Escola Superior
de Tecnologia
e Gestão
Politécnico de Coimbra

Relatório de Projeto

Programação IV

Avaliação [Periódica]

Autor:

Assinado por : **Jorge Miguel dos Santos Martins**
Num. de Identificação: BI13549641
Data: 2021.05.19 11:48:23+01'00'

CHAVE MÓVEL

Data: <Maio de 2021>

Resumo

Trabalho realizado com o objetivo da criação do jogo “tic-tac toe” ou mais conhecido pelo “jogo do galo”, ao qual além das opções de jogabilidade entre jogadores na mesma máquina, existe a possibilidade de jogabilidade entre aplicações diferentes e maquinas diferentes seguindo um protocolo de comunicação universal.

Palavras-chave

Sockets, Comunicação em Rede, Programação Orientada a Objetos, Projeto 2.

Índice

Lista de Figuras

Lista de Tabelas

Não foi encontrada nenhuma entrada do índice de ilustrações.

Lista de Acrónimos

IA	Inteligência Artificial

1. Introdução

Foi designado o desenvolvimento de uma aplicação de Jogo na linguagem Java que tenha como principal foco a comunicação em “socket” com outras aplicações, recorrendo a um protocolo fornecido inicialmente tornando a comunicação universal.

Como referenciado anteriormente este projeto diferencia-se pela introdução da funcionalidade de comunicação em “socket” entre aplicações.

No presente relatório a abordagem será demonstrar as classes dos objetos manipulados pelo programa, seguidamente serão demonstradas as opções utilizadas para uso de métodos estáticos de interação com o utilizador para a manipulação desses objetos e por fim demonstrar os métodos usados para a ligação em “socket”.

Juntamente com este relatório será enviado a documentação da aplicação em “Javadoc” para que com mais especificidade seja descrito a funcionalidade da aplicação.

2. Objectivos e Metodologias

Neste projecto da unidade curricular de Programação IV do segundo semestre do segundo ano da licenciatura em Sistemas e Tecnologias da Informação é esperado que o aluno construa um a aplicação de "jogo do galo" com o principal foco na comunicação em "sockets" com outras aplicações, através de um protocolo fornecido pelo docente para que a comunicação seja universal.

2.1. Ferramentas e Tecnologias

Para este Projeto de programação apenas foi utilizado o software Eclipse para ao desenvolvimento do código, pois o software guarda os seus dados em ficheiros ".dat" (ficheiros de objectos).

2.2. Planeamento

O planeamento do projeto foi realizado de forma linear, primeiramente aquando o projeto ficou disponível foi visualizado o enunciado e através dos requisitos foi desenvolvido o jogo de forma a que este ficasse funcional. Após o desenvolvimento da arquitetura do jogo, geriu-se o tempo por fases ao qual inicialmente desenvolveu-se a funcionalidade de jogo entre jogadores e em seguida de jogador para computador. Após o jogo estar funcional foram desenvolvidos os pormenores da aplicação do decorrer do jogo tais como gravação de jogos, armazenamento de dados estatísticos, listagens. Na ultima fase de desenvolvimento do projeto foi implementada a funcionalidade de desenvolvimento em rede, sendo que foi a que mais tempo foi ocupado no seu desenvolvimento.

3. Trabalho Desenvolvido

3.1. Requisitos Implementados

Tabela Requisitos							
Requisito	Estado	Requisito	Estado	Requisito	Estado	Requisito	Estado
R1	Completo	R21	Completo	R41	Completo	R61	Completo
R2	Completo	R22	Completo	R42	Completo	R62	Completo
R3	Completo	R23	Completo	R43	Completo	R63	Completo
R4	Completo	R24	Completo	R44	Completo	R64	Completo
R5	Completo	R25	Completo	R45	Completo	R65	Completo
R6	Completo	R26	Completo	R46	Completo		
R7	Completo	R27	Completo	R47	Completo		
R8	Completo	R28	Completo	R48	Completo		
R9	Completo	R29	Completo	R49	Completo		
R10	Completo	R30	Completo	R50	Completo		
R11	Completo	R31	Completo	R51	Completo		
R12	Completo	R32	Completo	R52	Completo		
R13	Completo	R33	Completo	R53	Completo		
R14	Completo	R34	Completo	R54	Completo		
R15	Completo	R35	Completo	R55	Completo		
R16	Completo	R36	Completo	R56	Completo		
R17	Completo	R37	Completo	R57	Completo		
R18	Completo	R38	Completo	R58	Completo		
R19	Completo	R39	Completo	R59	Completo		
R20	Completo	R40	Completo	R60	Completo		

3.2. Classes e *Packages*

Para uma melhor organização e legibilidade do meu código eu o dividi por packages agrupando as classes aos seus objetivos.

Neste ponto de um modo geral são demonstrados os packages e as suas classes ao qual serão usados diagramas de classe. Dividi por package cada representação de diagrama para que seja mais compreensível a organização do código e assim de uma forma ágil se compreender a estrutura do projeto.

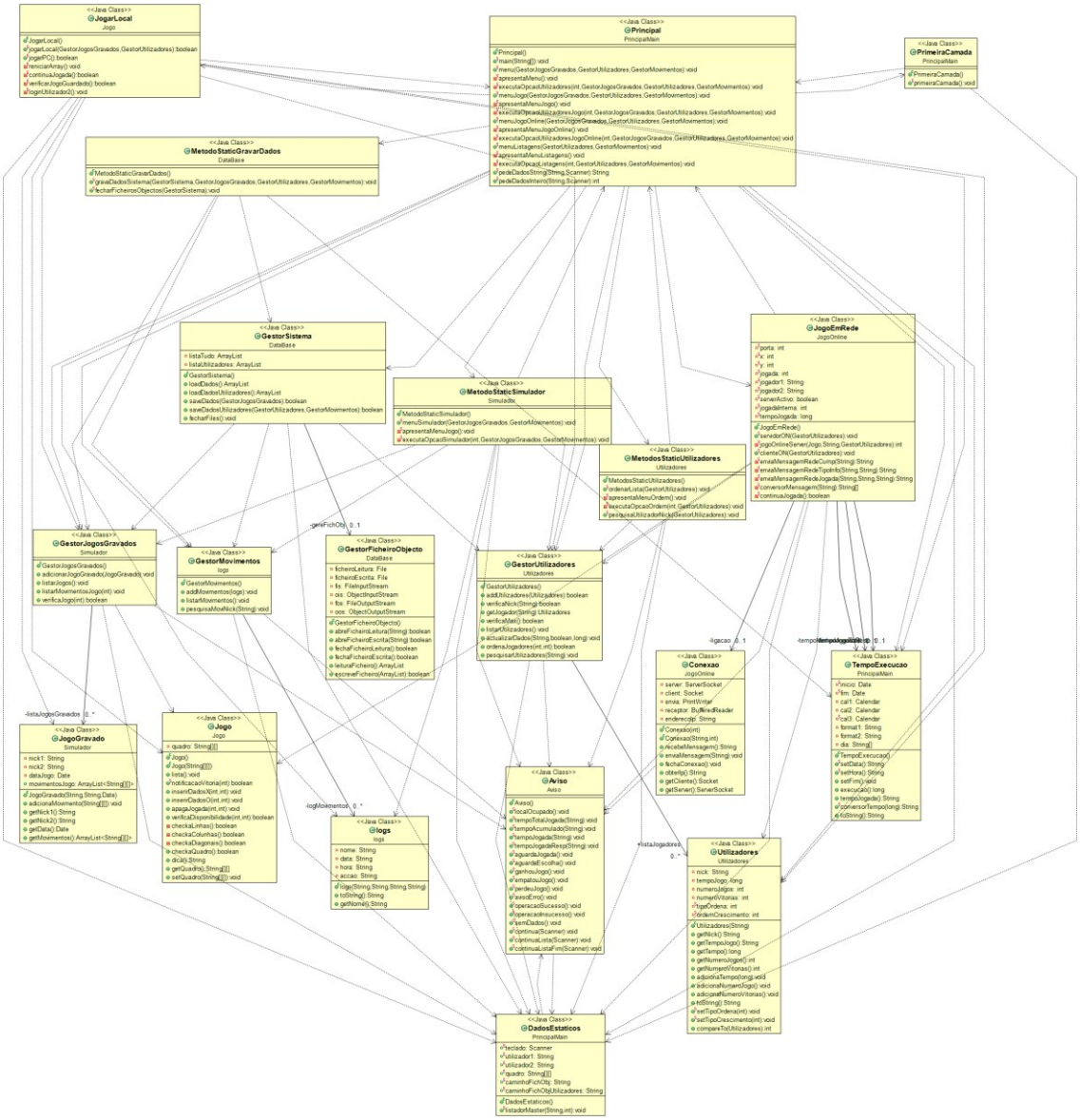

Figura 1 - Esquema UML de Classes geral

Para se verificar o esquema geral de classes e as suas dependências a figura 1 retrata como as classes se relacionam umas com as outras. Como referenciado as classes serão apresentadas de forma dividida pois neste contexto geral seria bastante confuso, apenas é demonstrado para se visualizar de forma geral a arquitetura da aplicação.

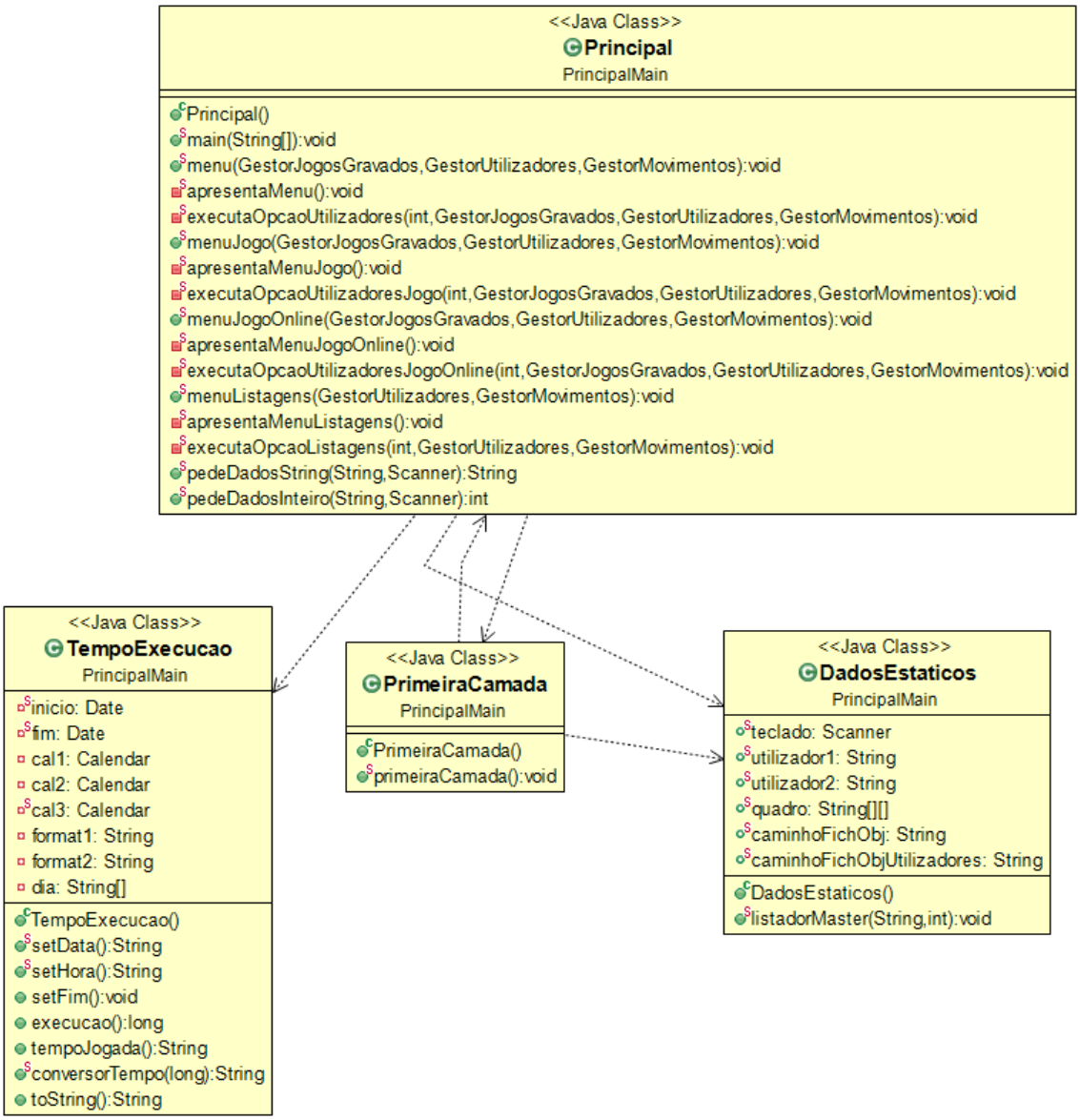

Figura 2 - Package Principal

Neste package encontra-se a classe que contem o método “main”, esta classe Principal contém os menus principais do software e dois métodos de inserção de dados com interação com o utilizador.

A classe da Primeira Camada como o nome indica é a primeira camada da aplicação esta é responsável por rececionar o utilizador pedindo se este entra com um “nickname” ou se entra de forma anonima.

A classe de Dados Estáticos é responsável pela aglomeração de variáveis estáticas para uso no sistema, de forma a que não esteja sempre a movimentar objetos para ações mínimas, como por exemplo criar sempre um objeto “Scanner” para ler do teclado um input, ou por exemplo ter acesso ao “nickname” do utilizador que está ligado no sistema.

A classe “TempoExecução” é a classe responsável pela monitorização do tempo de execução de ações no sistema. Esta classe é usada para verificar o tempo de utilização

da aplicação, o tempo de cada jogada realizada na aplicação e contabilização do tempo de jogo de cada jogador.

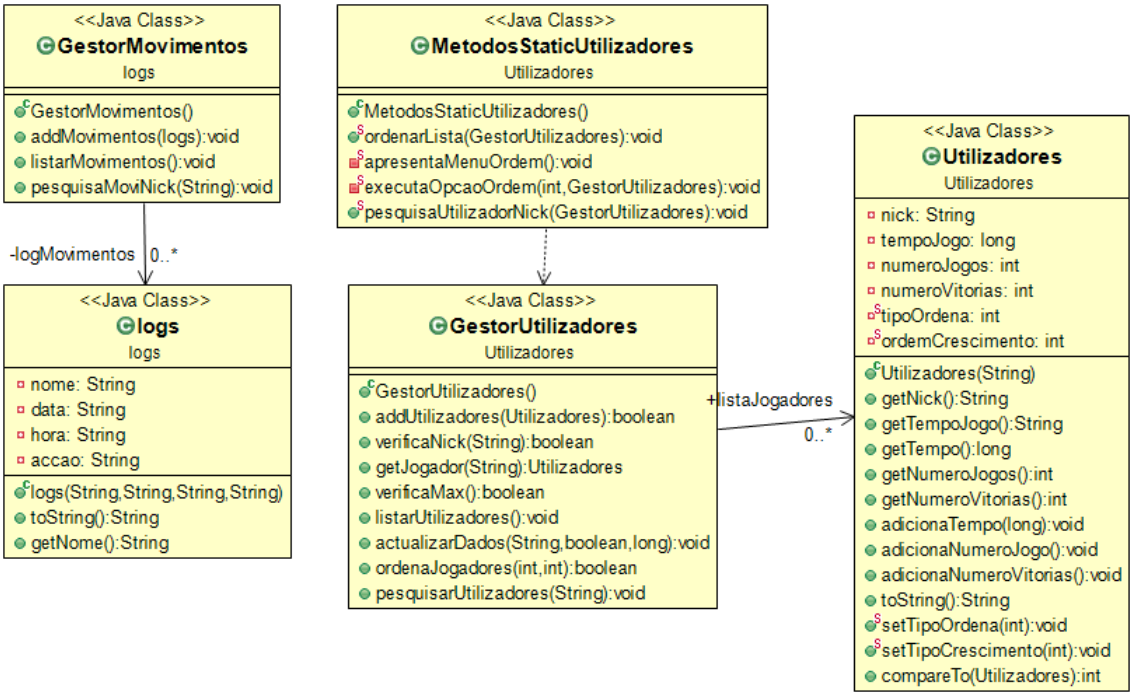

Figura 3 - Utilizadores e Logs

Na figura 3 pode-se verificar as classes responsáveis pelos Utilizadores / Jogadores, a classe Utilizadores é constituída além da variável que armazena o "nickname" do Utilizador, esta também armazena os dados estatísticos do mesmo.

A classe "GestorUtilizadores" como o nome indica é a responsável pela gestão dos objetos da classe de Utilizadores, sendo que nesta se encontra o "ArrayList" que contem os dados de utilizadores em memoria. A classe métodos estáticos utilizadores é a responsável pela interação com o utilizador, que por sua vez chama a classe gestora para a realização dos pedidos.

A classe logs é a classe responsável pelos dados de movimentos do utilizador no software, este objeto contem a data, hora, utilizador e ação que realizou a ação, inicialmente foi pensado em usar apenas uma "String" com esses dados, mas devido ao facto de que seria necessária a utilização de uma listagem condicionada, optou-se pela utilização de um objeto. A classe "GestorMovimentos" é como o nome indica a responsável pela gestão dos objetos "logs", estes dados são armazenados também num "ArrayList" ao iniciar a aplicação.

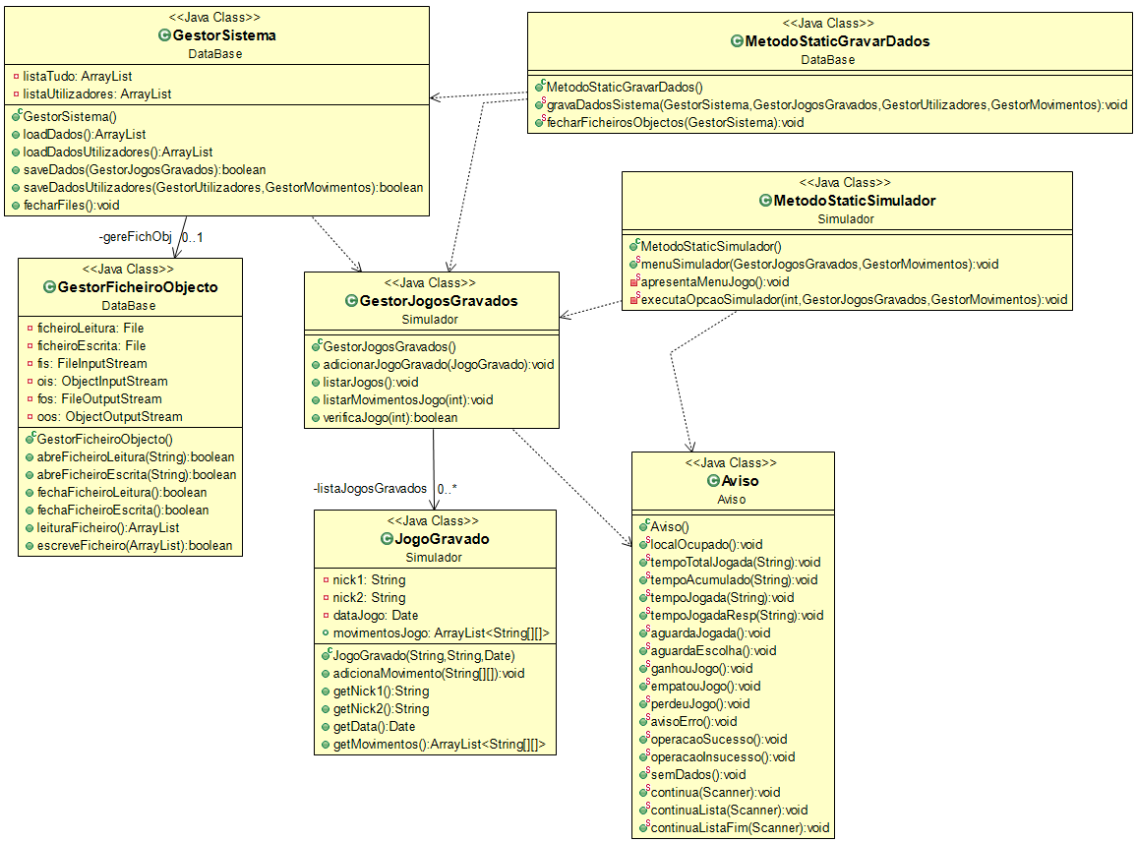

Figura 4 - Package Aviso, DataBase e Simulador

Na figura acima podem-se verificar as classes responsáveis pela gravação de dados como a classe responsável pelos avisos estáticos. Começando pela ultima referenciada esta apenas contem vários métodos de escrita de pequenos avisos de sistema, é utilizada de forma a deixar o código um pouco mais limpo.

Como os nomes indicam a classe “JogoGravado” é a classe responsável pela gravação dos movimentos de um jogo e a classe “GestorJogosGravados” a sua responsável de manipulação. Estas duas classes são chamadas pela classe “MetodoStaticSimulador” que é responsável pela interação com o utilizador, assim dando simulações de jogos locais realizados entre utilizadores.

Como todos os dados são gravados em ficheiros de objetos, existe uma classe de nome ”MetodoStaticGravarDados” que apenas condensa as ações a realizar (gravar e ler dados), para que no código fique um pouco mais limpo a apresentação do mesmo.

A classe “GestorFicheiroObjecto” é a classe que contém os métodos necessários para ler e escrever num ficheiro de objeto e por conseguinte a classe “GestorSistema” é aquela que direciona o uso da classe anterior para que os objetos sejam guardados de forma correta.

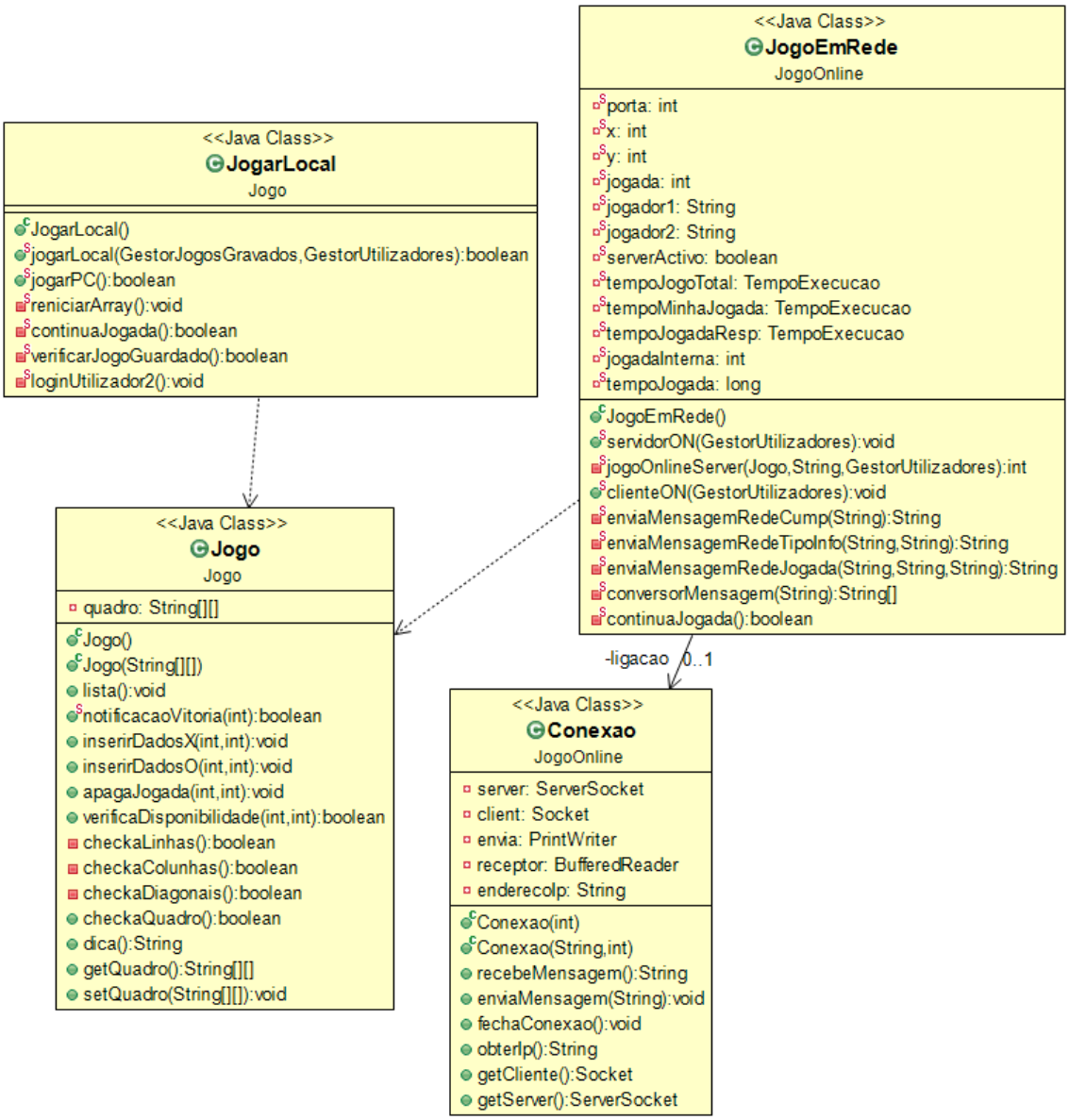

Figura 5 - Classes Relacionas com o Jogo

Na figura acima representada pode-se verificar as classes responsáveis pelo funcionar do jogo em si, sendo que a classe motor para este funcionar é a classe Jogo, que contém os métodos de escrita dos dados, listagem do quadro de jogo, verificação do mesmo, ou seja todo o jogo passa pela manipulação de um objeto desta classe.

Seguidamente pode-se verificar que existem duas classes distintas pelo seu modo de funcionar, uma de nome "JogarLocal" que contem os métodos para o jogo funcionar no modo "offline" sendo que existe a possibilidade de dois utilizadores jogarem na mesma máquina ou se jogar contra o próprio computador. A outra classe como o nome indica "JogarEmRede" é a classe responsável pelo foco maior deste projeto, que é a comunicação em "socket", ao qual estes são utilizados para enviar mensagens entre aplicações para que seja realizado o jogo seguindo o protocolo fornecido.

Por último a classe “Conexao” é a classe responsável para que seja realizada a conexão entre os dois sistemas operativos através da aplicação de jogo, esta contem os métodos de ligação, envio e receção de mensagens por via “socket”.

3.3. Algoritmos

Nesta seção do relatório serão vistos os algoritmos que mais se diferenciam no software, alguns como por exemplo a listagem são usados de forma similar entre classes.

```
public void listarUtilizadores() {

    if(listaJogadores != null){
        if(listaJogadores.size() != 0) {
        Iterator <Utilizadores> tabela = listaJogadores.iterator();
        String envio = "";
        Utilizadores auxiliar;
        int contador = 0;
           while(tabela.hasNext()) {
               auxiliar = tabela.next();
               envio += "Nick: "+auxiliar.getNick() + " Numero de Jogos: "+auxiliar.getNumeroJogos()+" Vitorias
               contador++;

               if(contador == 10) {
                   DadosEstaticos.listadorMaster(" Lista Jogadores \n" + envio + "\n", contador);
                   contador = 0;
                   envio = "";
               }
           }
           if(contador > 0) {
               DadosEstaticos.listadorMaster(" Lista Jogadores \n" + envio + "\n", contador);
           }
            }else {
                Aviso.semDados();
                Aviso.continua(DadosEstaticos.teclado);
            }
    }else {
        Aviso.semDados();
        Aviso.continua(DadosEstaticos.teclado);
    }
}
```

Figura 6 - Algoritmo Listagem 10 a 10

Para que seja possível listar de 10 em 10 elementos, é realizado um contador que verifica quantos elementos já foram inseridos na "string" de envio para o método "listadorMaster" que é responsável apenas por receber uma "string" e assim a listar.

Este algoritmo envia também a contagem para que no método "listadorMaster" seja verificado se a lista chegou ao fim (sendo o contador menor que 10) ou se ainda existem mais informações para listar.

```java
public class Jogo implements Serializable{
    private String [][] quadro = {{"","",""},{"","",""},{"","",""}};
    /**
     * Construtor vazio para inicialização de um novo jogo, imediatamente é listado o quadro em vazio
     */
    public Jogo() {
        lista();
    }
    /**
     * Construtor especificamente usado para o simulador que já recebe quadros com jogadas já realizadas e a
     * @param aQuadro objecto de array estatico de string com jogadas
     */
    public Jogo(String[][] aQuadro) {
        for(int i = 0; i<3; ++i) {
            for(int a=0; a<3; ++a) {
                quadro[i][a] = aQuadro[i][a];
            }
        }
    }

    /**
     * Metodo responsavel pela listagem dos dados (jogadas) inseridas no quadro do objecto da classe
     */
    public void lista() {
        for(int i=0; i<3; ++i) {
            for(int z=0; z<3; ++z) {
                if(z==2) {
                    if(i!=2) {
                        System.out.printf("%1s", quadro[i][z]);
                        System.out.printf("\n_+_+_\n");
                    }else {
                        System.out.printf("%1s", quadro[i][z]);
                    }
                }else {
                    if(z!=2)
                        System.out.printf("%1s", quadro[i][z]);
                        System.out.printf("|");
                }
            }
        }
        System.out.println("\n\n\n");
    }
```

Figura 7 - Algoritmo de Jogo

Sendo a figura principal da aplicação estes métodos são importantes a referenciar, a existência de dois construtores é justificada no facto de que ao iniciar um objeto com construtor vazio é utilizado para o decorrer de um jogo, enquanto que se o objeto for instanciado com a receção de um “array” bidimensional de “Strings” significa que o objeto irá ser usado para uma simulação de um jogo gravado ou está a ser usado para gravar uma jogada. O fato de utilizar um ciclo for é para transferir os dados para o “array” do objeto e não ficar apenas a apontar para o mesmo local de memoria.

É utilizado um ciclo para desenhar o quadro de jogo já com as jogadas, assim o mesmo usa o “printf” para que com a formatação os caracteres e os espaços sejam iguais para não deformar o desenho do quadro.

```
public boolean verificaDisponibilidade(int x, int y) {
    if(x <3 && x >= 0 && y < 3 && y >= 0) {
    if(quadro[x][y].length()==0) {
        return true;
    }}
    return false;
}

/**
 * Metodo responsavel por verificar se existem nas linhas do quadro 3 elementos iguais que significa vitoria
 * @return booleano com resultado da operação
 */
private boolean checkaLinhas() {
    for(int li=0; li<3; ++li) {
        if(!quadro[li][0].equals("") && !quadro[li][1].equals("") && !quadro[li][2].equals("")) {
            if(quadro[li][0].equals(quadro[li][1]) && quadro[li][1].equals(quadro[li][2])) {
                return true;
            }
        }
    }
    return false;
}
```

Figura 8 - Verifica Jogo

No mesmo objeto de jogo é verificado a disponibilidade da localização solicitada pelo jogador para inserir a sua jogada e após cada jogada são acionados mais dois métodos similares ao “checkaLinhas” para verificar se existem três caracteres iguais em linha, coluna ou diagonais.

```
public int compareTo(Utilizadores aObj) {    // 2.º passo ordenamento (esta a por por
    switch(tipoOrdena) {
    case 1: return ((nick.compareToIgnoreCase(aObj.getNick()))*ordemCrescimento);

    case 2: if(tempoJogo > aObj.getTempo()) {
                return (1 * ordemCrescimento);
            }else {return (-1 * ordemCrescimento);}

    case 3:  if(numeroJogos > aObj.getNumeroJogos()) {
                return (1 * ordemCrescimento);
            }else {return (-1 * ordemCrescimento);}

    case 4: if(numeroVitorias > aObj.getNumeroVitorias()) {
                return (1 * ordemCrescimento);
            }else {return (-1 * ordemCrescimento);}
    }
    return 1;

   }
```

Figura 9 - Metodo CompareTo

Sendo que para esta aplicação o armazenamento de dados é feito através de ficheiros de objetos, o método compareTo é necessário para que exista uma ordenação do “ArrayList” e devido ao facto de que existem vários parâmetros para ordenar, foi realizado um “switch” em que cada tipo de ordenação significa um dos “cases” do “switch”. Tanto a variável “tipoOrdenda” como a “ordemCrescimento” são estáticas da classe.

```
private static void executaOpcaoOrdem(int aOpcao, GestorUtilizadores gestorUtilizadores) {
    switch(aOpcao) {
    case 1: gestorUtilizadores.ordenaJogadores(1, 1); Aviso.operacaoSucesso(); break;
    case 2: gestorUtilizadores.ordenaJogadores(1, -1); Aviso.operacaoSucesso(); break;
    case 3: gestorUtilizadores.ordenaJogadores(2, 1); Aviso.operacaoSucesso(); break;
    case 4: gestorUtilizadores.ordenaJogadores(2, -1); Aviso.operacaoSucesso(); break;
    case 5: gestorUtilizadores.ordenaJogadores(3, 1); Aviso.operacaoSucesso(); break;
    case 6: gestorUtilizadores.ordenaJogadores(3, -1); Aviso.operacaoSucesso(); break;
    case 7: gestorUtilizadores.ordenaJogadores(4, 1); Aviso.operacaoSucesso(); break;
    case 8: gestorUtilizadores.ordenaJogadores(4, -1); Aviso.operacaoSucesso(); break;
    case 9: break;
    default: System.out.println("Opcao Errada!\n");
    }
}
```

Figura 10 - Inicio da Ordenação

Na figura acima pode-se observar que através da escolha do utilizador os dados que são enviados, sendo o primeiro inteiro relativo ao tipo de ordenação e o segundo à ordem de crescimento. No exemplo abaixo poderemos observar esse desenvolvimento.

```
public boolean ordenaJogadores(int aNovaOrdem, int aOrdemCrescimento) {
    if (listaJogadores != null && listaJogadores.size() > 1) {
      Utilizadores.setTipoOrdena(aNovaOrdem);
      Utilizadores.setTipoCrescimento(aOrdemCrescimento);
      Collections.sort(listaJogadores);      // 4.º passo ordenamento
      return true;
    }
    return false;
  }
```

Figura 11 - Gestor Utilizadores ordem de Listagem

Na figura pode-se observar que as variáveis de classe são alteradas conforme os valores recebidos, assim condicionando a listagem no “ArrayList”.

3.4. Estruturas de Dados

Para este software que manipula os dados na memoria as estruturas mais utilizadas são "ArrayList" que são usados para guardar os mais diversos objetos do software e também é bastante utilizado o "array" estático de "Strings" para a utilização do quadro de jogo.

3.5. Armazenamento de Dados

Como já referido anteriormente a escolha para o armazenamento de dados inclinou-se para a gravação de ficheiros de objetos, devido ao facto pelo meu conhecimento técnico ser a base de dados mais portátil para a utilização da aplicação, assim usar a minha aplicação em outros ambientes de trabalho não tem qualquer obstáculo de instalação de uma base de dados relacional para que a aplicação funcione corretamente.

A vantagem de utilizar os dados carregados na memoria da máquina é que estes são mais rápidos de aceder a desvantagem seria a ocupação de recursos de memoria, mas os pequenos dados que são usados pela aplicação seria necessário bastante tempo para que esta se tornasse bastante pesada para um sistema operativo atual. Aliando ao facto de que existia uma livre opção no uso da manipulação de dados.

```
public boolean saveDados(GestorJogosGravados gestorJogos){ //gravar os dados dos objectos
    if(gereFichObj.abreFicheiroEscrita(DadosEstaticos.caminhoFichObj)){
        try{
            listaTudo.add(DadosEstaticos.quadro);
            listaTudo.add(gestorJogos);
            gereFichObj.escreveFicheiro(listaTudo);
            listaTudo.clear(); //se gravar mais que uma vez esta sempre nesta ordem..
            return true;
        } catch(Exception e){
            e.printStackTrace();
        }
    }
    return false;
}
```

Figura 12 - Gravação de Objetos

Do standard usado para a gravação de ficheiros, além que a este relatório é fornecido a documentação em "JavaDoc" explicando cada método de forma simples, nesta seção é importante referir o uso de um objeto "ArrayList" em "RAW" pois no mesmo ficheiro de objeto são gravados dois objetos diferentes, sabendo a ordem de gravação a quando a leitura dos dados apenas é necessário a realizar da mesma forma e fazer um cast a esses objetos como demonstra na figura 13.

```
ArrayList jogosG = systemManager.loadDados();
systemManager.fecharFiles();
ArrayList utilizadoresG = systemManager.loadDadosUtilizadores();

systemManager.fecharFiles();
if(jogosG != null) {
    DadosEstaticos.quadro = (String[][]) jogosG.get(0);
    if(jogosG.size() > 1)
        gestorJogos = (GestorJogosGravados) jogosG.get(1);
}else {
    jogosG = new ArrayList();
}
if(utilizadoresG != null) {
    if(utilizadoresG.get(0)!=null)
        gestorUtilizadores = (GestorUtilizadores) utilizadoresG.get(0);
    if(utilizadoresG.get(1)!= null) {
        log = (GestorMovimentos) utilizadoresG.get(1);
    }
```

Figura 13 - Leitura dos Dados

Como se observa cada objeto recebe os dados lidos através de um "cast" para a sua classe.

3.6. Procedimentos de Teste

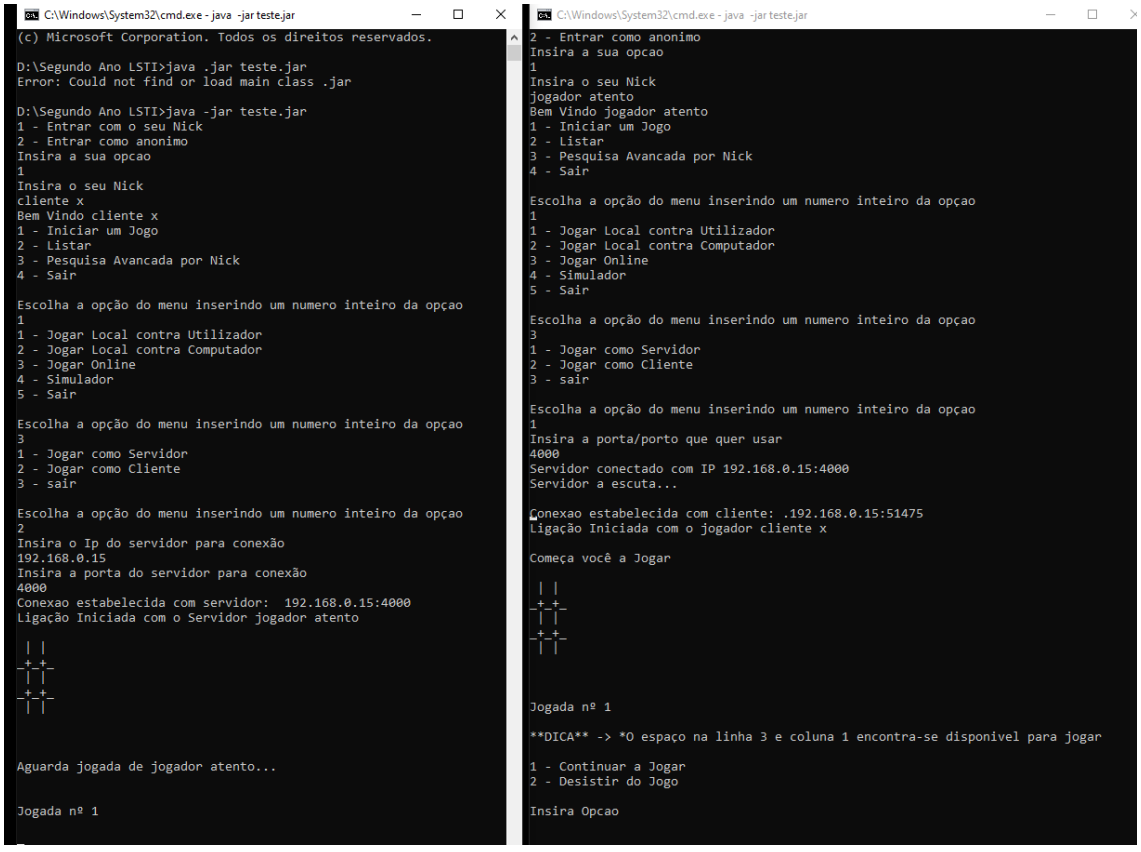

Figura 14 - Testes práticos da Aplicação

Para a verificação de erros na aplicação os métodos de teste da mesma foram os utilizados, inicialmente os testes foram aplicados na mesma máquina como está representado na figura 14, ao qual existem dois terminais da aplicação abertos e foi testado o jogo em rede (foco principal da aplicação).

A aplicação também testada para verificar o seu funcionamento offline tal como as suas listagens, ordenações e pesquisas. E por ultimo os testes com mais impacto foi a utilização da aplicação em comunicação com outras aplicações que seguem o mesmo protocolo, assim é verificado a existência de erros que podem por em causa a universalidade protocolar exigida neste projeto.

4. Conclusões

Este foi um projeto que me fez desenvolver bastante enquanto programador, pois no anterior projeto foram obtidos conhecimentos acerca da linguagem java e de base de dados, mas a ligação e conexão entre diferentes aplicações foi algo possível neste projeto. Desta forma também me possibilitou aprofundar e consolidar certos conhecimentos ao longo do desenvolvimento do projeto.

Contrariamente a relatórios anteriores decidi não criar um relatório com demasiada informação, pois documentação densa tende a algumas vezes confundir o leitor na perceção do funcionamento e construção do programa, assim foi entendido que os pontos principais acerca do funcionamento do mesmo foram abordados, demonstrando as logicas adotadas.

4.1. Forças

Este é um programa funcional e eficaz para os requisitos desejados, sendo que a sua portabilidade é o que o diferencia pois não existe a necessidade de instalação de base de dados relacional tornando assim simples a sua acessibilidade.

4.2. Limitações

Este trabalho ou projeto tem algumas limitações e uma encontra-se na sua eficiência, sei que poderia ter alguns dos métodos mais simplificados de forma a reutilizar em mais situações, dando como exemplo a classe de jogo em rede, sei que poderia diminuir o uso de “if´s”, tal como organizar por pequenos métodos melhor o código.

Uma limitação do software é que a IA é “dummy” ou seja não existe qualquer algoritmo de inteligência.

Uma limitação crucial que senti foi a falta de tempo, mesmo tendo conseguido realizar o projeto antes do tempo exigido, devido há existência de outros trabalhos para outras disciplinas não foi possível dedicar mais tempo na observação de bugs / erros no software, tal como na simplificação do código de forma a este ser o mais eficiente possível.

4.3. Trabalho Futuro

Para uma melhoria no projeto seria dedicação de mais tempo na resolução de uma simplificação do código e uma possível passagem do mesmo para um modo gráfico, deixando-o assim mais completo e mais acessível para um utilizador jogar confortavelmente.

5. Referências

YOUTUBE, 2021. Canal do utilizador “thenish khan”, vídeo Java socket programming – Simple client server program. Disponivel em: https://www.youtube.com/watch?v=-xKgxqG411c, acedido em: Maio de 2021, ultima atualização em: 2018.

STACKOVERFLOW, 2021. Plataforma de questões acerca da linguagem Java do site stackoverflow. Disponivel em: https://stackoverflow.com/questions/9481865/getting-the-ip-address-of-the-current-machine-using-java , acedido em: Maio 2021, ultima atualização: 2012.

JAVATPOINT, 2021. Categoria Java Networking do Menu Java. Disponível em: https://www.javatpoint.com/socket-programming, acedido em: Maio 2021, ultima atualização: 2017.

API JAVA, 2021. Documentação online da Oracle acerca da classe Socket. Disponível em: https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html, acedido em: Maio 2021, ultima atualização: 2017.

DEVMEDIA, 2021. Artigo online acerca de Java Socket. Disponivel em: https://www.devmedia.com.br/java-socket-entendendo-a-classe-socket-e-a-serversocket-em-detalhes/31894 , acedido em: Maio 2021, ultima atualização: 2015.

MARTINS, J, 2021. Projeto 1 da unidade curricular Programação III do Curso de LSTI.

MARTINS, J, 2021. Projeto 2 da unidade curricular Programação III do Curso de LSTI.

MARTINS, J, 2021. Projeto 1 da unidade curricular Programação IV do Curso de LSTI.

6. Anexos

1. Ficheiros Java e Extras
    - Ficheiros da pasta src com suas packages e ficheiros .java
    - Diagramas de Classes Gerados pela Eclipse Organizados por Package
2. Javadoc
3. Projecto2.jar (Executavel .jar)
4. Diagramas de Classes em Imagens