FACULDADE · DE · CIÊNCIAS · UNIVERSIDADE · DE · LISBOA
DEPARTAMENTO · DE · INFORMÁTICA

Grupo IIA018:
Celso Sousa Nº32441
Sérgio das Neves Nº32536
Henrique Cunha Nº33321

IIA 2008/2009
Relatório do projecto Nº5
Algoritmos Genéticos
Evolução do Jogador de Konane
(Damas havaianas)

Conteúdo

Introdução

Este texto refere-se ao projecto do grupo IIA018 com o tema: Konane (algoritmos genéticos).

O objectivo do nosso trabalho é a evolução do melhor jogador de Konane.

Este projecto vem no seguimento do campeonato de Konane, realizado no corrente ano lectivo de 2008/2009.

Definições

População: conjunto de cromossomas.
Cromossoma: cadeia de bits de um tamanho fixo; cada cromossoma é dividido em conjuntos de igual comprimento denominados genes.
Gene: No nosso contexto, representa o peso de uma característica de um jogador/cromossoma.

Descrição do projecto

Dada uma população inicial de jogadores criados aleatoriamente, pretende-se fazer evoluir essa população ao longo de um dado número de gerações, com o intuito de encontrar um bom jogador de Konane.

Cada jogador tem uma função de avaliação própria que devolve o valor da sua capacidade de sobrevivência. Esta função toma em linha de conta as várias características das peças no tabuleiro e pondera com pesos diferentes cada uma dessas características.

Considerando n características c_i e n pesos associados, a fórmula da função de avaliação utilizada é: (somatório com i de 1..n (p_i * c_i)):

$$f = \sum_{i}^{n} p_i \times c_i$$

Esta função de *fitness* define a qualidade de um jogador. Para isso realizamos torneios entre os vários jogadores que fazem parte de uma geração.

O que é um bom jogador de Konane?

Um bom jogador de Konane é aquele que, ao longo de determinado número de gerações, não só não é eliminado das populações que surgem ao longo dessas mesmas gerações, como ainda consegue ser o jogador com mais vitórias (maior valor de *fitness*) após a realização do torneio entre todos os jogadores da última geração verificada.

A função de avaliação que decidimos implementar tem em linha de conta doze características:

- O número de peças deste jogador;
- O número de peças do adversário;
- A mobilidade deste jogador;
- A mobilidade do adversário;
- O número de movimentos possíveis deste jogador;
- O número de movimentos possíveis do adversário;
- O número de peças deste jogador que podem comer duas peças do adversário;
- O número de peças do adversário que podem comer duas peças deste jogador;
- O número de peças (deste jogador) posicionadas nos cantos;
- O número de peças (do adversário) posicionadas nos cantos;
- O número de peças (deste jogador) posicionadas ao longo das paredes do tabuleiro (excepto os cantos);
- O número de peças (do adversário) posicionadas ao longo das paredes do tabuleiro (excepto os cantos).

Decidimos que cada gene dos cromossomas deve ter um comprimento de 4 bits, de modo a não termos cadeias de bits muito extensas.

Assim, os cromossomas de todos os jogadores terão um comprimento de 48 bits (12 características * tamanho do gene).

Estrutura de ficheiros do projecto

O nosso projecto que se encontra dentro da pasta geneticos/ apresenta a seguinte estrutura em árvore:

- geneticos/

- geneticos/konane/
- geneticos/geneticos_java/
- geneticos/lib/

- geneticos/konane/campeonato.pl
- geneticos/konane/oalphabetaLim.pl
- geneticos/konane/ojogo.pl
- geneticos/konane/oKonane.pl
- geneticos/konane/osjogadores.pl
- geneticos/konane/server.pl
- geneticos/geneticos_java/GeraPopulacao.java
- geneticos/geneticos_java/IdFit.java
- geneticos/geneticos_java/InterF.java
- geneticos/geneticos_java/Populacao.java
- geneticos/geneticos_java/PrologInteraction.java
- geneticos/geneticos_java/XOverCutOp.java
- geneticos/lib/geneticos.jar

Linguagens utilizadas

Na concretização deste projecto foram utilizadas as linguagens Java e Prolog.

Java

Usámos a linguagem Java para criar aleatoriamente e manipular uma população inicial, aplicando a selecção, a recombinação genética e a mutação, para a fazer evoluir.

Com este intento, usámos algumas classes da biblioteca gajit, que se encontra disponível para *download* no site:

http://www.micropraxis.com/gajit/index.html

Mais concretamente, usámos as classes Chrom, ChromItem, ExtendedBitSet, View, FixView, GenOp e MutOp.

Para além destas, implementámos as seguintes classes: Populacao, GeraPopulacao, PrologInteraction, IdFit, XOverCutOp e InterF.

Descrição da funcionalidade destas classes

- A classe Chrom (subclasse de ExtendedBitSet) representa um cromossoma, constituído por um id único e por uma cadeia binária.
- A classe ChromItem associa um *fitness* a um Chrom.
- A classe MutOp (subclasse de GenOp) aplica uma mutação a um cromossoma: mediante uma dada probabilidade de mutação, são trocados mais ou menos bits.
- A classe FixView (subclasse de View) mapeia um gene de um cromossoma (que são 4 bits), num double, dentro de um intervalo definido pelo programador.
- A classe Populacao representa uma população.
- A classe GeraPopulacao gera a população, constrói os pesos de cada cromossoma e comunica com a PrologInteraction para as trocas de informação com o Prolog.
- A classe PrologInteraction realiza a interacção entre o Java e o Prolog.
- A classe IdFit associa um id a um *fitness*.
- A Classe XOverCutOp é o operador que permite fazer a recombinação genética, usando um determinado ponto de corte, cria um novo cromossoma com os bits mais significativos de um dos pais e os bits menos significativos do outro pai (optámos por não usar os operadores de recombinação da biblioteca gajit porque apesar de algumas das classes utilizarem o ponto de corte, faziam operações que não nos interessavam tanto enquanto outras recebiam outro tipo de parâmetros.
- A classe InterF fornece uma interface gráfica para facilitar a utilização do programa.

Prolog

Usámos a linguagem Prolog para avaliar os jogadores.

Descrição dos ficheiros

- O ficheiro campeonato.pl realiza um campeonato entre todos os jogadores. Considerámos campeonato como sendo vários minicamps, cada um destes, constituído por um par de jogos entre cada jogador.
- O ficheiro osjogadores.pl constrói dinamicamente as funções de avaliação e os pesos. Contém os predicados das várias características tidas em conta e o predicado que calcula o valor do estado de um tabuleiro para um determinado jogador, considerando esses pesos e as características.
- O ficheiro server.pl faz a interacção com o java, recebendo a lista de pesos e respectivos ids e devolvendo a lista de ids e respectivos *fitness*, ordenada por ordem decrescente de *fitness*.
- Incluímos a biblioteca gajit.jar disponibilizada pelos professores. Deste modo facilitamos a execução do programa, caso contrário, quem o queria executar teria de fazer *download* e compilar a biblioteca.

Modo de utilização

Para representar a população usámos uma lista de ChromItems. Os cromossomas constituintes dos ChromItems foram criados aleatoriamente e os valores de *fitness* iniciais são inicializados a 0.0.

Utilizando a classe FixView, vamos a cada gene de cada cromossoma, convertendo-o de binário para um double no intervalo [-8,7]. Este valor é o peso de um gene (característica).

Tendo a lista de doubles e sabendo o número de genes, facilmente se consegue criar uma lista de tuplos, em que a primeira componente do tuplo é o id respeitante a um cromossoma e a segunda componente é a lista de pesos associada a esse cromossoma. No nosso caso, dado que o número de genes é 12, teremos listas de pesos com 12 doubles.

É esta lista de tuplos que é passada ao Prolog.

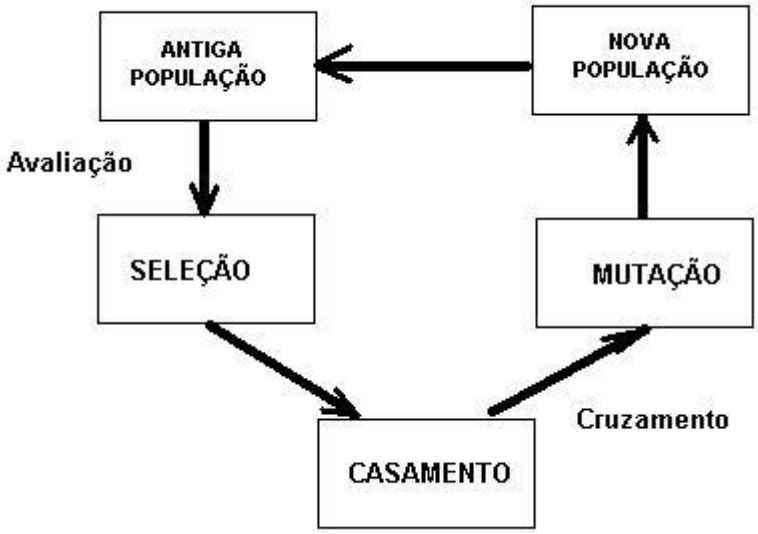
Por seu turno, o Prolog pega na lista de tuplos e faz as seguintes operações:

- Faz assert dos predicados que chamam as funções de avaliação, das próprias funções de avaliação e dos pesos.
- Chama o predicado cria_populacao_inicial/1, que dada a lista de tuplos vinda do Java, faz assert de tantos factos dados_jogador/2 quantos os tuplos da lista, recebendo como parâmetros o id e os pontos (com o valor zero), para que todos os jogadores comecem o campeonato em pé de igualdade.
- Realiza o campeonato entre todos os jogadores. Para cada minicamp, verifica se houve um jogador vitorioso ou um empate e actualiza os pontos em conformidade (2 pontos para a vitória, 1 para o empate e 0 para a derrota).
- No final do campeonato, devolve a lista de tuplos (id_jogador,pontos_jogador) ordenada por ordem decrescente de pontos.
- Faz retract dos predicados que chamam as funções de avaliação, das próprias funções de avaliação e dos pesos.
- Faz retract dos factos dados_jogador/2.

O Java recebe do Prolog esta lista e faz o seguinte:

- Constrói uma lista de IdFits, como forma de facilitar a associação entre cada id e cada *fitness*.
- Actualiza os valores de *fitness* de cada ChromItem.
- Cria uma nova geração.

Criação de uma nova geração



Selecção - método da roleta

O método da roleta consiste no seguinte:

- Somar os valores de todos os *fitness*;
- Gerar um número aleatório entre 0 e o (somatório dos *fitness*) -1;
- Para cada cromossoma, vamos somando o seu valor de *fitness* a uma variável auxiliar e vamos vendo quando é que essa variável ultrapassa o valor aleatório gerado. Quando assim for, retorna-se o cromossoma correspondente ao *fitness* ultimamente somado a essa variável.

Exemplo:

Jogador 1: *fitness* 30 (intervalo [0,29]).
Jogador 2: *fitness* 45 (intervalo [30,74].
Jogador 3: *fitness* 10 (intervalo [75,84]).

É gerado um valor aleatório entre 0 e 83. Supondo que sai 30, então é escolhido para reproduzir o jogador 2, visto o valor gerado situar-se no intervalo desse mesmo jogador.

Cruzamento

Selecciona dois cromossomas aleatoriamente (método da roleta), e cruza-os, utilizando a classe XOverCutOp, segundo um ponto de corte.

Como o tamanho da população é fixo de geração para geração (identificado pelo número de cromossomas), são feitos tantos cruzamentos quanto o número de filhos necessários para atingir esse tamanho.

Podem existir cruzamentos de um cromossoma com ele próprio, o que significa que ocorre elitismo, ou seja, estamos a fazer uma cópia de um indivíduo desta geração para a geração seguinte.

Mutação

Cada filho gerado sofre uma tentativa de mutação, tendo em conta uma determinada taxa de mutação (podem ser ou não alterados alguns bits).

Com a nova população gerada, o processo repete-se até atingir o número de gerações pretendido pelo utilizador.

Interface com o utilizador

Esta interface gráfica foi feita em java (classe InterF). Tentámos que fosse intuitiva e simples de utilizar.

Quando o programa é executado (ver adiante como fazer), são apresentadas 4 *text boxes*, onde se pode introduzir:

- O tamanho da população;
- O número de gerações que quer utilizar para evoluir um jogador de Konane.
- O ponto de corte, utilizado na recombinação.
- A taxa de mutação, utilizada quando é realizada mutação.

O ponto de corte e a taxa de mutação vêm, respectivamente, com os valores 23 e 0.005%, valores que achámos adequados para uma execução normal do programa. No entanto, o utilizador pode mudá-los ao seu gosto se assim o entender.

Quatro botões:

- "Encontrar melhor jogador" que é usado para dar início à evolução;
- *"Reset* dos dados", que põe nas *text boxes* os dados por defeito (que estavam no início da execução do programa);
- "Acerca", que apresenta uma caixa de diálogo com informação de quem desenvolveu o programa;
- "Sair", que abandona o programa.

Uma *text area*, onde são colocados os resultados após cada campeonato, mais precisamente, o id do jogador que venceu o campeonato e a lista de pesos associada a esse jogador. No final de todas as gerações, é apresentado o vencedor (id e lista de pesos).

Modo de execução do projecto

Para executar o projecto, é necessário primeiro configurar o Java e o Prolog para trabalharem em conjunto (ver link seguinte):

https://sicstus.sics.se/

Para além disso, é necessário adicionar à variável CLASSPATH a biblioteca gajit.jar, biblioteca essa que se encontra no nosso projecto, na directoria lib.

Após realizar todas as configurações, basta executar o servidor Prolog da seguinte forma:

1. Ir à linha de comandos e pôr a directoria konane (do nosso projecto) como directoria corrente.
2. Escrever:
   sicstus -l server.pl --goal "main."

Para executar o Java (cliente) basta:

1. Abrir outra linha de comandos e tornar a directoria geneticos_java como directoria corrente.
2. Escrever:
   java InterF

Dificuldades encontradas

Ao longo do desenvolvimento do projecto, tivemos algumas dificuldades, que foram supridas quer pelo nosso esforço, quer pela ajuda dos professores Luiz Moniz e Paulo Urbano. Foram elas:

- Compreensão da documentação da biblioteca gajit.
- Inserção no Prolog, de forma dinâmica, das funções de avaliação dos jogadores e dos pesos associados a esses jogadores.

Alterações efectuadas ao projecto entregue

- Correcção de bugs no método da recombinação.
- O ponto de corte passou a não constar na interface, sendo agora gerado aleatoriamente de forma interna, entre 0 e numGenes*tamanhoGene-1. A taxa de mutação também deixou de constar da interface, sendo agora fixada internamente (0.005).
- Na interface, é agora apresentada uma lista de características lidas de um ficheiro anteriormente escrito pelo prolog na raiz do projecto (de nome caracteristicas.txt), podendo o utilizador escolher quais delas quer usar para evoluir os jogadores.
- Criação de dois ficheiros do tipo .bat (server.bat e cliente.bat), sendo que o primeiro executa o servidor em prolog e o segunda corre o programa em Java.
- Criação do ficheiro osjogadoresteste.pl com o intuito de colocar dois quaisquer jogadores a jogar entre si. Permite testar se o melhor jogador que surgir de correr o nosso programa um certo número de gerações ganha a um jogador feito por nós. Basta passar a este predicado como parâmetro duas listas: a primeira contém tuplos constituídos pelo id e lista de pesos desse jogador e a segunda por uma lista de listas de características respectivas a cada jogador. Permite listas de pesos e de características de diferentes tamanhos.
- Aumentámos o tamanho dos genes de 4 para 5 bits.
- Alterámos o intervalo de [-8,7] para [-1,1] porque achámos que intuitivamente assim era melhor. Desta maneira estamos a dar mais valores dentro de um intervalo mais pequeno, ao passo que da outra estávamos a dar menos valores dentro de um intervalo maior.

Testes e resultados

Corremos o programa com uma população de tamanho 20 em 50 gerações com 8 características (mobilidades, paredes e cantos, possibilidade de comer duas peças).

- Lista de pesos do vencedor à 10ª geração: [-0,226; 0,290; -0,097; -0,419; 0,290; 0,419; 0,742; 0,742].
- Lista de pesos do vencedor à 20ª geração: [0,032; 0,935; -0,806; 0,032; 0,613; 0,871; 0,032; -0,226].
- Lista de pesos do vencedor à 30ª geração: [1,000; -0,871; 0,806; 0,161; -0,355; -0,419; 0,032; -0,226].
- Lista de pesos do vencedor à 40ª geração: [-0.032, -0.871, 0.806, 0.161, -0.355, -0.419, 0.032, -0.677].
- Lista de pesos do vencedor à 50ª geração: [0.032, 0.677, 0.226, 0.161, -0.419, -0.161, 0.032, -0.161].

Corremos o programa também com uma população de tamanho 20 em 25 gerações com 4 características (meus_moves_possiveis, moves_possiveis_dele, minha_mobilidade, mobilidade_dele).

- Lista de pesos do vencedor à 10ª geração: [0.806, 0.806, -0.226, 0.484].
- Lista de pesos do vencedor à 20ª geração: [-0.742, 0.290, 0.806, 0.484].
- Lista de pesos do vencedor à 25ª geração: [0.032, 0.935, -0.226, 1.000].

Para testar a função de avaliação que levámos ao campeonato com os melhores jogadores que obtivémos à 50ª e 25ª geração que obtivemos de correr o nosso programa, efectuámos no sicstus os seguintes testes, após consultar o ficheiro ojogo.pl:

```
iniciar_teste([(1,[2,-2.2,2,-2.2]),(2,[0.032,0.677,0.226,0.161,-0.419,-0.161,0.032,-0.161])],[[meus_moves_possiveis,moves_possiveis_dele,minha_mobilidade,mobilidade_dele],[minha_mobilidade,mobilidade_dele,minhasPodeComerDuas,delePodeComerDuas,meusCantosDaCor,deleCantosDaCor,minhasParedes,deleParedes]],LR).
```

UM PAR DE JOGOS

pretas = 1 CONTRA brancas = 2

Vitoria de 1

pretas = 2 CONTRA brancas = 1

Vitoria de 1
LR = [(1,2),(2,0)] ?

```
iniciar_teste([(1,[2,-2.2,2,-2.2]),(2,[0.032, 0.935, 0.226, 1.000])],
[[meus_moves_possiveis,moves_possiveis_dele,minha_mobilidade,mobilidade_dele],[meus_moves_possiveis,moves_possiveis_dele,minha_mobilidade,mobilidade_dele]],LR).
```

UM PAR DE JOGOS

pretas = 1 CONTRA brancas = 2

Vitoria de 1

pretas = 2 CONTRA brancas = 1

Vitoria de 1
LR = [(1,2),(2,0)] ?

Obtivemos como resultado a vitória do jogador com o id 1, que foi o jogador feito por nós em ambos os testes.

Conclusões

Apesar das experiências feitas, não conseguimos encontrar um jogador melhor do que o que tínhamos para o campeonato. Talvez se tivéssemos mais tempo e jogado em profundidade nível 5 já conseguíssemos, mas não podemos dar garantias disso, dado que não fizemos estudos sobre a evolução dos pesos.

Bibliografia

Para realizarmos este projecto utilizámos alguns artigos e os slides das aulas teóricas.

- http://users.encs.concordia.ca/~kharma/ResearchWeb/html/research/ayo.html
- https://www.tutorialspoint.com/genetic_algorithms/index.htm
- http://www.cs.nott.ac.uk/~gxk/papers/cec2002gxk.pdf
- https://cs.brynmawr.edu/Theses/Thompson.pdf

RUSSELL, S. e NORVIG, P. - *Artificial Intelligence: a modern approach*, Prentice-Hall, 2nd edition, paperback edition, 2003

COELHO, H. - *Inteligência Artificial em 25 lições*, Fundação Calouste Gulbenkian, 1995

ANEXO

Listagem do código Java desenvolvido

GeraPopulacao.java

```
import java.io.IOException;
import java.util.LinkedList;

import com.micropraxis.gajit.FixView;
import com.micropraxis.gajit.View;

/**
 * Classe que gera a populacao, constroi os pesos de cada cromossoma e efectua a interaccao com o
prolog.
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class GeraPopulacao {
  /**
   * A populacao.
   */
  private Populacao jogadores;
  /**
   * A vista.
   */
  private View vista;
  /**
   * A interaccao com o prolog.
   */
  private PrologInteraction interacao;
  /**
   * O numero de cromossomas desta populacao.
   */
  private int numCromossomas;
  /**
   * O numero de genes de cada cromossoma desta populacao.
   */
  private int numGenes;
  /**
   * O tamanho de cada gene.
   */
  private int tamGene;

  /**
   * Constroi uma populacao.
   * @param numCromossomas O numero de cromossomas desta populacao.
   * @param pontoCorte O ponto de corte usado na recombinacao.
   * @param taxaMutacao A taxa de mutacao.
   */
  public GeraPopulacao (int numCromossomas, int pontoCorte, double taxaMutacao) {
    this.numCromossomas = numCromossomas;
```

```
    numGenes = 12;
    tamGene = 4;
    jogadores = new Populacao (numCromossomas, tamGene, numGenes, pontoCorte, taxaMutacao);
    vista = new FixView(tamGene, -8.0, 16.0);
    try {
      interacao = new PrologInteraction(numCromossomas, numGenes);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  /**
   * Constroi os pesos, envia-os ao prolog, recebe a lista dos ids que
   * identificam cada cromossoma e respectivos fitness's, cria uma nova
   * geracao e retorna a string que contem o id e a respectiva lista de pesos do melhor cromossoma da
geracao antiga.
   *
   * @return A string contendo o id e os pesos do melhor jogador.
   */
  public String interage () {
    LinkedList<Double> pesos = constroiPesos();
    String s = interacao.empacotarDados(pesos,jogadores);
    String t = interacao.callProlog(s);
    LinkedList<IdFit> idFits = interacao.desempacotarDados(t);
    int fit = idFits.get(0).getFitness();
    int id = idFits.get(0).getId();
    for (IdFit idf : idFits)
      if (fit < idf.getFitness()){
        fit = idf.getFitness();
        id = idf.getId();
      }
    int index = jogadores.getIndexFromId(id);
    int comeco = index*numGenes;
    StringBuilder sb = new StringBuilder("");
    sb.append(id + ": [");
    for(int i = comeco; i < comeco+numGenes; i++) {
      sb.append(pesos.get(i));
      if (i != comeco+numGenes-1)
        sb.append(",");
    }
    sb.append("]");
    jogadores.newGeneration(idFits);
    return sb.toString();
  }

  /**
   * Devolve uma lista de pesos.
   *
   * @return A lista de pesos.
   */
  public LinkedList<Double> constroiPesos () {
    LinkedList<Double> pesos = new LinkedList<Double>();
    for(int i = 0; i < numCromossomas; i++) {
      for(int j = 0; j < numGenes; j++) {
        pesos.addFirst((Double)vista.getGene(jogadores.getChromItem(i).getChrom(), j));
      }
    }
    return pesos;
  }
}
```

IdFit.java

```
/**
 * A classe que associa um id a um fitness.
 *
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class IdFit {
  /**
   * O fitness.
   */
  private int fitness;
  /**
   * O id.
   */
  private int id;

  /**
   * Constroi um IdFit.
   * @param id O id.
   * @param fitness O fitness.
   */
  public IdFit(int id, int fitness) {
    this.fitness = fitness;
    this.id = id;
  }

  /**
   * Devolve o id deste IdFit.
   * @return O id deste IdFit.
   */
  public int getId(){
    return id;
  }

  /**
   * Devolve o fitness deste IdFit.
   * @return O fitness deste IdFit.
   */
  public int getFitness(){
    return fitness;
  }
}
```

InterF.java

```
import java.awt.Rectangle;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
import javax.swing.SwingWorker;

public class InterF extends JFrame {

  private static final long serialVersionUID = 1L;
  private JPanel jContentPane = null;
  private JTextField jTextFieldTamPop = null;
  private JTextField jTextFieldNumGeracoes = null;
  private JLabel jLabelTamPop = null;
  private JLabel jLabelNumGeracoes = null;
  private JTextField jTextFieldPontoCorte = null;
  private JTextField jTextFieldTaxaMutacao = null;
  private JLabel jLabelPontoCorte = null;
  private JLabel jLabelTaxaMutacao = null;
  private JButton jButtonEncontrarMelhorJogador = null;
  private JButton jButtonLimparDados = null;
  private JButton jButtonSair = null;
  private JButton jButtonAcercaDe = null;
  private JTextArea jTextAreaOMelhorJogador = null;
  private JLabel jLabelOMelhorJogador = null;
  private JScrollPane jScrollPane = null;
  /**
   * This method initializes jTextFieldTamPop
   *
   * @return javax.swing.JTextField
   */
  private JTextField getJTextFieldTamPop () {
    if (jTextFieldTamPop == null) {
      jTextFieldTamPop = new JTextField();
      jTextFieldTamPop.setBounds(new Rectangle(151, 18, 100, 20));
    }
    return jTextFieldTamPop;
  }

  /**
   * This method initializes jTextFieldNumGeracoes
   *
   * @return javax.swing.JTextField
   */
  private JTextField getJTextFieldNumGeracoes () {
    if (jTextFieldNumGeracoes == null) {
      jTextFieldNumGeracoes = new JTextField();
      jTextFieldNumGeracoes.setBounds(new Rectangle(152, 44, 100, 20));
    }
    return jTextFieldNumGeracoes;
```

```
 }

 /**
  * This method initializes jTextFieldPontoCorte
  *
  * @return javax.swing.JTextField
  */
 private JTextField getJTextFieldPontoCorte () {
  if (jTextFieldPontoCorte == null) {
    jTextFieldPontoCorte = new JTextField();
    jTextFieldPontoCorte.setBounds(new Rectangle(152, 71, 100, 20));
    jTextFieldPontoCorte.setText("23");
   }
  return jTextFieldPontoCorte;
 }

 /**
  * This method initializes jTextFieldTaxaMutacao
  *
  * @return javax.swing.JTextField
  */
 private JTextField getJTextFieldTaxaMutacao () {
  if (jTextFieldTaxaMutacao == null) {
    jTextFieldTaxaMutacao = new JTextField();
    jTextFieldTaxaMutacao.setBounds(new Rectangle(152, 102, 100, 20));
    jTextFieldTaxaMutacao.setText("0.005");
   }
  return jTextFieldTaxaMutacao;
 }

 /**
  * This method initializes jButtonEncontrarMelhorJogador
  *
  * @return javax.swing.JButton
  */
 private JButton getJButtonEncontrarMelhorJogador () {
  if (jButtonEncontrarMelhorJogador == null) {
    jButtonEncontrarMelhorJogador = new JButton();
    jButtonEncontrarMelhorJogador.setBounds(new Rectangle(9, 207, 179, 21));
    jButtonEncontrarMelhorJogador.setText("Encontrar melhor jogador");
    jButtonEncontrarMelhorJogador.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed (java.awt.event.ActionEvent e) {
        String tamPopStr = jTextFieldTamPop.getText();
        String numGeracoesStr = jTextFieldNumGeracoes.getText();
        String pontoCorteStr = jTextFieldPontoCorte.getText();
        String taxaMutacaoStr = jTextFieldTaxaMutacao.getText();
        try{
        Integer.parseInt(tamPopStr);
        Integer.parseInt(numGeracoesStr);
        Integer.parseInt(pontoCorteStr);
        Double.parseDouble(taxaMutacaoStr);
        } catch (NumberFormatException nfe) {
          JOptionPane.showMessageDialog(jContentPane,
          "Introduza apenas valores numéricos");
          return;
        }
        if(tamPopStr.equals("") || numGeracoesStr.equals("") || pontoCorteStr.equals("") ||
taxaMutacaoStr.equals("")){
          JOptionPane.showMessageDialog(jContentPane,
          "Introduza os valores em falta");
```

```
            return;
          }
          if (Integer.parseInt(tamPopStr) <= 1){
            JOptionPane.showMessageDialog(jContentPane,
            "Introduza um numero de jogadores maior que 1.");
            return;
          }
          if (Integer.parseInt(numGeracoesStr) <= 0){
            JOptionPane.showMessageDialog(jContentPane,
            "Introduza um numero de geracoes maior que 0.");
            return;
          }
          int tamPop = Integer.parseInt(tamPopStr);
          final int numGeracoes = Integer.parseInt(numGeracoesStr);
          int pontoCorte = Integer.parseInt(pontoCorteStr);
          double taxaMutacao = Double.parseDouble(taxaMutacaoStr);
          final GeraPopulacao gp = new GeraPopulacao(tamPop, pontoCorte, taxaMutacao);
          SwingWorker<Void,Void> worker = new SwingWorker<Void, Void>() {
            @Override
            public Void doInBackground() {
              String melhorJogador = "";
              int i = 0;
              while (i < numGeracoes){
                melhorJogador = gp.interage();
                jTextAreaOMelhorJogador.append("Geracao " + (i+1) + ":\nO vencedor foi:\n" +
melhorJogador + "\n\n");
                i++;
              }
              jTextAreaOMelhorJogador.append("O melhor jogador ao fim de " + numGeracoes +
                  ((numGeracoes == 1) ? " geração" :  " gerações") + " é o jogador:\n" + melhorJogador);
              return null;
            }
          };
          worker.execute();
        }
      });
    }
    return jButtonEncontrarMelhorJogador;
  }

  /**
   * This method initializes jButtonLimparDados
   *
   * @return javax.swing.JButton
   */
  private JButton getJButtonLimparDados () {
    if (jButtonLimparDados == null) {
      jButtonLimparDados = new JButton();
      jButtonLimparDados.setBounds(new Rectangle(9, 234, 144, 24));
      jButtonLimparDados.setText("Reset dos dados");
      jButtonLimparDados.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed (java.awt.event.ActionEvent e) {
          jTextFieldTamPop.setText("");
          jTextFieldNumGeracoes.setText("");
          jTextFieldPontoCorte.setText("23");
          jTextFieldTaxaMutacao.setText("0.005");
          jTextFieldTamPop.requestFocus();
        }
      });
    }
```

```
    return jButtonLimparDados;
  }

  /**
   * This method initializes jButtonSair
   *
   * @return javax.swing.JButton
   */
  private JButton getJButtonSair () {
    if (jButtonSair == null) {
      jButtonSair = new JButton();
      jButtonSair.setBounds(new Rectangle(12, 297, 75, 22));
      jButtonSair.setText("Sair");
      jButtonSair.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed (java.awt.event.ActionEvent e) {
          System.exit(0);
        }
      });
    }
    return jButtonSair;
  }

  /**
   * This method initializes jButtonAcercaDe
   *
   * @return javax.swing.JButton
   */
  private JButton getJButtonAcercaDe () {
    if (jButtonAcercaDe == null) {
      jButtonAcercaDe = new JButton();
      jButtonAcercaDe.setBounds(new Rectangle(12, 269, 100, 20));
      jButtonAcercaDe.setText("Acerca...");
      jButtonAcercaDe.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed (java.awt.event.ActionEvent e) {
          String message = "Programa desenvolvido pelo grupo IIA018 (Celso, Henrique e Sérgio), no ano
lectivo 2008/2009 :)";
          JOptionPane.showMessageDialog(jContentPane, message, "Acerca de geneticos",
JOptionPane.INFORMATION_MESSAGE);
        }
      });
    }
    return jButtonAcercaDe;
  }

  /**
   * This method initializes jTextAreaOMelhorJogador
   *
   * @return javax.swing.JTextArea
   */
  private JTextArea getJTextAreaOMelhorJogador () {
    if (jTextAreaOMelhorJogador == null) {
      jTextAreaOMelhorJogador = new JTextArea();
      jTextAreaOMelhorJogador.setEditable(false);
    }
    return jTextAreaOMelhorJogador;
  }

  /**
   * This method initializes jScrollPane
   *
```

```
 * @return javax.swing.JScrollPane
 */
private JScrollPane getJScrollPane () {
  if (jScrollPane == null) {
    jScrollPane = new JScrollPane();
    jScrollPane.setBounds(new Rectangle(399, 134, 164, 289));
    jScrollPane.setViewportView(getJTextAreaOMelhorJogador());
  }
  return jScrollPane;
}

/**
 * @param args
 */
public static void main (String[] args) {
  // TODO Auto-generated method stub

  SwingUtilities.invokeLater(new Runnable() {
    public void run () {
      InterF thisClass = new InterF();
      thisClass.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      thisClass.setVisible(true);
    }
  });
}

/**
 * This is the default constructor
 */
public InterF () {
  super();
  initialize();
}

/**
 * This method initializes this
 *
 * @return void
 */
private void initialize () {
  this.setSize(600, 600);
  this.setContentPane(getJContentPane());
  this.setTitle("JFrame");
}

/**
 * This method initializes jContentPane
 *
 * @return javax.swing.JPanel
 */
private JPanel getJContentPane () {
  if (jContentPane == null) {
    jLabelOMelhorJogador = new JLabel();
    jLabelOMelhorJogador.setBounds(new Rectangle(287, 141, 101, 16));
    jLabelOMelhorJogador.setText("O melhor jogador");
    jLabelTaxaMutacao = new JLabel();
    jLabelTaxaMutacao.setBounds(new Rectangle(10, 104, 104, 16));
    jLabelTaxaMutacao.setText("Taxa de mutação");
    jLabelPontoCorte = new JLabel();
    jLabelPontoCorte.setBounds(new Rectangle(10, 73, 88, 16));
```

```
        jLabelPontoCorte.setText("Ponto de corte");
        jLabelNumGeracoes = new JLabel();
        jLabelNumGeracoes.setBounds(new Rectangle(13, 45, 124, 16));
        jLabelNumGeracoes.setText("Número de gerações");
        jLabelTamPop = new JLabel();
        jLabelTamPop.setBounds(new Rectangle(8, 20, 138, 16));
        jLabelTamPop.setText("Tamanho da população");
        jContentPane = new JPanel();
        jContentPane.setLayout(null);
        jContentPane.add(getJTextFieldTamPop(), null);
        jContentPane.add(getJTextFieldNumGeracoes(), null);
        jContentPane.add(getJTextFieldPontoCorte(), null);
        jContentPane.add(getJTextFieldTaxaMutacao(), null);
        jContentPane.add(jLabelTamPop, null);
        jContentPane.add(jLabelNumGeracoes, null);
        jContentPane.add(jLabelPontoCorte, null);
        jContentPane.add(jLabelTaxaMutacao, null);
        jContentPane.add(jLabelOMelhorJogador, null);
        jContentPane.add(getJButtonEncontrarMelhorJogador(), null);
        jContentPane.add(getJButtonLimparDados(), null);
        jContentPane.add(getJButtonSair(), null);
        jContentPane.add(getJButtonAcercaDe(), null);
        jContentPane.add(getJScrollPane(), null);
      }
     return jContentPane;
    }

}
```

Populacao.java

```
import java.util.LinkedList;
import java.util.Random;

import com.micropraxis.gajit.Chrom;
import com.micropraxis.gajit.ChromItem;
import com.micropraxis.gajit.MutOp;

/**
 * Classe que representa uma populacao.
 *
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class Populacao {
  /**
   * A lista de ChromItems
   */
  private LinkedList<ChromItem> chromItems;
  /**
   * O numero de genes desta populacao.
   */
  private int numGenes;
  /**
   * O tamanho de um gene desta populacao.
   */
  private int tamanhoGene;
  /**
   * O numero de cromossomas desta populacao.
   */
  private int numCromossomas;
  /**
   * O operador de mutacao.
   */
  private MutOp mutador;
  /**
   * O operador de cruzamento.
   */
  private XOverCutOp cruzador;

  /**
   * Constroi uma populacao aleatoria de um dado numero de cromossomas com tamanho dos genes e
numero de genes definido.
   * Os fitness's em cada ChromItem são inicializados a 0.0.
   * @param numCromossomas O numero de cromossomas.
   * @param tamanhoGene O tamanho dos genes.
   * @param numGenes O numero de genes.
   */
  public Populacao (int numCromossomas, int tamanhoGene, int numGenes) {
    chromItems = new LinkedList<ChromItem>();
    this.numGenes = numGenes;
    this.tamanhoGene = tamanhoGene;
    this.numCromossomas = numCromossomas;
    for (int i = 0; i < numCromossomas; i++){
      Chrom c = new Chrom(this.tamanhoGene*this.numGenes);
```

```
      ChromItem ci = new ChromItem(0.0, c);
      chromItems.addFirst(ci);
    }
    cruzador = new XOverCutOp(((tamanhoGene*numGenes)/2)-1);
    mutador = new MutOp(0.005);
  }

  /**
   * Constroi uma populacao aleatoria de um dado numero de cromossomas com tamanho dos genes,
numero de genes e ponto de corte definidos.
   * A taxa de mutacao e' por defeito 0.005.
   * @param numCromossomas O numero de cromossomas.
   * @param tamanhoGene O tamanho dos genes.
   * @param numGenes O numero de genes.
   * @param pontoDeCorte O ponto de corte.
   */
  public Populacao (int numCromossomas, int tamanhoGene, int numGenes, int pontoDeCorte) {
    this(numCromossomas, tamanhoGene, numGenes);
    cruzador = new XOverCutOp(pontoDeCorte);
  }

  /**
   * Constroi uma populacao aleatoria de um dado numero de cromossomas com tamanho dos genes,
numero de genes e taxa de mutacao definidos.
   * O ponto de corte e' por defeito (tamanhoGene*numGenes)/2)-1) em que tamanhoGene e' 4 e
numGenes e' 12.
   * @param numCromossomas O numero de cromossomas.
   * @param tamanhoGene O tamanho dos genes.
   * @param numGenes O numero de genes.
   * @param taxaMutacao A taxa de mutacao.
   */
  public Populacao (int numCromossomas, int tamanhoGene, int numGenes, double taxaMutacao) {
    this(numCromossomas, tamanhoGene, numGenes);
    mutador = new MutOp(taxaMutacao);
  }

  /**
   * Constroi uma populacao aleatoria de um dado numero de cromossomas com tamanho dos genes,
numero de genes, ponto de corte e taxa de mutacao definidos.
   * @param numCromossomas O numero de cromossomas.
   * @param tamanhoGene O tamanho dos genes.
   * @param numGenes O numero de genes.
   * @param pontoDeCorte O ponto de corte.
   * @param taxaMutacao A taxa de mutacao.
   */
  public Populacao (int numCromossomas, int tamanhoGene, int numGenes, int pontoDeCorte, double
taxaMutacao) {
    this(numCromossomas, tamanhoGene, numGenes);
    cruzador = new XOverCutOp(pontoDeCorte);
    mutador = new MutOp(taxaMutacao);
  }

  /**
   * Actualiza em cada ChromItem o fitness respectivo utilizando para isso a lista de fitness's dada como
parametro
   * e efectua os cruzamentos necessários para obter os individuos da proxima geracao.
   * @param fit A lista de IdFits .
   */
  public void newGeneration(LinkedList<IdFit> fit){
    LinkedList<ChromItem> temp = new LinkedList<ChromItem>();
```

```
    assignFitsToChromItems(fit);
    for(int i = 0; i < numCromossomas; i++){
      Chrom pai = selectRandomChrom();
      Chrom mae = selectRandomChrom();
      Chrom filho = cruzador.apply(pai, mae);
      mutador.apply(filho);
      ChromItem ci = new ChromItem(0.0, filho);
      temp.addFirst(ci);
    }
    chromItems = temp;
  }

  /**
   * Selecciona e devolve um cromossoma aleatorio segundo o metodo de seleccao denominado roleta.
   * Este metodo funciona do seguinte modo:
   * Faz a soma de todos os fitness's para achar o valor maximo da roleta. De seguida, gera um numero
aleatorio entre 0 e a soma - 1.
   * Depois, percorre todos os ChromItems ate encontrar um, cujo fitness associado seja maior que o
numero aleatorio gerado e
   * devolve o cromossoma correspondente.
   * Quanto maior o fitness de um dado cromossoma, maior a probabilidade de ele ser escolhido.
   * @return O cromossoma.
   */
  private Chrom selectRandomChrom () {
    double soma = somatorioFits();
    Random gerador = new Random();
    int aleatorio = gerador.nextInt((int)soma);
    double verificador = 0.0;
    for (ChromItem ci : chromItems){
      verificador += ci.getDoubleFitness();
      if (verificador >= aleatorio)
        return ci.getChrom();
    }
    return null;
  }

  /**
   * Devolve o somatorio de todos os fitness's presentes na lista de ChromItems.
   * @return O somatorio dos fitness's.
   */
  private double somatorioFits () {
    double soma = 0.0;
    for (ChromItem ci : chromItems){
      soma += ci.getDoubleFitness();
    }
    return soma;
  }

  /**
   * Actualiza os ids na lista de chromItems atraves dos ids actualizados que vem na lista de IdFits.
   * @param fit A lista de IdFits.
   */
  private void assignFitsToChromItems(LinkedList<IdFit> fit){
    for (ChromItem ci : chromItems){
      for (IdFit f : fit){
        if (f.getId() == getIdFromChrom(ci.getChrom()))
          ci.setFitness(f.getFitness());
      }
    }
  }
```

```
 /**
  * Obtem o id de um dado cromossoma.
  * @param c O cromossoma.
  * @return O id que identifica este cromossoma.
  */
 public int getIdFromChrom(Chrom c){
   String[] arr = c.toString().split(":");
   return Integer.parseInt(arr[0].substring(1));
 }

 /**
  * Obtem o indice relativo ao cromossoma com o id dado como parametro.
  * @param id O id de um cromossoma.
  * @return O indice correspondente a este id.
  */
 public int getIndexFromId (int id) {
   for (int i = 0; i < chromItems.size(); i++)
     if (getIdFromChrom(chromItems.get(i).getChrom()) == id)
       return i;
   return -1;
 }

 /**
  * Obtem o chromItem no indice dado como parametro.
  * @param index O indice do chromItem.
  * @return O chromItem num dado indice.
  */
 public ChromItem getChromItem (int index) {
   return chromItems.get(index);
 }
}
```

PrologInteraction.java

```
import java.util.LinkedList;
import se.sics.prologbeans.Bindings;
import se.sics.prologbeans.PBTerm;
import se.sics.prologbeans.PrologSession;
import se.sics.prologbeans.QueryAnswer;

/**
 * Realiza a interaccao entre o java e o prolog.
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class PrologInteraction {
  /**
   * A sessao que servira para fazer a interaccao com o prolog.
   */
  private PrologSession session = new PrologSession();
  /**
   * O numero de cromossomas.
   */
  private int numCromossomas;
  /**
   * O numero de genes.
   */
  private int numGenes;
  /**
   * Liga o java ao prolog ate o java receber de volta uma resposta.
   * @param numCromossomas O numero de cromossomas.
   * @param numGenes O numero de genes.
   * @throws java.io.IOException
   */
  public PrologInteraction (int numCromossomas, int numGenes) throws java.io.IOException {
    this.numCromossomas = numCromossomas;
    this.numGenes = numGenes;
    session.setTimeout(0);
    session.connect();
  }

  /**
   * Devolve a string que sera passada ao prolog e que contem uma lista de tuplos
   * constituida por id e lista de pesos respectiva ao cromossoma com esse id.
   * @param pesos A lista de pesos relativa a esta populacao.
   * @param jogadores A populacao.
   * @return A string com a lista de ids e pesos respectivos que sera passada ao prolog.
   */
  public String empacotarDados (LinkedList<Double> pesos, Populacao jogadores) {
    StringBuilder sb = new StringBuilder("");
    sb.append("[(");
    for (int i = 0; i < numCromossomas; i++) {
      sb.append(jogadores.getIdFromChrom(jogadores.getChromItem(i).getChrom()) + ",[");
      for(int j = 0; j < numGenes; j++) {
        sb.append(pesos.get(j));
        if(j != (numGenes-1))
          sb.append(",");
      }
```

```
      if(i != (numCromossomas-1))
        sb.append("]),(");
      }
    sb.append("])]");
    return sb.toString();
  }

  /**
   * Coloca os pares dos ids de jogadores e fitness's respectivos numa lista de IdFits.
   * @param s A string que vem do prolog que contem a lista ordenada por fitness's.
   * @return Devolve uma lista de IdFits que sao tuplos (id,fitness).
   */
  public LinkedList<IdFit> desempacotarDados (String s) {
    String[] tuplos = s.split("\\),\\(");
    tuplos[0] = tuplos[0].substring(2);
    tuplos[numCromossomas-1] = tuplos[numCromossomas-1].substring(0, tuplos[numCromossomas-
1].length()-2);
    LinkedList<IdFit> idFits = new LinkedList<IdFit>();
    for (int i = 0; i < tuplos.length; i++){
      String[] arr = tuplos[i].split(",");
      idFits.add(new IdFit(Integer.parseInt(arr[0]), Integer.parseInt(arr[1])));
    }
    return idFits;
  }

  /**
   * Passa ao prolog a string com os ids e os pesos respectivos,
   * recebe a lista de ids com os fitness's respectivos.
   * @param s A string de ids e pesos respectivos que vai ser passada ao prolog.
   * @return A string com os ids e os fitness's ordenada por fitness's.
   */
  public String callProlog(String s) {
    try {
      Bindings bindings = new Bindings().bind("S", s + ".");
      QueryAnswer answer = session.executeQuery("my_predicate(S,R)", bindings);
      PBTerm result = answer.getValue("R");
      if (result != null) {
        String str = result.toString();//Este metodo toString() coloca uma virgula a mais antes de cada tuplo
da string.
        StringBuilder sb = new StringBuilder(str.replaceAll(",,", ","));//Tira todas as virgulas a mais excepto
a que esta antes do primeiro parentesis curvo.
        return sb.deleteCharAt(1).toString();//Apaga a primeira virgula da lista.
      } else {
        return "Error: " + answer.getError() + '\n';
      }
    } catch (Exception e) {
      e.printStackTrace();
      return("Error when querying Prolog Server: " +
          e.getMessage() + '\n');
    }
  }
}
```

XOverCutOp.java

```
import com.micropraxis.util.ExtendedBitSet;
import com.micropraxis.gajit.Chrom;
import com.micropraxis.gajit.GenOp;

/**
 * Classe que representa o operador de recombinacao genetica.
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class XOverCutOp extends GenOp {
  /**
   * O ponto de corte.
   */
  private int pontoDeCorte;

  /**
   * Cria o operador de cruzamento com um dado ponto de corte.
   * @param pontoDeCorte O ponto de corte.
   */
  public XOverCutOp (int pontoDeCorte) {
    super(true);
    this.pontoDeCorte = pontoDeCorte;
  }

  /**
   * Aplica o cruzamento sobre dois cromossomas gerando um cromossoma filho
   * (requer que os cromossomas tenham o mesmo tamanho).
   * @param c1 O cromossoma pai.
   * @param c2 O cromossoma mae.
   * @return O cromossoma filho.
   */
  public Chrom apply(Chrom c1, Chrom c2){
    String[] arr1 = c1.toString().split(":");
    String cadeia1 = arr1[1].substring(0, arr1[1].length()-1);
    String[] arr2 = c2.toString().split(":");
    String cadeia2 = arr2[1].substring(0, arr2[1].length()-1);
    String cadeiaNova = cadeia1.substring(0, pontoDeCorte+1) + cadeia2.substring(pontoDeCorte+1);
    return new Chrom(new ExtendedBitSet(cadeiaNova));
  }
}
```

Listagem (final) do código actualizado

Prolog

campeonato.pl

```
/*
Estes predicados em dynamic podem ser asserted e retracted da base de conhecimento.
*/
:- dynamic(dados_jogador/2).

/*
Encontra o melhor jogador de Konane, devolve a lista de jogadores
identificados pelos seus id's e ordenada por pontos.
*/
iniciar(L,LCars,LR) :-
        write(L),nl,adiciona_funcoes(L),
        adiciona_pesos(L,LCars),
        adiciona_func_avals(L),
        cria_populacao_inicial(L),
        jogar_campeonato(L),
        organiza_jogadores(LR),
        limpar_pontos,
        retira_func_avals(L),
        retira_pesos(L,LCars),
        retira_funcoes(L).

/*
Cria uma populacao inicial de jogadores,
todos com zero pontos e um numero de jogador individual unico.
*/
cria_populacao_inicial([]).
cria_populacao_inicial([(ID,_)|RL]) :-
        assert(dados_jogador(ID,0)),
        cria_populacao_inicial(RL).

/*
Limpa todos os jogadores na base de conhecimento ao fim de cada geracao.
*/
limpar_pontos :-
        retractall(dados_jogador(_,_)).

/*
Efectua todos os jogos de um campeonato.
*/
jogar_campeonato([(_,_)]).
jogar_campeonato([L|LS]) :-
        jogar_um_contra_todos([L|LS]),
        jogar_campeonato(LS).

/*
Coloca o jogador no primeiro tuplo da lista a jogar contra todos os restantes jogadores
contra quem ainda nao jogou.
*/
```

```
jogar_um_contra_todos([(_,_)]).
jogar_um_contra_todos([(ID,_),(IDOpon,_)|R]) :-
        minicamp(1,3,ID,IDOpon,VitoriasJog,VitoriasOponente),
        ve_quem_ganha(ID,IDOpon,VitoriasJog,VitoriasOponente),
        jogar_um_contra_todos([(ID,_)|R]).

/*
Verifica se o jogo teve um vencedor ou se houve empate e actualiza os pontos em conformidade.
*/
ve_quem_ganha(NumJogador,_,VitoriasJog,VitoriasOponente) :-
        VitoriasJog > VitoriasOponente,
        obter_pontos_jogador(NumJogador,Pontos),
        NovosPontos is Pontos + 2,
        actualiza_pontos_jogador(NumJogador,Pontos,NovosPontos).

ve_quem_ganha(_,NumOponente,VitoriasJog,VitoriasOponente) :-
        VitoriasOponente > VitoriasJog,
        obter_pontos_jogador(NumOponente,PontosOponente),
        NovosPontosOponente is PontosOponente + 2,
        actualiza_pontos_jogador(NumOponente,PontosOponente,NovosPontosOponente).

ve_quem_ganha(NumJogador,NumOponente,_,_) :-
        obter_pontos_jogador(NumJogador,Pontos),
        obter_pontos_jogador(NumOponente,PontosOponente),
        NovosPontos is Pontos + 1,
        NovosPontosOponente is PontosOponente + 1,
        actualiza_pontos_jogador(NumJogador,Pontos,NovosPontos),
        actualiza_pontos_jogador(NumOponente,PontosOponente,NovosPontosOponente).

/*
Obtem os pontos de um dado jogador.
*/
obter_pontos_jogador(NumJogador,Pontos) :-
        dados_jogador(NumJogador,Pontos).

/*
Actualiza o fitness de um dado jogador dado o seu id.
*/
actualiza_pontos_jogador(NumJogador,PontosAntigos,NovosPontos) :-
        retract(dados_jogador(NumJogador,PontosAntigos)),
        assert(dados_jogador(NumJogador,NovosPontos)).

/*
Obtém todos os jogadores e seus pontos e devolve outra lista com esses mesmos jogadores
identificados pelo seu id e ordenada pelos respectivos pontos do maior para o menor valor.
*/
organiza_jogadores(SortedPlayerList) :-
        setof((NumJogador,Pontos),dados_jogador(NumJogador,Pontos),UnsortedPlayerList),
        ordena_jogadores_por_pontos(UnsortedPlayerList,SortedPlayerList).

/*
Realiza a chamada ao predicado de ordenacao com a lista desordenada.
*/
ordena_jogadores_por_pontos(UnsortedPlayerList,SortedPlayerList) :-
        ordena_jogadores_por_pontos(UnsortedPlayerList,[],SortedPlayerList).

/*
Ordena todos os jogadores pelo criterio do maior fitness usando uma lista intermedia
que serve de suporte para a construcao da lista resultado que ser a lista ordenada,
os primeiros jogadores sao os que tem mais pontos
```

```
(em caso de empate quem tiver id menor aparece primeiro na lista ordenada).
*/
ordena_jogadores_por_pontos([],NewListSorted,NewListSorted).
ordena_jogadores_por_pontos(UnsortedPlayerList,LI,NewListSorted) :-
        maior_da_lista(UnsortedPlayerList,MaiorTuplo),
        apaga(MaiorTuplo,UnsortedPlayerList,UnsortedPlayerListWithoutGreater),
        inserir_na_cauda(LI,MaiorTuplo,NewLI),
        ordena_jogadores_por_pontos(UnsortedPlayerListWithoutGreater,NewLI,NewListSorted).

/*
Apaga um elemento (o primeiro) que aparece numa dada lista.
*/
apaga(X,[X|Xs],Xs).
apaga(X,[Y|Ys],[Y|Yss]) :-
        X \= Y,
        apaga(X,Ys,Yss).

/*
Devolve o comprimento de uma lista.
*/
comprimento([],0).
comprimento([_|Xs],Z) :-
        comprimento(Xs,V),
        Z is V + 1.

/*
Dada uma lista de tuplos (Numero_de_jogador,Pontos_desse_Jogador), devolve o tuplo com maior valor
de pontos na lista.
Este predicado usa o predicado auxiliar maior_da_lista/3 para facilitar o cálculo.
*/
maior_da_lista([(NumJog,Pontos)|Resto],(NumJogComMaisPontos,MaiorVal)) :-
        MaxIntermedio = Pontos,
        NumJogIntermedio = NumJog,
        maior_da_lista(Resto,MaxIntermedio,NumJogIntermedio,(NumJogComMaisPontos,MaiorVal)).

maior_da_lista([],MaxIntermedio,NumJogIntermedio,(NumJogComMaisPontos,MaxIntermedio)) :-
        NumJogComMaisPontos = NumJogIntermedio.
maior_da_lista([(NumJog,Pontos)|R],MaxIntermedio,NumJogIntermedio,(NumJogComMaisPontos,Maio
rVal)) :-
        (Pontos > MaxIntermedio,
        NumJogInt = NumJog,
        NovoMaior = Pontos,
        maior_da_lista(R,NovoMaior,NumJogInt,(NumJogComMaisPontos,MaiorVal)));
        (Pontos =< MaxIntermedio,
        maior_da_lista(R,MaxIntermedio,NumJogIntermedio,(NumJogComMaisPontos,MaiorVal))).
```

osjogadores.pl

```
/*
Fornece um gerador de numeros aleatorios.
*/
:- use_module(library(random)).

/*
Adiciona as funcoes de avaliacao a base de conhecimento.
*/
adiciona_funcoes([]).
adiciona_funcoes([(ID,_)|RL]) :-
        name(mr_arre_burro_,L),
        constroi_componentes(ID,LIC),
        constroi_nome(L,LIC,LR),
        name(Func,LR),
        Y =.. [Func,Cor,Tab,V],
        W =.. [final,Cor,Res],
        M =.. [valorFinal,Cor,Res,V],
        name(pesos_,LP),
        constroi_componentes(ID,LICP),
        constroi_nome(LP,LICP,LRP),
        name(FuncP,LRP),
        N =.. [FuncP,Pesos,Cars],
        O =.. [soma_pesada,Cor,Tab,Pesos,Cars,V],
        assert((Y :- (W,M);(N,O))),
        adiciona_funcoes(RL).

/*
Retira as funcoes de avaliacao da base de conhecimento.
*/
retira_funcoes([]).
retira_funcoes([(ID,_)|RL]) :-
        name(mr_arre_burro_,L),
        constroi_componentes(ID,LIC),
        constroi_nome(L,LIC,LR),
        name(Func,LR),
        Y =.. [Func,Cor,Tab,V],
        W =.. [final,Cor,Res],
        M =.. [valorFinal,Cor,Res,V],
        name(pesos_,LP),
        constroi_componentes(ID,LICP),
        constroi_nome(LP,LICP,LRP),
        name(FuncP,LRP),
        N =.. [FuncP,Pesos,Cars],
        O =.. [soma_pesada,Cor,Tab,Pesos,Cars,V],
        retract((Y :- (W,M);(N,O))),
        retira_funcoes(RL).

/*
Adiciona os predicados que chamam as funcoes de avaliacao a base de conhecimento.
*/
adiciona_func_avals([]).
adiciona_func_avals([(ID,_)|RL]) :-
        name(mr_arre_burro_,L),
        constroi_componentes(ID,LIC),
        constroi_nome(L,LIC,LR),
        name(Func,LR),
```

```
        A =.. [func_avaliacao,ID,Func],
        assert(A),
        adiciona_func_avals(RL).

/*
Retira os predicados que chamam as funcoes de avaliacao da base de conhecimento.
*/
retira_func_avals([]).
retira_func_avals([(ID,_)|RL]) :-
        name(mr_arre_burro_,L),
        constroi_componentes(ID,LIC),
        constroi_nome(L,LIC,LR), name(Func,LR),
        A =.. [func_avaliacao,ID,Func],
        retract(A),
        retira_func_avals(RL).

/*
Adiciona os pesos das caracteristicas de cada jogador a base de conhecimento.
*/
adiciona_pesos([],_).
adiciona_pesos([(ID,LPesos)|RL],LCars) :-
        name(pesos_,L),
        constroi_componentes(ID,LIC),
        constroi_nome(L,LIC,LR),
        name(Func,LR),
        W =.. [Func,LPesos,LCars],
        assert((W)),
        adiciona_pesos(RL,LCars).

/*
Retira os pesos das caracteristicas de cada jogador da base de conhecimento.
*/
retira_pesos([],_).
retira_pesos([(ID,LPesos)|RL],LCars) :-
        name(pesos_,L),
        constroi_componentes(ID,LIC),
        constroi_nome(L,LIC,LR),
        name(Func,LR),
        W =.. [Func,LPesos,LCars],
        retract((W)),
        retira_pesos(RL,LCars).

/*
Junta o nome do predicado na lista dada como primeiro argumento com a lista de componentes (codigos
ASCII) do valor do ID
de uma funcao de avaliacao dada como segundo argumento e coloca numa lista o resultado dessa juncao.
*/
constroi_nome(L,[],L).
constroi_nome(L,[P|LICR], LR) :-
        D is P + 48,
        inserir_na_cauda(L,D,LInt),
        constroi_nome(LInt,LICR,LR).

/*
Constroi a lista de inteiros (codigos ASCII) componentes de um dado valor (ID de uma funcao de
avaliacao).
*/
constroi_componentes(I, LIC) :-
        constroi_divisor(I,Div),
        constroi_lic([],I,Div,LIC).
```

```
/*
Pega num inteiro (por exemplo: 1210) e devolve os códigos ASCII constituintes dentro de uma lista
(resultado: [49,50,49,48]).
*/
constroi_lic(LIC,0,0,LIC).
constroi_lic(L,I,Div,LIC) :-
        R is I // Div,
        inserir_na_cauda(L,R,LInt),
        S is I mod Div,
        T is Div // 10,
        constroi_lic(LInt,S,T,LIC).

/*
Constroi o divisor de um numero de maneira a conseguir obter o numero inteiro na posicao mais a
esquerda do numero dado como argumento (usa predicado auxiliar constroi_divisor_aux e adicionaRec).
*/
constroi_divisor(X,Div) :-
        X >= 0,
        name(X,L),
        length(L,C),
        D is C - 1,
        constroi_divisor_aux(D,LDiv),
        name(Div,LDiv).

/*
Coloca um 1 no algarismo mais significativo do divisor em construcao
(se ID passado ao constroi_divisor tiver pelo menos dois digitos terao de ser adicionados zeros a direita).
*/
constroi_divisor_aux(0,IR) :-
        name(1,IR).
constroi_divisor_aux(I,IR) :-
        name(1,LI),
        adicionaRec(I,LI,IR).

/*
Coloca zeros a direita do 1 que esta como algarismo mais significativo do divisor em construcao.
*/
adicionaRec(0,LR,LR).
adicionaRec(I,LI,LR) :-
        inserir_na_cauda(LI,48,LInt),
        J is I - 1,
        adicionaRec(J,LInt,LR).

/*
Insere um elemento na cauda de uma dada lista.
*/
inserir_na_cauda(L,Elem,LR) :-
        comprimento(L,V),
        inserir(L,V,Elem,LR).

/*
Insere um elemento num dado indice de uma lista.
*/
inserir(L,0,E,[E|L]).
inserir([X|Xs],I,E,[X|Xss]) :-
        I1 is I - 1,
        inserir(Xs,I1,E,Xss).

/*
```

```
Dado quem esta a pensar e o resultado do jogo, se o primeiro for igual ao segundo devolve 1000;
caso contrario devolve -1000.
*/
valorFinal(Nome,Nome,1000) :- !.
valorFinal(_,_,-1000).

/*
Recebe como argumentos a cor do jogador para o qual vai ser efectuado o calculo, um estado do
tabuleiro,
a lista de pesos, a lista de caracteristicas e devolve o valor do calculo.
Faz o somatorio das multiplicacoes das caracteristicas pelos pesos respectivos (P1 * C1 + P2 * C2+ ... +
Pn * Cn).
*/
soma_pesada(_,_,[],[],0).
soma_pesada(Cor,Tab,[P1|RPs],[C1|RCs],Valor) :-
        soma_pesada(Cor,Tab,RPs,RCs,Rv),
        F1 =.. [C1,Cor,Tab,X],
        call(F1),
        Valor is Rv + X * P1.

%%---------------------------------%%
%%                                 %%
%%   As 12 caracteristicas usadas  %%
%%                                 %%
%%---------------------------------%%

%%-------------------------------------------------------------------------------------------------------------%%
%%   O numero de pecas do adversario que podem ser comidas por cada peca deste jogador (movimentos
possiveis)   %%
%%-------------------------------------------------------------------------------------------------------------%%

/*
Calcula os movimentos possiveis para o jogador que esta a pensar.
Para cada peca, ve para quantas direccoes e que ela pode comer (varia entre 0 e 4) e faz a soma do
numero
de movimentos possiveis de todas as pecas deste jogador.
*/
meus_moves_possiveis(brancas,tab(_,_,LB,LP),V) :-
        findall(J,jogada(tab(brancas,20,LB,LP),_,J),JS),
        length(JS,V).
meus_moves_possiveis(pretas,tab(_,_,LB,LP),V) :-
        findall(J,jogada(tab(pretas,20,LB,LP),_,J),JS),
        length(JS,V).

/*
Calcula os movimentos possiveis para o adversario do jogador que esta a pensar.
Para cada peca, ve para quantas direccoes e que ela pode comer (varia entre 0 e 4) e faz a soma do
numero
de movimentos possiveis de todas as pecas deste jogador.
*/
moves_possiveis_dele(pretas,tab(_,_,LB,LP),V) :-
        findall(J,jogada(tab(brancas,20,LB,LP),_,J),JS),
        length(JS,V).
moves_possiveis_dele(brancas,tab(_,_,LB,LP),V) :-
        findall(J,jogada(tab(pretas,20,LB,LP),_,J),JS),
        length(JS,V).

%%-----------------------------------------------------------------------%%
%%   O numero de pecas que podem comer uma peca do adversario (mobilidade)   %%
%%-----------------------------------------------------------------------%%
```

```
/*
Calcula o numero de pecas que podem comer do jogador que esta a pensar.
*/
minha_mobilidade(brancas,tab(_,_,LB,LP),V) :-
        setof(X,W^Y^Z^jogada(tab(brancas,20,LB,LP),W,come(X,Y,Z)),JS),
        length(JS,V),!.
minha_mobilidade(pretas,tab(_,_,LB,LP),V) :-
        setof(X,W^Y^Z^jogada(tab(pretas,20,LB,LP),W,come(X,Y,Z)),JS),
        length(JS,V),!.
minha_mobilidade(_,_,0).

/*
Calcula o numero de pecas que podem comer do adversario do jogador que esta a pensar.
*/
mobilidade_dele(pretas,tab(_,_,LB,LP),V) :-
        setof(X,W^Y^Z^jogada(tab(brancas,20,LB,LP),W,come(X,Y,Z)),JS),
        length(JS,V).
mobilidade_dele(brancas,tab(_,_,LB,LP),V) :-
        setof(X,W^Y^Z^jogada(tab(pretas,20,LB,LP),W,come(X,Y,Z)),JS),
        length(JS,V).
mobilidade_dele(_,_,0).

%%-------------------------------------------------%%
%%   Numero de pecas nos cantos que podem comer   %%
%%-------------------------------------------------%%

/*
Calcula o numero de pecas nos cantos do tabuleiro do jogador desta cor que podem comer.
*/
meusCantosBonsDaCor(Cor,Tab,N) :-
        findall(C,cantoBomDaCor(Cor,Tab,C),L),
        length(L,N).

/*
Calcula o numero de pecas nos cantos do tabuleiro do jogador adversario do jogador desta cor que podem
comer.
*/
deleCantosBonsDaCor(Cor,Tab,N) :-
        adversario(Cor,CorDele),
        findall(C,cantoBomDaCor(CorDele,Tab,C),L),
        length(L,N).

/*
Dada uma peca de uma dada cor num canto do tabuleiro, ve se ela pode comer alguma peca adversaria.
*/
cantoBomDaCor(Cor,Tab,CantoCor) :-
        canto_cor(Cor,Tab,CantoCor),
        jogada(Tab,_,come(CantoCor,_,_)).

/*
Obtem as pecas brancas que estiverem nos cantos do tabuleiro.
*/
canto_cor(brancas,tab(_,_,Bs,_),CantoCor) :-
        dim(Dim),
        member(CantoCor,[(1,Dim),(Dim,1),(1,1),(Dim,Dim)]),
        member(CantoCor,Bs).

/*
Obtem as pecas pretas que estiverem nos cantos do tabuleiro.
```

```
*/
canto_cor(pretas,tab(_,_,_,Ps),CantoCor) :-
        dim(Dim),
        member(CantoCor,[(1,Dim),(Dim,1),(1,1),(Dim,Dim)]),
        member(CantoCor,Ps).

%%-------------------------------------------------------------------------------%%
%%   Numero de pecas junto as paredes do tabuleiro (excepto cantos) que podem comer   %%
%%-------------------------------------------------------------------------------%%

/*
Calcula o numero de pecas junto as paredes do tabuleiro do jogador desta cor que podem comer.
*/
minhasParedesBoas(Cor,Tab,N) :-
        findall(C,paredeBoaDaCor(Cor,Tab,C),L),
        length(L,N).

/*
Calcula o numero de pecas junto as paredes do tabuleiro do jogador adversario do jogador desta cor que
podem comer.
*/
deleParedesBoas(Cor,Tab,N) :-
        adversario(Cor,CorDele),
        findall(C,paredeBoaDaCor(CorDele,Tab,C),L),
        length(L,N).

/*
Dada uma peca de uma dada cor junto a uma das paredes do tabuleiro,
ve se ela pode comer alguma peca adversaria.
*/
paredeBoaDaCor(Cor,Tab,ParedeCor) :-
        parede_cor(Cor,Tab,ParedeCor),
        jogada(Tab,_,come(ParedeCor,_,_)).

/*
Obtem as pecas brancas que estiverem junto as paredes do tabuleiro.
*/
parede_cor(brancas,tab(_,_,Bs,_),ParedeCor) :-
        member(ParedeCor,[(1,2),(1,3),(1,4),(1,5),(2,1),(3,1),(4,1),(5,1),(6,2),(6,3),(6,4),(6,5),(2,6),(3,6),
(4,6),(5,6)]),
        member(ParedeCor,Bs).

/*
Obtem as pecas pretas que estiverem junto as paredes do tabuleiro.
*/
parede_cor(pretas,tab(_,_,_,Ps),ParedeCor) :-
        member(ParedeCor,[(1,2),(1,3),(1,4),(1,5),(2,1),(3,1),(4,1),(5,1),(6,2),(6,3),(6,4),(6,5),(2,6),(3,6),
(4,6),(5,6)]),
        member(ParedeCor,Ps).

%%-----------------------------------------------%%
%%  Numero de pecas que podem comer duas pecas   %%
%%-----------------------------------------------%%

/*
Calcula o numero de pecas do jogador desta cor que podem comer duas pecas.
*/
minhasPodeComerDuas(Cor,Tab,N) :-
        findall(C,podeComerDuas(Cor,Tab,C),L),
        length(L,N).
```

```
/*
Calcula o numero de pecas do jogador adversario do jogador desta cor que podem comer duas pecas.
*/
delePodeComerDuas(Cor,Tab,N) :-
        adversario(Cor,CorDele),
        findall(C,podeComerDuas(CorDele,Tab,C),L),
        length(L,N).

/*
Dada uma peca de uma dada cor no tabuleiro,
ve se ela pode comer duas pecas adversarias na mesma jogada.
*/
podeComerDuas(Cor,Tab,Peca) :-
        possibilidadeComerDuas(Cor,Tab,Peca),
        jogada(Tab,_,come(Peca,_,2)).

/*
Obtem as pecas brancas no tabuleiro.
*/
possibilidadeComerDuas(brancas,tab(_,_,Bs,_),Peca) :-
        member(Peca,Bs).

/*
Obtem as pecas pretas no tabuleiro.
*/
possibilidadeComerDuas(pretas,tab(_,_,_,Ps),Peca) :-
        member(Peca,Ps).


%%-------------------------------------%%
%%   O numero de pecas no tabuleiro   %%
%%-------------------------------------%%

/*
Calcula o numero de pecas do jogador desta cor no tabuleiro.
*/
meuNumeroPecas(Cor,Tab,N) :-
        numeroPecas(Cor,Tab,N).

/*
Calcula o numero de pecas do adversario do jogador desta cor no tabuleiro.
*/
deleNumeroPecas(Cor,Tab,N) :-
        adversario(Cor,CorDele),
        numeroPecas(CorDele,Tab,N).

/*
Calcula o numero de pecas brancas no tabuleiro.
*/
numeroPecas(brancas,tab(_,_,Bs,_),N) :-
        length(Bs,N).

/*
Calcula o numero de pecas pretas no tabuleiro.
*/
numeroPecas(pretas,tab(_,_,_,Ps),N) :-
        length(Ps,N).

%%-------------------------------------%%
```

```
%%   Características acrescentadas   %%
%%-----------------------------------%%


%%-------------------------------------------------------------%%
%%    Numero de pecas nos cantos (não podem ser comidas)   %%
%%-------------------------------------------------------------%%

/*
Calcula o numero de pecas nos cantos do tabuleiro do jogador desta cor.
*/
meusCantosDaCor(Cor,Tab,N) :-
        findall(C,cantoCor(Cor,Tab,C),L),
        length(L,N).

/*
Calcula o numero de pecas nos cantos do tabuleiro do jogador adversario do jogador desta cor.
*/
deleCantosDaCor(Cor,Tab,N) :-
        adversario(Cor,CorDele),
        findall(C,cantoCor(CorDele,Tab,C),L),
        length(L,N).

/*
Obtem as pecas brancas que estiverem nos cantos do tabuleiro.
*/
cantoCor(brancas,tab(_,_,Bs,_),CantoCor) :-
        dim(Dim),
        member(CantoCor,[(1,Dim),(Dim,1),(1,1),(Dim,Dim)]),
        member(CantoCor,Bs).

/*
Obtem as pecas pretas que estiverem nos cantos do tabuleiro.
*/
cantoCor(pretas,tab(_,_,_,Ps),CantoCor) :-
        dim(Dim),
        member(CantoCor,[(1,Dim),(Dim,1),(1,1),(Dim,Dim)]),
        member(CantoCor,Ps).

%%-------------------------------------------------------------------%%
%%   Numero de pecas junto as paredes do tabuleiro (excepto cantos)   %%
%%-------------------------------------------------------------------%%

/*
Calcula o numero de pecas junto as paredes do tabuleiro do jogador desta cor.
*/
minhasParedes(Cor,Tab,N) :-
        findall(C,paredeDaCor(Cor,Tab,C),L),
        length(L,N).

/*
Calcula o numero de pecas junto as paredes do tabuleiro do jogador adversario do jogador desta cor.
*/
deleParedes(Cor,Tab,N) :-
        adversario(Cor,CorDele),
        findall(C,paredeDaCor(CorDele,Tab,C),L),
        length(L,N).

/*
Dada uma peca de uma dada cor junto a uma das paredes do tabuleiro.
```

```
*/
paredeDaCor(Cor,Tab,ParedeCor) :-
        paredeCor(Cor,Tab,ParedeCor).

/*
Obtem as pecas brancas que estiverem junto as paredes do tabuleiro.
*/
paredeCor(brancas,tab(_,_,Bs,_),ParedeCor) :-
        member(ParedeCor,[(1,2),(1,3),(1,4),(1,5),(2,1),(3,1),(4,1),(5,1),(6,2),(6,3),(6,4),(6,5),(2,6),(3,6),
(4,6),(5,6)]),
        member(ParedeCor,Bs).

/*
Obtem as pecas pretas que estiverem junto as paredes do tabuleiro.
*/
paredeCor(pretas,tab(_,_,_,Ps),ParedeCor) :-
        member(ParedeCor,[(1,2),(1,3),(1,4),(1,5),(2,1),(3,1),(4,1),(5,1),(6,2),(6,3),(6,4),(6,5),(2,6),(3,6),
(4,6),(5,6)]),
        member(ParedeCor,Ps).
```

osjogadoresteste.pl

```
/*
Fornece um gerador de numeros aleatorios.
*/
:- use_module(library(random)).

/*
Permite fazer minicamps entre dois jogadores.

exemplo de invocacao:
iniciar_teste([(1,[2,-2.2,2,-2.2]),(2,[0.032,0.677,0.226,0.161,-0.419,-0.161,0.032,-
0.161])],[[meus_moves_possiveis,moves_possiveis_dele,minha_mobilidade,mobilidade_dele],[minha_m
obilidade,mobilidade_dele,minhasPodeComerDuas,delePodeComerDuas,meusCantosDaCor,deleCantosD
aCor,minhasParedes,deleParedes]],LR).
*/
iniciar_teste(L,LCars,LR) :-
        adiciona_funcoes(L),
        adiciona_pesos_teste(L,LCars),
        adiciona_func_avals(L),
        cria_populacao_inicial(L),
        jogar_campeonato(L),
        organiza_jogadores(LR),
        limpar_pontos,
        retira_func_avals(L),
        retira_pesos_teste(L,LCars),
        retira_funcoes(L).

/*
Adiciona os pesos das caracteristicas de cada jogador a base de conhecimento.
*/
adiciona_pesos_teste([],_).
adiciona_pesos_teste([(ID,LPesos)|RL],[PCar|LCars]) :-
        name(pesos_,L),
        constroi_componentes(ID,LIC),
        constroi_nome(L,LIC,LR),
        name(Func,LR),
        W =.. [Func,LPesos,PCar],
        assert((W)),
        adiciona_pesos_teste(RL,LCars).

/*
Retira os pesos das caracteristicas de cada jogador da base de conhecimento.
*/
retira_pesos_teste([],_).
retira_pesos_teste([(ID,LPesos)|RL],[PCar|LCars]) :-
        name(pesos_,L),
        constroi_componentes(ID,LIC),
        constroi_nome(L,LIC,LR),
        name(Func,LR),
        W =.. [Func,LPesos,PCar],
        retract((W)),
        retira_pesos_teste(RL,LCars).
```

server.pl

```
:- module(evaluate,[main/0,arranque/0,inicio/3]).
:- use_module(library(prologbeans)).
:- use_module(library(codesio), [read_from_codes/2]).

/*
Faz o consult ao ficheiro ojogo.pl
(que realiza o consult aos restantes ficheiros do projecto)
e disponibiliza o motor de jogo.
*/
arranque :-
        consult(ojogo),
        tell('../caracteristicas.txt'),
        write([minha_mobilidade,mobilidade_dele,meus_moves_possiveis,moves_possiveis_dele,
        minhasPodeComerDuas,delePodeComerDuas,meuNumeroPecas,deleNumeroPecas,
        meusCantosDaCor,deleCantosDaCor,minhasParedes,deleParedes]),
        told.

/*
Arranca o prolog.
*/
:- arranque.

%% Register acceptable queries and start the server (using default port)
main :-
        register_query(my_predicate(LIDP,LCars,LR), inicio(LIDP,LCars,LR)),
        start.

/*
Converte a string recebida do java numa lista que o prolog possa manusear internamente
e chama o predicado que vai fazer a seleccao do melhor jogador de konane,
devolve a lista de jogadores identificados pelos seus id's e ordenada por pontos.
*/
inicio(LIDP,LCars,LR) :-
        read_from_codes(LIDP,LI),
        read_from_codes(LCars,LC),
        iniciar(LI,LC,LR).
```

Java

GeraPopulacao.java

```
import java.io.IOException;
import java.text.NumberFormat;
import java.util.LinkedList;
import java.util.Random;

import com.micropraxis.gajit.FixView;
import com.micropraxis.gajit.View;

/**
 * Classe que gera a populacao, constroi os pesos de cada cromossoma e efectua a interaccao com o
prolog.
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class GeraPopulacao {
	/**
	 * As caracteristicas.
	 */
	private String sCars;
	/**
	 * A populacao.
	 */
	private Populacao jogadores;
	/**
	 * A vista.
	 */
	private View vista;
	/**
	 * A interaccao com o prolog.
	 */
	private PrologInteraction interacao;
	/**
	 * O numero de cromossomas desta populacao.
	 */
	private int numCromossomas;
	/**
	 * O numero de genes de cada cromossoma desta populacao.
	 */
	private int numGenes;
	/**
	 * O tamanho de cada gene.
	 */
	private int tamGene;

	/**
	 * Constroi uma populacao.
	 * @param numCromossomas O numero de cromossomas desta populacao.
	 */
	public GeraPopulacao (int numCromossomas, String[] caracteristicas) {
```

```
                this.numCromossomas = numCromossomas;
                numGenes = caracteristicas.length;
                tamGene = 5;
                Random gen = new Random();
                int pontoCorte = gen.nextInt(numGenes*tamGene);
                double taxaMutacao = 0.005;
                jogadores = new Populacao (numCromossomas, tamGene, numGenes, pontoCorte,
taxaMutacao);
                vista = new FixView(tamGene, -1.0, 2.0);
                try {
                        interacao = new PrologInteraction(numCromossomas, numGenes);
                } catch (IOException e) {
                        e.printStackTrace();
                }
                sCars = interacao.empacotarCars(caracteristicas);
        }

        /**
         * Constroi os pesos, envia-os ao prolog, recebe a lista dos ids que
         * identificam cada cromossoma e respectivos fitness's, cria uma nova
         * geracao e retorna a string que contem o id e a respectiva lista de pesos do melhor cromossoma
da geracao antiga.
         *
         * @return A string contendo o id e os pesos do melhor jogador.
         */
        public String interage () {
                NumberFormat nf = NumberFormat.getInstance();
                nf.setMaximumFractionDigits(3);
                nf.setMinimumFractionDigits(3);
                LinkedList<Double> pesos = constroiPesos();
                String sPesos = interacao.empacotarDados(pesos,jogadores);
                String sFits = interacao.callProlog(sPesos, sCars);
                LinkedList<IdFit> idFits = interacao.desempacotarDados(sFits);
                int fit = idFits.get(0).getFitness();
                int id = idFits.get(0).getId();
                for (IdFit idf : idFits)
                        if (fit < idf.getFitness()){
                                fit = idf.getFitness();
                                id = idf.getId();
                        }
                int index = jogadores.getIndexFromId(id);
                int comeco = index*numGenes;
                StringBuilder sb = new StringBuilder("");
                sb.append(id + ": [");
                for(int i = comeco; i < comeco+numGenes; i++) {
                        sb.append(nf.format(pesos.get(i)));
                        if(i%3 == 0)
                                sb.append("\n");
                        if (i != comeco+numGenes-1)
                                sb.append(";");
                }
                sb.append("]");
                jogadores.newGeneration(idFits);
                return sb.toString();
        }

        /**
         * Devolve uma lista de pesos.
         *
         * @return A lista de pesos.
```

```
		*/
		public LinkedList<Double> constroiPesos () {
			LinkedList<Double> pesos = new LinkedList<Double>();
			for(int i = 0; i < numCromossomas; i++) {
				for(int j = 0; j < numGenes; j++) {

		pesos.addFirst((Double)vista.getGene(jogadores.getChromItem(i).getChrom(), j));
				}
			}
			return pesos;
		}
}
```

IdFit.java

```
/**
 * A classe que associa um id a um fitness.
 *
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class IdFit {
  /**
   * O fitness.
   */
  private int fitness;
  /**
   * O id.
   */
  private int id;

  /**
   * Constroi um IdFit.
   * @param id O id.
   * @param fitness O fitness.
   */
  public IdFit(int id, int fitness) {
    this.fitness = fitness;
    this.id = id;
  }

  /**
   * Devolve o id deste IdFit.
   * @return O id deste IdFit.
   */
  public int getId(){
    return id;
  }

  /**
   * Devolve o fitness deste IdFit.
   * @return O fitness deste IdFit.
   */
  public int getFitness(){
    return fitness;
  }

  /**
   * Devolve uma representacao textual deste IdFit.
   * @return A string com a representacao textual.
   */
  public String toString(){
    String str = fitness + ":" + id;
    return str;
  }
}
```

InterF.java

```
import java.awt.Rectangle;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
import javax.swing.SwingWorker;
import javax.swing.JList;

public class InterF extends JFrame {

  private static final long serialVersionUID = 1L;
  private JPanel jContentPane = null;
  private JTextField jTextFieldTamPop = null;
  private JTextField jTextFieldNumGeracoes = null;
  private JLabel jLabelTamPop = null;
  private JLabel jLabelNumGeracoes = null;
  private JButton jButtonEncontrarMelhorJogador = null;
  private JButton jButtonLimparDados = null;
  private JButton jButtonSair = null;
  private JButton jButtonAcercaDe = null;
  private JTextArea jTextAreaOMelhorJogador = null;
  private JLabel jLabelOMelhorJogador = null;
  private JScrollPane jScrollPane = null;
  private JScrollPane jScrollPaneListCars = null;
  private JList jListListaCars = null;
  private JLabel jLabelListaCars = null;
  /**
   * This method initializes jTextFieldTamPop
   *
   * @return javax.swing.JTextField
   */
  private JTextField getJTextFieldTamPop () {
    if (jTextFieldTamPop == null) {
      jTextFieldTamPop = new JTextField();
      jTextFieldTamPop.setBounds(new Rectangle(151, 18, 100, 20));
    }
    return jTextFieldTamPop;
  }

  /**
   * This method initializes jTextFieldNumGeracoes
   *
   * @return javax.swing.JTextField
   */
  private JTextField getJTextFieldNumGeracoes () {
    if (jTextFieldNumGeracoes == null) {
      jTextFieldNumGeracoes = new JTextField();
      jTextFieldNumGeracoes.setBounds(new Rectangle(152, 44, 100, 20));
    }
    return jTextFieldNumGeracoes;
```

```
    }

    /**
     * This method initializes jButtonEncontrarMelhorJogador
     *
     * @return javax.swing.JButton
     */
    private JButton getJButtonEncontrarMelhorJogador () {
      if (jButtonEncontrarMelhorJogador == null) {
        jButtonEncontrarMelhorJogador = new JButton();
        jButtonEncontrarMelhorJogador.setBounds(new Rectangle(13, 326, 179, 21));
        jButtonEncontrarMelhorJogador.setText("Encontrar melhor jogador");
        jButtonEncontrarMelhorJogador.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed (java.awt.event.ActionEvent e) {
            jButtonEncontrarMelhorJogador.setEnabled(false);
            String tamPopStr = jTextFieldTamPop.getText();
            String numGeracoesStr = jTextFieldNumGeracoes.getText();
            try{
              Integer.parseInt(tamPopStr);
              Integer.parseInt(numGeracoesStr);
            } catch (NumberFormatException nfe) {
              JOptionPane.showMessageDialog(jContentPane,
              "Introduza apenas valores numéricos");
              return;
            }
            if(tamPopStr.equals("") || numGeracoesStr.equals("")){
              JOptionPane.showMessageDialog(jContentPane,
              "Introduza os valores em falta");
              return;
            }
            if (Integer.parseInt(tamPopStr) <= 1){
              JOptionPane.showMessageDialog(jContentPane,
              "Introduza um numero de jogadores maior que 1.");
              return;
            }
            if (Integer.parseInt(numGeracoesStr) <= 0){
              JOptionPane.showMessageDialog(jContentPane,
              "Introduza um numero de geracoes maior que 0.");
              return;
            }
            int tamPop = Integer.parseInt(tamPopStr);
            final int numGeracoes = Integer.parseInt(numGeracoesStr);
            Object[] caractObjs = jListListaCars.getSelectedValues();
            String[] caracteristicas = new String[caractObjs.length];
            for (int i = 0; i < caractObjs.length; i++)
              caracteristicas[i] = (String)caractObjs[i];
            final GeraPopulacao gp;
            if (caracteristicas.length == 0)
              gp = new GeraPopulacao(tamPop, PrologInteraction.leCaracteristicas());
            else
              gp = new GeraPopulacao(tamPop, caracteristicas);
            SwingWorker<Void,Void> worker = new SwingWorker<Void, Void>() {
              @Override
              public Void doInBackground() {
                String melhorJogador = "";
                int i = 0;
                while (i < numGeracoes){
                  melhorJogador = gp.interage();
                  jTextAreaOMelhorJogador.append("Geracao " + (i+1) + ":\nO vencedor foi:\n" +
melhorJogador + "\n\n");
```

```
            i++;
           }
           jButtonEncontrarMelhorJogador.setEnabled(true);
           jTextAreaOMelhorJogador.append("O melhor jogador ao fim de " + numGeracoes +
              ((numGeracoes == 1) ? " geração" :  " gerações") + " é o jogador:\n" + melhorJogador);
           return null;
          }

        };
        worker.execute();
      }
    });
  }
  return jButtonEncontrarMelhorJogador;
}

/**
 * This method initializes jButtonLimparDados
 *
 * @return javax.swing.JButton
 */
private JButton getJButtonLimparDados () {
  if (jButtonLimparDados == null) {
    jButtonLimparDados = new JButton();
    jButtonLimparDados.setBounds(new Rectangle(13, 355, 144, 24));
    jButtonLimparDados.setText("Reset dos dados");
    jButtonLimparDados.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed (java.awt.event.ActionEvent e) {
        jTextFieldTamPop.setText("");
        jTextFieldNumGeracoes.setText("");
        jTextFieldTamPop.requestFocus();
      }
    });
  }
  return jButtonLimparDados;
}

/**
 * This method initializes jButtonSair
 *
 * @return javax.swing.JButton
 */
private JButton getJButtonSair () {
  if (jButtonSair == null) {
    jButtonSair = new JButton();
    jButtonSair.setBounds(new Rectangle(14, 422, 75, 22));
    jButtonSair.setText("Sair");
    jButtonSair.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed (java.awt.event.ActionEvent e) {
        System.exit(0);
      }
    });
  }
  return jButtonSair;
}

/**
 * This method initializes jButtonAcercaDe
 *
 * @return javax.swing.JButton
```

```
 */
 private JButton getJButtonAcercaDe () {
  if (jButtonAcercaDe == null) {
    jButtonAcercaDe = new JButton();
    jButtonAcercaDe.setBounds(new Rectangle(13, 393, 100, 20));
    jButtonAcercaDe.setText("Acerca...");
    jButtonAcercaDe.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed (java.awt.event.ActionEvent e) {
        String message = "Programa desenvolvido pelo grupo IIA018 (Celso, Henrique e Sérgio), no ano
lectivo 2008/2009 :)";
        JOptionPane.showMessageDialog(jContentPane, message, "Acerca de geneticos",
JOptionPane.INFORMATION_MESSAGE);
      }
    });
  }
  return jButtonAcercaDe;
 }

 /**
  * This method initializes jTextAreaOMelhorJogador
  *
  * @return javax.swing.JTextArea
  */
 private JTextArea getJTextAreaOMelhorJogador () {
  if (jTextAreaOMelhorJogador == null) {
    jTextAreaOMelhorJogador = new JTextArea();
    jTextAreaOMelhorJogador.setEditable(false);
  }
  return jTextAreaOMelhorJogador;
 }

 /**
  * This method initializes jScrollPane
  *
  * @return javax.swing.JScrollPane
  */
 private JScrollPane getJScrollPane () {
  if (jScrollPane == null) {
    jScrollPane = new JScrollPane();
    jScrollPane.setBounds(new Rectangle(399, 134, 164, 289));
    jScrollPane.setViewportView(getJTextAreaOMelhorJogador());
  }
  return jScrollPane;
 }

 /**
  * This method initializes jScrollPaneListCars
  *
  * @return javax.swing.JScrollPane
  */
 private JScrollPane getJScrollPaneListCars () {
  if (jScrollPaneListCars == null) {
    jScrollPaneListCars = new JScrollPane();
    jScrollPaneListCars.setBounds(new Rectangle(102, 174, 137, 151));
    jScrollPaneListCars.setViewportView(getJListListaCars());
  }
  return jScrollPaneListCars;
 }

 /**
```

```
 * This method initializes jListListaCars
 *
 * @return javax.swing.JList
 */
private JList getJListListaCars () {
  if (jListListaCars == null) {
    String[] arr = PrologInteraction.leCaracteristicas();
    jListListaCars = new JList(arr);
  }
  return jListListaCars;
}

/**
 * @param args
 */
public static void main (String[] args) {
  PrologInteraction.leCaracteristicas();
  SwingUtilities.invokeLater(new Runnable() {
    public void run () {
      InterF thisClass = new InterF();
      thisClass.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      thisClass.setVisible(true);
    }
  });
}

/**
 * This is the default constructor
 */
public InterF () {
  super();
  initialize();
}

/**
 * This method initializes this
 *
 * @return void
 */
private void initialize () {
  this.setSize(600, 600);
  this.setContentPane(getJContentPane());
  this.setTitle("JFrame");
}

/**
 * This method initializes jContentPane
 *
 * @return javax.swing.JPanel
 */
private JPanel getJContentPane () {
  if (jContentPane == null) {
    jLabelListaCars = new JLabel();
    jLabelListaCars.setBounds(new Rectangle(10, 154, 161, 16));
    jLabelListaCars.setText("Características pretendidas");
    jLabelOMelhorJogador = new JLabel();
    jLabelOMelhorJogador.setBounds(new Rectangle(287, 141, 101, 16));
    jLabelOMelhorJogador.setText("O melhor jogador");
    jLabelNumGeracoes = new JLabel();
    jLabelNumGeracoes.setBounds(new Rectangle(13, 45, 124, 16));
```

```
        jLabelNumGeracoes.setText("Número de gerações");
        jLabelTamPop = new JLabel();
        jLabelTamPop.setBounds(new Rectangle(8, 20, 138, 16));
        jLabelTamPop.setText("Tamanho da população");
        jContentPane = new JPanel();
        jContentPane.setLayout(null);
        jContentPane.add(getJTextFieldTamPop(), null);
        jContentPane.add(getJTextFieldNumGeracoes(), null);
        jContentPane.add(jLabelTamPop, null);
        jContentPane.add(jLabelNumGeracoes, null);
        jContentPane.add(jLabelOMelhorJogador, null);
        jContentPane.add(getJButtonEncontrarMelhorJogador(), null);
        jContentPane.add(getJButtonLimparDados(), null);
        jContentPane.add(getJButtonSair(), null);
        jContentPane.add(getJButtonAcercaDe(), null);
        jContentPane.add(getJScrollPane(), null);
        jContentPane.add(getJScrollPaneListCars(), null);
        jContentPane.add(jLabelListaCars, null);
      }
      return jContentPane;
    }

}
```

InterText.java

```
import java.util.Scanner;


public class InterText {

  /**
   * @param args
   */
  public static void main (String[] args) {
    Scanner le = new Scanner(System.in);
    System.out.println("Insira num populacao");
    int numPop = le.nextInt();
    System.out.println("Insira num geracoes");
    int numGer = le.nextInt();
    GeraPopulacao ger = new GeraPopulacao(numPop, PrologInteraction.leCaracteristicas());
    int i = 0;
    String melhorJogador = "";
    while (i < numGer){
      melhorJogador = ger.interage();
      System.out.println("Geracao " + (i+1) + ":\nO vencedor foi:\n" + melhorJogador + "\n\n");
      i++;
    }

    System.out.println("O melhor jogador ao fim de " + numGer +
        ((numGer == 1) ? " geração" :  " gerações") + " é o jogador:\n" + melhorJogador);

  }
}
```

Populacao.java

```
import java.util.LinkedList;
import java.util.Random;

import com.micropraxis.gajit.Chrom;
import com.micropraxis.gajit.ChromItem;
import com.micropraxis.gajit.MutOp;

/**
 * Classe que representa uma populacao.
 *
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class Populacao {
	/**
	 * A lista de ChromItems
	 */
	private LinkedList<ChromItem> chromItems;
	/**
	 * O numero de genes desta populacao.
	 */
	private int numGenes;
	/**
	 * O tamanho de um gene desta populacao.
	 */
	private int tamanhoGene;
	/**
	 * O numero de cromossomas desta populacao.
	 */
	private int numCromossomas;
	/**
	 * O operador de mutacao.
	 */
	private MutOp mutador;
	/**
	 * O operador de cruzamento.
	 */
	private XOverCutOp cruzador;

	/**
	 * Constroi uma populacao aleatoria de um dado numero de cromossomas com tamanho dos
genes e numero de genes definido.
	 * Os fitness's em cada ChromItem são inicializados a 0.0.
	 * @param numCromossomas O numero de cromossomas.
	 * @param tamanhoGene O tamanho dos genes.
	 * @param numGenes O numero de genes.
	 */
	public Populacao (int numCromossomas, int tamanhoGene, int numGenes) {
		chromItems = new LinkedList<ChromItem>();
		this.numGenes = numGenes;
		this.tamanhoGene = tamanhoGene;
		this.numCromossomas = numCromossomas;
		for (int i = 0; i < numCromossomas; i++){
			Chrom c = new Chrom(this.tamanhoGene*this.numGenes);
```

```
                    ChromItem ci = new ChromItem(0.0, c);
                    chromItems.addFirst(ci);
                }
                cruzador = new XOverCutOp(((tamanhoGene*numGenes)/2)-1, numGenes,
tamanhoGene);
                mutador = new MutOp(0.005);
        }

        /**
         * Constroi uma populacao aleatoria de um dado numero de cromossomas com tamanho dos
genes, numero de genes e ponto de corte definidos.
         * A taxa de mutacao e' por defeito 0.005.
         * @param numCromossomas O numero de cromossomas.
         * @param tamanhoGene O tamanho dos genes.
         * @param numGenes O numero de genes.
         * @param pontoDeCorte O ponto de corte.
         */
        public Populacao (int numCromossomas, int tamanhoGene, int numGenes, int pontoDeCorte) {
                this(numCromossomas, tamanhoGene, numGenes);
                cruzador = new XOverCutOp(pontoDeCorte, numGenes, tamanhoGene);
        }

        /**
         * Constroi uma populacao aleatoria de um dado numero de cromossomas com tamanho dos
genes, numero de genes e taxa de mutacao definidos.
         * O ponto de corte e' por defeito (tamanhoGene*numGenes)/2)-1) em que tamanhoGene e' 4 e
numGenes e' 12.
         * @param numCromossomas O numero de cromossomas.
         * @param tamanhoGene O tamanho dos genes.
         * @param numGenes O numero de genes.
         * @param taxaMutacao A taxa de mutacao.
         */
        public Populacao (int numCromossomas, int tamanhoGene, int numGenes, double taxaMutacao)
{
                this(numCromossomas, tamanhoGene, numGenes);
                mutador = new MutOp(taxaMutacao);
        }

        /**
         * Constroi uma populacao aleatoria de um dado numero de cromossomas com tamanho dos
genes, numero de genes, ponto de corte e taxa de mutacao definidos.
         * @param numCromossomas O numero de cromossomas.
         * @param tamanhoGene O tamanho dos genes.
         * @param numGenes O numero de genes.
         * @param pontoDeCorte O ponto de corte.
         * @param taxaMutacao A taxa de mutacao.
         */
        public Populacao (int numCromossomas, int tamanhoGene, int numGenes, int pontoDeCorte,
double taxaMutacao) {
                this(numCromossomas, tamanhoGene, numGenes);
                cruzador = new XOverCutOp(pontoDeCorte, numGenes, tamanhoGene);
                mutador = new MutOp(taxaMutacao);
        }

        /**
         * Actualiza em cada ChromItem o fitness respectivo utilizando para isso a lista de fitness's dada
como parametro
         * e efectua os cruzamentos necessários para obter os individuos da proxima geracao.
         * @param fit A lista de IdFits .
         */
```

```
	public void newGeneration(LinkedList<IdFit> fit){
		LinkedList<ChromItem> temp = new LinkedList<ChromItem>();
		assignFitsToChromItems(fit);
		for(int i = 0; i < numCromossomas; i++){
			Chrom pai = selectRandomChrom();
			Chrom mae = selectRandomChrom();
			Chrom filho = cruzador.apply(pai, mae);
			mutador.apply(filho);
			ChromItem ci = new ChromItem(0.0, filho);
			temp.addFirst(ci);
		}
		chromItems = temp;
	}

	/**
	 * Selecciona e devolve um cromossoma aleatorio segundo o metodo de seleccao denominado roleta.
	 * Este metodo funciona do seguinte modo:
	 * Faz a soma de todos os fitness's para achar o valor maximo da roleta. De seguida, gera um numero aleatorio entre 0 e a soma - 1.
	 * Depois, percorre todos os ChromItems ate encontrar um, cujo fitness associado seja maior que o numero aleatorio gerado e
	 * devolve o cromossoma correspondente.
	 * Quanto maior o fitness de um dado cromossoma, maior a probabilidade de ele ser escolhido.
	 * @return O cromossoma.
	 */
	private Chrom selectRandomChrom () {
		double soma = somatorioFits();
		Random gerador = new Random();
		int aleatorio = gerador.nextInt((int)soma);
		double verificador = 0.0;
		for (ChromItem ci : chromItems){
			verificador += ci.getDoubleFitness();
			if (verificador >= aleatorio)
				return ci.getChrom();
		}
		return null;
	}

	/**
	 * Devolve o somatorio de todos os fitness's presentes na lista de ChromItems.
	 * @return O somatorio dos fitness's.
	 */
	private double somatorioFits () {
		double soma = 0.0;
		for (ChromItem ci : chromItems){
			soma += ci.getDoubleFitness();
		}
		return soma;
	}

	/**
	 * Actualiza os ids na lista de chromItems atraves dos ids actualizados que vem na lista de IdFits.
	 * @param fit A lista de IdFits.
	 */
	private void assignFitsToChromItems(LinkedList<IdFit> fit){
		for (ChromItem ci : chromItems){
			for (IdFit f : fit){
				if (f.getId() == getIdFromChrom(ci.getChrom()))
					ci.setFitness(f.getFitness());
```

```
                        }
                }
        }

        /**
         * Obtem o id de um dado cromossoma.
         * @param c O cromossoma.
         * @return O id que identifica este cromossoma.
         */
        public int getIdFromChrom(Chrom c){
                String[] arr = c.toString().split(":");
                return Integer.parseInt(arr[0].substring(1));
        }

        /**
         * Obtem o indice relativo ao cromossoma com o id dado como parametro.
         * @param id O id de um cromossoma.
         * @return O indice correspondente a este id.
         */
        public int getIndexFromId (int id) {
                for (int i = 0; i < chromItems.size(); i++)
                        if (getIdFromChrom(chromItems.get(i).getChrom()) == id)
                                return i;
                return -1;
        }

        /**
         * Obtem o chromItem no indice dado como parametro.
         * @param index O indice do chromItem.
         * @return O chromItem num dado indice.
         */
        public ChromItem getChromItem (int index) {
                return chromItems.get(index);
        }
}
```

PrologInteraction.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.LinkedList;
import java.util.Scanner;

import se.sics.prologbeans.Bindings;
import se.sics.prologbeans.PBTerm;
import se.sics.prologbeans.PrologSession;
import se.sics.prologbeans.QueryAnswer;

/**
 * Realiza a interaccao entre o java e o prolog.
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class PrologInteraction {
	/**
	 * A sessao que servira para fazer a interaccao com o prolog.
	 */
	private PrologSession session = new PrologSession();
	/**
	 * O numero de cromossomas.
	 */
	private int numCromossomas;
	/**
	 * O numero de genes.
	 */
	private int numGenes;
	/**
	 * Liga o java ao prolog ate o java receber de volta uma resposta.
	 * @param numCromossomas O numero de cromossomas.
	 * @param numGenes O numero de genes.
	 * @throws java.io.IOException
	 */
	public PrologInteraction (int numCromossomas, int numGenes) throws java.io.IOException {
		this.numCromossomas = numCromossomas;
		this.numGenes = numGenes;
		session.setTimeout(0);
		session.connect();
	}

	/**
	 * Devolve a string que sera passada ao prolog e que contem uma lista de tuplos
	 * constituida por id e lista de pesos respectiva ao cromossoma com esse id.
	 * @param pesos A lista de pesos relativa a esta populacao.
	 * @param jogadores A populacao.
	 * @return A string com a lista de ids e pesos respectivos que sera passada ao prolog.
	 */
	public String empacotarDados (LinkedList<Double> pesos, Populacao jogadores) {
		StringBuilder sb = new StringBuilder("");
		sb.append("[(");
		for (int i = 0; i < numCromossomas; i++) {

	sb.append(jogadores.getIdFromChrom(jogadores.getChromItem(i).getChrom()) + ",[");
```

```
                        for(int j = 0; j < numGenes; j++) {
                                sb.append(pesos.get(j));
                                if(j != (numGenes-1))
                                        sb.append(",");
                        }
                        if(i != (numCromossomas-1))
                                sb.append("]),(");
                }
                sb.append("])]");
                return sb.toString();
        }

        /**
         * Devolve uma string que será passada ao prolog e que contem uma lista constituida pelas
caracteristicas.
         * @param caracteristicas O array de strings a empacotar.
         * @return A string contendo a lista de caracteristicas.
         */
        public String empacotarCars(String[] caracteristicas){
                StringBuilder sb = new StringBuilder("");
                sb.append("[");
                for (int i = 0; i < caracteristicas.length; i++){
                        sb.append(caracteristicas[i]);
                        if (i != caracteristicas.length-1)
                                sb.append(",");
                }
                sb.append("]");
                return sb.toString();
        }
        /**
         * Coloca os pares dos ids de jogadores e fitness's respectivos numa lista de IdFits.
         * @param s A string que vem do prolog que contem a lista ordenada por fitness's.
         * @return Devolve uma lista de IdFits que sao tuplos (id,fitness).
         */
        public LinkedList<IdFit> desempacotarDados (String s) {
                String[] tuplos = s.split("\\),\\(");
                tuplos[0] = tuplos[0].substring(2);
                tuplos[numCromossomas-1] = tuplos[numCromossomas-1].substring(0,
tuplos[numCromossomas-1].length()-2);
                LinkedList<IdFit> idFits = new LinkedList<IdFit>();
                for (int i = 0; i < tuplos.length; i++){
                        String[] arr = tuplos[i].split(",");
                        idFits.add(new IdFit(Integer.parseInt(arr[0]), Integer.parseInt(arr[1])));
                }
                return idFits;
        }

        /**
         * Passa ao prolog a string com os ids e os pesos respectivos,
         * recebe a lista de ids com os fitness's respectivos.
         * @param s A string de ids e pesos respectivos que vai ser passada ao prolog.
         * @return A string com os ids e os fitness's ordenada por fitness's.
         */
        public String callProlog(String sPesos, String sCars) {
                try {
                        Bindings bindings = new Bindings();
                        bindings.bind("Pesos", sPesos + ".");
                        bindings.bind("Cars", sCars + ".");
                        QueryAnswer answer = session.executeQuery("my_predicate(Pesos,Cars,R)",
bindings);
```

```
                        PBTerm result = answer.getValue("R");
                        if (result != null) {
                                String str = result.toString();//Este metodo toString() coloca uma
virgula a mais antes de cada tuplo da string.
                                StringBuilder sb = new StringBuilder(str.replaceAll(",,", ","));//Tira
todas as virgulas a mais excepto a que esta antes do primeiro parentesis curvo.
                                return sb.deleteCharAt(1).toString();//Apaga a primeira virgula da
lista.
                        } else {
                                return "Error: " + answer.getError() + '\n';
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                        return("Error when querying Prolog Server: " +
                                        e.getMessage() + '\n');
                }
        }

        public static String[] leCaracteristicas(){
                Scanner leitor = null;
                try {
                        leitor = new Scanner(new File("../caracteristicas.txt"));
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }
                String linha = leitor.nextLine();
                String[] arrCars = linha.split(",");
                arrCars[0] = arrCars[0].substring(1);
                int comprimentoUltimaCar = arrCars[arrCars.length-1].length();
                arrCars[arrCars.length-1] = arrCars[arrCars.length-
1].substring(0,comprimentoUltimaCar-1);
                return arrCars;
        }
}
```

XOverCutOp.java

```
import com.micropraxis.util.ExtendedBitSet;
import com.micropraxis.gajit.Chrom;
import com.micropraxis.gajit.GenOp;

/**
 * Classe que representa o operador de recombinacao genetica.
 * @author Grupo018
 * Celso Sousa Nº32441
 * Sergio das Neves Nº32536
 * Henrique Cunha Nº33321
 */
public class XOverCutOp extends GenOp {
	/**
	 * O ponto de corte.
	 */
	private int pontoDeCorte;

	/**
	 * O número de genes (caracteristicas).
	 */
	private int numGenes;
	/**
	 * O tamanho de acda gene.
	 */
	private int tamGene;

	/**
	 * Cria o operador de cruzamento com um dado ponto de corte.
	 * @param pontoDeCorte O ponto de corte.
	 */
	public XOverCutOp (int pontoDeCorte, int numGenes, int tamGene) {
		super(true);
		this.pontoDeCorte = pontoDeCorte;
		this.numGenes = numGenes;
		this.tamGene = tamGene;
	}

	/**
	 * Aplica o cruzamento sobre dois cromossomas gerando um cromossoma filho
	 * (requer que os cromossomas tenham o mesmo tamanho).
	 * @param c1 O cromossoma pai.
	 * @param c2 O cromossoma mae.
	 * @return O cromossoma filho.
	 */
	public Chrom apply(Chrom c1, Chrom c2){
		String[] arr1 = c1.toString().split(":");
		String cadeia1 = arr1[1].substring(0, arr1[1].length()-1);
		String[] arr2 = c2.toString().split(":");
		String cadeia2 = arr2[1].substring(0, arr2[1].length()-1);
		String cadeiaNova = "";
		if (pontoDeCorte == 0)
			cadeiaNova = cadeia2;
		else if(pontoDeCorte == ((numGenes*tamGene)-1))
			cadeiaNova = cadeia1;
		else
```

```
                cadeiaNova = cadeia1.substring(0, pontoDeCorte) +
cadeia2.substring(pontoDeCorte);
                return new Chrom(new ExtendedBitSet(cadeiaNova));
        }
}
```