

Faculdade de Engenharia da Universidade do Porto



Universidade do Porto
Faculdade de Engenharia
FEUP

Hasami Shogi

**Relatório Final 1º Projecto – Programação em
Lógica (3º Ano – 1º Semestre)**

2010/2011

Autores:

Carlos Tiago Rocha Babo
Felipe de Souza Schmitt
Hélder Alexandre dos Santos Moreira
ID do Grupo: 106

Resumo

Este projecto tem como objectivo implementar uma versão do jogo “Hasami Shogi” em linguagem Prolog no contexto da unidade curricular “Programação em Lógica” do 3º Ano, 1º Semestre, do Mestrado Integrado em Engenharia Informática e Computação na Faculdade de Engenharia da Universidade do Porto. Deste modo, este relatório contém uma visão geral do funcionamento do jogo, as várias formas utilizadas para abordar os problemas ao longo do desenvolvimento do trabalho, assim como as expectativas iniciais para o trabalho acompanhadas com as suas resoluções.

Abstract

This project has as its goal to implement the game “Hasami Shogi” using Prolog programming language learnt during the discipline of “Programação em Lógica” on the 1st Semester of the 3rd year in Integrated Master in Informatics and Computing Engineering at Faculty of Engineering at University of Porto. This paper has a global detail about the rules and methods of the game, the several ways of approach to the problems that we encountered during the development of the program, as well as the initial expectations and its resolutions.

Índice

Resumo	3
Abstract	4
Ilustrações.....	6
1. Introdução	7
1.1 Enquadramento	7
1.2 Motivação	7
1.3 Objectivos	7
1.4 Estrutura do relatório	8
2. Descrição do Problema	9
2.1 Conceito do jogo	9
2.1.1 Movimento das Peças.....	10
2.1.2 Capturando Inimigos	10
2.1.3 Modos de Jogo.....	11
2.1.4 Realizar um Jogo	12
3. Arquitectura do sistema.....	13
4. Módulo de Lógica do Jogo.....	14
4.1 Representação do Estado de Jogo	14
4.2 Representação de um Movimento	16
4.3 Visualização do Tabuleiro	17
4.4 Lista de jogadas válidas.....	18
4.5 Final do Jogo	18
5. Interface com o Utilizador.....	19
6. Conclusões e Perspectivas de Desenvolvimento	20
Bibliografia.....	21
Anexo A – Código Fonte	22

Ilustrações

Ilustração 1 - Movimento no tabuleiro	10
Ilustração 2 - Capturar uma peça na horizontal.....	10
Ilustração 3 - Capturar uma peça na vertical	10
Ilustração 4 - Capturar várias peças em simultâneo	11
Ilustração 5 - Capturar uma peça num canto.....	11
Ilustração 6 - Tabuleiro inicial.....	14
Ilustração 7 - Tabuleiro numa fase intermédia	15
Ilustração 8 - Tabuleiro numa situação final	15
Ilustração 9 - Tabuleiro em linha de código	17

1. Introdução

1.1 Enquadramento

O trabalho descrito neste relatório surge no âmbito da disciplina de Programação em Lógica, do 1º semestre do ano lectivo de 2010/2011 do curso de Engenharia Informática e de Computação. Tendo como objectivo principal desenvolver um jogo de tabuleiro, foi utilizado o programa Swi-Prolog para o desenvolvimento de código bem como para os testes.

1.2 Motivação

A motivação para a realização deste trabalho provém da importância para adquirir novos conhecimentos em estilos de programação diferentes do que estamos habituados a trabalhar.

Durante o desenvolvimento deste trabalho escolhemos um jogo que fosse divertido e interessante assim como fizesse com que o utilizador tivesse que se manter concentrado durante o jogo pois é necessário realizar estratégias para se obter um melhor resultado. Uma das principais razões da nossa motivação foi a programação da inteligência artificial pois é algo do qual é considerado interessante e o resultado é gratificante.

No final espera-se com uma grande expectativa a implementação da interface gráfica a realizar na disciplina de Laboratório de Aplicações com Interface Gráfica, desta forma o trabalho tornar-se-á mais interessante devido ao aspecto visual.

1.3 Objectivos

Os objectivos deste projecto relacionam-se essencialmente com o estudo e compreensão da programação em lógica, recorrendo ao Prolog, em contraste à programação funcional já estudada.

Os objectivos específicos para este projecto são:

- Realização de uma versão do jogo “Hasami Shogi” em linguagem prolog;
- Condições de terminação do jogo correctamente definidas;
- Vários modos de jogo: Jogador vs Jogador, Jogador vs Computador e Computador vs Computador;

- Modo de jogo com Inteligência Artificial aplicada com vários graus de dificuldade ao utilizar diferentes algoritmos;
- Aplicação “*User-friendly*” para facilitar a interface para o utilizador;
- Futura adaptação da visualização de gráficos 3D através da disciplina de LAIG com a utilização de sockets.

No final do projecto esperamos conseguir discernir quais os problemas que podem ser resolvidos recorrendo à programação em lógica e, efectivamente, encontrar soluções. Para além disso, e em relação à vertente interdisciplinar deste projecto, pretendemos ser capazes de ligar duas linguagens de programação diferentes (Prolog e C++), recorrendo a *sockets*.

1.4 Estrutura do relatório

O relatório está dividido em 6 capítulos, o primeiro capítulo é composto por uma introdução do trabalho, definição dos objectivos e enquadramento do projecto assim como a motivação para o mesmo.

No segundo capítulo é apresentada a descrição do projecto a desenvolver assim como o seu conceito, as regras e a história do jogo.

No terceiro capítulo é descrito a arquitectura do sistema e os seus principais módulos do programa.

O quarto capítulo contém a descrição do projecto assim como a sua implementação, representação do estado do tabuleiro, validações de jogadas, verificação de fim de jogo, entre outras.

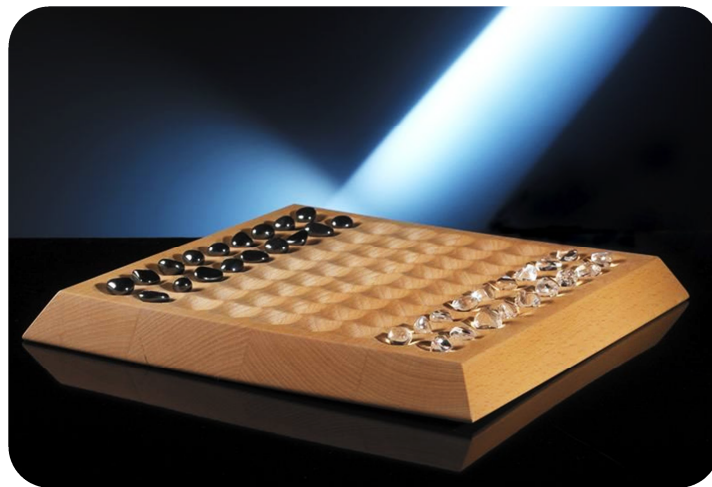
Já o quinto capítulo é sobre a interface de texto com o utilizador, a sua descrição e implementação.

Por ultimo, no sexto capítulo apresenta-se as conclusões do trabalho e resultados obtidos assim como perspectivas de desenvolvimentos futuros na utilização da linguagem Prolog.

2. Descrição do Problema

2.1 Conceito do jogo

Hasami Shogi (はさみ将棋 *hasami shōgi*, *sandwiching chess*) é uma variante do jogo popular japonês denominado GoBang, que se tornou popular ao ser jogado nas ruas pelas crianças, pelo facto de não ser necessário um tabuleiro, podendo ser jogado utilizando simplesmente lápis e papel. O nome Shogi provém do tabuleiro (Shogi-ban), que para além de ser constituído por 9X9 espaços, é utilizado em diversos jogos.



Para as peças são usadas as 9 *fhuyo* (peões) brancas e 9 pretas, provenientes de um *shogi set* sendo que estas são posicionadas inicialmente na linha mais próxima do seu jogador.

Este jogo torna-se interessante, chamando-nos à atenção, visto que possui um conjunto de regras extremamente simples e que podem ser aprendidas rapidamente. No entanto, o Hasami Shogi é bastante aliciante devido à simplicidade aliada às múltiplas estratégias possíveis, mas que requerem grande concentração e previsão do desenrolar das jogadas.

2.2 Movimento das Peças

O objectivo do Hashami Shogi é simples: conseguir retirar as peças do adversário prensando-as com as nossas. Para isso, podemos mover as peças para a frente, trás, esquerda, e direita, sem restrições de espaço.

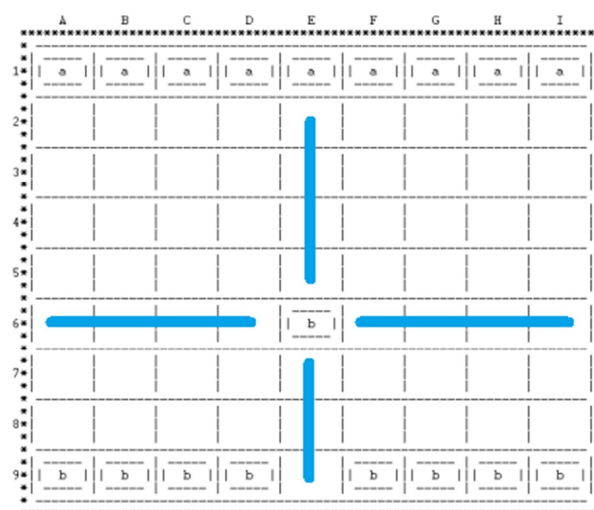


Ilustração 1 - Movimento no tabuleiro

2.2.1 Capturando Inimigos

O processo de captura passa por rodear a peça inimiga com duas nossas (a diagonal não é considerada). No entanto, se o inimigo mover uma peça para um espaço entre duas peças nossas, a captura não é efectuada.

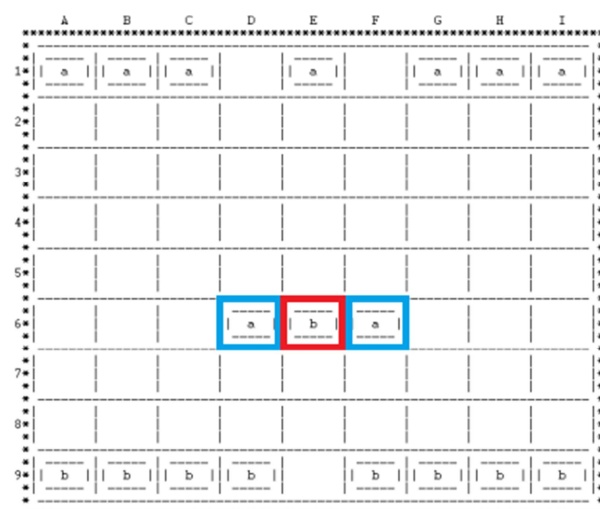


Ilustração 2 - Capturar uma peça na horizontal

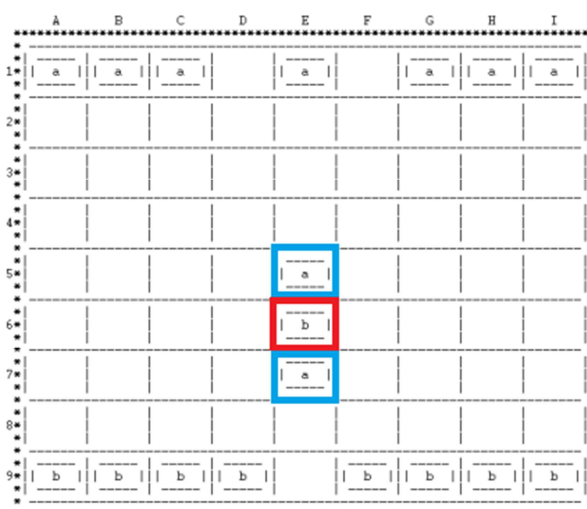


Ilustração 3 - Capturar uma peça na vertical

É também possível usar a mesma técnica com várias peças inimigas.

	A	B	C	D	E	F	G	H	I
1	a	a		a	a	a		a	a
2									
3									
4									
5									
6			a	b	b	b		a	
7									
8									
9	b	b	b				b	b	b

Ilustração 4 - Capturar várias peças em simultâneo

O mesmo acontece se uma peça (e apenas uma) for encurralada num canto.

	A	B	C	D	E	F	G	H	I
1	a	a	a	a	a	a			
2									
3									
4									
5									
6								b	
7									
8								a	a
9	b	b	b	b	b	b	b	a	b

Ilustração 5 - Capturar uma peça num canto

2.2.2 Modos de Jogo

O projecto tem três modos de jogo:

- Jogador vs Computador;
- Jogador vs Jogador;
- Computador vs Computador.

Nos modos de jogo onde incluíam o computador poderá escolher vários níveis de dificuldades.

Neste projecto existem três tipos de dificuldades diferentes programadas com os seguintes diferentes algoritmos:

- [Dificuldade 1] Random
 - Este algoritmo consiste em através da regra findall obter uma lista com todas as jogadas possíveis e realizar um random entre zero e o tamanho dessa lista e realizar uma dessas jogadas.
- [Dificuldade 2] Greedy
 - Após realizar as mesmas operações para obter a lista com todas as jogadas possíveis este verifica qual das jogadas irá capturar um maior número de inimigos e cria uma lista com as possibilidades em que este número seja o mais elevado. Posteriormente realiza um random para escolher uma dessas jogadas, verificando se alguma dessas jogadas irão provocar uma captura por parte do outro jogador.
- [Dificuldade 3] Minimax
 - Nesta dificuldade é utilizada o algoritmo de minimax de forma a minimizar a perda máxima possível, desta forma o computador pode calcular as várias jogadas possíveis com vários níveis de profundidades de forma a obter um melhor resultado.

2.2.3 Realizar um Jogo

O primeiro jogador é sorteado aleatoriamente e o primeiro a ficar com apenas duas peças em jogo, perde.

3. Arquitectura do sistema

O jogo é exectuda da seguinte forma:

- Após a execução do programa é visualizado um menu onde são disponibilizadas as seguintes opções de jogo, Jogador vs Jogador, Jogador vs CPU, CPU vs CPU e Como Jogar?;
- De seguida é apresentado o tabuleiro com 9 peças para cada jogador nas suas posições iniciais, assim como é aleatoriamente designado qual o primeiro jogador a começar a jogar;
- É assim inicializado um novo jogo, onde existe um loop principal que irá promover a continuidade do jogo até ao seu fim, numa função onde é recebido o tabuleiro, jogador e o nível de dificuldade;
- Em cada jogada é verificado se os movimentos inseridos pelo utilizador são válidos, caso o sejam é realizada a jogada e é impresso o novo tabuleiro no ecrã, caso contrário é pedido novamente ao utilizador as coordenadas do a posição de destino.

4. Módulo de Lógica do Jogo

4.1 Representação do Estado de Jogo

O tabuleiro é constituído por oitenta e uma células, das quais dezoito se encontram inicialmente preenchidas. As peças são posicionadas de acordo com a seguinte imagem:

	A	B	C	D	E	F	G	H	I
1	a	a	a	a	a	a	a	a	a
2									
3									
4									
5									
6									
7									
8									
9	b	b	b	b	b	b	b	b	b

Ilustração 6 - Tabuleiro inicial

Matriz inicial do tabuleiro:

```
tabuleiro(  
    [ [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ],  
      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
      [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ] ] ).
```

Para construir o tabuleiro recorreremos a uma lista de listas, onde o valor 1 e 2 indicam os jogadores e o 0 a posição vazia. Cada número tem associado um desenho específico, para que seja mais fácil e intuitivo jogar.

Com o decorrer do jogo, muitas das peças deixam o tabuleiro e num passo intermédio, o tabuleiro assemelha-se a isto:

	A	B	C	D	E	F	G	H	I
1	a					a	a		a
2									
3									
4			a						
5				a	a	b			
6						b			
7		b							
8									
9	b			b	b		b		b

Ilustração 7 - Tabuleiro numa fase intermédia

Já num estado final, quando apenas restarem duas peças de um dos lados, o jogo termina:

	A	B	C	D	E	F	G	H	I
1	a								
2									
3									
4		a							
5						b			
6						b			
7		b					b		
8									
9									

Ilustração 8 - Tabuleiro numa situação final

Neste caso, ganhou o jogador b.

Para se deslocar no tabuleiro, o jogador deve indicar a peça que pretende mover e o seu destino. Para isso, o tabuleiro contém indicações - alfabeto para as colunas e números para as linhas - para padronizar o método de jogo.

4.2 Representação de um Movimento

Tal como já referido nas regras, no Hasami Shogi o jogador pode efectuar as jogadas na direcção vertical e horizontal, de quantas casa quiser, tendo em conta as peças do adversário no caminho e o limite do tabuleiro. Assim, cada jogada é representada pela peça a ser movida e pelas coordenadas para onde se pretende deslocá-la.

O processo associado a cada jogada envolve três passos: validação da jogada, verificando se obedece às restrições descritas anteriormente; movimento definitivo da peça para a sua nova posição no tabuleiro, e, finalmente, é verificado se há peças do adversário conquistadas.

Deste modo, o primeiro passo chama a regra **verificaPeca(Tabuleiro,Xf,Yf,0)**, verificando se a posição para qual o jogador deseja deslocar é uma casa vazia, de seguida é chamada a mesma regra para a posição X e Y e verifica se a peça que deseja mover pertence ao jogador actual. O primeiro argumento desta regra é a matriz do tabuleiro, onde é guardada toda a informação sobre as peças e as suas posições. Por conseguinte, e depois de confirmar-se que existe uma peça pertencente ao jogador actual na posição (X,Y) e que a posição (Xf,Yf) encontra-se vazia irá ser chamada a regra – **verificaCaminho(Jogador,X,Y,Xf,Yf,Tabuleiro)**, onde é verificado se existe alguma peça entre a posição inicial e a posição final, se o movimento é apenas vertical ou horizontal e caso estas condições se verifiquem a jogada é válida, logo irá ser chamada a regra **muda_tab(0, Jogador, Xf, Yf, T, Tnovo)** onde a posição final na matriz passa a ter a peça do jogador e de seguida esta regra é chamada novamente para que a posição inicial do jogador passe a 0 pois ficou vazia. De seguida é verificado através da chamada da regra **conquistaPecas(Tabuleiro, NovoTab, Jogador)** onde irá ser processado todas as conquistas que possam ter ocorrido durante essa jogada.

Se houver peças conquistadas, é feita a actualização do tabuleiro e da estrutura que guarda o número de peças conquistadas pelo jogador recorrendo à regra **processaRemocoes([X-Y-Xf-Yf|T], Tabuleiro, NovoTabuleiro, Jogador)**. Por fim, é verificado se o jogo foi terminado através da regra **terminouJogo(Tnovo, Jogador2)** caso o final do jogo ainda não tenha ocorrido a regra do modo de jogo é chamada novamente.

4.3 Visualização do Tabuleiro

Para desenhar o tabuleiro, recorreremos a uma série de factos, regras e à recursividade. A regra principal para o seu desenho é a seguinte:

```
desenha(Tabuleiro) :-  
    linhaLetrasV(X),  
    linhaLetras, nl,  
    linhaLimite, nl,  
    linhaDivH, nl,  
    printPecas(Tabuleiro,X),  
    linhaLimite.
```

Basicamente é criado um novo tabuleiro e todos os componentes desenhados. Depois de ser criado um novo tabuleiro e desenhados os limites superiores, é chamada a regra **printPecas(T,X)** que recebe o tabuleiro e percorre a matriz de jogo, imprimindo as peças respectivas:

	A	B	C	D	E	F	G	H	I
1	a	a	a	a	a	a	a	a	a
2									
3									
4									
5									
6									
7									
8									
9	b	b	b	b	b	b	b	b	b

Ilustração 9 - Tabuleiro em linha de código

As funções **printLinhaPecaX(Y)** auxiliam o desenho das três componentes de cada peça. O código completo do tabuleiro pode ser encontrado em anexo.

4.4 Lista de jogadas válidas

Para listar todas as jogadas válidas utilizamos a regra **findall(X-Y-Xf-Yf, verificaCaminho(Jogador, X,Y,Xf,Yf, T), L)** que devolve uma lista com todas as jogadas possíveis para todas as peças do jogador em que Xf e Yf são respectivamente a coluna e a linha onde é possível mover a peça da posição X e Y.

4.5 Final do Jogo

O final do jogo é verificado quando um dos jogadores só possui duas peças no tabuleiro, neste caso o jogador oposto é declarado como vencedor.

Esta verificação é realizada no fim de cada ciclo de jogada através da seguinte regra:

```
terminouJogo(T,Jogador) :-  
terminouJogoaux(T,1,1,0,Jogador).  
terminouJogoaux(T,9,9,NPecas,Jogador):-!, NPecas <  
3,desenha(T),nl,nl,nl,  
    write('Terminou o jogo. O jogador '),  
    troca(Jogador, Jogador2), write(Jogador2), write(''  
venceu.''),nl,nl,nl.  
  
terminouJogoaux(T,X,Y,NPecas,Jogador) :-  
    if(verificaPeca(T, X, Y, Jogador), NPecasNovo is  
NPecas+1, NPecasNovo is NPecas),  
    if(X == 9, (X1 is 1, Y1 is Y+1), (X1 is X+1, Y1 is  
Y)),  
    terminouJogoaux(T,X1,Y1,NPecasNovo, Jogador).
```

5. Interface com o Utilizador

O projecto foi planeado de forma a que a sua interface gráfica ser simples e de fácil acesso.

Inicialmente são apresentadas ao utilizador várias escolhas num menu numerado, com as opções de jogo: 'Jogador vs Jogador', 'Jogador vs CPU', 'CPU vs CPU' e 'Como jogar?'.

De seguida ao entrar num modo de jogo o tabuleiro é impresso no ecrã onde o utilizador pode ver as suas peças, assim como as do utilizador oposto. Também é apresentado o nome do jogador actual e é pedido a posição da peça que o utilizador deseja mover seguido da posição final dessa mesma peça. Quando o jogo acaba é apresentado no ecrã o jogador vencedor.

No futuro será utilizada através de sockets uma interface gráfica 3D no âmbito da disciplina de LAIG, o que irá melhorar significativamente a interacção com o utilizador assim como tornar a sua interface mais interessante.

6. Conclusões e Perspectivas de Desenvolvimento

Foi realizado com sucesso uma versão do jogo “Hasami Shogi” em linguagem prolog. Na fase final do trabalho é possível realizar jogos de contra outros jogadores ou até mesmo contra o computador em vários níveis de dificuldade devido à programação de vários algoritmos de inteligência artificial em Prolog. Foi realizada uma interface de texto simples e intuitiva de forma a ser fácil para o utilizador adaptar-se à interface do jogo.

Com isto é possível afirmar que todos os objectivos inicialmente propostos foram cumpridos com sucesso.

O trabalho foi um processo muito importante na aquisição de conhecimentos relativos à linguagem Prolog de uma forma contínua. Desta forma possibilitando o desenvolvimento de futuros projectos utilizando a linguagem Prolog através dos conhecimentos obtidos ao longo do desenvolvimento do projecto.

Bibliografia

- [1] Hasami Shogi, http://en.wikipedia.org/wiki/Hasami_shogi (consultado em 30-09-2010)
- [2] Hasami Shogi Web Game, <http://www.afsgames.com/e/shogi.htm> (consultado em 01-10-2010)
- [3] Shogi, <http://en.wikipedia.org/wiki/Shogi> (consultado em 30-09-2010)
- [4] Leon Sterling e Ehud Shapiro, The Art of Prolog Second Edition – Advanced Programming Techniques: Capitulo 20, Secção 2 “Searching Game Trees”.


```
printLinhaPeca([]).
printLinhaPeca([A | R]):-
    piece1(A),
    printLinhaPeca(R).
```

```
printLinhaPeca3([]).
printLinhaPeca3([A | R]):-
    piece3(A),
    printLinhaPeca3(R).
```

```
printLinhaPeca2([]).
printLinhaPeca2([A | R]):-
    piece2(A),
    printLinhaPeca2(R).
```

```
printLinha([]).
printLinha([A | R]):-
    write(A),
    printLinha(R).
```

```
printPecas([],[]).
printPecas([A | R],[X | Y]):-
    write(' * | '),
    printLinhaPeca(A),
    write('*'), nl,
    write(X),
    write('* | '),
    printLinhaPeca2(A),
    write('*'), nl,
    write(' * | '),
    printLinhaPeca3(A),
    write('*'), nl,
    linhaDivH, nl,
    printPecas(R, Y).
```

```
desenha(Tabuleiro):-  
    linhaNumerosV(X),  
    linhaLetras, nl,  
    linhaLimite, nl,  
    linhaDivH, nl,  
    printPecas(Tabuleiro,X),  
    linhaLimite.
```

%FIM DO DESENHO DO TABULEIRO

%MENU DO JOGO

```
menu:-  
    write('1 - Jogador vs CPU'),nl,  
    write('2 - Jogador vs Jogador'),nl,  
    write('3 - CPU vs CPU'),nl,  
    write('stop - para sair'),nl,  
    write('Escolha uma opcao'),nl,  
    read(X),  
    verifica(X).
```

```
verifica(1):-jVsCpu, !.  
verifica(2):-tabuleiro(T), desenha(T), !.  
verifica(3):-tabuleiro(T), desenha(T), !.  
verifica(_).
```

```
jVsCpu:-  
    write('1 - Fácil'),nl,  
    write('2 - Intermédio'), nl,  
    write('3 - Difícil'),nl,  
    write('stop - para sair'),nl,  
    read(X),  
    write(X),  
    dificuldade(X).
```

```
dificuldade(1):-tabuleiro(T), modoJogador(T, 1, 1), !.  
dificuldade(2):-tabuleiro(T), modoJogador(T, 1, 2), !.  
dificuldade(3):-jVsCpu, !.  
dificuldade(_).
```

%FIM DO DESENHO DO JOGO

%AVALIACAO DA JOGADA%

troca(1,2).

troca(2,1).

if(Condition, TrueClause, FalseClause) :-

 Condition, !, TrueClause;

 !, FalseClause.

interacaoJogador(Y,X,Yf,Xf,Jogador):-

 write('Jogador actual: '),

 write(Jogador),nl,

 write('Peca a mover:') ,nl,

 write('Linha (Ex: 1) : '),

 read(Y),

 nl,

 write('Coluna (Ex: A) : '),

 read(Xt),

 letra(Xt, X),

 write('Posicao desejada:'),

 nl,

 write('Linha (Ex: 1) : '),

 read(Yf),

 nl,

 write('Coluna (Ex: A) : '),

 read(Xt2),

 letra(Xt2, Xf).

modoJogador(T, Jogador, ModoCPU):-

 Jogador == 1,

 desenha(T), nl,

 interacaoJogador(Y,X,Yf,Xf,Jogador),

 if((verificaPeca(T,Xf,Yf,0),verificaPeca(T,X,Y,Jogador),

verificaCaminho(Jogador,X,Y,Xf,Yf,T)),

 (muda_tab(0,Jogador,Xf,Yf,T,TNovo),

 muda_tab(Jogador,0,X,Y,TNovo,TNovo2),

 troca(Jogador, Jogador2),

 conquistaPecas(TNovo2, TNovo3, Jogador),

 conquistaPecas(TNovo3, TNovo4, Jogador2),

 if(terminouJogo(TNovo4,Jogador2),menu,modoCPU(TNovo4,Jogador2,

ModoCPU))), modoJogador(T, Jogador,ModoCPU)).

% CICLO DO BOT DEPENDENDO DA DIFICULDADE ESCOLHIDA

modoCPU(T,Jogador, ModoCPU):-

```
    ModoCPU == 1,  
    Jogador == 2,  
    findall(X-Y-Xf-Yf, verificaCaminho(Jogador,X,Y,Xf,Yf, T), L),  
    choose(L, M),  
    modificaT(Jogador,M,T, TNovo2),  
    conquistaPecas(TNovo2, TNovo3, Jogador),  
    conquistaPecas(TNovo3, TNovo4, Jogador2),  
    troca(Jogador, Jogador2),  
    if(terminouJogo(TNovo4,Jogador2),menu,modoJogador(TNovo4,Jogador2,  
ModoCPU)).
```

% MODO INTERMEDIO

modoCPU(T, Jogador, ModoCPU):-

```
    ModoCPU == 2,  
    Jogador == 2,  
    greedy(T,L,1,X, Jogador),  
    L == X,l,  
    findall(X-Y-Xf-Yf, verificaCaminho(Jogador,X,Y,Xf,Yf, T),L1),  
    choose(L1, M),  
    modificaT(Jogador,M,T, TNovo2),  
    conquistaPecas(TNovo2, TNovo3, Jogador),  
    conquistaPecas(TNovo3, TNovo4, Jogador2),  
    troca(Jogador, Jogador2),  
    if(terminouJogo(TNovo4,Jogador2),menu,modoJogador(TNovo4,Jogador2,  
ModoCPU)).
```

```

modoCPU(T, Jogador, ModoCPU):-
    ModoCPU == 2,
    Jogador == 2,
    greedy(T,L,1,X,Jogador),
    \+ L == X,
    choose(L, M),
    modificaT(Jogador,M,T, TNovo2),
    conquistaPecas(TNovo2, TNovo3, Jogador),
    conquistaPecas(TNovo3, TNovo4, Jogador2),
    troca(Jogador, Jogador2),
    if(terminouJogo(TNovo4,Jogador2),menu,modoJogador(TNovo4,Jogador2,
ModoCPU)).

```

```

greedy(T,L, P, _, J):-
    findall(X-Y-Xf-Yf,(verificaCaminho(J,X,Y,Xf,Yf,T),modificaT(J,X-Y-Xf-
Yf,T,TNovo2),conquistas(J,_,_,_,TNovo2,1,N),N>P),L1),
    \+ L1 = [],
    P1 is P+1,
    greedy(T,L,P1,L1,J), !.
greedy(_,L, _, L,_).

```

% FIM DO BOT INTERMEDIO

```

modificaT(J,X-Y-Xf-Yf,T,TNovo):-
    muda_tab(J,0,X,Y,T,NovoTab),
    muda_tab(0,J,Xf,Yf,NovoTab,TNovo).

```

```

choose([], []).
choose(List, Elt) :-
    length(List, Length),
    random(0, Length, Index),
    nth0(Index, List, Elt).

```

% VERIFICA SE A PECA E' DO UTILIZADOR

verificaPeca(T,X,Y,Jogador) :- verificaPecaAux(T,X,Y,Jogador,1).

verificaPecaAux([T|_],X,Y,Jogador,Y) :-

verificaPecaLinha(T,X,Jogador, 1).

verificaPecaAux([_|R],X,Y,Jogador,Linha) :-

Linha2 is Linha+1,

verificaPecaAux(R,X,Y,Jogador,Linha2).

verificaPecaLinha([Jogador|_], X, Jogador, X).

verificaPecaLinha([_|R], X, Jogador, Coluna) :-

N1 is Coluna+1,

verificaPecaLinha(R, X, Jogador, N1).

% VERIFICA SE HA UM CAMINHO ENTRE PECAS

adjacente(X1,Y1,X2,Y2):-X1 == X2, Y1 == Y2+1.

adjacente(X1,Y1,X2,Y2):-X1 == X2, Y1 == Y2-1.

adjacente(X1,Y1,X2,Y2):-Y1 == Y2, X1 == X2+1.

adjacente(X1,Y1,X2,Y2):-Y1 == Y2, X1 == X2-1.

adjacente2(X1,Y1,X2,Y2):-X1 == X2, Y1 == Y2+1.

adjacente2(X1,Y1,X2,Y2):-Y1 == Y2, X1 == X2+1.

adjacente3(X1,Y1,X2,Y2):-X1 == X2, Y1 == Y2-1.

adjacente3(X1,Y1,X2,Y2):-Y1 == Y2, X1 == X2-1.

diagonal(X1,Y1,X2,Y2):-X1==X2, Y1\==Y2.

diagonal(X1,Y1,X2,Y2):-Y1==Y2, X1\==X2.

verificaCaminho(Jog, X,Y,Xf,Yf, Tab):-

verificaPeca(Tab,X,Y,Jog),

verificaPeca(Tab,Xf,Yf,0),

adjacente(X,Y,Xf,Yf).

verificaCaminho(Jog, X,Y,Xf,Yf, Tab):-

verificaPeca(Tab, X,Y, Jog),

verificaPeca(Tab, Xf,Yf, 0),

verificaPeca(Tab, Xa, Ya, 0),

diagonal(X,Y,Xf,Yf),

adjacente(Xa,Ya,X,Y),

muda_tab(0,Jog,Xa,Ya,Tab,NovoTab),

verificaCaminho(Jog, Xa, Ya, Xf, Yf, NovoTab).

% VERIFICA SE HA PECAS CONQUISTADAS

% PROCESSA TODAS AS CONQUISTAS

```
conquistaPecas(Tab, NovoTab, Jog):-  
    findall(X-Y-Xf-Yf, conquistas(Jog, X,Y,Xf,Yf,Tab,1,_), L),  
    processaRemocoes(L, Tab, NovoTab, Jog).
```

% CHAMA A FUNCAO PARA REMOVER AS PECAS PARA CADA CONQUISTA

```
processaRemocoes([], Tab, Tab, _).
```

% CONQUISTA NA VERTICAL

```
processaRemocoes([X-Y-Xf-Yf|T], Tab, NovoTab,Jog):-  
    Y1 is Y+1,  
    Y2 is Yf-1,  
    retiraPecas(X,Y1,X,Y2, Tab, NovoTab2, Jog),  
    processaRemocoes(T, NovoTab2, NovoTab, Jog).
```

% CONQUISTA NA HORIZONTAL

```
processaRemocoes([X-Y-Xf-Y|T], Tab, NovoTab,Jog):-  
    X1 is X+1,  
    X2 is Xf-1,  
    retiraPecas(X1,Y,X2,Y, Tab, NovoTab2, Jog),  
    processaRemocoes(T, NovoTab2, NovoTab, Jog).
```

% RETIRA AS PECAS DO INTERVALO PASSADO COMO ARGUMENTO

```
retiraPecas(X,Y,X,Y, Tab, TRes, Jog):-  
    troca(Jog, Jog2),  
    muda_tab(Jog2, 0, X, Y,Tab, TRes).  
retiraPecas(X,Y1,X,Y2, Tab, NovoTab2, Jog):-  
    troca(Jog, Jog2),  
    muda_tab(Jog2, 0, X, Y1,Tab, NovoTab3),  
    Y3 is Y1+1,  
    retiraPecas(X,Y3,X,Y2,NovoTab3, NovoTab2, Jog).  
retiraPecas(X1,Y,X2,Y, Tab, NovoTab2, Jog):-  
    troca(Jog, Jog2),  
    muda_tab(Jog2, 0, X1, Y,Tab, NovoTab3),  
    X3 is X1+1,  
    retiraPecas(X3,Y,X2,Y,NovoTab3, NovoTab2, Jog).
```

% RETORNA UMA CONQUISTA

```
conquistas(_, X,Y,Xf,Yf, _, N, N):-  
    N > 1,  
    adjacente3(X,Y,Xf,Yf).
```

```
conquistas(Jog, X,Y,Xf,Yf, Tab, N, Cont):-  
    verificaPeca(Tab, X,Y, Jog),  
    verificaPeca(Tab, Xf,Yf, Jog),  
    troca(Jog, Jog2),  
    verificaPeca(Tab, Xa, Ya, Jog2),  
    diagonal(X,Y,Xf,Yf),  
    adjacente2(Xa,Ya,X,Y),  
    muda_tab(Jog2,Jog,Xa,Ya,Tab,NovoTab),  
    N1 is N+1,  
    conquistas(Jog, Xa, Ya, Xf, Yf, NovoTab, N1,Cont).
```

% ALTERA A POSICAO DA PECA DO JOGADOR

```
muda_tab(Peca,Pnov,X,Y,Tab,NovoTab):-  
    muda_tab2(1,Peca,Pnov,X,Y,Tab,NovoTab).
```

```
muda_tab2(_,_,_,_,[],[]).
```

```
muda_tab2(Y,Peca,Pnov,X,Y,[Lin | Resto],[NovLin | Resto2]):-  
    muda_linha(1,Peca,Pnov,X,Lin,NovLin),  
    N2 is Y+1,  
    muda_tab2(N2,Peca,Pnov,X,Y,Resto,Resto2).
```

```
muda_tab2(N,Peca,Pnov,X,Y,[Lin | Resto],[Lin | Resto2]):-  
    N\=Y, N2 is N+1,  
    muda_tab2(N2,Peca,Pnov,X,Y,Resto,Resto2).
```

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

```
muda_linha(X,Peca,Pnov,X,[Peca | Resto],[Pnov | Resto2]):-  
    N2 is X+1,  
    muda_linha(N2,Peca,Pnov,X,Resto,Resto2).
```

```
muda_linha(N,Peca,Pnov,X,[El | Resto],[El | Resto2]):-  
    N\=X, N2 is N+1,  
    muda_linha(N2,Peca,Pnov,X,Resto,Resto2).
```

%VERIFICA SE O JOGO TERMINOU

```
terminouJogo(T,Jogador) :- terminouJogoaux(T,1,1,0,Jogador).
terminouJogoaux(T,9,9,NPecas,Jogador):-!, NPecas < 3,desenha(T),nl,nl,nl,
    write('Terminou o jogo. O jogador '),
    troca(Jogador, Jogador2), write(Jogador2), write(' venceu.'),nl,nl,nl.

terminouJogoaux(T,X,Y,NPecas,Jogador) :-
    if(verificaPeca(T, X, Y, Jogador), NPecasNovo is NPecas+1, NPecasNovo is
NPecas),
    if(X == 9, (X1 is 1, Y1 is Y+1), (X1 is X+1, Y1 is Y)),
    terminouJogoaux(T,X1,Y1,NPecasNovo, Jogador).
```