Analog Clock Example
Qt Widget 5.14.1

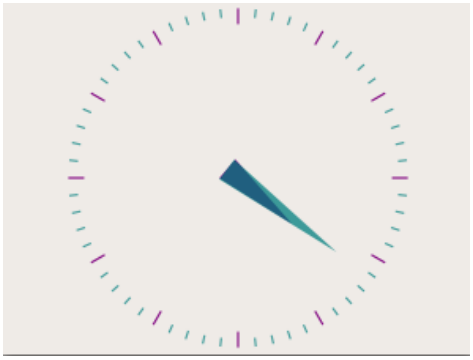

Abb.1 Analog Clock Example, Qt Documentation

Neben dem „Digital Clock Example“, also einer digitalen Uhrenanzeige, gibt es noch die Möglichkeit eine analoge Uhrenanzeige - im folgenden „Analog Clock Example“ - für das Projekt zu nutzen. Hierzu gibt die Qt Dokumentation her, dass das „Analog Clock Example“ die Anzeige von Stunden, Minuten und Sekunden ermöglicht.

Die `AnalogClock` Klasse bietet genau diese Funktionalität:

```
class AnalogClock : public QWidget //wir nutzen hier die QWidget Subklasse
{
    Q_OBJECT

public:
    AnalogClock(QWidget *parent = nullptr);

protected:
    void paintEvent(QPaintEvent *event) override;//und manipulieren die
                                                 // paintEvent()-Funktion.
};                                               // QPaintEvent sorgt fuer
                                                 // die Aktualsierung des
                                                 // Widgets.
```

Durch die Manipulation der Standard paintEvent() – Funktion, haben wir nun die Möglichkeit, unser „Watch-Face“, also Uhrenlayout wunschweise zu modifizieren.

```
AnalogClock::AnalogClock(QWidget *parent)
    : QWidget(parent)
{
    QTimer *timer = new QTimer(this);
    connect(timer, &QTimer::timeout, this, QOverload<>::of(&AnalogClock::update));
    timer->start(1000); // Setzen des Sekundenzeigers und Verknüpfung mit der
                        // update()-Funktion.
    setWindowTitle(tr("Analog Clock"));
    resize(200, 200);
}
```

Durch die Verknüpfung des Sekundenzeigers mit der standardmäßigen update() – Funktion können wir nun unser Sekundenzeiger aktualisieren, wenn die timeout() – Funktion (alternativ: start(), stop()) angesprochen wird - laut Qt Documents alle fünf Sekunden. Die update() – Funktion hat den Vorteil, dass eine Aktualisierung oder eine „Neuzeichnung“ der Zeiger („repaint“) nicht sofort geschieht, sondern erst dann ausgelöst wird, wenn Qt aus der „main event loop“ (also der Haupt-Event-Schleife) zurückgeliefert wird. Dies soll vor allem die Geschwindigkeit optimieren und Flackern vermeiden.

Die paintEvent() – Funktion wird zu Anfang, also wenn das Widget gestartet wird, aufgerufen und immer dann, wenn die update() – Funktion angesprochen wird. Durch die Verknüpfung der update() – Funktion mit der timeout() – Funktion geschieht dies im fünft Sekunden – Takt.

Die ersten Schwierigkeiten liegen jedoch hier schon auf der Hand: die timeout() – Funktion ist ein „privat signal“ und somit nicht vom Nutzer steuerbar. Für eine genaue Zeitmessung im Rallye Sport also zunächst ungeeignet. Alternativ lässt sich jedoch noch die start()- und stop() – Funktion betrachten, wobei start () wiederrum an ein Intervall gebunden ist: die start(int msec) – Funktion, übergibt ein Intervall in Millisekunden. Ob dies für unser Projekt nutzbar ist, muss erörtert werden.

Neben dem Sekundenzeiger müssen auch Stunden- und Minutenzeiger implementiert werden. Diese werden mithilfe der QPoints Klasse, welche einen Punkt in einer x-y-Koordinate (aufrufbar mit x() oder y()) symbolisiert und der QColor Klasse, also Farben (RGB oder CMYK) abbildet, umgesetzt.

```
void AnalogClock::paintEvent(QPaintEvent *)
{
    static const QPoint hourHand[3] = { // setzten der Koordinaten fuer die Stunde
        QPoint(7, 8),
        QPoint(-7, 8),
        QPoint(0, -40)
    };
    static const QPoint minuteHand[3] = {// setzten der Koordianten fuer die Minute
        QPoint(7, 8),
        QPoint(-7, 8),
        QPoint(0, -70)
    };

    QColor hourColor(127, 0, 127);  // farbliche Gestaltung fuer den Stundenzeiger
    QColor minuteColor(0, 127, 127, 191); // farbliche Gestaltung fuer den Minuten-
                                          // zeiger.
    int side = qMin(width(), height()); //Groesse des Fensters setzten
    QTime time = QTime::currentTime();
```

Die Inhalte werden dann mit der QPainter Klasse umgesetzt. QPainter können auf alle QPainterDevice Klassen (Basisklasse von Objekten, auf die die QPainter Klasse zugreifen kann) angewandt werden.

```
QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);//Aktivierung
                                                  // Antialaising
    painter.translate(width() / 2, height() / 2); //Anpassung des Widgets
    painter.scale(side / 200.0, side / 200.0);    // ans Fenster
```

Um diagonal Linien der Zeiger sauber darzustellen, ist die Aktivierung von Antiailasing durchaus sinnvoll. Renderanweisungen wie in diesem Code – Beispiel nutzen Spezifizierungen der QPainter Klasse, in diesem Fall QPainter::Antialiasing. Zudem ist die Anpassung des Widgets unumgänglich und wird im „Analog Clock Example“ mit einer Skalierung von 100 auf der x-Koordinate und -100 y-Koordinate umgesetzt. Die Dokumentation gibt uns zudem den Hinweis, mit der translate() – Funktion eine dauerhafte Fixierung des Widgets umzusetzten, um den Code so einfach wie möglich zu halten.
Um den Stundenzeiger und den Minutenzeiger umzusetzen, kann folgender Code implementiert werden:

```
painter.setPen(Qt::NoPen); // Wir wollen keine durchgezogenen Linien
                           // auf unserem Uhren Layout.
painter.setBrush(hourColor); // Setzten der Stunden
```

```
painter.save();
painter.rotate(30.0 * ((time.hour() + time.minute() / 60.0))); //Rotation erzeugen
painter.drawConvexPolygon(hourHand, 3); // Zeichnen
painter.restore(); // Veränderung sichern

painter.setPen(hourColor);

    for (int i = 0; i < 12; ++i) {   //Makierungen für jede Stunde setzten.
        painter.drawLine(88, 0, 96, 0);
        painter.rotate(30.0); //Rotieren des Koordinatensystems fuer nächste Aktion
    }


// und das gleiche Prinzip fuer den Minutenzeiger

painter.setPen(Qt::NoPen);
    painter.setBrush(minuteColor);

    painter.save();
    painter.rotate(6.0 * (time.minute() + time.second() / 60.0));
    painter.drawConvexPolygon(minuteHand, 3);
    painter.restore();

    painter.setPen(minuteColor);

    for (int j = 0; j < 60; ++j) {
        if ((j % 5) != 0) // Makrierungen fuer jede fuenf Minuten setzten.
        painter.drawLine(92, 0, 96, 0);
        painter.rotate(6.0);
    }
}
```

Ob die Nutzung einer analogen Uhr für eine Geschwindigkeitsmessung bis auf hundertstel Sekunden im Hinblick auf Genauigkeit als sinnvoll erscheint, muss jeder selbst bewerten. Aus meiner Sicht jedoch, scheint die analoge Uhr den Anforderungen unseres Projektes nicht ausreichend zu erfüllen. Hier ist eine digitale Lösung zu bevorzugen.

Name: Sammy Sabih Al Bayati
Datum: 31.03.2020
Tasktitel: Übersichtserstellung: Graphische Qt Elemente, die besonders für Uhren geeignet sind

Quellen

Qt. (o. J.). Analog Clock Example | Qt Widgets 5.14.1. Abgerufen 30. März 2020, von https://doc.qt.io/qt-5/qtwidgets-widgets-analogclock-example.html