```
#ifndef GENERATOR_H
#define GENERATOR_H

#include <cmath>

enum class RodzajSygnalu { Skok, Sinusoida, Prostokatny };

class Generator
{
    RodzajSygnalu rodzaj;
    double A, T, p, czasAktywacji=0.0;

public:
    Generator(RodzajSygnalu r = RodzajSygnalu::Skok, double a = 0.0, double t = 0.0, double
pp = 0.0, double czasAkt = 0.0);
    double generuj(double czas);
    RodzajSygnalu getRodzaj();
    double getAmplituda();
    double getOkres(){return T;};
    double getWypelnienie(){return p;}
    void setRodzaj(RodzajSygnalu r){rodzaj=r;}
    void setAmplituda(double a){A=a;}
    void setOkres(double o){T=o;}
    void setWypelnienie(double w){p=w;}
    void setCzasAktywacji(double ca){czasAktywacji=ca;}
};

#endif // GENERATOR_H

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "symulator.h"
#include "wykresy.h"
#include <QMainWindow>
#include <QTimer>
#include <QtWidgets/QApplication>
#include <QtWidgets/QMainWindow>
#include <QtCharts/QChartView>
#include <QtCharts/QLineSeries>
#include <QtCharts/QValueAxis>
#include "warstwauslug.h"
#include "oknogenerator.h"
#include "oknoregulator.h"
#include "oknoobiektarx.h"
#include <QFileDialog>
#include <QTextStream>
#include <QFile>
```

```
QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE
class WarstwaUslug;

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr,
               WarstwaUslug* program = nullptr);
    ~MainWindow();
    void UstawienieGUI();

private slots:
    void on_Start_clicked();
    void on_Stop_clicked();
    void on_Interwal_textChanged(const QString &arg1);
    void on_UstawieniaGeneratora_clicked();
    void on_UstawieniaObiektuARX_clicked();
    void on_UstawieniaRegulatora_clicked();
    void PokazWykres(symulator* s);
    void Blad();
    void on_Reset_clicked();
    void on_Zapisz_clicked();
    //void on_ResetID_clicked();
    void on_Wczytaj_clicked();

private:
    Ui::MainWindow *ui;
    QTimer *simulationTimer;
    WarstwaUslug* usluga ;
    OknoGenerator* okno_gen;
    OknoRegulator* okno_reg;
    OknoObiektARX* okno_obiekt;
    double czas;
    int interwalCzasowy=0;
    bool isSimulationRunning=true;
    Wykresy* wykres;
    void UstawienieLayout();
    QVBoxLayout *layout[4];
    void fileSelected(QString fileName);
    QFile plik;
    QString nazwaqPliku;
    void obslugaZapisu();
```

```
    QPushButton *Wczytaj;
};
#endif // MAINWINDOW_H

#ifndef OBIEKTARX_H
#define OBIEKTARX_H

#include <vector>
#include <random>
#include <deque>

class ObiektARX
{
    double k, z;
    std::vector<double> a, b;
    //std::vector<double> b;
    std::deque<double> ui, yi;
    //std::deque<double> yi;
    double mean, stdev;
    //double stdev;
    std::mt19937 generator;
    std::normal_distribution<double> zaklocenie;
    void zaktualizujZaklocenie();
public:
    ObiektARX();
    ObiektARX(double kk, double zz, std::vector<double> aa, std::vector<double> bb,
std::mt19937 gen, double mean = 0.3, double stdev = 0.0);
    void setZaklocenie(double newMean, double newStdev);
    double getZaklocenie();
    double obliczWyjscie(double uii);
    void setOpoznienie(double o){k=o;}
    void setWielomianA(std::vector<double> wa){a=wa;}
    void setWielomianB(std::vector<double> wb){b=wb;}
    double getOpoznienie(){return k;}
    std::vector<double> getWielomianA(){return a;}
    std::vector<double> getWielomianB(){return b;}
    void setGenerator(std::mt19937 g){generator = g;}
    void setMean(double m){mean =m;}
    void setStdev(double s){stdev =s;}
};

#endif // OBIEKTARX_H

#ifndef OKNOGENERATOR_H
#define OKNOGENERATOR_H

#include <QDialog>
#include "generator.h"
```

```
#include "warstwauslug.h"
#include <QPropertyAnimation>


namespace Ui {
class OknoGenerator;
}

class OknoGenerator : public QDialog
{
    Q_OBJECT
public:
    explicit OknoGenerator(QWidget *parent = nullptr);
    ~OknoGenerator();
    void setWarstwaUslug(WarstwaUslug* w){usluga=w;};
    void UstawienieGUI();
    void UstawienieOkna();
private slots:
    void on_RodzajeSygnalu_clicked();
    void on_RodzajeSygnalu_triggered(QAction *arg1);
    void on_ZatwierdzenieUstawien_accepted();
private:
    Ui::OknoGenerator *ui;
    Generator *gen;
    WarstwaUslug *usluga;
};

#endif // OKNOGENERATOR_H

#ifndef OKNOOBIEKTARX_H
#define OKNOOBIEKTARX_H

#include <QDialog>
#include "obiektarx.h"
#include "warstwauslug.h"
#include <QPropertyAnimation>


namespace Ui {
class OknoObiektARX;
}

class OknoObiektARX : public QDialog
{
    Q_OBJECT
public:
    explicit OknoObiektARX(QWidget *parent = nullptr);
    ~OknoObiektARX();
```

```
    void setWarstwaUslug(WarstwaUslug* w){usluga=w;};
    void UstawienieARX();
private slots:
    void on_ZatwierdzenieUstawien_accepted();
private:
    Ui::OknoObiektARX *ui;
    ObiektARX *obiekt;
    WarstwaUslug *usluga;
    std::mt19937 generator;
};

#endif // OKNOOBIEKTARX_H

#ifndef OKNOREGULATOR_H
#define OKNOREGULATOR_H

#include <QDialog>
#include "regulator.h"
#include "warstwauslug.h"
#include <QPropertyAnimation>

namespace Ui {
class OknoRegulator;
}

class OknoRegulator : public QDialog
{
    Q_OBJECT
public:
    explicit OknoRegulator(QWidget *parent = nullptr);
    ~OknoRegulator();
    void setWarstwaUslug(WarstwaUslug* w){usluga=w;};
    void UstawienieReg();
private slots:
    void on_ZatwierdzenieUstawien_accepted();
private:
    Ui::OknoRegulator *ui;
    Regulator *reg;
    WarstwaUslug *usluga;
};

#endif // OKNOREGULATOR_H

#ifndef REGULATOR_H
#define REGULATOR_H
```

```
class Regulator
{
   double wartoscZadana, wzmocnienieP, stalaI, stalaD, Uchyb, WczesniejszyUchyb,
sumaUchybow, WartoscSterujaca, nastawaP, nastawaI, nastawaD;
public:
   Regulator();
   Regulator(double kp, double ki, double kd);
   void setWartoscZadana(double war);
   double getWartoscZadana() const;
   void aktualizujUchyb(double wartoscRegulowana);
   double obliczSterowanie();
   double getWartoscSterujaca();
   double getUchyb();
   double getNastawaP();
   double getNastawaI();
   double getNastawaD();
   double getWzmocnienie(){return wzmocnienieP;}
   double getStalaI(){return stalaI;}
   double getStalaD(){return stalaD;}
   void ZerowanieNastawaI();
   void ZerowanieNastawaD();
   void ZerowanieNastawaP();
   void setWartoscSterujaca(double on){WartoscSterujaca=on;}
   void setWzmocnienie(double w){wzmocnienieP=w;}
   void setStalaI(double si){stalaI=si;}
   void setStalaD(double sd){stalaD=sd;}
};

#endif // REGULATOR_H

#ifndef SYMULATOR_H
#define SYMULATOR_H


#include "Generator.h"
#include "Regulator.h"
#include "ObiektARX.h"

class symulator
{

   Generator generator;
   Regulator regulator;
   ObiektARX obiekt;
   double poprzednieWyjscie = 0;
   double wyjscieObiektu = 0;
   double lastRegulatorValue = 0;
   double lastObjectOutput = 0 ;
```

```
public:
    symulator();
    symulator(Generator g, Regulator r, ObiektARX o);
    double symulujKrok(double czas);
    double getWartoscZadana();
    double getZaklocenie();
    double getSterowanie();
    double getWyjscieObiektu();
    void setGenerator(Generator g);
    void setRegulator(Regulator r);
    void setObiektARX(ObiektARX o);
    Generator getGenerator();
    Regulator getRegulator();
    ObiektARX getObiektARX();
    void setLastRegulatorValue(double value) { lastRegulatorValue = value; }
    void setLastObjectOutput(double value) { lastObjectOutput = value; }
    double getLastRegulatorValue()  { return lastRegulatorValue; }
    double getLastObjectOutput()  { return lastObjectOutput; }
    void setWyjscieObiektu(double wo){wyjscieObiektu=wo;};
};

#endif // SYMULATOR_H

#ifndef WARSTWAUSLUG_H
#define WARSTWAUSLUG_H

#include <QObject>
#include <QMainWindow>
#include <QDebug>
#include "symulator.h"
#include <QFileDialog>
#include <QTextStream>
#include <QFile>
#include <QMessageBox>
#include <QTimer>
#include <QtWidgets/QApplication>
#include <QtWidgets/QMainWindow>
#include "symulator.h"
#include <QtWidgets>

class MainWindow;

class WarstwaUslug : public QObject
{
    Q_OBJECT
public:
    explicit WarstwaUslug(QObject *parent = nullptr);
    void SprawdzenieWszystkichDanych(double i);
```

```
    void SprawdzenieRegulatora(Regulator* r);
    void SprawdzenieGeneratora(Generator* g);
    void SprawdzenieObiektu(ObiektARX* o);
    void setGUI(MainWindow* ui = nullptr){ GUI = ui; }
    void setSymulator(symulator* sym){s=sym;};
    symulator* getSymulator(){return s;};
    void konfiguracjaZapis();
signals:
    void PoprawneDane(symulator* s);
    void BledneDane();
    void sygnalZapisano();
private:
    symulator* s;
    double interwal;
    MainWindow* GUI = nullptr;
    WarstwaUslug* usluga ;
};

#endif // WARSTWAUSLUG_H

#ifndef WYKRESY_H
#define WYKRESY_H

#include <QObject>
#include <QWidget>
#include <QChart>
#include <QLineSeries>
#include <QChartView>
#include <QValueAxis>
#include <QVBoxLayout>
#include "symulator.h"

class Wykresy : public QObject
{
    Q_OBJECT
public:
    explicit Wykresy(QWidget *parent = nullptr);
    void inicjalizacjaWykresuWartosciZadanej(QVBoxLayout *layout);
    void inicjalizacjaWykresuUchybu(QVBoxLayout *layout);
    void inicjalizacjaWykresuPID(QVBoxLayout *layout);
    void inicjalizacjaWykresuWartosciSterowania(QVBoxLayout *layout);
    void WykresWartosciZadanej();
    void WykresUchybu();
    void WykresPID();
    void WykresWartosciSterowania();
    void AktualizujWykresy();
    void InicjalizujWykresy(QVBoxLayout *layout[4]);
    void setSymulator(symulator* sym){s=sym;};
```

```
    void wyczyscLayout(QLayout* layout);
    void ResetCzas(){czas=0;};
    void ResetujWykresy();
private:
    QLineSeries *seria[7];
    QChart *wykres[4];
    QValueAxis *osX[4];
    QValueAxis *osY[4];
    QChartView *Widok[4];
    double czas;
    QWidget* parent;
    symulator* s;
signals:
};

#endif // WYKRESY_H

#include "generator.h"

Generator::Generator(RodzajSygnalu r, double a, double t, double pp, double czasAkt)
    : rodzaj(r), A(a), T(t), p(pp), czasAktywacji(czasAkt) {}

double Generator::generuj(double czas) {
    switch (rodzaj) {
    case RodzajSygnalu::Skok:
        return (czas >= czasAktywacji) ? A : 0;
    case RodzajSygnalu::Sinusoida:
        return A * sin((2 * 3.14159265359 / T) * fmod(czas, T));
    case RodzajSygnalu::Prostokatny:
        return (fmod(czas, T) < p * T) ? A : 0;
    default:
        return 0.0;
    }
}
RodzajSygnalu Generator::getRodzaj(){
    return rodzaj;
}
double Generator::getAmplituda(){
    return A;
}

#include "mainwindow.h"
#include "warstwauslug.h"
#include "oknogenerator.h"
#include "oknoregulator.h"
#include "oknoobiektarx.h"

#include <QApplication>
```

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    WarstwaUslug* usluga = new WarstwaUslug;
    MainWindow w(nullptr,usluga);
    w.show();
    return a.exec();
}

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "oknogenerator.h"
#include "oknoregulator.h"
#include "oknoobiektarx.h"
#include <QDebug>
#include <QMessageBox>
#include <QGraphicsDropShadowEffect>

std::random_device srng;
std::mt19937 rng;


void MainWindow::UstawienieLayout(){
    qDebug() << "Ustawianie layoutu";
    for(int i=0;i<4;i++){
        layout[i] = new QVBoxLayout();
        layout[i]->setContentsMargins(0, 0, 0, 0);
    }
    ui->wykresWarZad->setLayout(layout[0]);
    ui->WykUchyb->setLayout(layout[1]);
    ui->WykPID->setLayout(layout[2]);
    ui->WykSter->setLayout(layout[3]);
    qDebug() << "Layout ustawiony";
}

MainWindow::MainWindow(QWidget *parent, WarstwaUslug *prog)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
    , simulationTimer(new QTimer(this))
    ,usluga(prog)
    ,czas(0.0)
{
    qDebug() << "MainWindow constructor";
    usluga->setGUI(this);
    ui->setupUi(this);
    wykres=new  Wykresy();
    okno_gen = new OknoGenerator(this);
```

```
    okno_gen->setWarstwaUslug(usluga);
    okno_reg = new OknoRegulator(this);
    okno_reg->setWarstwaUslug(usluga);
    okno_obiekt = new OknoObiektARX(this);
    okno_obiekt->setWarstwaUslug(usluga);

    this->showMaximized();
    simulationTimer = new QTimer(this);

    UstawienieGUI();
   // fileSelected("");


    //symulator* s=new symulator;

    UstawienieLayout();
    wykres->InicjalizujWykresy(layout);
    connect(usluga, &WarstwaUslug::PoprawneDane, this, &MainWindow::PokazWykres);
    connect(usluga, &WarstwaUslug::BledneDane, this, &MainWindow::Blad);
    connect(usluga, &WarstwaUslug::sygnalZapisano, this, &MainWindow::obslugaZapisu);
    qDebug() << "Sygnały połączone";
    Wczytaj = new QPushButton("Wczytaj konfigurację", this);
    connect(Wczytaj, &QPushButton::clicked, this, &MainWindow::on_Wczytaj_clicked);
    //ui->Layout->addWidget(Wczytaj);  // Dodanie do układu przycisków
}


MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_Start_clicked()
{
    qDebug() << "Przycisk Start kliknięty";
    usluga->SprawdzenieWszystkichDanych(interwalCzasowy);
    qDebug() << "Sprawdzenie wszystkich danych zakończone";
}

void MainWindow::on_Stop_clicked()
{
    qDebug() << "Przycisk Stop kliknięty";
    simulationTimer->stop();
    qDebug() << "Timer zatrzymany";
}

void MainWindow::on_Interwal_textChanged(const QString &arg1)
```

```
{
    qDebug() << "Zmiana interwału czasowego";

    interwalCzasowy=arg1.toInt();
    qDebug() << "Nowy interwał czasowy: " << interwalCzasowy;

}

void MainWindow::PokazWykres(symulator* s) {
    if (!s) {
        qDebug() << "symulator* s jest nullptr!";
        return;
    }

    qDebug() << "Inicjalizacja wykresów";
     // Ustawienie symulatora
    wykres->setSymulator(s); // Ustawienie symulatora dla obiektu wykresów

    // Inicjalizowanie wykresów dla każdego layoutu


    // Połączenie timera z funkcją aktualizującą wykresy

    connect(simulationTimer, &QTimer::timeout, this, [=]() {
        wykres->WykresWartosciZadanej();
    });
    connect(simulationTimer, &QTimer::timeout, this, [=]() {
        wykres->WykresUchybu();
    });
    connect(simulationTimer, &QTimer::timeout, this, [=]() {
        wykres->WykresPID();
    });
    connect(simulationTimer, &QTimer::timeout, this, [=]() {
        wykres->WykresWartosciSterowania();
    });

    // Uruchomienie timera, jeśli interwał czasowy jest większy niż 0
    if (interwalCzasowy > 0) {
        simulationTimer->start(interwalCzasowy);  // Rozpoczęcie timera
        qDebug() << "Timer uruchomiony z interwałem " << interwalCzasowy;
    } else {
        qDebug() << "Błędny interwał czasowy";
    }
}



void MainWindow::Blad(){
```

```
    qDebug() << "Wywołano funkcję Blad()";
    QMessageBox::warning(this, "Ostrzeżenie", "Nie uzupełniłeś Wszystkich Danych");
}

void MainWindow::on_UstawieniaGeneratora_clicked()
{
    qDebug() << "Ustawienia Generatora kliknięte";
    okno_gen->exec();
}

void MainWindow::on_UstawieniaObiektuARX_clicked()
{
    qDebug() << "Ustawienia Obiektu ARX kliknięte";
    okno_obiekt->exec();
}

void MainWindow::on_UstawieniaRegulatora_clicked()
{
    qDebug() << "Ustawienia Regulatora kliknięte";
    okno_reg->exec();
}

void MainWindow::on_Reset_clicked()
{
    // Zatrzymanie timera i odłączenie sygnałów
    disconnect(simulationTimer, nullptr, nullptr, nullptr);
    simulationTimer->stop();

    // Resetowanie wykresów i czasu
    wykres->ResetujWykresy();
    wykres->ResetCzas();

    // Resetowanie zmiennych symulatora
    symulator* sym = usluga->getSymulator();
    if (sym) {
        sym->getRegulator().ZerowanieNastawaP();
        sym->getRegulator().ZerowanieNastawaI();
        sym->getRegulator().ZerowanieNastawaD();
        sym->setWyjscieObiektu(0);
        sym->setLastRegulatorValue(0);
        sym->setLastObjectOutput(0);

        // Reset generatora sygnału
        Generator gen = sym->getGenerator();
        gen.setAmplituda(0);
        gen.setOkres(1);
        gen.setWypelnienie(0.5);
        sym->setGenerator(gen);
```

```
        // Reset regulatora
        Regulator reg;
        sym->setRegulator(reg);

        // Reset obiektu ARX
        ObiektARX obiekt;
        sym->setObiektARX(obiekt);
    }
    // Resetowanie GUI
    czas = 0;
    interwalCzasowy = 0;
    isSimulationRunning = false;

    // Ponowna inicjalizacja wykresów
    wykres->InicjalizujWykresy(layout);

    qDebug() << "Symulacja i wykresy zostały zresetowane.";
}

void MainWindow::UstawienieGUI(){
    QString buttonStyleStart =
        "QPushButton {"
        "    background-color: white;"
        "    color: black;"
        "    border-radius: 10px;"
        "    border: 2px solid black;"
        "    padding: 5px;"
        "    transition: 0.5s ease-in-out;"

        "}"
        "QPushButton:hover {"
        "    background-color: green;"
        "    color: white;"
        "    padding: 8px;"
        "    border: 3px solid #fff;"
        "}";
    QString buttonStyleStop =
        "QPushButton {"
        "    background-color: white;"
        "    color: black;"
        "    border-radius: 10px;"
        "    border: 2px solid black;"
        "    padding: 5px;"
        "    transition: 0.5s ease-in-out;"
        "}"
        "QPushButton:hover {"
        "    background-color: red;"
```

```
    "    color: white;"
    "    padding: 8px;"
    "    border: 3px solid #fff;"
    "}";
QString buttonStyleReset =
    "QPushButton {"
    "    background-color: white;"
    "    color: black;"
    "    border-radius: 10px;"
    "    border: 2px solid black;"
    "    padding: 5px;"
    "    transition: 0.5s ease-in-out;"
    "}"
    "QPushButton:hover {"
    "    background-color: orange;"
    "    color: white;"
    "    padding: 8px;"
    "    border: 3px solid #fff;"
    "}";
QString buttonStyleReszta =
    "QPushButton {"
    "    background-color: white;"
    "    color: black;"
    "    border-radius: 10px;"
    "    border: 2px solid black;"
    "    padding: 5px;"
    "    transition: 0.5s ease-in-out;"
    "}"
    "QPushButton:hover {"
    "    background-color: grey;"
    "    color: white;"
    "    padding: 8px;"
    "    border: 3px solid #fff;"
    "}";

this->setStyleSheet(
    "QWidget { color: white; }"
    "QMainWindow { background-color: rgb(90,90,90); }"
    );

ui->Interwal->setStyleSheet(
    "QLineEdit {"
    "    background-color: white;"
    "    color: black;"
    "}"
    );
ui->Start->setStyleSheet(buttonStyleStart);
ui->Stop->setStyleSheet(buttonStyleStop);
```

```
ui->Reset->setStyleSheet(buttonStyleReset);
ui->Wczytaj->setStyleSheet(buttonStyleReszta);
ui->UstawieniaGeneratora->setStyleSheet(buttonStyleReszta);
ui->UstawieniaRegulatora->setStyleSheet(buttonStyleReszta);
ui->UstawieniaObiektuARX->setStyleSheet(buttonStyleReszta);
ui->Zapisz->setStyleSheet(buttonStyleReszta);

ui->TytulWykres1->setStyleSheet(
   "QLabel {"
   "    background-color: rgb(130, 130, 130);"
   "    color: white;"
   "    border: 1px solid black;"
   "    padding: 5px;"
   "}"
   );

ui->TytulWykres2->setStyleSheet(
   "QLabel {"
   "    background-color: rgb(130, 130, 130);"
   "    color: white;"
   "    border: 1px solid black;"
   "    padding: 5px;"
   "}"
   );

ui->TytulWykres3->setStyleSheet(
   "QLabel {"
   "    background-color: rgb(130, 130, 130);"
   "    color: white;"
   "    border: 1px solid black;"
   "    padding: 5px;"
   "}"
   );

ui->TytulWykres4->setStyleSheet(
   "QLabel {"
   "    background-color: rgb(130, 130, 130);"
   "    color: white;"
   "    border: 1px solid black;"
   "    padding: 5px;"
   "}"
   );

ui->TytulGlowny->setStyleSheet(
   "QLabel {"
   "    background-color: rgb(130, 130, 130);"
   "    color: white;"
   "    border: 1px solid black;"
```

```
        "    padding: 5px;"
        "}"
        );
    ui->InterwalCzasowyLabel->setStyleSheet(
        "QLabel {"
        "    background-color: rgb(130, 130, 130);"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );

    QGraphicsDropShadowEffect *effect[8]; //

    for (int i = 0; i < 8; ++i) {
        effect[i] = new QGraphicsDropShadowEffect();
        effect[i]->setOffset(0, 0);          // Ustawienie przesunięcia cienia (0, 0 dla cienia wokół)
        effect[i]->setBlurRadius(10);        // Rozmycie cienia
        effect[i]->setColor(Qt::black);      // Kolor cienia
    }
    ui->Start->setGraphicsEffect(effect[0]);
    ui->Stop->setGraphicsEffect(effect[1]);
    ui->Reset->setGraphicsEffect(effect[2]);
    ui->UstawieniaGeneratora->setGraphicsEffect(effect[3]);
    ui->UstawieniaObiektuARX->setGraphicsEffect(effect[4]);
    ui->UstawieniaRegulatora->setGraphicsEffect(effect[5]);
    ui->Zapisz->setGraphicsEffect(effect[6]);
    ui->Wczytaj->setGraphicsEffect(effect[7]);
}

void MainWindow::obslugaZapisu()
{
    QMessageBox::information(this, "Zapis konfiguracji", "Konfiguracja została zapisana.");
    qDebug() << "Odebrano sygnał zapisu.";
}

void MainWindow::on_Zapisz_clicked()
{
    Generator gen = usluga->getSymulator()->getGenerator();
    Regulator reg = usluga->getSymulator()->getRegulator();
    ObiektARX obiekt = usluga->getSymulator()->getObiektARX();
    QFile file("konfiguracja.txt");
    if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream out(&file);
        out << "Generator:\n";
        out << "Amplituda: " << gen.getAmplituda() << "\n";
        out << "Okres: " << gen.getOkres() << "\n";
        out << "Wypelnienie: " << gen.getWypelnienie() << "\n";
```

```
        out << "Rodzaj Sygnalu: " << static_cast<int>(gen.getRodzaj()) << "\n";
        out << "\nRegulator:\n";
        out << "Wzmocnienie: " << reg.getWzmocnienie() << "\n";
        out << "Stala I: " << reg.getStalaI() << "\n";
        out << "Stala D: " << reg.getStalaD() << "\n";
        out << "\nObiekt ARX:\n";
        out << "Opoznienie: " << obiekt.getOpoznienie() << "\n";
        out << "Współczynniki A: ";
        for (auto a : obiekt.getWielomianA()) {
            out << a << " ";
        }
        out << "\nWspółczynniki B: ";
        for (auto b : obiekt.getWielomianB()) {
            out << b << " ";
        }
        out << "\n";
        file.close();
        QMessageBox::information(this, "Sukces", "Konfiguracja została zapisana do pliku.");
    } else {
        QMessageBox::warning(this, "Błąd", "Nie udało się otworzyć pliku do zapisu.");
    }
}

void MainWindow::on_Wczytaj_clicked()
{
    QFile file("konfiguracja.txt");
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Błąd", "Nie udało się otworzyć pliku.");
        return;
    }

    QTextStream in(&file);
    Generator gen;
    Regulator reg;
    ObiektARX obiekt;
    std::vector<double> A_coeffs, B_coeffs;
    double amp, okres, wyp, wz, stI, stD, opoznienie;

    QString line;
    while (!in.atEnd()) {
        line = in.readLine();
        if (line.startsWith("Amplituda:")) amp = line.split(": ")[1].toDouble();
        if (line.startsWith("Okres:")) okres = line.split(": ")[1].toDouble();
        if (line.startsWith("Wypelnienie:")) wyp = line.split(": ")[1].toDouble();
        if (line.startsWith("Rodzaj Sygnalu:"))
gen.setRodzaj(static_cast<RodzajSygnalu>(line.split(": ")[1].toInt()));

        if (line.startsWith("Wzmocnienie:")) wz = line.split(": ")[1].toDouble();
```

```
        if (line.startsWith("Stala I:")) stI = line.split(": ")[1].toDouble();
        if (line.startsWith("Stala D:")) stD = line.split(": ")[1].toDouble();

        if (line.startsWith("Opoznienie:")) opoznienie = line.split(": ")[1].toDouble();

        if (line.startsWith("Współczynniki A:")) {
            QStringList values = line.split(": ")[1].split(" ");
            for (QString val : values) A_coeffs.push_back(val.toDouble());
        }
        if (line.startsWith("Współczynniki B:")) {
            QStringList values = line.split(": ")[1].split(" ");
            for (QString val : values) B_coeffs.push_back(val.toDouble());
        }
    }
    file.close();

    // Ustawienie wartości w obiektach
    gen.setAmplituda(amp);
    gen.setOkres(okres);
    gen.setWypelnienie(wyp);

    reg.setWzmocnienie(wz);
    reg.setStalaI(stI);
    reg.setStalaD(stD);

    obiekt.setOpoznienie(opoznienie);
    obiekt.setWielomianA(A_coeffs);
    obiekt.setWielomianB(B_coeffs);

    // Ustawienie w symulatorze
    symulator* sym = usluga->getSymulator();
    if (sym) {
        sym->setGenerator(gen);
        sym->setRegulator(reg);
        sym->setObiektARX(obiekt);
    }
    QMessageBox::information(this, "Sukces", "Konfiguracja została wczytana.");
}
#include "obiektarx.h"

ObiektARX::ObiektARX(){}

ObiektARX::ObiektARX(double kk, double zz, std::vector<double> aa, std::vector<double>
bb, std::mt19937 gen, double mean, double stdev)
    : k(kk), z(zz), a(aa), b(bb), ui(bb.size() + static_cast<int>(kk), 0),
    yi(aa.size(), 0), mean(mean), stdev(stdev), generator(gen), zaklocenie(mean, stdev){}

void ObiektARX::setZaklocenie(double newMean, double newStdev) {
```

```
    mean = newMean;
    stdev = newStdev;
    zaktualizujZaklocenie();
}

double ObiektARX::getZaklocenie() {
    return z;
}

void ObiektARX::zaktualizujZaklocenie() {
    zaklocenie = std::normal_distribution<double>(mean, stdev);
}

double ObiektARX::obliczWyjscie(double uii) {
    ui.push_front(uii);
    if (ui.size() > b.size() + static_cast<int>(k)) {
        ui.pop_back();
    }
    double wynik = 0.0;

    for (size_t j = 0; j < b.size(); ++j) {
        if (ui.size() > j + static_cast<int>(k)) {
            wynik += b[j] * ui[j + static_cast<int>(k)];
        }
    }

    for (size_t j = 0; j < a.size(); ++j) {
        if (yi.size() > j) {
            wynik -= a[j] * yi[j];
        }
    }
    z = zaklocenie(generator);
    wynik += z;

    yi.push_front(wynik);
    if (yi.size() > a.size()) {
        yi.pop_back();
    }

    return wynik;
}

#include "oknogenerator.h"
#include "ui_oknogenerator.h"
#include "warstwauslug.h"

OknoGenerator::OknoGenerator(QWidget *parent)
    : QDialog(parent)
```

```
    , ui(new Ui::OknoGenerator)
{
    ui->setupUi(this);
    gen = new Generator;
    ui->Amplituda->setRange(-100, 1000);
    ui->Okres->setRange(0, 1000);
    ui->Wypelnienie->setRange(0, 1);
    ui->Amplituda->setSingleStep(0.1);
    ui->Wypelnienie->setSingleStep(0.05);
    ui->RodzajeSygnalu->addAction("Sygnał Skokowy");
    ui->RodzajeSygnalu->addAction("Sygnał Sinusoidalny");
    ui->RodzajeSygnalu->addAction("Sygnał Prostokątny");
    UstawienieOkna();


}

OknoGenerator::~OknoGenerator()
{
    delete ui;
}

void OknoGenerator::on_RodzajeSygnalu_clicked()
{
    ui->RodzajeSygnalu->showMenu();
}

void OknoGenerator::on_RodzajeSygnalu_triggered(QAction *arg1)
{
    QString wybor = arg1->text();
    if (wybor == "Sygnał Skokowy") {
        gen->setRodzaj(RodzajSygnalu::Skok);
        ui->RodzajeSygnalu->setText("Sygnał Skokowy");
    } else if (wybor == "Sygnał Sinusoidalny") {
        gen->setRodzaj(RodzajSygnalu::Sinusoida);
        ui->RodzajeSygnalu->setText("Sygnał Sinusoidalny");
    } else if (wybor == "Sygnał Prostokątny") {
        gen->setRodzaj(RodzajSygnalu::Prostokatny);
        ui->RodzajeSygnalu->setText("Sygnał Prostokątny");
    }
}

void OknoGenerator::on_ZatwierdzenieUstawien_accepted()
{
    qDebug() << "Przed ustawieniem wartości generatora";
    gen->setAmplituda(ui->Amplituda->value());
    gen->setOkres(ui->Okres->value());
    gen->setWypelnienie(ui->Wypelnienie->value());
```

```
    qDebug() << "Dane generatora: " << gen->getAmplituda() << ", " << gen->getOkres() << ",
" << gen->getWypelnienie();
    usluga->SprawdzenieGeneratora(gen);
}
void OknoGenerator::UstawienieGUI(){
    this->setStyleSheet("background-color: rgb(90,90,90);");
}

void OknoGenerator::UstawienieOkna(){

    this->setStyleSheet("background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1, "
                        "stop:0 rgb(50, 50, 50), stop:1 rgb(30, 30, 30)); "
                        "color: white;");

    //this->setStyleSheet("background-color: grey;");
    //ui->AmplitudaNapis->setStyleSheet("color: grey;");
    //ui->OkresNapis->setStyleSheet("color: grey;");
    //ui->WypelnienieNapis->setStyleSheet("color: grey;");
    ui->Amplituda->setStyleSheet("background-color: white);");
    ui->RodzajeSygnalu->setStyleSheet(
        "QToolButton {"
        "    background-color: grey;"
        "    color: white;"
        "    border-radius: 5px;"
        "    border: 1px solid black;"
        "}"
        "QToolButton:hover {"
        "    background-color: white;"
        "    color: black;"
        "}"
        );
    ui->OkresNapis->setStyleSheet(
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );
    ui->AmplitudaNapis->setStyleSheet(
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );
    ui->WypelnienieNapis->setStyleSheet(
```

```
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );
    ui->ZatwierdzenieUstawien->setStyleSheet(
        "QDialogButtonBox {"
        "    background-color: white;"
        "    color: black;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );

}

#include "oknoobiektarx.h"
#include "ui_oknoobiektarx.h"
#include "warstwauslug.h"

OknoObiektARX::OknoObiektARX(QWidget *parent)
    : QDialog(parent)
    , ui(new Ui::OknoObiektARX)

{
    ui->setupUi(this);
    obiekt = new ObiektARX;
    usluga = new WarstwaUslug;
    ui->A1->setRange(-1000, 1000);
    ui->A1->setValue(0.5);
    ui->A2->setRange(-1000, 1000);
    ui->A2->setValue(0.4);
    ui->A3->setRange(-1000, 1000);
    ui->A3->setValue(0.3);
    ui->B1->setRange(-1000, 1000);
    ui->B1->setValue(0.3);
    ui->B2->setRange(-1000, 1000);
    ui->B2->setValue(0.2);
    ui->B3->setRange(-1000, 1000);
    ui->B3->setValue(0.1);
    ui->Opoznienie->setRange(0, 1000);
    ui->Opoznienie->setValue(1);
    ui->A1->setSingleStep(0.1);
    ui->A2->setSingleStep(0.1);
    ui->A3->setSingleStep(0.1);
    ui->B1->setSingleStep(0.1);
```

```
    ui->B2->setSingleStep(0.1);
    ui->B3->setSingleStep(0.1);
    ui->Opoznienie->setSingleStep(0.1);
    UstawienieARX();
}

OknoObiektARX::~OknoObiektARX()
{
    delete ui;
}

void OknoObiektARX::on_ZatwierdzenieUstawien_accepted()
{
    qDebug() << "Przed ustawieniem wartości obiektu";
    obiekt->setOpoznienie(ui->Opoznienie->value());
    obiekt->setWielomianA({ui->A1->value(), ui->A2->value(), ui->A3->value()});
    obiekt->setWielomianB({ui->B1->value(), ui->B2->value(), ui->B3->value()});
    qDebug() << "Dane obiektu: " << obiekt->getOpoznienie() << ", " <<
obiekt->getWielomianA() << ", " << obiekt->getWielomianB();
    usluga->SprawdzenieObiektu(obiekt);
}
void OknoObiektARX::UstawienieARX()
{
    this->setStyleSheet("background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1, "
                        "stop:0 rgb(50, 50, 50), stop:1 rgb(30, 30, 30)); "
                        "color: white;");

    ui->ALebel->setStyleSheet(
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );
    ui->Blebel->setStyleSheet(
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );
    ui->OpoznienieLabel->setStyleSheet(
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
```

```
        "    padding: 5px;"
        "}"
        );
    ui->label_7->setStyleSheet(
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );
}

#include "oknoregulator.h"
#include "ui_oknoregulator.h"
#include "warstwauslug.h"

OknoRegulator::OknoRegulator(QWidget *parent)
    : QDialog(parent)
    , ui(new Ui::OknoRegulator)
{
    ui->setupUi(this);
    reg = new Regulator;
    ui->Wzmocnienie->setRange(0, 1000);
    ui->Wzmocnienie->setValue(0.1);
    ui->StalaI->setRange(0, 1000);
    ui->StalaI->setValue(5);
    ui->StalaD->setRange(0, 1000);
    ui->StalaD->setValue(0.1);
    ui->Wzmocnienie->setSingleStep(0.1);
    ui->StalaI->setSingleStep(0.1);
    ui->StalaD->setSingleStep(0.1);
    UstawienieReg();
}

OknoRegulator::~OknoRegulator()
{
    delete ui;
}

void OknoRegulator::on_ZatwierdzenieUstawien_accepted()
{
    qDebug() << "Przed ustawieniem wartości regulatora";
    reg->setWzmocnienie(ui->Wzmocnienie->value());
    reg->setStalaI(ui->StalaI->value());
    reg->setStalaD(ui->StalaD->value());
    qDebug() << "Dane regulatora: " << reg->getWzmocnienie() << ", " << reg->getStalaI() <<
", " << reg->getStalaD();
```

```
    usluga->SprawdzenieRegulatora(reg);
}
void OknoRegulator::UstawienieReg()
{
    this->setStyleSheet("background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1, "
                        "stop:0 rgb(50, 50, 50), stop:1 rgb(30, 30, 30)); "
                        "color: white;");
    ui->NastawaPopis->setStyleSheet(
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );
    ui->NastawaDopis->setStyleSheet(
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );
    ui->NastawaIopis->setStyleSheet(
        "QLabel {"
        "    background-color: grey;"
        "    color: white;"
        "    border: 1px solid black;"
        "    padding: 5px;"
        "}"
        );
}

#include "regulator.h"


Regulator::Regulator(double kp, double ki, double kd)
    : wartoscZadana(0), wzmocnienieP(kp), stalaI(ki), stalaD(kd), Uchyb(0),
WczesniejszyUchyb(0), sumaUchybow(0), WartoscSterujaca(0) {}
void Regulator::setWartoscZadana(double war) { wartoscZadana = war; }
double Regulator::getWartoscZadana() const { return wartoscZadana; }

void Regulator::aktualizujUchyb(double wartoscRegulowana) {
    WczesniejszyUchyb = Uchyb;
    Uchyb = wartoscZadana - wartoscRegulowana;
    if (stalaI != 0) {
        sumaUchybow += Uchyb;
    }
```

```
}
Regulator::Regulator(){};
double Regulator::obliczSterowanie() {
    nastawaP = wzmocnienieP * Uchyb;
    nastawaI = 0;
    if (stalaI != 0) {
        nastawaI = sumaUchybow / stalaI;
    }
    nastawaD = stalaD * (Uchyb - WczesniejszyUchyb);

    WartoscSterujaca = nastawaP + nastawaI + nastawaD;
    return WartoscSterujaca;
}

double Regulator::getWartoscSterujaca() {
    return WartoscSterujaca;
}
double Regulator::getUchyb(){
    return Uchyb;
}
double Regulator::getNastawaP(){
    return nastawaP;
}
double Regulator::getNastawaI(){
    return nastawaI;
}
double Regulator::getNastawaD(){
    return nastawaD;
}
void Regulator::ZerowanieNastawaD(){
    sumaUchybow=0;
    nastawaI=0;
}
void Regulator::ZerowanieNastawaI(){
    nastawaI=0;
    nastawaD =0;
}
void Regulator::ZerowanieNastawaP(){
    nastawaP=0;
    Uchyb=0;
}

#include "symulator.h"
#include "qtpreprocessorsupport.h"

symulator::symulator()
    : generator(), regulator(), obiekt() {}
symulator::symulator(Generator g, Regulator r, ObiektARX o)
```

```
    : generator(g), regulator(r), obiekt(o) {}

double symulator::symulujKrok(double czas) {
    double wartoscZadana = generator.generuj(czas);  // Używamy obiektu
    regulator.setWartoscZadana(wartoscZadana);
    poprzednieWyjscie = wyjscieObiektu;
    wyjscieObiektu = obiekt.obliczWyjscie(regulator.getWartoscSterujaca());
    setLastRegulatorValue(regulator.getWartoscSterujaca());
    setLastObjectOutput(wyjscieObiektu);
    regulator.aktualizujUchyb(wyjscieObiektu);
    obiekt.setZaklocenie(0.1, 0.3);
    double sygnalSterowania = regulator.obliczSterowanie();
    Q_UNUSED(sygnalSterowania);
    return wyjscieObiektu;
}

double symulator::getWartoscZadana() { return regulator.getWartoscZadana(); }
double symulator::getZaklocenie() { return obiekt.getZaklocenie(); }
double symulator::getSterowanie() { return regulator.getWartoscSterujaca(); }
double symulator::getWyjscieObiektu() { return wyjscieObiektu; }
void symulator::setGenerator(Generator g) { generator = g; }
void symulator::setRegulator(Regulator r) { regulator = r; }
void symulator::setObiektARX(ObiektARX o) { obiekt = o; }
Generator symulator::getGenerator(){
    return generator;
};
Regulator symulator::getRegulator(){
    return regulator;
};
ObiektARX symulator::getObiektARX(){
    return obiekt;
};


#include "warstwauslug.h"

WarstwaUslug::WarstwaUslug(QObject *parent)
    : QObject{parent}, s(new symulator())
{
    qDebug() << "WarstwaUslug constructor";
    qDebug() << "symulator zainicjalizowany";
}

void WarstwaUslug::SprawdzenieGeneratora(Generator* g){
    qDebug() << "SprawdzenieGeneratora: " << g->getAmplituda() << ", " << g->getOkres()
<< ", " << g->getWypelnienie();
    if(g->getRodzaj() == RodzajSygnalu::Skok && g->getAmplituda() > 0) {
        s->setGenerator(*g);
```

```
        qDebug() << "Generator ustawiony (Skok)";
    } else if(g->getRodzaj() == RodzajSygnalu::Sinusoida && g->getAmplituda() > 0 &&
g->getOkres() > 0) {
        s->setGenerator(*g);
        qDebug() << "Generator ustawiony (Sinusoida)";
    } else if(g->getRodzaj() == RodzajSygnalu::Prostokatny && g->getAmplituda() > 0 &&
g->getOkres() > 0 && g->getWypelnienie() > 0 && g->getWypelnienie() <= 1) {
        s->setGenerator(*g);
        qDebug() << "Generator ustawiony (Prostokatny)";
    } else {
        qDebug() << "Błędny generator";
    }
    qDebug() << "Aktualny stan generatora: " << s->getGenerator().getAmplituda();
}

void WarstwaUslug::SprawdzenieRegulatora(Regulator* r){
    qDebug() << "SprawdzenieRegulatora: " << r->getStalaD() << ", " << r->getStalaI() << ", "
<< r->getWzmocnienie();
    if(r->getStalaD() >= 0 && r->getStalaI() >= 0 && r->getWzmocnienie() >= 0) {
        if(r->getStalaD() > 0 || r->getStalaI() > 0 || r->getWzmocnienie() > 0) {
            s->setRegulator(*r);
            qDebug() << "Regulator ustawiony";
        } else {
            qDebug() << "Błędny regulator";
        }
    } else {
        qDebug() << "Błędne parametry regulatora";
    }
    qDebug() << "Aktualny stan regulatora: " << s->getRegulator().getWzmocnienie();
}

void WarstwaUslug::SprawdzenieObiektu(ObiektARX* o){
    qDebug() << "SprawdzenieObiektu: " << o->getOpoznienie() << ", " <<
o->getWielomianA() << ", " << o->getWielomianB();
    int a = 0, b = 0;
    for(double wsp : o->getWielomianA()) {
        if(wsp == 0) a++;
    }
    for(double wsp : o->getWielomianB()) {
        if(wsp == 0) b++;
    }
    if(b != 3 && a != 3 && o->getOpoznienie() >= 0) {
        s->setObiektARX(*o);
        qDebug() << "Obiekt ustawiony";
    } else {
        qDebug() << "Błędny obiekt";
    }
    qDebug() << "Aktualny stan obiektu: " << s->getObiektARX().getOpoznienie();
```

```
}

void WarstwaUslug::SprawdzenieWszystkichDanych(double i) {
    bool poprawnyGenerator = true;
    bool poprawnyRegulator = true;
    bool poprawnyObiekt = true;

    // Sprawdzenie generatora
    Generator generator = s->getGenerator();
    qDebug() << "Sprawdzanie generatora: " <<generator.getWypelnienie();
    if (generator.getRodzaj() == RodzajSygnalu::Skok && generator.getAmplituda() <= 0) {
        poprawnyGenerator = false;
    } else if (generator.getRodzaj() == RodzajSygnalu::Sinusoida &&
               (generator.getAmplituda() <= 0 || generator.getOkres() <= 0)) {
        poprawnyGenerator = false;
    } else if (generator.getRodzaj() == RodzajSygnalu::Prostokatny &&
               (generator.getAmplituda() <= 0 || generator.getOkres() <= 0 ||
                generator.getWypelnienie() <= 0 || generator.getWypelnienie() > 1)) {
        poprawnyGenerator = false;
    }

    // Sprawdzenie regulatora
    Regulator regulator = s->getRegulator();
    qDebug() << "Sprawdzanie regulatora: " << regulator.getStalaD();
    if (regulator.getStalaD() < 0 || regulator.getStalaI() < 0 || regulator.getWzmocnienie() < 0 ||
        (regulator.getStalaD() == 0 && regulator.getStalaI() == 0 &&
regulator.getWzmocnienie() == 0)) {
        poprawnyRegulator = false;
    }

    // Sprawdzenie obiektu
    ObiektARX obiekt = s->getObiektARX();
    qDebug() << "Sprawdzanie obiektu: " << obiekt.getOpoznienie();
    int liczbaZerA = 0;
    int liczbaZerB = 0;

    for (double wsp : obiekt.getWielomianA()) {
        if (wsp == 0) liczbaZerA++;
    }

    for (double wsp : obiekt.getWielomianB()) {
        if (wsp == 0) liczbaZerB++;
    }

    if (liczbaZerA == 3 || liczbaZerB == 3 || obiekt.getOpoznienie() < 0) {
        poprawnyObiekt = false;
    }
```

```
    if (!poprawnyGenerator || !poprawnyRegulator || !poprawnyObiekt||i<=0) {
        emit BledneDane();
        qDebug() << "Błędne dane - sprawdź generator, regulator i obiekt.";
    } else {
        emit PoprawneDane(s);
        qDebug() << "Dane poprawne - generowanie wykresów.";
    }

}
void WarstwaUslug::konfiguracjaZapis()
{
    if (!usluga) {
        qDebug() << "Błąd: usluga jest NULL w konfiguracjaZapis()!";
        return;
    }
    if (!s) {
        qDebug() << "Błąd: symulator s jest NULL!";
        return;
    }
    Generator gen = usluga->getSymulator()->getGenerator();
    Regulator reg = usluga->getSymulator()->getRegulator();
    ObiektARX obiekt = usluga->getSymulator()->getObiektARX();
    QFile file("konfiguracja.txt");
    if (file.open(QIODevice::ReadWrite | QIODevice::Text)) {
        QTextStream out(&file);
        out << "Generator:\n";
        out << "Amplituda: " << gen.getAmplituda() << "\n";
        out << "Okres: " << gen.getOkres() << "\n";
        out << "Wypelnienie: " << gen.getWypelnienie() << "\n";
        out << "Rodzaj Sygnalu: " << static_cast<int>(gen.getRodzaj()) << "\n";
        out << "\nRegulator:\n";
        out << "Wzmocnienie: " << reg.getWzmocnienie() << "\n";
        out << "Stala I: " << reg.getStalaI() << "\n";
        out << "Stala D: " << reg.getStalaD() << "\n";
        out << "\nObiekt ARX:\n";
        out << "Opoznienie: " << obiekt.getOpoznienie() << "\n";
        out << "Współczynniki A: ";
        for (auto a : obiekt.getWielomianA()) {
            out << a << " ";
        }
        out << "\nWspółczynniki B: ";
        for (auto b : obiekt.getWielomianB()) {
            out << b << " ";
        }
        out << "\n";
        file.close();
    }
```

```
   emit sygnalZapisano();
}
/*
void WarstwaUslug::konfiguracja()
{
    Generator gen = usluga->getSymulator()->getGenerator();
    Regulator reg = usluga->getSymulator()->getRegulator();
    ObiektARX obiekt = usluga->getSymulator()->getObiektARX();
    QFile file("konfiguracja.txt");
    if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream out(&file);
        out << "Generator:\n";
        out << "Amplituda: " << gen.getAmplituda() << "\n";
        out << "Okres: " << gen.getOkres() << "\n";
        out << "Wypelnienie: " << gen.getWypelnienie() << "\n";
        out << "Rodzaj Sygnalu: " << static_cast<int>(gen.getRodzaj()) << "\n";
        out << "\nRegulator:\n";
        out << "Wzmocnienie: " << reg.getWzmocnienie() << "\n";
        out << "Stala I: " << reg.getStalaI() << "\n";
        out << "Stala D: " << reg.getStalaD() << "\n";
        out << "\nObiekt ARX:\n";
        out << "Opoznienie: " << obiekt.getOpoznienie() << "\n";
        out << "Współczynniki A: ";
        for (auto a : obiekt.getWielomianA()) {
            out << a << " ";
        }
        out << "\nWspółczynniki B: ";
        for (auto b : obiekt.getWielomianB()) {
            out << b << " ";
        }
        out << "\n";
        file.close();
        QMessageBox::information(this, "Sukces", "Konfiguracja została zapisana do pliku.");
    } else {
        QMessageBox::warning(this, "Błąd", "Nie udało się otworzyć pliku do zapisu.");
    }
}
*/


#include "wykresy.h"

Wykresy::Wykresy(QWidget *parent)
    : QObject(nullptr),parent(parent)
{
    s=new symulator();
}
```

```
void Wykresy::wyczyscLayout(QLayout* layout) {
    if (!layout) return;
    while (QLayoutItem* item = layout->takeAt(0)) {
        if (QWidget* widget = item->widget()) {
            widget->deleteLater();
        }
        if (QLayout* childLayout = item->layout()) {
            wyczyscLayout(childLayout);
        }
        delete item;
    }
}
void Wykresy::inicjalizacjaWykresuWartosciZadanej(QVBoxLayout *layout){
    wyczyscLayout(layout);
    // Tworzenie serii danych
    seria[0] = new QLineSeries();
    seria[1] = new QLineSeries();
    seria[0]->setName("Wartość Regulowana");
    seria[1]->setName("Wartość Zadana");
    seria[1]->setColor(Qt::red); // Zmiana koloru drugiej serii
    // Tworzenie wykresu
    wykres[0] = new QChart();
    wykres[0]->addSeries(seria[0]);
    wykres[0]->addSeries(seria[1]);
    // Tworzenie osi X i Y
    osX[0] = new QValueAxis();
    osX[0]->setRange(0, 30); // Początkowy zakres dla osi X
    osY[0] = new QValueAxis();
    // Ustawienie początkowego zakresu osi Y
    osY[0]->setRange(0, 10);
    // Dodanie osi X i Y do wykresu
    wykres[0]->addAxis(osX[0], Qt::AlignBottom);
    wykres[0]->addAxis(osY[0], Qt::AlignLeft);
    // Powiązanie osi z serią danych
    seria[0]->attachAxis(osX[0]);
    seria[0]->attachAxis(osY[0]);
    // Powiązanie osi Y z drugą serią
    seria[1]->attachAxis(osX[0]);
    seria[1]->attachAxis(osY[0]);
    // Tworzenie widoku wykresu
    Widok[0] = new QChartView(wykres[0]);
    Widok[0]->setRenderHint(QPainter::Antialiasing);
    Widok[0]->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    //Dodanie wykresu do widoku w UI
    layout->addWidget(Widok[0]);
}
void Wykresy::inicjalizacjaWykresuUchybu(QVBoxLayout *layout){
```

```
    wyczyscLayout(layout);

    seria[2] = new QLineSeries();
    seria[2]->setName("Wartość Uchybu");
    wykres[1] = new QChart();
    wykres[1]->addSeries(seria[2]);
    osX[1] = new QValueAxis();
    osX[1]->setRange(0, 30);
    osY[1] = new QValueAxis();
    osY[1]->setRange(0, 10);
    wykres[1]->addAxis(osX[1], Qt::AlignBottom);
    wykres[1]->addAxis(osY[1], Qt::AlignLeft);
    seria[2]->attachAxis(osX[1]);
    seria[2]->attachAxis(osY[1]);
    Widok[1] = new QChartView(wykres[1]);
    Widok[1]->setRenderHint(QPainter::Antialiasing);
    Widok[1]->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    layout->addWidget(Widok[1]);
}
void Wykresy::inicjalizacjaWykresuPID(QVBoxLayout *layout){
    wyczyscLayout(layout);
    seria[3] = new QLineSeries();
    seria[3]->setName("Proporcjonalna");
    seria[4] = new QLineSeries();
    seria[4]->setColor(Qt::red);
    seria[4]->setName("Całkująca");
    seria[5] = new QLineSeries();
    seria[5]->setColor(Qt::black);
    seria[5]->setName("Różniczkująca");
    wykres[2] = new QChart();
    wykres[2]->addSeries(seria[3]);
    wykres[2]->addSeries(seria[4]);
    wykres[2]->addSeries(seria[5]);
    osX[2] = new QValueAxis();
    osX[2]->setRange(0, 30);
    osY[2] = new QValueAxis();
    osY[2]->setRange(0, 10);
    wykres[2]->addAxis(osX[2], Qt::AlignBottom);
    wykres[2]->addAxis(osY[2], Qt::AlignLeft);
    seria[3]->attachAxis(osX[2]);
    seria[3]->attachAxis(osY[2]);
    seria[4]->attachAxis(osX[2]);
    seria[4]->attachAxis(osY[2]);
    seria[5]->attachAxis(osX[2]);
    seria[5]->attachAxis(osY[2]);
    Widok[2] = new QChartView(wykres[2]);
    Widok[2]->setRenderHint(QPainter::Antialiasing);
    Widok[2]->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
```

```
    layout->addWidget(Widok[2]);
}
void Wykresy::inicjalizacjaWykresuWartosciSterowania(QVBoxLayout *layout){
    wyczyscLayout(layout);
    seria[6] = new QLineSeries();
    seria[6]->setName("Wartość Sterująca");
    wykres[3] = new QChart();
    wykres[3]->addSeries(seria[6]);
    osX[3] = new QValueAxis();
    osX[3]->setRange(0, 30);
    osY[3] = new QValueAxis();
    osY[3]->setRange(0, 10);
    wykres[3]->addAxis(osX[3], Qt::AlignBottom);
    wykres[3]->addAxis(osY[3], Qt::AlignLeft);
    seria[6]->attachAxis(osX[3]);
    seria[6]->attachAxis(osY[3]);
    Widok[3] = new QChartView(wykres[3]);
    Widok[3]->setRenderHint(QPainter::Antialiasing);
    Widok[3]->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    layout->addWidget(Widok[3]);
}
void Wykresy::WykresWartosciZadanej() {
    double wyjscieObiektu = s->symulujKrok(czas);
    seria[0]->append(czas, wyjscieObiektu);
    seria[1]->append(czas, s->getWartoscZadana());
    // Usuwanie starych punktów, aby seria była "przesuwana"
    const int maxPoints = 30;
    if (seria[0]->count() > maxPoints) {
        seria[0]->remove(0);
        seria[1]->remove(0);
    }
    // Przesunięcie osi X
    if (czas > 30) {
        osX[0]->setRange(czas - 30, czas);
    }
    // Dynamically adjusting Y axis range
    double minY =s->getWyjscieObiektu(), maxY = s->getWartoscZadana();
    Generator generator = s->getGenerator();
    if (generator.getRodzaj() == RodzajSygnalu::Skok) {
        minY = 0;
    }
    else if (generator.getRodzaj() ==
RodzajSygnalu::Sinusoida&&generator.getRodzaj()==RodzajSygnalu::Prostokatny) {
        double amplituda = generator.getAmplituda();
        minY = -amplituda;
        maxY = amplituda;
    }
    double margin = maxY*0.1;
```

```
    osY[0]->setRange(minY - margin, maxY + margin);
    czas++;
}

void Wykresy::WykresUchybu() {
    double wyjscieObiektu = s->symulujKrok(czas);
    Q_UNUSED(wyjscieObiektu);
    double uchyb = s->getRegulator().getUchyb();
    seria[2]->append(czas, uchyb);
    const int maxPoints = 30;
    if (seria[2]->count() > maxPoints) {
        seria[2]->remove(0);
    }

    if (czas > 30) {
        osX[1]->setRange(czas - 30, czas);
    }

    double minY = uchyb;
    double maxY = uchyb;
    for (int i = 0; i < seria[2]->count(); ++i) {
        double yValue = seria[2]->at(i).y();
        minY = std::min(minY, yValue);
        maxY = std::max(maxY, yValue);
    }
    double margin = 0.1 * (maxY - minY);
    osY[1]->setRange(minY - margin, maxY + margin);
    czas++;
}

void Wykresy::WykresPID() {
    double wyjscieObiektu = s->symulujKrok(czas);
    Q_UNUSED(wyjscieObiektu);
    Regulator regulator = s->getRegulator();
    seria[3]->append(czas, regulator.getNastawaP());
    seria[4]->append(czas, regulator.getNastawaI());
    seria[5]->append(czas, regulator.getNastawaD());

    const int maxPoints = 30;
    if (seria[3]->count() > maxPoints) {
        seria[3]->remove(0);
        seria[4]->remove(0);
        seria[5]->remove(0);
    }

    if (czas > maxPoints) {
        osX[2]->setRange(czas - maxPoints, czas);
```

```
    }

    double minY = std::numeric_limits<double>::max();
    double maxY = std::numeric_limits<double>::lowest();

    for (int i = 0; i < seria[3]->count(); ++i) {
        double yValueP = seria[3]->at(i).y();
        double yValueI = seria[4]->at(i).y();
        double yValueD = seria[5]->at(i).y();

        minY = std::min({minY, yValueP, yValueI, yValueD});
        maxY = std::max({maxY, yValueP, yValueI, yValueD});
    }

    double margin = maxY*0.1;
    minY -= margin;
    maxY += margin;

    if (osY[2]->min() != minY || osY[2]->max() != maxY) {
        osY[2]->setRange(minY, maxY);
    }
    czas++;
}
void Wykresy::WykresWartosciSterowania(){
    // Symulacja kolejnego kroku
    double wyjscieObiektu = s->symulujKrok(czas);
    Q_UNUSED(wyjscieObiektu);
    double Sterujaca = s->getRegulator().getWartoscSterujaca();
    seria[6]->append(czas, Sterujaca);
    qDebug()<<Sterujaca;

    // Usuwanie starych punktów, aby seria była "przesuwana"
    const int maxPoints = 30; // Maksymalna liczba widocznych punktów
    if (seria[6]->count() > maxPoints) {
        seria[6]->remove(0); // Usunięcie najstarszego punktu
    }

    // Ustawienie zakresu osi X, przesuwanie w prawo
    if (czas > 30) {
        osX[3]->setRange(czas - 30, czas); // Przesunięcie osi X
    }

    // Dynamically adjust Y axis range based on the current minimum and maximum values of
uchyb
    double minY = Sterujaca;
    double maxY = Sterujaca;

    // Przeszukaj całą serię punktów, aby znaleźć minimum i maksimum
```

```
    for (int i = 0; i < seria[6]->count(); ++i) {
        double yValue = seria[6]->at(i).y();
        minY = std::min(minY, yValue);
        maxY = std::max(maxY, yValue);
    }
    /*for (int i = 0; i < seria[2]->count(); ++i) {
        double yValue = seria[2]->at(i).y();
        minY = std::min(minY, yValue);
        maxY = std::max(maxY, yValue);
    }*/

    // Dodaj margines do zakresu Y, aby lepiej widoczny był wykres
    double margin = 0.1 * (maxY - minY); // 10% marginesu wokół wartości minimum i
maksimum
    osY[3]->setRange(minY - margin, maxY + margin);
    czas++;
}
void Wykresy::ResetujWykresy(){
    for(int i=0;i<7;i++){
        if (seria[i]) {
            seria[i]->clear();
        }
    }
}

void Wykresy::InicjalizujWykresy(QVBoxLayout *layout[4] ){
    inicjalizacjaWykresuWartosciZadanej(layout[0]);
    inicjalizacjaWykresuUchybu(layout[1]);
    inicjalizacjaWykresuPID(layout[2]);
    inicjalizacjaWykresuWartosciSterowania(layout[3]);
}
```