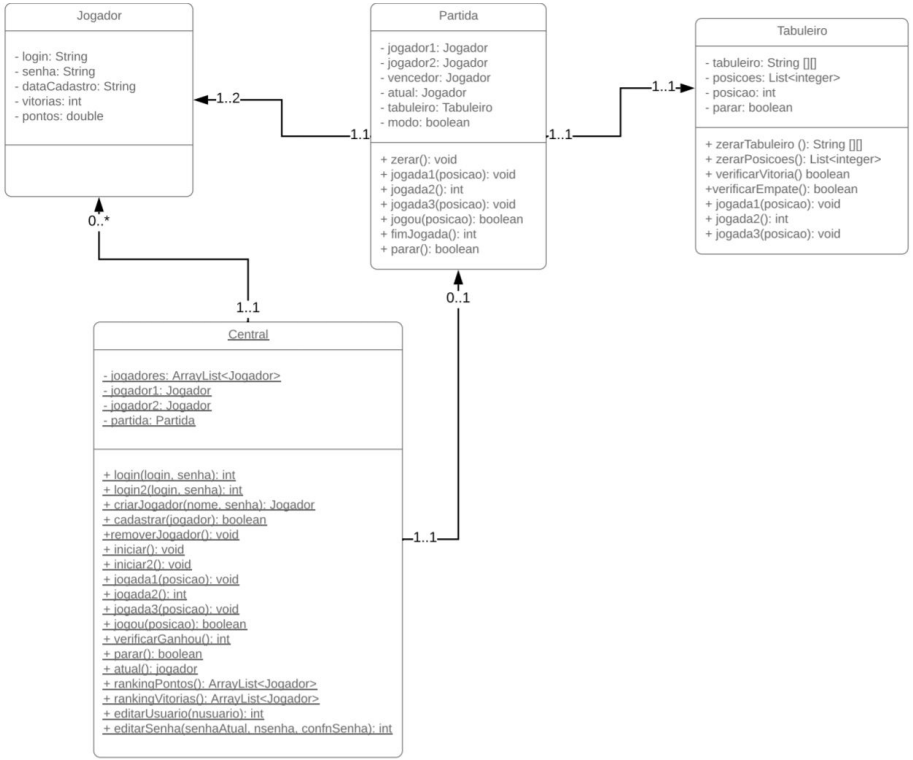
TIC TAC TOE

Alunos: Jônatas Tavares dos Santos
Lucas Silva Nascimento
Maria Gabriela Pereira da Silva
Maria Letícia Silva Mendes

Diagrama UML



Classe Tabuleiro

Classe Tabuleiro

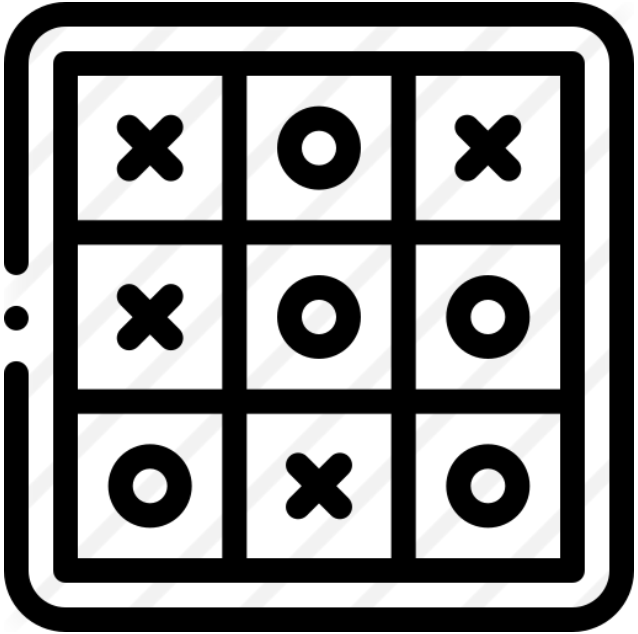
Funções

- zerarTabuleiro()
- zerarPosicoes()
- jogada1(posicao)
- jogada2()
- jogada3(posicao)
- verificarVitoria()
- verificarEmpate()

Classe Partida

Classe Partida

- Possui dois construtores para diferenciar o modo de jogo;
- Inicializa o tabuleiro e efetua as jogadas no mesmo;
- Possui importantes funções de verificação executadas a cada jogada;



Pontuação

Contra o Computador:

- Vitória: +1 ponto;
- Derrota: -0.5 ponto;
- Empate: +0.5 ponto.

Contra outro usuário:

- Vencedor: +4 pontos;
- Perdedor: -4 pontos.

Classe Jogador

Classe Jogador

- Representa o usuário que realiza partidas no sistema;
- Organiza importantes informações do usuário;
- Seus valores vão ser guardados;
- Utiliza as bibliotecas Date e SimpleDateFormat.

Classe Central

Classe Central

- Funciona como um intermédio entre as classes e a interface, por meio de suas funções;
- Os métodos importados na interface são estáticos;

Funções

Funções:		
Gets e sets geral	rankingVitorias	verificarGanhou
login	iniciar1	parar
login2	iniciar2	atual
criarJogador	jogada1	editarUsuario
cadastrar	jogada2	editarSenha
removerJogador	jogada3	
rankingPontos	jogou	

Cadastrar

```
93      public static Jogador criarJogador(String nome, String senha) {
94          Jogador jogador = new Jogador(nome, senha);
95
96          return jogador;
97      }
98
99      public static boolean cadastrar(Jogador jogador) {
100         boolean existe = false;
101         for (Jogador j : jogadores) {
102             if (j.getLogin().equals(jogador.getLogin())) {
103                 existe = true;
104                 break;
105             }
106         }
107         if (!existe && jogador.getSenha().length()>=6) {
108             Central.jogadores.add(jogador);
109             BancoDeDados.gerarArquivo(Central.jogadores);
110         }
111         return !existe;
112     }
```

Editar credenciais

```
275
276     public static int editarUsuario(String nusuario) {
            if (!Central.getJogadores().contains(nusuario)) {
278             getJogador1().setLogin(nusuario);
279             BancoDeDados.gerarArquivo(Central.getJogadores());
280             return 1;
281         } else {
282             return 0;
283         }
284     }
285
286     public static int editarSenha(String senhaAtual, String nsenha, String confnSenha) {
287         if (!senhaAtual.equals(Central.getJogador1().getSenha())) {
288             return 1;
289         } else if (nsenha.length() < 6) {
290             return 2;
291         } else if (senhaAtual.equals(Central.getJogador1().getSenha()) && nsenha.equals(confnSenha)) {
292             Central.getJogador1().setSenha(nsenha);
293             BancoDeDados.gerarArquivo(Central.getJogadores());
294             return 3;
295         } else {
296             return 0;
297         }
298     }
```

FUNÇÃO INICIAR

```
public static void iniciar() {
    Central.setPartida(new Partida(Central.getJogador1()));
    partida.setAtual(Central.getJogador1());
    partida.zerar();
    Jogador jogador1 = partida.getJogador1();
    for (Jogador j : jogadores) {
        if (j.getLogin().equals(jogador1.getLogin())) {
            j.setPontos(jogador1.getPontos());
        }
    }
    BancoDeDados.gerarArquivo(Central.jogadores);
}
```

FUNÇÃO RANKING

```
public static ArrayList<Jogador> rankingPontos() {
    double maior = 0;
    Jogador j1 = new Jogador(null, null);
    Jogador j2 = new Jogador(null, null);
    Jogador j3 = new Jogador(null, null);
    for (Jogador j : jogadores) {
        if (j.getPontos() > maior) {
            maior = j.getPontos();
            j1 = j;
        } else if (j.getPontos() == maior && maior == 0) {
            maior = j.getPontos();
            j1 = j;
        }

    }
    maior = 0;
    for (Jogador j : jogadores) {
        if (j.getPontos() > maior && j != j1) {
            maior = j.getPontos();
```

FUNÇÕES DE JOGADAS

```
public static void jogada1(int posicao) {
    if (partida.isModo() == false) {
        partida.setAtual(jogador1);
        partida.jogada1(posicao);
    } else {
        partida.jogada1(posicao);
        partida.setAtual(jogador2);
    }
}

public static int jogada2() {
    return partida.jogada2();
}

public static void jogada3(int posicao) {
    partida.jogada3(posicao);
    partida.setAtual(jogador1);
}
```

OUTRAS FUNCIONALIDADES

- removerJogador()
- rankingPontos()
- jogada1()
- jogada2()
- jogada3()
- jogou()
- verificarGanhou()
- parar()
- atual()

Persistência de Dados

Persistência de Dados

- Utilizamos um arquivo XML;
- Classe BancoDeDados lê e escreve dados no arquivo;
- A biblioteca XStream facilitou a serialização e deserialização;
- Os valores são atualizados de forma constante e rápida.

Estrutura do XML

```
<?xml version="1.0"?>
<Jogadores>
  - <Jogador>
        <login>MariaGabriela</login>
        <senha>244466666</senha>
        <dataCadastro>17/11/2019 15:30</dataCadastro>
        <vitorias>5</vitorias>
        <pontos>8.0</pontos>
    </Jogador>
  - <Jogador>
        <login>LucasSilva</login>
        <senha>lucas666</senha>
        <dataCadastro>19/11/2019 20:03</dataCadastro>
        <vitorias>3</vitorias>
        <pontos>7.5</pontos>
    </Jogador>
</Jogadores>
```

Interface grafica

Telas da interface

Telas do pacote	
Cadastro	Jogador 1
Dados da conta	Jogador 2
Editar senha	Menu
Editar usuario	Ranking
Instruções	Segundo jogador
Listar jogadores	Login

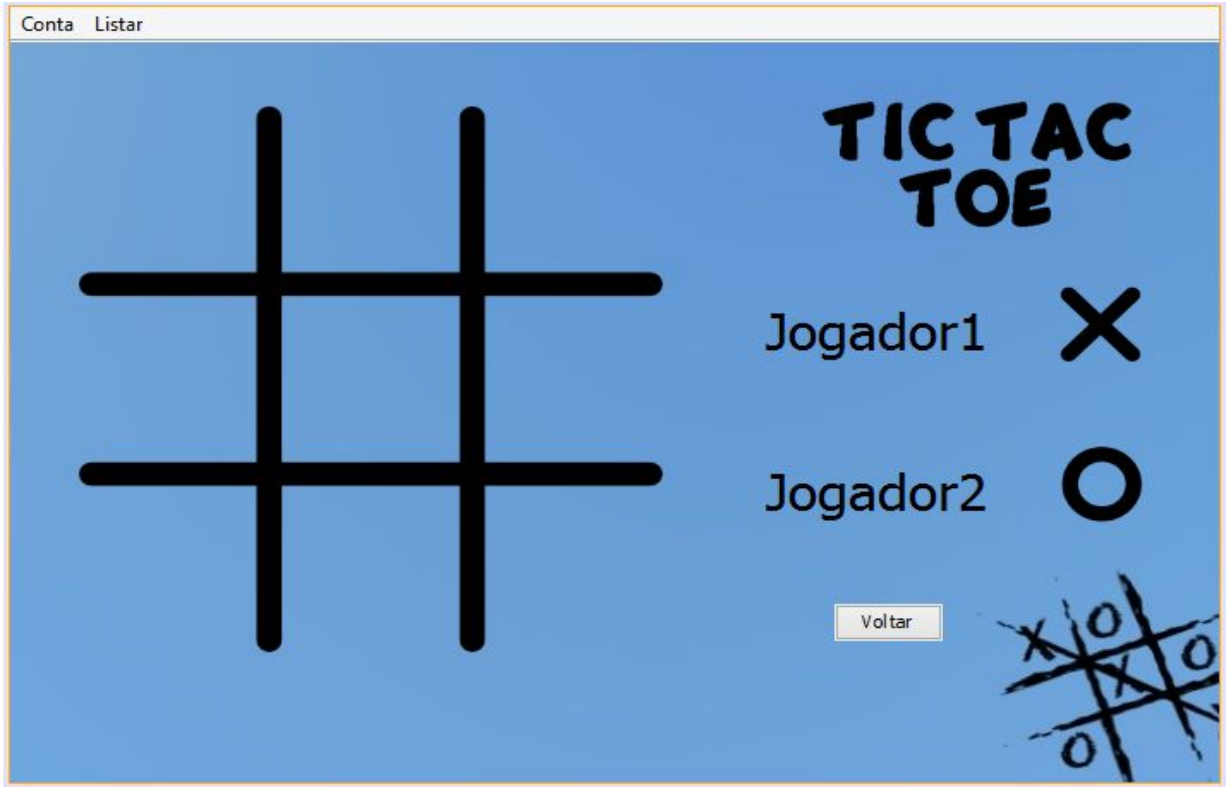
Listar


```
public class ListarJogadores extends javax.swing.JDialog {

    public ListarJogadores(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
        table();
    }

    public void table(){
        DefaultTableModel model = (DefaultTableModel) tabela.getModel();
        for(Jogador j : Central.getJogadores()){
            model.addRow(new Object[] {j.getLogin(),j.getPontos()});
        }
    }
```

Componentes da tela de 2 jogadores



Vamos ao vídeo

OBRIGADO!