Universidade do Porto
Faculdade de Engenharia
FEUP

Pesquisa com Adversários: Jogo de Tabuleiro – Hex

Relatório Final

Inteligência Artificial
3º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo:
Carlos Frias – 040509116 – ei04116@fe.up.pt

Porto, 30 de Abril de 2011

1. Objetivo:

O presente relatório pretende documentar o desenvolvimento, do projeto da cadeira de Inteligência Artificial do curso Mestrado Integrado em Engenharia Informática e Computação na Faculdade de Engenharia da Universidade do Porto.

O objetivo do trabalho é desenvolver uma aplicação que simule o jogo de tabuleiro Hex, jogado com dois jogadores. A aplicação permite jogar humano contra humano, humano contra computador e computador contra computador, com três níveis de dificuldade: fácil, médio e difícil.

Na implementação do trabalho foram usadas as seguintes técnicas de Inteligência Artificial: o algoritmo minimax com cortes alfa-beta e variações deste e ainda o algoritmo subir-a-colina "ascensão íngreme", com adaptações para a pesquisa com adversários.

2. Descrição:

2.1. Funcionalidades:

O jogo a desenvolver é o jogo de tabuleiro Hex que é jogado num tabuleiro tipicamente 11x11, podendo haver as variações 13x13 e 19x19, com uma grelha hexagonal.

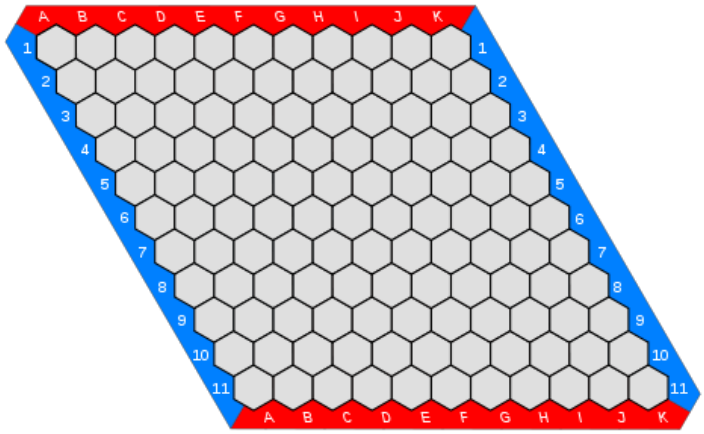

Figura 1: Tabuleiro de jogo 11x11

A cada jogador é atribuída uma cor, azul ou vermelha. O jogadores colocam alternadamente uma peça da sua cor numa das posições ainda não ocupadas do tabuleiro. O objetivo é formar um caminho de peças da sua cor que ligue dois lados opostos do tabuleiro. O primeiro jogador a completar o caminho ganha o jogo. As quatro casas dos cantos do tabuleiro pertencem a ambos os lados adjacentes. Como o primeiro jogador tem uma ligeira vantagem, o segundo jogador pode escolher se quer ou não trocar de posições com o primeiro jogador após este ter feito a primeira jogada.

A aplicação permite as seguintes variedades de jogo: jogador humano vs jogador humano,

jogador humano vs jogador computador e jogador computador vs jogador computador, com três níveis de dificuldade: fácil, médio e difícil.

Para o nível de dificuldade fácil, o jogador computador irá calcular a sua jogada com o algoritmo subir-a-colina "subida íngreme", isto é, a cada passo o jogador efetua a jogada que apresenta melhor valor, neste caso o menor possível, da função avaliação. De salientar que essa jogada poderá não ser única, isto é, podem existir várias jogadas com o mesmo valor ótimo "mínimo" a um dado momento. Mais adiante explicarei, pormenorizadamente, a função de avaliação.

No nível de dificuldade médio, o jogador computador irá realizar 90% das suas jogadas baseando-se no algoritmo de pesquisa com adversários minimax com cortes alfa-beta, com a função de avaliação, sendo as restantes 10% das jogadas realizadas ao acaso, simulando erros ocasionais.

Para o nível de dificuldade difícil é utilizado unicamente o algoritmo de pesquisa minimax com cortes alfa-beta, com a mesma função de avaliação.

2.2. Estrutura do programa:

O programa está repartido por seis classes de objetos: *Comp*, *JanelaHex*, *JanelaMensagem*, *MeuPanel*, *Ponto* e *Tabuleiro*.

Na classe *Comp* encontram-se as funções que permitem ao jogador realizar as jogadas, a saber: *jogadaAleatoria()*; *jogadaGulosa()* e *minimaxAlfaBeta(...)*;

A classe *JanelaHex* é uma subclasse de *JFrame* é responsável pela apresentação da janela principal da aplicação, da apresentação do tabuleiro de jogo e das jogadas.

A classe *JanelaMensagem* é também uma subclasse de *JFrame* e é utilizada para apresentar uma janela com mensagens, nomeadamente, uma janela que mostra as regras do jogo e outra que mostra informação acerca da aplicação desenvolvida.

A classe *MeuPanel* é uma subclasse de *JPanel* e é utiliza para criar o "painel" com a imagem do tabuleiro que é visualizado na janela principal da aplicação.

A classe *Ponto* é utilizada para representar cada uma das casas do tabuleiro de jogo, que são caracterizadas por um *char*, *coluna*, e um *inteiro*, *linha*. Por exemplo, A1 é um ponto do tabuleiro.

Finalmente, a classe *Tabuleiro* é responsável pelo controlo do tabuleiro, que é representado por um *array* bidimensional de *inteiros*, pela realização das jogadas de cada jogador, pelo controlo dos trilhos ou caminhos já realizados a dado momento por cada um dos jogadores, por averiguar se um jogador venceu o jogo e pela função de avaliação do estado do jogo num dado momento em relação a um jogador. Nesta classe e na classe *Comp* estão as principais funções da aplicação desenvolvida.

2.3. Esquemas de representação de conhecimento:

Em termos de representação de conhecimento usei um *array* de inteiros de dimensão dois para representar o tabuleiro em cada um dos seus estados de jogo. Cada elemento do *array*, em cada instante, apenas pode ter os valores 0, 1 ou 2: o valor 0 representa uma casa vazia, o valor 1 representa uma casa preenchida com uma peça do jogador um e o valor 2 representa uma casa preenchida por uma peça do jogador dois.

As funções responsáveis pela alteração dos valores no array, e portanto, pela realização das jogadas são: *boolean setJogada(char x, int y, int jogador)* ou *boolean setJogada(int x, int y, int jogador)* que recebem a jogada (em dois formatos diferentes) e o jogador que a pretende realizar e devolve um booleano, true se a jogada é possível e foi efetuada ou false se a jogada não é possível, por já se encontrar a casa preenchida, e portanto, a jogada não ter sido realizada.

Como a finalidade do jogo é fazer um caminho de peças da sua cor ligando lados opostos do tabuleiro, surgiu a necessidade de guardar esses caminhos numa estrutura de dados. Optei por vectores de vectores de objectos da classe Ponto, designado cada um desses vectores por trilhos, ou seja, guardo para cada jogador um vector de trilhos de peças de sua cor.

Entenda-se por trilho, um conjunto de pontos "casas do tabuleiro" com a mesma cor e na qual é possível ligar duas quaisquer "casas" desse trilho por casas vizinhas "adjacentes" com a mesma cor.

Optei por uma estrutura deste género pela facilidade de manuseamento, nomeadamente, adicionar novos pontos, unir vectores, simulando a junção de dois trilhos, remoção de elementos de um vector, entre outros.

Para a gestão dos trilhos de cada jogador uso o seguinte procedimento da classe Tabuleiro:

- *private void acrescenta_trilhos(int x, int y, int jogador);*

  A ideia deste procedimento é a seguinte:

  Sempre que se realize uma nova jogada, digamos para a posição *xy*. Este procedimento percorre os passos:

  - Calcula os vizinhos de *xy*, isto é as posições adjacentes a *xy* no tabuleiro;
  - Percorre cada um dos trilhos existentes para esse jogador e:
    - se esse trilho contém um vizinho de *xy* e não contém *xy*, então *xy* é acrescentado ao trilho;
  - Se *xy* não for acrescentado a nenhum trilho então cria um novo trilho apenas com esse elemento;
  - Se *xy* for acrescentado a mais que um trilho, então à que juntar esses trilhos num só

trilho junção. Para isso:

- remover *xy* de cada trilho a que foi adicionado;
- concatenar esses trilhos num só trilho junção;
- remover os trilhos iniciais que originaram a junção;
- acrescentar ao trilho junção o elemento *xy*;
- adicionar esse trilho ao conjunto de trilhos existentes desse jogador.

Um aspecto importante que deve ser explicado também é o de saber quando é que há um vencedor do jogo. Para tal é utilizada a seguinte função da classe Tabuleiro:

- *boolean venceu(int jogador)*;

  Para tornar eficiente a verificação de vencedor a função faz uma pesquisa inicial pelas casas de cada um dos quatro lados fronteira do tabuleiro e guarda num *array* duplo de inteiros 11x4 a informação 0 se nessa casa não está o jogador e 1 caso contrário.

  Depois deste array preenchido o procedimento é o seguinte:

  - Percorrer cada par, primeira/última linha e primeira/última coluna do tabuleiro e:
    - por cada trilho do jogador verificar se:
      - esse trilho contém o par de pontos (primeira/última linha/coluna):
        - se sim então retorna true;
  - Se não encontrou na pesquisa anterior um par no mesmo trilho, retorna false.

Para implementação do algoritmo minimax com cortes alfa-beta houve necessidade de implementar uma função de avaliação para cada estado do jogo em relação a cada um dos jogadores.

Essa avaliação é feita na classe Tabuleiro pelo método:

- *int avaliaEstado(int jogador)*;

  Esta função serve-se do método *distanciaMin* para calcular a dado momento o número mínimo de jogadas que permitem ao jogador completar um trilho (em qualquer uma das direcções: horizontal, vertical e diagonal) para vencer o jogo.

  É calculada a distância mínima para o jogador, para o adversário e é devolvido o seguinte resultado: $distanciaMin(jogador) - \frac{1}{2} distanciaMin(adv(jogador))$

  Desta forma dá-se importância à jogada para o jogador vencer, mas também "metade da importância" à jogada para evitar que o adversário vença (bloquear o adversário).

Assim os resultados possíveis para a avaliação são todos os valores inteiros compreendidos entre -5 e 11, pois:

$-5 = 0 - \frac{11}{2}$ onde 0 significa que o jogador vai vencer nesta jogada e o adversário teria que realizar 11 jogadas para vencer.

$11 = 11 - \frac{0}{2}$ onde 11 significa que o jogador teria que jogar 11 vezes para ganhar, mas o adversário ganha já nesta jogada.

Estas duas são as situações extremas.

Obviamente, nesta medição, quanto menor o valor da função de avaliação melhor.

Explique-se agora a função que calcula a distância mínima.

- *private int distanciaMin(int jogador)*;

  A ideia base desta função é:

  - Percorrer todos os pontos de cada trilho do jogador e:
    - calcular as distâncias do ponto à coluna A e à coluna K, se não existirem pelo caminho peças do adversário;
    - somar as distâncias mínimas à coluna A e à coluna K;
    - Calcular as distâncias do ponto à linha 1 e à linha 11, se não existirem pelo caminho peças do adversário;
    - somar as distâncias mínimas à linha A e à linha 11.
  - retornar o mínimo das duas somas.

O método mais importante na aplicação é seguinte método da classe Comp:

- *int minimaxAlfaBeta(int jogador, int nivel, Tabuleiro tab)*;

  Este método realiza o algoritmo minimax com cortes alfa-beta para pesquisa por parte do jogador computador da melhor jogada a cada momento. É um método recursivo que tem a seguinte ideia de funcionamento:

  - Determina se o nível é: topo ou profundidade limite ou maximizador ou minimizador. Notar que o método é chamado a primeira vez passando no argumento nivel o valor zero, representando o estado inícial (ou actual) do tabuleiro. Os níveis maximizadores representam jogadas do jogador actual e os níveis minimizadores representam jogadas do jogador adversário.
  - Se o nível for topo "zero neste caso", então alfa = Integer.MIN_VALUE e beta = Integer.MAX_VALUE, simulando, respectivamente, $-\infty$ e $+\infty$;

- Se o nível é de profundidade máxima então é computada a função de avaliação referida anteriormente e é retornado esse valor.
- Se o nível é maximizador, então:(resto da divisão de nivel por 2 igual a 1)
  - enquanto (todos os sucessores "no caso os elementos do vector posVazias" não forem examinados com minimaxAlfaBeta E alfa < beta) fazer:
    - alfa é o maior dos valores seguintes:
      - valor herdado do nível anterior e resultado de minimaxAlfaBeta aplicado aos sucessores;
      - Aplicar ao próximo sucessor já os novos valores de alfa e beta;
  - se alfa < beta retorna alfa, senão retorna beta.
- Se nível é minimizador, então:
  - enquanto (todos os sucessores não forem examinados e alfa < beta) fazer:
    - beta é o menor dos valores seguintes:
    - valor herdado do nível anterior e resultado de minimaxAlfaBeta aplicado aos sucessores;
    - Aplique ao próximo sucessor já os novos valores de alfa e beta;
  - Se alfa < beta retorna beta senão retorna alfa.

Para guardar os valores de alfa e de beta utilizo duas variáveis globais inteiras alfa e beta.

Infelizmente, por falta de tempo e também inexperiência, não consegui determinar com a certeza absoluta qual a jogada que deve ser realizada pelo jogador computador ao fim de executar o algoritmo minimax. Armazeno na variável global *jogadaMinMax*, do tipo *Ponto* a jogada que me parece ser a mais indicada.

Para o valor da profundidade máxima testei vários valores e optei por um que realiza a tarefa com um bom grau de garantias, num tempo de execução razoável. Para a profundidade máxima de seis, o tempo de execução do algoritmo é de 1 segundo, pelo que escolhi essa profundidade. Para profundidades superiores de 7, 8 e 9, o tempo de execução seria, respectivamente, de 3 segundos, 13 segundos e 1 minuto e 15 segundos.

2.4. Análise da complexidade dos algoritmos utilizados

O principal algoritmo utilizado foi o minimax com cortes alfa-beta, que tipicamente é procedimento do tipo primeiro em profundidade, ao que acarreta desvantagens em tempo de processamento se o grau de ramificação for elevado, que é o caso, 121 ramificações e decrementa

uma em cada subnível, e também profundidades máximas elevadas. No entanto, é vantajoso em termos de espaço de memória utilizado, uma vez que não necessita de guardar os nós da estrutura em árvore na qual pesquisa.

O facto de se usar cortes alfa-beta à pesquisa significa que parte dos ramos da árvore são ignorados o que reduz em parte o tempo de execução, quanto melhor for a função de avaliação mais ramos serão eliminados e mais eficiente se torna o algoritmo.

2.5. Ambiente de desenvolvimento:

A aplicação foi desenvolvida em ambiente Windows, utilizando o JAVA com linguagem de programação, com o NetBeans como editor.

A máquina na qual se desenvolveu a aplicação e se correram os teste ao programa é um ASUS com processador intel core i5 com 4Gb de memória ram.

2.6. Avaliação do programa:

Ao longo do desenvolvimento da aplicação foram sendo realizados testes de execução para deteção e eliminação de bugs de programação.

Infelizmente, por falta de tempo e má gestão do mesmo, não pude fazer testes exaustivos para apresentar estatísticas claras da execução do programa nos diferentes modos de jogo e de dificuldade.

Apesar disso posso afirmar, com algum grau de certeza que, relativamente ao modo de execução computador vs computador nos diversos graus de dificuldade, o jogador que inicia a partida tem uma probabilidade de vencer o jogo de entre os 65% aos 80%.

3. Conclusões:

Em jeito de conclusão penso que poderia ter sido feito um melhor trabalho, embora pense ter conseguido atingir os requisitos mínimos.

Ficaram por implementar algumas funcionalidades que a principio queria ter feito, a saber:

A função avaliação tem apenas em conta as distâncias mínimas que o jogador e o adversário têm para completar o trilho, faltou ter em consideração a possibilidade de existência de pontes entre trilhos, casos em que uma única jogada podia reduzir em muito a função de avaliação.

A possibilidade de troca entre jogadores no final da primeira jogada ficou também por implementar, uma vez que deixei para último lugar pensado depois ter tempo para tal;

Alguns bugs apresentam-se na janela principal da aplicação, nomeadamente, ao fim de se

executar um jogo, deveria ser possível no menu modos de jogo mandar efectuar novo jogo. Essa seria a ideia, mas o facto é que o tabuleiro é limpo, mas não é possível jogar de novo. Para isso é necessário fechar a janela e reiniciar a aplicação. Outro bug prende-se com as janelas de mensagens que ao fecha-las também fecha a aplicação principal;

Finalmente o algoritmo minimax com cortes alfa beta não está seguramente a funcionar a 100%, como já referi atrás.

Não querendo arranjar desculpas, estes problemas devem-se às seguintes situações: ter realizado o trabalho sozinho, ser trabalhador-estudante e, portanto, ter o tempo reduzido para dedicar a esta disciplina, impossibilidade de poder ir às aulas teóricas e ter "faltado" a muitas das aulas práticas.

Apesar de todo isto, considero que foi benéfico a realização do trabalho e o contacto com alguns dos algoritmos de Inteligência Artificial. Aprendi bastante e penso que me ajudou a tornar melhor programador.

4. Recursos:

- Artificial Intelligence - A Modern Approach 3rd ed - S. Russell, P. Norvig (Prentice-Hall, 2010);
- Inteligência Artificial, apontamentos da cadeira, disponível em: http://paginas.fe.up.pt/~eol/IA/ia1011.html. Último acesso em 10 de Abril de 2011;
- Hex (board game), disponível em: http://en.wikipedia.org/wiki/Hex_%28board_game%29. Último acesso em 10 de Abril de 2011;
- NetBeans IDE 6.9.1, disponível em: http://netbeans.org/;
- Java SE Development Kit 6u24, disponível em: http://www.oracle.com/.

5. Anexos

5.1. Manual de utilizador:

A aplicação é "Hex.jar" um executável java que pode ser executado com um simples duplo clique com o rato, ou pela linha de comandos, com o comando *§ java -jar Hex.jar*

Ao lançar o programa surge seguinte janela:

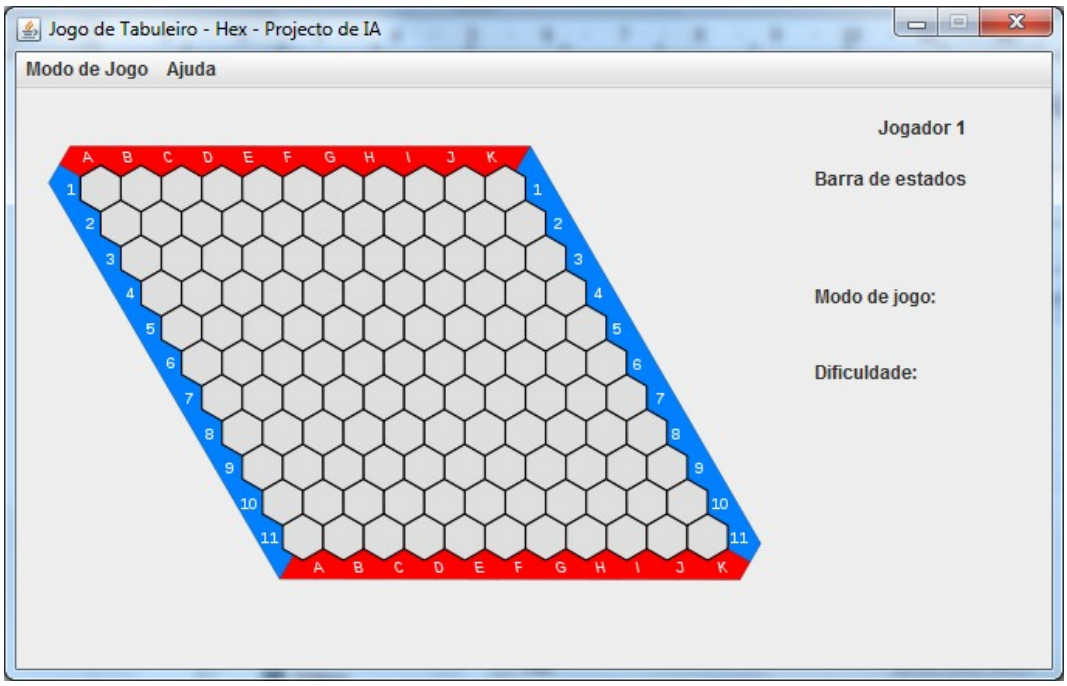

Figura 2: Janela de visualização de jogo.

A janela de visualização apresenta uma barra de menus, onde se pode escolher o modo de jogo e grau de dificuldade.

Figura 3: Menu Modo de Jogo

Ao escolher a dificuldade e o modo de jogo, o jogo inicia de seguida. Notar que para cada uma das opções de jogo existem teclas de inicialização rápida, não havendo necessidade de ir com o rato aos menus.

No menu ajuda são apresentadas as regras do jogo e as informações sobre a aplicação desenvolvida.

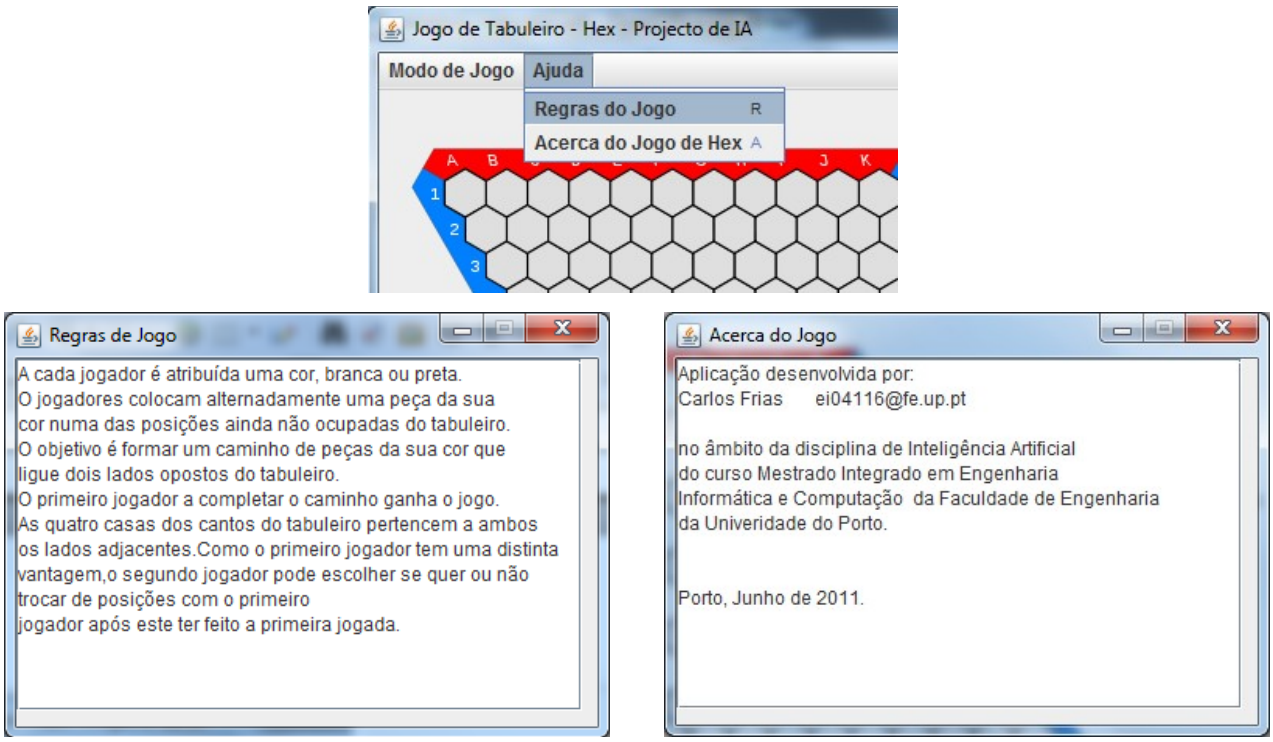

Figura 4: Menu Ajuda e Janelas: Regras de Jogo e Acerca do Jogo.

Na janela principail surgem as indicações do próximo jogador a jogar, da posição sobre onde o rato está no tabuleiro, qual o vencedor se for o caso, qual o modo de jogo e qual o grau de dificuldade.

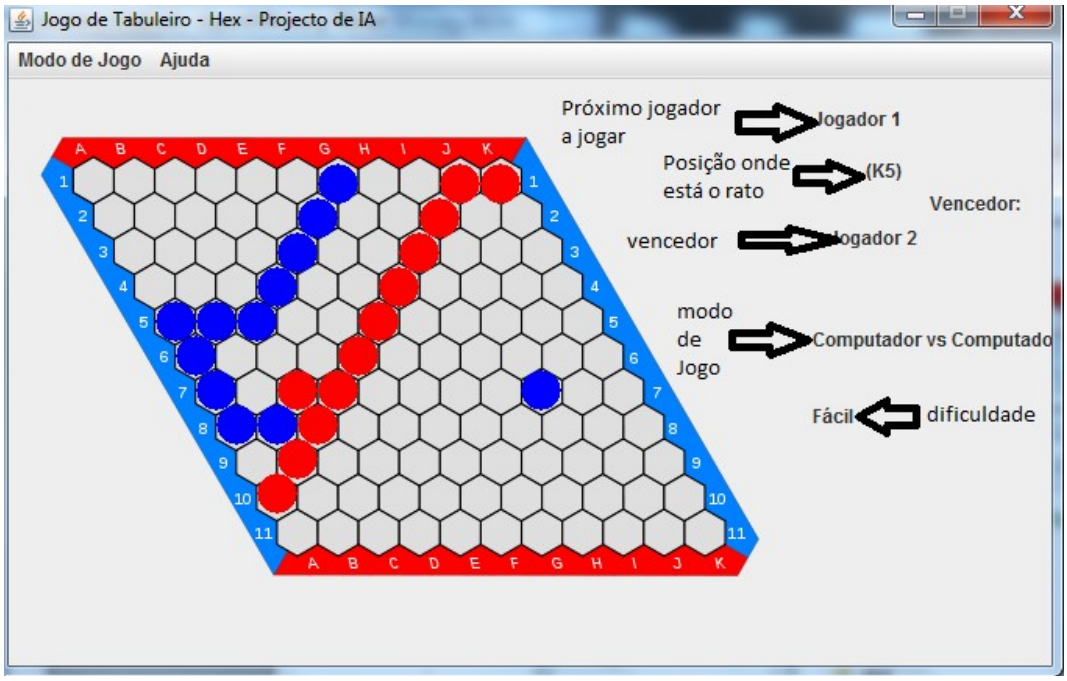

Figura 5: Indicações do jogo na janela principal.

Para realizar a jogada no modo humano basta deslocar o rato para a posição desejada e clicar. Para que seja executada a jogada do jogador computador basta clicar com o rato em qualquer

local dentro da janela principal.

Para fechar a aplicação basta clicar no botão sair da aplicação.