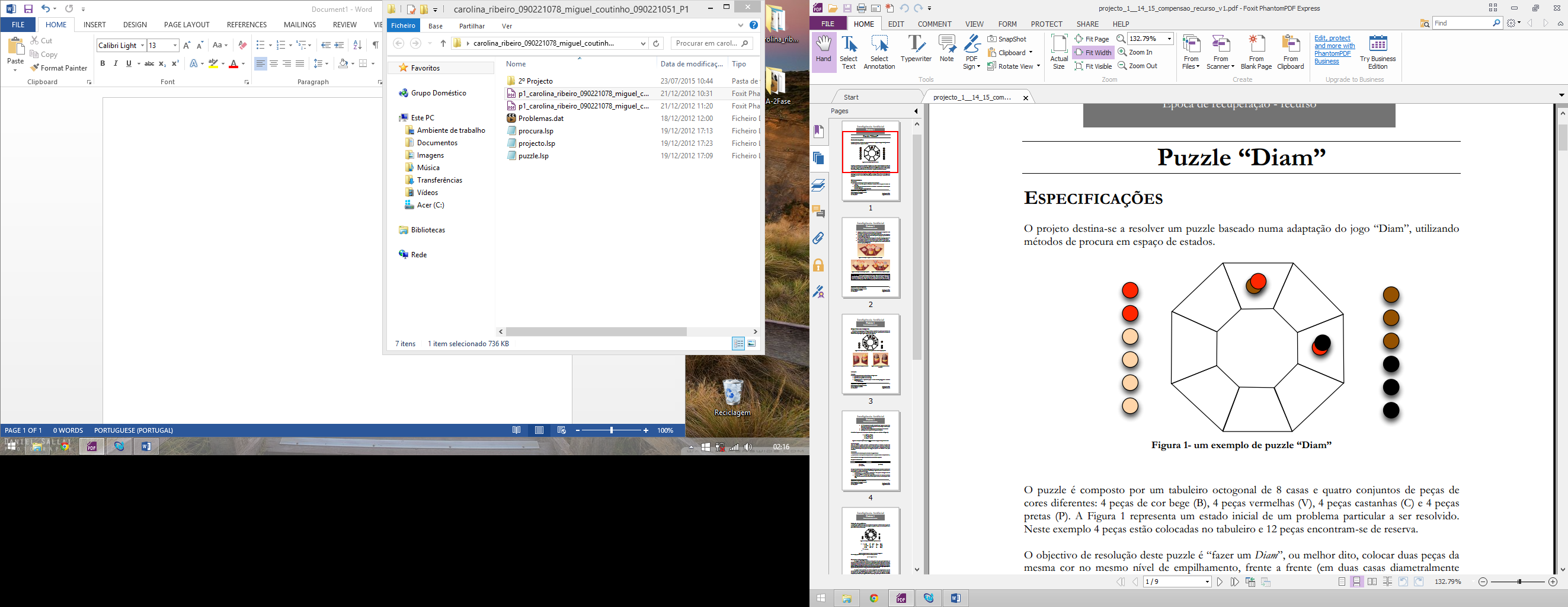
Manual Técnico

**Projeto 1**

“DIAM”

**Docentes**

Prof. Joaquim Filipe

Prof. Cedric Grueau

***Trabalho realizado por:***

Daniel Costa 120221058

Patrícia Garcia 120221008

Conteúdos

[Introdução 3](#_Toc428648444)

[Ambiente de Desenvolvimento 4](#_Toc428648445)

[Hardware 4](#_Toc428648446)

[Software 4](#_Toc428648447)

[Algoritmo Geral 5](#_Toc428648448)

[Procura Genérica 5](#_Toc428648449)

[Funções Implementadas procura.lisp 6](#_Toc428648450)

[Funções Implementadas projeto.lisp 7](#_Toc428648451)

[Funções Implementadas puzzle.lisp 8](#_Toc428648452)

[Heurísticas 10](#_Toc428648453)

[Estrutura de um nó 11](#_Toc428648454)

[Estrutura dos Resultados 11](#_Toc428648455)

[Análise de Resultados Obtidos 12](#_Toc428648456)

[Resultados e Análise Crítica 12](#_Toc428648457)

[Limitações da Aplicação 15](#_Toc428648458)

[Ficheiros do Projeto 16](#_Toc428648459)

[Procura.lisp 16](#_Toc428648460)

[Projeto.lisp 18](#_Toc428648461)

[Puzzle.lisp 22](#_Toc428648462)

[Conclusão 38](#_Toc428648463)

# Introdução

No âmbito da disciplina de Inteligência Artificial, foi proposto o desenvolvimento de um projeto com base nas aulas teóricas assim como as práticas e laboratórios, de um jogo denominado “Diam” utilizando métodos de procura em espaço de estados e técnicas de programação em Lisp.

Este jogo é baseado numa tabela octogonal de 8 casas e 4 conjuntos de peças de cores diferentes: bege, vermelho, castanho e preto.

O objetivo deste jogo é “fazer um Diam”, ou seja, colocar duas peças da mesma cor no mesmo nível de empilhamento, frente a frente (em duas casas diametralmente opostas).

Este documento contem todas as decisões tomadas pelo grupo, sobre as funções utilizadas assim como certas análises necessárias para ter uma perceção do trabalho desenvolvido.

# Ambiente de Desenvolvimento

## Hardware

A tabela contem o hardware utilizado para o desenvolvimento do projeto.

|  |  |
| --- | --- |
| Hardware | Descrição |
| Intel Core i7-4510U CPU@ 2.00GHz 4GB RAM | Computador Portátil |
| Intel Core i7-4510U CPU@ 2.00GHz 16GB RAM | Computador Portátil |

Tabela - Hardware

## Software

A tabela contem o software utilizado para o desenvolvimento do projeto.

|  |  |
| --- | --- |
| Software | Descrição |
| Windows 8.1 | Sistema Operativo |
| LispWorks Personal Edition 6.1.1 | Ferramenta de Desenvolvimento |
| Notepad++ | Editor de Texto |

Tabela - Software

# Algoritmo Geral

## Procura Genérica

Para a resolução do problema proposto foi pedido que se implementassem três algoritmos de procura: breadth-first (em largura), depth-first (em profundidade) e A\*.

* **Breadth-first** – a lista de sucessores é colocada no fim da lista de nós abertos seguindo a estrutura de dados FIFO – First in, First out;
* **Depth-first** – a lista de sucessores é colocada no início da lista de nós abertos seguindo a estrutura de dados LIFO – Last in, First out;
* **A\*** - coloca a lista de sucessores no fim da lista de nós abertos, ordenando-a de acordo com o valor de F(h) que se obtém através da soma do valor heurístico do nó - H(h) - com o valor do custo do caminho até ao nó - G(h). Se H(h) = 0 então o resultado da procura será igual ao algoritmo Breadth-first.

## Funções Implementadas procura.lisp

|  |  |  |  |
| --- | --- | --- | --- |
| Designação | Descrição | Input | Output |
| Procura-generica | Função que implementa os algoritmos de procura breadth-first, depth-first e A\* | Lista de nós abertos  Lista de nós fechados  Algoritmo  Heurística  Tempo inicial  Profundidade máxima  Limite de memória | Informação sobre a procura realizada |
| Breadth-first | Coloca a lista de nós sucessores no final da lista de nós abertos | Lista de nós abertos  Lista de nós sucessores | Lista de nós abertos |
| Depth-first | Coloca a lista de sucessores no início da lista de nós abertos | Lista de nós abertos  Lista de nós sucessores | Lista de nós abertos |
| A-ast | Insere ordenadamente a lista de nós sucessores na lista de nós abertos | Lista de nós abertos  Lista de nós sucessores | Lista de nós abertos, ordenada |
| Custo | Devolve o valor calculado para cada um dos nós dependendo da heurística escolhida | Puzzle | Valor calculado pela heuristica |
| No-visitado | Verifica se um nó gerado está na lista de nós fechados | Nó  Lista de nós fechados | T – se o nó já foi explorado  NIL – se o nó não está na lista de nós fechados |
| Insere-ordenado | Introduz um no na lista ordenado pelo seu valor de f(h) | Nó a ser introduzido  Lista de nós abertos | Lista com o nó inserido |
| Ordena | Ordena a lista de nós abertos | Lista de nós abertos | Lista de nós abertos, ordenada |
| Current-date-string | Devolve o valor da data atual do sistema | - | Data atual do sistema |

Tabela - funções implementadas ficheiro procura.lisp

## Funções Implementadas projeto.lisp

|  |  |  |  |
| --- | --- | --- | --- |
| Designação | Descrição | Input | Output |
| Menu-principal | Apresenta o menu principal com as opções do programa | - | Devolve o menu correspondente á opção pretendida |
| Carrega-puzzle | Apresenta o menu que permite escolher um puzzle para jogar | - | Imprime o puzzle escolhido e apresenta o menu de escolha do algoritmo |
| Escolhe-algoritmo | Apresenta o menu para escolher o algoritmo para efetuar a procura | Puzzle | Imprime o resultado da procura efetuada para o algoritmo Breadth-first ou outros menus no caso de outros algoritmos |
| Escolhe-profundidade | Apresenta o menu para escolher a profundidade máxima de procura | Puzzle | Imprime o resultado da procura efetuada para o algoritmo A\* |
| Escolhe-heurística | Devolve o resultado da procura efetuada com o algoritmo e a heurística escolhidos para um determinado puzzle | Puzzle  Algoritmo | Imprime o resultado da procura efetuada |
| Imprime-puzzle | Imprime o puzzle que foi escolhido para o jogo | Puzzle | Devolve o puzzle escolhido |
| Imprime-estatísticas | Imprime os valores estatísticos sobre o jogo, num ficheiro e no ecrã | Algoritmo  Heurística  Solução  Nós abertos  Nós fechados | Devolve os resultados obtidos |
| Regras | Apresenta as regras do jogo Diam | - | Imprime as regras do jogo |

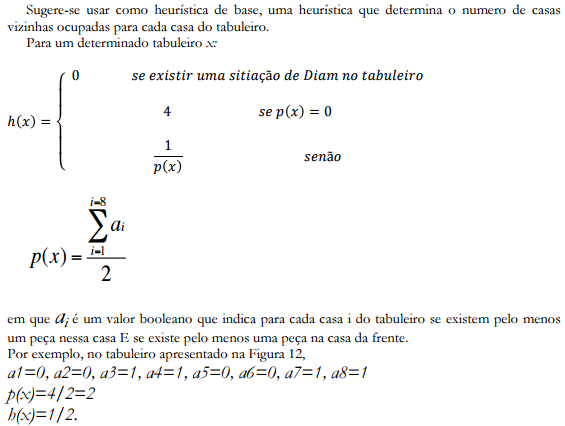
## Funções Implementadas puzzle.lisp

|  |  |  |  |
| --- | --- | --- | --- |
| Designação | Descrição | Input | Output |
| Verifica-posicao-valida | Verifica se a casa onde a peça esta colocada é valida. | Casa  Nível | T – posição válida  NIL – posição inválida |
| Get-peca-posicao | Devolve a peça que se encontra numa determinada casa e nível do puzzle. | Puzzle  Casa  Nível | B, V, C ou P |
| Verifica-tipo-peca | Verifica a cor da peça que está colocada no puzzle ou se não há nenhuma peça naquela casa/nível. | Puzzle  Casa  Nível | T – peça válida  NIL – peça inválida |
| Puzzle-antes-movimentos | Guarda o puzzle antes de se efetuar movimentação de peças. | Puzzle  Origem | Lista de peças sem a casa de origem antes de efetuar um movimento |
| Puzzle-antes-movimentos-8 | Guarda o puzzle antes de efetuar movimentação da peça quando a casa passa de 1 para 8. | Puzzle  Origem | Lista de peças sem a casa de origem antes de efetuar um movimento |
| Pecas-jogar | Devolve o estado da pilha a partir do nível pretendido | Casa  Nível | Devolve a pilha a partir do nível inserido |
| Pecas-apos-movimentação | Devolve a pilha com as peças que ficam após a jogada. | Casa  Nível | Peças que ficam na pilha após a jogada |
| Casas-frente-origem | Devolve as casas que estejam em frente á casa de origem. | Puzzle  Origem | Devolve as casas depois da casa de origem |
| Verifica-posição-destino | Verifica se a posição para onde se pretende efetuar o movimento não fica com mais do que 4 peças empilhadas. | Destino  Peças | T – se a pilha fica com menos de 4 peças  NIL – caso contrário |
| E | Efetua o movimento de deslocação para a esquerda. | Puzzle  Origem  Nível movimentação | Puzzle com o movimento já realizado |
| D | Efetua o movimento de deslocação para a direita. | Puzzle  Origem  Nível movimentação | Puzzle com o movimento já realizado |
| Posição-cor-peca | Obtém a posição da peça no puzzle através da sua cor. | Puzzle  Casa  Cor | Peça na casa pretendida |
| Remover-peca-reserva | Remove a peça de cor escolhida da lista de peças de reserva. | Puzzle  Cor | Número de peças após remover a peça da reserva |
| A | Adiciona uma nova peça ao puzzle. | Puzzle  Casa  Cor | Puzzle com o movimento já realizado |
| Calcula-tamanho-nível | Calcula o número de peças em cada nível. | Puzzle  Casa  Nível | Tamanho do nível |
| Casas-frente-frente | Devolve o número de casas que tem peças frente a frente. | Puzzle  Origem (opcional) | Número de casas que tem peças frente a frente |
| Conta-pecas | Conta as pilhas de peças nas casas. | Puzzle  Casa (opcional) | Número de pilhas com mais de 2 peças |
| Conta-casas-seguidas | Conta o número de pilhas em casas seguidas que tenham níveis distintos. | Puzzle  Casa (opcional) | Número de pilhas em casas frente a frente com níveis diferentes. |
| Conta-reservas | Conta o número de peças na lista de reserva. | Puzzle  Casa (opcional) | Número de peças na reserva |
| Heurística | Seleciona qual a heurística escolhida e efetua o calculo heurístico. | Heurística  Puzzle  Origem (opcional) | Valor calculado pela heurística inserida |
| Diam | Verifica se ocorreu diam e devolve a cor das peças em diam | Puzzle  Nível  Casa  Casa b (opcionais) | Lista com a cor das peças em diam e as suas posições |
| Verifica-objetivo | Verifica se a função diam devolve true ou false. | Puzzle  Heurística  Nível (opcionais) | Lista com a cor das peças em diam e as suas posições |
| Lista-movimentos | Junta o movimento dado pelo operador á lista de movimentos que já existem no nó, caso existam e cria uma lista nova se não existirem ainda. | Operador  Nó  Casa  Nível | Lista de movimentos efetuados |
| Sucessores-posição | Devolve os sucessores de uma determinada posição do puzzle. | Nó  Casa  Nível  Heurística  Operadores | Devolve o nó com o puzzle, a profundidade, a heurística e a lista de movimentações efetuadas |
| Sucessores-reserva | Mostra todas as adições de peças possíveis ao puzzle através das peças que se encontram na reserva | Nó  Heurística  Casa  Cor  (opcionais) | Lista de todas as jogadas possíveis com as peças na reserva |
| Sucessores-esquerda | Mostra todas as jogadas possíveis de ‘e’ | Nó  Heurística  Casa  Nível  (opcionais) | Devolve todas as jogadas possíveis das peças no puzzle para a esquerda |
| Sucessores-direita | Mostra todas as jogadas possíveis de ‘d’ | Nó  Heurística  Casa  Nível  (opcionais) | Devolve todas as jogadas possíveis das peças no puzzle para a direita |
| Constrói-no | Constrói lista com o puzzle, profundidade, heurística e lista de movimentos realizados | Nó  Casa  Nível  Heurística  Resultados  Operadores  Lista de movimentos | Devolve lista com o puzzle, a profundidade, a heurística e a lista de movimentações efetuadas |
| Sucessores | Mostra todos os movimentos possíveis em cada posição do puzzle | Nó  Heurística | Devolve todos os sucessores do puzzle |
| Casas-vizinhas-ocupadas | Função que calcula o valor da heurística pedida no enunciado. Devolve valores para a heurística pedida tendo em conta fatores como ter ocorrido diam (0); o valor devolvido da função auxiliar ser 0 (4); ou caso contrário efetua o cálculo h(x) = 1/p(x)  em que p(x) é a função auxiliar conta-casas-frente. | Puzzle  Nível  Casa  Casa-b  (opcionais) | Devolve 0, 4 ou o valor de h(x)=1/p(x). |
| Conta-casas-frente | Função que atribui valor 1 ou 0 dependendo se a casa A e a casa B (casa em frente de A) tem alguma peça - atribuindo 1. Se as casas frente-a-frente não tiverem peças atribui 0. Devolve o somatório do número de casas frente-a-frente que contem peças. | Puzzle  Nível  Casa  Casa-b  (opcionais) | Devolve 0 caso a casa e a casa-b não tenham peças (a casa-b é a casa em frente á primeira), caso contrário devolve o somatório de casas com peças frente-a-frente. |
|  |  |  |  |

# Heurísticas

As heurísticas permitem representar o grau de conhecimento que se tem do domínio de um problema, tendo em conta a aplicação e as regras ou fatores específicos da mesma aplicação.

A heurística pedida no enunciado pretende determinar o número de casas vizinhas ocupadas para cada casa do tabuleiro. A imagem em baixo representa a heurística pedida.



Em que *a1* é um valor booleano que indica para cada casa *i* do tabuleiro se existe pelo menos uma peça nessa casa e se existe pelo menos uma peça na casa da frente.

Foi criada uma heurística extra que conta as pilhas de peças nas casas com um nível superior a 2, em seguida analisa se nas casas frente a frente existem diferentes níveis uma vez que se estes se verificarem a hipótese de fazer *diam* na jogada seguinte serão maiores, por fim analisam-se as peças que se encontram na reserva para saber se é possível fazer *diam* na jogada seguinte.

# Estrutura de um nó

Um nó na aplicação é composto por quatro elementos: o puzzle, o nível, a heurística e a lista de movimentos, e estes serão detalhados em baixo.

* **Puzzle** – corresponde ao estado atual do puzzle contendo as casas e as peças em reserva;
* **Nível** – corresponde ao nível em que se encontra o nó na árvore gerado pelos algoritmos de procura;
* **Heurística** – corresponde á estimativa do caminho mais curto até á solução;
* **Lista de movimentos** – conjunto de movimentos realizados para chegar ao estado atual do puzzle.

# Estrutura dos Resultados

De forma a obter os resultados pretendidos e preencher o ficheiro das estatísticas foi necessário imprimir no mesmo os seguintes dados:

* Hora de geração;
* Heurística;
* Algoritmo;
* Nós Gerados;
* Nós Expandidos;
* Penetrância;
* Profundidade da solução;
* Solução do problema;
* Hora de fim da geração;

# Análise de Resultados Obtidos

A tabela seguinte contem informação sobre os algoritmos que resolvem os puzzles referidos no enunciado do projeto. No caso de conter X significa que o algoritmo resolve o problema, caso contenha – é porque o algoritmo não resolve o problema.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Puzzle | Breadth-First | Depth-First | A\*  (s/ heurística) | A\*  (heurística pedida) | A\*  (heurística extra) |
| Puzzle A | X | X | X | X | X |
| Puzzle B | X | X | X | X | X |
| Puzzle C | X | X | X | X | X |
| Puzzle D | X | X | X | X | X |
| Puzzle E | X | X | X | X | X |
| Puzzle F | X | X | X | X | X |
| Puzzle G | - | - | - | - | - |

Todos os algoritmos implementados resolvem todos os problemas propostos.

## Resultados e Análise Crítica

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Breadth-First | | | | | |
| Puzzle | **Nós Gerados** | **Nós Expandidos** | **Penetrância** | **Profundidade** | **Solução** |
| Puzzle A | 365 | 22 | 0.0909 | 2 | (A 1 C) (D 4 1) |
| Puzzle B | 203 | 13 | 0.0769 | 1 | (D 5 2) |
| Puzzle C | 405 | 23 | 0.0869 | 2 | (A 1 C) (D 4 1) |
| Puzzle D | 296 | 22 | 0.0909 | 2 | (D 3 1) (E 4 1) |
| Puzzle E | 48 | 4 | 0.25 | 1 | (D 5 1) |
| Puzzle F | 178 | 16 | 0.125 | 2 | (D 1 4) (D 5 2) |
| Puzzle G | - | - | - | - | - |

Tabela - Análise de Dados do Algoritmo Breadht-First

O algoritmo breadth-first encontra sempre solução nos primeiros níveis de profundidade, contudo o número de nós gerados e expandidos é geralmente elevado mas varia em função do puzzle e da ordem de geração dos nós sucessores.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Depth-First – Profundidade 5000 | | | | | | |
| Puzzle | **Nós Gerados** | **Nós Expandidos** | **Penetrância** | **Profundidade** | | **Solução** |
| Puzzle A | 244 | 35 | 1 | 35 | (A 1 C) (D 1 3) (D 2 1) (D 3 3) (D 4 2) (D 3 1) (D 5 1) (D 4 1) (D 6 4) (D 7 4) (D 6 3) (D 5 2) (D 8 4) (D 1 3) (D 2 1) (D 3 1) (D 4 1) (D 7 4) (D 8 4) (D 1 3) (D 2 1) (D 3 1) (D 4 1) (E 5 1) (D 4 2) (D 4 1) (E 5 1) (D 4 2) (D 4 1) (E 5 1) (D 4 3) (E 4 1) (D 3 2) (D 3 1) (E 4 1) | |
| Puzzle B | 186 | 22 | 1 | 22 | (A 1 C) (D 1 2) (D 2 2) (D 3 1) (D 4 3) (D 5 2) (D 4 1) (D 6 4) (D 5 3) (D 7 4) (D 6 4) (D 5 2) (D 8 4) (D 1 2) (D 2 2) (D 3 1) (D 4 1) (D 7 4) (D 6 4) (D 5 2) (D 8 4) (D 1 2) | |
| Puzzle C | 270 | 37 | 1 | 37 | (A 1 C) (D 1 3) (D 2 1) (D 3 3) (D 4 2) (D 3 1) (D 5 2) (D 4 1) (D 6 3) (D 5 3) (D 7 4) (D 8 4) (D 1 3) (D 2 1) (D 3 1) (D 4 1) (D 7 3) (D 6 3) (D 5 2) (D 8 4) (D 1 3) (D 2 1) (D 3 1) (D 4 1) (E 5 1) (D 4 2) (D 4 1) (E 5 1) (E 4 1) (D 3 2) (E 3 1) (E 2 1) (E 1 3) (E 4 1) (D 8 4) (D 1 3) (D 2 1) | |
| Puzzle D | 113 | 16 | 1 | 16 | (D 3 1) (D 4 2) (D 5 3) (D 4 1) (D 6 3) (D 5 2) (E 5 1) (E 4 1) (E 3 1) (E 2 2) (E 1 3) (E 6 1) (D 5 2) (D 5 1) (D 8 4) (D 1 3) | |
| Puzzle E | 133 | 17 | 1 | 17 | (D 3 1) (D 4 3) (D 5 3) (D 4 1) (D 7 1) (D 6 1) (D 5 1) (D 8 3) (D 1 3) (D 2 3) (D 3 1) (D 4 1) (D 7 3) (D 6 3) (D 5 1) (D 8 3) (D 1 3) | |
| Puzzle F | 38 | 3 | 1 | 3 | (D 1 4) (D 2 2) (D 1 1) | |
| Puzzle G | - | - | - | - | - | |

Tabela - Análise de dados do algoritmo depth-first c/ profundidade 5000

Conclui-se que em relação á procura em largura, esta gera normalmente menos nós, mas expande mais nós e a Penetrância da procura em profundidade é sempre 1 porque a profundidade da solução corresponde sempre ao número de nós expandidos, a profundidade da solução encontrada em si é geralmente maior que no algoritmo breadth-first onde é bastante mais reduzida.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Depth-First – Profundidade 2 | | | | | |
| Puzzle | **Nós Gerados** | **Nós Expandidos** | **Penetrância** | **Profundidade** | **Solução** |
| Puzzle A | 35 | 4 | 0.5 | 2 | (A 1 C) (D 4 1) |
| Puzzle B | 35 | 7 | 0.2857 | 2 | (A 1 C) (D 5 2) |
| Puzzle C | 37 | 4 | 0.5 | 2 | (A 1 C) (D 4 1) |
| Puzzle D | 29 | 8 | 0.25 | 2 | (D 3 1) (E 4 1) |
| Puzzle E | 25 | 3 | 0.6666 | 2 | (D 3 1) (D 5 1) |
| Puzzle F | 25 | 5 | 0.4 | 2 | (D 1 4) (D 5 2) |
| Puzzle G | - | - | - | - | - |

Tabela - Análise de dados do algoritmo depth-first c/ profundidade 2

Com profundidade 2 o algoritmo encontra sempre a solução nessa profundidade e não necessita de expandir muitos nós para encontrar a solução.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A\* - Sem Heurística | | | | | |
| Puzzle | **Nós Gerados** | **Nós Expandidos** | **Penetrância** | **Profundidade** | **Solução** |
| Puzzle A | 365 | 22 | 0.0909 | 2 | (A 1 C) (D 4 1) |
| Puzzle B | 203 | 13 | 0.0769 | 1 | (D 5 2) |
| Puzzle C | 405 | 23 | 0.0869 | 2 | (A 1 C) (D 4 1) |
| Puzzle D | 296 | 22 | 0.0909 | 2 | (D 3 1) (E 4 1) |
| Puzzle E | 48 | 4 | 0.25 | 1 | (D 5 1) |
| Puzzle F | 178 | 16 | 0.125 | 2 | (D 1 4) (D 5 2) |
| Puzzle G | - | - | - | - | - |

Tabela - Análise de dados do algoritmo A\* (sem heurística)

Os dados obtidos com o algoritmo A\* sem heurística são iguais aos dados obtidos com o algoritmo breadth-First.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A\* - Heurística Pedida | | | | | |
| Puzzle | **Nós Gerados** | **Nós Expandidos** | **Penetrância** | **Profundidade** | **Solução** |
| Puzzle A | 396 | 29 | 0.1724 | 5 | (A 2 C) (D 5 2) (E 4 2) (E 3 3) (E 7 2) |
| Puzzle B | 75 | 4 | 0.5 | 2 | (A 3 C) (D 2 2) |
| Puzzle C | 444 | 32 | 0.125 | 4 | (A 2 C) (D 4 2) (D 7 2) (D 8 4) |
| Puzzle D | 887 | 65 | 0.0461 | 3 | (E 4 2) (E 7 2) (D 6 2) |
| Puzzle E | 27 | 2 | 0.5 | 1 | (D 5 2) |
| Puzzle F | 161 | 13 | 0.2307 | 3 | (E 5 2) (D 1 4) (D 2 2) |
| Puzzle G | - | - | - | - | - |

Tabela - Análise de dados do algoritmo A\* c/ a heurística pedida

Com a heurística pedida no enunciado concluímos que o número de nós expandidos é elevado em alguns dos tabuleiros, contudo a profundidade da solução não é muito elevada o que faz com que a penetrância em si não seja maior do que 0.5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A\* - Heurística Extra | | | | | |
| Puzzle | **Nós Gerados** | **Nós Expandidos** | **Penetrância** | **Profundidade** | **Solução** |
| Puzzle A | 163 | 14 | 0.5 | 7 | (E 4 2) (A 8 C) (D 3 1) (E 7 1) (D 5 2) (D 6 1) (E 4 1) |
| Puzzle B | 155 | 11 | 0.3636 | 4 | (E 5 2) (D 2 2) (A 8 C) (E 4 2) |
| Puzzle C | 116 | 9 | 0.6666 | 6 | (A 2 C) (D 3 1) (D 7 2) (E 4 1) (D 3 2) (E 5 2) |
| Puzzle D | 177 | 15 | 0.4666 | 7 | (E 6 2) (D 3 1) (D 7 2) (E 4 1) (D 3 2) (D 4 2) (D 8 4) |
| Puzzle E | 28 | 2 | 1 | 2 | (E 4 2) (D 5 1) |
| Puzzle F | 74 | 6 | 0.3333 | 2 | (E 5 1) (E 4 2) |
| Puzzle G | - | - | - | - | - |

Tabela - Análise de dados do algoritmo A\* c/ a heurística extra

Com a heurística desenvolvida é possível perceber que a profundidade da solução encontrada é mais elevada nos primeiros puzzles que contem menos peças no puzzle e mais peças em reserva, tendo consequentemente um número de nós gerados e expandidos mais elevados. Só existe um caso em que a penetrância é igual a 1 no puzzle E.

# Limitações da Aplicação

Na implementação deste projeto ocorreram alguns percalços que nos fizeram optar por outras escolhas como por exemplo o facto de a nossa reserva de peças ser composta não só pela cor da peça em si, mas também pela quantidade de peças dessa mesma cor que existem na lista de reserva.

Não foi implementado o algoritmo de procura opcional (IDA\*, SMA ou RBFS), não por falta de conhecimento sobre o funcionamento desses algoritmos, mas por dificuldade em implementa-los concretamente.

# Ficheiros do Projeto

## Procura.lisp

;;Daniel Costa 120221058

;;Patricia Garcia 120221008

; procura.lisp

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PROCURA GENERICA

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;função de procura genérica para todos os algoritmos

(defun procura-generica (abertos fechados algoritmo heuristica inicio &optional (profundidadeMaxima 10000) (memoriaMaxima 3))

(cond

((null abertos) (format t "Não foi encontrada uma solução ~%") )

((verifica-objetivo (car (car (car abertos)))) (imprime-estatisticas algoritmo heuristica (reverse (fourth (car abertos))) abertos fechados))

((no-visitado (car abertos) fechados) (procura-generica (cdr abertos) fechados algoritmo heuristica inicio profundidadeMaxima memoriaMaxima))

((and (equal algoritmo 'depth-first) (= (second (car abertos)) profundidadeMaxima)) (procura-generica (cdr abertos) (cons (car abertos) fechados) algoritmo heuristica inicio profundidadeMaxima))

;; se o algoritmo for depth-first e o limite esteja definido

((> (+ (second (car abertos)) 1) profundidadeMaxima) (format t "Não foi encontrada uma solução! ~%"))

((equal profundidadeMaxima (cadar abertos)) (procura-generica (cdr abertos) fechados algoritmo heuristica inicio profundidadeMaxima memoriaMaxima))

(T (procura-generica (funcall algoritmo (cdr abertos)(sucessores (car abertos) heuristica))(cons (car abertos) fechados) algoritmo heuristica inicio profundidadeMaxima memoriaMaxima)

)

)

)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; ALGORITMOS

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Algoritmo de procura em breadth-first (Breath-First)

(defun breadth-first (abertos sucessores)

(append abertos sucessores)

)

;Algoritmo de procura em profundidade sem profundidade máxima

(defun depth-first (abertos sucessores)

(append sucessores abertos)

)

;Algoritmo de procura a-ast

;; ALTERADO - tinhamos sucessores abertos -> Correto: abertos sucessores = depth-first

(defun a-ast (abertos sucessores)

(sort (append abertos sucessores) #'< :key #'custo)

)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; AUXILIARES

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Funcao que retorna o valor calculado com a heuristica escolhida em cada um dos nós

(defun custo (puzzle)

(car(cdr(cdr puzzle)))

)

;;verfica se um nó já foi visitado

(defun no-visitado (no fechados)

(cond

;se a lista for null devolve nil

((null fechados) nil)

;se o tabuleiro do no for igual ao tabuleiro do primeiro no da lista de fechados é porque ja foi visita

((equal (car no) (car (car fechados))) T)

;caso contrario verifica os nós seguintes

(T (no-visitado no (cdr fechados)))

)

)

;funcao para inserir nós ordenados pelo valor f

(defun insere-ordenado (no lista)

(cond

((null lista) (list no))

((atom lista) NIL)

((< (+ (second no) (third no)) (+ (second (car lista)) (third (first lista)))) (cons no lista))

(T (cons (first lista)(insere-ordenado no (cdr lista))))

)

)

;funcao para inserir nós ordenados pelo valor f

(defun ordena (lista)

(cond

((null lista) nil)

(T (insere-ordenado (car lista) (ordena (cdr lista))))

)

)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; STRING

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;funcao para devolver o valor da data atual do sistema

(defun current-date-string ()

"Returns current date as a string."

(multiple-value-bind (sec min hr day mon yr dow dst-p tz)

(get-decoded-time)

(declare (ignore dow dst-p tz))

;;de modo aparecer a data no seguinte formato: "YYYY-MM-DD : HH:MM:SS

(format nil "~A-~A-~A : ~A:~A:~A" yr mon day hr min sec)

)

)

## Projeto.lisp

;;Daniel Costa 120221058

;;Patricia Garcia 120221008

;;Carregamento das regras e funcionalidades do jogo

(load "C:\\IA\\P1\\puzzle.lsp")

(load "C:\\IA\\P1\\procura.lsp")

;----------------------------------------------------------------------------------------------------------------------------------------------------------

;----------------------------------------------------------------------------------------------------------------------------------------------------------

;;Inicio do projecto/jogo

;;; MENU PRINCIPAL

(defun menu-principal ()

"Apresenta o menu principal com as opcões do programa"

(loop

(progn

(format t "~% ------------------------------------------------------")

(format t "~%| DIAM |")

(format t "~%| |")

(format t "~%| 1 - Resolver um puzzle |")

(format t "~%| 2 - Regras do Puzzle |")

;;(format t "~%| 3 - Mostrar um Puzzle |")

(format t "~%| 0 - Sair |")

(format t "~%| |")

(format t "~% ------------------------------------------------------")

(format t "~%~%Opcao:")

)

(cond ((not (let ((escolha (read)))

(cond

((and (< escolha 5) (> escolha 0)) (case escolha

(1 (progn (carrega-puzzle) t))

(2 (progn (regras) t))

;;(3 (progn (imprime-puzzle) t))

(0 (progn (format t "Programa Terminado") nil)))

)

( T (progn (format t "~%A opcao que selecionou nao e valida. Por favor escolha outra opcao! ~%~%Opcao: ")

(setf escolha (read))

)))

)) (return)))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CARREGA-PUZZLE - Apresenta as opções de puzzle disponíveis e faz

;;; Load do puzzle selecionado

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun carrega-puzzle ()

(format t "~%~%Selecione o puzzle que pretende resolver:~%")

(format t " -------------- ~%")

(format t " 1 - Puzzle A ~%")

(format t " 2 - Puzzle B ~%")

(format t " 3 - Puzzle C ~%")

(format t " 4 - Puzzle D ~%")

(format t " 5 - Puzzle E ~%")

(format t " 6 - Puzzle F ~%")

(format t " 7 - Puzzle G ~%")

(format t " 0 - Sair ~%")

(format t " -------------- ~%")

(format t "Opcao: ")

(let ((opcao (read)))

;; permite ler o puzzle de um ficheiro

(with-open-file (ficheiro "C:\\IA\\P1\\problemas.dat" :direction :input :if-does-not-exist :error)

(cond

((equal opcao '1) (escolhe-algoritmo (nth 0 (read ficheiro))))

((equal opcao '2) (escolhe-algoritmo (nth 1 (read ficheiro))))

((equal opcao '3) (escolhe-algoritmo (nth 2 (read ficheiro))))

((equal opcao '4) (escolhe-algoritmo (nth 3 (read ficheiro))))

((equal opcao '5) (escolhe-algoritmo (nth 4 (read ficheiro))))

((equal opcao '6) (escolhe-algoritmo (nth 5 (read ficheiro))))

((equal opcao '7) (escolhe-algoritmo (nth 6 (read ficheiro))))

((equal opcao '0) NIL)

;;como introduziu opção inválida vai surgir o menu de novo para escolher o puzzle

( T (format T "A opcao que selecionou nao e valida. Por favor escolha outra opcao! ~%") (carrega-puzzle))

)

)

)

)

;----------------------------------------------------------------------------------------------------------------------------------------------------------

;--------------------------------------------------~%----------------------------------------------------------------------------------------------------------------------------

;;Algoritmos para resolver o problema

(defun escolhe-algoritmo (puzzle)

(format T " Problema Escolhido:~%")

(imprime-puzzle puzzle)

(terpri) ;; imprimir uma linha em branco

(format t "~%~%Selecione o algoritmo para resolver o puzzle:~%")

(format t " ----------------- ~%")

(format t " 1 - Breadth-first ~%")

(format t " 2 - Depth-first ~%")

(format t " 3 - A\* ~%")

(format t " 9 - Menu anterior ~%")

(format t " 0 - Sair ~%")

(format t " ----------------- ~%")

(format t "Opcao: ")

(let ((opcao (read)))

(cond

;;escolhe algoritmo breadth-first (largura)

;; inicia a procura com algoritmo de Largura (Breath-First)

((equal opcao '1) (format t "~a"(time (procura-generica (list (append (append (append (list puzzle) (list 0)) (list 0)) (list NIL))) NIL 'breadth-first 'heuristicaN (get-universal-time) 10000 ))))

;;escolhe algoritmo depth-first (profundidade) e escolhe profundidade

((equal opcao '2) (escolhe-profundidade puzzle))

;; inicia a procura com algoritmo A\*

((equal opcao '3) (escolhe-heuristica puzzle 'a-ast))

;; Opcao de voltar atras e escolher outro puzzleuleiro

((equal opcao '5) (carrega-puzzle))

;; Sair da aplicacao

((equal opcao '0) NIL)

;;como introduziu opção inválida vai surgir o menu de novo para escolher o algoritmo

(T (format t "A opcao que selecionou nao e valida. Por favor escolha outra opcao! ~%") (escolhe-algoritmo puzzle))

)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; ESCOLHE-PROFUNDIDADE - escolha do limite de profundidade

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun escolhe-profundidade (puzzle)

(format t "Selecione a profundidade maxima:~%")

(format t "~%~%Opcao:")

(let ((profundidade (read)))

(format t "~a"(time (procura-generica (list (append (append (append (list puzzle) (list 0)) (list 0)) (list NIL))) NIL 'depth-first 'heuristicaN (get-universal-time) profundidade )))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; ESCOLHE-HEURISTICA

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun escolhe-heuristica (puzzle algoritmo)

(format t " -------------------------------------~%")

(format t " Qual a heuristica que pretende usar? ~%")

(format t " 1 - Nao quero usar heuristica ~%")

(format t " 2 - Heuristica Base ~%")

(format t " 3 - Heuristica Extra ~%")

(format t " 0 - Sair ~%")

(format t " ---------------------------------~%")

(let ((opcao (read)))

(cond

((equal opcao '1) (format t "~a"(time (procura-generica (list (append (append (append (list puzzle) (list 0)) (list 0)) (list NIL))) NIL algoritmo 'heuristicaN (get-universal-time) 10000))))

((equal opcao '2) (format t "~a"(time (procura-generica (list (append (append (append (list puzzle) (list 0)) (list 0)) (list NIL))) NIL algoritmo 'heuristica1 (get-universal-time) 10000))))

((equal opcao '3) (format t "~a"(time (procura-generica (list (append (append (append (list puzzle) (list 0)) (list 0)) (list NIL))) NIL algoritmo 'heuristica2 (get-universal-time) 10000))))

((equal opcao '0) nil)

(format T "A opcao que selecionou nao e valida. Por favor escolha outra opcao! ~%") (escolhe-heuristica puzzle algoritmo))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; IMPRIME-TABULEIRO

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun imprime-puzzle (puzzle)

(princ (car puzzle))

(princ (second puzzle))

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; IMPRIME-ESTATÍSTICAS

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun imprime-estatisticas (algoritmo heuristica solucao abertos fechados)

(with-open-file (write "C:\\IA\\P1\\estatisticas.dat"

:direction :output

:if-exists :append

:if-does-not-exist :create)

(format write "~%-----------------------------------------~%")

(format write "Gerado em ~s~%" (current-date-string))

(format write "Heuristica: ~s ~%" heuristica)

(format write "Algoritmo: ~s ~%" algoritmo)

(format write "Nos Gerados: ~s ~%" (+ (length abertos)(length fechados)))

(format write "Nos Expandidos: ~s ~%" (length fechados))

(format write "Penetrancia: ~a ~%" (float (/ (second (car abertos))(length fechados))))

(format write "Profundidade : ~s ~%" (second (car abertos)))

(format write "Solucao: ~s ~%" solucao)

(format write "Terminado em ~s~%" (current-date-string))

(format write "-----------------------------------------~%")

)

;; para teste Consola

(format t "~%-----------------------------------------~%")

(format t "Gerado em ~s~%" (current-date-string))

(format t "Heuristica: ~s ~%" heuristica)

(format t "Algoritmo: ~s ~%" algoritmo)

(format t "Nos Gerados: ~s ~%" (+ (length abertos)(length fechados)))

(format t "Nos expandidos: ~s ~%" (length fechados))

(format t "Penetrancia: ~a ~%" (float (/ (second (car abertos))(length fechados))))

(format t "Profundidade da Solução: ~s ~%" (second (car abertos)))

(format t "Solucao: ~s ~%" solucao)

(format t "Terminado em ~s~%" (current-date-string))

(format t "-----------------------------------------~%")

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; REGRAS DO JOGO

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun regras ()

(format t "

-------------------------- Regras do Puzzle DIAM ------------------------------------

| |

| O puzzle e composto por um puzzle octogonal de 8 casas e 4 conjuntos de pecas de |

| cores diferentes (bege, vermelho, castanho e preto). |

| Para resolver este puzzle e necessario fazer um 'Diam' - colocar duas pecas da mesma |

| cor no mesmo nivel de empilhamento, frente a frente (em duas casas diametralmente |

| opostas. |

| Para fazer um Diam, o nivel 1 (mais baixo) nao e tido em consideracao, somente os |

| niveis 2, 3 e 4 permitem fazer um Diam. |

| |

----------------------------------------------------------------------------------------

"

))

## Puzzle.lisp

;;;;puzzle.lisp

;;Daniel Costa 120221058

;;Patricia Garcia 120221008

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;;;

;;; VERIFICACAO DE POSICIONAMENTO DAS PECAS ;;;

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;;;

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; VERIFICA-POSICAO-VALIDA

;;; A posicao da peca e constituida pela casa onde se encontra e pelo

;;; nivel em que esta colocada.

;;; Verifica se a casa onde a peca esta colocada e menor que 1 e maior

;;; que 8, e se o nivel e menor que 1 e maior que 4

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun verifica-posicao-valida (casa nivel)

(cond

((or (< casa 1) (> casa 8) (< nivel 1) (> nivel 4)) NIL)

(T T)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; GET-PECA-POSICAO

;;; Devolve a peca que se encontra numa determinada casa e nivel do

;;; puzzle

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun get-peca-posicao (puzzle casa nivel)

(cond

((or (< (1- casa) 0)(< (- 5 nivel) 0)) 1)

((verifica-posicao-valida casa nivel) (nth (1- nivel) (nth (1- casa) puzzle)))

(T NIL)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; VERIFICA-TIPO-PECA

;;; Verifica a cor da peca que esta colocada no tabuleiro ou se não há

;;; nenhuma peca naquela casa/nivel

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun verifica-tipo-peca (puzzle casa nivel)

(or

(equal (get-peca-posicao puzzle casa nivel) 'B)

(equal (get-peca-posicao puzzle casa nivel) 'C)

(equal (get-peca-posicao puzzle casa nivel) 'V)

(equal (get-peca-posicao puzzle casa nivel) 'P)

(equal (get-peca-posicao puzzle casa nivel) ())

)

(get-peca-posicao puzzle casa nivel)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;;;

;;; MOVIMENTACAO DAS PECAS NO PUZZLE ;;;

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;;;

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; PUZZLE-ANTES-MOVIMENTOS

;;; Funcao que Guarda o puzzle antes de se efetuar movimentacao de peca

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun puzzle-antes-movimentos (puzzle origem)

(cond

((equal origem 1) nil)

(T (cons (car puzzle) (puzzle-antes-movimentos (cdr puzzle) (1- origem))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; PUZZLE-ANTES-MOVIMENTOS-8

;;; Guarda o puzzle antes de se efetuar movimentacao de peca quando a

;;; casa passa de 1 para 8

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun puzzle-antes-movimentos-8 (puzzle origem)

(cond

((equal origem 8) nil)

(T (cons (car puzzle) (puzzle-antes-movimentos-8 (cdr puzzle) (1+ origem))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; PECAS-JOGAR

;;; Devolve o estado da pilha a partir do nivel pretendido

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun pecas-jogar (casa nivel)

(cond

((> nivel (length casa)) nil)

((equal nivel 1) casa)

((equal nivel 2) (cdr casa))

((equal nivel 3) (cddr casa))

((equal nivel 4) (cdddr casa))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; PECAS-APOS-MOVIMENTACAO

;;; Devolve a pilha com as peças que ficam após a jogada

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun pecas-apos-movimentacao (casa nivel)

(cond

((> nivel (length casa)) nil)

((eq nivel 1) nil)

(T (cons (car casa) (pecas-apos-movimentacao (cdr casa) (1- nivel))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CASAS-FRENTE-ORIGEM

;;; Devolve as casas que estejam em frente a casa de origem

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun casa-frente-origem (puzzle origem)

(cond

((equal origem 1) puzzle)

(T (casa-frente-origem (cdr puzzle) (1- origem)))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; VERIFICA-POSICAO-DESTINO

;;; Verifica se a posicao para onde se quer jogar não fica com mais do

;;; que 4 peças empilhadas depois da movimentacao

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun verifica-posicao-destino (destino pecas)

(cond

((equal destino nil)(append destino pecas))

((> (+ (length pecas) (length destino)) 4) nil)

(T (append destino pecas))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; E

;;; Efetua movimentacao para a esquerda

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun e (puzzle origem nivel-movimento)

(cond

((and(equal origem 1)(not(equal(verifica-posicao-destino (nth 7(car puzzle)) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)) nil)))

(cons

(append

(list (pecas-apos-movimentacao (car (casa-frente-origem (car puzzle) origem)) nivel-movimento))

(cdr (puzzle-antes-movimentos-8 (car puzzle) origem))

(list (verifica-posicao-destino (nth 7(car puzzle)) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)))

)

(list(second puzzle)))

)

((and(> origem 1)(< origem 9)(not(equal (verifica-posicao-destino (car (casa-frente-origem (car puzzle) (- origem 1))) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)) nil)))

(cons

(append

(puzzle-antes-movimentos (car puzzle) (- origem 1))

(list (verifica-posicao-destino (car (casa-frente-origem (car puzzle) (- origem 1))) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)))

(list (pecas-apos-movimentacao (car (casa-frente-origem (car puzzle) origem)) nivel-movimento))

(cdr(casa-frente-origem (car puzzle) origem)))(list(second puzzle)))

)

(T nil)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; D

;;; Efetua movimentacao para a direita

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun d (puzzle origem nivel-movimento)

(cond

((and(equal origem 8)(not(equal(verifica-posicao-destino (car (car puzzle)) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)) nil)))

(cons

(append

(list (verifica-posicao-destino (car (car puzzle)) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)))

(cdr (puzzle-antes-movimentos (car puzzle) origem))

(list (pecas-apos-movimentacao (car (casa-frente-origem (car puzzle) origem)) nivel-movimento))

)

(list(second puzzle)))

)

((and(> origem 0)(< origem 8)(not(equal(verifica-posicao-destino (car(cdr (casa-frente-origem (car puzzle) origem)))(pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)) nil)))

(cons

(append

(puzzle-antes-movimentos (car puzzle) origem)

(list (pecas-apos-movimentacao (car (casa-frente-origem (car puzzle) origem)) nivel-movimento))

(list (verifica-posicao-destino (car(cdr (casa-frente-origem (car puzzle) origem)))(pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)));; Casa Destino

(cddr (casa-frente-origem (car puzzle) origem))

)

(list(second puzzle)))

)

(T nil)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; POSICAO-COR-PECA

;;; Obter a posicao da peca no puzzle através da sua cor

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun posicao-cor-peca(puzzle casa cor)

(cond

((equal (verifica-tipo-peca puzzle casa 1) cor) casa)

(T (posicao-cor-peca puzzle (+ casa 1) cor))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; REMOVER-PECA-RESERVA

;;; Remove a peca da cor escolhida da reserva de pecas

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun remover-peca-reserva(puzzle cor)

(1- (verifica-tipo-peca (second puzzle) (posicao-cor-peca (second puzzle) 1 cor) 2))

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; A

;;; Adicao de uma nova peca ao puzzle

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun a (puzzle casa cor)

(cond

((equal (verifica-tipo-peca (second puzzle) (posicao-cor-peca (second puzzle) 1 cor) 2) 0) nil)

(t (cons

(append ;puzzle

(puzzle-antes-movimentos (car puzzle) casa)

(list (verifica-posicao-destino (car (casa-frente-origem (car puzzle) casa))(pecas-jogar (list (verifica-tipo-peca (second puzzle) (posicao-cor-peca (second puzzle) 1 cor) 1)) 1)))

(cdr (casa-frente-origem (car puzzle) casa))

)

(list(append ;reserva

(puzzle-antes-movimentos (second puzzle) (posicao-cor-peca (second puzzle) 1 cor))

(list (cons (verifica-tipo-peca (second puzzle) (posicao-cor-peca (second puzzle) 1 cor) 1)(list(remover-peca-reserva puzzle cor))))

(cdr (casa-frente-origem (second puzzle) (posicao-cor-peca (second puzzle) 1 cor)))

))

)

)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;;;

;;; HEURISTICAS ;;;

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;;;

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CALCULA-TAMANHO-NIVEL

;;; Calcula o numero de pecas em cada nivel

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun calcula-tamanho-nivel (puzzle casa nivel)

(cond

((equal (verifica-tipo-peca puzzle casa nivel) nil) 0)

(T (1+ (calcula-tamanho-nivel puzzle casa (1+ nivel))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CONTA-PECAS

;;; Conta as pilhas de pecas nas casas

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun conta-pecas (puzzle &optional (casa 1))

(cond

((equal casa 9) 0)

((> (calcula-tamanho-nivel puzzle casa 1) 1) (1+ (conta-pecas puzzle (1+ casa) )))

((equal (calcula-tamanho-nivel puzzle casa 1) 0) (conta-pecas puzzle (1+ casa)))

(T (conta-pecas puzzle (1+ casa)))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CONTA-CASAS-SEGUIDAS

;;; Conta numero de pilhas em casas seguidas que tenham niveis distintos

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun conta-casas-seguidas(puzzle &optional (casa 1))

(cond

((equal casa 5) 0)

((and (equal casa 1)(equal (calcula-tamanho-nivel puzzle casa 1) (calcula-tamanho-nivel puzzle 5 1))) (conta-casas-seguidas puzzle (1+ casa)))

((and (equal casa 2)(equal (calcula-tamanho-nivel puzzle casa 1) (calcula-tamanho-nivel puzzle 6 1))) (conta-casas-seguidas puzzle (1+ casa)))

((and (equal casa 3)(equal (calcula-tamanho-nivel puzzle casa 1) (calcula-tamanho-nivel puzzle 7 1))) (conta-casas-seguidas puzzle (1+ casa)))

((and (equal casa 4)(equal (calcula-tamanho-nivel puzzle casa 1) (calcula-tamanho-nivel puzzle 8 1))) (conta-casas-seguidas puzzle (1+ casa)))

(T (1+ (conta-casas-seguidas puzzle (1+ casa))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CONTA-RESERVA

;;; Conta numero de pecas na lista de reserva

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun conta-reservas(puzzle &optional (casa 1))

(cond

((equal casa 5) 0)

((equal (get-peca-posicao puzzle casa 2) 0) (conta-reservas puzzle (1+ casa)))

(T (1+ (conta-reservas puzzle (1+ casa))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; HEURISTICA

;;; Seleciona qual a heuristica escolhida e efetua o calculo heuristico

;;; 'heuristicaN -> sem heuristica

;;; 'heuristica1 -> heuristica dada

;;; 'heuristica2 -> numero pilhas + pecas reserva + pilhas com niveis

;;; diferentes em casas frente-a-frente

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun heuristica(heuristica puzzle &optional(origem 1))

(cond

((equal heuristica 'heuristicaN) 1) ;; Nao existe heuristica

((equal heuristica 'heuristica1) (casas-vizinhas-ocupadas (car puzzle)))

((equal heuristica 'heuristica2) (+(conta-pecas (car puzzle))(conta-reservas (second puzzle))(conta-casas-seguidas (car puzzle))))

(T 0)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CASAS-VIZINHAS-OCUPADAS

;;; Funcao que devolve valores para a heuristica pedida tendo em conta

;;; factores como ter ocorrido diam (0); o valor devolvido da funcao

;;; auxiliar ser 0 (4); ou caso contrário efetua o cálculo h(x) = 1/p(x)

;;; em que p(x) é a função auxiliar conta-casas-frente.

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun casas-vizinhas-ocupadas (puzzle &optional (nivel 1)(casa 1) (casa-b 5))

(cond

((diam puzzle nivel casa casa-b) 0)

((equal (conta-casas-frente puzzle nivel casa casa-b) 0) 4)

(T (/ 1 (conta-casas-frente puzzle nivel casa casa-b)))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CONTA-CASAS-FRENTE

;;; Funcao que atribui valor 1 ou 0 dependendo se a casa A e a casa B

;;; (casa em frente de A) tem alguma peça - atribuindo 1. Se as casas

;;; frente-a-frente não tiverem peças atribui 0. Devolve o somatório

;;; do número de casas frente-a-frente que contem peças.

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun conta-casas-frente (puzzle &optional (nivel 2) (casa 1) (casa-b 5))

(cond

((or(equal casa 5)(equal casa-b 9)) 0)

((or(equal (verifica-tipo-peca puzzle casa nivel) nil)(equal (verifica-tipo-peca puzzle casa-b nivel) nil)) (conta-casas-frente puzzle nivel (+ casa 1) (+ casa-b 1)))

((and (not (equal(verifica-tipo-peca puzzle casa nivel) nil)) (equal(verifica-tipo-peca puzzle casa-b nivel) nil)) (conta-casas-frente puzzle nivel (+ casa 1) (+ casa-b 1)))

((and (equal(verifica-tipo-peca puzzle casa nivel) nil) (not (equal(verifica-tipo-peca puzzle casa-b nivel) nil))) (conta-casas-frente puzzle nivel (+ casa 1) (+ casa-b 1)))

(T (and (not (equal(verifica-tipo-peca puzzle casa nivel) nil)) (not (equal(verifica-tipo-peca puzzle casa-b nivel) nil))) (+ 2 (conta-casas-frente puzzle nivel (+ casa 1) (+ casa-b 1))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; DIAM

;;; Verifica se ocorreu diam e devolve a cor das pecas em diam

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun diam(puzzle &optional (nivel 2) (casa 1) (casa-b 5))

(cond

((or(equal nivel 1)(equal nivel 0)) nil)

((or(equal casa 5)(equal casa-b 9)) nil)

((or(equal (verifica-tipo-peca puzzle casa nivel) nil)(equal (verifica-tipo-peca puzzle casa-b nivel) nil)) (diam puzzle nivel (+ casa 1)(+ casa-b 1)))

((equal (verifica-tipo-peca puzzle casa nivel) (verifica-tipo-peca puzzle casa-b nivel)) (list (verifica-tipo-peca puzzle casa nivel) nivel casa casa-b))

(T (diam puzzle (1+ nivel) casa casa-b))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; VERIFICA-OBJETIVO

;;; Verifica se ocorreu o objetivo do jogo ou seja se a funcao diam

;;; devolve true ou nil

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun verifica-objetivo (puzzle &optional heuristica (nivel 2))

(cond

((null puzzle) nil)

((equal (car (diam puzzle 2)) nil) nil)

(T (diam puzzle 2))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; LISTA-MOVIMENTOS

;;; Junta o movimento dado pelo operador á lista de movimentos que já

;;; existem no nó se existirem, caso não existam cria uma lista apenas

;;; com o no do operador em questão

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun lista-movimentos (operador no casa nivel)

(cond

((null (fourth no)) (list (list (list operador casa nivel))))

(T (list (cons (list operador casa nivel) (fourth no)))

)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES-POSICAO

;;; Devolve os sucessores de uma determinada posicao do puzzle.

;;; Se nao existirem movimentos nao faz nada, se o operador devolver nil

;;; segue para o operador seguinte da l-operadores, se não, constroi o

;;; no.

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores-posicao (no casa nivel heuristica operadores)

(cond

;; Se nao existirem mais movimentos para serem verificados, termina

((null operadores) nil)

;; se o operador verificado devolver NIL, segue para o proximo operador na operadores

((not (funcall (car operadores) (car no) casa nivel)) (sucessores-posicao no casa nivel heuristica (cdr operadores) ))

;; Caso exista movimento valido para o operador verificado, ira fazer a contrucao do no

(T (constroi-no no casa nivel heuristica (funcall (car operadores) (car no) casa nivel) (cdr operadores) (lista-movimentos (car operadores) no casa nivel)))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES-RESERVA

;;; Mostra todas as adicoes de pecas possiveis ao puzzle atraves das

;;; pecas que se encontram na reserva

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores-reserva (no heuristica &optional (casa 1) (cor 1))

(cond

((and (= casa 1)(= cor 5)) nil)

((= casa 9) (sucessores-reserva no heuristica 1 (+ cor 1)))

((equal (get-peca-posicao (second(car no)) cor 1) 'P) (sucessores-reserva no heuristica casa (1+ cor)))

((equal (get-peca-posicao (second(car no)) cor 1) 'B) (sucessores-reserva no heuristica casa (1+ cor)))

((equal (get-peca-posicao (second(car no)) cor 2) 0) (sucessores-reserva no heuristica casa (1+ cor)))

((verifica-tipo-peca (second(car no)) cor 1) (append (sucessores-posicao no casa (get-peca-posicao (second(car no)) cor 1) heuristica '(a))(sucessores-reserva no heuristica (+ casa 1) cor)))

(T T)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES-ESQUERDA

;;; Mostra todas jogadas possiveis de 'e'

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores-esquerda (no heuristica &optional (casa 1) (nivel 1))

(cond

((and (= casa 9)(= nivel 1)) nil)

((= nivel 5) (sucessores-esquerda no heuristica (+ casa 1) 1 ))

((equal (get-peca-posicao (car(car no)) casa nivel) nil) (sucessores-esquerda no heuristica (1+ casa) 1))

((equal (get-peca-posicao (car(car no)) casa nivel) 'P) (sucessores-esquerda no heuristica casa (+ nivel 1)))

((equal (get-peca-posicao (car(car no)) casa nivel) 'B) (sucessores-esquerda no heuristica casa (+ nivel 1)))

((verifica-tipo-peca (car(car no)) casa nivel) (append (sucessores-posicao no casa nivel heuristica '(e))(sucessores-esquerda no heuristica casa (+ nivel 1))))

(T T)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES-DIREITA

;;; Mostra todas jogadas possiveis de 'd'

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores-direita (no heuristica &optional (casa 1) (nivel 1))

(cond

((and (= casa 9)(= nivel 1)) nil)

((= nivel 5) (sucessores-direita no heuristica (+ casa 1) 1 ))

((equal (get-peca-posicao (car(car no)) casa nivel) nil) (sucessores-direita no heuristica (1+ casa) 1))

((equal (get-peca-posicao (car(car no)) casa nivel) 'P) (sucessores-direita no heuristica casa (+ nivel 1)))

((equal (get-peca-posicao (car(car no)) casa nivel) 'B) (sucessores-direita no heuristica casa (+ nivel 1)))

((verifica-tipo-peca (car(car no)) casa nivel) (append (sucessores-posicao no casa nivel heuristica '(d))(sucessores-direita no heuristica casa (+ nivel 1))))

(T T)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES

;;; Mostra todos os movimentos possiveis em cada posicao do puzzle

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores (no heuristica)

(append(sucessores-reserva no heuristica)(sucessores-direita no heuristica)(sucessores-esquerda no heuristica))

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CONSTROI-NO

;;; Constroi a lista com o puzzle, profundidade, heuristica e lista de

;;; movimentos realizados

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun constroi-no (no casa nivel heuristica resultados operadores l-movimentos)

(cons (append (append (cons resultados (list (+ (second no) 1))) (list (heuristica heuristica resultados))) l-movimentos) (sucessores-posicao no casa nivel heuristica operadores)

)

) (cond

((> nivel (length casa)) nil)

((equal nivel 1) casa)

((equal nivel 2) (cdr casa))

((equal nivel 3) (cddr casa))

((equal nivel 4) (cdddr casa))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; PECAS-APOS-MOVIMENTACAO

;;; Devolve a pilha com as peças que ficam após a jogada

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun pecas-apos-movimentacao (casa nivel)

(cond

((> nivel (length casa)) nil)

((eq nivel 1) nil)

(T (cons (car casa) (pecas-apos-movimentacao (cdr casa) (1- nivel))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CASAS-FRENTE-ORIGEM

;;; Devolve as casas que estejam em frente a casa de origem

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun casa-frente-origem (puzzle origem)

(cond

((equal origem 1) puzzle)

(T (casa-frente-origem (cdr puzzle) (1- origem)))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; VERIFICA-POSICAO-DESTINO

;;; Verifica se a posicao para onde se quer jogar não fica com mais do

;;; que 4 peças empilhadas depois da movimentacao

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun verifica-posicao-destino (destino pecas)

(cond

((equal destino nil)(append destino pecas))

((> (+ (length pecas) (length destino)) 4) nil)

(T (append destino pecas))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; E

;;; Efetua movimentacao para a e

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun e (puzzle origem nivel-movimento)

(cond

((and(equal origem 1)(not(equal(verifica-posicao-destino (nth 7(car puzzle)) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)) nil)))

(cons

(append

(list (pecas-apos-movimentacao (car (casa-frente-origem (car puzzle) origem)) nivel-movimento))

(cdr (puzzle-antes-movimentos-8 (car puzzle) origem))

(list (verifica-posicao-destino (nth 7(car puzzle)) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)))

)

(list(second puzzle)))

)

((and(> origem 1)(< origem 9)(not(equal (verifica-posicao-destino (car (casa-frente-origem (car puzzle) (- origem 1))) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)) nil)))

(cons

(append

(puzzle-antes-movimentos (car puzzle) (- origem 1))

(list (verifica-posicao-destino (car (casa-frente-origem (car puzzle) (- origem 1))) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)))

(list (pecas-apos-movimentacao (car (casa-frente-origem (car puzzle) origem)) nivel-movimento))

(cdr(casa-frente-origem (car puzzle) origem)))(list(second puzzle)))

)

(T nil)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; D

;;; Efetua movimentacao para a d

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun d (puzzle origem nivel-movimento)

(cond

((and(equal origem 8)(not(equal(verifica-posicao-destino (car (car puzzle)) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)) nil)))

(cons

(append

(list (verifica-posicao-destino (car (car puzzle)) (pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)))

(cdr (puzzle-antes-movimentos (car puzzle) origem))

(list (pecas-apos-movimentacao (car (casa-frente-origem (car puzzle) origem)) nivel-movimento))

)

(list(second puzzle)))

)

((and(> origem 0)(< origem 8)(not(equal(verifica-posicao-destino (car(cdr (casa-frente-origem (car puzzle) origem)))(pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)) nil)))

(cons

(append

(puzzle-antes-movimentos (car puzzle) origem)

(list (pecas-apos-movimentacao (car (casa-frente-origem (car puzzle) origem)) nivel-movimento))

(list (verifica-posicao-destino (car(cdr (casa-frente-origem (car puzzle) origem)))(pecas-jogar (car (casa-frente-origem (car puzzle) origem)) nivel-movimento)));; Casa Destino

(cddr (casa-frente-origem (car puzzle) origem))

)

(list(second puzzle)))

)

(T nil)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; POSICAO-COR-PECA

;;; Obter a posicao da peca no puzzle através da sua cor

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun posicao-cor-peca(puzzle casa cor)

(cond

((equal (verifica-tipo-peca puzzle casa 1) cor) casa)

(T (posicao-cor-peca puzzle (+ casa 1) cor))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; REMOVER-PECA-RESERVA

;;; Remove a peca da cor escolhida da reserva de pecas

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun remover-peca-reserva(puzzle cor)

(1- (verifica-tipo-peca (second puzzle) (posicao-cor-peca (second puzzle) 1 cor) 2))

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; A

;;; Adicao de uma nova peca ao puzzle (a da peca)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun a (puzzle casa cor)

(cond

((equal (verifica-tipo-peca (second puzzle) (posicao-cor-peca (second puzzle) 1 cor) 2) 0) nil)

(t (cons

(append ;puzzle

(puzzle-antes-movimentos (car puzzle) casa)

(list (verifica-posicao-destino (car (casa-frente-origem (car puzzle) casa))(pecas-jogar (list (verifica-tipo-peca (second puzzle) (posicao-cor-peca (second puzzle) 1 cor) 1)) 1)))

(cdr (casa-frente-origem (car puzzle) casa))

)

(list(append ;reserva

(puzzle-antes-movimentos (second puzzle) (posicao-cor-peca (second puzzle) 1 cor))

(list (cons (verifica-tipo-peca (second puzzle) (posicao-cor-peca (second puzzle) 1 cor) 1)(list(remover-peca-reserva puzzle cor))))

(cdr (casa-frente-origem (second puzzle) (posicao-cor-peca (second puzzle) 1 cor)))

))

)

)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;;;

;;; HEURISTICAS ;;;

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;;;

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CALCULA-TAMANHO-NIVEL

;;; Calcula o numero de pecas em cada nivel

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun calcula-tamanho-nivel (puzzle casa nivel)

(cond

((equal (verifica-tipo-peca puzzle casa nivel) nil) 0)

(T (1+ (calcula-tamanho-nivel puzzle casa (1+ nivel))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CASAS-FRENTE-FRENTE

;;; Devolve o numero de casas que tem pecas frente a frente

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun casas-frente-frente (puzzle &optional (origem 1))

(cond

((null origem) 0)

(T (casas-frente-frente(cdr puzzle) (1+ origem)))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CONTA-PECAS

;;; Conta as pilhas de pecas nas casas

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun conta-pecas (puzzle &optional (casa 1))

(cond

((equal casa 9) 0)

((> (calcula-tamanho-nivel puzzle casa 1) 1) (1+ (conta-pecas puzzle (1+ casa) )))

((equal (calcula-tamanho-nivel puzzle casa 1) 0) (conta-pecas puzzle (1+ casa)))

(T (conta-pecas puzzle (1+ casa)))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CONTA-CASAS-SEGUIDAS

;;; Conta numero de pilhas em casas seguidas que tenham niveis distintos

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun conta-casas-seguidas(puzzle &optional (casa 1))

(cond

((equal casa 5) 0)

((and (equal casa 1)(equal (calcula-tamanho-nivel puzzle casa 1) (calcula-tamanho-nivel puzzle 5 1))) (conta-casas-seguidas puzzle (1+ casa)))

((and (equal casa 2)(equal (calcula-tamanho-nivel puzzle casa 1) (calcula-tamanho-nivel puzzle 6 1))) (conta-casas-seguidas puzzle (1+ casa)))

((and (equal casa 3)(equal (calcula-tamanho-nivel puzzle casa 1) (calcula-tamanho-nivel puzzle 7 1))) (conta-casas-seguidas puzzle (1+ casa)))

((and (equal casa 4)(equal (calcula-tamanho-nivel puzzle casa 1) (calcula-tamanho-nivel puzzle 8 1))) (conta-casas-seguidas puzzle (1+ casa)))

(T (1+ (conta-casas-seguidas puzzle (1+ casa))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CONTA-RESERVA

;;; Conta numero de pecas na lista de reserva

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun conta-reservas(puzzle &optional (casa 1))

(cond

((equal casa 5) 0)

((equal (get-peca-posicao puzzle casa 2) 0) (conta-reservas puzzle (1+ casa)))

(T (1+ (conta-reservas puzzle (1+ casa))))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; HEURISTICA

;;; Seleciona qual a heuristica escolhida e efetua o calculo heuristico

;;; 'heuristicaN -> sem heuristica

;;; 'heuristica1 -> heuristica dada h(x) = 1/p(x)

;;; 'heuristica2 -> numero pilhas + pecas reserva + pilhas com niveis

;;; diferentes em casas frente-a-frente

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun heuristica(heuristica puzzle &optional(origem 1))

(cond

((equal heuristica 'heuristicaN) 1) ;; Nao existe heuristica

((equal heuristica 'heuristica1) (casas-frente-frente (car puzzle)))

((equal heuristica 'heuristica2) (+(conta-pecas (car puzzle))(conta-reservas (second puzzle))(conta-casas-seguidas (car puzzle))))

(T 0)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; DIAM

;;; Verifica se ocorreu diam e devolve a cor das pecas em diam

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun diam(puzzle &optional (nivel 2) (casa 1) (casa-b 5))

(cond

((or(equal nivel 1)(equal nivel 0)) nil)

((or(equal casa 5)(equal casa-b 9)) nil)

((or(equal (verifica-tipo-peca puzzle casa nivel) nil)(equal (verifica-tipo-peca puzzle casa-b nivel) nil)) (diam puzzle nivel (+ casa 1)(+ casa-b 1)))

((equal (verifica-tipo-peca puzzle casa nivel) (verifica-tipo-peca puzzle casa-b nivel)) (list (verifica-tipo-peca puzzle casa nivel) nivel casa casa-b))

(T (diam puzzle (1+ nivel) casa casa-b))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; VERIFICA-OBJETIVO

;;; Verifica se ocorreu o objetivo do jogo ou seja se a funcao diam

;;; devolve true ou nil

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun verifica-objetivo (puzzle &optional heuristica (nivel 2))

(cond

((null puzzle) nil)

((equal (car (diam puzzle 2)) nil) nil)

(T (diam puzzle 2))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; LISTA-MOVIMENTOS

;;; Junta o movimento dado pelo operador á lista de movimentos que já

;;; existem no nó se existirem, caso não existam cria uma lista apenas

;;; com o no do operador em questão

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun lista-movimentos (operador no casa nivel)

(cond

((null (fourth no)) (list (list (list operador casa nivel))))

(T (list (cons (list operador casa nivel) (fourth no)))

)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES-POSICAO

;;; Devolve os sucessores de uma determinada posicao do puzzle.

;;; Se nao existirem movimentos nao faz nada, se o operador devolver nil

;;; segue para o operador seguinte da l-operadores, se não, constroi o

;;; no.

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores-posicao (no casa nivel heuristica operadores)

(cond

;; Se nao existirem mais movimentos para serem verificados, termina

((null operadores) nil)

;; se o operador verificado devolver NIL, segue para o proximo operador na operadores

((not (funcall (car operadores) (car no) casa nivel)) (sucessores-posicao no casa nivel heuristica (cdr operadores) ))

;; Caso exista movimento valido para o operador verificado, ira fazer a contrucao do no

(T (constroi-no no casa nivel heuristica (funcall (car operadores) (car no) casa nivel) (cdr operadores) (lista-movimentos (car operadores) no casa nivel)))

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES-RESERVA

;;; Mostra todas as adicoes de pecas possiveis ao puzzle atraves das

;;; pecas que se encontram na reserva

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores-reserva (no heuristica &optional (casa 1) (cor 1))

(cond

((and (= casa 1)(= cor 5)) nil)

((= casa 9) (sucessores-reserva no heuristica 1 (+ cor 1)))

((equal (get-peca-posicao (second(car no)) cor 1) 'P) (sucessores-reserva no heuristica casa (1+ cor)))

((equal (get-peca-posicao (second(car no)) cor 1) 'B) (sucessores-reserva no heuristica casa (1+ cor)))

((equal (get-peca-posicao (second(car no)) cor 2) 0) (sucessores-reserva no heuristica casa (1+ cor)))

((verifica-tipo-peca (second(car no)) cor 1) (append (sucessores-posicao no casa (get-peca-posicao (second(car no)) cor 1) heuristica '(a))(sucessores-reserva no heuristica (+ casa 1) cor)))

(T T)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES-ESQUERDA

;;; Mostra todas jogadas possiveis de 'e'

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores-esquerda (no heuristica &optional (casa 1) (nivel 1))

(cond

((and (= casa 9)(= nivel 1)) nil)

((= nivel 5) (sucessores-esquerda no heuristica (+ casa 1) 1 ))

((equal (get-peca-posicao (car(car no)) casa nivel) nil) (sucessores-esquerda no heuristica (1+ casa) 1))

((equal (get-peca-posicao (car(car no)) casa nivel) 'P) (sucessores-esquerda no heuristica casa (+ nivel 1)))

((equal (get-peca-posicao (car(car no)) casa nivel) 'B) (sucessores-esquerda no heuristica casa (+ nivel 1)))

((verifica-tipo-peca (car(car no)) casa nivel) (append (sucessores-posicao no casa nivel heuristica '(e))(sucessores-esquerda no heuristica casa (+ nivel 1))))

(T T)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES-DIREITA

;;; Mostra todas jogadas possiveis de 'd'

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores-direita (no heuristica &optional (casa 1) (nivel 1))

(cond

((and (= casa 9)(= nivel 1)) nil)

((= nivel 5) (sucessores-direita no heuristica (+ casa 1) 1 ))

((equal (get-peca-posicao (car(car no)) casa nivel) nil) (sucessores-direita no heuristica (1+ casa) 1))

((equal (get-peca-posicao (car(car no)) casa nivel) 'P) (sucessores-direita no heuristica casa (+ nivel 1)))

((equal (get-peca-posicao (car(car no)) casa nivel) 'B) (sucessores-direita no heuristica casa (+ nivel 1)))

((verifica-tipo-peca (car(car no)) casa nivel) (append (sucessores-posicao no casa nivel heuristica '(d))(sucessores-direita no heuristica casa (+ nivel 1))))

(T T)

)

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; SUCESSORES

;;; Mostra todos os movimentos possiveis em cada posicao do puzzle

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sucessores (no heuristica)

(append(sucessores-reserva no heuristica)(sucessores-direita no heuristica)(sucessores-esquerda no heuristica))

)

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; CONSTROI-NO

;;; Constroi a lista com o puzzle, profundidade, heuristica e lista de

;;; movimentos realizados

;;;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun constroi-no (no casa nivel heuristica resultados operadores l-movimentos)

(cons (append (append (cons resultados (list (+ (second no) 1))) (list (heuristica heuristica resultados))) l-movimentos) (sucessores-posicao no casa nivel heuristica operadores)

)

)

# Conclusão

Com o desenvolvimento da primeira parte deste projeto foi possível concluir que LISP não é uma linguagem tão fácil e intuitiva como as que estamos habituados a utilizar, mas que no entanto é bastante útil para a criação de aplicações que necessitam de mudanças de etapa em etapa. Concluímos também que é um projeto interessante de desenvolver e que poderia ter corrido ainda melhor se tivéssemos tido mais tempo para o desenvolver e um melhor conhecimento dos conceitos e da melhor forma de os aplicar.