Universidade Federal do ABC
UFABC

J2Velha

Uma Implementação Java do Jogo da Velha Utilizando o Algoritmo MiniMax

André Filipe de Moraes Batista
andre.batista@ufabc.edu.br

Luis Fernando de Oliveira Jacintho
luis.jacintho@ufabc.edu.br

Disciplina de Inteligência Artificial
Profº Jerônimo Pellegrini

Santo André, Junho de 2008

Sumário

Capítulo 1

Jogos em IA

Para a maioria das pessoas o termo jogo é considerado como um passatempo do dia-a-dia. Para as crianças serve como um modo de fugir aos trabalhos de casa e entrar em um mundo virtual de infinitas possibilidades. Para os adultos o termo jogo pode invocar imagens de jogadores que procuram estratégias que lhes dêem vantagens sobre os adversários. Ou seja, o resultado do jogo é determinado pelas estratégias utilizadas pelos jogadores. O ramo da Matemática que pensa de semelhante maneira é denominado **Teoria dos Jogos**.

A Teoria dos Jogos tem por objetivo visualizar qualquer ambiente multiagente como um jogo, desde que o impacto de cada agente sobre os outros seja, de algum modo, significativo. Os jogos são uma das áreas mais antigas desenvolvidas em Inteligência Artificial (IA). Em 1950, desde que os computadores se tornaram programáveis, o primeiro jogo de xadrez foi criado por Claude Shannon e por Alan Turing. Desde então diversos progressos ocorreram nesta área, de tal forma que os sistemas atuais são capazes de rivalizar e ganhar de um dos melhores jogadores de xadrez da história, Garry Kasparov.

Em meados de 1996 ocorreu o primeiro confronto entre Garry Kasparov e o *Deep Blue*. Trata-se de um super computador de alta performance desenvolvido pela IBM, seu código de programação é em linguagem C e é executado no sistema operacional AIX. O resultado é uma máquina escalável capaz de calcular entre 100 e 200 bilhões de jogadas em aproximadamente 3 minutos. No primeiro confronto entre os dois, a vitória foi de Kasparov. Até que em 1997 a IBM desdobrou-se em constantes desenvolvimentos e atualizações de modo a melhorar o desempenho do *Deep Blue*. O resultado de tanto esforço foi que Garry Kasparov foi vencido em 1997 pelo *Deep Blue*. A Figura 1.1 mostra uma cena desta disputa.

1.1 Minimax - Algoritmo de Busca Competitiva

Em um ambiente multiagente os agentes convivem com situações de cooperação e competição. Um ambiente competitivo é aquele em que as metas dos agentes estão em constante

Figura 1.1: Cena de um Disputa de Kasparov *versus* DeepBlue

.

conflito. Para tais situações é preciso desenvolver técnicas de busca competitiva entre os agentes. É neste ponto que a teoria dos jogos pode auxiliar na construção de um agente racional.

Em IA os jogos normalmente são de um tipo bastante especializados - algumas vezes denominados determinísticos de revezamento de dois jogadores de soma zero com informações perfeitas. Isto representa ambiente determinísticos completamente observáveis em que existem dois agentes cujas ações devem se alternar e em que os valores de utilidade no fim do jogo são sempre iguais e opostos. Por exemplo, se um jogador ganha um jogo de xadrez (+1), o outro jogador necessariamente perde (-1). Essa oposição entre as funções de utilidades dos agentes que gera a situação de competição.

O MiniMax é um algoritmo de busca competitiva que seleciona a melhor ação a ser feita em uma situação ou em um jogo, onde dois jogadores se empenham em alcançar objetivos mutuamente exclusivos. Ele se aplica especialmente na busca em árvores de jogo para determinar qual a melhor jogada para o jogador atual. O algoritmo se baseia no princípio de que em cada jogada, o jogador irá escolher o melhor movimento possível.

A árvore de jogo consiste de todas as jogadas possíveis para o jogador atual como nós filhos da raiz, e todas as jogadas disponíveis para o próximo jogador como filhas destes nós e assim por diante, até o nível que se desejar. Cada ramificação da árvore representa um movimento que o jogador pode fazer em tal momento do jogo. Uma busca mais profunda na árvore fornece mais informações sobre as possíveis vantagens ou armadilhas e portanto resulta em uma jogada melhor.

O MiniMax faz uma busca que determina todas as possíveis continuações do jogo até o nível desejado, avaliando e atribuindo um valor a cada movimento possível. A busca

então retorna na árvore de jogo alternando entre escolher o valor mais alto e o valor mais baixo entre os valores da jogadas em um nível. O método de busca consiste na idéia de maximizar a utilidade supondo que o adversário vai tentar minimizá-la. Em termos de busca, é realiza uma busca cega em profundidade, o agente é o MAX e seu adversário é o MIN.

Algoritmo 1 MINIMAX

função DECISÃO-MINIMAX(*estado*) **retorna** uma ação
entradas: *estado*, estado corrente no jogo
$v \leftarrow$ VALOR-MAX(estado)
retornar a ação em SUCESSORES(*estado*) com valor v

função VALOR-MAX(*estado*) **retorna** um valor de utilidade
se TESTE-TERMINAL(*estado*) **então retornar** UTILIDADE(*estado*)
$v \leftarrow -\infty$
para a, s em SUCESSORES(*estado*) **faça**
$v \leftarrow$ MAX(v, VALOR-MIN(s))
retornar v

função VALOR-MIN(*estado*) **retorna** um valor de utilidade
se TESTE-TERMINAL(*estado*) **então retornar** UTILIDADE(*estado*)
$v \leftarrow \infty$
para a, s em SUCESSORES(*estado*) **faça**
$v \leftarrow$ MAX(v, VALOR-MAX(s))
retornar v

Se fosse o caso de se tratar de uma busca normal, bastava percorrer-se a árvore até aos nós terminais e escolher o caminho que levasse ao nó com maior valor de utilidade. Mas não é assim, visto existir outro jogador. Assim, é necessário, escolher a partir de cada nó filho, o menor valor de utilidade, e copia-lo para o nó pai, recursivamente até ao nó inicial. Este é o **algoritmo MiniMax**. Isto deve-se ao fato, do jogador MIN tentar minimizar o ganho do jogador MAX, pois ele tentará escolher uma jogada, dentro das possíveis, que dê menos pontos ao jogador adversário. Na Caixa de Algoritmo 1 tem-se o algoritmo MiniMax.

No Capítulo que segue tem-se uma implementação do Jogo da Velha utilizando a linguagem Java e o algoritmo MiniMax.

Capítulo 2

J*2*Velha: Uma Abordagem Java ao Jogo da Velha

Conhecido também como "Jogo do Galo", ou "*Tic Tac Toe*", o jogo da velha é um jogo extremamente simples, que não possui grandes dificuldades para seus jogadores. Seu nome teria se originado na Inglaterra, quando nos finais de tarde, mulheres se reuniriam para conversar e bordar. A mulheres idosas, por não terem mais condições de bordar em razão da fraqueza de suas vistas, jogavam este jogo simples.

O jogo da velha é um dos exemplos mais clássicos de utilização do algoritmo Minimax. O estado inicial e os movimentos válidos para cada lado definem a árvore do jogo correspondente ao jogo. A Figura 2.1 mostra parte da árvore de jogo para o jogo da velha. A partir do estado inicial, MAX tem nove movimentos possíveis. O jogo se alterna entre a colocação de um `X` por MAX e a colocação de um `O` por MIN até que se alcance nós de folhas correspondentes a estados terminais, tais que um jogador tem três símbolos em uma linha, coluna ou ainda diagonal; ou até que todos os quadrados estejam preenchidos. O número em cada nó de folha indica o valor de utilidade do estado terminal, do ponto de vista de MAX; valores altos são considerados bons para MAX e ruins para MIN. Cabe a MAX usar a árvore de busca para determinar o melhor movimento.

2.1 J*2*Velha

J*2*Velha (*Java 2 Velha*) é uma implementação do Jogo da Velha desenvolvida na Linguagem Java utilizando o algoritmo MiniMax. Consiste de 4 classes, quais sejam:

1. **Velha.java** - Classe principal da Aplicação;
2. **Minimax.java** - Classe responsável em aplicar o algoritmo MiniMax;
3. **Tabuleiro.java** - Classe responsável pela manipulação do tabuleiro do jogo;

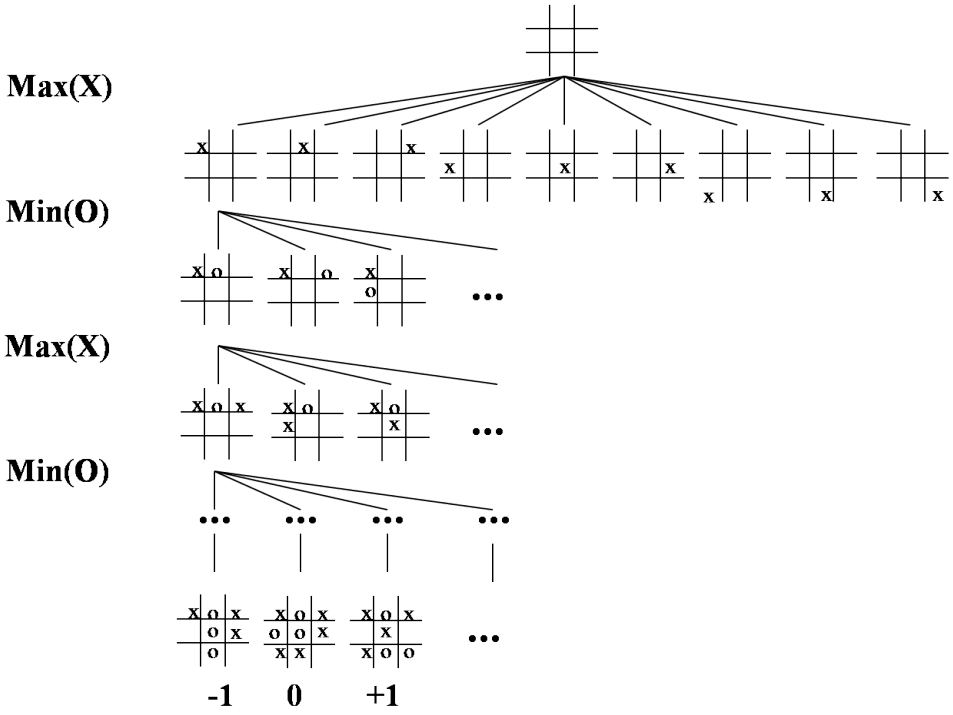

Figura 2.1: Árvore de busca parcial para o jogo da velha

.

4. **Sucessor.java** - Classe responsável em gerar os sucessores, utilizados no algoritmo MiniMax.

O algoritmo Minimax desenvolvido no J2Velha pode buscar por profundidade infinita (até que se encontre um estado terminal) ou por alguma profundidade determinada. A implementação da escolha de profundidade deu-se em função da complexidade do algoritmo MiniMax. Se a profundidade máxima da árvore é `m` e existem `b` movimento válidos em cada ponto, a complexidade de tempo do algoritmo MiniMax é $\mathbf{O}(b^m)$.

Na Caixa de Código XX tem-se um trecho do algoritmo MiniMax contido na classe `Minimax.java`. É possível comparar esta implementação com o algoritmo apresentado no Capítulo anterior.

Código 2.1: Implementação do Algoritmo MiniMax

```
/*
 * Método de decisão do MiniMax
 */
  public int[][] decisao_minimax (int[][] tab)
  {
    /*
     * Limpa os sucessores
```

```
     */
9    sucessores.clear();

     /*
12    * Recebe a utilidade máxima
      */
     int v = valor_max(tab, true, 1);
15
     /*
      * Percorre a lista em busca do primeiro sucessor com utilidade máxima
18    */
     for (Sucessor s: sucessores)
       if (s.utilidade == v)
21       return s.tabuleiro;

     return tab;
24  }

    public int valor_max(int[][] tab, boolean prim, int prof)
27  {
     /*
      * Se a profundidade for maior que a máxima ou o jogo acabou, retorna a
30    * utilidade
      */
     if (prof++ > maxProf || teste_terminal(tab))
33     return utilidade(tab);

     /*
36    * Atribui o menor valor de um inteiro para v (-infinito)
      */
     int v = Integer.MIN_VALUE;
39
     /*
      * Percorre os não sucessores de MAX
42    */
     for (Sucessor s: gerar_sucessores(tab, 1))
     {
45     v = Math.max(v, valor_min(s.tabuleiro, prof));
       s.utilidade = v;
       /*
48      * Se forem os primeiros sucessores, adiciona na lista de sucessores
            ...
        */
       if (prim)
51       sucessores.add(s);
     }
```

```
54      return v;
    }

57  public int valor_min (int[][] tab, int prof)
    {
      /*
60     * Se a profundidade for maior que a máxima ou o jogo acabou, retorna a
       * utilidade
       */
63    if (prof++ > maxProf || teste_terminal (tab))
        return utilidade (tab);

66    /*
       * Atribui +Infinito
       */
69    int v = Integer.MAX_VALUE;

      /*
72     * Percorre os nóss sucessores de MIN
       */
      for (Sucessor s: gerar_sucessores (tab, -1))
75    {
        v = Math.min(v, valor_max (s.tabuleiro, false, prof));
        s.utilidade = v;
78    }

      return v;
81  }
```

2.2 Exemplos

A seguir tem-se a execução de algumas jogadas. Primeiramente vamos utilizar um tabuleiro de tamanho 3x3. Para tal não se faz necessária a definição de uma profundidade máxima, pois a resposta do algoritmo é rápida. Executando o classe `Velha.java` tem-se a seguinte saída no prompt de comando:

```
UFABC - J2VELHA
Bem vindo ao Jogo!
Boa Sorte!


   |   |
---+---+---
   |   |
---+---+---
   |   |

Sua jogada:
Linha [0 - 2]:
```

O jogador decide jogar na linha 0, coluna 1. Tem-se então o resultado da jogada do computador:

```
...
Sua jogada:
Linha [0 - 2]: 0
Coluna [0 - 2]: 1
   | o |
---+---+---
   |   |
---+---+---
   |   |

Jogada do Computador:
 x | o |
---+---+---
   |   |
---+---+---
   |   |
```

```
Sua jogada:
Linha [0 - 2]:
```

Para decidir onde jogar, o computador efetuou todo o algoritmo minimax e escolheu uma posição que lhe favoreça, ao mesmo tempo que prejudique (não agora, pode ser nas próximas jogadas) o adversário. O jogador agora decide jogar na linha 1, coluna 1. Tem-se a seguinte jogada do computador:

```
...
Linha [0 - 2]: 1
Coluna [0 - 2]: 1
 x | o |
---+---+---
   | o |
---+---+---
   |   |

Jogada do Computador:
 x | o |
---+---+---
   | o |
---+---+---
   | x |

Sua jogada:
Linha [0 - 2]:
```

Observe que o computador decidiu jogar em uma posição que evita que o adversário ganhe. O jogador decide jogar na linha 0, coluna 2. Tem-se a jogada do computador:

```
...
Linha [0 - 2]: 0
Coluna [0 - 2]: 2
 x | o | o
---+---+---
   | o |
---+---+---
   | x |
```

```
Jogada do Computador:
 x | o | o
---+---+---
   | o |
---+---+---
 x | x |

Sua jogada:
Linha [0 - 2]:
```

Observe que de qualquer forma o computador ganhará a partida. O jogador decide jogar na linha 2, coluna 2. Tem-se a vitória do computador:

```
...
Linha [0 - 2]: 2
Coluna [0 - 2]: 2
 x | o | o
---+---+---
   | o |
---+---+---
 x | x | o

Jogada do Computador:
 x | o | o
---+---+---
 x | o |
---+---+---
 x | x | o

O computador ganhou!
```

Você pode verificar o funcionamento do jogo com um tabuleiro 4x4. Basta mudar as variáveis `TAM` e `PROF` na classe `Velha`. Devido à complexidade do algoritmo recomenda-se utilizar uma profundidade 5 para que o tempo de execução do mesmo seja razoável. A seguir tem-se uma partida completa utilizando um tabuleiro 4x4:

```
UFABC - J2VELHA
Bem vindo ao Jogo!
Boa Sorte!


   |   |   |
---+---+---+---
   |   |   |
---+---+---+---
   |   |   |
---+---+---+---
   |   |   |

Sua jogada:
Linha [0 - 3]: 0
Coluna [0 - 3]: 0
 o |   |   |
---+---+---+---
   |   |   |
---+---+---+---
   |   |   |
---+---+---+---
   |   |   |

Jogada do Computador:
 o | x |   |
---+---+---+---
   |   |   |
---+---+---+---
   |   |   |
---+---+---+---
   |   |   |



Sua jogada:
Linha [0 - 3]: 1
Coluna [0 - 3]: 0
```

```
 o | x |   |
---+---+---+---
 o |   |   |
---+---+---+---
   |   |   |
---+---+---+---
   |   |   |

Jogada do Computador:
 o | x | x |
---+---+---+---
 o |   |   |
---+---+---+---
   |   |   |
---+---+---+---
   |   |   |


Sua jogada:
Linha [0 - 3]: 2
Coluna [0 - 3]: 0
 o | x | x |
---+---+---+---
 o |   |   |
---+---+---+---
 o |   |   |
---+---+---+---
   |   |   |

Jogada do Computador:
 o | x | x |
---+---+---+---
 o |   |   |
---+---+---+---
 o |   |   |
---+---+---+---
 x |   |   |
```

```
Sua jogada:
Linha [0 - 3]: 1
Coluna [0 - 3]: 1
 o | x | x |
---+---+---+---
 o | o |   |
---+---+---+---
 o |   |   |
---+---+---+---
 x |   |   |

Jogada do Computador:
 o | x | x | x
---+---+---+---
 o | o |   |
---+---+---+---
 o |   |   |
---+---+---+---
 x |   |   |

Sua jogada:
Linha [0 - 3]: 2
Coluna [0 - 3]: 2
 o | x | x | x
---+---+---+---
 o | o |   |
---+---+---+---
 o |   | o |
---+---+---+---
 x |   |   |

Jogada do Computador:
 o | x | x | x
---+---+---+---
 o | o |   |
---+---+---+---
 o |   | o |
---+---+---+---
 x |   |   | x
```

```
Sua jogada:
Linha [0 - 3]: 1
Coluna [0 - 3]: 2
 o | x | x | x
---+---+---+---
 o | o | o |
---+---+---+---
 o |   | o |
---+---+---+---
 x |   |   | x

Jogada do Computador:
 o | x | x | x
---+---+---+---
 o | o | o | x
---+---+---+---
 o |   | o |
---+---+---+---
 x |   |   | x

Sua jogada:
Linha [0 - 3]: 2
Coluna [0 - 3]: 3
 o | x | x | x
---+---+---+---
 o | o | o | x
---+---+---+---
 o |   | o | o
---+---+---+---
 x |   |   | x
Jogada do Computador:
 o | x | x | x
---+---+---+---
 o | o | o | x
---+---+---+---
 o | x | o | o
---+---+---+---
 x |   |   | x
```

```
Sua jogada:
Linha [0 - 3]: 3
Coluna [0 - 3]: 2
 o | x | x | x
---+---+---+---
 o | o | o | x
---+---+---+---
 o | x | o | o
---+---+---+---
 x |   | o | x

Jogada do Computador:
 o | x | x | x
---+---+---+---
 o | o | o | x
---+---+---+---
 o | x | o | o
---+---+---+---
 x | x | o | x

Empate!
```

O código completo da implementação encontra-se no Anexo I.

Anexo A

J2Velha: Códigos

A seguir tem-se a codificação completa da aplicação. Esta foi desenvolvida utilizando a IDE NetBeans e JDK 1.6.

A.1 Classe `Velha.java`

```
/*
 * UFABC - Unversidade Federal do ABC
 * MC 3303 - Inteligência Artificial
 * Professor Jerônimo Pellegrini
 * Alunos:
 *       André Filipe de Moraes Batista
 *       Luís Fernando de Oliveira Jacintho
 */

/*
 * CLASSE VELHA  - CLASSE PRINCIPAL DA APLICACAO
 */


//Biblioteca Scanner para captura da jogada do usuário
import java.util.Scanner;

public class Velha
{
  /*
   * CONSTANTES UTILIDAZAS
   * TAM -> Tamanho do Tabuleiro
   * PROF -> Profundidade máxima da busca no MiniMax. Se PROF = -1 o
       algoritmo
   * minimax irá buscar até um estado terminal.
   */
```

```
  static int TAM = 3, PROF = -1;

  public static void main (String[] args)
  {
    Scanner ent = new Scanner (System.in);
    //Objeto da Classe Tabuleiro
    Tabuleiro t = new Tabuleiro (TAM);
    //Objeto da Classe Minimax
    MiniMax mm = new MiniMax (TAM, PROF);
    System.out.println("UFABC - J2VELHA\nBem vindo ao Jogo!\nBoa Sorte!\n\n
        ");
    //Imprime o tabuleiro na Tela
    t.imprimir ();
    do
    {  //Captura jogada do usuário
      int l, c;
      System.out.printf ("Sua jogada:\r\nLinha [0 - %d]: ", (TAM-1));
      l = ent.nextInt ();
      System.out.printf ("Coluna [0 - %d]: ", (TAM-1));
      c = ent.nextInt ();
      //Realiza jogada do usuário
      t.fazerJogada(l, c);
      t.imprimir ();
      //Verifica se não é um estado terminal
      if (!mm.teste_terminal(t.tabuleiro))
      {
        //Aplica o algoritmo minimax ao tabuleiro
        t.tabuleiro = mm.decisao_minimax(t.tabuleiro);
        System.out.println ("Jogada do Computador:");
        t.imprimir ();
      }
    } while (!mm.teste_terminal(t.tabuleiro));
   //Verifica o ganhador, ou um empate
    if (mm.ganhou(t.tabuleiro, 1))
      System.out.println("O computador ganhou!");
    else if (mm.ganhou(t.tabuleiro, -1))
      System.out.println("Você ganhou!");
    else
      System.out.println("Empate!");
  }
}
```

A.2 Classe Tabuleiro.java

```
/*
 * UFABC - Unversidade Federal do ABC
 * MC 3303 - Inteligência Artificial
 * Professor Jerônimo Pellegrini
 * Alunos:
 *       André Filipe de Moraes Batista
 *       Luís Fernando de Oliveira Jacintho
 */

/*
 * CLASSE TABULEIRO - REPRESENTA O TABULEIRO NO JOGO DA VELHA
 */



public class Tabuleiro
{
  /*
   * Vetor de conversão para impressão na tela
   */
  static char[] conversao = {'o', ' ', 'x'};
  /*
   * Matriz do tabuleiro
   */
  static int[][] tabuleiro;
  /*
   * Tamanho do tabuleiro
   */
  int tam;
  /*
   * Divisor das linhas na tela
   */
  String divisor;

  /*
   * O método construtor recebe como parametro o tamanho do tabuleiro
   */
  public Tabuleiro (int tam)
  {
    this.tam = tam;
    tabuleiro = new int[tam][tam];
    divisor = gerarDivisor ();
  }

  /*
```

```
     * Método invocado para a jogada do Jogador
     */
48   public void fazerJogada (int l, int c)
    {
      if (tabuleiro[l][c] == 0)
51      tabuleiro[l][c] = -1;
      else
       System.out.println ("Posicao ja ocupada, perdeu a vez!");
54   }

    /*
57   * Metodo para a impressão do tabuleiro na tela
     */
    public void imprimir ()
60  {
      for (int i = 0; i < tam; i++)
      {
63      for (int j = 0; j < tam; j++)
        {
          System.out.printf (" %c %c", conversao[tabuleiro[i][j] + 1], j == (
              tam-1) ? ' ' : '|');
66      }
        if (i != (tam-1))
          System.out.println(divisor);
69    }
      System.out.println("\r\n");
    }
72

    /*
     * Metodo para Gerar o Divisor de Linhas. Serve para auxilio da
         visualizacao
75   * grafica do tabuleiro
     */
    public String gerarDivisor ()
78  {
      String d = new String ("\r\n");

81    for (int i = 0; i < (tam - 1); i++)
      {
        d += "---+";
84    }

      d += "---";
87

      return d;
    }
90 }
```

A.3 Classe Sucessor.java

```
/*
 * UFABC - Unversidade Federal do ABC
 * MC 3303 - Inteligência Artificial
 * Professor Jerônimo Pellegrini
 * Alunos:
 *       André Filipe de Moraes Batista
 *       Luís Fernando de Oliveira Jacintho
 */
/*
 * CLASSE SUCESSOR  - GERA OS ESTADOS DO JOGO DA VELHA
 */

public class Sucessor
{
  int[][] tabuleiro;
  int utilidade;

  /*
   *  Metodo Construtor
   */
  public Sucessor (int[][] tab)
  {
    /*
     * Cria um novo tabuleiro, baseado no que foi passado
     */
    int tam = tab.length;
    tabuleiro = new int[tam][tam];

    for (int i = 0; i < tam; i++)
      for (int j = 0; j < tam; j++)
        tabuleiro[i][j] = tab[i][j];
  }
}
```

A.4 Classe Minimax.java

```
/*
 * UFABC - Unversidade Federal do ABC
 * MC 3303 - Inteligência Artificial
 * Professor Jerônimo Pellegrini
 * Alunos:
 *       André Filipe de Moraes Batista
 *       Luís Fernando de Oliveira Jacintho
 */
/*
 * CLASSE MINIMAX - ALGORITMO DE BUSCA COMPETITIVA
 */



import java.util.ArrayList;
import java.util.Collections;

public class MiniMax
{
  /*
   * Lista de Sucessores. Esta lista Ã© armazenada utilizando
   * um ArrayList
   */
  static ArrayList<Sucessor> sucessores = new ArrayList<Sucessor> ();
  int tam, maxProf;

  /*
   * Construtor recebe o tamanho do tabuleiro e a profundidade mÃ¡xima da
       busca
   */
  public MiniMax (int tam, int maxProf)
  {
    this.tam = tam;
    if (maxProf > 0)
      this.maxProf = maxProf;
    else
      this.maxProf = Integer.MAX_VALUE; //Recebe o maior valor de um
          inteiro.
  }

  /*
   * Metodo de decisao do MiniMax
   */
  public int[][] decisao_minimax (int[][] tab)
  {
```

```
    /*
45   * Limpa os sucessores
     */
    sucessores.clear();
48
    /*
     * Recebe a utilidade mÃ¡xima
51   */
    int v = valor_max(tab, true, 1);

54  /*
     * Percorre a lista em busca do primeiro sucessor com utilidade mÃ¡xima
     */
57  for (Sucessor s: sucessores)
      if (s.utilidade == v)
        return s.tabuleiro;
60
    return tab;
  }
63
  public int valor_max(int[][] tab, boolean prim, int prof)
  {
66  /*
     * Se a profundidade for maior que a mÃ¡xima ou o jogo acabou, retorna
        a
     * utilidade
69   */
    if (prof++ > maxProf || teste_terminal(tab))
      return utilidade(tab);
72
    /*
     * Atribui o menor valor de um inteiro para v (-infinito)
75   */
    int v = Integer.MIN_VALUE;

78  /*
     * Percorre os nós sucessores de MAX
     */
81  for (Sucessor s: gerar_sucessores(tab, 1))
    {
      v = Math.max(v, valor_min(s.tabuleiro, prof));
84    s.utilidade = v;
      /*
       * Se forem os primeiros sucessores, adiciona na lista de sucessores
          ...
87     */
      if (prim)
```

```
            sucessores.add(s);
90        }

          return v;
93      }

        public int valor_min (int[][] tab, int prof)
96      {
          /*
           * Se a profundidade for maior que a máxima ou o jogo acabou, retorna a
99         * utilidade
           */
          if (prof++ > maxProf || teste_terminal (tab))
102         return utilidade (tab);

          /*
105        * Atribui +Infinito
           */
          int v = Integer.MAX_VALUE;
108
          /*
           * Percorre os nós sucessores de MIN
111        */
          for (Sucessor s: gerar_sucessores (tab, -1))
          {
114         v = Math.min(v, valor_max (s.tabuleiro, false, prof));
            s.utilidade = v;
          }
117
          return v;
        }
120
        /*
         * Gera os sucessores de um jogador, a partir do estado atual
123      */
        public ArrayList<Sucessor> gerar_sucessores (int[][] tab, int v)
        {
126       ArrayList<Sucessor> suc = new ArrayList<Sucessor> ();
          for (int i = 0; i < tam; i++)
          {
129         for (int j = 0; j < tam; j++)
            {
              if (tab[i][j] == 0)
132           {
                tab[i][j] = v;
                suc.add(new Sucessor (tab));
135             tab[i][j] = 0;
```

```
        }
      }
    }

    return suc;
  }

  /*
   * Verifica se chegou em algum estado terminal e caso afirmativo finaliza
       o jogo
   */
  public boolean teste_terminal (int[][] tab)
  {
    return (ganhou (tab, 1) || ganhou (tab, -1) || semEspaco (tab));
  }

  /*
   * Retorna a utilidade
   */
  public int utilidade (int[][] tab)
  {
    if (ganhou (tab, 1))
      return 1;
    else if (ganhou (tab, -1))
      return -1;
    else
      return 0;
  }

  /*
   * Verifica se jogador ganhou
   */
  public boolean ganhou (int[][] tab, int v)
  {
    for (int i = 0; i < tam; i++)
      if (ganhouLinha (tab, i, v) || ganhouColuna (tab, i, v))
        return true;

    if (ganhouDiag1(tab, v) || ganhouDiag2 (tab, v))
      return true;

    return false;
  }

  /*
   * Ganhou na sequencia de linhas?
   */
```

```
  private boolean ganhouLinha (int[][] tab, int l, int v)
183 {
    for (int i = 0; i < tam; i++)
      if (tab[l][i] != v)
186     return false;

    return true;
189 }

    /*
192  * Ganhou na sequencia de colunas?
     */
    private boolean ganhouColuna (int[][] tab, int c, int v)
195 {
    for (int i = 0; i < tam; i++)
      if (tab[i][c] != v)
198     return false;

    return true;
201 }

    /*
204  * Ganhou na sequencia diagonal principal?
     */
    private boolean ganhouDiag1 (int[][] tab, int v)
207 {
    for (int i = 0; i < tam; i++)
      if (tab[i][i] != v)
210     return false;

    return true;
213 }

    /*
216  * Ganhou na sequencia diagonal secundaria?
     */
    private boolean ganhouDiag2 (int[][] tab, int v)
219 {
    for (int i = 0; i < tam; i++)
      if (tab[(tam-1)-i][i] != v)
222     return false;

    return true;
225 }

    /*
228  * Nao tem mais espacos restantes no tabuleiro..
```

```
    */
   public boolean semEspaco (int[][] tab)
231 {
     for (int l = 0; l < tam; l++)
       for (int c = 0; c < tam; c++)
234      if (tab[l][c] == 0)
           return false;

237  return true;
   }
}
```

Anexo B

J*2*Velha: Novas Funcionalidades

A seguir tem-se a codificação completa de novas funcionalidades do J*2*Velha. O programa agora realiza o algoritmo minimax juntamente com o mecanismo de Poda Alfa-Beta (*Alpha-beta pruning*). Além disto, existe a possibilidade de jogar com elementos de acaso, isto é, as peças podem deslizar em determinada jogada. Todo o código está comentado para que estas funcionalidades sejam entendidas mais facilmente.

B.1 Classe `Velha.java`

```
/*
 * UFABC - Unversidade Federal do ABC
 * MC 3303 - Inteligência Artificial
 * Professor Jerônimo Pellegrini
 * Alunos:
 *       André Filipe de Moraes Batista
 *       Luís Fernando de Oliveira Jacintho
 */

import java.util.Scanner;

public class Velha
{
  /*
   * Peças escorregadias?
   */
  static boolean ESCORREGA;

  public static void main (String[] args)
  {
    Scanner ent = new Scanner (System.in);
```

```
    System.out.print ("Você deseja jogar com peças escorregadias? [s/n]: ")
        ;
    String esc = ent.nextLine();

    if (esc.charAt(0) == 's' || esc.charAt(0) == 'S')
    {
      ESCORREGA = true;
      System.out.println ("Peças escorregadias ativadas.");
    }
    else
    {
      ESCORREGA = false;
      System.out.println ("Peças escorregadias desativadas.");
    }

    Tabuleiro t = new Tabuleiro (ESCORREGA);
    MiniMax mm = new MiniMax (ESCORREGA);
    t.imprimir ();


    do
    {
      int l, c;
      System.out.printf ("Sua jogada:\r\nLinha [0 - 3]: ");
      l = ent.nextInt ();
      System.out.printf ("Coluna [0 - 3]: ");
      c = ent.nextInt ();
      t.fazerJogada(l, c);
      t.imprimir ();
      if (!mm.teste_terminal(t.tabuleiro))
      {
        long time = System.currentTimeMillis ();
        t.tabuleiro = mm.decisao_minimax(t.tabuleiro);
        time = System.currentTimeMillis () - time;
        System.out.println ("Jogada do Computador (" + time + " ms):");
        t.imprimir ();
      }
    } while (!mm.teste_terminal(t.tabuleiro));

int u = mm.utilidade(t.tabuleiro);
    if (u < 0)
      System.out.println ("Parabens! Voce ganhou...");
    else if (u == 0)
      System.out.println ("Empatou!");
    else
      System.out.println ("Voce realmente e pior que um computador...");
```

```
        System.out.println("Você marcou " + mm.contaPontos(t.tabuleiro, -1) + "
            pontos.");
69      System.out.println("O computador marcou " + mm.contaPontos(t.tabuleiro,
            1) + " pontos.");

    }
72  }
```

B.2 Classe Tabuleiro.java

```
/*
 * UFABC - Unversidade Federal do ABC
 * MC 3303 - Inteligência Artificial
 * Professor Jerônimo Pellegrini
 * Alunos:
 *       André Filipe de Moraes Batista
 *       Luís Fernando de Oliveira Jacintho
 */****************************************/

import java.util.ArrayList;

public class Tabuleiro
{
  /*
   * Vetor de conversão para impressão na tela
   */
  static char[] conversao = {'o', ' ', 'x'};
  /*
   * Matriz do tabuleiro
   */
  static int[][] tabuleiro;
  /*
   * PeÃ§as Escorregadias?
   */
  boolean escorrega;

  /*
   * Construtor
   *   entrada: tamanho do tabuleiro
   */
  public Tabuleiro (boolean escorrega)
  {
    this.escorrega = escorrega;
    tabuleiro = new int[4][4];
  }

  /*
   * Método invocado para a jogada do Jogador!
   */
  public void fazerJogada (int l, int c)
  {
    if (tabuleiro[l][c] == 0)
    {
      /*
       * Se estiver jogando com pecas escorregadias...
```

```
        */
       if (escorrega)
48     {
         /*
          * Verifica os vizinhos livre da posiÃ§Ã£o..
51        */
         ArrayList<int []> vizinhos = vizinhosLivres(l, c);

54       /*
          * Se houver ao menos um vizinho livre, tem 20% de chance da peça
          * escorregar..
57        */
         if (vizinhos.size() > 0 && Math.random() <= 0.2)
         {
60         /*
            * Escolhe um dos vizinhos aleatoriamente..
            */
63         int x = (int) (Math.random() * vizinhos.size());
           /*
            * Transforma as coordenadas atuais nas coordenadas do vizinho
66          * escolhido..
            */
           l = vizinhos.get(x)[0];
69         c = vizinhos.get(x)[1];
           System.out.println ("A peça escorregou e caiu na posição: " + l +
               ", " + c);
         }
72     }
       tabuleiro[l][c] = -1;
     }
75   else
       System.out.println ("Posição já ocupada, perdeu a vez!");
   }
78
   /*
    * Método que verifica se há vizinhos livres, considerando as diagonais
       ...
81  */
   public ArrayList<int []> vizinhosLivres (int l, int c)
   {
84   ArrayList<int []> vizinhos = new ArrayList<int []> ();

     /*
87    * Vizinhos da linha anterior, se houver...
      */
     if (l > 0)
90   {
```

```
      if (c > 0)
        if (tabuleiro[l-1][c-1] == 0)
93        vizinhos.add(new int[] {l-1, c-1});

      if (tabuleiro[l-1][c] == 0)
96      vizinhos.add(new int[] {l-1, c});

      if (c < 3)
99      if (tabuleiro[l-1][c+1] == 0)
          vizinhos.add(new int[] {l-1, c+1});
    }
102
    /*
     * Vizinhos da mesma linha...
105  */
    if (c > 0)
      if (tabuleiro[l][c-1] == 0)
108     vizinhos.add(new int[] {l, c-1});

    if (c < 3)
111   if (tabuleiro[l][c+1] == 0)
        vizinhos.add(new int[] {l, c+1});

114 /*
     * Vizinhos da linha posterior, se houver...
     */
117 if (l < 3)
    {
      if (c > 0)
120     if (tabuleiro[l+1][c-1] == 0)
          vizinhos.add(new int[] {l+1, c-1});

123   if (tabuleiro[l+1][c] == 0)
        vizinhos.add(new int[] {l+1, c});

126   if (c < 3)
        if (tabuleiro[l+1][c+1] == 0)
          vizinhos.add(new int[] {l+1, c+1});
129 }

    return vizinhos;
132 }

    /*
135  * Método para a impressÃ£o do tabuleiro na tela
     */
    public void imprimir ()
```

```
138   {
        for (int i = 0; i < 4; i++)
        {
141       for (int j = 0; j < 4; j++)
          {
            System.out.printf (" %c %c", conversao[tabuleiro[i][j] + 1], j == 3
                ? ' ' : '|');
144       }
          if (i != (3))
            System.out.println("\r\n---+---+---+---");
147     }
        System.out.println("\r\n");
      }
150 }
```

B.3 Classe Sucessor.java

```
/*
 * UFABC - Unversidade Federal do ABC
 * MC 3303 - Inteligência Artificial
 * Professor Jerônimo Pellegrini
 * Alunos:
 *       André Filipe de Moraes Batista
 *       Luís Fernando de Oliveira Jacintho
 */
public class Sucessor
{
  int[][] tabuleiro;
  int utilidade;

  /*
   * Construtor
   */
  public Sucessor (int[][] tab)
  {
    /*
     * Cria um novo tabuleiro, baseado no que foi passado
     */
    int tam = tab.length;
    tabuleiro = new int[tam][tam];

    for (int i = 0; i < tam; i++)
      for (int j = 0; j < tam; j++)
        tabuleiro[i][j] = tab[i][j];
  }
}
```

B.4 Classe Minimax.java

```
/*
 * UFABC - Unversidade Federal do ABC
 * MC 3303 - Inteligência Artificial
 * Professor Jerônimo Pellegrini
 * Alunos:
 *       André Filipe de Moraes Batista
 *       Luís Fernando de Oliveira Jacintho
 */

import java.util.ArrayList;

public class MiniMax
{
  /*
   * Lista dos nós sucessores
   */
  static ArrayList<Sucessor> sucessores = new ArrayList<Sucessor> ();
  /*
   * Jogar com peças escorregadias?
   */
  boolean escorrega;

  /*
   * Construtor
   */
  public MiniMax (boolean escorrega)
  {
    this.escorrega = escorrega;
  }

  /*
   * Método de decisão do MiniMax
   */
  public int[][] decisao_minimax (int[][] tab)
  {
    /*
     * Limpa os sucessores
     */
    sucessores.clear ();

    /*
     * Recebe a utilidade máxima
     */
    int v = valor_max (tab, Integer.MIN_VALUE, Integer.MAX_VALUE, true);

```

```
    /*
     * Percorre a lista em busca do primeiro sucessor com utilidade máxima
48   */
    for (Sucessor s: sucessores)
      if (s.utilidade == v)
51      return s.tabuleiro;

    return tab;
54  }

    public int valor_max (int[][] tab, int alfa, int beta, boolean prim)
57  {
    /*
     * Se a profundidade for maior que a máxima ou o jogo acabou, retorna a
60   * utilidade
     */
    if (teste_terminal (tab))
63    return utilidade (tab);

    /*
66   * Atribui -Infinito
     */
    int v = Integer.MIN_VALUE;
69
    /*
     * Percorre os nós sucessores de MAX
72   */
    for (Sucessor s: gerar_sucessores (tab, 1))
    {
75    v = Math.max(v, valor_min (s.tabuleiro, alfa, beta));
      s.utilidade = v;

78    /*
       * Se forem os primeiros sucessores, adiciona na lista de sucessores
           ...
       */
81    if (prim)
        sucessores.add(s);

84    /*
       * Poda Beta - Se o valor for maior que beta, retorna o valor..
       */
87    if (v >= beta)
        return v;

90    /*
       * Se o valor for maior que Alfa, Alfa recebe o valor...
```

```
         */
93       alfa = Math.max(alfa, v);
       }

96     return v;
     }

99   public int valor_min (int[][] tab, int alfa, int beta)
     {
       /*
102     * Se a profundidade for maior que a máxima ou o jogo acabou, retorna a
        * utilidade
        */
105    if (teste_terminal (tab))
         return utilidade (tab);

108    /*
        * Atribui +Infinito
        */
111    int v = Integer.MAX_VALUE;

       /*
114     * Percorre os nós sucessores de MIN
        */
       for (Sucessor s: gerar_sucessores (tab, -1))
117    {
         v = Math.min(v, valor_max (s.tabuleiro, alfa, beta, false));
         s.utilidade = v;
120
         /*
          * Poda Alfa - Se o valor for menor que alfa, retorna o valor...
123       */
         if (v <= alfa)
           return v;
126
         /*
          * Se valor menor que Beta, Beta o recebe...
129       */
         beta = Math.min(beta, v);
       }
132
       return v;
     }
135
     /*
      * Gera os sucessores de um jogador, a partir do estado atual
138   */
```

```
  public ArrayList<Sucessor> gerar_sucessores (int[][] tab, int v)
  {
141   ArrayList<Sucessor> suc = new ArrayList<Sucessor> ();
    for (int i = 0; i < 4; i++)
    {
144   for (int j = 0; j < 4; j++)
      {
        if (tab[i][j] == 0)
147     {
          /*
           * Se estiver jogando com pecas escorregadias...
150        */
          if (escorrega)
          {
153         /*
             * Verifica os vizinhos livre da posição..
             */
156         ArrayList<int []> vizinhos = vizinhosLivres(tab, i, j);
            /*
             * Se houver ao menos um vizinho livre, tem 20% de chance da
                peÃ§a
159          * escorregar..
             */
            if (vizinhos.size() > 0 && Math.random() <= 0.2)
162         {
                /*
                 * Escolhe um dos vizinhos aleatoriamente..
165              */
              int x = (int) (Math.random() * vizinhos.size());
              /*
168            * Transforma as coordenadas atuais nas coordenadas do
                  vizinho
               * escolhido..
               */
171           i = vizinhos.get(x)[0];
              j = vizinhos.get(x)[1];
            }
174       }

          tab[i][j] = v;
177       suc.add(new Sucessor (tab));
          tab[i][j] = 0;
        }
180   }
    }

183   return suc;
```

```
  }

186 /*
   * Método que verifica se há vizinhos livres, considerando as diagonais
       ...
   */
189 public ArrayList<int []> vizinhosLivres (int[][] tabuleiro, int l, int c)
  {
    ArrayList<int []> vizinhos = new ArrayList<int []> ();
192
    /*
     * Vizinhos da linha anterior, se houver...
195  */
    if (l > 0)
    {
198   if (c > 0)
        if (tabuleiro[l-1][c-1] == 0)
          vizinhos.add(new int[] {l-1, c-1});
201
      if (tabuleiro[l-1][c] == 0)
        vizinhos.add(new int[] {l-1, c});
204
      if (c < 3)
        if (tabuleiro[l-1][c+1] == 0)
207       vizinhos.add(new int[] {l-1, c+1});
    }

210 /*
     * Vizinhos da mesma linha...
     */
213 if (c > 0)
      if (tabuleiro[l][c-1] == 0)
        vizinhos.add(new int[] {l, c-1});
216
    if (c < 3)
      if (tabuleiro[l][c+1] == 0)
219     vizinhos.add(new int[] {l, c+1});

    /*
222  * Vizinhos da linha posterior, se houver...
     */
    if (l < 3)
225 {
      if (c > 0)
        if (tabuleiro[l+1][c-1] == 0)
228       vizinhos.add(new int[] {l+1, c-1});
```

```
      if (tabuleiro[l+1][c] == 0)
231     vizinhos.add(new int[] {l+1, c});

      if (c < 3)
234     if (tabuleiro[l+1][c+1] == 0)
          vizinhos.add(new int[] {l+1, c+1});
    }
237
    return vizinhos;
  }
240
  /*
   * Fim de jogo?
243 * O jogo só termina se não houver mais espaço para jogadas...
   */
  public boolean teste_terminal (int[][] tab)
246 {
    return (semEspaco (tab));
  }
249
  /*
   * Retorna a utilidade...
252 * Aqui a utilidade considerada é a diferença de pontos entre o
      computador e
   * o jogador, o computador não deseja apenas vencer, mas também humilhar
      =P
   */
255 public int utilidade (int[][] tab)
  {
    int pc, usr;
258
    pc = contaPontos(tab, 1);
    usr = contaPontos (tab, -1);
261
    return (pc-usr);
  }
264
  /*
   * Verifica se jogador ganhou
267 */
  public int contaPontos (int[][] tab, int v)
  {
270 int pontos = 0;

    for (int i = 0; i < 4; i++)
273 {
      pontos += contaLinha (tab, i, v);
```

```
        pontos += contaColuna (tab, i, v);
276     }

        pontos += contaDiag1(tab, v);
279     pontos += contaDiag2(tab, v);

        return pontos;
282   }

      /*
285    * Pontos na sequencia de linhas?
       *
       * Método de contagem binária.. um byte é desnecessário, precisaria
           apenas
288    * de 4 bits.. Basicamente, para cada posição atribui-se o valor 1 na
           mesma
       * posição do byte, indicando que ali é dele. No final checamos as 3
       * possibilidades de marcar pontos, 4 posições vizinhas (1111) ou 3
           posicoes
291    * vizinhas (0111 ou 1110). Qualquer outra combinação teria menos do que
       * 3 posições vizinhas e não marcariam pontos.
       */
294   private int contaLinha (int[][] tab, int l, int v)
      {
        byte soma = 0;
297
        for (int i = 0; i < 4; i++)
          if (tab[l][i] == v)
300         soma += (1 << i);

        if (soma == 15) // 1111
303       return 3;
        else if ((soma == 7) ||  (soma == 14)) // 0111 v 1110
          return 1;
306     else
          return 0;
      }
309
      /*
       * Pontos na sequencia de colunas?
312    */
      private int contaColuna (int[][] tab, int c, int v)
      {
315     int soma = 0;

        for (int i = 0; i < 4; i++)
318       if (tab[i][c] == v)
```

```
        soma += (1 << i);

    if (soma == 15) // 1111
      return 3;
    else if ((soma == 7) || (soma == 14)) // 0111 v 1110
      return 1;
    else
      return 0;
  }

  /*
   * Ganhou na sequencia diagonal?
   */
  private int contaDiag1 (int[][] tab, int v)
  {
    int soma = 0;
    int pextra = 0;

    for (int i = 0; i < 4; i++)
      if (tab[i][i] == v)
        soma += (1 << i);

    /*
     * Nas duas diagonais a seguir só é possível formar sequencias de 3,
     * podendo-se adicionar entao apenas 1 ponto....
     */
    if (tab[1][0] == v && tab[2][1] == v && tab[3][2] == v)
      pextra++;
    if (tab[0][1] == v && tab[1][2] == v && tab[2][3] == v)
      pextra++;

    if (soma == 15)
      return 3 + pextra;
    else if ((soma == 7) || (soma == 14))
      return 1 + pextra;
    else
      return 0 + pextra;
  }

  /*
   * Ganhou na sequencia diagonal?
   */

   private int contaDiag2 (int[][] tab, int v)
  {
    int soma = 0;
    int pextra = 0;
```

```
366
        for (int i = 0; i < 4; i++)
          if (tab[3-i][i] == v)
369         soma += (1 << i);

        /*
372      * Nas duas diagonais a seguir só é possível formar sequencias de 3,
         * podendo-se adicionar entao apenas 1 ponto....
         */
375     if (tab[0][2] == v && tab[1][1] == v && tab[2][0] == v)
          pextra++;
        if (tab[1][3] == v && tab[2][2] == v && tab[3][1] == v)
378       pextra++;

        if (soma == 15)
381       return 3 + pextra;
        else if ((soma == 7) || (soma == 14))
          return 1 + pextra;
384     else
          return 0 + pextra;
      }
387
      /*
       * Não tem mais espacos restantes no tabuleiro..
390    */
      public boolean semEspaco (int[][] tab)
      {
393     for (int l = 0; l < 4; l++)
          for (int c = 0; c < 4; c++)
            if (tab[l][c] == 0)
396           return false;

        return true;
399   }
}
```