Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
Про виконання лабораторної роботи №1-4
на тему:
«Середовище програмування»

Лектор:
доц. Коротєєва Т.О.

Виконав:
ст. гр. ПЗ-11
Солтисюк Д.А.

Прийняла:
доц. Коротєєва Т.О.

« __ » _____ 2022 р.

∑ = _____ _____ .

Львів – 2022

ЛАБОРАТОРНА РОБОТА №1

Тема: Ознайомлення з середовищем розробки Qt. Створення проекту та налаштування його властивостей.
Мета: Засвоїти принцип візуального програмування шляхом створення та налаштування проекту.

ТЕОРЕТИЧНІ ВІДОМОСТІ

1. Середовище розробки Qt можна умовно поділити на три частини:

   1) Навігаційне меню зліва

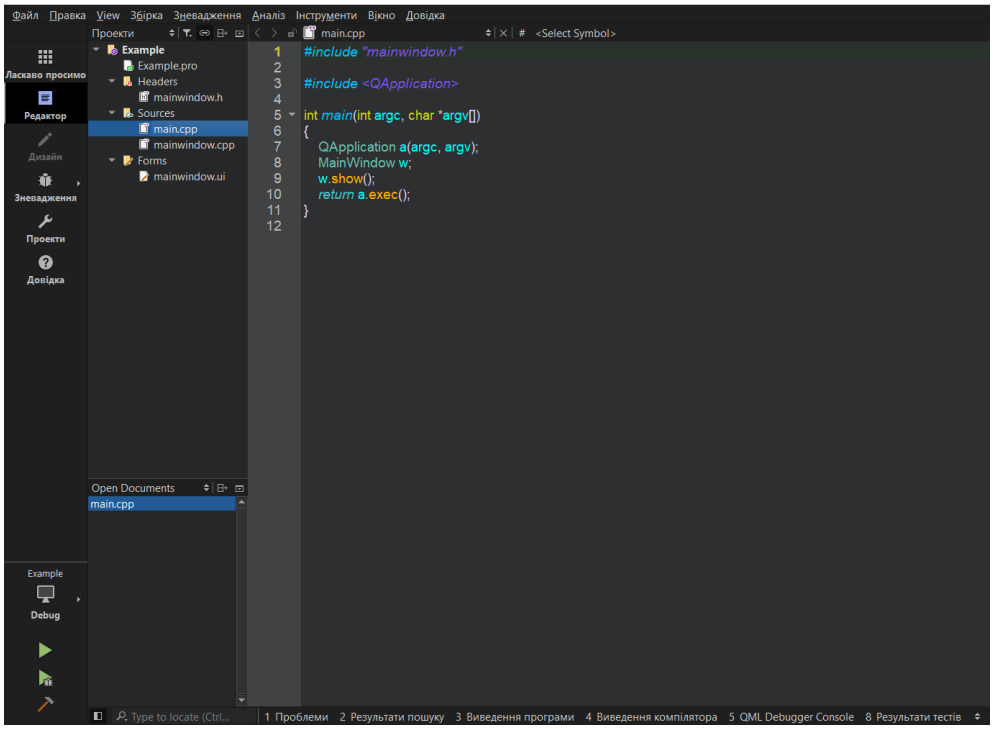
   2) Вікно огляду складових проекту

   3) Вікно редактора коду.

Навігаційне меню складається з п’яти вкладок, які слугують для переміщення між розділами середовища, його налаштування чи переходу в режим «зневадження» проекту. Знизу знаходяться кнопки для взаємодії з програмою, а саме: Запуск в звичайному режимі, запуск у режимі «зневадження» та збірка проекту.

Вікно огляду складових дає змогу розглянути всі компоненти, програми та перейти до роботи з ними. Стандартний проект Qt складається із файлу збірки (Example.pro), header-файлу, двох .cpp файлів та UI-форми, яка слугує для розробки інтерфейсу.

Вікно редактора коду відображає код програми та дає змогу користувачу працювати з ним.



1.2.Організація проекту в Qt

Mainwindow.ui є файлом інтерфейсу проекту. У вікні редактора форми містяться: форма вікна, інспектор об'єктів, вікно властивостей об'єкта, список об'єктів.

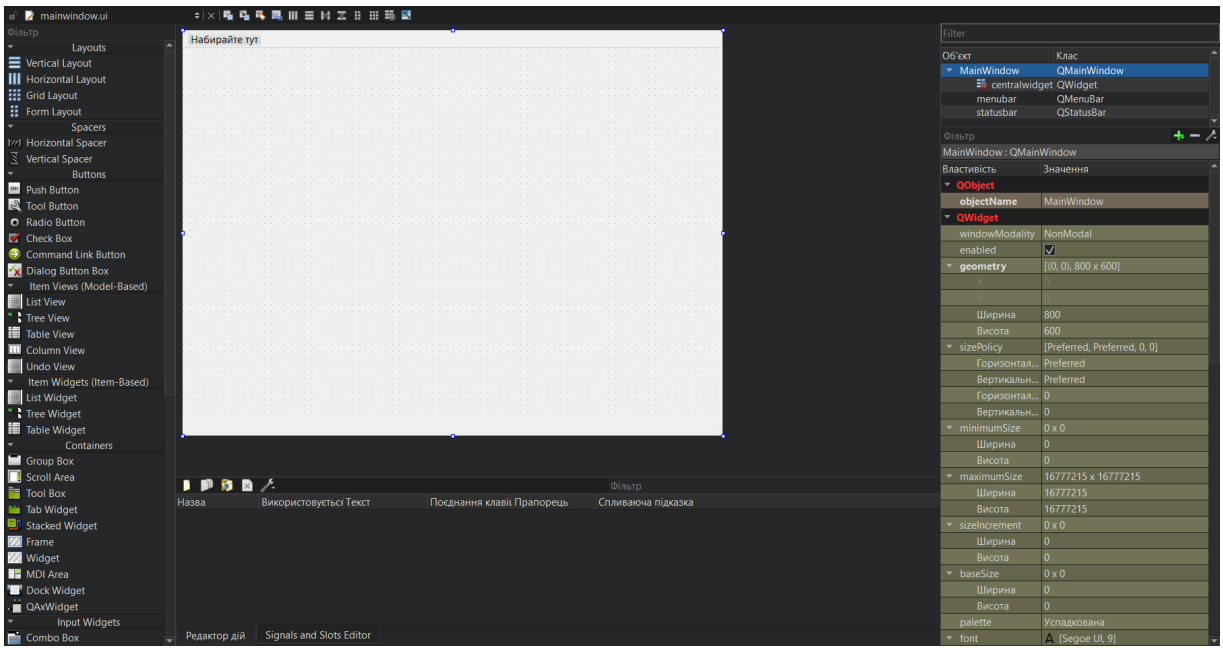
Список об'єктів поділений на категорії, залежно від використання.

Layouts – використовується для впорядкованого розміщення об’єктів на формі.

Spacers – дозволяє розмежовувати структури на формі.

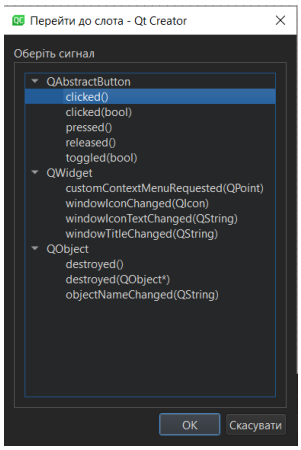
Buttons – кнопки та елементи взаємодії користувача з програмою.

Будь-який об’єкт має свої властивості, які вказані у відведеному вікні. Воно дає змогу переглядати та змінювати характеристики об’єктів та форми. При редагуванні візуальних властивостей складових інтерфейсу використовують CSS Stylesheets, що надає більші можливості в плані стилізації.



Об’єкти можна розмістити на формі перетягнувши їх. Після цього можна змінити його властивості або перейти до слота.

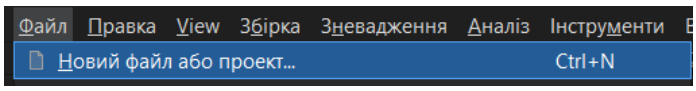
Зробивши це, ми обираємо тип сигналу та переходимо до редактора коду, де й описуємо дії, які виконуватиме програма у разі взаємодії користувача з даним об’єктом.



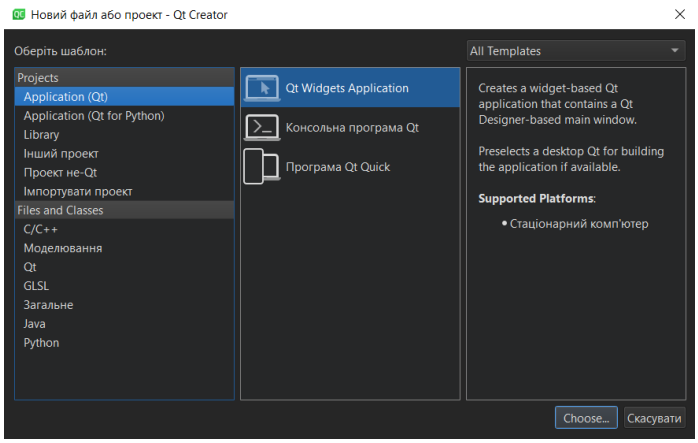

```
void MainWindow::on_pushButton_clicked()
{

}
```

1.3 Створення та збереження нового проекту.



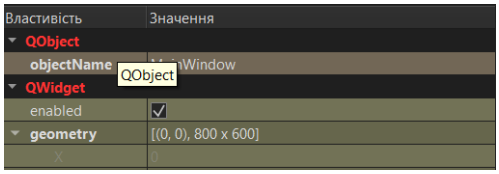
Новий проект можна створити натиснувши на вкладку **Файл→Новий файл або проект** або ж використати комбінацію клавіш **Ctrl+N**.



Щоб зберегти проект, потрібно перейти до вкладки **Файл→ Save All** або використати комбінацію **Ctrl+Shift+S**.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

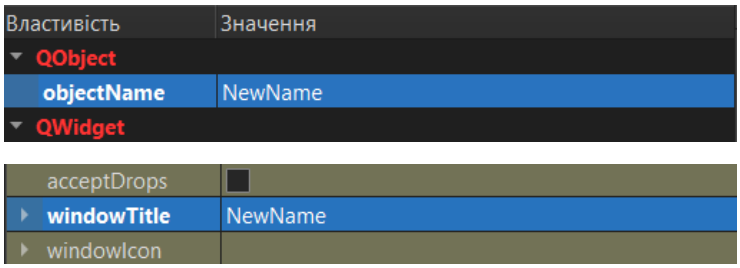
1. Ознайомитись із середовищем Qt.
2. Створити новий проект. Зберегти його двома способами – через комбінації швидких клавіш та через меню.
3. Проглянути у вікні інспектора об’єктів властивості форми. Змінити назву форми та її розміри.



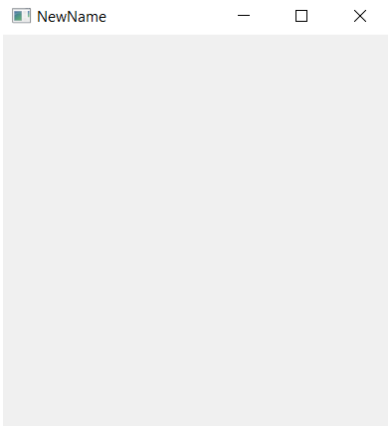
Назву форми можна змінити двома способами: у полі ObjectName та в полі WindowTitle. Проте в першому випадку ми змінюємо саме назву вікна, як об’єкта, тому при подальшій роботі з кодом, повинні

будемо звертатись до нього саме за цим іменем. У другому зміна імені не впливатиме на роботу з кодом.

Виконання:



4. Запустити на виконання застосування.



5. Відкрити опції проекту, змінити налаштування на закладках **Application, Compiler, Packages.** Запустити на виконання застосування.

ВИСНОВКИ

Виконавши лабораторну роботу №1, я ознайомився із середовищем розробки Qt Creator та засвоїв принцип створення та налаштування віконних проектів.

ЛАБОРАТОРНА РОБОТА №2

Тема: Базові візуальні компоненти Qt Creator. Створення проекту із використанням візуальних компонент.

Мета: Створити віконний проект та продемонструвати використання візуальних компонент Qt Creator.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Середовище Qt Creator містить набори візуальних компонент, які дозволяють створювати користувацький інтерфейс для взаємодії з програмою. Крім цього, є можливість створити власну компоненту і розмістити її на закладці, імпортувати набір компонент сторонніх розробників, змінювати розташування закладок.

Компоненти додаються на форму способом їх перетягування на робочу область.

Компонента **QPushButton** – це клас, який ідентифікує візуальну компоненту типу «кнопка». Ця кнопка має визначені типи взаємодії: натиснути, клацнути, натиснути й відпустити тощо. Об'єкти мають властивості, деякі з них: колір, колір тексту, обрамлення, його товщина і заокруглення. Об'єкти також можуть вміщати текст, який, у свою чергу має й свої властивості.

Компонента **QLabel** – це клас, об'єктом якого є область, яка вміщає текстовий напис. Цей клас дозволяє також розміщувати в об'єкті графічні зображення, оскільки безпосереднє вставлення зображень не передбачене у Qt Creator. Основною функцією таких об'єктів є відображення тексту. Він може бути введений безпосередньо у редакторі форм, чи за допомогою команд у редакторі коду, проте не може бути змінений способом вводу під час виконання програми. QLabel, як і кожен **QWidget** має властивості: колір, колір тексту, обрамлення тощо.

Компонента **QLineEdit –** це клас, об'єкти якого за функціоналом схожі до **QLabel**, проте володіючи всіма його властивостями, **QLineEdit** дозволяє змінювати текст у собі, в ході виконання програми.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Ознайомитись із палітрою компонент **Qt Creator**.
2. Створити віконний проект, додати розглянуті візуальні компоненти.
3. Реалізувати калькулятор.

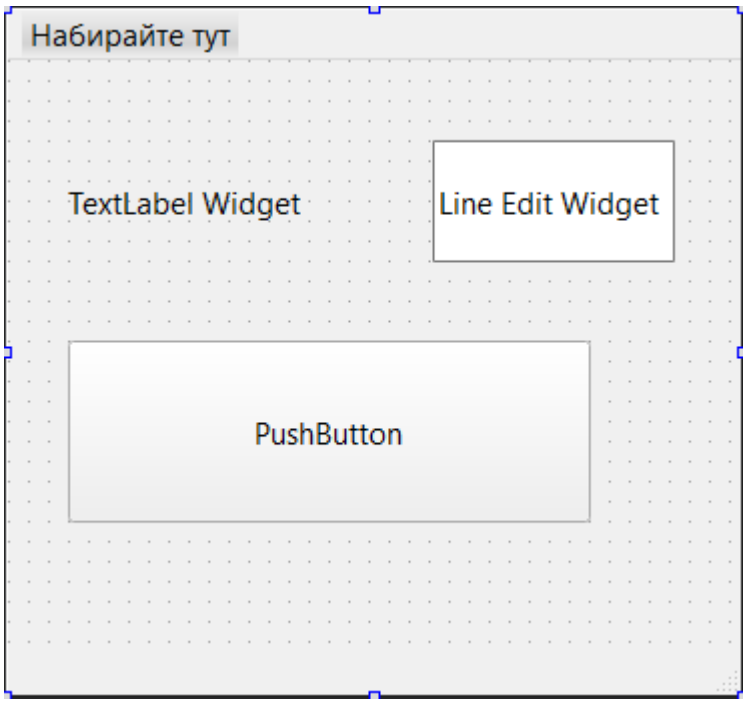


Рис.1 Об’єкти, розміщені на формі

3. Реалізація калькулятора

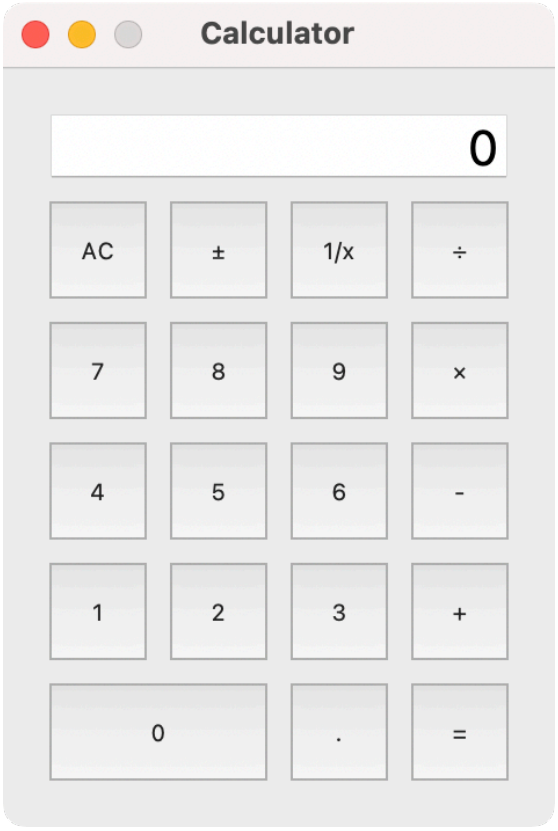


Рис.2 UI форма програми

Файл button.cpp

```
#include "button.h"

Button::Button(const QString &text, QWidget *parent) : QToolButton(parent) {
  setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Preferred);
  setText(text);
}

QSize Button::sizeHint() const {
  QSize size = QToolButton::sizeHint();

  size.rheight() += 20;
  size.rwidth() = qMax(size.width(), size.height());

  return size;
}
```

Файл button.h

```
#include <QToolButton>

class Button : public QToolButton {
  Q_OBJECT

public:
  explicit Button(const QString &text, QWidget *parent = nullptr);

  QSize sizeHint() const override;
};
```

Файл calculator.cpp

```
#include "calculator.h"

#include <QGridLayout>
#include <QLineEdit>
#include <QtMath>

Calculator::Calculator(QWidget *parent)
    : QWidget(parent), sumInMemory(0.0), sumSoFar(0.0), factorSoFar(0.0),
      waitingForOperand(true) {
  display = new QLineEdit("0");
  display->setReadOnly(true);
  display->setAlignment(Qt::AlignRight);
  display->setMaxLength(15);

  QFont font = display->font();

  font.setPointSize(font.pointSize() + 8);
  display->setFont(font);

  for (int i = 0; i < NumDigitButtons; ++i)
    digitButtons[i] = createButton(QString::number(i), SLOT(digitClicked()));

  Button *pointButton = createButton(tr("."), SLOT(pointClicked()));
  Button *changeSignButton =
      createButton(tr("\302\261"), SLOT(changeSignClicked()));
```

```
    // Button *backspaceButton =
    //     createButton(tr("Backspace"), SLOT(backspaceClicked()));
    // Button *clearButton = createButton(tr("Clear"), SLOT(clear()));
    Button *clearAllButton = createButton(tr("AC"), SLOT(clearAll()));

    // Button *clearMemoryButton = createButton(tr("MC"), SLOT(clearMemory()));
    // Button *readMemoryButton = createButton(tr("MR"), SLOT(readMemory()));
    // Button *setMemoryButton = createButton(tr("MS"), SLOT(setMemory()));
    // Button *addToMemoryButton = createButton(tr("M+"), SLOT(addToMemory()));

    Button *divisionButton =
        createButton(tr("\303\267"), SLOT(multiplicativeOperatorClicked()));
    Button *timesButton =
        createButton(tr("\303\227"), SLOT(multiplicativeOperatorClicked()));
    Button *minusButton = createButton(tr("-"), SLOT(additiveOperatorClicked()));
    Button *plusButton = createButton(tr("+"), SLOT(additiveOperatorClicked()));

    // Button *squareRootButton =
    //     createButton(tr("Sqrt"), SLOT(unaryOperatorClicked()));
    // Button *powerButton =
    //     createButton(tr("x\302\262"), SLOT(unaryOperatorClicked()));
    Button *reciprocalButton =
        createButton(tr("1/x"), SLOT(unaryOperatorClicked()));
    Button *equalButton = createButton(tr("="), SLOT(equalClicked()));

    QGridLayout *mainLayout = new QGridLayout;

    mainLayout->setSizeConstraint(QLayout::SetFixedSize);

    mainLayout->addWidget(display, 0, 0, 1, 4);
    mainLayout->addWidget(clearAllButton, 1, 0);
    mainLayout->addWidget(changeSignButton, 1, 1);
    mainLayout->addWidget(reciprocalButton, 1, 2);

    for (int i = 1; i < NumDigitButtons; ++i) {
        int row = ((9 - i) / 3) + 2;
        int column = ((i - 1) % 3);

        mainLayout->addWidget(digitButtons[i], row, column);
    }

    mainLayout->addWidget(digitButtons[0], 5, 0, 1, 2);
    mainLayout->addWidget(pointButton, 5, 2);

    mainLayout->addWidget(divisionButton, 1, 3);
    mainLayout->addWidget(timesButton, 2, 3);
    mainLayout->addWidget(minusButton, 3, 3);
    mainLayout->addWidget(plusButton, 4, 3);
    mainLayout->addWidget(equalButton, 5, 3);

    setLayout(mainLayout);
    setWindowTitle(tr("Calculator"));
}

void Calculator::digitClicked() {
    Button *clickedButton = qobject_cast<Button *>(sender());
    int digitValue = clickedButton->text().toInt();

    if (display->text() == "0" && digitValue == 0.0)
        return;
```

```
    if (waitingForOperand) {
        display->clear();
        waitingForOperand = false;
    }

    display->setText(display->text() + QString::number(digitValue));
}

void Calculator::unaryOperatorClicked() {
    Button *clickedButton = qobject_cast<Button *>(sender());
    QString clickedOperator = clickedButton->text();
    double operand = display->text().toDouble();
    double result = 0.0;

    if (clickedOperator == tr("Sqrt")) {
        if (operand < 0.0) {
            abortOperation();
            return;
        }

        result = std::sqrt(operand);
    } else if (clickedOperator == tr("x\302\262")) {
        result = std::pow(operand, 2.0);
    } else if (clickedOperator == tr("1/x")) {
        if (operand == 0.0) {
            abortOperation();
            return;
        }

        result = 1.0 / operand;
    }

    display->setText(QString::number(result));
    waitingForOperand = true;
}

void Calculator::additiveOperatorClicked() {
    Button *clickedButton = qobject_cast<Button *>(sender());
    if (!clickedButton)
        return;

    QString clickedOperator = clickedButton->text();
    double operand = display->text().toDouble();

    if (!pendingMultiplicativeOperator.isEmpty()) {
        if (!calculate(operand, pendingMultiplicativeOperator)) {
            abortOperation();
            return;
        }

        display->setText(QString::number(factorSoFar));

        operand = factorSoFar;
        factorSoFar = 0.0;
        pendingMultiplicativeOperator.clear();
    }

    if (!pendingAdditiveOperator.isEmpty()) {
        if (!calculate(operand, pendingAdditiveOperator)) {
```

```
      abortOperation();
      return;
    }

    display->setText(QString::number(sumSoFar));
  } else {
    sumSoFar = operand;
  }

  pendingAdditiveOperator = clickedOperator;
  waitingForOperand = true;
}

void Calculator::multiplicativeOperatorClicked() {
  Button *clickedButton = qobject_cast<Button *>(sender());

  if (!clickedButton)
    return;

  QString clickedOperator = clickedButton->text();
  double operand = display->text().toDouble();

  if (!pendingMultiplicativeOperator.isEmpty()) {
    if (!calculate(operand, pendingMultiplicativeOperator)) {
      abortOperation();
      return;
    }

    display->setText(QString::number(factorSoFar));
  } else {
    factorSoFar = operand;
  }

  pendingMultiplicativeOperator = clickedOperator;
  waitingForOperand = true;
}

void Calculator::equalClicked() {
  double operand = display->text().toDouble();

  if (!pendingMultiplicativeOperator.isEmpty()) {
    if (!calculate(operand, pendingMultiplicativeOperator)) {
      abortOperation();
      return;
    }

    operand = factorSoFar;
    factorSoFar = 0.0;
    pendingMultiplicativeOperator.clear();
  }
  if (!pendingAdditiveOperator.isEmpty()) {
    if (!calculate(operand, pendingAdditiveOperator)) {
      abortOperation();
      return;
    }

    pendingAdditiveOperator.clear();
  } else {
    sumSoFar = operand;
  }
```

```
  display->setText(QString::number(sumSoFar));

  sumSoFar = 0.0;
  waitingForOperand = true;
}

void Calculator::pointClicked() {
  if (waitingForOperand)
    display->setText("0");

  if (!display->text().contains('.'))
    display->setText(display->text() + tr("."));

  waitingForOperand = false;
}

void Calculator::changeSignClicked() {
  QString text = display->text();
  double value = text.toDouble();

  if (value > 0.0) {
    text.prepend(tr("-"));
  } else if (value < 0.0) {
    text.remove(0, 1);
  }

  display->setText(text);
}

void Calculator::backspaceClicked() {
  if (waitingForOperand)
    return;

  QString text = display->text();

  text.chop(1);

  if (text.isEmpty()) {
    text = "0";
    waitingForOperand = true;
  }

  display->setText(text);
}

void Calculator::clear() {
  if (waitingForOperand)
    return;

  display->setText("0");
  waitingForOperand = true;
}

void Calculator::clearAll() {
  sumSoFar = 0.0;
  factorSoFar = 0.0;

  pendingAdditiveOperator.clear();
  pendingMultiplicativeOperator.clear();
```

```
  display->setText("0");

  waitingForOperand = true;
}

void Calculator::clearMemory() { sumInMemory = 0.0; }

void Calculator::readMemory() {
  display->setText(QString::number(sumInMemory));

  waitingForOperand = true;
}

void Calculator::setMemory() {
  equalClicked();

  sumInMemory = display->text().toDouble();
}

void Calculator::addToMemory() {
  equalClicked();

  sumInMemory += display->text().toDouble();
}

Button *Calculator::createButton(const QString &text, const char *member) {
  Button *button = new Button(text);

  connect(button, SIGNAL(clicked()), this, member);

  return button;
}

void Calculator::abortOperation() {
  clearAll();
  display->setText(tr("####"));
}

bool Calculator::calculate(double rightOperand,
                           const QString &pendingOperator) {
  if (pendingOperator == tr("+")) {
    sumSoFar += rightOperand;
  } else if (pendingOperator == tr("-")) {
    sumSoFar -= rightOperand;
  } else if (pendingOperator == tr("\303\227")) {
    factorSoFar *= rightOperand;
  } else if (pendingOperator == tr("\303\267")) {
    if (rightOperand == 0.0)
      return false;
    factorSoFar /= rightOperand;
  }
  return true;
}
```

Файл calculator.h

```
#include "button.h"

#include <QLineEdit>
#include <QWidget>
```

```
class Calculator : public QWidget {
  Q_OBJECT

public:
  Calculator(QWidget *parent = nullptr);

private slots:
  void digitClicked();
  void unaryOperatorClicked();
  void additiveOperatorClicked();
  void multiplicativeOperatorClicked();
  void equalClicked();
  void pointClicked();
  void changeSignClicked();
  void backspaceClicked();
  void clear();
  void clearAll();
  void clearMemory();
  void readMemory();
  void setMemory();
  void addToMemory();

private:
  QLineEdit *display;

  QString pendingAdditiveOperator;
  QString pendingMultiplicativeOperator;

  enum { NumDigitButtons = 10 };

  Button *digitButtons[NumDigitButtons];
  Button *createButton(const QString &text, const char *member);

  void abortOperation();
  bool calculate(double rightOperand, const QString &pendingOperator);

  double sumInMemory;
  double sumSoFar;
  double factorSoFar;

  bool waitingForOperand;
};
```

Файл main.cpp

```
#include "calculator.h"

#include <QApplication>

int main(int argc, char *argv[]) {
  QApplication a(argc, argv);
  Calculator c;

  c.show();

  return a.exec();
}
```

ВИСНОВКИ

Виконавши лабораторну роботу №2, я ознайомився із візуальними компонентами середовища розробки Qt Creator, створив та налаштував віконний проект з візуальними компонентами.

ЛАБОРАТОРНА РОБОТА №3

Тема: Базові візуальні компоненти Qt Creator. Створення проекту із використанням невізуальних компонент.
Мета: Створити віконний проект та продемонструвати використання невізуальних компонент Qt Creator.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Невізуальні компоненти – це компоненти невидимі при виконанні програми, а у режимі конструювання відображаються у вигляді іконки. Такі компоненти можна розміщати в будь-якому місці форми.

Основними невізуальними компонентами є системні діалоги. Існує всього 10 таких діалогів передбачених операційною системою Microsoft Windows: **OpenDialog, SaveDialog, OpenPictureDialog, SavePictureDialog, FontDialog, ColorDialog, PrintDialog, PrinterSetupDialog, FindDialog, ReplaceDialog.**

При виклику кожного з них з’являється нове діалогове вікно, і залежно від його типу, користувачу надаються певні можливості. Системні діалоги можуть бути викликані лише під час виконання програми.

Коротка характеристика:

OpenDialog – використовується для відкриття файлів.

SaveDialog - дає змогу зберігати файл з певним розширенням.

PrintDialog – дозволяє роздрукувати документ на папері.

І т.д.

У процесі роботи із системними діалогами, широко використовуються фільтри форматів. Вони задають параметри файлів, з якими взаємодіятиме діалог, встановлюючи дозволені розширення файлів.

Ще одним із видів невізуальних компонент є об’єкти класу QMenuBar. Вони розташовуються на верхній панелі віконного застосунку та виконують роль способу навігації функціями програми. Одним із різновидів цього меню є PopUpMenu, яке є додатковою вкладкою функцій, яка розгортається при наведенні на неї курсором.

Рис. 1 Навігаційне меню

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Створити віконний проект. Додати головне та контекстне меню, необхідні системні діалоги.
2. Реалізувати текстовий редактор і переглядач графічних файлів

Реалізація текстового редактора та переглядача графічних файлів

Файл main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[]) {
  QApplication a(argc, argv);
  MainWindow w;

  w.show();

  return a.exec();
}
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QLabel>
#include <QtCore>
#include <QtGui>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow) {
  ui->setupUi(this);
  this->setCentralWidget(ui->textEdit);
}

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

void MainWindow::on_actionNew_triggered() {
  currentFile.clear();
  ui->textEdit->setText(QString());
}

void MainWindow::on_actionOpen_triggered() {
  QString filename = QFileDialog::getOpenFileName((this), "Open the file");
  QFile file(filename);

  currentFile = filename;
```

```
  if (!file.open(QIODevice::ReadOnly | QFile::Text)) {
    QMessageBox::warning(this, "Warning",
                         "Cannot open file:" + file.errorString());
    return;
  }

  setWindowTitle(filename);

  QTextStream in(&file);
  QString text = in.readAll();

  ui->textEdit->setText(text);
  file.close();
}

void MainWindow::on_actionSave_as_triggered() {
  QString filename = QFileDialog::getSaveFileName(this, "Save as");
  QFile file(filename);

  if (!file.open(QFile::WriteOnly | QFile::Text)) {
    QMessageBox::warning(this, "Warning",
                         "Cannot save file:" + file.errorString());
    return;
  }

  currentFile = filename;

  setWindowTitle(filename);

  QTextStream out(&file);
  QString text = ui->textEdit->toPlainText();

  out << text;
  file.close();
}

void MainWindow::on_actionExit_triggered() { QApplication::quit(); }

void MainWindow::on_actionCopy_triggered() { ui->textEdit->copy(); }

void MainWindow::on_actionPaste_triggered() { ui->textEdit->paste(); }

void MainWindow::on_actionCut_triggered() { ui->textEdit->cut(); }

void MainWindow::on_actionUndo_triggered() { ui->textEdit->undo(); }

void MainWindow::on_actionRedo_triggered() { ui->textEdit->redo(); }

void MainWindow::on_actionOpen_Image_triggered() {
  QString filename = QFileDialog::getOpenFileName((this), "Open the file");
  QFile file(filename);

  currentFile = filename;

  if (!file.open(QIODevice::ReadOnly | QFile::Text)) {
    QMessageBox::warning(this, "Warning",
                         "Cannot open file:" + file.errorString());
    return;
  }
```

```
  setWindowTitle(filename);
  QPixmap image(currentFile);

  QLabel *imageLabel = new QLabel();
  imageLabel->setPixmap(image);
  imageLabel->show();
}
```

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtCore>
#include <QtGui>
#include <QFile>
#include <QFileDialog>
#include <QTextStream>
#include <QMessageBox>
#include <QObject>


QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
void AddRoot(QString name, QString Description);

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_actionNew_triggered();

    void on_actionOpen_triggered();

    void on_actionSave_as_triggered();

    void on_actionExit_triggered();

    void on_actionCopy_triggered();

    void on_actionPaste_triggered();

    void on_actionCut_triggered();

    void on_actionUndo_triggered();

    void on_actionRedo_triggered();

    void on_actionOpen_Image_triggered();
```

```
private:
    Ui::MainWindow *ui;
    QString currentFile="";
};
#endif // MAINWINDOW_H
```

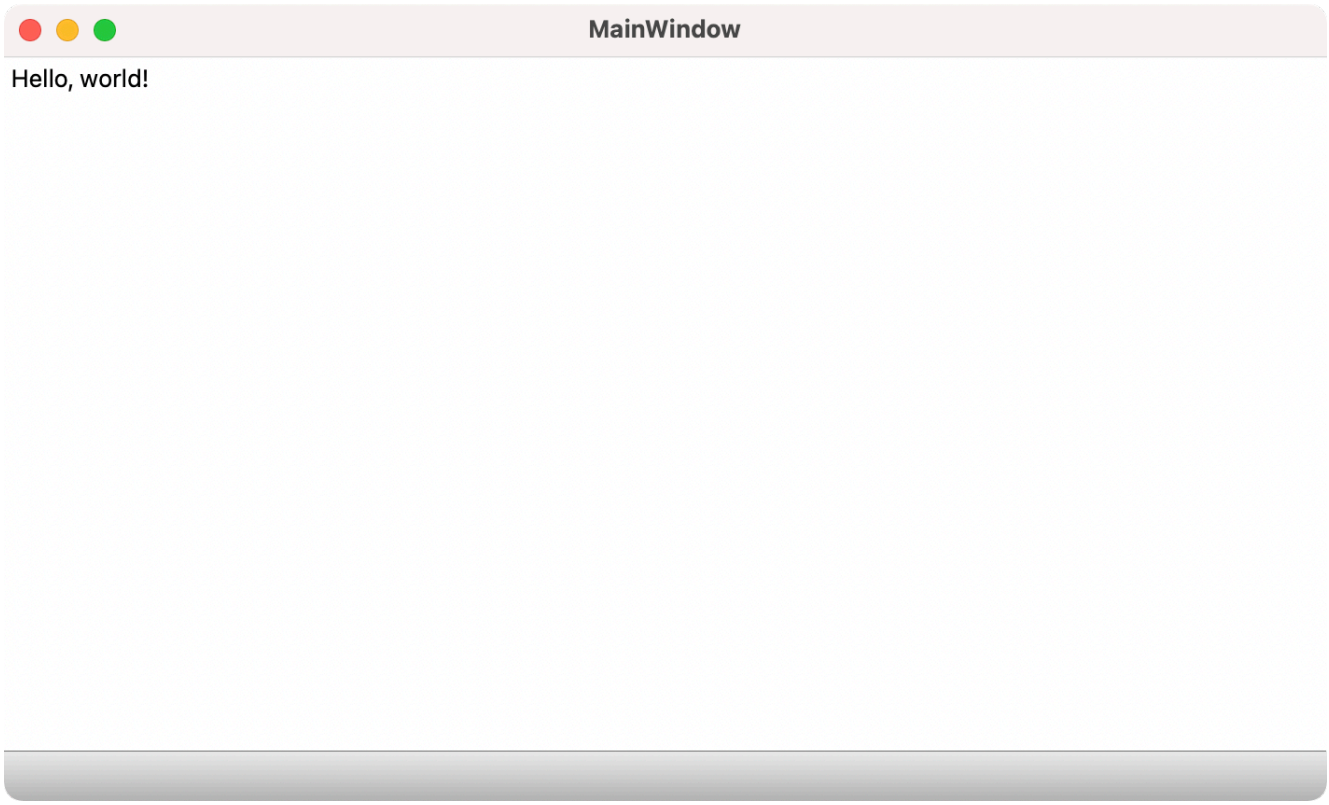


Рис. 2 Інтерфейс програми текстового редактора

Рис. 3 Інтерфейс програми переглядача картинок

ВИСНОВКИ

Виконавши лабораторну роботу №3, я ознайомився із невізуальними компонентами середовища розробки Qt Creator, створив та налаштував віконний проект з використовуючи ці компоненти.

ЛАБОРАТОРНА РОБОТА №4

Тема: Компоненти Qt Creator для представлення даних.
Мета: Створити віконний проект та продемонструвати використання компонент призначених для відображення та опрацювання даних.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Qt Creator, як і будь-яке інше середовище розробки надає змогу використовувати таблиці для організації даних. Для їх створення використовується компонента **TableWidget**. Вона є членом класу **QTableWidget**. Редагування таблиці може відбуватись за допомогою команд, написаних у редакторі коду. За допомогою методів **setRowCount(int value)** та **setColumnCount(int value)** задаються розмірність таблиці, а саме кількість рядків та стовпців. Методи **setHorizontalHeaderLabels()** та **setVerticalHeaderLabels**() дають назву кожному із стовпців та рядків. Скориставшись методами **setColumnWidth()** та **setRowHeight()** можна вказати розміри кожної з клітинок таблиці.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Ознайомитись із компонентою TableWidget.
2. Реалізувати гру.

Файл direction.h

```
#ifndef DIRECTION_H
#define DIRECTION_H

typedef unsigned char direction;

#define NO_DIRECTION 0b000

#define DIRECTION_UP 0b100
#define DIRECTION_DOWN 0b111
#define DIRECTION_RIGHT 0b101
#define DIRECTION_LEFT 0b110

inline direction oppositeTo(direction dir) {
    return (~dir & 0b11) | 0b100;
```

```
}

inline bool isOppositeTo(direction a, direction b) {
    return a == oppositeTo(b);
}

inline void moveCoordinates(int *x, int *y, direction dir) {
    switch (dir) {
    case DIRECTION_UP:
        --(*y);
        break;
    case DIRECTION_DOWN:
        ++(*y);
        break;
    case DIRECTION_LEFT:
        --(*x);
        break;
    case DIRECTION_RIGHT:
        ++(*x);
        break;
    }
}

#endif // DIRECTION_H
```

Файл gamecontroller.cpp

```
#include "gamecontroller.h"

GameController::GameController(int fieldSize) {
  this->fieldSize = fieldSize;
  allocateField();
}

GameController::~GameController() { deleteField(); }

void GameController::forceFieldRedraw() {
  for (int x = 0; x < fieldSize; ++x) {
    for (int y = 0; y < fieldSize; ++y) {
      emit tileUpdateEvent(x, y, tileToEnum(field[x][y]));
    }
  }
}

inline void GameController::updateTileAt(int x, int y, tile newTile) {
  field[x][y] = newTile;
  emit tileUpdateEvent(x, y, tileToEnum(newTile));
}

TileType GameController::tileToEnum(tile t) {
  switch (t & TILE_TYPE_MASK) {
  case EMPTY_TILE:
    return EMPTY;
  case SNAKE_TILE:
    return SNAKE_BODY;
  case FRUIT_TILE:
    return FRUIT;
  case WALL_TILE:
    return WALL;
```

```
  default:
    return EMPTY;
  }
}

void GameController::resetGame() { updateGameState(READY_FOR_START); }

void GameController::startGame() {
  if (gameState() != READY_FOR_START)
    return;

  updateGameState(RUNNING);
}

void GameController::tick() {
  tiletype type = getTileTypeInFront();

  if (type == WALL_TILE || type == SNAKE_TILE) {
    updateGameState(STOPPED);
    return;
  }

  if (type == FRUIT_TILE) {
    placeFruit();
    ++snakeLength;
  } else {
    moveTail();
  }
  moveHead(newSnakeHeadDirection);

  snakeHeadDirection = newSnakeHeadDirection;
}

void GameController::getRandomCoordinates(int *x, int *y) {
  *x = rand() % fieldSize;
  *y = rand() % fieldSize;
}

void GameController::moveHead(const direction headDirection) {
  int px = snakeHeadX, py = snakeHeadY;
  moveCoordinates(&snakeHeadX, &snakeHeadY, headDirection);

  updateTileAt(px, py, convertHeadToBodyTile(field[px][py]));
  updateTileAt(snakeHeadX, snakeHeadY,
               createSnakeHeadTile(oppositeTo(headDirection)));
}

void GameController::moveTail() {
  int tailX = snakeHeadX, tailY = snakeHeadY;
  tile tile = field[tailX][tailY];
  int px = 0, py = 0;
  while (!isSnakeTail(tile)) {
    px = tailX;
    py = tailY;

    direction dir = getDirection(tile);
    moveCoordinates(&tailX, &tailY, dir);
    tile = field[tailX][tailY];
  }
  updateTileAt(tailX, tailY, EMPTY_TILE);
```

```
  updateTileAt(px, py, createSnakeTailTile());
}

void GameController::updateGameState(GameState newState) {
  this->state = newState;
  switch (newState) {
  case STOPPED:
    killTimer(timerId);
    emit gameStopEvent(snakeLength);
    break;
  case RUNNING:
    timerId = startTimer(BASE_TICK_TIME_MS / gameSpeed);
    emit gameStartEvent();
  case READY_FOR_START:
    prepareFieldForStart();
    break;
  }
}

void GameController::allocateField() {
  this->field = new tile *[fieldSize];
  for (int i = 0; i < fieldSize; ++i) {
    field[i] = new tile[fieldSize];
  }
}

void GameController::prepareFieldForStart() {
  for (int i = 0; i < fieldSize; ++i) {
    for (int j = 0; j < fieldSize; ++j) {
      updateTileAt(i, j, EMPTY_TILE);
    }
  }

  // UL corner
  updateTileAt(2, 2, WALL_TILE);
  updateTileAt(3, 2, WALL_TILE);
  updateTileAt(4, 2, WALL_TILE);

  updateTileAt(2, 4, WALL_TILE);
  updateTileAt(3, 4, WALL_TILE);
  updateTileAt(4, 4, WALL_TILE);

  // UR corner
  updateTileAt(fieldSize - 3, 2, WALL_TILE);
  updateTileAt(fieldSize - 4, 2, WALL_TILE);
  updateTileAt(fieldSize - 5, 2, WALL_TILE);

  updateTileAt(fieldSize - 3, 4, WALL_TILE);
  updateTileAt(fieldSize - 4, 4, WALL_TILE);
  updateTileAt(fieldSize - 5, 4, WALL_TILE);

  // LL corner
  updateTileAt(1, fieldSize - 1, WALL_TILE);
  updateTileAt(0, fieldSize - 1, WALL_TILE);
  updateTileAt(0, fieldSize - 2, WALL_TILE);

  // LR corner
  updateTileAt(fieldSize - 2, fieldSize - 1, WALL_TILE);
  updateTileAt(fieldSize - 1, fieldSize - 1, WALL_TILE);
  updateTileAt(fieldSize - 1, fieldSize - 2, WALL_TILE);
```

```
  // lines
  for (int y = 4; y < fieldSize - 4; ++y) {
    updateTileAt(3, y, WALL_TILE);
    updateTileAt(fieldSize - 4, y, WALL_TILE);
  }

  prepareSnake();
  placeFruit();
}

void GameController::prepareSnake() {
  snakeLength = INITIAL_SNAKE_LENGTH;
  snakeHeadX = INITIAL_SNAKE_POSITION_X;
  snakeHeadY = INITIAL_SNAKE_POSITION_Y;

  updateTileAt(snakeHeadX, snakeHeadY, createSnakeHeadTile(DIRECTION_LEFT));

  newSnakeHeadDirection = snakeHeadDirection = DIRECTION_RIGHT;

  for (int dx = 1; dx < INITIAL_SNAKE_LENGTH - 1; ++dx) {
    const int bodyX = snakeHeadX - dx;
    updateTileAt(bodyX, snakeHeadY, createSnakeBodyTile(DIRECTION_LEFT));
  }

  const int tailX = snakeHeadX - INITIAL_SNAKE_LENGTH + 1;
  updateTileAt(tailX, snakeHeadY, createSnakeTailTile());
}

void GameController::timerEvent(QTimerEvent *event) {
  if (event->timerId() != this->timerId)
    return;
  tick();
}

void GameController::setGameSpeed(int speed) {
  if (gameState() != RUNNING) {
    this->gameSpeed = speed;
  }
}

tiletype GameController::getTileTypeInFront() {
  int x = snakeHeadX, y = snakeHeadY;
  moveCoordinates(&x, &y, newSnakeHeadDirection);

  if (x < 0 || x >= fieldSize)
    return WALL_TILE;
  if (y < 0 || y >= fieldSize)
    return WALL_TILE;

  return getTileType(field[x][y]);
}

void GameController::deleteField() {
  for (int i = 0; i < fieldSize; ++i) {
    delete[] field[i];
  }
  delete[] field;
}
```

```
void GameController::placeFruit() {
  int x, y;
  do {
    getRandomCoordinates(&x, &y);
  } while (field[x][y] != EMPTY_TILE);
  updateTileAt(x, y, createFruitTile());
}

void GameController::tryChangeDirection(direction newDirection) {
  if (gameState() != RUNNING)
    return;

  if (!isOppositeTo(snakeHeadDirection, newDirection)) {
    newSnakeHeadDirection = newDirection;
  }
}
```

Файл gamecontroller.h

```
#ifndef GAMECONTROLLER_H
#define GAMECONTROLLER_H

#include <QObject>
#include <QTimerEvent>

#include <TileType.h>
#include <direction.h>
#include <tiles.h>

#define BASE_TICK_TIME_MS 1000

enum GameState { READY_FOR_START, RUNNING, STOPPED };

class GameController : public QObject {
  Q_OBJECT

public:
  GameController(int fieldSize);
  ~GameController();

  GameState gameState() { return state; }

  void forceFieldRedraw();

  void snakeLeft() { tryChangeDirection(DIRECTION_LEFT); }
  void snakeRight() { tryChangeDirection(DIRECTION_RIGHT); }
  void snakeUp() { tryChangeDirection(DIRECTION_UP); }
  void snakeDown() { tryChangeDirection(DIRECTION_DOWN); }

  void resetGame();
  void startGame();

  void setGameSpeed(int speed);

signals:
  void gameStartEvent();

  void tileUpdateEvent(int x, int y, TileType newType);
```

```
  void gameStopEvent(int score);

private slots:
  void timerEvent(QTimerEvent *event);

private:
  const int INITIAL_SNAKE_LENGTH = 3;
  const int INITIAL_SNAKE_POSITION_X = 5;
  const int INITIAL_SNAKE_POSITION_Y = 3;

  GameState state;
  int timerId;

  int fieldSize;
  tile **field;

  int snakeHeadX;
  int snakeHeadY;
  direction snakeHeadDirection;
  direction newSnakeHeadDirection;

  int snakeLength;

  int gameSpeed = 3;

  tiletype getTileTypeInFront();

  void moveHead(direction headDirection);
  void moveTail();

  void updateGameState(GameState newState);

  void allocateField();
  void prepareFieldForStart();
  void prepareSnake();
  void deleteField();

  void placeFruit();

  void tick();
  void getRandomCoordinates(int *x, int *y);

  void tryChangeDirection(direction newDirection);

  void updateTileAt(int x, int y, tile newTile);
  TileType tileToEnum(tile t);
};

#endif // GAMECONTROLLER_H
```

Файл main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
```

```
    w.show();
    return a.exec();
}
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"

#define PLAYING_FIELD_SIZE_TILES 16
#define PLAYING_FIELD_TILE_SIZE 32

GamePalette desertPalette = {
    Qt::black,
    Qt::darkGray,
    Qt::darkCyan,
    Qt::darkMagenta,
};

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow) {
  ui->setupUi(this);

  setUpPlayingField();
  setUpGameController();

  gameController->startGame();
}

MainWindow::~MainWindow() {
  delete ui;

  deleteBrushes();
}

void MainWindow::deleteBrush(QBrush **brush) {
  if (*brush != nullptr) {
    delete *brush;
    *brush = nullptr;
  }
}

void MainWindow::deleteBrushes() {
  deleteBrush(&backgroundBrush);
  deleteBrush(&wallBrush);
  deleteBrush(&fruitBrush);
  deleteBrush(&snakeBrush);
}

void MainWindow::onTileUpdate(int x, int y, TileType type) {
  auto cell = cellAt(x, y);
  switch (type) {
  case SNAKE_BODY:
    cell->setBackground(*snakeBrush);
    break;
  case FRUIT:
    cell->setBackground(*fruitBrush);
    break;
  case EMPTY:
    cell->setBackground(*backgroundBrush);
```

```
    break;
  case WALL:
    cell->setBackground(*wallBrush);
    break;
  }
}

void MainWindow::onGameStopped(int score) {
  QMessageBox gameLost(this);
  gameLost.setWindowTitle("You lost");
  gameLost.setText(tr("Your points: %1").arg(score));
  gameLost.setStandardButtons(QMessageBox::Retry);

  auto button = gameLost.exec();
  gameController->resetGame();

  if (button == QMessageBox::Retry) {
    gameController->startGame();
  }
}

QTableWidgetItem *MainWindow::cellAt(int x, int y) {
  QTableWidgetItem *item = ui->tableWidget->item(y, x);
  if (item == nullptr) {
    item = new QTableWidgetItem();
    item->setTextAlignment(Qt::AlignCenter);
    ui->tableWidget->setItem(y, x, item);
  }
  return item;
}

void MainWindow::keyPressEvent(QKeyEvent *keyEvent) {
  switch (keyEvent->key()) {
  case Qt::Key_W:
  case Qt::Key_Up:
  case Qt::Key_K:
    gameController->snakeUp();
    keyEvent->accept();
    break;
  case Qt::Key_S:
  case Qt::Key_Down:
  case Qt::Key_J:
    gameController->snakeDown();
    keyEvent->accept();
    break;
  case Qt::Key_A:
  case Qt::Key_Left:
  case Qt::Key_H:
    gameController->snakeLeft();
    keyEvent->accept();
    break;
  case Qt::Key_D:
  case Qt::Key_Right:
  case Qt::Key_L:
    keyEvent->accept();
    gameController->snakeRight();
    break;
  }
}
```

```
void MainWindow::setUpPlayingField() {
  ui->tableWidget->setColumnCount(PLAYING_FIELD_SIZE_TILES);
  ui->tableWidget->setRowCount(PLAYING_FIELD_SIZE_TILES);

  ui->tableWidget->horizontalHeader()->setMinimumSectionSize(0);
  for (int i = 0; i < PLAYING_FIELD_SIZE_TILES; ++i) {
    ui->tableWidget->setColumnWidth(i, PLAYING_FIELD_TILE_SIZE);
    ui->tableWidget->setRowHeight(i, PLAYING_FIELD_TILE_SIZE);
  }

  ui->tableWidget->horizontalHeader()->setVisible(false);
  ui->tableWidget->verticalHeader()->setVisible(false);

  QSize size = this->size();
#ifdef _WIN32
  size.setWidth(PLAYING_FIELD_SIZE_TILES * PLAYING_FIELD_TILE_SIZE);
  size.setHeight(PLAYING_FIELD_SIZE_TILES * PLAYING_FIELD_TILE_SIZE +
                 ui->menubar->height());
#elif TARGET_OS_MAC
  size.setWidth(PLAYING_FIELD_SIZE_TILES * PLAYING_FIELD_TILE_SIZE);
  size.setHeight(PLAYING_FIELD_SIZE_TILES * PLAYING_FIELD_TILE_SIZE);
#endif
  this->resize(size);
  this->setMaximumSize(size);
  this->setMinimumSize(size);

  updatePalette(&desertPalette);
}

void MainWindow::setUpGameController() {
  // focus on window to receive key events
  this->setFocusPolicy(Qt::StrongFocus);
  this->setFocus();

  gameController = new GameController(PLAYING_FIELD_SIZE_TILES);

  connect(gameController, SIGNAL(tileUpdateEvent(int, int, TileType)), this,
          SLOT(onTileUpdate(int, int, TileType)));
  connect(gameController, SIGNAL(gameStopEvent(int)), this,
          SLOT(onGameStopped(int)));

  gameController->resetGame();
  gameController->setGameSpeed(3);
}

QBrush *MainWindow::createBrush(Qt::GlobalColor color) {
  return new QBrush(color);
}

void MainWindow::updatePalette(GamePalette *newPalette) {
  gamePalette = newPalette;

  deleteBrushes();

  backgroundBrush = createBrush(gamePalette->backgroundColor);
  wallBrush = createBrush(gamePalette->wallColor);
  snakeBrush = createBrush(gamePalette->snakeColor);
  fruitBrush = createBrush(gamePalette->fruitColor);

  if (gameController != nullptr) {
```

```
    gameController->forceFieldRedraw();
  }
}
```

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QKeyEvent>
#include <QMainWindow>
#include <QMessageBox>
#include <QRgb>
#include <QTableWidgetItem>

#include <TileType.h>
#include <gamecontroller.h>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

struct GamePalette {
  Qt::GlobalColor backgroundColor;
  Qt::GlobalColor snakeColor;
  Qt::GlobalColor wallColor;
  Qt::GlobalColor fruitColor;
};

class MainWindow : public QMainWindow {
  Q_OBJECT

public:
  MainWindow(QWidget *parent = nullptr);
  ~MainWindow();

private slots:
  void onTileUpdate(int x, int y, TileType type);
  void onGameStopped(int score);

private:
  Ui::MainWindow *ui;
  GameController *gameController = nullptr;
  GamePalette *gamePalette = nullptr;

  QBrush *backgroundBrush = nullptr;
  QBrush *snakeBrush = nullptr;
  QBrush *wallBrush = nullptr;
  QBrush *fruitBrush = nullptr;

  QTableWidgetItem *cellAt(int x, int y);

  void keyPressEvent(QKeyEvent *event);

  void setUpPlayingField();
  void setUpGameController();

  void updatePalette(GamePalette *newPalette);
```

```
  QBrush *createBrush(Qt::GlobalColor color);
  void deleteBrushes();
  void deleteBrush(QBrush **brush);
};
#endif // MAINWINDOW_H
```

Файл mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>MainWindow</class>
  <widget class="QMainWindow" name="MainWindow">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>567</width>
        <height>498</height>
      </rect>
    </property>
    <property name="sizePolicy">
      <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
        <horstretch>0</horstretch>
        <verstretch>0</verstretch>
      </sizepolicy>
    </property>
    <property name="minimumSize">
      <size>
        <width>0</width>
        <height>0</height>
      </size>
    </property>
    <property name="windowTitle">
      <string>Snake game</string>
    </property>
    <widget class="QWidget" name="centralwidget">
      <property name="sizePolicy">
        <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
          <horstretch>0</horstretch>
          <verstretch>0</verstretch>
        </sizepolicy>
      </property>
      <layout class="QVBoxLayout" name="verticalLayout">
        <property name="leftMargin">
          <number>0</number>
        </property>
        <property name="topMargin">
          <number>0</number>
        </property>
        <property name="rightMargin">
          <number>0</number>
        </property>
        <property name="bottomMargin">
          <number>0</number>
        </property>
        <item>
          <widget class="QTableWidget" name="tableWidget">
            <property name="frameShape">
              <enum>QFrame::NoFrame</enum>
            </property>
            <property name="verticalScrollBarPolicy">
```

```
            <enum>Qt::ScrollBarAlwaysOff</enum>
           </property>
           <property name="horizontalScrollBarPolicy">
            <enum>Qt::ScrollBarAlwaysOff</enum>
           </property>
           <property name="autoScroll">
            <bool>false</bool>
           </property>
           <property name="editTriggers">
            <set>QAbstractItemView::NoEditTriggers</set>
           </property>
           <property name="tabKeyNavigation">
            <bool>false</bool>
           </property>
           <property name="selectionMode">
            <enum>QAbstractItemView::NoSelection</enum>
           </property>
           <property name="showGrid">
            <bool>false</bool>
           </property>
           <property name="gridStyle">
            <enum>Qt::NoPen</enum>
           </property>
           <property name="cornerButtonEnabled">
            <bool>false</bool>
           </property>
           <property name="rowCount">
            <number>0</number>
           </property>
           <attribute name="horizontalHeaderVisible">
            <bool>false</bool>
           </attribute>
           <attribute name="verticalHeaderVisible">
            <bool>false</bool>
           </attribute>
          </widget>
         </item>
        </layout>
       </widget>
       <widget class="QMenuBar" name="menubar">
        <property name="geometry">
         <rect>
          <x>0</x>
          <y>0</y>
          <width>567</width>
          <height>20</height>
         </rect>
        </property>
        <widget class="QMenu" name="menuGame">
         <property name="title">
          <string>Game</string>
         </property>
        </widget>
        <addaction name="menuGame" />
       </widget>
      </widget>
      <resources />
      <connections />
     </ui>
```

Файл tiles.cpp

```
#include "tiles.h"

tile createSnakeHeadTile(direction directionToNextTile)
{
    return createSnakeBodyTile(directionToNextTile);
}

tile createSnakeBodyTile(direction directionToNextTile)
{
    return (directionToNextTile << 2) | SNAKE_TILE;
}

tile createSnakeTailTile()
{
    return SNAKE_TILE;
}
```

Файл tiles.h

```
#ifndef TILES_H
#define TILES_H

#include <direction.h>

#define EMPTY_TILE 0b00
#define SNAKE_TILE 0b01
#define FRUIT_TILE 0b10
#define WALL_TILE  0b11

typedef unsigned char tiletype;

#define DIRECTION_MASK 0b11100
#define TILE_TYPE_MASK 0b11

/* tile bit layout is as follows
 * 0: 2 bits - tile type
 * 2: 3 bits - next snake tile (if applicable)
 */
typedef unsigned char tile;

tile createSnakeHeadTile(direction nextTileDirection);
tile createSnakeBodyTile(direction nextTileDirection);
tile createSnakeTailTile();

inline tiletype getTileType(tile t) { return t & TILE_TYPE_MASK; }

inline tile createFruitTile() { return FRUIT_TILE; }
inline tile createEmptyTile() { return EMPTY_TILE; }

inline tile convertHeadToBodyTile(tile t) { return t; }

inline bool isEmpty(tile t) { return (t & TILE_TYPE_MASK) == EMPTY_TILE; }
inline bool isSnake(tile t) { return (t & TILE_TYPE_MASK) == SNAKE_TILE; }
inline bool isSnakeTail(tile t) { return isSnake(t) && ((t & DIRECTION_MASK) ==
NO_DIRECTION); }
inline bool isWallTile(tile t) { return (t & TILE_TYPE_MASK) == WALL_TILE; }

inline direction getDirection(tile t) { return (t & DIRECTION_MASK) >> 2; }
```

```
#endif // TILES_H
```

Файл TileType.h

```
#ifndef TILETYPE_H
#define TILETYPE_H

enum TileType {
    EMPTY,
    SNAKE_BODY,
    FRUIT,
    WALL
};

#endif // TILETYPE_H
```

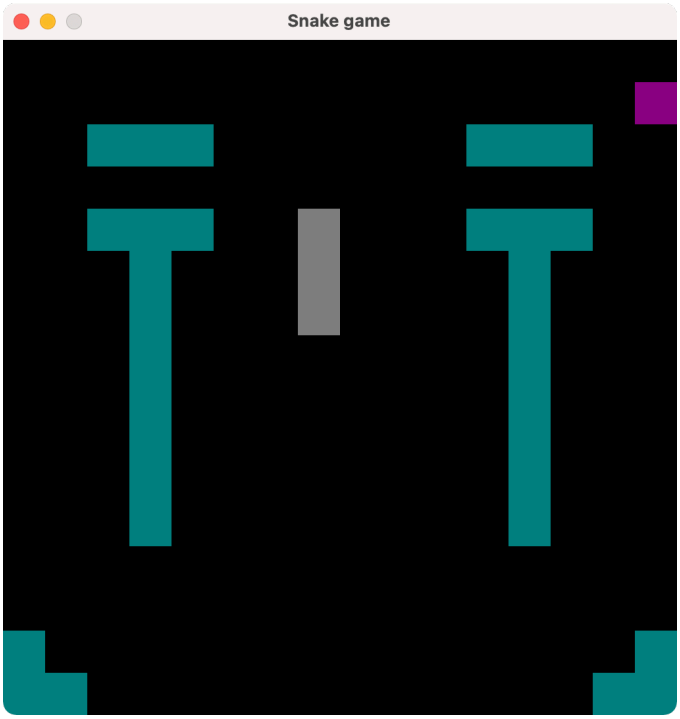


Рис. 1 Реалізація гри

ВИСНОВКИ

Виконавши лабораторну роботу №4, я ознайомився із організацією таблиць за допомогою компоненти TableWidget середовища розробки Qt Creator, створив та налаштував віконний проект з цією візуальною компонентою.