UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS DE RIO PARANAÍBA
SISTEMAS DE INFORMAÇÃO

THIAGO ANTÔNIO MORAIS FERREIRA 5991
VICTOR PAIVA DE CASTRO 5973
EDUARDO NUNES DE OLIVEIRA 6021

PROJETO FINAL SIN323

RIO PARANAÍBA
2021

THIAGO ANTÔNIO MORAIS FERREIRA 5991
VICTOR PAIVA DE CASTRO 5973
EDUARDO NUNES DE OLIVEIRA 6021

PROJETO FINAL SIN323

Projeto final apresentado na disciplina Inteligencia Artificial - SIN 323

Orientador: Matheus Haddad

RIO PARANAÍBA

2021

Resumo

Esse trabalho foi elaborado como uma das exigências da disciplina SIN 323 - Inteligência Artificial. O presente trabalho tem como objetivo a modelagem do ambiente do jogo Popeye, definindo um conjunto de regras com o intuito de solucionar um problema de busca com a programação Prolog. O algoritmo deve possibilitar que o Popeye se movimente em um ambiente, coletando corações e por fim busque uma lata de espinafre. Após a coleta de todos os corações, e o espinafre, possibilita-se que o Popeye caminhe até a posição final do problema de busca, derrotando seu inimigo Brutus.

Palavras-chaves: Popeye, Prolog, Problema de busca, Inteligência Artificial.

Lista de ilustrações

Sumário

1 Introdução

A inteligência artificial se define como um campo da ciência, e envolve um agrupamento de diversas tecnologias, como algoritmos, sistemas de aprendizado, entre outros que possibilitam simular capacidades humanas ligadas à inteligência. Como exemplo, temos o raciocínio, a percepção de ambiente e a habilidade de análise para a tomada de decisão. A inteligência artificial se aplica em diversas áreas, tais como: jogos, carros autônomos, sistemas de atendimento dos hospitais e buscadores de internet. O presente projeto, tem como objetivo a criação adaptada de um famoso jogo de plataforma arcade conhecido como Popeye, utilizando-se da linguagem de programação Prolog e inteligência artificial.

1.1 Objetivo Geral

O objetivo geral do projeto consiste em utilizar-se da linguagem de programação Prolog e de conceitos de inteligência artificial apresentados em aula, com o objetivo de aprimorar o conhecimento dos alunos.

1.2 Objetivos Específicos

Como síntese dos objetivos específicos, a partir de um ambiente programável, deve-se buscar as sub-metas e guardar os caminhos resultantes acumulativos, gerando um caminho final até a posição final do ambiente de busca.

2 Definição do Ambiente

O ambiente definido para o jogo será uma matriz de 5 andares e 10 espaços para andar, totalizando 50 posições. Considerando-se que objetos poderão estar definidos em quaisquer posição dese ambiente (programável).

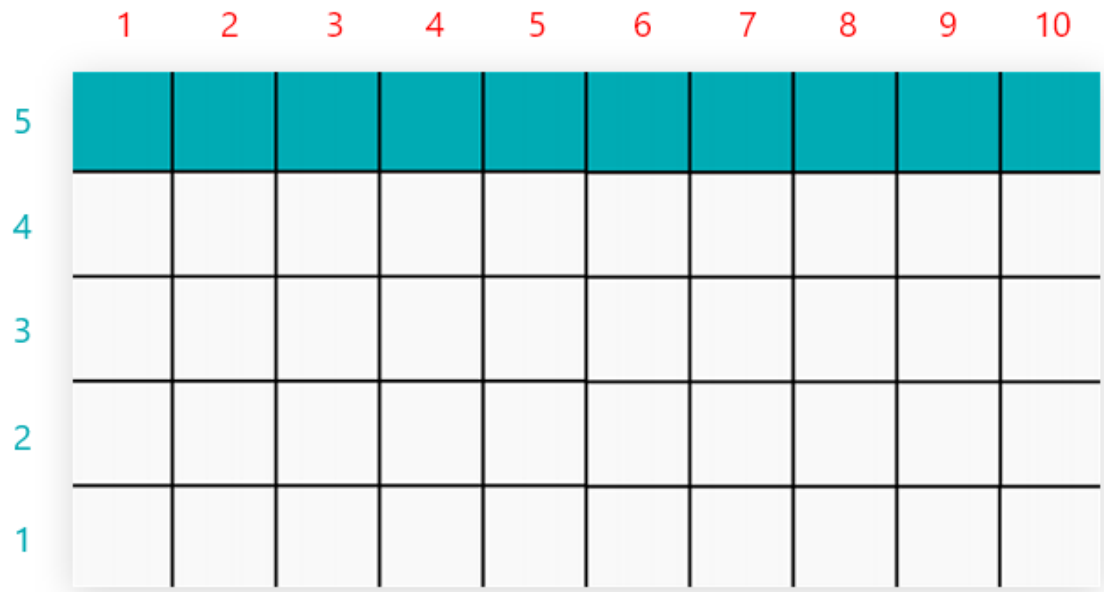

Figura 1 – Ambiente sem objetos

3 Agentes

3.1 Popeye

O Popeye é um agente, qual define a posição inicial do problema de busca. Ele pode ser inserido em qualquer posição do ambiente, preferencialmente no primeiro andar. Ele pode se movimentar para a direita, esquerda ou subir ou descer escadas. A sua movimentação pode ser dificultada por outros objetos no ambiente.

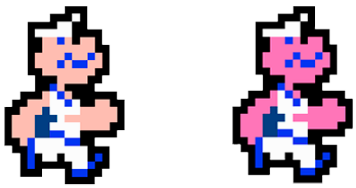

Figura 2 – Popeye - Antes e Após a coleta do Espinafre

3.2 Brutus

O Brutus é objetivo do jogo, qual define a posição final do problema de busca. Ele pode ser inserido em qualquer posição do ambiente, preferencialmente no último andar. Brutus também é um bloqueio no caminho de Popeye, no caso do Popeye ainda não ter coletado o espinafre.

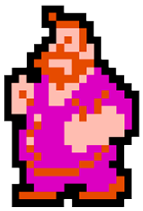

Figura 3 – Brutus

4 Objetos

4.1 Coração

O coração é um objeto do jogo, e pode existir mais de um no ambiente. Ele pode ser inserido em qualquer posição vazia do ambiente. A coleta de um coração acrescenta uma pontuação respectiva ao andar em que o coração se encontra. Essa pontuação é acumulativa e é exibida ao usuário no termino do jogo. Ele se caracteriza como a primeira meta no caminho do Popeye em busca do Brutus.

Figura 4 – Coração

4.2 Espinafre

O espinafre é um objeto do jogo, e só pode existir um no ambiente. Ele pode ser inserido em qualquer posição vazia do ambiente. O espinafre se caracteriza como a chave para conclusão da busca do Brutus. Ele se pode ser coletado após a completa coleta de corações no ambiente.

Figura 5 – Espinafre

4.3 Escada

A escada é um objeto do jogo, qual permite a movimentação do Popeye em andares diferentes. Esse objeto ocupa duas posições do ambiente (andar de baixo e andar de cima) e é montada de forma diagonal.

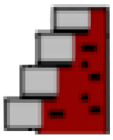

Figura 6 – Escada

4.4 Garrafa

A garrafa é um objeto do jogo, e pode existir mais de uma no ambiente. Ela pode ser inserida em qualquer posição vazia do ambiente. A garrafa implica um bloqueio na movimentação de Popeye, onde uma garrafa possibilita o Popeye de pulá-la, caso os dois quadrados adjacentes à garrafa não possuírem nenhum objeto. A movimentação de Popeye é impedida caso existam duas garrafas consecutivas.

Figura 7 – Garrafa

5 Restrições

- O Popeye só pode se movimentar entre andares e de forma diagonal, se houver uma escada e o Popeye estiver na mesma posição de origem da escada.
- O Popeye só pode alcançar a posição final do problema de busca (Brutus) se todos os corações e o espinafre tiverem sido coletados.
- O Popeye não pode se movimentar no caso de 2 garrafas em posições consecutivas.
- O Popeye só pode coletar o espinafre caso todos os corações tiverem sido coletados.
- Caso o Popeye esteja em uma extremidade do ambiente, ele não pode atravessar essa extremidade chegando ao outro lado de forma instantânea.

6 Método implementado

O trabalho foi implementado na linguagem de programação Prolog e o método utilizado para solução do problema de busca foi o algoritmo de Busca em Largura.O Algoritmo de busca em largura realiza a busca pelo objetivo (meta ou estados meta). A busca é feita começando por um vértice, em nosso caso uma posição do ambiente, e depois visita todos os vizinhos dessa posição, e por assim em diante, considerando diversos níveis de vizinhança até encontrar o estado meta. Dessa forma, partindo-se de uma posição, o algoritmo retorna como solução o menor caminho de cada estado meta.

O algoritmo armazena as soluções em uma lista, e vai concatenando cada solução gerada por uma posição/meta em uma nova solução. No fim, a solução final gerada contempla todas as soluções com os estados explorados que compõe o caminho, no caso, o caminho do estado inicial até o último estado meta.

7 Programação do ambiente

Como foi visto anteriormente, o ambiente implementado consiste em uma matriz de 5 andares e 10 espaços por andar, qual totaliza 50 posições. Dessa forma, a imagem a seguir representa a parte inicial da codificação, onde são definidas as coordenadas dos objetos e a do agente Brutus, ou seja, os estados finais e obstáculos do jogo são estabelecidos.

Considerando-se que qualquer objeto pode ser setado em qualquer posição vazia do ambiente, ao programar-se as posições de objetos e do agente Brutus, o ambiente estará estabelecido. O Popeye é setado no ambiente na posição passada como parâmetro na inicialização do jogo.

A programação de uma escada se difere da programação de outros objetos e agentes, pois a escada ocupa duas posições do ambiente, sendo elas: posição correspondente ao andar de baixo e posição correspondente ao andar de cima. A programação de um coração também se difere da programação de outros objetos e agentes, pois armazena-se uma lista de coordenadas, com as posições de cada um dos corações do ambiente.

```
% DEFINIÇÂO DO AMBIENTE

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

escada([[1,3],[2,4]]).
escada([[2,2],[3,3]]).
escada([[3,7],[4,6]]).
escada([[4,3],[5,4]]).

coracao([[1,2],[1,3],[1,4],[2,6]]).

espinafre([3,8]).

brutus([5,10]).
```

Figura 8 – Exemplificação de como se programar um Ambiente

8 Funcionalidades Implementadas

8.1 Regras

8.1.1 Movimentação

8.1.1.1 Movimentação sem Espinafre:

O conjunto de regras definidos para movimentação sem espinafre é criado com o intuito de impedir Popeye de passar por Brutus antes da coleta do espinafre. Dessa forma, Popeye consegue andar para a esquerda, direita, pular garrafas e subir e descer escadas em busca de corações ou em busca do espinafre desde que não tenha o Brutus pelo caminho.

- **Movimentação Diagonal:**

Para que Popeye se movimente entre andares, deve-se haver uma escada, qual a posição de origem da escada ([X,Y]) deve se encontrar na mesma posição de Popeye. A posição de destino do Popeye é definida atráves de unificação com as posições da escada. Dessa forma, confere se existe uma escada nessa posição, comparando-se com os fatos definidos no ambiente e se Brutus não se encontra na posição de destino da escada.

I - Movimentação para subir uma escada:

Quando Popeye vai subir uma escada, a posição de destino se encontra no andar de cima, portanto deve-se unificar [X2,Y2] com o [andardecima] de escadas ([andardebaixo],[andardecima]).

```
movimentacaoSemEspinafre([X,Y],[X2,Y2],Brutus):-
    escada([[X,Y],[X2,Y2]]),
    [X2,Y2] \= Brutus.
```

II - Movimentação para descer uma escada:

Quando Popeye vai descer uma escada, a posição de destino se encontra no andar de baixo, portanto deve-se unificar [X2,Y2] com o [andardebaixo]

de escadas([andardebaixo],[andardecima]).

```
movimentacaoSemEspinafre([X,Y],[X2,Y2],Brutus):-
    escada([[X2,Y2],[X,Y]]),
    [X2,Y2] \= Brutus.
```

- **Movimentação Horizontal sem Obstáculos:**

Para que Popeye se movimente para esquerda ou direita, deve-se verificar se a posição de destino ([X,Y-1] ou [X,Y+1]) de Popeye não encontra nenhum obstáculo. Dessa forma é feita uma verificação se a posição é válida, ou seja, se essa posição pertence ao ambiente, por fim é verificado se a posição de destino difere-se das posições de garrafas e Brutus.

I - Movimentacao para direita:

```
movimentacaoSemEspinafre([X,Y],[X,Y2],Brutus):-
    Y<10,
    Y2 is Y+1,
    not(garrafa([X,Y2])),
    [X,Y2] \= Brutus.
```

II - Movimentacao para esquerda:

```
movimentacaoSemEspinafre([X,Y],[X,Y2],Brutus):-
    Y>1,
    Y2 is Y-1,
    not(garrafa([X,Y2])),
    [X,Y2] \= Brutus.
```

- **Movimentação Horizontal com Obstáculos:**

Para que Popeye se movimente, deve-se verificar se a posição de destino ([X,Y+2] ou [X,Y-2]) de Popeye não encontra nenhum obstáculo e se Brutus não está localizado na posição anterior. Dessa forma é feita uma verificação se a posição é válida, ou seja, se essa posição pertence ao ambiente e difere-se das posições de garrafas e Brutus.

I - Movimentação para direita:

movimentacaoSemEspinafre([X,Y],[X,Y2],Brutus):-

Y<9,

Y1 is Y+1,

Y2 is Y+2,

garrafa([X,Y1]),

not(garrafa([X,Y2])),

[X,Y1] $\bar{B}$rutus,

[X,Y2] $\bar{B}$rutus.

II - Movimentacao para esquerda:

movimentacaoSemEspinafre([X,Y],[X,Y2],Brutus):-

Y>1,

Y1 is Y-1,

Y2 is Y-2,

garrafa([X,Y1]),

not(garrafa([X,Y2])),

[X,Y1] $\bar{B}$rutus,

[X,Y2] $\bar{B}$rutus.

8.1.1.2 Movimentação com Espinafre:

O conjunto de regras definidos para movimentação com espinafre é criado com o intuito de Popeye conseguir chegar em Brutus (último estado meta) após a coleta do espinafre. Dessa forma, Popeye consegue andar para a esquerda, direita, pular garrafas e subir e descer escadas em busca de Brutus.

Essa movimentação se assemelha com a movimentação sem espinafre, com a diferença de não impedir chegar em Brutus.

- **Movimentação Diagonal:**

  I - Movimentação subir escada

  movimentacaoComEspinafre([X,Y],[X2,Y2]):-

  escada([[X,Y],[X2,Y2]]).

  II - Movimentação para descer uma escada

  movimentacaoComEspinafre([X,Y],[X2,Y2]):-

  escada([[X2,Y2],[X,Y]]).

- **Movimentação Horizontal sem Obstáculo:**

  I - Movimentação para direita

  movimentacaoComEspinafre([X,Y],[X,Y2]):-

  Y<10,

  Y2 is Y+1,

  not(garrafa([X,Y2])).

  II - Movimentação para esquerda

  movimentacaoComEspinafre([X,Y],[X,Y2]):-

  Y>1,

  Y2 is Y-1,

  not(garrafa([X,Y2])).

- **Movimentação Horizontal com Obstáculo:**

  I - Movimentação para direita

  movimentacaoComEspinafre([X,Y],[X,Y2]):-

  Y<9,

  Y1 is Y+1,

  Y2 is Y+2,

  garrafa([X,Y1]),

  not(garrafa([X,Y2])).

II - Movimentação para esquerda

```
movimentacaoComEspinafre([X,Y],[X,Y2]):-
    Y>2,
    Y1 is Y-1,
    Y2 is Y-2,
    garrafa([X,Y1]),
    not(garrafa([X,Y2])).
```

8.1.2 Manipulação de Listas

Verificar se o elemento pertence a lista:

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

Concatenar lista:

```
concatena([ ], L, L).
concatena([Cab|L1], L2,[Cab|L3]):-
    concatena(L1, L2, L3).
```

Inverter lista:

```
inverter([ ],[ ]).
inverter([Elem|Cauda], Lista_Invertida):-
     inverter(Cauda,Cauda_Invertida),
     concatena(Cauda_Invertida,[Elem], Lista_Invertida).
```

Retirar um elemento da lista:

```
retirar_elemento(Elem,[Elem|Cauda],Cauda).
retirar_elemento(Elem,[Elem1|Cauda],[Elem1|Cauda1]):-
    retirar_elemento(Elem,Cauda,Cauda1).
```

Exibir último elemento da lista:

```
retornar_ultimo(X,[X]).
retornar_ultimo(X, [_|T]):-
    retornar_ultimo(X,T).
```

Retornar o primeiro elemento da lista:

```
retornar_primeiro(X,[X|_]).
```

8.1.3 Busca em Largura

A função de busca em largura usado nesse projeto tem como base os códigos apresentados em sala de aula na disciplina de Inteligência artificial. Será feito uma breve explicação sobre o código implementado.

O algoritmo abaixo define o caso base da busca em largura, quando o destino da meta, qual foi passada como parâmetro, é igual ao estado que Popeye se encontra.

```
busca_em_largura([[Estado|Caminho]|_],Destino,[Estado|Caminho]):-
    Destino=Estado.
```

Caso o caso base não seja verdadeiro, precisamos procurar nos sucessores. Dessa forma, os sucessores e seus caminhos são colocados em "Sucessores", que concatena aos "Outros" estados de fronteira formando uma "NovaFronteira". Por fim, a função busca_em _largura é chamada novamente passando a nova fronteira.

Estender significa encontrar uma lista com os estados sucessores.

A função bagof é a que faz com que o algoritmo gere todos os caminhos possíveis e só irá escolher um com base nas restrições impostas nessa função.

```
busca_em_largura_([Primeiro|Outros],Destino,Solucao):-
    estende_(Primeiro,Sucessores),
    concatena(Outros,Sucessores,NovaFronteira),
    busca_em_largura(NovaFronteira,Destino,Solucao).

estende([Estado|Caminho],ListaSucessores):-
    bagof([Sucessor,Estado|Caminho],(movimentacao(Estado,Sucessor),
not(pertence(Sucessor,[Estado|Caminho]))),ListaSucessores),!
```

estende_(_,[ ]).

8.1.3.1 Codificação para Busca em Largura sem Espinafre

Essa busca em largura foi feita para os casos em que o Popeye ainda não coletou o espinafre. Nesse caso é necessário passar Brutus como parâmetro, de forma que na movimentação seja verificado se o estado sucessor não se encontra Brutus.

```
busca_em_largura_sem_espinafre([[Estado|Caminho]|_],Destino,[Estado|Caminho],Brutus):-
    Destino=Estado,Brutus=Brutus.
busca_em_largura_sem _espinafre([Primeiro|Outros],Destino,Solucao,Brutus):-
    estende_sem_espinafre(Primeiro,Sucessores,Brutus),
    concatena(Outros,Sucessores,NovaFronteira),
    busca_em_largura_sem_espinafre(NovaFronteira,Destino,Solucao,Brutus).

estende_sem_espinafre([Estado|Caminho],ListaSucessores,Brutus):-
    bagof([Sucessor,Estado|Caminho],(movimentacaoSemEspinafre(Estado,Sucessor,Brutus),
not(pertence(Sucessor,[Estado|Caminho]))),ListaSucessores), !.

estende_sem_espinafre(_,[ ],_).
```

8.1.3.2 Codificação para Busca em Largura com Espinafre

Essa busca em largura foi feita para os casos em que o Popeye já coletou o espinafre. Nesse caso não é necessário passar Brutus como parâmetro.

```
    busca_em_largura_com_espinafre([[Estado|Caminho]|_],Destino,[Estado|Caminho]):-
        Destino=Estado.

    busca_em_largura_com_espinafre([Primeiro|Outros],Destino,Solucao):-
        estende_com_espinafre(Primeiro,Sucessores),
        concatena(Outros,Sucessores,NovaFronteira),
        busca_em_largura_com_espinafre(NovaFronteira,Destino,Solucao).
```

estende_com_espinafre([Estado|Caminho],ListaSucessores):-

bagof([Sucessor,Estado|Caminho], (movimentacaoComEspinafre(Estado,Sucessor), not(pertence(Sucessor,[Estado|Caminho]))), ListaSucessores),

!.

estende_com_espinafre(_,[ ]).

8.1.4 Implementação de uma regra recursiva para soma de caminhos de um coração até o outro

Essa regra recursiva tem como objetivo retornar a solução contendo o caminho do primeiro coração até o último coração.

PARÂMETROS:

- **Elem: Se define como o coração de origem.**
- **Cauda: Se define como o coração de destino.**
- **Cauda1: Recebe o restante de corações.**
- **Solução: Armazena a solução final, contendo o caminho do primeiro até o último coração.**
- **Brutus: A posição do agente Brutus é passada, pelo fato de não ter sido coletado o espinafre.**
- **SomaSolucao: É um parâmetro que se inicia zerado, pois ainda não foi somado nenhuma solução a ele.**
- **Cauda2: É um parâmetro que se inicia zerado, pois ele define a posição que se repete após concatenar duas soluções, para que essa posição seja retirada da solução concatenada.**
- **TotalPontos: É um parâmetro que se inicia zerado, e recebe apenas o valor total após todos os corações terem sido coletados.**
- **SomaPontos: É um parâmetro que se inicia zerado, e conforme ocorre a coleta de corações vai sendo atribuído a ele o valor do coração atualmente coletado mais aqueles que já foram coletados anteriormente.**

PASSO A PASSO:

1ª: É feita uma busca em largura do primeiro coração até o segundo coração e chama a solução desta busca de Solução2.

2ª: Concatena a solução obtida com a SomaSolucao, que inicialmente está zerada.

3ª: Verifica se é a primeira vez que essa regra é chamada.

4ª: Caso seja a primeira vez que a regra é chamada, verifica se possui apenas dois corações no ambiente, ou seja Cauda1 está vazia.

5ª: Se Cauda1 estiver vazia, é feita a concatenação da solucao2 com vazia e chama o resultado de Solucao, pois não haverá mais corações a serem buscados. Após isso é feito o cálculo da pontuação desses dois corações encontrados e a regra recursiva é encerrada, pois Solução e TotalPontos já foram unificados.

6ª: Caso a cauda1 não esteja vazia, ou seja, ainda possui um ou mais corações a serem encontrados, é feito o cálculo da pontuação de coração origem e chama a regra recursiva novamente, porém agora:

Elem = Cauda

Cauda = cabeça de Cauda1

Cauda1 = Calda1 sem a cabeça

Solucao = Solucao (ainda sem nada)

Brutus = Brutus

SomaSolucao = Solucão2

Cauda2 = Cauda

TotalPontos = TotalPontos (ainda sem nada).

SomaPontos = pontuação do primeiro coração.

7ª: Caso não seja a primeira vez em que a regra recursiva seja chamada é retirado a Cauda2 da Solucao2 e verifica se não possui mais algum coração a ser visitado, ou seja, Cauda1 está vazia.

8ª: Caso não possua mais corações a serem visitados, é feita uma concatenação da Solução3, que foi gerada após a retirada do elemento Cauda2, com vazia, faz a soma da pontuação do coração de origem (Elem) com o último coração, soma com SomaPontos e armazena em TotalPontos.

9ª: Caso ainda possua mais corações a serem visitados, é feito o cálculo da pontuação de coração origem (Elem) e chama a regra recursiva novamente, porém agora:

Elem = Cauda

Cauda = cabeça de Cauda1

Cauda1 = Calda1 sem a cabeça

Solucao = Solucao (ainda sem nada)

Brutus = Brutus

SomaSolucao = Solucão3

Cauda2 = Cauda

TotalPontos = TotalPontos (ainda sem nada).

SomaPontos = a soma de todos corações origens até o momento.

A REGRA RECURSIVA:

```
recursivo(Elem,[Cauda|Cauda1],Solucao,Brutus,SomaSolucao,Cauda2,TotalPontos,SomaPontos):-

busca _em _largura _sem _espinafre([[Elem]],Cauda,Solucao0,Brutus),
concatena(Solucao0,SomaSolucao,Solucao2),
(
Cauda2 = [ ] ,
    (Cauda1 = [ ],
    concatena(Solucao2,[ ],Solucao),
    retornar _primeiro(PontosTemp1,Elem),
     PontosTemp is PontosTemp1 *100,
    retornar _primeiro(PontosTemp2,Cauda),
    TotalPontos is PontosTemp2 *100 + PontosTemp
     ;
    retornar _primeiro(PontosTemp1,Elem),
    PontosTemp is PontosTemp1 *100,
    recursivo(Cauda,Cauda1,Solucao,Brutus,Solucao2,Cauda,TotalPontos,PontosTemp))
;
retirar _elemento(Cauda2,Solucao2,Solucao3),
(Cauda1 = [ ],
    concatena(Solucao3,[ ],Solucao),
```

```
    retornar _primeiro(PontosTemp1,Elem),
    PontosTemp is PontosTemp1 *100 + SomaPontos,
    retornar _primeiro(PontosTemp3,Cauda),
    TotalPontos is PontosTemp3 *100 + PontosTemp
    ;
    retornar _primeiro(PontosTemp1,Elem),
    PontosTemp is PontosTemp1 *100 + SomaPontos,
    PontosTemp2 is PontosTemp,
    recursivo(Cauda,Cauda1,Solucao,Brutus,Solucao3,Cauda,TotalPontos,PontosTemp2))
).
```

8.1.5 Implementação de uma regra para agrupamento das buscas em largura em uma solução final

Essa regra tem como objetivo juntar todas as soluções de busca em largura para as submetas em uma solução final.

PASSO A PASSO:

1ª: É feita a busca largura de Popeye até o primeiro coração

2ª: Depois, é feita uma verificação, para caso o ambiente contenha apenas um coração.

3ª: Caso o ambiente contenha apenas um coração é realizada feita a contabilização da pontuação desse coração, e chamamos a busca em largura do coração até espinafre, concatenamos o caminho resultante dessa busca com o caminho de Popeye até o primeiro coração e retiramos o elemento coração uma vez, pois ele estaria repetido nessa solução encontrada, chamamos a solução com o caminho de Popeye até o coração de Solucao7

4ª: Caso o ambiente tenha mais de um coração, chamamos a regra recursiva onde nos retornará a pontuação final junto com a solução dos caminhos entre os corações do jogo. Em seguida concatenamos esta solução com o caminho de Popeye até o primeiro coração e retiramos a posição do primeiro coração que se repete no caminho. Após isso iniciaremos a busca em largura partindo do último coração até o Espinafre, concatenamos esta solução gerada com a que contém o caminho de Popeye até o último coração e chamamos a solução com o caminho de Popeye até o último coração de Solucao7.

5ª: Agora, independente da quantidade de corações no ambiente, realizamos uma busca em largura de Espinafre até Brutus, concatenamos este resultado com a Solucao7, retiramos a posição de espinafre que se repete e por fim invertemos esta Solucao final e

chamamos de Solucao para que seja unificada com Solucao do parâmetro.

A REGRA SOLUCAO _BL:

```
solucao _bl(Popeye,[Coracao|Cauda],Espinafre,Brutus,Solucao,Pontos):-
        busca _em _largura _sem _espinafre([[Popeye]],Coracao,Solucao1,Brutus),
        (
            Cauda = [ ] ,
            retornar _primeiro(PontosTemp,Coracao),
            Pontos is PontosTemp *100,
            busca _em _largura _sem _espinafre([[Coracao]],Espinafre,Solucao5,Brutus),
            concatena(Solucao5,Solucao1,Solucao6),
            retirar _elemento(Coracao,Solucao6,Solucao7)
        ;
            recursivo(Coracao,Cauda,Solucao2,Brutus,[ ],[ ],Pontos,Pontos),
            concatena(Solucao2,Solucao1,Solucao3),
            retirar _elemento(Coracao,Solucao3,Solucao4),
            retornar _ultimo(X,Cauda),
            busca _em _largura _sem _espinafre([[X]],Espinafre,Solucao6,Brutus),
            retirar _elemento(X,Solucao4,Solucao5),
            concatena(Solucao6,Solucao5,Solucao7)
        ),
        busca _em _largura _com _espinafre([[Espinafre]],Brutus,Solucao8),
        concatena(Solucao8,Solucao7,Solucao9),
        retirar _elemento(Espinafre,Solucao9,Solucao10),
        inverter(Solucao10,Solucao).
```

8.1.6 MAIN

Essa regra recebe como parâmetro do usuário a posição inicial de Popeye, unifica as variáveis X, Y, Z com coração, espinafre e Brutus respectivamente. Por fim a regra solucao_bl é chamada, e essas variáveis são passadas por parâmetro. E também é responsável pela exibição da solução final, posição dos corações encontrados, posição do espinafre e do Brutus na tela para o usuário.

main(Popeye):-

coracao(X),

espinafre(Y),

brutus(Z),

solucao$_bl(Popeye, X, Y, Z, Solucao, Pontos)$,

write("

Caminho total percorrido por Popeye :

"), write(Solucao),

write("

Localizacao do Coracao : "),write(X),

write("

Localizacao do Espinafre : "),write(Y),

write("

Localizacao do Brutus : "),write(Z),

write("

Pontuacao : "),write(Pontos),

write("

"),

!.

9 Instruções de uso

Nesse momento, é importante que algumas instruções sejam seguidas para execução do jogo de forma correta. O programa SWI-Prolog deve ser o programa utilizado para execução do jogo.

Com o SWI-Prolog e o arquivo aberto, basta digitar o comando main(), qual passa como parâmetro a posição inicial do Popeye. Por exemplo: main([1,1]).