da Lista.

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

Figura 7 – Regra 'pertence'

2.1.4 Regra 'concatenar'

A regra 'concatenar' junta duas Listas em uma terceira Lista. Se o primeiro argumento é uma lista vazia, então o segundo e o terceiro argumentos são os mesmos. Se o primeiro argumento é uma Lista não vazia, então ela tem uma cabeça e uma cauda da forma [Elem|Lista1], concatenar [Elem|Lista1] com uma segunda lista Lista2 resulta na lista [Elem|Lista3], onde Lista3 é a concatenação de Lista1 e Lista2.

```
%concatena duas listas
concatenar([], Lista, Lista).
concatenar([Elem|Lista1], Lista2, [Elem|Lista3]) :-
    concatenar(Lista1, Lista2, Lista3).
```

Figura 8 – Regra 'concatenar'

2.1.5 Regra 'calcular'

Nessa regra, calculamos a pontuação do jogador ao coletar um coração. A variável 'Andar1' recebe o número do andar que o coração foi coletado, em seguida, multiplicamos por 100 e armazenamos o resultado no nosso predicado dinâmico 'pontuacao'.

```
%calcula os pontos dos corações adiquiridos
calcular(Andar) :-
    Andar1 is (Andar + 1),
    Pts is (Andar1 * 100),
    retract(pontuacao(Pontos)),
    PontuacaoFinal is (Pontos + Pts),
    asserta(pontuacao(PontuacaoFinal)).
```

Figura 9 – Regra 'calcular'

2.1.6 Predicados dinâmicos

Utilizamos predicados dinâmicos pois o SWI Prolog só permite que um predicado seja instanciado uma vez, então, com o predicado dinâmico conseguimos realizar o cálculo das pontuações e armazenar a última posição(utilizada para iniciar a busca ao espinafre partindo da posição que o Popeye parou). Para a utilização dos mesmos, precisamos inicializá-los, iniciando a pontuação com '0' e a ultimapos com uma lista vazia '[]' como demonstrado na Figura 11.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        Predicados dinâmicos             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- dynamic pontuacao/1.
:- dynamic ultimapos/1.
```

Figura 10 – Criando os predicados dinâmicos

```
%inicializa os predicados dinâmicos
inicializa :-
    asserta(pontuacao(0)),
    asserta(ultimapos([])).
```

Figura 11 – Inicializando os predicados dinâmicos

2.1.7 Main

Na nossa regra 'main' é onde as buscas são chamadas, primeiro inicializamos os predicados dinâmicos, então, chamamos a regra 'coleta_coracoes' para coletarmos todos os corações passados pela lista 'Coracoes'. Após coletar todos corações iremos coletar o espinafre, imprimimos o caminho para o espinafre e partimos para o Brutus, quando o mesmo for derrotado, imprimimos o caminho e a pontuação final do Popeye.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Main(realiza a busca)                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

main(Inicio, Escadas, Garrafas, Espinafre, Brutus, Coracoes, Solucao) :-
    inicializa,
    coleta_coracoes(Coracoes, Inicio, Escadas, Garrafas, Brutus),
    ultimapos(UltimaPos),
    busca_em_largura(UltimaPos, Escadas, Garrafas, Brutus, 0, Espinafre, CaminhoEspin
    reverse(CaminhoEspinafre, CaminhoECerto),
    write('Espinafre na posicao '),
    write(Espinafre),
    write(' coletado pelo seguinte caminho:'),
    writeln(CaminhoECerto),
    busca_em_largura(Espinafre, Escadas, Garrafas, Brutus, 1, Brutus, CaminhoBrutus),
    reverse(CaminhoBrutus, CaminhoBCerto),
    write('Brutus na posicao '),
    write(Brutus),
    write(' derrotado pelo seguinte caminho:'),
    writeln(CaminhoBCerto),
    write('Pontuacao final: '),
    pontuacao(Pontuacao),
    write(Pontuacao).
```

Figura 12 – Codificação da 'main'

2.2 Iniciando a busca

Para que uma busca seja iniciada, o usuário pode criar o seu próprio cenário ou ele pode escolher uma das três dificuldades(fácil, médio ou difícil), e então, selecionar a posição de início do Popeye, lembrando que ele não pode escolher uma posição que não esteja vazia, a posição dos corações, do espinafre e do Brutus são geradas aleatoriamente em um determinado intervalo, o Brutus sempre será encontrado no último andar(5º andar).

Para escolher a dificuldade, basta digitar: '?- facil(0).', '?- medio(0).' ou '?- dificil(0).', onde na posição que o número zero se encontra, é onde deve ser passada a posição que o jogador deseja iniciar a busca, se atentando aos obstáculos do primeiro andar. Caso o usuário deseja criar o seu cenário, ele deve inserir os valores manualmente através da pesquisa: '? - main([posição do Popeye], [Escadas], [Garrafas], [Espinafre], [Brutus], [Coracoes], Solucao)', um exemplo se encontra na Figura 13.

Caso o jogador insira uma posição onde existe um obstáculo, temos uma sequencia de verificações que não deixará o ambiente ser criado, retornando '**false.**', exemplo na imagem 18. Caso contrário, a busca é realizada e é impresso na tela o caminho encontrado pela busca e a pontuação total do jogador, que é definida pela soma da pontuação de todos os corações coletados, o cálculo da pontuação de um coração é feito observando o andar de cada coração multiplicado por 100, Ex: um coração é encontrado no segundo andar, a pontuação realizada foi de 2*100 = 200, e então, retorna '**true.**'. Exemplo na imagem 17.

```
?- main([0,0], [[0,3],[0,6],[1,2],[2,5],[3,3]],
 [[0,4],[0,5],[2,1],[3,8],[4,1],[4,5]], [4,2],
[4,9], [[1,9],[2,9],[3,7]], Solucao).
```

Figura 13 – Exemplo de um cenário sendo criado pelo usuário.

```
%chama a main com a predefinição da fase fácil(corações, espinafre e Brutus são aleatórios)
facil(Inicio) :-
    (Inicio >= 0,
    Inicio < 10,
    Inicio =\= 3,
    Inicio =\= 4,
    Inicio =\= 5,
    Inicio =\= 6 ->
    random(3, 9, Cora1),
    random(6, 9, Cora2),
    random(4, 7, Cora3),
    random(2, 4, Espina),
    random(7, 9, Brut),
    main([0,Inicio], [[0,3],[0,6],[1,2],[2,5],[3,3]], [[0,4],[0,5],[2,1],[3,8],[4,1],[4,5]],
    [4,Espina], [4,Brut], [[1,Cora1], [2,Cora2], [3,Cora3]], Solucao)
    ).
```

Figura 14 – Modo fácil sendo inicializado

```
%chama a main com a predefinição da fase médio(corações, espinafre e Brutus são aleatórios)
medio(Inicio) :-
    (Inicio >= 0,
    Inicio < 10,
    Inicio =\= 2,
    Inicio =\= 5,
    Inicio =\= 6,
    Inicio =\= 7 ->
    random(1, 4, Cora1),
    random(0, 2, Cora2),
    random(6, 8, Cora3),
    random(4, 7, Cora4),
    random(6, 7, Espina),
    random(4, 9, Brut),
    main([0,Inicio], [[0,2],[0,7],[1,0],[2,9],[3,3]], [[0,5],[0,6],[1,5],[2,3],[2,5],[3,8],
    [4,1],[4,2]], [1,Espina], [4,Brut], [[1,Cora1], [2,Cora2], [2,Cora3], [3,Cora4]], Solucao)
    ).
```

Figura 15 – Modo médio sendo inicializado

```
%chama a main com a predefinição da fase difícil(corações, espinafre e Brutus são aleatórios)
dificil(Inicio) :-
    (Inicio >= 0,
    Inicio < 10,
    Inicio =\= 1,
    Inicio =\= 4,
    Inicio =\= 7 ->
    random(2, 5, Cora1),
    random(6, 8, Cora2),
    random(3, 6, Cora3),
    random(0, 2, Cora4),
    random(1, 3, Cora5),
    random(0, 1, Espina),
    main([0,Inicio], [[0,4],[1,8],[2,0],[2,9],[3,1],[3,6]], [[0,1],[0,7],[1,6],[2,4],
    [2,5],[3,3],[3,4],[3,8],[4,7]], [1,Espina], [4,9], [[1,Cora1], [2,Cora2], [4,Cora3],
    [4,Cora4], [2,Cora5]], Solucao)
    ).
```

Figura 16 – Modo difícil sendo inicializado

```
?- facil(0).
Coracao na posicao [1,6] coletado pelo seguinte caminho:[
[0,0],[0,1],[0,2],[0,3],[1,3],[1,4],[1,5],[1,6]]
Pontuacao atual: 200
Coracao na posicao [2,8] coletado pelo seguinte caminho:[
[1,6],[1,5],[1,4],[1,3],[1,2],[2,2],[2,3],[2,4],[2,5],[2,
6],[2,7],[2,8]]
Pontuacao atual: 500
Coracao na posicao [3,6] coletado pelo seguinte caminho:[
[2,8],[2,7],[2,6],[2,5],[3,5],[3,6]]
Pontuacao atual: 900
Espinafre na posicao [4,3] coletado pelo seguinte caminho
:[[3,6],[3,5],[3,4],[3,3],[4,3]]
Brutus na posicao [4,7] derrotado pelo seguinte caminho:[
[4,3],[4,4],[4,6],[4,7]]
Pontuacao final: 900
```

Figura 17 – Execução no modo fácil retornando true.

```
?- facil(4).
false.
```

Figura 18 – Execução no modo fácil retornando false.

2.3 Buscando os Objetivos

Primeiramente, chamamos a regra 'coleta_coracoes', nessa regra, chamamos a busca em largura para buscar os corações, buscamos todos os corações de maneira recursiva. Dentro da nossa regra, calculamos e imprimimos a pontuação a cada coração recolhido, também imprimimos o caminho percorrido para cada coração. O código dessa regra se encontra na Figura 19.

```
%recolhe os corações e calcula e imprime a pontuação(recursiva)
coleta_coracoes([], _, _, _, _).
coleta_coracoes([Cabeca|Cauda], Inicio, Escadas, Garrafas, Brutus) :-
    Objetivo = Cabeca,
    busca_em_largura(Inicio, Escadas, Garrafas, Brutus, 0, Objetivo, CaminhoObjetivo),
    pegaPrimeiro(CaminhoObjetivo, I),
    pegaPrimeiro(Cabeca, Andar),
    calcular(Andar),
    write('Coracao na posicao '),
    write(I),
    write(' coletado pelo seguinte caminho:'),
    reverse(CaminhoObjetivo, CaminhoCorreto),
    writeln(CaminhoCorreto),
    write('Pontuacao atual: '),
    pontuacao(Pontuacao),
    writeln(Pontuacao),
    retract(ultimapos(L)),
    pegaPrimeiro(CaminhoObjetivo, U),
    asserta(ultimapos(U)),
    coleta_coracoes(Cauda, I, Escadas, Garrafas, Brutus).
```

Figura 19 – Coletando todos os corações

2.4 Movimentação

2.4.1 Direita e esquerda

Para que a movimentação para a esquerda e para a direita sejam realizadas, primeiramente vamos verificar a próxima posição(direita ou esquerda) utilizando a regra 'verificaDireita' ou 'verificaEsquerda'(23). A variável 'Yprox' recebe a posição do próximo quadrado(direita) e 'Yant' recebe a posição do quadrado anterior(esquerda).

```
    %movimenta para direita e chama a regra para verificar
prox([X, Y], Escadas, Garrafas, Brutus, TemEspinafre, [X, Yprox]) :-
    Yprox is Y+1,
    verificaDireita([X, Yprox], Escadas, Garrafas, Brutus, TemEspinafre).

    %movimenta para esquerda e chama a regra para verificar
prox([X, Y], Escadas, Garrafas, Brutus, TemEspinafre, [X, Yant]) :-
    Yant is Y-1,
    verificaEsquerda([X, Yant], Escadas, Garrafas, Brutus, TemEspinafre).
```

Figura 20 – Movimentação para direita e esquerda

2.4.2 Cima e baixo

Para que a movimentação para cima e para baixo sejam realizadas, primeiramente vamos verificar a próxima posição(acima ou abaixo) utilizando a regra 'verificaAcima' ou 'verificaAbaixo'(24). Onde 'Xprox' recebe a posição do quadrado acima e 'Xant' recebe a posição do quadrado abaixo.

```
    %movimenta para cima e chama a regra para verificar
prox([X, Y], Escadas, Garrafas, Brutus, _, [Xprox, Y]) :-
    Xprox is X+1,
    verificaAcima([Xprox, Y], Escadas, Garrafas, Brutus).

    %movimenta para baixo e chama a regra para verificar
prox([X, Y], Escadas, Garrafas, Brutus, _, [Xant, Y]) :-
    Xant is X-1,
    verificaAbaixo([Xant, Y], Escadas, Garrafas, Brutus).
```

Figura 21 – Movimentação para cima e baixo

2.4.3 Pular uma garrafa para direita e esquerda

Para pular uma garrafa, vamos verificar se existe uma garrafa na próxima posição(direita ou esquerda), e então, vamos verificar se não existe uma segunda garrafa através da regra 'verificaDoisObstaculosDireita' ou 'verificaDoisObstaculosEsquerda'(25). Onde 'Yprox' recebe a posição do próximo quadrado(direita), 'Yant' recebe a posição do quadrado anterior(esquerda) e 'Ypulo' recebe a posição após a garrafa, tanto para esquerda(-2) quanto para a direita(+2).

```
    %pula uma garrafa pela direita(caso possível)
prox([X, Y], Escadas, Garrafas, Brutus, TemEspinafre, [X, Ypulo]) :-
    Yprox is Y+1,
    pertence([X,Yprox], Garrafas),
    \+ verificaDoisObstaculosDireita([X, Yprox], Escadas, Garrafas, Brutus),
    Ypulo is Y+2.

    %pula uma garrafa pela esquerda(caso possível)
prox([X, Y], Escadas, Garrafas, Brutus, TemEspinafre, [X, Ypulo]) :-
    Yant is Y-1,
    pertence([X,Yant], Garrafas),
    \+ verificaDoisObstaculosEsquerda([X, Yant], Escadas, Garrafas, Brutus),
    Ypulo is Y-2.
```

Figura 22 – Pular uma garrafa

2.4.4 Verificar direita e esquerda

A primeira verificação realizada é se a próxima posição do Y está dentro do nosso ambiente(se é menor ou igual a 9 ou se é maior ou igual a 0), em seguida, verificamos se existem dois obstáculos(para a direita ou para a esquerda), e então, verificamos se o

Brutus não está impedindo a passagem do Popeye, caso o Popeye tenha o espinafre, ele poderá derrotar o Brutus.

```
%verifica se existe um caminho pela direita e se o Popeye pode passar
verificaDireita([X, Y], Escadas, Garrafas, Brutus, TemEspinafre) :-
    Y=<9,
    \+ verificaDoisObstaculosDireita([X, Y], Escadas, Garrafas, Brutus),
    \+ ([X,Y]==Brutus,TemEspinafre==0).

%verifica se existe um caminho pela esquerda e se o Popeye pode passar
verificaEsquerda([X, Y], Escadas, Garrafas, Brutus, TemEspinafre) :-
    Y>=0,
    \+verificaDoisObstaculosEsquerda([X, Y], Escadas, Garrafas, Brutus),
    \+ ([X,Y]==Brutus,TemEspinafre==0).
```

Figura 23 – Verifica as posições da direita e esquerda

2.4.5 Verificar acima e abaixo

A primeira verificação realizada é se a posição do X está dentro do ambiente(se é menor ou igual a 4 ou se é maior ou igual a 0), então, verificamos se existe uma escada para que ele possa subir ou descer naquela posição e também verificamos se no destino não existe alguma garrafa. Onde 'Xabaixo' recebe a posição abaixo da escada para verificar se existe alguma garrafa naquela posição e 'Xacima' recebe a posição acima da escada para verificar se existe alguma garrafa naquela posição.

```
%verifica se existe um caminho por cima e se o Popeye pode passar
verificaAcima([X, Y], Escadas, Garrafas, Brutus) :-
    X=<4,
    Xabaixo is X-1,
    pertence([Xabaixo, Y], Escadas),
    \+pertence([X, Y], Garrafas).

%verifica se existe um caminho por baixo e se o Popeye pode passar
verificaAbaixo([X, Y], Escadas, Garrafas, Brutus) :-
    X>=0,
    Xacima is X+1,
    pertence([X, Y], Escadas),
    \+pertence([Xacima, Y], Garrafas).
```

Figura 24 – Verifica as posições de cima e baixo

2.4.6 Verificar dois obstáculos(direita e esquerda)

Primeiro, verificamos se existe uma garrafa na próxima posição, e então, verificamos a posição após a garrafa, caso houver uma outra garrafa ou o Brutus, o caminho está bloqueado. Onde 'Yproximo' é a posição após a garrafa(direita) e 'Yanterior' é a posição anterior a garrafa(esquerda).

```
%verifica se existe duas garrafas ou uma garrafa e um Brutus em sequência(pe
verificaDoisObstaculosDireita([X, Y], Escadas, Garrafas, Brutus) :-
    Yproximo is Y+1,
    pertence([X, Y], Garrafas),
    (pertence([X, Yproximo], Garrafas);[X,Yproximo]==Brutus).

%verifica se existe duas garrafas ou uma garrafa e um Brutus em sequência(pe
verificaDoisObstaculosEsquerda([X, Y], Escadas, Garrafas, Brutus) :-
    Yanterior is Y-1,
    pertence([X, Y], Garrafas),
    (pertence([X, Yanterior], Garrafas);[X,Yanterior]==Brutus).
```

Figura 25 – Verifica as posições dos obstáculos

3 Exemplos de busca

Criamos o cenário apresentado na figura 26, que pode ser visualizado em uma animação neste link, o resultado obtido pela busca pode ser visualizado na figura 27. Também executamos esse mesmo cenário, onde o usuário escolheu começar por outra posição, como demonstrado na figura 28 e que pode ser visualizado em uma animação neste link, o resultado obtido pode ser conferido na figura 29. O exemplo 3 pode ser visualizado na figura 30, e também pode ser visualizado em uma animação neste link, o seu caminho está na figura 31. O exemplo 4 pode ser visualizado na figura 32, e a animação se encontra neste link, o caminho encontrado pelo algoritmo está na figura 33.

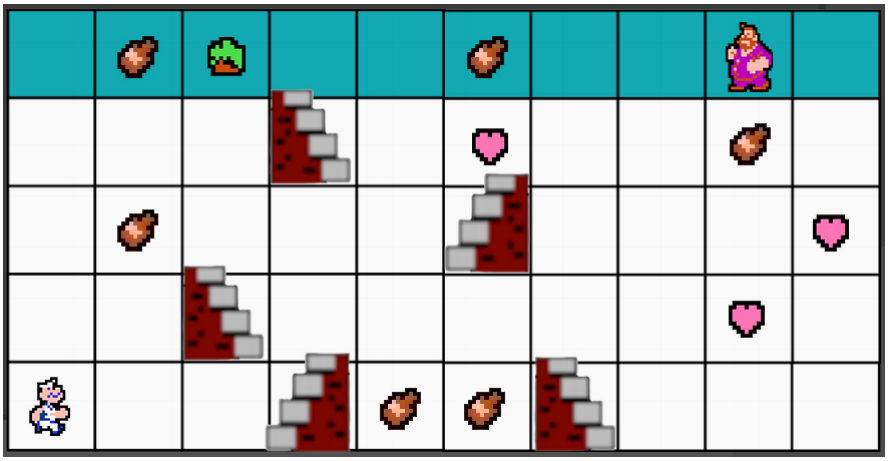

Figura 26 – Exemplo 1 no ambiente fácil

```
?- facil(0).
Coracao na posicao [1,6] coletado pelo seguinte caminho:[
[0,0],[0,1],[0,2],[0,3],[1,3],[1,4],[1,5],[1,6]]
Pontuacao atual: 200
Coracao na posicao [2,8] coletado pelo seguinte caminho:[
[1,6],[1,5],[1,4],[1,3],[1,2],[2,2],[2,3],[2,4],[2,5],[2,
6],[2,7],[2,8]]
Pontuacao atual: 500
Coracao na posicao [3,6] coletado pelo seguinte caminho:[
[2,8],[2,7],[2,6],[2,5],[3,5],[3,6]]
Pontuacao atual: 900
Espinafre na posicao [4,3] coletado pelo seguinte caminho
:[[3,6],[3,5],[3,4],[3,3],[4,3]]
Brutus na posicao [4,7] derrotado pelo seguinte caminho:[
[4,3],[4,4],[4,6],[4,7]]
Pontuacao final: 900
```

Figura 27 – Retorno da busca realizada no exemplo 1

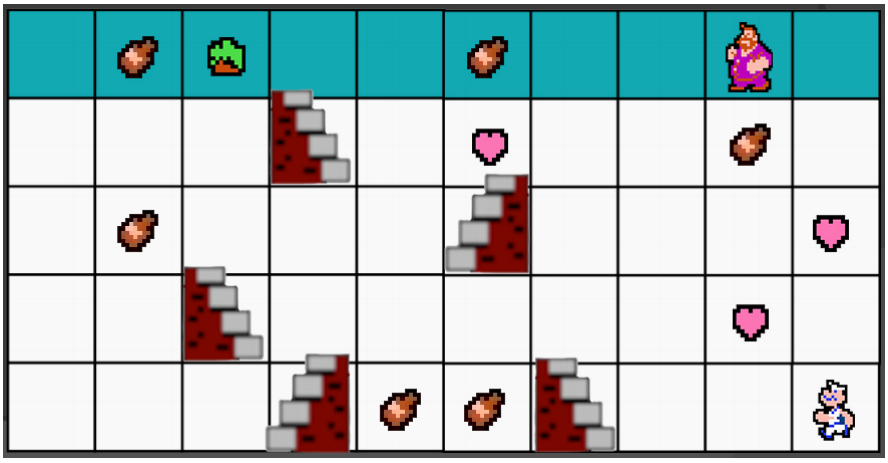

Figura 28 – Exemplo 2 no ambiente fácil

```
?- facil(9).
Coracao na posicao [1,5] coletado pelo seguinte caminho:[
[0,9],[0,8],[0,7],[0,6],[1,6],[1,5]]
Pontuacao atual: 200
Coracao na posicao [2,6] coletado pelo seguinte caminho:[
[1,5],[1,4],[1,3],[1,2],[2,2],[2,3],[2,4],[2,5],[2,6]]
Pontuacao atual: 500
Coracao na posicao [3,4] coletado pelo seguinte caminho:[
[2,6],[2,5],[3,5],[3,4]]
Pontuacao atual: 900
Espinafre na posicao [4,3] coletado pelo seguinte caminho
:[[3,4],[3,3],[4,3]]
Brutus na posicao [4,7] derrotado pelo seguinte caminho:[
[4,3],[4,4],[4,6],[4,7]]
Pontuacao final: 900
```

Figura 29 – Retorno da busca realizada no exemplo 2

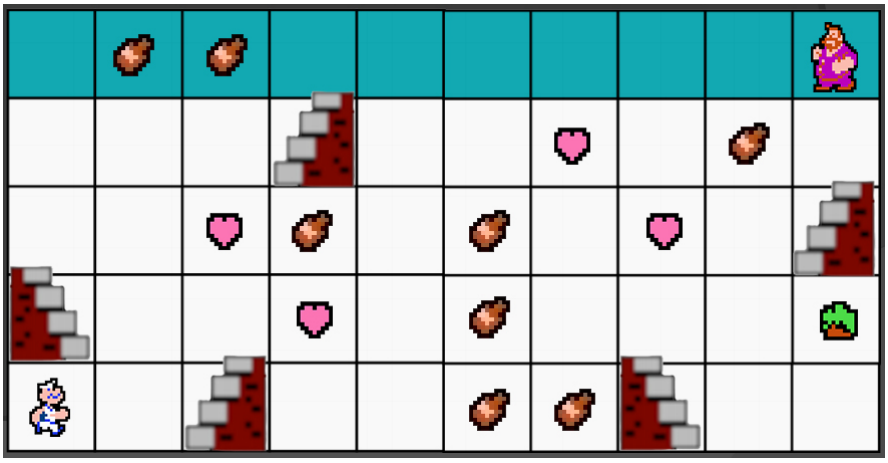

Figura 30 – Exemplo 3 no ambiente médio

```
?- medio(0).
Coracao na posicao [1,3] coletado pelo seguinte caminho:[
[0,0],[0,1],[0,2],[1,2],[1,3]]
Pontuacao atual: 200
Coracao na posicao [2,2] coletado pelo seguinte caminho:[
[1,3],[1,2],[1,1],[1,0],[2,0],[2,1],[2,2]]
Pontuacao atual: 500
Coracao na posicao [2,7] coletado pelo seguinte caminho:[
[2,2],[2,4],[2,6],[2,7]]
Pontuacao atual: 800
Coracao na posicao [3,6] coletado pelo seguinte caminho:[
[2,7],[2,8],[2,9],[3,9],[3,7],[3,6]]
Pontuacao atual: 1200
Espinafre na posicao [1,9] coletado pelo seguinte caminho
:[[3,6],[3,7],[3,9],[2,9],[2,8],[2,7],[2,6],[2,4],[2,2],[
2,1],[2,0],[1,0],[1,1],[1,2],[1,3],[1,4],[1,6],[1,7],[1,8
],[1,9]]
Brutus na posicao [4,9] derrotado pelo seguinte caminho:[
[1,9],[1,8],[1,7],[1,6],[1,4],[1,3],[1,2],[1,1],[1,0],[2,
0],[2,1],[2,2],[2,4],[2,6],[2,7],[2,8],[2,9],[3,9],[3,7],
[3,6],[3,5],[3,4],[3,3],[4,3],[4,4],[4,5],[4,6],[4,7],[4,
8],[4,9]]
Pontuacao final: 1200
```

Figura 31 – Retorno da busca realizada no exemplo 3

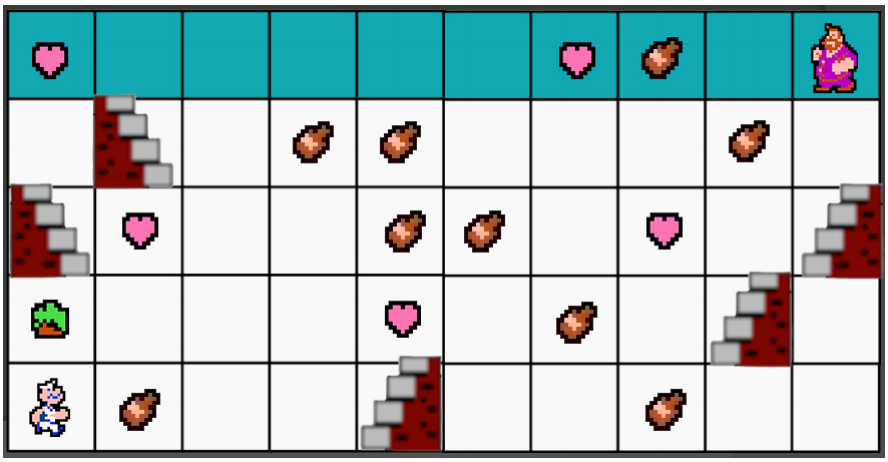

Figura 32 – Exemplo 4 no ambiente difícil

```
?- dificil(0).
Coracao na posicao [1,4] coletado pelo seguinte caminho:[
[0,0],[0,2],[0,3],[0,4],[1,4]]
Pontuacao atual: 200
Coracao na posicao [2,7] coletado pelo seguinte caminho:[
[1,4],[1,5],[1,7],[1,8],[2,8],[2,7]]
Pontuacao atual: 500
Coracao na posicao [4,6] coletado pelo seguinte caminho:[
[2,7],[2,8],[2,9],[3,9],[3,7],[3,6],[4,6]]
Pontuacao atual: 1000
Coracao na posicao [4,0] coletado pelo seguinte caminho:[
[4,6],[4,5],[4,4],[4,3],[4,2],[4,1],[4,0]]
Pontuacao atual: 1500
Coracao na posicao [2,1] coletado pelo seguinte caminho:[
[4,0],[4,1],[3,1],[3,0],[2,0],[2,1]]
Pontuacao atual: 1800
Espinafre na posicao [1,0] coletado pelo seguinte caminho
:[[2,1],[2,0],[3,0],[3,1],[4,1],[4,2],[4,3],[4,4],[4,5],[
4,6],[3,6],[3,7],[3,9],[2,9],[2,8],[1,8],[1,7],[1,5],[1,4
],[1,3],[1,2],[1,1],[1,0]]
Brutus na posicao [4,9] derrotado pelo seguinte caminho:[
[1,0],[1,1],[1,2],[1,3],[1,4],[1,5],[1,7],[1,8],[2,8],[2,
9],[3,9],[3,7],[3,6],[4,6],[4,8],[4,9]]
Pontuacao final: 1800
```

Figura 33 – Retorno da busca realizada no exemplo 4