Licenciatura em Engenharia Informática
Relatório de Estruturas de Informação

Game Project

Turma: 2DG Grupo 1

Raúl Piloto da Silva Nunes Correia – 1090657

Rui Miguel Teixeira Ribeiro – 1150344

Turma: 2DG Grupo 1

Raúl Piloto da Silva Nunes Correia – 1090657

Rui Miguel Teixeira Ribeiro – 1150344

Introdução

No âmbito da unidade curricular de Estruturas de Informação foi-nos proposto o desenvolvimento de uma aplicação que permite gerir o funcionamento de um jogo de computador de estratégia. A aplicação desenvolvida baseia-se num jogo de conquistas através de um sistema de locais, estradas, forças e alianças entre jogadores.

Posto isto, um Local é caracterizado pelo nome, uma personagem (dono), se tiver, e a sua dificuldade.

As estradas são ligações entre locais, caracterizadas também por uma dificuldade.

Uma personagem é caracterizada pelo seu nome e pela sua força.

Uma aliança é uma parceria entre personagens, caracterizada pela sua força que é o resultado da soma das forças dos 2 aliados multiplicado pelo fator de compatibilidade.

Uma conquista acontece quando uma personagem tem força necessária para percorrer a estrada, aceder ao local e, caso exista, derrotar o dono atual do local a conquistar. Este processo repete-se para todos os locais,estradas e personagens intermédios entre o Local A e Local B.

Para a primeira parte do trabalho, foi criada a classe ControloDoJogo, que funciona como um Controller, onde são guardadas a rede de locais e estradas e a rede de personagens e alianças e onde se encontram as respostas ás perguntas presentes no enunciado.

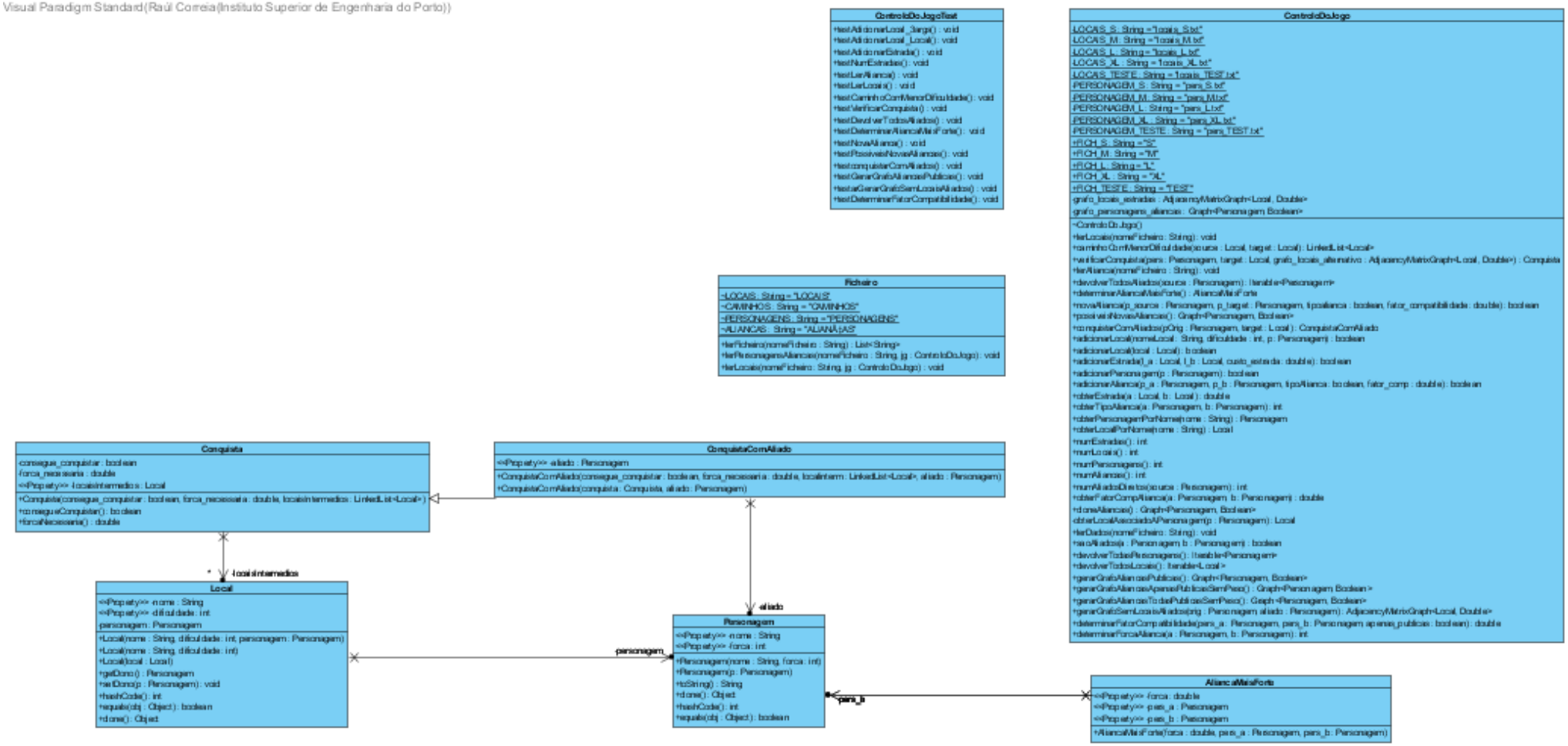
Em termos de algoritmos, utilizou-se as classes realizadas nas aulas PL e a complementar, a classe AlgoritmosJogo que contém uma adaptação do algoritmo de dijkstra para considerar a dificuldade de estradas, locais e personagens donos dos locais.

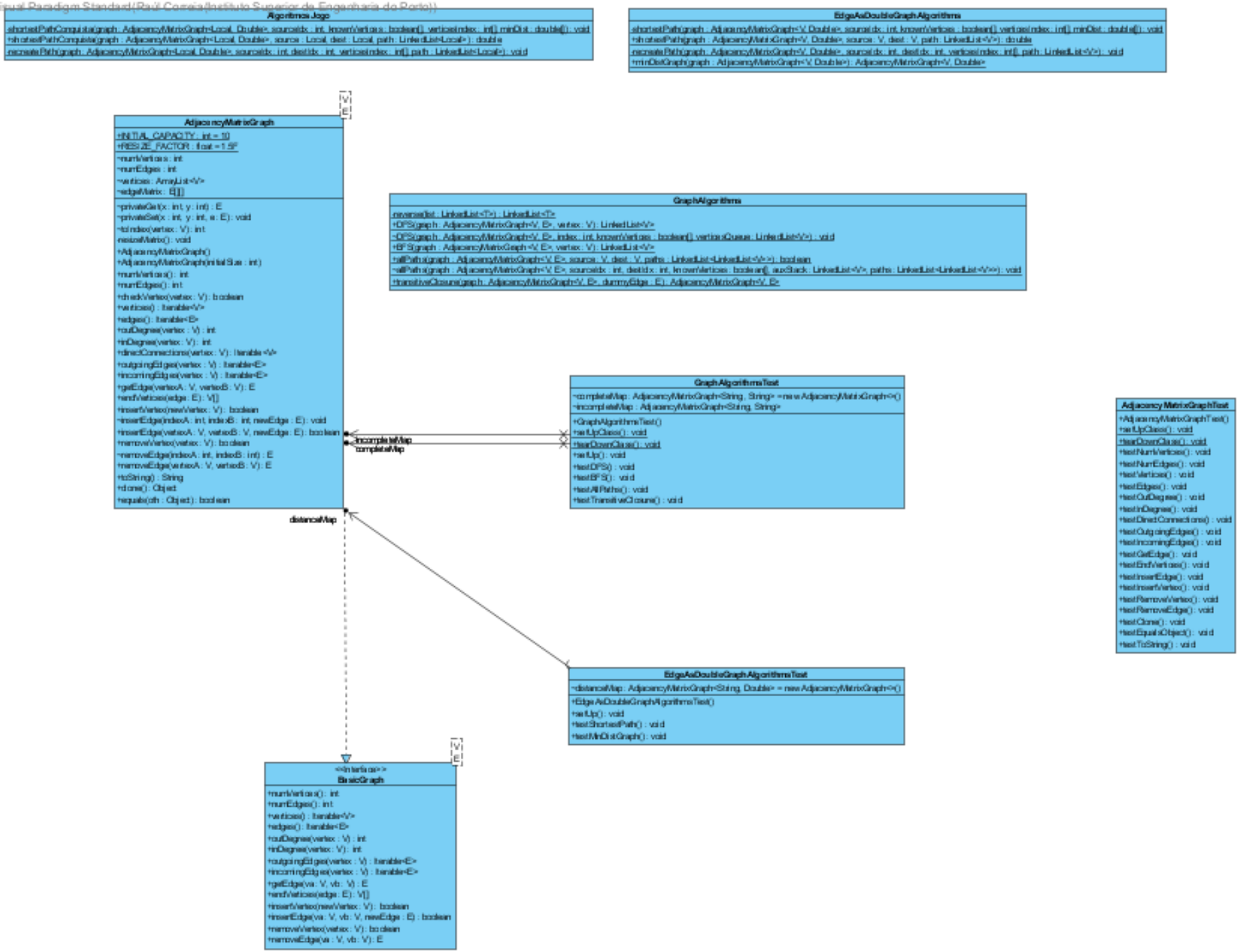
As restantes classes são objetos adicionais que complementam a execução do jogo, tanto na criação dos elementos do jogo como na execução do mesmo.

Foi criada uma classe Ficheiro, que contêm os métodos para ler ficheiros genéricos de texto e métodos de leitura para Personagens, Alianças, Locais e Estradas

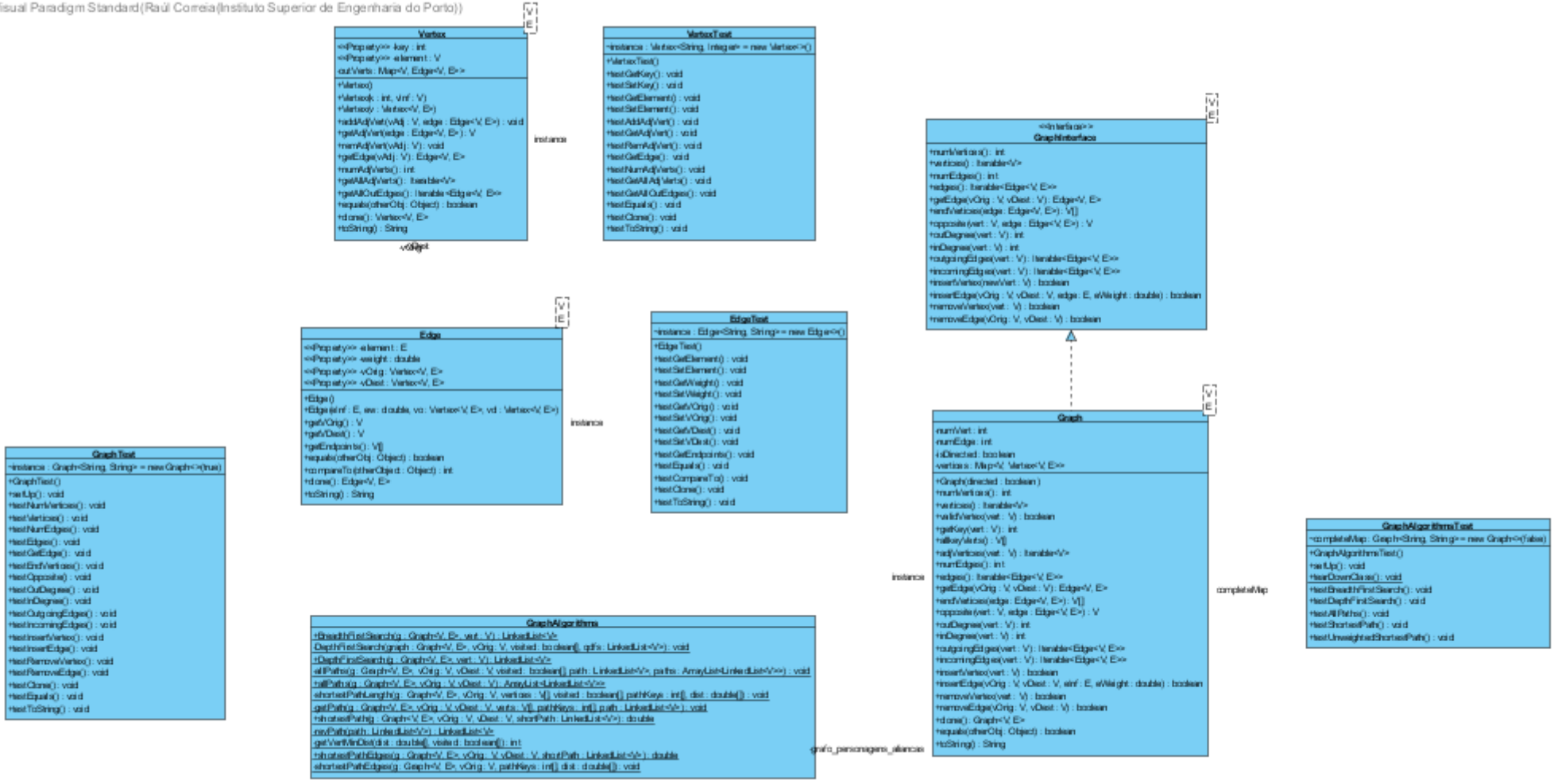
Conteúdo

Diagrama de Classes



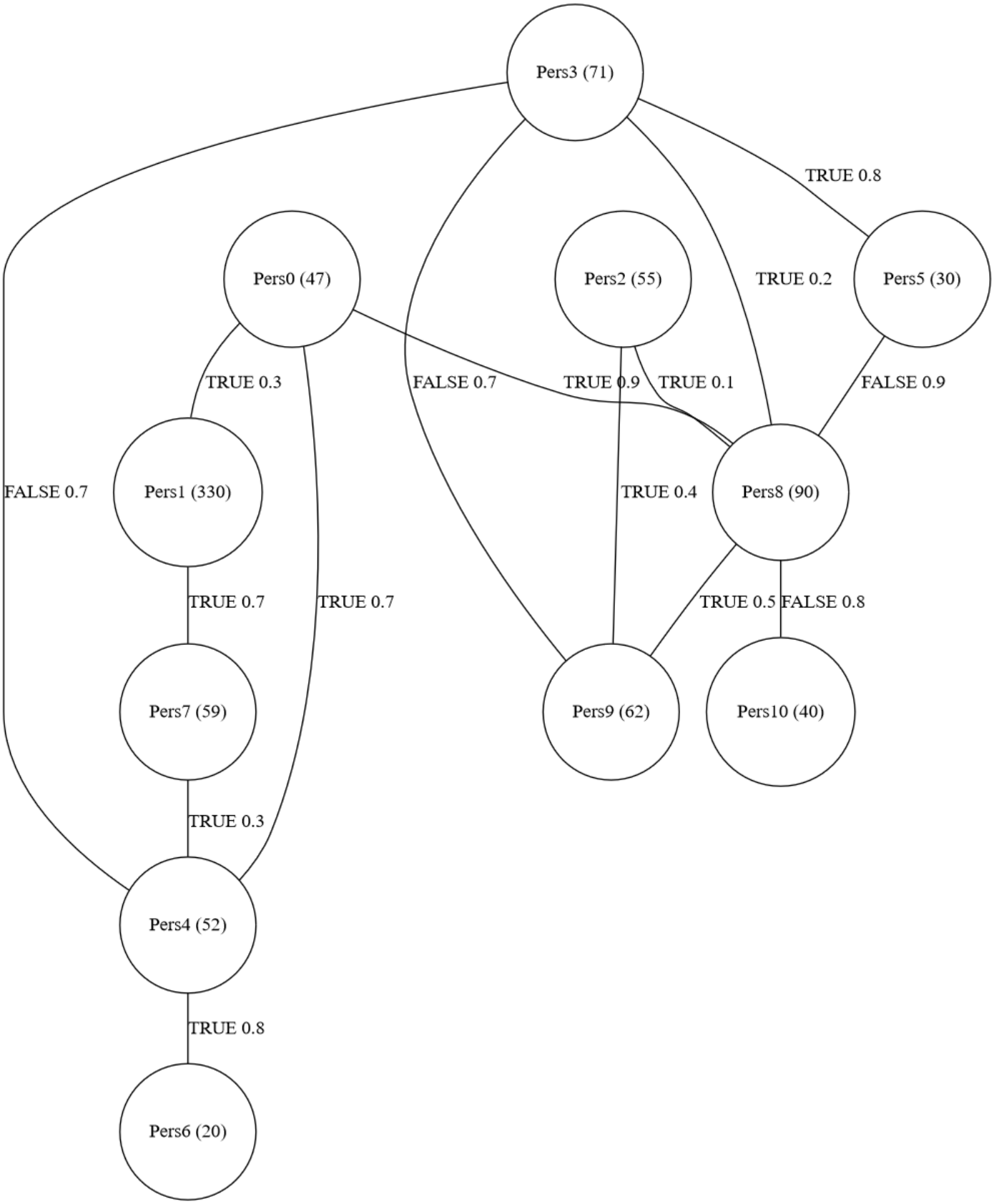


Visual Paradigm Standard(Raúl Correia(Instituto Superior de Engenharia do Porto))

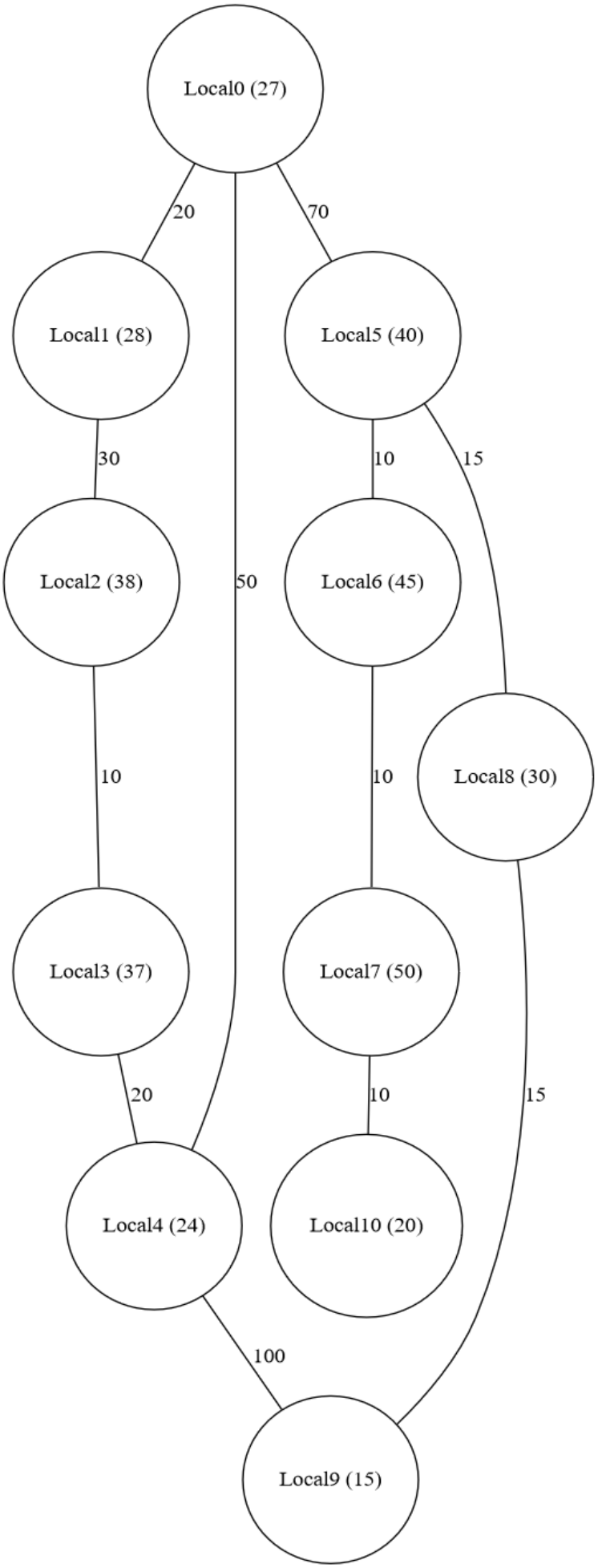


Grafos de Teste

Grafo Personagens



Grafo Locais



Complexidade

Resumo:

- 1
  - A) Ler ficheiro de dados
    - ControloDoJogo.lerLocais(String nomeFicheiro) $O(n*V^2_{local})$
  - B) Caminho com menor dificuldade
    - ControloDoJogo.caminhoComMenorDificuldade(Local source, Local target) $O(V^2_{local})$
  - C) Verificar se a conquista é possível
    - ControloDoJogo.verificarConquista(Personagem pers, Local source, Local target, AdjacencyMatrixGraph<Local, Double> grafo_locais_alternativo) $O(V^2_{locais})$
- 2
  - A) Ler ficheiro de Alianças
    - ControloDoJogo.lerAlianca(String nomeFicheiro) $O(n*V_{pers})$
  - B) Listar os aliados de uma personagem
    - ControloDoJogo.devolverTodosAliados(Personagem source) $O(1)$
  - C) Determinar a aliança mais forte
    - ControloDoJogo.determinarAliancaMaisForte() $O(V^2_{personagem})$
  - D) Realizar uma nova aliança
    - ControloDoJogo.novaAlianca(Personagem p_source, Personagem p_target, boolean tipoalianca, double fator_compatibilidade) $O(V^2_{personagem})$
  - E) Criar grafo de alianças
    - ControloDoJogo.possiveisNovasAliancas() $O(V^3_{personagem})$
- 3
  - F) Verificar se a conquista é possível com um dos aliados
    - ControloDoJogo.conquistaComAliados(Personagem a, Local target) $O(V_{personagem}*V^2_{locais})$

Completo:

1.

a. lerLocais(String nomeFicheiro) $O(n*V^2_{local})$

```
public List<String> lerFicheiro(String nomeFicheiro) {
    Scanner scn = null;                                          //O(1)
    List<String> lista = new ArrayList<>();                      //O(1)
    try {
      scn = new Scanner(new FileReader(nomeFicheiro));           //O(1)
      while (scn.hasNext()) {                                    //O(n)
        lista.add(scn.next());                                    //O(1)
      }
    } catch (FileNotFoundException ex) {
      System.out.printf("Foi impossvel ler o ficheiro %s\n", nomeFicheiro);//O(1)
    } finally {
      if (scn != null) {                                         //O(1)
        scn.close();                                              //O(1)
      }
    }
    return lista;                                                //O(1)
  }                                                              //Total O(n)

public void lerLocais(String nomeFicheiro) {
  Ficheiro f = new Ficheiro();
  f.lerLocais(nomeFicheiro, this);                                    //Total: O(n*Vlocal^2)
}

public void lerLocais(String nomeFicheiro, ControloDoJogo jg) {
    List<String> conteudoFich = lerFicheiro(nomeFicheiro);            //O(n)
    boolean lerLocais = false;                                        //O(1)
    boolean lerCaminhos = false;                                      //O(1)

    String linhaSplit[] = null;                                       //O(1)
    for (String linha : conteudoFich) {                               //O(n)
      if (linha.equals(LOCAIS)) {                                     //O(1)
        lerLocais = true;                                             //O(1)
        continue;
      }
      if (linha.equals(CAMINHOS)) {                                   //O(1)
        lerLocais = false;                                            //O(1)
        lerCaminhos = true;                                           //O(1)
        continue;
      }
      if (lerLocais == true) {                                        //O(1)
        linhaSplit = linha.split(",");                                 //O(z)
        Local l = new Local(linhaSplit[0], Integer.parseInt(linhaSplit[1]));  //O(1)
        if (linhaSplit.length >= 3) {                                 //O(1)
          Personagem p = jg.obterPersonagemPorNome(linhaSplit[2]);    //O(V)
          if (p != null) {                                            //O(1)
            l.setDono(p);                                             //O(1)
          }
        }
        jg.adicionarLocal(l);                                         //O(v^2)
        continue;
      }
      if (lerCaminhos == true) {                                      //O(1)
        final int CAMPO_LOCAL_A = 0;                                  //O(1)
        final int CAMPO_LOCAL_B = 1;                                  //O(1)
        final int CAMPO_DIF_ESTRADA = 2;                              //O(1)

        linhaSplit = linha.split(",");                                 //O(m)

        String string_local_a = linhaSplit[CAMPO_LOCAL_A];            //O(1)
        String string_local_b = linhaSplit[CAMPO_LOCAL_B];            //O(1)

        Local locala = null, localb = null;                           //O(1)
        for (Local l : jg.devolverTodosLocais()) {                    //O(v)
          if (string_local_a.equals(l.getNome())) {                   //O(1)
```

```
                locala = l;
            }
            if (string_local_b.equals(l.getNome())) {                    //O(1)
                localb = l;                                              //O(1)
            }
            if (locala != null && localb != null) {                      //O(1)
                double e = Double.parseDouble(linhaSplit[CAMPO_DIF_ESTRADA]);//O(1)
                jg.adicionarEstrada(locala, localb, e);                  //O(V^2local)
                break; //break ao for
            }

        }
    }

  }
}                                       //Total:O(n)*(O(z)+O(Vlocal^2)+O(m)+O(e)+(Vlocal^2)
                                        //Total: O(n*Vlocal^2)
```

b. caminhoComMenorDificuldade(Local source, Local target) $O(V^2_{locais})$

```
public LinkedList<Local> caminhoComMenorDificuldade(Local source, Local target) {
    LinkedList<Local> path = new LinkedList<>();                         //O(1)
    EdgeAsDoubleGraphAlgorithms.shortestPath(grafo_locais_estradas, source, target, path); //O(V^2)
    return path;                                                         //O(1)
}                                                                        //Total:O(V^2)
```

c. verificarConquista(Personagem pers, Local source, Local target, AdjacencyMatrixGraph<Local, Double> grafo_locais_alternativo) $O(V^2_{locais})$

```
public Conquista verificarConquista(Personagem pers, Local source, Local target, AdjacencyMatrixGraph<Local, Double>
grafo_locais_alternativo) {
    AdjacencyMatrixGraph<Local, Double> grafo_locais_a_utilizar = grafo_locais_estradas;
    if (grafo_locais_alternativo != null) {                              //O(1)
        grafo_locais_a_utilizar = grafo_locais_alternativo;              //O(1)
    }
    if (!grafo_personagens_aliancas.validVertex(pers) || !grafo_locais_a_utilizar.checkVertex(target)) {//O(1)
        return new Conquista(false, -1, null);                           //O(1)
    }
    if (target.getDono() != null && target.getDono().equals(pers)) {     //O(1)
        return new Conquista(false, -1, null);                           //O(1)
    }
    if (source.getDono() != null && !source.getDono().equals(pers)) {
        return new Conquista(false, -1, null);
    }
    final double SEM_CAMINHO = -1;                                       //O(1)
    LinkedList<Local> path = new LinkedList<>();                         //O(1)

    double dificuldade = graph.AlgoritmosJogo.shortestPathConquista(grafo_locais_a_utilizar, source, target, path,pers);
                                                                         //O(V^2)
    if (dificuldade != SEM_CAMINHO) {                                    //O(1)
        if (path.peekFirst() == source) {                                //O(1)
            path.removeFirst();                                          //O(1)
        }
        if (path.peekLast() == target) {                                 //O(1)
            path.removeLast();                                           //O(1)
        }
        boolean consegue_conquistar = false;                             //O(1)
        if (pers.getForca() > dificuldade) {                             //O(1)
            consegue_conquistar = true;                                  //O(1)
        }
        Conquista cq = new Conquista(consegue_conquistar, dificuldade, path); //O(1)
        return cq;                                                       //O(1)
    }
    return new Conquista(false, -1, path);                               //O(1)
}                                                                        //Total: O(V^2)
```

2.

a. lerAlianca(String nomeFicheiro) $O(n*V_{pers})$

```
public void lerAlianca(String nomeFicheiro) {
   Ficheiro f = new Ficheiro();
   f.lerPersonagensAliancas(nomeFicheiro, this);                    //Total O(n*Vpers)
}
```

```
public void lerPersonagensAliancas(String nomeFicheiro, ControloDoJogo jg) {
     List<String> conteudoFich = lerFicheiro(nomeFicheiro);          //O(n)
     boolean lerPersonagens = false;                                 //O(1)
     boolean lerAliancas = false;                                    //O(1)

     String linhaSplit[] = null;                                     //O(1)
     for (String linha : conteudoFich) {                             //O(n)
       if (linha.equals(PERSONAGENS)) {                              //O(1)
          lerPersonagens = true;                                     //O(1)
          continue;
       }
       if (linha.equals(ALIANCAS)) {                                 //O(1)
          lerPersonagens = false;                                    //O(1)
          lerAliancas = true;                                        //O(1)
          continue;
       }
       if (lerPersonagens == true) {                                 //O(1)
          linhaSplit = linha.split(",");                             //O(1)
          Personagem p = new Personagem(linhaSplit[0], Integer.parseInt(linhaSplit[1]));  //O(1)
          jg.adicionarPersonagem(p);                                 //O(1)
          continue;
       }
       if (lerAliancas == true) {                                    //O(1)
          final int CAMPO_PERS_A = 0;                                //O(1)
          final int CAMPO_PERS_B = 1;                                //O(1)

          final int CAMPO_TIPO_ALIANCA = 2;                          //O(1)
          final int CAMPO_ALIANCA_FATOR_COMPATIBILIDADE = 3;         //O(1)
          linhaSplit = linha.split(",");                             //O(1)

          String pers_a = linhaSplit[CAMPO_PERS_A];                  //O(1)
          String pers_b = linhaSplit[CAMPO_PERS_B];                  //O(1)
          Boolean tipoAlianca = Boolean.parseBoolean(linhaSplit[CAMPO_TIPO_ALIANCA]);  //O(1)
          Double fator_comp = Double.parseDouble(linhaSplit[CAMPO_ALIANCA_FATOR_COMPATIBILIDADE]);//O(1)
          Personagem persA = null, persB = null;                     //O(1)
          for (Personagem p : jg.devolverTodasPersonagens()) {       //O(V)
            if (pers_a.equals(p.getNome())) {                        //O(1)
               persA = p;                                            //O(1)
               continue;
            }
            if (pers_b.equals(p.getNome())) {                        //O(1)
               persB = p;                                            //O(1)
            }
            if (persA != null && persB != null) {                    //O(1)
               jg.adicionarAlianca(persA, persB, tipoAlianca, fator_comp);  //O(1)
               break;
            }
          }

       }
     }
   }                                                                 //Total O(n*Vpers)
```

b. devolverTodosAliados(Personagem source) O(1)

```
public Iterable<Personagem> devolverTodosAliados(Personagem source) {
  if (!grafo_personagens_aliancas.validVertex(source)) {                    //O(1)
    return null;
  }
  return grafo_personagens_aliancas.adjVertices(source);                    //O(1)
}                                                                           //Total: O(1)
```

c. determinarAliancaMaisForte() $O(V^2_{personagens})$

```
public AliancaMaisForte determinarAliancaMaisForte() {
  Personagem p_a = null;                                                    //O(1)
  Personagem p_b = null;                                                    //O(1)

  double forca_alianca = -1;                                                //O(1)
  for (Personagem p : grafo_personagens_aliancas.vertices()) {              //O(V)
    for (Personagem pAdj : grafo_personagens_aliancas.adjVertices(p)) {     //O(V)
      if (forca_alianca == -1) {                                            //O(1)
        p_a = p;                                                            //O(1)
        p_b = pAdj;                                                         //O(1)
        forca_alianca = (p.getForca() + pAdj.getForca()) * grafo_personagens_aliancas.getEdge(p, pAdj).getWeight();//O(1)
      } else {
        double outra_forca = (p.getForca() + pAdj.getForca()) * grafo_personagens_aliancas.getEdge(p, pAdj).getWeight(); //O(1)
        if (outra_forca > forca_alianca) {                                  //O(1)
          p_a = p;                                                          //O(1)
          p_b = pAdj;                                                       //O(1)
          forca_alianca = outra_forca;                                      //O(1)
        }

      }
    }
  }                                                                         //Total bloco : O(V^2)
  if (p_a != null && p_b != null) {
    AliancaMaisForte amf = new AliancaMaisForte(forca_alianca, p_a, p_b);//O(1)
    return amf;                                                             //O(1)

  }
  return null;                                                              //O(1)
}                                                                           //Total O(V^2)
```

d. novaAlianca(Personagem p_source, Personagem p_target, boolean tipoalianca, double fator_compatibilidade) $O(V^2_{personagens})$

```
public boolean novaAlianca(Personagem p_source, Personagem p_target, boolean tipoalianca, double fator_compatibilidade) {
  final double SEM_CAMINHO = -1;
  if (!grafo_personagens_aliancas.validVertex(p_source) || !grafo_personagens_aliancas.validVertex(p_target)) {      //O(1)
    return false;                                                                                                    //O(1)
  }

  LinkedList<Personagem> path = new LinkedList<>();                                                                  //O(1)
  double dist = 0;                                                                                                   //O(1)

  //Vamos ver o caminho de p_source para p_target
  Graph<Personagem, Boolean> grafo_aliancas_publicas_sem_peso = gerarGrafoAliancasApenasPublicasSemPeso(); //O(V^2)
  dist = graphbase.GraphAlgorithms.shortestPath(grafo_aliancas_publicas_sem_peso, p_source, p_target, path);     //O(V^2)

  if (dist == SEM_CAMINHO) {                                                                                         //O(1)
    return adicionarAlianca(p_source, p_target, tipoalianca, fator_compatibilidade);                                 //O(1)

  } else {
    int numPers = path.size();                                                                                       //O(1)
    //Se o número de Personagens é maior do que 1, então é n-1 ramos
    if (numPers > 1) {                                                                                               //O(1)
      numPers--;                                                                                                     //O(1)
    }
    double fator_comp = 0;                                                                                           //O(1)
    Personagem a = null;                                                                                             //O(1)
    Personagem b = null;                                                                                             //O(1)
    while (!path.isEmpty()) {                                                                                        //O(V)
      if (a == null && b == null) {                                                                                  //O(1)
        a = path.pop();                                                                                              //O(1)
        b = path.pop();                                                                                              //O(1)
        fator_comp = grafo_personagens_aliancas.getEdge(a, b).getWeight();                                           //O(1)
      } else {
        a = b;                                                                                                       //O(1)
        b = path.pop();                                                                                              //O(1)
        fator_comp += grafo_personagens_aliancas.getEdge(a, b).getWeight();                                          //O(1)
      }
    }
    double mediaComp = fator_comp / numPers;                                                                         //O(1)
    grafo_personagens_aliancas.insertEdge(p_source, p_target, tipoalianca, mediaComp);                               //O(1)
    return true;                                                                                                     //O(1)
  }
}                                                                                                                    //Total:O(V^2)
```

```
public Graph<Personagem, Boolean> gerarGrafoAliancasApenasPublicasSemPeso() {
  Graph<Personagem, Boolean> grafo_aliancas_publicas = grafo_personagens_aliancas.clone();
  for (Personagem pOrig : grafo_personagens_aliancas.vertices()) {                                  //O(V)
    for (Personagem pAdj : grafo_personagens_aliancas.adjVertices(pOrig)) {                         //O(V)
      if (grafo_personagens_aliancas.getEdge(pOrig, pAdj) != null && grafo_personagens_aliancas.getEdge(pOrig, pAdj).getElement(
) == false) {                                                                                       //O(1)
        grafo_aliancas_publicas.removeEdge(pOrig, pAdj);                                            //O(1)
      } else {
        if (grafo_personagens_aliancas.getEdge(pOrig, pAdj) != null && grafo_personagens_aliancas.getEdge(pOrig, pAdj).getEleme
nt() == true) {                                                                                     //O(1)
          grafo_aliancas_publicas.getEdge(pOrig, pAdj).setWeight(1);                                //O(1)
        }

      }
    }
  }

  return grafo_aliancas_publicas;
}                                                                                                   //O(V^2)

                                                                                                    //Total: O(V^2)
```

e. possiveisNovasAliancas() $O(V^3_{personagens})$

```
public Graph<Personagem, Boolean> possiveisNovasAliancas() {

   Graph<Personagem, Boolean> ng = (Graph<Personagem, Boolean>) grafo_personagens_aliancas.clone();
   for (Personagem p_k : grafo_personagens_aliancas.vertices()) {                    //O(v)
      for (Personagem p_i : grafo_personagens_aliancas.vertices()) {                 //O(v)
         if (!p_i.equals(p_k) && ng.getEdge(p_i, p_k) != null) {
            for (Personagem p_j : grafo_personagens_aliancas.vertices()) {          //O(v)
               if (!p_i.equals(p_j) && !p_j.equals(p_k) && ng.getEdge(p_k, p_j) == null) {
                  if (ng.getEdge(p_i, p_j) != null) {                               //O(1)
                     double ed1 = ng.getEdge(p_k, p_i).getWeight();                 //O(1)
                     double ed2 = ng.getEdge(p_i, p_j).getWeight();                 //O(1)
                     double media = (ed1 + ed2) / 2;                                //O(1)
                     ng.insertEdge(p_k, p_j, Boolean.TRUE, media);                  //O(1)
                  }
               }
            }
         }
      }
   }
   return ng;                                                                       //O(1)
}                                                                  //O(V) * O(V) * O(V)
```

3.

f. conquistarComAliados(Personagem a, Local source, Local target) $O(\underline{V_{personagens}} * V^2_{locais})$

```
public ConquistaComAliado conquistarComAliados(Personagem pOrig, Local source, Local target) {
   final double INVALIDO = -1;
   if (!grafo_personagens_aliancas.validVertex(pOrig) || !grafo_locais_estradas.checkVertex(target)) {          //O(v)
      return null;
   }
   if (target.getDono() != null && target.getDono().equals(pOrig)) {                                            //O(1)
      return new ConquistaComAliado(false, -1, null,null);                                                      //O(1)
   }
   if (source.getDono() != null && !source.getDono().equals(pOrig)) {                                           //O(1)
       return new ConquistaComAliado(false, -1, null,null);                                                     //O(1)
   }

   int forca_antiga = pOrig.getForca();                                                                         //O(1)
   for (Personagem aliado : grafo_personagens_aliancas.adjVertices(pOrig)) {                                    //O(V)
      AdjacencyMatrixGraph<Local, Double> grafo_sem_locais_aliados = gerarGrafoSemLocaisAliados(pOrig, aliado);//O(V^2)
      pOrig.setForca(forca_antiga);                                                                             //O(1)
      pOrig.setForca(determinarForcaAlianca(pOrig, aliado));                                                    //O(1)
      Conquista cq = verificarConquista(pOrig,source,target, grafo_sem_locais_aliados);                         //O(V_locais_^2)
      if (cq.consegueConquistar()) {                                                                            //O(1)
         pOrig.setForca(forca_antiga);                                                                          //O(1)
         ConquistaComAliado cqal = new ConquistaComAliado(cq, aliado);                                          //O(1)
         return cqal;                                                                                           //O(1)
      }
   }
   pOrig.setForca(forca_antiga);                                                                                //O(1)

   return new ConquistaComAliado(false, INVALIDO, new LinkedList<Local>(), null);                              //O(1)

}                                                                  //O(Vpersonagens) * (O(V^2 Locais) + O(V^2 Locais))
```

```
public AdjacencyMatrixGraph<Local, Double> gerarGrafoSemLocaisAliados(Personagem orig, Personagem aliado) {
     AdjacencyMatrixGraph<Local, Double> grafo_locais_sem_o_aliado = (AdjacencyMatrixGraph<Local, Double>) grafo_locais_estra
das.clone();                                                                                                    //O(V^2)

     for (Local loc : grafo_locais_estradas.vertices()) {                                                       //O(n)
        if (loc.getDono() != null) {                                                                            //O(1)
           if (loc.getDono().equals(aliado)) {                                                                  //O(1)
              grafo_locais_sem_o_aliado.removeVertex(loc);                                                      //O(V)
           }
        }

     }
     return grafo_locais_sem_o_aliado;
  }                                                                                 //O(V^2)
```