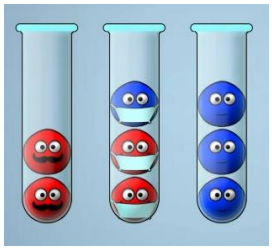
Trabalho Final de Programação Lógica

Nome: Vitor Barbosa Lemes Fernandes Nº de Matrícula: 11921BCC035

Tema escolhido: Lyfoes

Representação: O jogo é representado por uma lista, composta por sublistas, onde cada sublista representa um tubo. Dentro de cada tubo tem até 4 Lyfoes de qualquer uma das 10 cores disponíveis, sendo a head da sublista o Lyfoe do topo do tubo.

Ex: para representar o caso da figura abaixo usamos: [[r,r],[b,r,r],[b,b,b]].



Abaixo está o código do trabalho, com pequenos comentários. Após o código está o relatório, com as descrições, justificativas de utilização e testes de cada um dos subproblemas.

```
% Cores: Red, Blue, Green, White, Violet, Orange, Pink, Yellow, Cyan, Light Pink.
cor(r).
cor(b).
cor(g).
cor(w).
cor(v).
cor(o).
cor(p).
cor(y).
cor(c).
cor(l).

% 1- Verifica se todo mundo eh cor.
ehCor([[H|T1]|T2]):- cor(H), ehCor([T1|T2]).
ehCor([[]|T2]):- ehCor(T2).
ehCor([]).

% 2- Verifica se cada tubo tem no maximo 4 lyfoes.
max4([H|T]):- length(H,X), X=<4, max4(T).
max4([]).

% 3- Compara dois tubos para garantir que o Lyfoe nao volte para o msm tubo.
igual([H1|T1],[H2|T2]):- H1==H2, igual(T1,T2).
igual([],[]).
```

```
% 4- Confere se um tubo foi resolvido, e chama o proximo tubo.
verificaTubo([[H1,H2|T1]|T2],L):- H1==H2, verificaTubo([[H1|T1]|T2],[H1|L]),!.
verificaTubo([[X]|T],L):- proxTubo(T,[X|L]).
verificaTubo([[]|T],L):- proxTubo(T,L).

% 5- Avanca um tubo na verificacao de conclusao.
proxTubo([[H|T1]|T2],L):- not(member(H,L)), verificaTubo([[H|T1]|T2],L).
proxTubo([[]|T],L):- proxTubo(T,L).
proxTubo([],_).

% 6- Retira um Lyfoe de algum tubo.
retira([[H|T1]|T2],[T1|T2], H,T1).
retira([[H|T1]|T2],[[H|T1]|R], X,Ant):- retira(T2,R,X,Ant).
retira([[]|T2], [[]|R],X,Ant):- retira(T2,R,X,Ant).
retira([],_,_,_):- !, fail.

% 7- Verifica se um movimento é idiota.
movIdiota([H1|Prox],[H2|Ant],X):- verificaTubo([[H1|Prox]],[]),
                                  verificaTubo([[H2|Ant]],[]), length([H2|Ant],X1),
                                  length([H1|Prox],X2), X1>1, X2==1, H1==X, H2==X.
movIdiota([],[H2|Ant],X):- verificaTubo([[H2|Ant]],[]), H2==X.

% 8- Insere um Lyfoe em algum tubo.
insere([[H|T1]|T2],[[X,H|T1]|T2],X,Ant):- length([H|T1],Qnt), not(igual([H|T1],Ant)),
                                          not(movIdiota([H|T1],Ant,X)), Qnt<4, H==X.
insere([[]|T],[[X]|T],X,Ant):- not(igual(Ant,[])), not(movIdiota([],Ant,X)).
insere([H|T],[H|L],X,Ant):- insere(T,L,X,Ant).

% 9- Retira de um Lyfoe de um tubo e insere em outro.
move(L,Hist,R,Mov):- Mov>0, retira(L,LN,H,T1), insere(LN,LNN,H,T1),
                     not(member(LNN,Hist)), confere(LNN,[LNN|Hist],R,Mov).

% 10- Confere se o problema foi resolvido.
confere(L,Hist,Hist,_):- verificaTubo(L,[]),!.
confere(L,Hits,R,Mov):- Mov>0, Mov1 is Mov-1, move(L,Hits,R,Mov1).

% 11- Calcula a quantidade de movimentos.
qntMov(X,X1):- X<10, X1 is X*3.
qntMov(X,X1):- X>=10, X<15, X1 is X*4.
qntMov(X,X1):- X>=15, X1 is X*6.
```

```
% 12- Imprimi o resultado linha a linha.
imprime([H|T]):- write(H), nl, imprime(T).
imprime([]).

% 13.A- Resolve o problema dado.
lyfoes(L):-  verificaTubo(L,[]),!.
lyfoes(L):- ehCor(L), max4(L), length(L,X), qntMov(X,X1),
            move(L,[L],R,X1), reverse(R,R1), imprime(R1).

% 13.B- Resolve o problema dado, com entrada de limite de movimentos.
lyfoes(L,_):-  verificaTubo(L,[]),!.
lyfoes(L,X):- ehCor(L), max4(L), move(L,[L],R,X), reverse(R,R1), imprime(R1).
```

Relatório

1 - ehCor(L): Verifica se todos os elementos de todas as sublistas são de fato uma das 10 cores disponíveis no jogo.
Teste 1: ehCor([[r],[g],[b]]). Retorna true.
Teste 2: ehCor([[r],[g],[t]]). Retorna false.

2 - max4(L): Em Lyfoes cada tubo pode conter no máximo 4 Lyfoes. Esta função verifica se cada sublista da lista possui no máximo 4 elementos.
Teste 1: max4([[r,y,b,r],[y,b,r,b],[]]). Retorna true.
Teste 2: max4([[r,y,b,r],[b,y,b,r,b],[]]). Retorna false.

3 - igual(L1,L2): Uma regra que compara o tubo que um Lyfoe foi retirado com o possível tubo que ele será inserido e verifica se eles são iguais. Essa regra tem como objetivo evitar que um Lyfoe seja inserido no mesmo tubo em que estava, ou em outro tubo igual, o que manteria exatamente o mesmo cenário.
Teste 1: igual([r,r,r],[r,r,r]). Retorna true.
Teste 2: igual([r,b,r],[r,r,r]). Retorna false.

4 - verificaTubo(L,[]): Verifica se todos os Lyfoes de um tubo são da mesma família, ou seja, se todos os elementos de uma sublista são iguais. Em caso de sucesso, chama a função **proxTubo**, que chama **verificaTubo** de volta, porém com o próximo tubo da lista. Um dos parâmetros é uma lista vazia, que é necessária para guardar o elemento anterior, que será comparado com o atual. No final esta lista é descartada, pois não há necessidade dela além desta função.
Resumidamente está regra verifica se o jogo está concluído.
Teste 1: verificaTubo([[r,r,r,r],[b,b,b,b],[]],[]). Retorna true.
Teste 2: verificaTubo([[r,r,r,r],[b,g,b,b],[]],[]). Retorna false.
Teste 3: verificaTubo([[r,r],[b,b,b,b],[r,r]],[]). Retorna false, pois apesar de cada tubo só conter Lyfoes da mesma família, uma família não pode estar separada.

5 - proxTubo(L,Hist): É sempre chamada pela função verificaTubo. Recebe a lista de tubos e o histórico de Lyfoes dos tubos anteriores. Basicamente verifica se o primeiro elemento do próximo tubo já apareceu antes, em caso negativo, chama **verificaTubo** de volta. Em caso positivo, retorna false, pois uma mesma família não pode ficar separa. Caso a lista recebida esteja vazia, retorna true, pois significa que todos os tubos já foram verificados e o jogo está resolvido.
Teste 1: proxTubo([[r,r,r,r],[b,b,b,b]],[r,r,r,r]). Retorna false, pois "r", já apareceu no histórico.
Teste 2: proxTubo([[r,r,r,r],[b,b,b,b]],[g,g,g,g]). Chama **verificaTubo**, e caso todos os outros tubos estejam corretos, retorna true.

verificaTubo e proxTubo estão ligadas uma a outra, e sempre se chamam, até chegar em true, para jogo resolvido, ou false, para jogo não resolvido.

6 - retira(L,LN,X,Ant): Retira o primeiro elemento de uma das sublistas da lista L, e devolve LN, que é a lista L sem o elemento removido, além de X, que é o elemento removido, e Ant, que é a sublista que o elemento foi removido.
Teste 1: retira([[r,b],[b,r]],LN,X,Ant). Retorna LN=[[b], [b, r]], X=r, Ant=[b]. Ou LN=[[r,b],[r]], X=b, Ant=[r].
Teste 2: retira([[r,b],[]],LN,X,Ant). Retorna LN=[[b],[]], X=r, Ant[b].
Teste 3: retira([[],[]],LN,X,Ant). Retorna false.

7 - movIdiota(Prox,Ant,X): Verifica se o movimento de remover X de Ant e inseri-lo em Prox é considerado um movimento idiota. Recebe Ant já sem o elemento X. Um movimento idiota é caracterizado por remover um Lyfoe de parte de sua familia e coloca-lo com outro Lyfoe isolado, ou em um tubo vazio.
Resumidamente esta regra serve para evitar movimentos que sejam o caminho mais longo.
Teste 1: movIdiota([r],[r,r],r). Retorna true, pois é um movimento idiota, por que é mais fácil mover o elemento isolado para junto de sua família.
Teste 2: movIdiota([],[r,r],r). Retorna true, pois é um movimento idiota, por que está removendo o Lyfoe de sua família e colocando-o em um tubo vazio.
Teste 3: movIdiota([r,r],[],r).Retornara false, pois não é um movimento idiota. Está pegando o Lyfoe isolado e juntando ele com a família.

8 - insere(L,LN,X,Ant): Insere o elemento X no topo de alguma sublista da lista L, e devolve a lista LN. Para que a inserção seja possível, o topo da sublista onde vai ser inserido o elemento deve ser igual ao elemento X, ou a sublista deve estar vazia. Além disso o tubo pode ter no máximo 3 elementos, sem contar o X. O tubo também não pode ser igual ao tubo anterior (Ant), e o movimento **não** pode ser idiota. Caso todos estes requisitos sejam satisfeitos, o elemento será inserido com sucesso.
Teste 1: insere([[r,r],[b,b],[]],LN,b,[r,r]). Retorna LN=[[r, r], [b, b, b], []] ou LN=[[r, r], [b, b], [b]].

Teste 2: insere([[r,r],[g,g]],LN,b,[r,r]). Retorna false, pois não há nenhuma possibilidade de inserção.
Teste 3: insere([[r,r],[r],[r,r]],LN,r,[r,r]). Retorna false, pois o elemento não pode ser inserido em uma lista igual a que foi removido, e o outro movimento possível é considerado idiota.
Teste 4: insere([[r,r],[]],LN,r,[]). Retorna apenas LN= [[r, r, r], []], já que o elemento não pode ser inserido de volta onde foi removido.

9 - move(L,Hist,R,Mov): Esta regra retira um Lyfoe de um tubo e insere em outro, ou seja, move um Lyfoe. L é a lista atual. Hist é o histórico de movimentos, contendo todos os estados anteriores da lista. R é onde será guardada a nova lista, com o novo estado. E Mov é a quantidade de movimentos que ainda restam.
Esta é a regra que de fato resolve o problema.
A primeira coisa que a regra faz é verificar se ainda está dentro do limite de movimentos. Então ela chama a regra **retira,** e, com a nova lista gerada por **retira,** chama a regra **insere**. Com a lista gerada por **insere**, ela verifica se aquele estado já apareceu antes no Hist. Isso não pode acontecer, pois senão tudo que está entre os dois estados iguais não valeu de nada, e essa não será uma solução otimizada. Depois de passar por esse requisitos, ela chama a função **confere**.

10 - confere(L,Hist,R,Mov): Sempre é chamada por **move.** Esta regra confere se o movimento feito por **move** resolve o jogo, chamando a função **verificaTubo**. Em caso positivo retorna em R todo o histórico de estados, sendo R uma lista, e suas sublistas os estados, e as sublistas das sublistas os tubos de cada estado. Em caso negativo, verifica se ainda está dentro do limite de movimentos, desconta 1 (um) movimento, e chama **move** de novo. Se o limite de movimentos é atingido e o problema não foi resolvido, retorna false.

Como move e confere estão ligadas uma a outra, não dá para testa-las de modo separado, por isso seus testes foram feitos juntos.

Teste 1: move([[b,b,r],[r,r],[]],[],R,3). Retorna R= [[[], [r, r, r], [b, b]], [[r], [r, r], [b, b]], [[b, r], [r, r], [b]]]. A lista fica de trás para frente.
Teste 2: move([[b,b,r],[r,r],[]],[],R,2). Retorna false, pois não dá para resolver este problema com apenas dois movimentos.
Teste 3: move([[b,b,r],[r,r]],[],R,10). Retorna false, pois nenhum movimento é possível.

11 - qntMov(X,Y): Esta regra basicamente recebe quantos tubos tem no problema (X), e devolve quantos movimentos serão permitidos (Y). Ela serve para que não sejam dados movimentos demais para um problema pequeno. A quantidade de movimentos que a regra devolve é uma quantidade segura, acima da média de movimentos que eu mesmo precisava para resolver os problemas com cada quantidade de tubos.
Teste 1: qntMov(5,Y). Retorna Y=15.
Teste 2: qntMov(11,Y). Retorna Y=44.

Teste 3: qntMov(16,Y). Retorna Y=96.

12 - imprime(L): Como **move** devolve todos os estados em R, de uma forma um pouco bagunçada para problemas maiores, a função imprime basicamente imprime cada estado em uma linha, para que resultado seja melhor visualizado pelo usuário.
Por exemplo, pegando o resultado do Teste 1 da função **move,** e aplicando na função imprime, teremos:
imprime([[[], [r, r, r], [b, b]], [[r], [r, r], [b, b]], [[b, r], [r, r], [b]]]).

[[], [r, r, r], [b, b]]
[[r], [r, r], [b, b]]
[[b, r], [r, r], [b]]
true

13.A - lyfoes(L): Esta regra recebe a lista do problema, e imprime na tela a solução passo a passo. Primeiro ela verifica se o problema já não está resolvido. Se estiver retorna true. Caso contrário, confere se todos os elementos são cores válidas, utilizando a função **ehCor,** depois confere se todos os tubos tem no máximo quadro elementos, utilizando a função **max4**, caso passe nestes requisitos, chama a função qntMov, que calcula a quantidade máxima de movimentos que serão permitidos, e chama a função **move**, que retornará a solução caso possível, ou falhará caso o problema não tenha solução. A solução de **move** fica de trás para frente, como mostrado na explicação da função **move**, então depois de receber a solução, a regra chama o predicado **reverse** para inverter a solução e deixa-la na ordem mais fácil de compreender, e por ultimo chama a função **imprime**, que imprime linha a linha o passo a passo da solução.

13.B – lyfoes(L,Mov): É igual a **lyfoes(L)**, porém recebe do próprio usuário a quantidade máxima de movimentos permitidos, permitindo que o usuário consiga resultados mais otimizados que o cálculo automático de limite de movimentos.

Teste 1: lyfoes([[b,r,b,b],[r,r,b,r],[]]). Imprime na tela:
[[b, r, b, b], [r, r, b, r], []]
[[b, r, b, b], [r, b, r], [r]]
[[b, r, b, b], [b, r], [r, r]]
[[r, b, b], [b, b, r], [r, r]]
[[b, b], [b, b, r], [r, r, r]]
[[b, b, b], [b, r], [r, r, r]]
[[b, b, b, b], [r], [r, r, r]]
[[b, b, b, b], [], [r, r, r, r]]

Teste 2: lyfoes([[b,r,b,b],[r,r,b,r],[]],5). Retorna false, pois não é possível resolver este problema com apenas 5 (cinco) movimentos.

Teste 3: lyfoes([[b,r,b,b],[r,r,b,r]],100). Retorna false, pois este é um problema impossível de ser resolvido.

Teste 4: lyfoes([[b,r,b,b],[r,r,r,b,r],[]]). Retorna false, pois um dos tubos contem mais de 4 elementos.

Teste 5: lyfoes([[b,t,b,b],[t,t,b,t],[]]). Retorna false pois contem elementos que não são uma das 10 cores válidas.

Teste 6: lyfoes([[y,b,y,r],[b,b,r,r],[y,y,r,b],[],[]]). Imprime na tela:
[[y, b, y, r], [b, b, r, r], [y, y, r, b], [], []]
[[b, y, r], [b, b, r, r], [y, y, r, b], [y], []]
[[y, r], [b, b, r, r], [y, y, r, b], [y], [b]]
[[r], [b, b, r, r], [y, y, r, b], [y, y], [b]]
[[r], [b, r, r], [y, y, r, b], [y, y], [b, b]]
[[r], [r, r], [y, y, r, b], [y, y], [b, b, b]]
[[], [r, r, r], [y, y, r, b], [y, y], [b, b, b]]
[[y], [r, r, r], [y, r, b], [y, y], [b, b, b]]
[[], [r, r, r], [y, r, b], [y, y, y], [b, b, b]]
[[y], [r, r, r], [r, b], [y, y, y], [b, b, b]]
[[], [r, r, r], [r, b], [y, y, y, y], [b, b, b]]
[[], [r, r], [r, r, b], [y, y, y, y], [b, b, b]]
[[r], [r, r], [r, b], [y, y, y, y], [b, b, b]]
[[r], [r, r, r], [b], [y, y, y, y], [b, b, b]]
[[], [r, r, r, r], [b], [y, y, y, y], [b, b, b]]
[[], [r, r, r, r], [], [y, y, y, y], [b, b, b, b]]

Teste 7: lyfoes([[p,y,y,b],[b,g,p,g],[g,b,p,b],[r,r,r,y],[r,p,y,g],[],[]]). Imprime na tela:
[[p, y, y, b], [b, g, p, g], [g, b, p, b], [r, r, r, y], [r, p, y, g], [], []]
[[y, y, b], [b, g, p, g], [g, b, p, b], [r, r, r, y], [r, p, y, g], [p], []]
[[y, b], [b, g, p, g], [g, b, p, b], [r, r, r, y], [r, p, y, g], [p], [y]]
[[b], [b, g, p, g], [g, b, p, b], [r, r, r, y], [r, p, y, g], [p], [y, y]]
[[b, b], [g, p, g], [g, b, p, b], [r, r, r, y], [r, p, y, g], [p], [y, y]]
[[b, b], [g, g, p, g], [b, p, b], [r, r, r, y], [r, p, y, g], [p], [y, y]]
[[b, b, b], [g, g, p, g], [p, b], [r, r, r, y], [r, p, y, g], [p], [y, y]]
[[b, b, b], [g, g, p, g], [b], [r, r, r, y], [r, p, y, g], [p, p], [y, y]]
[[b, b, b, b], [g, g, p, g], [], [r, r, r, y], [r, p, y, g], [p, p], [y, y]]
[[b, b, b, b], [g, p, g], [g], [r, r, r, y], [r, p, y, g], [p, p], [y, y]]
[[b, b, b, b], [p, g], [g, g], [r, r, r, y], [r, p, y, g], [p, p], [y, y]]
[[b, b, b, b], [g], [g, g], [r, r, r, y], [r, p, y, g], [p, p, p], [y, y]]
[[b, b, b, b], [], [g, g, g], [r, r, r, y], [r, p, y, g], [p, p, p], [y, y]]
[[b, b, b, b], [r], [g, g, g], [r, r, y], [r, p, y, g], [p, p, p], [y, y]]
[[b, b, b, b], [r, r], [g, g, g], [r, y], [r, p, y, g], [p, p, p], [y, y]]
[[b, b, b, b], [r, r, r], [g, g, g], [y], [r, p, y, g], [p, p, p], [y, y]]

[[b, b, b, b], [r, r, r], [g, g, g], [], [r, p, y, g], [p, p, p], [y, y, y]]
[[b, b, b, b], [r, r, r, r], [g, g, g], [], [p, y, g], [p, p, p], [y, y, y]]
[[b, b, b, b], [r, r, r, r], [g, g, g], [p], [y, g], [p, p, p], [y, y, y]]
[[b, b, b, b], [r, r, r, r], [g, g, g], [], [y, g], [p, p, p, p], [y, y, y]]
[[b, b, b, b], [r, r, r, r], [g, g, g], [], [g], [p, p, p, p], [y, y, y, y]]
[[b, b, b, b], [r, r, r, r], [g, g, g, g], [], [], [p, p, p, p], [y, y, y, y]]

Obviamente os exemplos mais complexos possuem mais de uma solução, alguns possuindo milhares, mas aqui foram mostradas apenas uma de cada, pois não há necessidade de mais.