
UNIVERSIDADE FEDERAL
DE SANTA CATARINA

# Trabalho Final Programação 3

Acadêmico: Gabriel de Souza Ferreira

Professor: Gian Ricardo Berkenbrock

Joinville

2018

# Introdução

O presente relatório tem como objetivo explicar resumidamente o trabalho final do curso de programação 3. Este foi concluído utilizando das práticas de programação em C++ aprendidas no curso, além de ferramentas para o desenvolvimento integrado, como a (IDE) QT Creator que ajudou na criação de gráficos e na interação do usuário com o aplicativo.

Além dos conceitos de programação, foi possível aprender técnicas de geolocalização e saber um pouco mais sobre o mercado de trabalho para a área de engenharia mecatrônica, por ser uma real necessidade de uma empresa que tive que solucionar. Foi possível também uma atualização sobre o que tem de novo, em tecnologia.

# Desenvolvimento

Em teoria este projeto parece ser simples, mas no caminhar foram aparecendo muitas dificuldades. Primeiramente, para se familiarizar com o QT Creator levou algum tempo, ao mesmo tempo que ele é uma ferramenta muito boa, é um tanto quanto complexa e demora um pouco para entender a lógica do programa.

A segunda grande dificuldade foi abrir e leitura o arquivo, que levou algum tempo ate aprender a utilizar uma classe do QT para fazer isto. Foram utilizadas basicamente duas classes, “QFile” para abrir o arquivo e “QTextStream” para ler.

jogador.cpp <Select Symbol>

```
64
65      QFile file(arquivo.c_str());
66      if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
67          return m;
68
69      QTextStream xin(&file);
70      std::vector<QStringList> v;
71      while (!xin.atEnd()) {
72          auto line = xin.readLine();
73          v.push_back(line.split(","));
74      }
75      file.close();
76
77      for(auto i=1; i<v.size(); i++){
78          estado e;
79
80          e.utc_date = v[i][3].toStdString();
81          e.utc_time = v[i][4].toStdString();
82          e.local_date = v[i][5].toStdString();
83          e.local_time = v[i][6].toStdString();
84          e.ms = v[i][7].toInt();
85          e.latitude = v[i][8].toDouble();
86          e.ns = v[i][9].toStdString();
87          e.longitude = v[i][10].toDouble();
88          e.ew = v[i][11].toStdString();
89          e.altitude = v[i][12].toDouble();
90          e.velocidade = v[i][13].toDouble();
91          e.heading = v[i][14].toDouble();
92          e.gx = v[i][15].toDouble();
93          e.gy = v[i][16].toDouble();
94          e.gz = v[i][16].toDouble();
95
96          m[stoi(v[i][1].toStdString())]._estados.push_back(e);
97
98
99      }
00
01      return m;
02  }
03
04
```

Após fazer isto, foi criado uma classe para armazenar e implementar funções que retornassem os dados necessários. Uma das funções mais difíceis de fazer a implementação foi a que retornava a distancia percorrida pelo atleta durante todo o jogo.

Testei diversas formulas ate chegar a uma que batesse o resultado, mas tiver que fazer algumas modificações pois a função só retornava a distancia entre dois pontos, então utilizei vetores para fazer os somatórios da distância dos pontos durante todo o jogo.

```
}


double Jogador::distancia_Total(){      //Função que retorna distancia total percorrida pelo jogad
    int tam = mj[jog]._estados.size();
    double distancia_T = 0;
    double d2r = 0.017453292519943295769236;
    double r = 6371.0;

    for(int i=1; i<tam; i++){
        double dlat = (mj[jog]._estados[i].latitude - mj[jog]._estados[i-1].latitude) * d2r;
        double dlon = (mj[jog]._estados[i].longitude - mj[jog]._estados[i-1].longitude) * d2r;
        double a = sin(dlat / 2.0) * sin(dlat / 2.0)
                    + cos(d2r * (mj[jog]._estados[i-1].latitude))
                    * cos(d2r * (mj[jog]._estados[i-1].latitude))
                    * sin(dlon / 2.0) * sin(dlon / 2.0);
        double c = 2.0 * atan2(sqrt(a), sqrt(1.0 - a));
        distancia_T += (r * c * 1000.0);

    }

    return distancia_T;
}
```

O desafio seguinte foi a leitura e o calculo com os horários, o usuário passa eles como strings e isso dificulta um pouco o cálculo com esses valores. Foi feita a transformação de string para inteiro e separado horas de minutos. Esses valores foram calculados separadamente (horas e minutos) e depois retornados para serem impressos.

```
double Jogador::tempo_2_Hr(){
    int hrt = 0;
    int H = 0;

    if(hrf2 < hri2){                    //Total de horas segundo tempo
        hrt = ((hrf2 - hri2) + 24.0);
    }
    else{
        hrt = (hrf2 - hri2);
    }

    if(mini2 == 0 || hri2 == hrf2){
        H = hrt;
    }
    else{
        H = hrt - 1;
    }

    return H;

}

double Jogador::tempo_1_Min(){
    int tmin=0;

    if(minf1 < mini1){                  //Total de minutos primeiro tempo
        tmin = ((minf1 - mini1) + 60);
    }
    else{
        tmin = (minf1 - mini1);
    }

    return tmin;
}

double Jogador::tempo_2_Min(){
    int tmin=0;

    if(minf2 < mini2){                  //Total de minutos segundo tempo
        tmin = ((minf2 - mini2) + 60);
    }
```

Por último os gráficos que foram os que deram mais trabalho para implementar, levou algum tempo para entender a logica e fazer com que eles trabalhassem com os valores do arquivo. De forma parecida com o calculo das distancias foi utilizado vetores para fazer com que o gráfico recebesse todos os pontos passados em latitude e longitude.

```
//Graficos:

void MainWindow::makePlot()
{

    numj = ui->comboBox_Jogador->currentText().toInt(); // Armazena o valor do jogador selecionado

    ui->customPlot->clearPlottables();
    ui->customPlot->clearItems();
    ui->customPlot->clearGraphs();
    ui->customPlot->clearFocus();
    ui->customPlot->clearMask();

    // create empty curve objects:
    QCPCurve *fermatSpiral1 = new QCPCurve(ui->customPlot->xAxis, ui->customPlot->yAxis);

    // generate the curve data points:
    QVector<QCPCurveData> dataSpiral1(tam);

        for(int i=0; i<tam; i++){

            dataSpiral1[i] = QCPCurveData(i, _m[numj]._estados[i].latitude, _m[numj]._estados[i].longitude);
        }



    // pass the data to the curves; we know t (i in loop above) is ascending, so set alreadySorted=true (saves
    fermatSpiral1->data()->set(dataSpiral1, true);

    // color the curves:
    fermatSpiral1->setPen(QPen(Qt::blue));
    // set some basic ui->customPlot config:
    ui->customPlot->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom | QCP::iSelectPlottables);
    ui->customPlot->axisRect()->setupFullAxesBox();
    ui->customPlot->rescaleAxes();

    ui->customPlot->replot();
```

# Conclusão

Ao fazer este trabalho consegui exercitar muito os conceitos de programação Orientada a Objetos aprendidos no curso além de ter uma experiencia muito próxima ao mercado de trabalho, preparando da melhor forma, na pratica. Também foi possível conhecer uma nova fermenta de desenvolvimento, o QT Creator que com muita certeza ajudara em futuros possíveis projetos. A programação orientada a objetos se mostrou muito útil no desenvolvimento desse projeto, tornado o código mais organizado e mais legivel, mesmo sendo um código mais extenso.