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, o relatório contém uma visão geral do funcionamento do jogo, assim como as várias formas encontradas para resolver os problemas e desafios que foram surgindo ao longo do desenvolvimento do jogo.

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 Faculdade de Engenharia da Universidade do Porto. This paper has a global detail about the rules and methods of the game and also contains details about the ways we used to solve the various problems and challenges that came along during the development phase.

Indíce

Ilustrações

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 assim como para os testes.

1.2 Motivação

A motivação para a realização deste trabalho provêm da importância para adquirir novos conhecimentos em estilos de programação diferentes do que estamos habituados a trabalhar. Este ponto revelou ser extremamente aliciante visto que é um tipo de programação bastante diferente do tipo de programação a que estamos habituados e revelou-se desafiante o desenvolvimento deste jogo.

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.

Para além disso, o interesse foi ainda maior no que diz respeito à inteligência artificial, dado que se trata de um ramo bastante aliciante e cujo resultado é gratificante.

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 foram:

- Realizar de uma versão do jogo “Hasami Shogi” em linguagem prolog;
- Definir condições de terminação do jogo correctamente;
- 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.

O objectivo passa agora por 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.

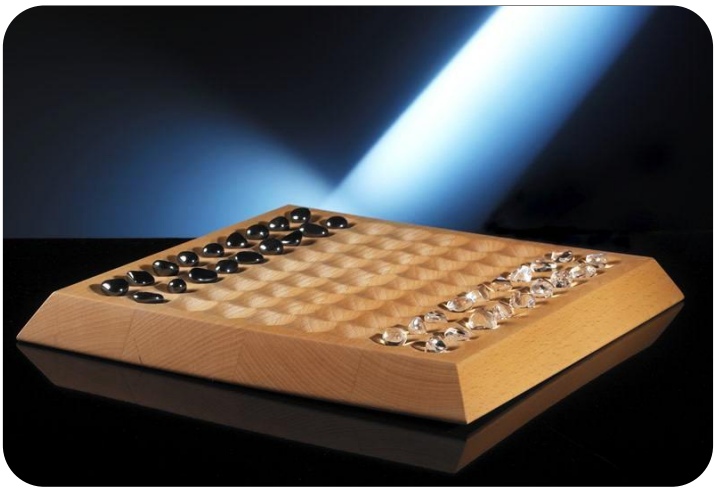
Já o quinto capítulo refere-se à interface de texto com o utilizador, a sua descrição e implementação.

Por último, 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.

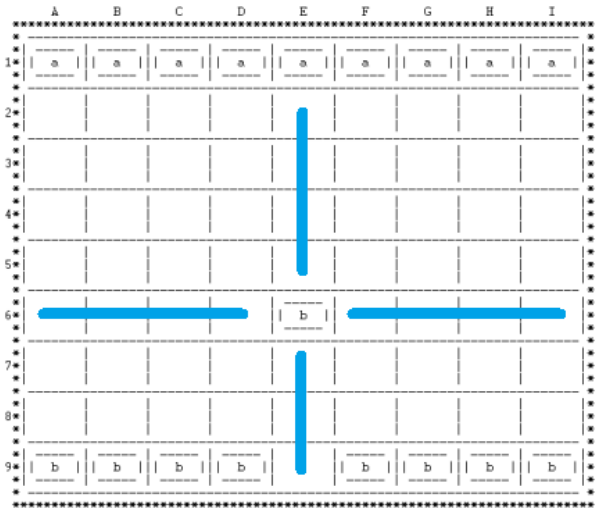


Ilustração 1 - Movimento no tabuleiro

2.2.1 Capturando Inimigos

O processo de captura passa por rodear a peça inimiga com duas nossas na horizontal ou na vertical (a diagonal não é considerada).

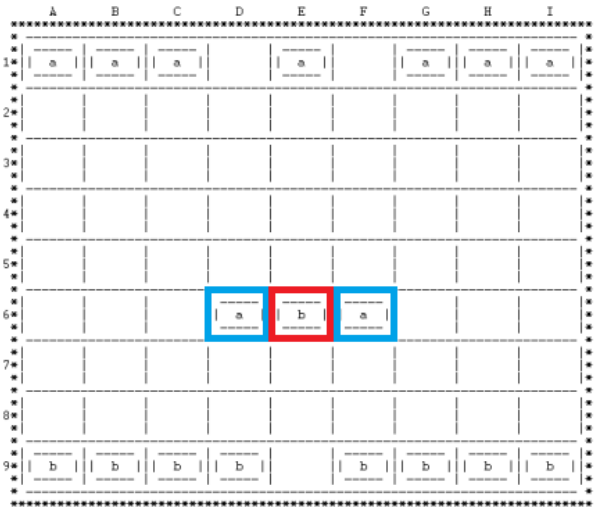


Ilustração 2 - Capturar uma peça na horizontal

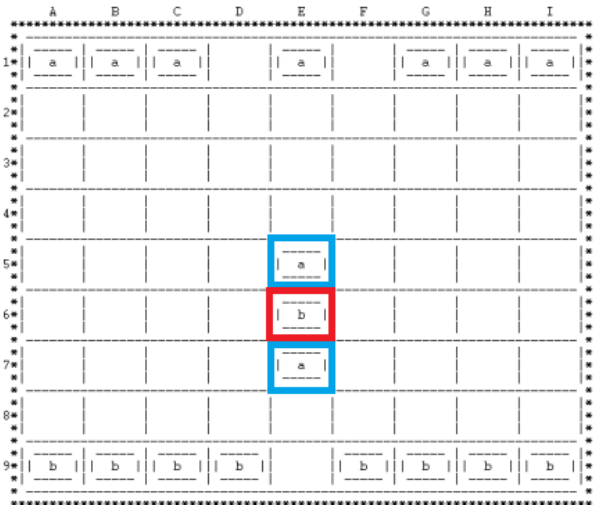


Ilustração 3 - Capturar uma peça na vertical

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

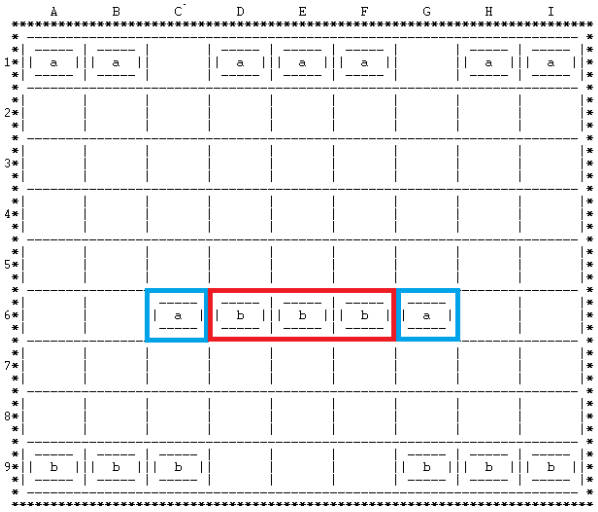


Ilustração 4 - Capturar várias peças em simultâneo

2.2.2 Modos de Jogo

É possível escolher três modos de jogo:

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

Nos modos de jogo que incluem o computador é também necessário escolher o nível de dificuldade pretendido.

Assim, existem três modos de dificuldades diferentes programadas com os seguintes algoritmos:

- [Dificuldade 1] Random
  - Este algoritmo consiste em escolher uma jogada aleatoriamente, de entre todas as jogadas possíveis.
- [Dificuldade 2] Greedy
  - Neste modo o computador procura escolher uma jogada onde irá ficar em posição de proceder a uma conquista de peça do adversário, tendo no entanto o cuidado de não se colocar em posição de ser conquistado. Caso não haja nenhuma possibilidade nestas condições, escolhe uma jogada aleatória de entre as possíveis.

- [Dificuldade 3] Minimax
    - Nesta dificuldade é utilizado o algoritmo *minimax* de forma a minimizar a perda máxima possível, desta forma o computador pode calcular as várias jogadas possíveis (suas e do adversário, tentando prever qual a jogada que o adversário escolherá) com vários níveis de profundidades de forma a obter um melhor resultado.

3. Arquitectura do sistema

O jogo inicia-se com o seguinte predicado: **iniciaJogo**. e é executado 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.
- 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, verificando se existe alguma conquista de peça, e é então impresso o novo tabuleiro no ecrã, caso contrário é pedido novamente ao utilizador as coordenadas do posição de destino.
- O jogo prossegue até que um dos jogadores fique apenas com 2 peças, situação em que este termina.

Em relação à comunicação por sockets, a ideia passa por:

- Receber a inicialização da interface: initialize;
- Devolver ok.
- Receber informacao do modo de jogo: modoJogo(modo);
- Devolver o Tabuleiro e o Jogador inicial: ok(Tabuleiro,Jogador);
- Receber um movimento: execute(Jogador,Mov,Tabuleiro);
- Verificar que tipo de jogo é, processar e responder com: ok(NovoTabuleiro, Jogador2), caso o movimento seja válido (se não for, enviar reject);
- Recebe verificação do fim do jogo: game_end(Tabuleiro).
- Responde de volta: ok(Vencedor), ou reject, caso não tenha terminado o jogo.
- No final, quando o jogo termina, recebe bye e responde com ok.

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:

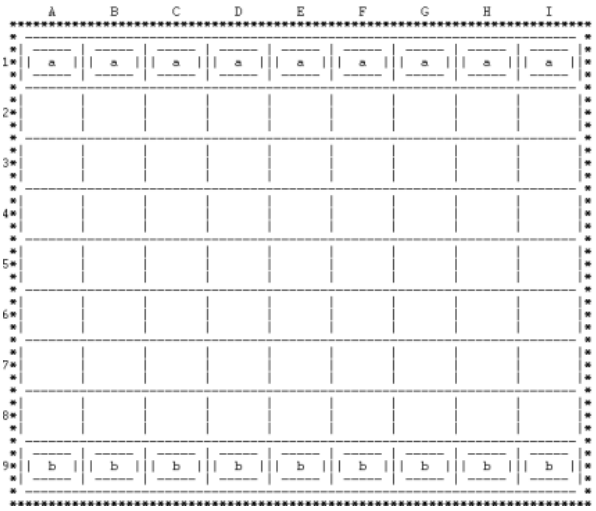


Ilustração 5 - Tabuleiro inicial

Matriz inicial do tabuleiro:

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

Para construir o tabuleiro recorremos 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:

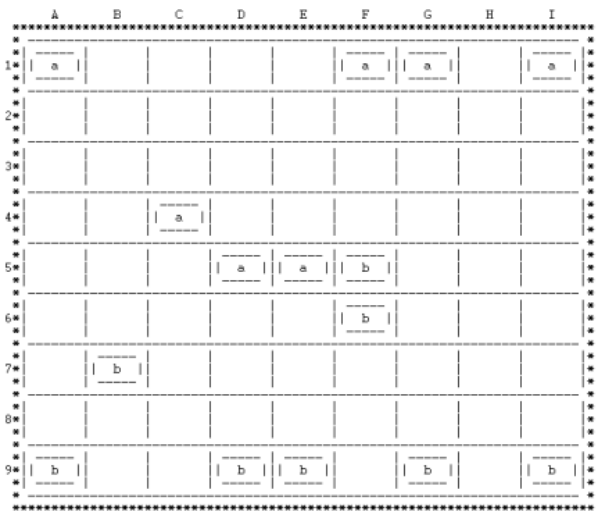


Ilustração 6 - Tabuleiro numa fase intermédia

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

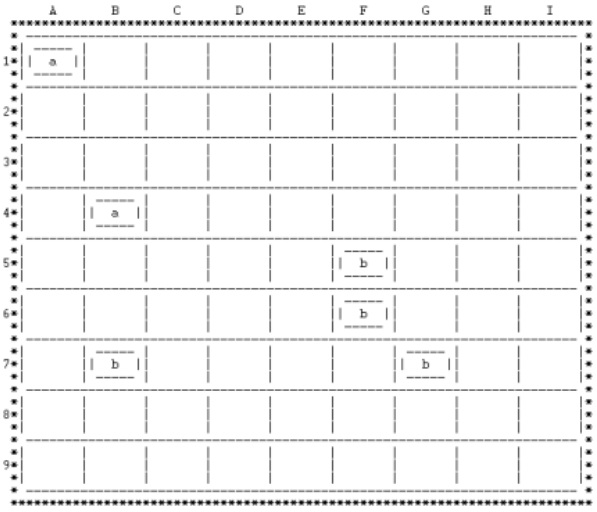


Ilustração 7 - Tabuleiro numa situação final

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 contem 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 desloca-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 valida, 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, recorremos 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:

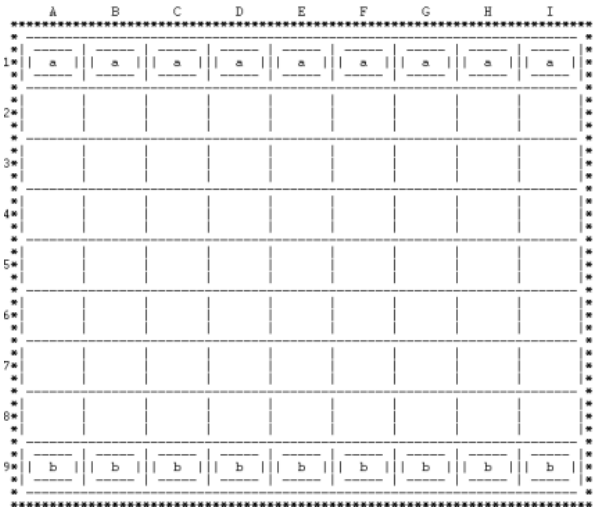


Ilustração 8 - Tabuleiro em linha de código

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 possivéis 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.

Está 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 e executado de forma que a sua interface gráfica fosse 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

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

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

O trabalho foi um processo muito importante e sobretudo aliciante devido à linguagem em que foi desenvolvido. É uma linguagem diferente das restantes estudadas até ao momento e tornou-se desafiante a sua compreensão e a compreensão do seu funcionamento para que conseguíssemos implementar os objectivos propostos. Ficou ciente a utilidade da linguagem e a pertinência dos seus paradigmas, sendo que prevemos que seja importante para situações futuras em que sejamos confrontados com desafios semelhantes.

Sentimos no entanto que o periodo de adaptação à linguagem nos atrasou um pouco o desenvolvimento, e que apesar de tudo ter sido implementado, com mais tempo de desenvolvimento conseguiriamos ter ido um pouco mais longe no que toca à inteligência artificial, de modo a tornar o jogo cada vez mais competitivo e aliciante.

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 Thechniques: Capitulo 20, Secção 2 “Searching Game Trees”.

Anexo A – Código Fonte

```
%UTILITARIOS
conv(Let,Valor):- maiuscula(Let), Valor is Let-64.
conv(Let,Valor):- minuscula(Let), Valor is Let-96.
conv(Let,Valor):- numero(Let), Valor is Let-48.

maiuscula(Let):- Let>=65, Let=<74.
minuscula(Let):- Let>=97, Let=<106.
numero(Let):- Let>=49, Let=<58.

letra(a, 1). letra(b, 2). letra(c, 3). letra(d, 4). letra(e, 5). letra(f, 6). letra(g, 7). letra(h, 8).
letra(i, 9).

troca(1,2).
troca(2,1).

if(Condition, TrueClause, FalseClause) :-
          Condition, !, TrueClause;
     !, FalseClause.

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

cls :- put(27), put("["), put("2"), put("J").

%DESENHAR O TABULEIRO
linhaLimite:-printLinha(['
',*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*
,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,' ']).
linhaLetras:-printLinha([' ',' ',' ',' ',' ',' ','a',' ',' ',' ',' ',' ',' ',' ','b',' ',' ',' ',' ',' ',' ',' ','c',' ',' ',' ',' ','
',' ',' ',' ','d',' ',' ',' ',' ',' ',' ',' ','e',' ',' ',' ',' ',' ',' ',' ','f',' ',' ',' ',' ',' ',' ',' ','g',' ',' ',' ',' ',' ',' ',' ','h','
',' ',' ',' ',' ',' ',' ','i',' ',' ',' ',' ',' ']).
linhaNumerosV(['1','2','3','4','5','6','7','8','9']).
linhaDivH:-printLinha(['  ',*,'  ',-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-
,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,' ',*]).

tabuleiro(
          [[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],
            [2,2,2,2,2,2,2,2,2]]).
/*tabuleiro(
           [[1,1,0,0,0,0,0,0,0],
            [0,2,0,0,0,0,0,0,0],
            [2,0,2,0,2,0,0,0,0],
            [0,1,0,0,2,0,0,0,0],
            [2,0,2,2,2,0,0,0,0],
            [2,0,2,2,0,0,0,0,0],
            [2,0,0,0,2,0,0,0,0],
            [0,2,2,2,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0]]).*/
/*tabuleiro(
           [[0,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0],
            [0,0,1,0,0,0,0,0,0],
            [0,0,0,2,1,0,0,0,0],
            [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]]).*/

piece1(1):- write(' ----- |').
piece1(2):- write(' ----- |').
piece1(0):- write('       |').

piece2(1):- write('|  1  ||').
piece2(2):- write('|  2  ||').
piece2(0):- write((('       |')).

piece3(1):- write(' ----- |').
piece3(2):- write(' ----- |').
piece3(0):- write('       |').

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

%ESCOLHA DOS MODOS DE JOGO
iniciaJogo:-cls, menu.
menu:-
   nl,nl,write('        *** MENU *** '), nl,nl,
   write('1 - Jogador vs CPU'),nl,
   write('2 - Jogador vs Jogador'),nl,
   write('3 - CPU vs CPU'),nl,
   write('4 - Como Jogar?'),nl,
   write('5 - Sair'),nl,
   write('Escolha uma opcao (ex: 1) : '), get_code(Op),skip_line,Op>=49,conv(Op,Esc),
   verifica(Esc).

verifica(1):-jVsCpu, !.
verifica(2):-jVsJ, !.
verifica(3):-cpuVsCpu,!.
verifica(4):-descricaoJogo,menu,!.
verifica(5):-write('Até logo...'),nl,!.
verifica(_):-menu.


modoJ1(T,Jogador,Modo):-
        (Modo == -1;Modo == 1;Modo==2;Modo==3),
        modoJogador(T,Jogador,Modo), !.
modoJ1(T,Jogador,Modo):-
        (Modo == 4;Modo == 5;Modo==6),
        modoCPU(T,Jogador,Modo), !.
modoJ2(T,Jogador,Modo):-
        (Modo == 1;Modo==2;Modo==3;Modo==4,Modo==5;Modo==6),
        modoCPU(T,Jogador,Modo),!.
modoJ2(T,Jogador,Modo):-modoJogador(T,Jogador,Modo).

%MODO JOGADOR VS JOGADOR

jVsJ:-tabuleiro(T), modoJ1(T, 1, -1).

%MODO JOGADOR VS CPU
```

```
jVsCpu:-
        write('1 - Fácil'),nl,
        write('2 - Intermédio'), nl,
        write('3 - Difícil'),nl,
        write('4 - para sair'),nl,
        repeat,write('Opcao (Ex: 1) : '), get_code(Op),skip_line,Op>=49,
Op=<53,conv(Op,Esc),
        dificuldade(Esc).
dificuldade(1):-tabuleiro(T), random(1,3,J), if(J == 1, modoJ1(T, J, 1), modoJ2(T,J,1)), !.
dificuldade(2):-tabuleiro(T), random(1,3,J), if(J == 1, modoJ1(T, J, 2), modoJ2(T,J,1)), !.
dificuldade(3):-tabuleiro(T), random(1,3,J), if(J == 1, modoJ1(T, J, 3), modoJ2(T,J,1)), !.
dificuldade(_):-write('Até logo...'),nl.

%MODO CPU VS CPU
cpuVsCpu:-
        write('1 - Fácil'),nl,
        write('2 - Intermédio'), nl,
        write('3 - Difícil'),nl,
        write('4 - para sair'),nl,
        repeat,write('Opcao (Ex: 1) : '), get_code(Op),skip_line,Op>=49,
Op=<53,conv(Op,Esc),
        dificuldade2(Esc).

dificuldade2(1):-tabuleiro(T), modoJ1(T, 1, 4), !.
dificuldade2(2):-tabuleiro(T), modoJ1(T, 1, 5), !.
dificuldade2(3):-tabuleiro(T), modoJ1(T, 1, 6), !.
dificuldade2(_):-write('Até logo...'),nl.

%FIM DA ESCOLHA DOS MODOS DE JOGO

%AVALIACAO DA JOGADA

interaccaoJogador(T,Y,X,Yf,Xf,Jogador):-
        desenha(T), nl,
        write('Jogador actual: '),write(Jogador),nl,
        write('Peca a mover:'),nl,
        write('Linha (Ex: 1) : '),read(Y),
        write('Coluna (Ex: a) : '),read(Xt),letra(Xt, X),
        write('Posicao desejada:'),nl,
        write('Linha Final (Ex: 2) : '),read(Yf),
        write('Coluna Final (Ex: b) : '),read(Xt2),letra(Xt2, Xf).

modoJogador(T, Jogador, Modo):-
        if((interaccaoJogador(T,Y,X,Yf,Xf,Jogador),
           verificaCaminho(Jogador,X,Y,Xf,Yf,T),
```

```
            modificaT(Jogador,X-Y-Xf-Yf,T,TNovo2)),
            (troca(Jogador, Jogador2),
            conquistaPecas(TNovo2, TNovo3, Jogador),
            conquistaPecas(TNovo3, TNovo4, Jogador2),
            (((terminouJogo(TNovo4,Jogador);terminouJogo(TNovo4,Jogador2)),menu)   ;
modoJ2(TNovo4,Jogador2,Modo))),
          modoJogador(T,Jogador,Modo)).

% CICLO DO BOT DEPENDENDO DA DIFICULDADE ESCOLHIDA
modoCPU(T, Jogador, Modo):-
        (Modo == 1; Modo == 4),!,
        desenha(T), nl,
        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),
        (((terminouJogo(TNovo4,Jogador);terminouJogo(TNovo4,Jogador2)),menu);
modoJ1(TNovo4, Jogador2, Modo)).

% BOT DO MODO INTERMEDIO
modoCPU(T, Jogador, Modo):-
        (Modo == 2; Modo == 5),!,
        desenha(T), nl,
        write('Estou a Pensar! Aguarde um momento por favor...  '),nl,sleep(1),
        greedy(T,L,1,X, Jogador),
        ((L == X,
        findall(X-Y-Xf-Yf, verificaCaminho(Jogador, X,Y,Xf,Yf, T),L1),
        choose(L1,M1),escolheNaoSuicida(T,Jogador,L1,M1,M));choose(L,M)),
        modificaT(Jogador,M,T, TNovo2),
        conquistaPecas(TNovo2, TNovo3, Jogador),
        conquistaPecas(TNovo3, TNovo4, Jogador2),
        troca(Jogador, Jogador2),
        (((terminouJogo(TNovo4,Jogador);terminouJogo(TNovo4,Jogador2)),menu);
modoJ1(TNovo4, Jogador2, Modo)).
% BOT DO MODO DIFICIL
modoCPU(T, Jogador, Modo):-
        (Modo == 3; Modo == 6),!,
        desenha(T), nl,
        write('Estou a Pensar! Aguarde um momento por favor...  '),nl,
        countPieces(Jogador,T,NPecas),
        findall(X-Y-Xf-Yf, verificaCaminho(Jogador, X,Y,Xf,Yf, T),L),
        shuffle(L,[Head|L2]),
        if(NPecas < 5,
        evaluate_and_choose(Jogador,L2,T,2,1,(Head,-1000),(M,_)),
```

```
        evaluate_and_choose(Jogador,L2,T,0,1,(Head,-1000),(M,_))),
        modificaT(Jogador,M,T, TNovo2),
        conquistaPecas(TNovo2, TNovo3, Jogador),
        conquistaPecas(TNovo3, TNovo4, Jogador2),
        troca(Jogador, Jogador2),
        (((terminouJogo(TNovo4,Jogador);terminouJogo(TNovo4,Jogador2)),menu);
modoJ1(TNovo4, Jogador2, Modo)).

% UTILITARIOS DO GREEDY

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,_).

escolheNaoSuicida(Tab,J,_,X-Y-Xf-Yf,X-Y-Xf-Yf):-
        troca(J,J2),
        modificaT(J,X-Y-Xf-Yf,Tab,TNovo2),
        conquistas(J2,_,_,_,_,TNovo2,1,N),
        N=<1,!.
escolheNaoSuicida(Tab,J,L,M2,M):-escolheNaoSuicida2(Tab,J,L,M2,M).
escolheNaoSuicida2(Tab,J,[X-Y-Xf-Yf|Tail],_,M):-
        troca(J,J2),
        modificaT(J,X-Y-Xf-Yf,Tab,TNovo2),
        conquistas(J2,_,_,_,_,TNovo2,1,N),
        N>1,
        escolheNaoSuicida2(Tab,J,Tail,X-Y-Xf-Yf,M),!.
escolheNaoSuicida2(_,_,_,M,M).

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

% UTILITARIOS DO MINIMAX

shuffle([], []).
shuffle(List, [Element|Rest]) :-
    choose(List, Element),
    delete(List, Element, NewList),
    shuffle(NewList, Rest).

evaluate_and_choose(J,[Move|Moves] ,Position,D ,MaxMin,Record,Best):-
        modificaT(J,Move,Position, Position1),
```

```
        minimax(J,D,Position1,MaxMin,_,Value),
        update(J,Move,Value,Record,Recordl),
        evaluate_and_choose(J,Moves,Position,D,MaxMin,Recordl,Best).


evaluate_and_choose(_,[],_,_ ,_ ,Record ,Record).

minimax(J, 0,Position,MaxMin,_,Value):-
        troca(J,J2),
        value(J, Position, V1),
        value(J2, Position, V2),
        conquistaPecas(Position, Position2, J),
        conquistaPecas(Position2, Position3, J2),
        value(J, Position3,V3),
        value(J2, Position3, V4),
        if(V2 > V4, V5 is 1000, V5 is 0),
        if(V1 < V3, V6 is -1000, V6 is 0),
        V is V5+V6+V4*100+V3*(-100),
        Value is V*MaxMin.

minimax(J,D,Position,MaxMin,Move,Value):-
        D > 0,
        findall(X-Y-Xf-Yf, verificaCaminho(J, X,Y,Xf,Yf, Position),Moves),
        D1 is D-1,
        MinMax is -MaxMin,
        evaluate_and_choose(J,Moves,Position,D1,MinMax,(nil,-1000),(Move,Value)).

update(_,_,Value,(Move1,Value1),(Move1,Value1)):-
        Value =< Value1.

update(_,Move,Value,(_,Value1),(Move,Value)):-
        Value > Value1.


value(Jog, Pos, Val):-
        countPieces(Jog, Pos, Val).

countPieces(Jog, Pos, Val):-
        countPiecesAux(Pos, 1,1,0,Jog,Val).

countPiecesAux(_,1,10,NPecas,_,NPecas):-!.

countPiecesAux(T,X,Y,NPecas,Jogador,Val) :-
        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)),
```

```
        countPiecesAux(T,X1,Y1,NPecasNovo, Jogador,Val).

% FIM DOS BOTS

% VERIFICA SE A PECA E' DO UTLIZADOR

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-X-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,1,10,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).

%DESCRICAO DO JOGO
descricaoJogo:-
        write('Como jogar:'),nl,
```

write('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. O processo de captura passa por rodear a peça inimiga com duas nossas na horizontal ou vertical (a diagonal não é considerada). Perde aquele que ficar primeiro com apenas duas peças no tabuleiro.').