|  |
| --- |
| faculdade de engenharia da universidade do porto |
|  |
| Hasami Shogi |
| Relatório Intercalar do 1º Projecto – Programação em Lógica (3º Ano – 1º Semestre) |
|  |
|  |
| **Autores:** |
| **Hélder Alexandre dos Santos Moreira** |
| **Carlos Tiago Rocha Babo** |
| **Felipe de Souza Schmitt** |
| **ID do Grupo: 106** |
| **10/3/2010** |

# 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. Assim, este relatório contém uma visão geral do funcionamento do jogo, assim como as nossas primeiras ideias relativamente à forma como iremos abordar o problema.

Indíce

[Resumo 1](#_Toc273726448)

[1. Introdução 3](#_Toc273726449)

[2. Descrição do Problema 4](#_Toc273726450)

[2.1 Movimento das Peças 4](#_Toc273726451)

[2.2 Capturando Inimigos 4](#_Toc273726452)

[2.3 Níveis de Dificuldade 5](#_Toc273726453)

[2.4 Realizar um Jogo 5](#_Toc273726454)

[3. Representação do Estado de Jogo 6](#_Toc273726455)

[3.1 O tabuleiro 6](#_Toc273726456)

[4. Representação de um Movimento 8](#_Toc273726457)

[5. Visualização do Tabuleiro 9](#_Toc273726458)

[6. Conclusões e Perspectivas de Desenvolvimento 10](#_Toc273726459)

[Bibliografia 11](#_Toc273726460)

[Anexo 12](#_Toc273726461)

[Ilustração 1 - Movimento no tabuleiro 4](#_Toc273726462)

[Ilustração 2 - Capturar uma peça na horizontal 4](file:///C:\Users\Tiago\Desktop\relatório_plog.docx#_Toc273726463)

[Ilustração 3 - Capturar uma peça na vertical 4](#_Toc273726464)

[Ilustração 4 - Capturar várias peças em simultâneo 5](#_Toc273726465)

[Ilustração 5 - Capturar uma peça num canto 5](#_Toc273726466)

[Ilustração 6 - Tabuleiro inicial 6](#_Toc273726467)

[Ilustração 7 - Tabuleiro numa fase intermédia 7](#_Toc273726468)

[Ilustração 8 - Tabuleiro numa situação final 7](#_Toc273726469)

[Ilustração 9 - Tabuleiro em linha de código 9](#_Toc273726470)

# Introdução

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 torna-se bastante aliciante devido à simplicidade aliada às múltiplas estratégias possíveis, mas que requerem grande concentração e previsão do desenrolar das jogadas.

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. 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*.

O relatório contém uma descrição do problema, a representação do estado do jogo e de um movimento, a visualização do tabuleiro e as conclusões e perspectivas de desenvolvimento. Em anexo pode ser encontrado o código-fonte já desenvolvido.

# Descrição do Problema

## 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 - Movimento no tabuleiro

## 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 - Capturar uma peça na horizontal

Ilustração - Capturar uma peça na vertical

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



Ilustração - Capturar várias peças em simultâneo

O mesmo acontece se uma peça for encurralada num canto.

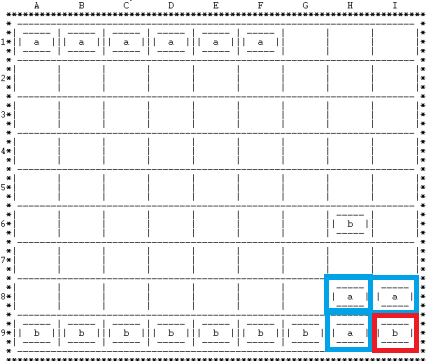


Ilustração - Capturar uma peça num canto

## Níveis de Dificuldade

O projecto terá pelo menos dois níveis de dificuldade:

* Fácil: o computador efectua as jogadas, calculando cerca de 4 iterações de jogo, verificando qual a melhor opção de jogada;
* Difícil: semelhante à dificuldade fácil, mas com maior número de iterações.

Se houver oportunidade, será ainda adicionada a dificuldade “Imbatível” em que o computador percorre toda a árvore de escolhas e selecciona a melhor.

## Realizar um Jogo

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

# Representação do Estado de Jogo

## O tabuleiro

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 - 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 - Tabuleiro numa fase intermédia

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



Ilustração - 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.

# Representação de um Movimento

# 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(T),

linhaLetrasV(X),

linhaLetras, nl,

linhaLimite, nl,

linhaDivH, nl,

printPecas(T,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 - 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.

# Conclusões e Perspectivas de Desenvolvimento

# 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

Código fonte já desenvolvido:

%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]]).

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

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

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

piece2(1):- write('| a ||').

piece2(2):- write('| b ||').

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(T),

linhaNumerosV(X),

linhaLetras, nl,

linhaLimite, nl,

linhaDivH, nl,

printPecas(T,X),

linhaLimite.

%FIM DO DESENHO DO TABULEIRO