**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

Кафедра програмних засобів

(повна назва кафедри, циклової комісії)

**КУРСОВИЙ ПРОЕКТ**

**(РОБОТА)**

з дисципліни «Об’єктно-орієнтоване програмування»

(назва дисципліни)

на тему: «Розробка редактора плану приміщень»

Студентів 2 курсу

КНТ-127, КНТ-137 груп

напряму підготовки Комп’ютерні науки

спеціальності інженерія

програмного забезпечення

Бережний О. Ю.

(прізвище та ініціали)

Козлов В. В.

(прізвище та ініціали)

Судаков В. Р.

(прізвище та ініціали)

Керівник Доцент к.н.т., Табунщик Г. В.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала

Кількість балів:

Оцінка: ECTS

Члени комісії Табунщик Г. В.

(підпис) (прізвище та ініціали)

Миронова Н. О.

(підпис) (прізвище та ініціали)

Каплієнко Т. І.

(підпис) (прізвище та ініціали)

Завдання  
на курсовий проект студентів

Бережного О. Ю., Козлова В. В., Судакова В. Р.

1. Тема проекту: Розробка редактора плану приміщень

2. Термін здачі студентами закінченого проекту: 28 грудня 2018

3. Вихідні дані до проекту: Реалізувати редактор плану приміщень

дані: інформація про розміри об’єктів, позиції та кути нахилу.

вихідні дані: файл проекту та зображення спроектованої кімнати

4. Зміст розрахунково-пояснювальної записки:

1. Аналіз предметної області;

2. Аналіз програмних засобів;

3. Основні рішення з реалізації компонентів системи;

4. Керівництво програміста;

5. Керівництво оператора;

Висновки

Додаток А. Текст програми

Додаток Б. Інтерфейс програми

Додаток В. Слайди презентації

5. Дата видачі завдання: 16 вересня 2018 року

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів курсового  проекту (роботи) | Термін виконання етапів  проекту (роботи) | Примітка |
| 1. | Аналіз індивідуального завдання. | 1-2 тиждень |  |
| 2. | Аналіз програмних засобів, що будуть використовуватись в роботі. | 3-4 тиждень |  |
| 3. | Аналіз структур даних, що необхідно використати в курсової роботі. | 4-5 тиждень |  |
| 4. | Вивчення можливостей програмної реалізації структур даних та інтерфейсу користувача. | 5-6 тиждень |  |
| 5. | Оформлення відповідних пунктів пояснювальної записки. | 6-8 тиждень | Розділи 1,2 ПЗ |
| 6. | Аналіз вимог до апаратних засобів | 9 тиждень |  |
| 7. | Розробка програмного забезпечення | 10-15 тиждень |  |
| 8. | Оформлення, відповідних пунктів пояснювальної записки. | 9-16 тиждень | Розділи 3-5 ПЗ |
| 9. | Захист курсової роботи. | 17 тиждень |  |

Студент Бережний О. Ю.

Студент Козлов В. В.

Студент Судаков В. Р.

Керівник Табунщик Г. В.

«\_\_\_\_\_\_\_» 2018 р.

Реферат

Мета роботи – розробка програмного забезпечення для проектування та редагування плану приміщень або внутрішнього інтер’єру.

Завданням курсового проекту є розробка програми з мети роботи мовою С++ в інтегрованому середовищі розробки Qt Creator.

Проведено аналіз предметної області, досліджено аналогічне програмне забезпечення, а також більшість існуючих методів та програмних засобів для вирішення завдання.

Досліджено особливості мови програмування та середовища розробки.

Здійснено опис прийнятих рішень, реалізованих класів, наведено опис полів та методів реалізованих класів.

Ключові слова: Об’єктно-орієнтоване програмування, програмування, середовище розробки, клас, об’єкт, алгоритм, база даних, планувальник приміщень, програміст, оператор.

Зміст

[Завдання на курсовий проект студентів 2](#_Toc533648982)

[Календарний план 3](#_Toc533648983)

[Реферат 4](#_Toc533648984)

[Зміст 5](#_Toc533648985)

[Вступ 7](#_Toc533648986)

[1 Аналіз предметної області 8](#_Toc533648987)

[1.1 Огляд існуючих методів вирішення завдання 8](#_Toc533648988)

[1.2 Огляд існуючих планувальників 9](#_Toc533648989)

[1.2.1 Smart Draw 9](#_Toc533648990)

[1.2.2 Floor Planner 10](#_Toc533648991)

[1.2.3 Planner 5D 11](#_Toc533648992)

[1.3 Постановка завдання 12](#_Toc533648993)

[2 Аналіз програмних засобів 13](#_Toc533648994)

[2.1 Огляд особливостей мови програмування 13](#_Toc533648995)

[2.2 Огляд особливостей обраного компілятору 14](#_Toc533648996)

[2.3 Огляд можливостей та особливостей обраної системи керування базами даних - SQLite 14](#_Toc533648997)

[2.4 Огляд особливостей застосованого текстового формату збереження даних - JSON 15](#_Toc533648998)

[2.5 Огляд класів Qt, що використовуються в роботі 17](#_Toc533648999)

[2.6 Висновки з розділу 20](#_Toc533649000)

[3 Основні рішення з реалізації компонентів системи 21](#_Toc533649001)

[3.1 Основні рішення щодо уявлення даних системи 21](#_Toc533649002)

[3.2 Основні розроблені алгоритми 33](#_Toc533649003)

[3.3 Особливості реалізації системи 34](#_Toc533649004)

[3.4 Результати тестування системи 38](#_Toc533649005)

[3.5 Основні рішення щодо збереження та відтворення інформації. 39](#_Toc533649006)

[3.5.1 Запити SQL, що були використані 39](#_Toc533649007)

[3.5.2 Основні рішення використані при взаємодії з БД 40](#_Toc533649008)

[3.5.3 Основні рішення щодо збереження даних в JSON файлах 41](#_Toc533649009)

[3.6 Висновки з розділу 41](#_Toc533649010)

[4 Керівництво програміста 43](#_Toc533649011)

[4.1 Призначення та умови застосування програми 43](#_Toc533649012)

[4.2 Характеристики програми 43](#_Toc533649013)

[4.3 Звертання до програми 47](#_Toc533649014)

[4.4 Вхідні та вихідні дані 48](#_Toc533649015)

[4.5 Повідомлення 49](#_Toc533649016)

[5 Інструкція користувача 50](#_Toc533649017)

[5.1 Призначення програми 50](#_Toc533649018)

[5.2 Умови використання програми 50](#_Toc533649019)

[5.3 Як запустити програму 50](#_Toc533649020)

[5.4 Виконання програми 51](#_Toc533649021)

[5.5 Повідомлення користувачу 54](#_Toc533649022)

[Висновки 56](#_Toc533649023)

[Перелік посилань 57](#_Toc533649024)

[Додаток А Текст програми 58](#_Toc533649025)

[Додаток Б Інтерфейс програми 101](#_Toc533649026)

[Додаток В Слайди презентації 103](#_Toc533649027)

Вступ

Актуальність роботи полягає у дослідженні існуючих програм для проектування плану приміщень, аналізі їх переваг та недоліків, необхідності створення програмного продукту з простим та інтуїтивно зрозумілим інтерфейсом, та можливістю збереження готового проекту для подальшого редагування.

Метою даного курсового проекту є розробка програмного забезпечення, що реалізує можливість проектування та редагування плану і внутрішнього інтер’єру приміщень.

Задачі курсового проекту: аналіз предметної області, розробка відповідних структур даних, проектування графічного інтерфейсу, розробка програми мовою С++ в інтегрованому середовищі розробки Qt Creator та її тестування.

1. Аналіз предметної області

В даному розділі розглянуто існуючі методи вирішення завдання та огляд програм, що реалізують проектування та редагування планів приміщення.

* 1. Огляд існуючих методів вирішення завдання

Програма для проектування та редагування плану приміщень призначена для заощадження часу, фізичних та матеріальних сил людей, які планують будувати приміщення або бажають змінити інтер’єр у ньому. Тобто програма для проектування та редагування плану приміщень для зручної роботи повинна мати інтуїтивно зрозумілий інтерфейс, базу «шаблонних» предметів інтер’єру та підтримку української мовної локалізації.

Проектування — процес створення проекту, прототипу, прообразу майбутнього об'єкта.

Редагування — приведення об'єкта редагування у відповідність із чинними у певний час нормами, а також його творча оптимізація, метою чого є отримання заданого соціального ефекту.

План – креслення, що відбиває на площині в умовних знаках і певному масштабі предмет, споруду та ін..

Приміщення – будівля, квартира чи окрема кімната, а якій поміщається хто-, що-небудь, яка використовується або буде використовуватись для чогось.

Інтер’єр – архітектурно й художньо оздоблена внутрішня частина будинку, приміщення, що забезпечує комфорт людині та визначає функціональне призначення приміщення.

Інтуїтивний зрозумілий інтерфейс — сукупність засобів і правил, що забезпечують взаємодію користувача із програмним забезпеченням, яке наслідує розповсюджені і звичні для багатьох принципи побудови взаємодії. При роботі з таким інтерфейсом користувач використовує звичні для нього дії особливо не задумуючись над тим що та як зробити, його дії ніби інтуїтивні, звідки й назва.

База «шаблонних» предметів інтер’єру – це набір найуживаніших елементів інтер’єру: двері, ліжко, шафа, лампа, телевізор та т. ін..

Мовна локалізація — культурна адаптація продукту до особливостей певної країни, регіону чи групи населення.

* 1. Огляд існуючих планувальників
     1. Smart Draw

Офіційний сайт програми: www.smartdraw.com.

Переваги даного програмного продукту:

* наявність 7 днів безкоштовного використання продукту;
* можливість вибору форми приміщення (квадратна, прямокутна) перед створенням проекту;
* наявність готових шаблонів приміщень;
* схожість інтерфейсу до MS Office;
* функція застосування текстур до фігур;
* підтримка «гарячих клавіш»;
* демонстрація розмірів об’єкту під час проектування;

Недоліками даного програмного продукту:

* часова обмеженість безкоштовного використання;
* відсутність функції завдання розмірів площини перед створенням проекту;
* погана читабельність розмірів на фоні;
* відсутність української локалізації.

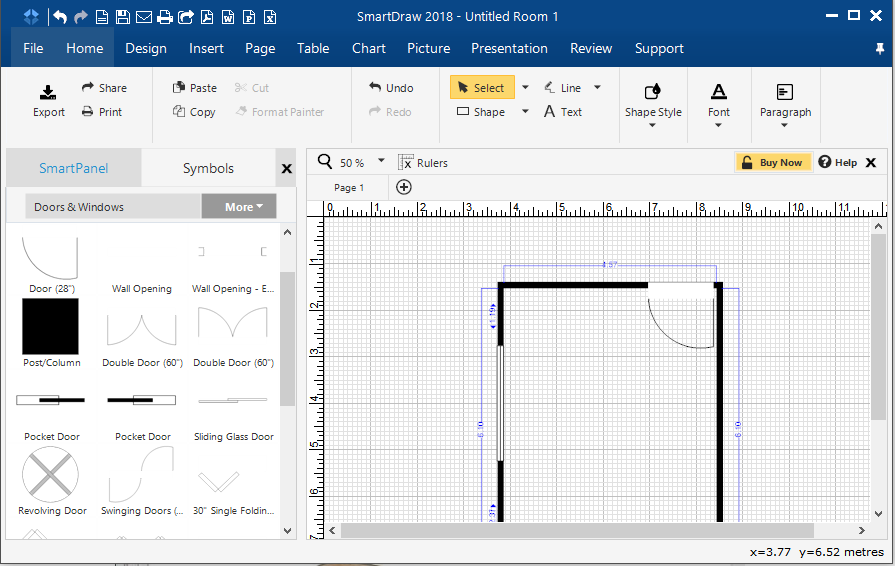


Рисунок 1.2.1 – Інтерфейс програми Smart Draw

* + 1. Floor Planner

Офіційний сайт програми: www.floorplanner.com.

Переваги даного програмного продукту:

* наявність функції швидкого перегляду поточного плану;
* легкодоступність елементарних елементів (стіни, вікна, двері тощо);
* наявність меню операцій у верхній частині екрану;
* безкоштовність;
* докладна довідка користувача.

Недоліками даного програмного продукту:

* відсутність десктопної версії продукту;
* відсутність розмірів об’єктів;
* не підтримуються «гарячі клавіші»;
* відсутність української локалізації.

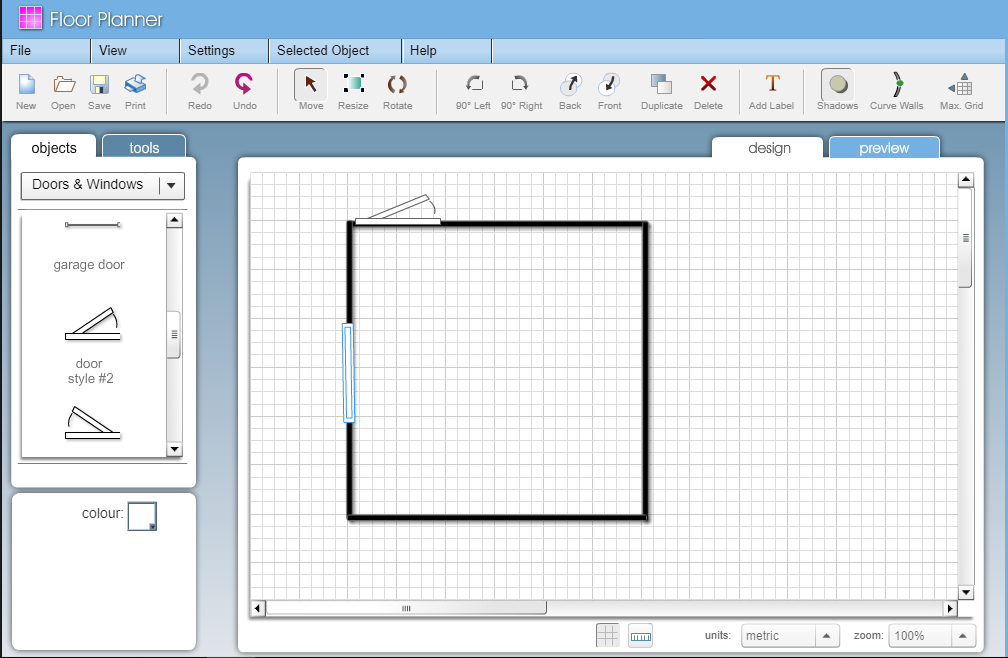


Рисунок 1.2.2 – Інтерфейс програми Floor Planner

* + 1. Planner 5D

Офіційний сайт програми: www.planner5d.com.

Переваги даного програмного продукту:

* велика база предметів інтер’єру;
* підтримка декількох мов;
* безкоштовність;
* кросплатформність;

Недоліками даного програмного продукту:

* орієнтованість у більшому ступені на 3-х вимірний простір;
* не підтримуються «гарячі клавіші»;
* відсутність української локалізації.

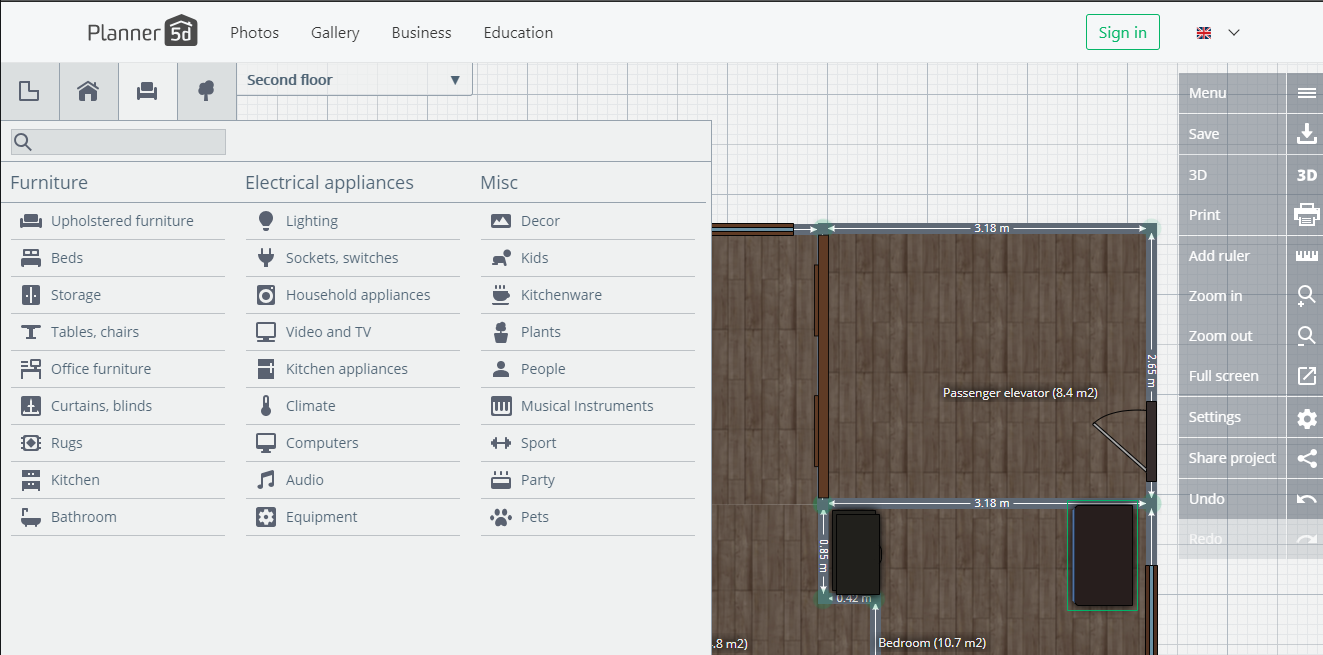


Рисунок 1.2.3 – Інтерфейс програми Planner 5D

* 1. Постановка завдання

Задача поставлена перед програмою – надати користувачу інструменти для швидкого створення та редагування плану приміщень.

Функції, що будуть реалізовані системою:

* проектування форми приміщення (стін);
* додавання елементів інтер’єру;
* можливість зберігати проект та експортувати результат роботи у зображення;
* можливість відкривати попередньо збережені проекти;
* можливість змінювати масштаб робочої області;
* можливість змінювати мову інтерфейсу в програмі (англійська, російська, українська);
* робота з програмою за допомогою «гарячих клавіш»;
* «гумовий» макет (розмір вмісту змінюється разом з розмірами вікон, щоб вони були завжди «заповнені»).

1. Аналіз програмних засобів

В даному розділі розглянуті основні особливості програмних засобів, за допомогою який реалізовано курсовий проект.

* 1. Огляд особливостей мови програмування

C++ – це мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом 1979 року [1].

Мова С++ має синтаксис, заснований на синтаксисі С.

Основними відмінностями С++ від С є:

* підтримка об'єктно-орієнтованого програмування;
* підтримка узагальненого програмування (шаблони);
* оновлена та розширена стандартна бібліотека;
* додаткові типи даних;
* обробка винятків;
* вбудовані функції;
* перевантаження операторів;
* перевантаження імен функцій;
* посилання і оператори управління вільно розподіленою пам'яттю [2].

Швидкість роботи програм на С++ практично не поступається програмам на С, при цьому С++ реалізовано більше можливостей і засобів. На мові C++ розробляють програми для найрізноманітніших платформ і систем. Окрім цього існує можливість роботи на низькому рівні з пам'яттю, адресами, портами, можливість створення узагальнених алгоритмів для різних типів даних, їхня спеціалізація, і обчислення на етапі компіляції, з використанням шаблонів.

* 1. Огляд особливостей обраного компілятору

Qt Creator — кросплатформна вільна IDE для розробки на С, С ++ і QML [3]. Розроблено для роботи з фреймворком Qt. Включає в себе графічний інтерфейс відладчика і візуальні засоби розробки інтерфейсу. Підтримувані компілятори: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW. Інтегроване середовище розробки, призначене для створення кросплатформних застосунків з використанням бібліотеки Qt. Відповідно підтримується розробка програм мовою C++.

Основне завдання Qt Creator – спростити розробку програми за допомогою фреймворку Qt на різних платформах.

В Qt Creator реалізовано автодоповнення, в тому числі ключових слів, введених в стандарті C++ 11 (починаючи з версії 2.5), підсвічування коду. Також, починаючи з версії 2.4, є можливість завдання стилю вирівнювання, відступів і постановки дужок [4]. Це все спрощує сприйняття коду.

IDE підтримує системи контролю версії, такі як Git, Subversion, Mercurial, CVS, Bazaar, Perforce.

* 1. Огляд можливостей та особливостей обраної системи керування базами даних - SQLite

SQLite — полегшена реляційна система керування базами даних.

Особливістю SQLite є те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремим процесом, з яким взаємодіє застосунок, а надає бібліотеку, з якою програма компілюється і рушій стає складовою частиною програми. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виконується застосунок.

Бібліотека SQLite написана мовою C. Проте розроблено механізм підключення й роботи з БД через цю бібліотеку використовуючи інші мови програмування, зокрема мову C++.

Основними особливостями SQLite є:

* транзакції атомарні, послідовні, ізольовані, і міцні навіть після збоїв системи і збоїв живлення;
* не потребує ані установки, ані адміністрування;
* база даних зберігається в одному крос-платформовому файлі на диску;
* підтримка BLOBів;
* малий розмір коду: менше ніж 350KB повністю налаштований, і менш з опущеними додатковими функціями;
* швидший за популярні рушії клієнт-серверних баз даних для найпоширеніших операцій
* відносно простий, легкий у використанні API
* крос-платформовість;
* кілька процесів або потоків можуть одночасно без жодних проблем читати дані з однієї бази. Запис в базу можна здійснити тільки в тому випадку, коли жодних інших запитів у цей час не обслуговується.
  1. Огляд особливостей застосованого текстового формату збереження даних - JSON

JSON — це текстовий формат збереження даних та обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Розробив і популяризував формат Дуглас Крокфорд.

JSON має переваги XML - дозволяє складні структури в атрибутах і займає менше місця.

JSON будується на двох структурах:

* набір пар назва/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативним масивом;
* впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список, або послідовність.

У JSON використовуються такі їхні форми:

Об'єкт — це послідовність пар назва/значення. Об'єкт починається з символу { і закінчується символом } . Кожне значення слідує за : і пари назва/значення відділяються комами.

Масив — це послідовність значень. Масив починається символом [ і закінчується символом ]. Значення відділяються комами.

Значення може бути рядком в подвійних лапках, або числом, або логічними true чи false, або null, або об'єктом, або масивом. Ці структури можуть бути вкладені одна в одну.

Рядок — це послідовність з нуля або більше символів юнікода, обмежена подвійними лапками, з використанням escape-послідовностей, що починаються зі зворотної косої риски \. Символи представляються простим рядком.

* 1. Огляд класів Qt, що використовуються в роботі

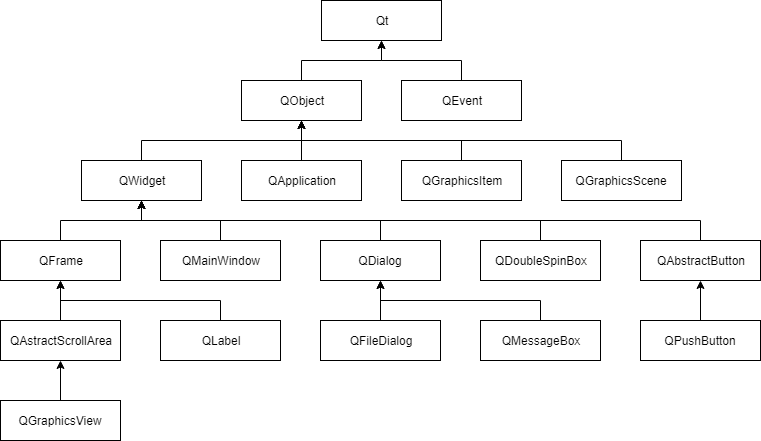


Рисунок 2.1 – Ієрархія основних класів Qt

Qt має власну ієрархію класів. Ієрархія основних класів наведена на рис. 2.1, до яких належать такі класи [5]: Qt, QObject, QEvent, QWidget, QApplication, QGraphicsItem, QGraphicsScene, QFrame, QMainWindow, QDialog, QFileDialog, QMessageBox, QDoubleSpinBox, QAbstractButton, QPushbutton, QAbstractScrollArea, QGraphicsView.

Клас QApplication керує логікою GUI. Він містить основний цикл подій, в якому всі події від віконної системи та інших джерел обробляються і координуються. Він також обробляє ініціалізацію і завершення програми та забезпечує управління сесіями. Крім того, QApplication обробляє більшість загальносистемних і загальних програмних налаштувань.

Клас QObject є базовим для всіх об'єктів Qt і центром об'єктної моделі Qt. Головна особливість в цій моделі – це дуже потужний механізм для зв'язку об'єктів, званий сигналами і слотами. Ви можете з'єднати сигнал зі слотом і роз'єднати. Не має ні конструктора копіювання, ні оператора присвоювання.

Клас QMainWindow надає головне вікно програми. Має власний компонувальник, в який ви можете додавати QToolBar и, QDockWidget, QMenuBar, і QStatusBar. Компонувальник має центральну область, яка може бути зайнята будь-яким віджетом.

Клас QFile надає інтерфейс для читання і запису файлів, і являє собою пристрій вводу/виводу для читання і запису тексту і бінарних файлів і ресурсів. QFile можуть бути використані окремо або, що більш зручно, з QTextStream або QDataStream.

Клас QMessageBox надає цілу серію статичних методів, за допомогою яких можна створювати вікна повідомлень. Ці методи надають підтримку для трьох рівнів: інформаційного, застережливого, критичного. Тип вікна вибирається залежно від обставин. Вікна можуть містити до трьох кнопок.

Клас QWidget є базовим для всіх об'єктів для користувача інтерфейсу. QWidget має безліч функцій-членів, але деякі з них мають досить обмежену функціональність: наприклад, QWidget має властивість керуючу шрифтом, але ніколи не використовує його безпосередньо. Є безліч його підкласів, які забезпечують реальну функціональність, такі як QPushButton, QListWidget, QTabWidget і т.д.

Віджет QPushButton являє собою командну кнопку. Кнопка, або командна кнопка, є найбільш часто використовуваним віджетом в будь-якому графічному інтерфейсі. Натискання кнопки вказує комп'ютеру, що необхідно виконати дію або відповісти на запитання.

Клас QFileDialog забезпечує простий та зручний діалог, для обрання файлів або каталогу користувачем. Вхідним значенням може бути рядок, число або елемент зі списку.

Клас QEvent є базовим класом для всіх класів повідомлень. Об'єкти повідомлень містять параметри повідомлень. Головний цикл обробки повідомлень Qt отримує нативні повідомлення віконної системи з черги повідомлень, конвертує їх в QEvents, і пересилає конвертовані повідомлення в QObject.

Клас QGraphicsItem є базовим класом для всіх графічних елементів в QGraphicsScene. Він надає зручну основу для написання своїх власних елементів. Вона включає визначення геометрії елемента, виявлення зіткнень, реалізацію його відтворення і взаємодія елементів за допомогою покажчиків. QGraphicsItem є частиною каркаса графічного представлення.

Клас QGraphicsScene надає поверхню для управління великою кількістю графічних 2D елементів. Цей клас є контейнером для QGraphicsItem. Він використовується разом з QGraphicsView для демонстрації зображень, таких як лінії, прямокутники, текст або навіть власні елементи на двомірної поверхні. QGraphicsScene також надає функціональність, яка дозволить вам ефективно визначати положення елементів і які елементи видимі всередині довільної області сцени.

Клас QGraphicsView надає віджет для відображення вмісту QGraphicsScene. QGraphicsView відображає вміст QGraphicsScene у прокручуваній області. QGraphicsView є частиною каркаса графічного представлення.

Клас QDoubleSpinBox надає віджет спін-боксу, що приймає дійсні числа. QDoubleSpinBox дозволяє користувачеві вибирати значення, натискаючи кнопки вгору та вниз або натискаючи клавіші "Вгору" та "Вниз" на клавіатурі, щоб збільшити або зменшити поточне значення. Користувач може також ввести значення вручну.

Клас QSettings надає постійні платформонезалежні налаштування програми. Зазвичай користувачі очікують, що додаток буде запам'ятовувати свої настройки (розміри і позиції вікон, параметри і т.д.) між сесіями. Ця інформація часто зберігається в системному реєстрі в Windows і в файлах налаштувань XML в Mac OS X. В Unix-системах, у відсутності стандарту, більшість додатків (включаючи програми KDE) використовують текстові INI-файли. QSettings - це абстракція навколо цих технологій, що дозволяє зберегти і відновити налаштування програми портативним способом. Він також підтримує користувальницькі формати зберігання.

Клас QTranslator забезпечує підтримку інтернаціоналізації для виведення тексту. Об'єкт цього класу містить набір перекладів з вихідної мови на цільову мову. QTranslator надає функції для пошуку перекладів у файлі перекладу. Файли перекладу створюються за допомогою Qt Linguist. Найбільш поширеним способом використання QTranslator є: завантажити файл перекладу, встановити його за допомогою QApplication::installTranslator() і використати через QObject::tr().

* 1. Висновки з розділу

У даному розділі було описано особливості мови програмування, середи розробки, обраного компілятору та системи керування базами даних. Окрім цього у ньому описані класи бібліотеки Qt, що були використані у ході розробки програми: Qt, QObject, QEvent, QWidget, QApplication, QGraphicsItem, QGraphicsScene, QFrame, QMainWindow, QDialog, QFileDialog, QMessageBox, QDoubleSpinBox, QAbstractButton, QPushbutton, QAbstractScrollArea, QGraphicsView.

1. Основні рішення з реалізації  
   компонентів системи

В даному розділі розглянуті основні рішення розроблені класи, основні розроблені алгоритми, рішення щодо розробки інтерфейсу користувача, рішення щодо збереження даних та використання бази даних.

* 1. Основні рішення щодо уявлення даних системи

В ході роботи над програмою курсового проекту було розроблено класи:

* AboutWindow – форма з інформацією про програмний продукт
* DataBase – клас для роботи з SQLite
* ItemEditWindow – форма редагування властивостей об’єкта
* FileOpen – клас з реалізацією відкриття файлів
* FileSave – клас з реалізацією збереження файлів
* FileWay – базовий клас для FileOpen та FileSave
* FormCreate – форма створення нового проекту
* GraphicItem – клас графічного об'єкта
* ItemByColor – клас графічного об’єкта представленого кольором
* ItemByPicture – клас графічного об’єкта представленого зображенням
* JsonData – клас для роботи з файлами \*.json
* MainWindow – головна форма програми
* MdiChild – форма з графічною сценою проекту
* Points – клас з визначеними точками графічних об’єктів

Класову структуру програми зображено на рисунку 3.1.

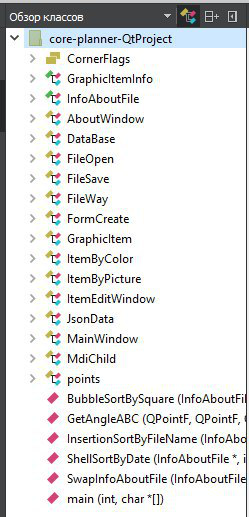


Рисунок 3.1 – Класова структура програми

Клас AboutWindow містить у собі опис програмного забезпечення та контакти авторів. Дані та методи класу наведені в табл. 3.1.

Таблиця 3.1 – Дані та методи класу AboutWindow

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| Ui::AboutWindow \*ui; | Покажчик на графічний інтерфейс |
| QPixmap backgroundImage; | Зображення фону |
| void setupOtherUi(); | Метод генерації графічного інтерфейсу |
| ***private slots:*** | |
| void on\_pushButtonClose\_clicked(); | Обробка закриття |
| void on\_labelSystemNick\_linkActivated  (const QString &link); | Обробка натискання на посилання |
| void on\_labelx8357238\_linkActivated  (const QString &link); | Обробка натискання на посилання |
| void on\_labelvaleriyzp\_linkActivated  (const QString &link); | Обробка натискання на посилання |
| ***public:*** | |
| AboutWindow(QWidget \*parent = 0); | Конструктор |
| ~AboutWindow(); | Деструктор |

Клас DataBase містить у собі засоби для роботи з базою даних через систему керування базами даних SQLite. Дані та методи класу наведені в табл. 3.2.

Таблиця 3.2 – Дані та методи класу DataBase

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| QSqlDatabase dataBase; |  |
| int maxItems; | Ліміт записів |
| int itemsLeft; | Кількість виконаних запитів |
| int itemsLeftDataBase(); | Кількість виконаних запитів |
| bool createTable(); | Створення таблиці БД |
| bool openDataBase(); | Відкриття БД |
| bool restoreDataBase(); | Відновлення БД |
| bool removeRecord  (const QString &fWay); | Видалення запису |
| ***public:*** | |
| DataBase(); | Конструктор |
| ~DataBase(); | Деструктор |
| void connectToDataBase(); | З’єднання з БД |
| bool inserIntoTable  (const QString &fName,  const QString &fData,  const QString &fWay,  const double &fSquare); | Вставка в таблицю |
| void closeDataBase(); | Закриття БД |
| void WriteTableToArray  (InfoAboutFile \*Array,  int MaximumCount,  int \*CurrentCount); | Запис таблиці у масив |

Клас ItemEditWindow містить у собі реалізацію елементів керування для зміни властивостей об’єктів сцени проекту. Дані та методи класу наведені в табл. 3.3.

Таблиця 3.3 – Дані та методи класу ItemEditWindow

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| Ui::ItemEditWindow \*ui; | Покажчик на графічний інтерфейс |
| ***private slots:*** | |
| void on\_pushButtonApply\_clicked(); | Обробка натиску на кнопку |
| ***public:*** | |
| ItemEditWindow(); | Конструктор |

Продовження таблиці 3.3

|  |  |
| --- | --- |
| ItemEditWindow  (MainWindow \*Main,  MdiChild \*Parent,  QWidget \*parent = 0); | Конструктор |
| ~ItemEditWindow(); | Деструктор |
| MdiChild \*FormWithScene; | Покажчик на дочірну форму |
| MainWindow \*MainWin; | Покажчик на базову форму |

Клас FileOpen містить у собі реалізацію функцій для відкриття проекту. Дані та методи класу наведені в табл. 3.4.

Таблиця 3.4 – Дані та методи класу FileOpen

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| MainWindow \*Main; | Покажчик на базову форму |
| ***public:*** | |
| FileOpen(MainWindow \*Parent); | Конструктор |
| ~FileOpen(); | Деструктор |
| void SetWay(); | Встановлення шляху до файлу |
| void OpenFile(); | Відкриття файлу |
| void OpenFileByWay  (QString Way, QString FileName); | Відкриття файлу за шляхом |

Клас FileSave містить у собі реалізацію функцій для збереження проекту. Дані та методи класу наведені в табл. 3.5.

Таблиця 3.5 – Дані та методи класу FileSave

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***public:*** | |
| FileSave(); | Конструктор |
| ~FileSave(); | Деструктор |
| MdiChild \*ParenWithScene; | Покажчик на дочірню форму |
| QString HistoryFileWay; | Шлях до файлу з адресами проектів |
| void SetWay(); | Встановлення шляху |

Продовження таблиці 3.5

|  |  |
| --- | --- |
| void Save(MainWindow \*Main); | Зберігання файлу |
| void SaveAs(MainWindow \*Main); | Зберігання файлу за новими форматами |
| void SaveToHistory(); | Зберігання до файлу з адресами проектів |
| void InitHistory(); | Ініціалізація файлу з адресами проектів |
| void LoadFromHistory(int Version); | Завантаження інформації з фалу з адресами |
| void DeleteHighVersions(int Version); | Видалення старих записів |

Клас FileWay містить у собі опис адреси та назви файлу. Дані та методи класу наведені в табл. 3.6.

Таблиця 3.6 – Дані та методи класу FileWay

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***public:*** | |
| FileWay(); | Конструктор |
| FileWay(QString WayToFile); | Конструктор |
| ~FileWay(); | Деструктор |
| QString Way; | Шлях до файлу |
| QString FileName; | Назва файлу |
| QString GetFileNameFromWay  (QString AdressFile); | Отримання назви файлу з його шляху |
| virtual void SetWay() = 0; | Встановлення шляху |

Клас FormCreate містить у собі реалізацію інтерфейсу для створення нових проектів чи навігації по нещодавнім. Дані та методи класу наведені в табл. 3.7.

Таблиця 3.7 – Дані та методи класу FormCreate

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| InfoAboutFile \*Files; | Інформація про файл |
| int CountOfFiles; | Кількість файлів |
| Ui::FormCreate \*ui; | Показник на графічний інтерфейс |
| int WayToOpen; |  |

Продовження таблиці 3.7

|  |  |
| --- | --- |
| ***private slots:*** | |
| void SetFilesButtons(); | Встановлення тексту кнопок |
| void SetButtonsTextAlign(); | Встановлення вирівнювання тексту |
| void on\_pushButtonCreateNew\_clicked(); | Обробка натискання на кнопку |
| void on\_pushButtonProjectName\_clicked(); | Обробка натискання на кнопку |
| void on\_pushButtonProjectTime\_clicked(); | Обробка натискання на кнопку |
| void on\_pushButtonProjectSquare\_clicked(); | Обробка натискання на кнопку |
| void on\_pushButton\_1\_clicked(); | Обробка натискання на кнопку |
| void on\_pushButton\_2\_clicked(); | Обробка натискання на кнопку |
| void on\_pushButton\_3\_clicked(); | Обробка натискання на кнопку |
| void on\_pushButton\_4\_clicked(); | Обробка натискання на кнопку |
| void on\_pushButton\_5\_clicked(); | Обробка натискання на кнопку |
| void on\_pushButtonOpen\_clicked(); | Обробка натискання на кнопку |
| ***public:*** | |
| FormCreate(); | Конструктор |
| ~FormCreate(); | Деструктор |
| MainWindow \*MainWin; | Покажчик на базову форму |

Клас GraphicItem містить у собі реалізацію поводження графічного об’єкта на сцені проекту. Дані та методи класу наведені в табл. 3.8.

Таблиця 3.8 – Дані та методи класу GraphicItem

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***public:*** | |
| GraphicItem(); | Конструктор |
| GraphicItem  (double setWidth, double setHeight,  MdiChild \*MdiChildParent,  QObject \*parent = 0); | Конструктор |
| ~GraphicItem(); | Деструктор |
| points MidlePoints; | Координати середніх точок |
| double Width; | Ширина |
| double Height; | Висота |
| int FlagCorner; | Флаг куту |
| bool IsEditNow; | Флаг можливості редагування |
| bool IsResizeModNow; | Флаг зміни розміру |
| int SizeOfPoint; | Розмір точки |
| double MinimumWidth; | Мінімальна ширина |
| double MinimumHeight; | Мінімальна висота |

Продовження таблиці 3.8

|  |  |
| --- | --- |
| double CurrentAngleOfRotation; | Поточний кут нахилу |
| QPointF ClickedPointOnScene; | Координата натиснення на сцену |
| MdiChild \*ParenWithScene; | Покажчик на дочірню форму |
| bool IsSceneEdited; | Флаг редагування сцени |
| virtual QString GetType() = 0; | Отримання типу об’єкту |
| double GetHeight(); | Отримання висоти об’єкту |
| double GetWidth(); | Отримання ширини об’єкту |
| double GetRotationAngle(); | Отримання куту нахилу об’єкту |
| void SetRotationAngle  (double RotationAngleToSet); | Встановлення куту нахилу |
| void SetHeight(double HeightToSet); | Встановлення висоти |
| void SetWidth(double WidthToSet); | Встановлення ширини |
| void SetUnActive(); | Встановлення неактивності |
| virtual GraphicItemInfo GetItemInfo() = 0; | Отримання інформації |
| QPointF m\_shiftMouseCoords; | Координати миші |
| QRectF boundingRect() const; | Прямокутник |
| void paint(QPainter \*painter,  const QStyleOptionGraphicsItem \*option,  QWidget \*widget) = 0; | Метод малювання |
| void mouseMoveEvent  (QGraphicsSceneMouseEvent \*event); | Обробка руху миші |
| void mousePressEvent  (QGraphicsSceneMouseEvent \*event); | Обробка натискання миші |
| void mouseReleaseEvent  (QGraphicsSceneMouseEvent \*event); | Обробка відпускання клавіші миші |
| void mouseDoubleClickEvent  (QGraphicsSceneMouseEvent \*event); | Обробка подвійного кліку миші |
| void hoverMoveEvent  (QGraphicsSceneHoverEvent \*event); | Обробка наведення миші |
| void ResetCursor  (double CursorX, double CursorY,  bool MouseMove); | Повернути курсор |

Клас ItemByColor містить у собі засоби відображення об'єкту представленого кольоровою палітрою. Дані та методи класу наведені в табл. 3.9.

Таблиця 3.9 – Дані та методи класу ItemByColor

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| QColor ColorOfItem; | Колір |
| void paint(QPainter \*painter,  const QStyleOptionGraphicsItem \*option,  QWidget \*widget); | Метод малювання |
| ***public:*** | |
| ItemByColor(); | Конструктор |
| ItemByColor(double setWidth,  double setHeight, QColor setColor,  MdiChild \*MdiChildParent,  QObject \*parent = 0); | Конструктор |
| ~ItemByColor(); | Деструктор |
| QString GetType(); | Отримання типу |
| GraphicItemInfo GetItemInfo(); | Отримання інформації графіки |

Клас ItemByPicture містить у собі засоби відображення об'єкту представленого зображенням. Дані та методи класу наведені в табл. 3.10.

Таблиця 3.10 – Дані та методи класу ItemByPicture

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| QString NameOfPicture; | Назва зображення |
| QImage PictureOfItem; | Зображення |
| void paint(QPainter \*painter,  const QStyleOptionGraphicsItem \*option,  QWidget \*widget); | Метод малювання |
| ***public:*** | |
| ItemByPicture(); | Конструктор |
| ItemByPicture(double setWidth,  double setHeight, QString PictureName,  MdiChild \*MdiChildParent,  QObject \*parent = 0); | Конструктор |
| ~ItemByPicture(); | Деструктор |
| QString GetType(); | Отримання типу |
| GraphicItemInfo GetItemInfo(); | Отримання інформації |

Клас JsonData містить у собі засоби для роботи з файлом у форматі \*.json, а саме: зчитування, видалення, змінення. Дані та методи класу наведені в табл. 3.11.

Таблиця 3.11 – Дані та методи класу JsonData

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| QJsonDocument doc; | Зміст з файлу |
| QJsonParseError docError; | Повідомлення про помилку |
| ***public:*** | |
| JsonData(); | Конструктор |
| ~JsonData(); | Деструктор |
| void inFromFile  (const QString&fWay, MdiChild \*Parent); | Запис до сцени з файлу об’єктів |
| void outToFile  (const QString& fWay, MdiChild \*Parent); | Запис проекту у файл |
| void outHistoryToFile  (const QString& fWay, MdiChild \*Parent); | Запис історії у файл |
| void outInitHistoryFile  (const QString& fWay, MdiChild \*Parent); | Ініціалізація файлу історії |
| void InitializeSceneFromFile  (MdiChild \*Parent,QJsonArray &docArr); | Ініціалізація сцени з файлу |
| void InitScenFromFile  (const QString&fWay, MdiChild \*Parent,  int Version); | Завантаження сцени з файлу |
| void DeleteHighVersions  (const QString&fWay, int Version); | Видалення версій |

Клас MainWindow містить у собі реалізацію відображення панелі інструментів для планування приміщень та області відкритих проектів. Дані та методи класу наведені в табл. 3.12.

Таблиця 3.12 – Дані та методи класу MainWindow

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| Ui::MainWindow \*ui; | Покажчик на елементи інтерфейсу |
| int openProjects = 0; | Кількість відкритих проектів |
| QStandardItemModel \*modelBuildTools; | Модель списку інструментів будування |

Продовження таблиці 3.12

|  |  |
| --- | --- |
| QSortFilterProxyModel \*proxyModelBuildTools; | Сортування |
| QStandardItemModel \*modelFurnitureTools; | Модель списку інструментів меблі |
| QSortFilterProxyModel \*proxyModelFurnitureTools; | Сортування |
| QStandardItemModel \*modelDecorTools; | Модель списку інструментів декору |
| QSortFilterProxyModel \*proxyModelDecorTools; | Сортування |
| int FilesCount = 0; | Кількість файлів |
| void setupOtherUi(); | Встановлення інших об’єктів |
| void setupBuildTools(); | Встановлення інструментів |
| void setupFurnitureTools(); | Встановлення фурнітури |
| void setupDecorTools(); | Встановлення декору |
| QTranslator\* translator; | Об’єкт перекладача |
| void changeTranslator  (QString postfix); | Метод, що підключає перекладач з новим словником |
| void changeEvent(QEvent \*event); | Метод для обробки подій |
| ***private slots:*** | |
| void on\_actionNew\_triggered(); | Обробка створення нового проекту |
| void on\_actionOpen\_triggered(); | Обробка відкриття проекту |
| void on\_actionSave\_triggered(); | Обробка збереження проекту |
| void on\_actionSaveAs\_triggered(); | Обробка збереження проекту за варіантами |
| void on\_actionClose\_triggered(); | Обробка закриття проекту |
| void on\_actionAbout\_triggered(); | Обробка виклику форми про програму |
| void on\_listViewBuildTools\_clicked  (const QModelIndex &index); | Обробка активації списку інструментів будування |
| void on\_listViewFurnitureTools\_clicked  (const QModelIndex &index); | Обробка активації списку інструментів меблі |
| void on\_listViewDecorTools\_clicked  (const QModelIndex &index); | Обробка активації списку інструментів декору |
| void on\_actionUk\_triggered(); | Обробка обрання української локалізації |
| void on\_actionEn\_triggered(); | Обробка обрання англійської локалізації |
| void on\_actionRu\_triggered(); | Обробка обрання російської локалізації |
| void on\_actionUndo\_triggered(); | Обробка відміни |

Продовження таблиці 3.12

|  |  |
| --- | --- |
| void on\_actionRedo\_triggered(); | Обробка повтору |
| void on\_actionDelete\_triggered(); | Обробка видалення |
| void on\_actionClear\_triggered(); | Обробка очистки |
| void on\_actionZoomIn\_triggered(); | Обробка приближення |
| void on\_actionZoomOut\_triggered(); | Обробка віддалення |
| void on\_actionSetItemSize\_triggered(); | Обробка встановлення розміру об’єкту |
| ***public:*** | |
| explicit MainWindow  (QWidget \*parent = 0); | Конструктор |
| ~MainWindow(); | Деструктор |
| int countOpen(); | Кількість відкритих |
| MdiChild \*activeMdiChild() const; | Покажчик на дочірню форму |
| DataBase database; | База даних SQLite |
| int uniqueProjectID = 1; | Унікальний ідентифікатор |
| void loadProject(QWidget \*widget); | Завантаження проекту |

Клас MdiChild містить у собі реалізацію інтерфейсу з графічною сценою відкритого проекту. Дані та методи класу наведені в табл. 3.13.

Таблиця 3.13 – Дані та методи класу MdiChild

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| Ui::MdiChild \*ui; | Покажчик на графічний інтерфейс |
| int Width, Height; | Ширина та висота |
| QVector<QGraphicsLineItem\*> HorizontalLines; | Масив горизонтальних ліній |
| QVector<QGraphicsLineItem\*> VerticalLines; | Масив вертикальних ліній |
| void DrawHorizontalLines(); | Малювання горизонтальних ліній |
| void DrawVerticalLines(); | Малювання вертикальних ліній |
| ***public:*** | |
| explicit MdiChild(QWidget \*parent); | Конструктор |
| ~MdiChild(); | Деструктор |
| GraphicItem \*LastActive = 0; | Покажчик на останній активний об’єкт |
| QGraphicsScene \*scene; | Покажчик на графічну сцену |
| void addItem(QString Name,  double setWidth, double setHeight); | Метод додавання об’єкту за ім’ям |

Продовження таблиці 3.13

|  |  |
| --- | --- |
| void addItem(QColor ItemColor,  double setWidth, double setHeight); | Метод додавання об’єкту за кольором |
| void UpdateScene(); | Оновлення сцени |
| void SaveAsImage  (QString NameOfFile); | Метод збереження за варіантами |
| void setScaleToView  (double X, double Y); | Метод встановлення масштабу |
| void DrawGrid(); | Метод малювання сітки |
| void DeleteGrid(); | Метод видалення сітки |
| void SetSceneParametrs  (int setWidth, int setHeight); | Метод встановлення параметрів сцени |
| FileSave SaveFileInfo; | Збереження інформації в файл |
| int VersionOfScene; | Версія сцени |
| int MaximumVersionOfScene; | Ліміт сцен |
| bool AfterUndo; | Флаг здійснення відміни |

Клас Points містить у собі значення і функції задавання точок з об'єкту. Дані та методи класу наведені в табл. 3.14.

Таблиця 3.14 – Дані та методи класу Points

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| 1 | 2 |
| ***private:*** | |
| QPointF TopMinimumCenter; | Верхня точка центру об’єкту за мінімальної висоти |
| QPointF LeftMinimumCenter; | Ліва точка центру об’єкту за мінімальної висоти |
| QPointF RightMinimumCenter; | Права точка центру об’єкту за мінімальної висоти |
| QPointF BottomMinimumCenter; | Нижня точка центру об’єкту за мінімальної висоти |
| ***public:*** | |
| Points(); | Конструктор |
| ~Points(); | Деструктор |
| void SetTopMinimumCenter  (QPointF point); | Встановлення верхньої точки центру об’єкту за мінімальної висоти |
| void SetLeftMinimumCenter  (QPointF point); | Встановлення лівої точки центру об’єкту за мінімальної висоти |
| void SetRightMinimumCenter  (QPointF point); | Встановлення правої точки центру об’єкту за мінімальної висоти |

Продовження таблиці 3.14

|  |  |
| --- | --- |
| void SetBottomMinimumCenter  (QPointF point); | Встановлення нижньої точки центру об’єкту за мінімальної висоти |
| QPointF GetTopMinimumCenter(); | Отримання верхньої точки центру об’єкту за мінімальної висоти |
| QPointF GetLeftMinimumCenter(); | Отримання лівої точки центру об’єкту за мінімальної висоти |
| QPointF GetRightMinimumCenter(); | Отримання правої точки центру об’єкту за мінімальної висоти |
| QPointF GetBottomMinimumCenter(); | Отримання нижньої точки центру об’єкту за мінімальної висоти |

* 1. Основні розроблені алгоритми

Під час розробки було реалізовано наступні алгоритми:

* файлове зчитування та запис у файл;
* відображення елементів візуальної форми програми;
* Сортування даних методом Шелла;
* Сортування даних бульбашковим методом;
* Сортування даних методом вставок;
* Розрахунок кута за трьома точками;
* Визначення назви файлу з повного шляху;
* Розрахунок зміни розмірів і позицій об'єкту;
* Розрахунок зміни зовнішнього виду курсору.

Створення нових та редагування вже збережених проектів дозволяє комфортно працювати над розробкою плану приміщення з можливістю зберігти та відкласти працю на потім.

* 1. Особливості реалізації системи

Особливістю реалізації системи є дуже зрозумілий дружній до користувача інтерфейс. Будь-яка людина зможе просто користуватися розробленим додатком, вибирати потрібні дані та створювати свої замітки.

На головній формі MainWindow розміщені наступні елементи, які зображені на рис. 3.2.

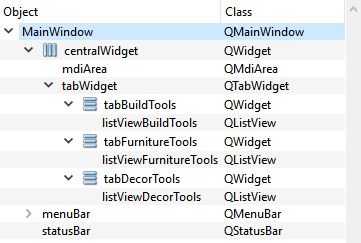


Рисунок 3.2 – Елементи головної форми MainWindow

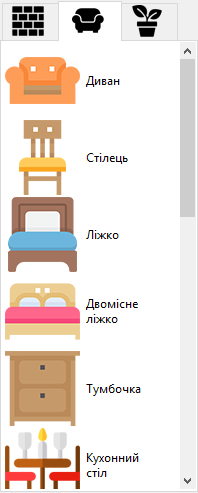


Рисунок 3.2 – Список інструментів

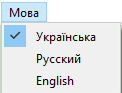
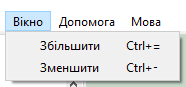
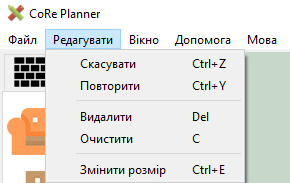
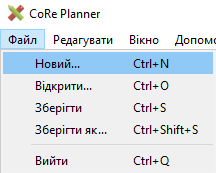


Рисунок 3.3 – Пункти головного меню

Область для дочірніх форм типу QMdiArea призначена для відображення розшифрованого змісту файлів проекту у окремих вікнах.

Клас QListView відображає списком інструменти планувальника у вигляді: піктограма, назва розміщуваного об’єкта. Він призначений для розміщення нових об’єктів на графічній сцені проекту.

Клас QTabWidget зберігає згруповані за типом списки інструментів планувальника.

Клас QMenuBar зберігає усі необхідні об’єкти класу QAction, які призначені для:

* виклику форм створення нових або відкриття для редагування попередньо розроблених та збережених проектів;
* відкату змін та інших маніпуляцій з активним проектом;
* виклику довідки та інформації про програму;
* зміни мови інтерфейсу програми.

На формі MdiChild розміщені наступні елементи, які зображені на рис. 3.4.

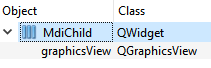


Рисунок 3.4 – Елементи форми MdiChild

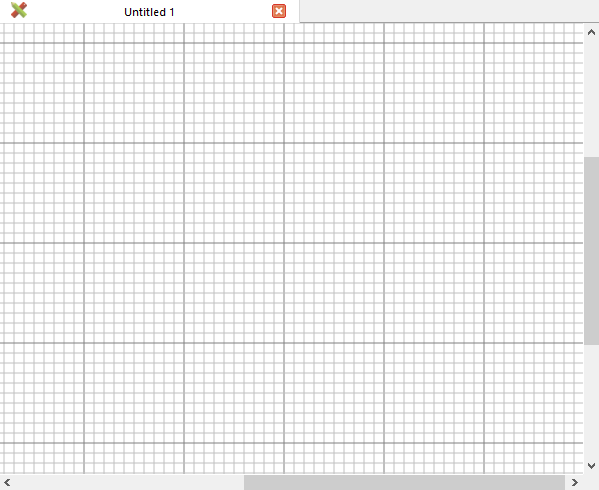


Рисунок 3.5 – Форма MdiChild

Об’єкт graphicsView класу QGraphicsView призначений для відображення сцени проекту та взаємодії з графічними об’єктами розміщеними на об’єкті класу QGraphicsScene.

На формі ItemEditWindow розміщені наступні елементи, які зображені на рис. 3.6.

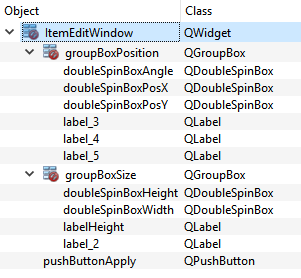


Рисунок 3.6 – Елементи форми ItemEditWindow

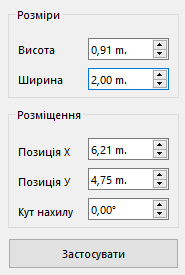


Рисунок 3.7 – Елементи форми ItemEditWindow

Клас QGroupBox використаний для логічного групування елементів керування.

На формі FormCreate розміщені наступні елементи, які зображені на рис. 3.8.

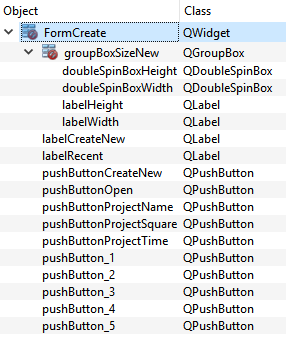


Рисунок 3.8 – Елементи форми FormCreate

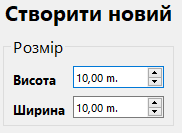


Рисунок 3.9 – Частина форми FormCreate

Клас QPushButton використаний для визначення рішення щодо створення нового проекту чи відкриття нещодавнього та для відображення ім’я, дати відкриття і площі обраного нещодавнього проекту.

Клас QLabel створює підписи для допомоги орієнтування користувача.

Клас QDoubleSpinBox призначений для введення ширини та висоти нового проекту.

На формі AboutWindow розміщені наступні елементи, які зображені на рис. 3.10.

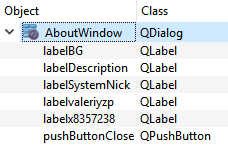


Рисунок 3.10 – Елементи форми AboutWindow

Клас QLabel слугує для відображення опису програмного продукту та для надання клікабельних посилань на аккаунти Telegram авторів.

Клас QPushButton використаний для закривання форми.

* 1. Результати тестування системи

У ході тестування програми усі помилки було виправлено. На даному етапі програма функціонує правильно та злагоджено.

На рис. 3.11 зображено повідомлення, яке виникає при неправильному відкритті файлу.

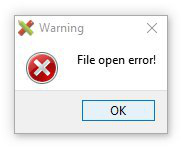


Рисунок 3.11 – Помилка відкриття файлу

* 1. Основні рішення щодо збереження та відтворення інформації.

Для роботи з даними були використана бібліотека QtSql. База даних застосована для збереження історії збережених файлів.

SQLite було обрано через низку переваг, а саме: база даних зберігається в одному крос-платформовому файлі на диску, наявна підтримка BLOBів, швидший за популярні рушії клієнт-серверних баз даних для найпоширеніших операцій.

Наявна можливість збереження проекту в файл формату:

- JSON з можливістю подальшого відтворення у програмі та редагування;

- JPG, PNG, BMP.

* + 1. Запити SQL, що були використані

У під час роботи проекту застосовані такі запити:

- “ CREATE TABLE ім’я\_таблиці (перелік полів та їх типу) ” – запит на створення таблиці в БД;

- “ SELECT \* FROM ім’я\_таблиці ” - запит на отримання значень усіх полів з таблиці;

- “ INSERT INTO ім’я\_таблиці VALUES (значення \_1, значення \_2, …, значення \_N) ” - запит на вставку вказаних значень у поля таблиці;

- “ DELETE FROM ім’я\_таблиці WHERE значення\_поля = параметр ” - запит на видалення поля таблиці, у якого визначене значення певного поля дорівнює параметру.

* + 1. Основні рішення використані при взаємодії з БД

Класс DataBase створен для поліпшення роботи з базою даних . У цьому класі реалізовано метод створення, завантаження бази даних, отримання усіх значень з таблиці, видалення певного значення за параметром.

З’єднання з базою даних забезпечує метод DataBase::createTable(), де наявна перевірка чи була створена у файлі таблиця, чи ні: якщо так – файл відкривається (DataBase::openDataBase()), якщо ні – ініціалізується (DataBase::createTable()). Структуру таблиці зазначено на рисунку 3.15.

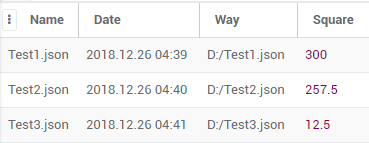


Рисунок 3.15 – структура таблиці «FileHistory»

Таблиця «FileHistory» зберігає дані про збережені проекти. Вона має наступні поля:

- Name – ім’я збереженого проекту;

- Date – поле, яке містить дату зберігання проекту;

- Way – поле, яке містить шлях до збереженого проекту.

- Square – поле, яке зберігає площадь приміщення.

БД використовується при відкритті вікна FormCreate(створення нового фалйу).

При збереженні будь-якого проекту у форматі JSON викликається функція DataBase::inserIntoTable(const QString&fName,const QString&fData,const QString&fWay,const double&fSquare), яка записує до БД новий член.

* + 1. Основні рішення щодо збереження даних в JSON файлах

Класс JsonData створен для поліпшення роботи з базою даних . У цьому класі реалізовано метод створення, завантаження бази даних, отримання усіх значень з таблиці, видалення певного значення за параметром.

Для взаємодії з файлами формату JSON використовувалися такі методи: JsonData::inFromFile() та JsonData::outFromFile() – для збереження проекту та відкриття проекту; JsonData::outHistoryToFile() та JsonData::InitScenFromFile() – для збереження змін в файл історії змін, та для завантаження з файлу історії змін.

* 1. Висновки з розділу

В ході роботи над програмою курсового проекту було розроблено класи: AboutWindow, DataBase, ItemEditWindow, FileOpen, FileSave, FileWay, FormCreate, GraphicItem, ItemByColor, ItemByPicture, JsonData, MainWindow, MdiChild, Points.

При виконанні курсової роботи був використаний механізм сигналів та слотів. Таким чином, було розроблено декілька алгоритмів (файлове зчитування та запис у файл, відображення елементів візуальної форми програми) які відповідають тим чи іншим функціональним частинам програми, що допомагає створити максимально зручний та зрозумілий користувачеві інтерфейс, а також організувати досить швидку реалізацію програми.

Було реалізовано взаємодію з базою даних, та з форматом JSON.

При розробці інтерфейсу програми були використані такі візуальні компоненти Qt: QPushButton, QDoubleSpinBox, QLineEdit, QTextEdit, QLabel, QGraphicsScene, QMdiArea, QListView, QTabWidget, QMenuBar та QStatusBar.

У ході тестування системи усі помилки було виправлено.

1. Керівництво програміста

В даному розділі розглянуті призначення, умови застосування, характеристика програми, звертання до програми, початкові та вихідні дані та представлені повідомлення.

* 1. Призначення та умови застосування програми

Призначення програми – проектування плану приміщень будівлі.

Умовою застосування для коректної роботи є використання останніх версій П3, що необхідно для роботи програми, наявність дисплею, маніпулятору миші, клавіатури.

* 1. Характеристики програми

Програма виконана за допомогою мови програмування високого рівня С++ в середовищі розробки Qt Creator 5.3. Проект (рис. 4.1) містить класи, їх реалізацію, файли ресурсів та проекту, файли формату \*.sql, де зберігаються дані та файли формату \*.ts, де зберігаються тексти перекладів.

Система розподілена на класи з даними і функціями, які використовуються цими даними та виконують свої задачі для роботи системи. Загалом система поділяється на наступні модулі: «Програмна область», «Робоча область», «Редагування об’єкту», «Параметризація проекту», «Зворотній зв’язок».

Основою всього проекту є .pro файл. Основні особливості .pro файлу:

* QT += core gui sql – зазначення основних модулів Qt, що використовуються програмою;
* HEADERS += \*.h - усі .h файли проекту. Ці файли визначають основні використані бібліотеки та класи програми і створені для роздроблення програми на модулі та окремі файли. До них належить 20 файлів програми;
* SOURCES += \*.cpp - усі .cpp файли проекту. Ці файли називають вихідними файлами. До них належить 21 .cpp файл;
* FORMS += \*.ui - усі форми програми. Форми відповідають за візуальний інтерфейс, з яким взаємодіє користувач. До них належать 5 форм;
* RESOURCES += \*.qrc - файли ресурсів, використані програмою. В випадку розробленої програми до ресурсів належить іконки та .png зображення, що розбиті на різні папки для зручнішого доступу та кращого інтуїтивного сприйняття;
* QMAKE\_CXXFLAGS += -std=c++0x – визначення стандарту мови, на якому розроблено проект;
* TRANSLATIONS += \*.ts – файли перекладу, використані програмою. Програма підтримує відображення інтерфейсу на 3 мовах.

Проект включає в себе такі файли:

* aboutwindow.h - файл, що містить оголошення класу AboutWindow, який призначений для відображення на екрані інформаційного вікна з даними про програму та її авторів;
* database.h - файл, що містить оголошення класу DataBase, який призначений для роботи з SQLite, а саме: для збереження, відтворення та зміни істрії файлів проекту ;
* edititemwindow.h - файл, що містить оголошення класу EditItemWindow, який призначений для відображення на екрані вікна редагування даних обраного об’єкту;
* fileopen.h - файл, що містить оголошення класу FileOpen, який призначений для відкриття вже існуючого проекту у програмі;
* filesave.h - файл, що містить оголошення класу FileSave, який призначений для збереження проекту у вигляді редагуємого файлу або зображення;
* fileway.h - файл, що містить оголошення класу FileWay, який призначений для зберігання та редагування даних про знаходження файлу у системі;
* formcreate.h - файл, що містить оголошення класу FormCreate, який призначений для відображення на екрані вікна з даними для створення нового проекту або відкриття нещодавного;
* graphicitem.h - файл, що містить оголошення класу GraphicItem, який призначений для обробки дії користувача при роботі з об’єктами;
* itembycolor.h - файл, що містить оголошення класу ItemByColor, який призначений для роботи з прямокутним об’єктом даного кольору;
* itembypicture.h - файл, що містить оголошення класу ItemByPicture, який призначений для призначений для роботи з прямокутним об’єктом, який містить зображення;
* jsondata.h - файл, що містить оголошення класу JsonData, який призначений для роботи з JSON форматом, а саме: для збереження та відтворення проектів;
* mainwindow.h - файл, що містить оголошення класу MainWindow, який призначений для відображення на екрані головного вікна програми;
* mdichild.h - файл, що містить оголошення класу MdiChild, який призначений для відображення на екрані проекту;
* points.h - файл, що містить оголошення класу Points, який призначений для зберігання та редагування даних про мінімальні позиції центрів об’єкту, що редагується;
* aboutwindow.cpp - файл для реалізації класу AboutWindow;
* database.cpp - файл для реалізації класу DataBase;
* itemeditwindow.cpp - файл для реалізації класу ItemEditWindow;
* fileopen.cpp - файл для реалізації класу FileOpen;
* filesave.cpp - файл для реалізації класу FileSave;
* fileway.cpp - файл для реалізації класу FileWay;
* formcreate.cpp - файл для реалізації класу FormCreate;
* graphicitem.cpp - файл для реалізації класу GraphicItem;
* itembycolor.cpp - файл для реалізації класу ItemByColor;
* itembypicture.cpp - файл для реалізації класу ItemByPicture;
* jsondata.cpp - файл для реалізації класу JsonData;
* mainwindow.cpp - файл для реалізації класу MainWindow;
* mdichild.cpp - файл для реалізації класу MdiChild;
* points.cpp - файл для реалізації класу Points;
* aboutwindow.ui - файл xml формату, що містить графічний опис класу AboutWindow;
* formcreate.ui - файл xml формату, що містить графічний опис класу FormCreate;
* itemeditwindow.ui - файл xml формату, що містить графічний опис класу ItemEditWindow;
* mainwindow.ui - файл xml формату, що містить графічний опис класу MainWindow;
* mdichild.ui - файл xml формату, що містить графічний опис класу MdiChild;
* core-cursors.qrc: ResizeD1.png, ResizeD2.png, ResizeH.png, ResizeV.png, Rotation.png – набір різновидів курсору;
* core-images.qrc: bath.png, bed-double.png, bed-single.png, bedside.png, car.png, chair.png, door.png, door-garage.png, oven.png, plant.png, sofa.png, table-dining.png, table-glass.png, table-rectangle.png, table-wooden.png, table-work.png, toilet.png, window.png – набір іконок елементів меню редагувальника; cube.ico, icon.ico – набір іконок програми; about.png, splash.png – набір фону форм програми;
* scene-images.qrc: bath.png, bed-double.png, bed-single.png, bedside.png, car.png, chair.png, door.png, door-garage.png, oven.png, plant.png, sofa.png, table-dining.png, table-glass.png, table-rectangle.png, table-wooden.png, table-work.png, toilet.png, window.png – набір зображень елементів для відображення у проекті редагувальника;
* Core\_Planner\_en.ts - файл xml формату, що містить переклад інтерфейсу на англійську мову;
* Core\_Planner\_ru.ts - файл xml формату, що містить переклад інтерфейсу на російську мову;

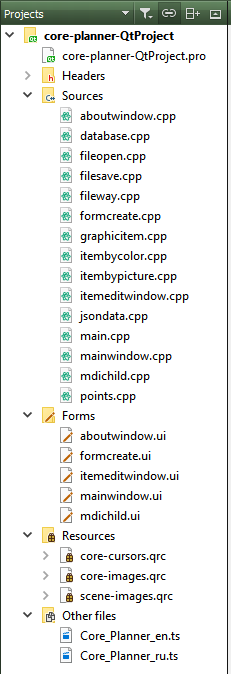


Рисунок 4.1 – Структура проекту

* 1. Звертання до програми

Для звернення, по-перше, необхідно розархівувати теку з програмою та розташувати її у каталог, до якого має доступ оператор. По-друге, для запуску програми у папці core-planner-QtProject (рис.4.2) треба запустити файл core-planner-QtProject.pro, дочекатися запуску проекту у QtCreator. По-третє, натиснути кнопку «Збірка» і програма готова до роботи. Програма стабільно працює при використанні Desktop Qt 5.3 MinGW 32bit Kit, приклад налаштування збірки наведений на рис. 4.3.

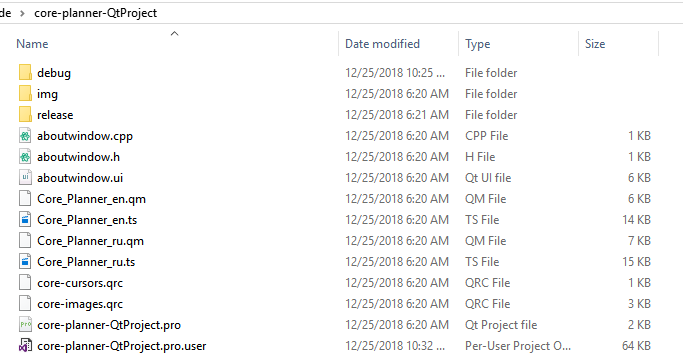


Рисунок 4.2 – Каталог з файлами програми

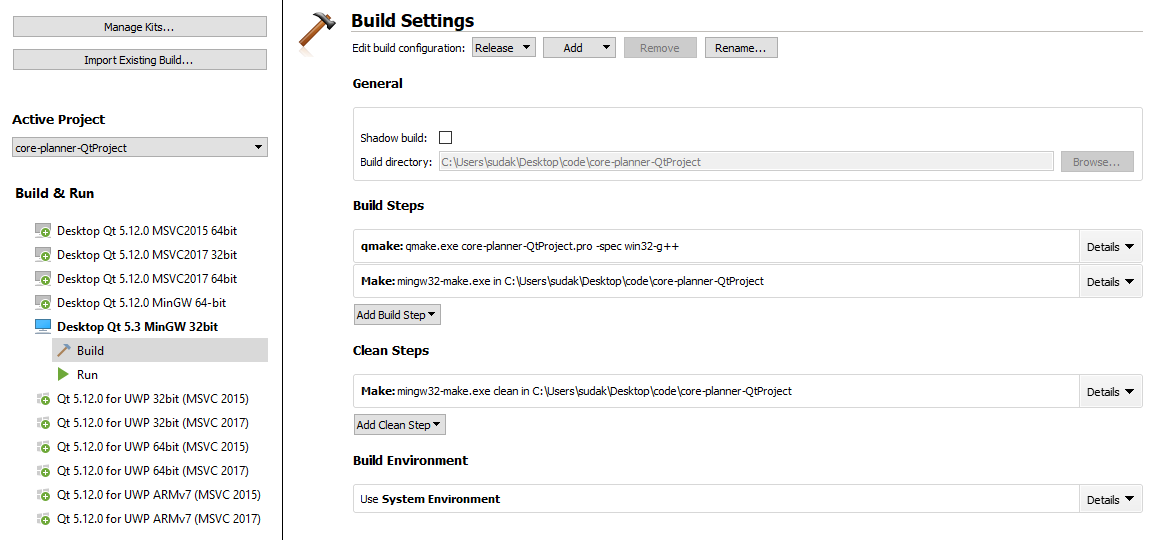


Рисунок 4.3 – Налаштування збірки проекту

* 1. Вхідні та вихідні дані

Вхідними даними виступають різні набори даних: розміри доступної площі для проектування; дані об’єктів у проекті (тип, найменування, розмір, положення на сцені, кут нахилу). Вихідними даними виступають \*.json файли, що зберігають впорядковану інформацію, або графічні (\*.png, \*jpg, \*.bmp) вже експортовані файли.

Також в програмі реалізована обробка виняткових ситуацій для окремих класів, що оброблюють помилки відкриття вхідних та вихідних файлів.

Для зберігання проектів використовуються файли формату \*.json.

* 1. Повідомлення

В процесі роботи програми можуть виникнути помилки в роботі програми. В такому разі оператор отримає інформаційні повідомлення щодо деталей помилки.

При роботі можуть бути отримані такі повідомлення:

* помилка відкриття файлу, така помилка може виникнути у випадку, якщо файл з інформацією знаходиться в невірній директорії або його пошкоджено (рис. 4.3). В такому разі необхідно перевірити правильність прописаного в коді шляху до текстового файлу, в якому зберігаються дані.

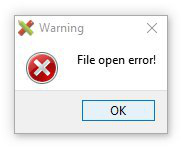


Рисунок 4.3 – Помилка відкриття файлу

У системі реалізовано функції винятків, тому усі непередбачені ситуації оброблюються та програма продовжує стабільно працювати.

1. Інструкція користувача

В даному розділі розглянуто призначення програми, умови її виконання, процес виконання програми та повідомлення для користувача.

* 1. Призначення програми

Програма призначена для роботи з різними наборами інформації, їх оновлення, зміни та зберігання в окремих \*.json файлах.

* 1. Умови використання програми

Програма має коректно працювати за умов:

* запуску останньої версії програми;
* наявність дисплею;
* наявність маніпулятору миші;
* наявність клавіатури;
* збереження наборів інформації у директорії, в яких є права на читання та запис файлів, що не є заблокованими.

До користувачів висуваються наступні вимоги:

* навички роботи з мишею, клавіатурою;
* навички роботи з графічним інтерфейсом операційної системи;
* володіння інформацією про призначення основних елементів керування більшості програм.
  1. Як запустити програму

Для роботи у програмі необхідно розархівувати теку з програмою та розташувати її у каталог, до якого ви маєте доступ. Для запуску програми у папці core-planner-QtProject (рис.4.2) треба запустити файл core-planner-QtProject.pro, дочекатися запуску проекту у QtCreator і натиснути кнопку «Збірка». Програма готова до роботи.

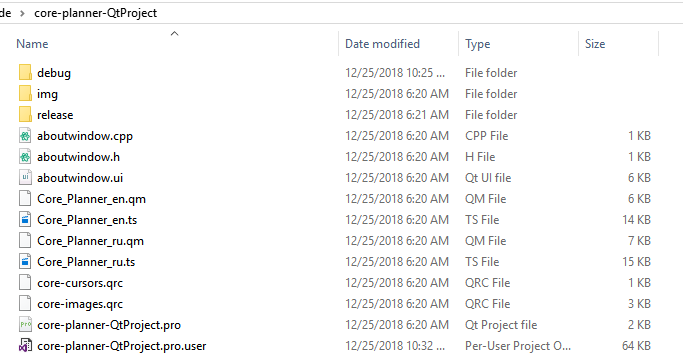


Рисунок 5.1 – Каталог з файлами програми

* 1. Виконання програми

Після того, як було запущено програму, перед користувачем постає головна форма з вільною Mdi областю. Головна форма зображена на рис. 5.2.

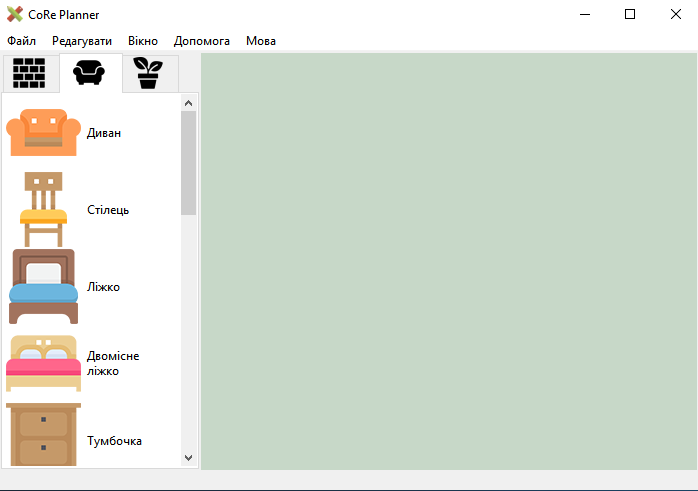


Рисунок 5.2 – Головна форма програми

Після створення нового проекту або відкриття вже існуючого, використовуючи пункти меню «Файл», у області відкритих проектів відобразиться сцена активного проекту планувальника (рис. 5.3). Для додавання нових об’єктів до проекту необхідно скористатися списком інструментів у лівій частині головної форми (натиснути лівою клавішею миші на необхідний об’єкт).

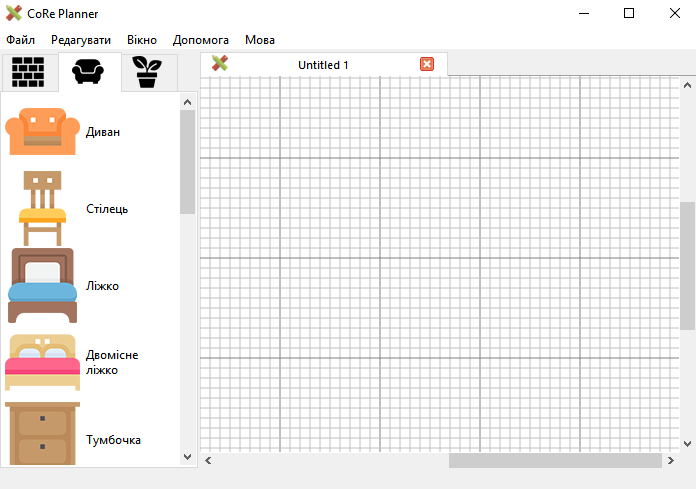


Рисунок 5.3 – Головна форма з відкритим проектом

Для створення нового проекту чи відкриття нещодавнього, користувач повинен натиснути пункт меню «Новий» на панелі головного меню. Після цього з’явиться форма створення проекту де користувач має змогу вказати розміри сцени проекту або відкрити нещодавній попередньо збережений проект.

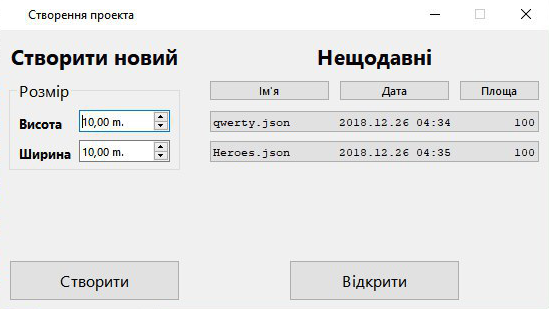


Рисунок 5.4 – Форма створення нового проекту або відкриття нещодавнього

Для точного вказання параметрів об’єкта необхідно обрати цей об’єкт на сцені та викликати «гарячу клавішу» Ctrl+E або обрати пункт головного меню «Змінити розмір». Це викликає форму, яка відображає поточні значення розміру, положення та куту нахилу об’єкта. У цій формі можна вказати нові значення параметрів та застосувати їх.

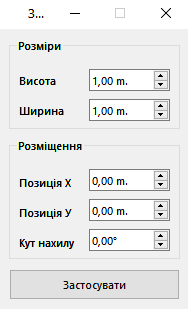


Рисунок 5.5 – Форма редагування властивостей об’єкта

Для перегляду інформації про програмний продукт та отримання контактів розробників існує спеціальна форма. Для її виклику необхідно обрати пункт головного меню «Про CoRe Planner…».

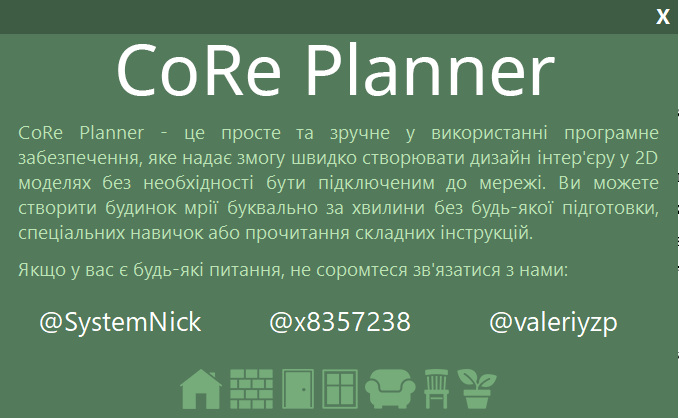


Рисунок 5.5 – Форма зворотного зв’язку

* 1. Повідомлення користувачу

В процесі роботи програми можуть виникнути помилки в роботі програми. В такому разі користувач отримає інформаційні повідомлення щодо деталей помилки.

У системі реалізовано функції винятків, тому усі непередбачені ситуації оброблюються та програма продовжує стабільно працювати. Одне з можливих повідомлень, що можуть бути отримані при роботі – помилка відкриття файлу, така помилка може виникнути у випадку, якщо файл з інформацією знаходиться в невірній директорії або його пошкоджено. Ілюстрація повідомлення приведена на рис. 5.6.

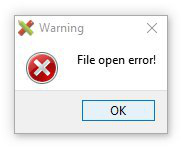


Рисунок 5.6 – Помилка відкриття файлу для читання

Висновки

Під час виконання курсового проекту було розроблено «Програмне забезпечення для проектування та редагування плану приміщень та внутрішнього інтер’єру».

Створена система організовує систему для забезпечення можливості проектувати нового плану приміщення або завантаження вже існуючого, додавання та видалення елементів інтер’єру, зберігання проекту у різних форматах. Основними характеристиками графічних об’єктів проектів є їх найменування, тип, розмір, розміщення та кут нахилу.

Також було виконано огляд та аналіз існуючих методів вирішення завдання та існуючих програмних засобів, таких як Smart Draw, Floor Planner, Planner 5D.

Вирішені наступні завдання курсового проекту:

* виконано огляд сучасних програмних засобів ведення нотаток;
* розглянуті можливості мови програмування С++, середовища розробки Qt Creator;
* розроблено програмне забезпечення для проектування та редагування плану приміщень та внутрішнього інтер’єру, яке повинне надавати користувачу можливість проектувати новий план приміщення або завантажувати вже існуючий, додавати та прибирати елементи інтер’єру, зберігати проект у різних форматах.

Перелік посилань

1. С++ – Вікіпедія: [Електрон. ресурс]. –  
   Режим доступу: https://uk.wikipedia.org/wiki/C%2B%2B.
2. Особливості С++: [Електрон. ресурс]. –  
   Режим доступу: http://test.tanet.edu.te.ua/articles/category/61/message/4398/
3. Qt Creator – Вікіпедія: [Електрон. ресурс]. –  
   Режим доступу: https://uk.wikipedia.org/wiki/Qt\_Creator.
4. Табунщик, Г.В. Методичні вказівки до виконання курсової роботи з дисципліни “Об’єктно-орієнтоване програмування” для студентів напряму підготовки 6.050103 «Програмна інженерія» всіх форм навчання. Частина 2. Основні теоретичні відомості [Текст] /Укл.: Г.В. Табунщик, Г.В. Неласа, Н.О. Миронова – Запоріжжя: ЗНТУ, 2010. – 70 с.
5. Все классы Qt (главный указатель): [Електрон. ресурс]. –  
   Режим доступа: http://www.doc.crossplatform.ru/qt/4.6.x/classes.html.

Додаток А  
Текст програми

А.1 Текст файлу aboutwindow.h

#ifndef ABOUTWINDOW\_H

#define ABOUTWINDOW\_H

#include <QtWidgets>

#include "ui\_aboutwindow.h"

namespace **Ui** {

class **AboutWindow**;

}

class **AboutWindow** : public QDialog

{

Q\_OBJECT

public:

explicit **AboutWindow**(QWidget \*parent = 0);

~***AboutWindow***();

private slots:

void **on\_pushButtonClose\_clicked**();

void **on\_labelSystemNick\_linkActivated**(const QString &link);

void **on\_labelx8357238\_linkActivated**(const QString &link);

void **on\_labelvaleriyzp\_linkActivated**(const QString &link);

private:

Ui::AboutWindow \*ui;

QPixmap backgroundImage;

void **setupOtherUi**();

};

#endif // ABOUTWINDOW\_H

А.2 Текст файлу aboutwindow.cpp

#include "aboutwindow.h"

AboutWindow::AboutWindow(QWidget \*parent) :

QDialog(parent),

ui(new Ui::AboutWindow)

{

ui->setupUi(this);

setupOtherUi();

}

AboutWindow::~AboutWindow()

{

delete ui;

}

void AboutWindow::setupOtherUi()

{

setWindowFlags(Qt::Window | Qt::FramelessWindowHint);

ui->pushButtonClose->setStyleSheet("QPushButton {color: white;}");

}

void AboutWindow::on\_pushButtonClose\_clicked()

{

close();

}

void AboutWindow::on\_labelSystemNick\_linkActivated(const QString &link)

{

QDesktopServices::openUrl(QUrl(link));

}

void AboutWindow::on\_labelx8357238\_linkActivated(const QString &link)

{

QDesktopServices::openUrl(QUrl(link));

}

void AboutWindow::on\_labelvaleriyzp\_linkActivated(const QString &link)

{

QDesktopServices::openUrl(QUrl(link));

}

А.3 Текст файлу database.h

#ifndef DATABASE\_H

#define DATABASE\_H

#include <QtSql>

#define DATABASE\_NAME "FileHistory.db"

#define TABLE "FileHistory"

#define TABLE\_FNAME "Name"

#define TABLE\_FDATE "Date"

#define TABLE\_FWAY "Way"

#define TABLE\_FSQUARE "Square"

struct **InfoAboutFile** {

public:

QString FileName;

QString Way;

QString DateOfSave;

double Square;

};

class **DataBase**

{

private:

QSqlDatabase dataBase;

int maxItems;

int itemsLeft;

int **itemsLeftDataBase**();

void **createTable**();

void **openDataBase**();

void **restoreDataBase**();

void **removeRecord**(const QString &fWay);

public:

**DataBase**();

~**DataBase**();

void **connectToDataBase**();

void **inserIntoTable**(const QString &fName, const QString &fData, const QString &fWay, const double &fSquare);

void **WriteTableToArray**(InfoAboutFile \*Array, int MaximumCount, int \*CurrentCount);

};

#endif // DATABASE\_H

А.4 Текст файлу database.cpp

#include "database.h"

DataBase::**DataBase**()

{

maxItems = 5;

}

DataBase::~**DataBase**()

{

dataBase.close();

}

void DataBase::**connectToDataBase**()

{

if(!QFile(DATABASE\_NAME).exists())

{

this->restoreDataBase();

}

else

{

this->openDataBase();

}

}

void DataBase::**restoreDataBase**()

{

this->openDataBase();

this->createTable();

}

void DataBase::**openDataBase**()

{

dataBase = QSqlDatabase::addDatabase("QSQLITE");

dataBase.setDatabaseName(DATABASE\_NAME);

try {

if(!dataBase.open())

{

throw QString("NOT OPEN SQLITE: " DATABASE\_NAME);

}

}

catch (QString error) {

qDebug()<<error;

}

}

void DataBase::**createTable**()

{

QSqlQuery query;

if(!query.exec( "CREATE TABLE " TABLE " ("

TABLE\_FNAME " BLOB NOT NULL,"

TABLE\_FDATE " VARCHAR(50) NOT NULL,"

TABLE\_FWAY " BLOB NOT NULL,"

TABLE\_FSQUARE " REAL NOT NULL"

" )"

))

{

throw QString("NOT CREATE TABLE SQLITE: " TABLE);

}

}

void DataBase::**inserIntoTable**(const QString&fName,const QString&fData,const QString&fWay,const double&fSquare)

{

removeRecord(fWay);

itemsLeft = itemsLeftDataBase();

QSqlQuery query;

if(itemsLeft < 0)

{

query.exec("SELECT \* FROM " TABLE );

query.first();

removeRecord(query.value(2).toString());

}

query.prepare("INSERT INTO " TABLE " ( " TABLE\_FNAME ", "

TABLE\_FDATE ", "

TABLE\_FWAY ", "

TABLE\_FSQUARE" ) "

"VALUES (:FName, :FData, :FWay, :FSquare)");

query.bindValue(":FName", fName);

query.bindValue(":FData", fData);

query.bindValue(":FWay", fWay);

query.bindValue(":FSquare", fSquare);

if(!query.exec())

{

throw QString("ERROR INSERT INTO SQLITE TABLE:" TABLE);

}

}

void DataBase::**WriteTableToArray**(InfoAboutFile \*Array, int MaximumCount, int \*CurrentCount)

{

QSqlQuery query;

query.exec("SELECT \* FROM " TABLE );

try{

if(!query.exec())

{

throw QString("ERROR WRITE TO ARRAY SQLITE: " TABLE);

}

while (query.next() && \*CurrentCount < MaximumCount)

{

QFile file;

file.setFileName(query.value(2).toString());

if(file.*open*(QIODevice::ReadOnly|QFile::Text))

{

file.*close*();

Array[\*CurrentCount].FileName = query.value(0).toString();

Array[\*CurrentCount].Way = query.value(2).toString();

Array[\*CurrentCount].DateOfSave = query.value(1).toString();

Array[\*CurrentCount].Square = query.value(3).toDouble();

\*CurrentCount += 1;

}

else

{

removeRecord(query.value(2).toString());

}

}

}

catch (QString error) {

qDebug()<<error;

}

}

void DataBase::**removeRecord**(const QString&fWay)

{

try{

QSqlQuery query;

query.prepare("DELETE FROM " TABLE " WHERE " TABLE\_FWAY "= '" + fWay+"'");

query.bindValue(TABLE\_FWAY,fWay);

if(!query.exec())

{

throw QString("ERROR REMOVE FROM SQLITE: " TABLE);

}

}

catch (QString error) {

qDebug()<<error;

}

}

int DataBase::**itemsLeftDataBase**(){

int res = maxItems-1;

QSqlQuery query;

query.exec("SELECT \* FROM " TABLE );

while(query.next())

res--;

return res;

}

А.5 Текст файлу itemeditwindow.h

#ifndef ITEMEDITWINDOW\_H

#define ITEMEDITWINDOW\_H

#include <QWidget>

#include "ui\_itemeditwindow.h"

#include "mdichild.h"

namespace Ui {

class ItemEditWindow;

}

class ItemEditWindow : public QWidget

{

Q\_OBJECT

public:

explicit ItemEditWindow(QWidget \*parent = 0);

ItemEditWindow(MainWindow \*Main, MdiChild \*Parent, QWidget \*parent = 0);

~ItemEditWindow();

private slots:

void on\_pushButtonApply\_clicked();

private:

Ui::ItemEditWindow \*ui;

MdiChild \*FormWithScene;

MainWindow \*MainWin;

};

#endif // ITEMEDITWINDOW\_H

А.6 Текст файлу itemeditwindow.cpp

#include "itemeditwindow.h"

ItemEditWindow::ItemEditWindow(QWidget \*parent) :

QWidget(parent),

ui(new Ui::ItemEditWindow)

{

ui->setupUi(this);

}

ItemEditWindow::~ItemEditWindow()

{

delete ui;

}

void ItemEditWindow::on\_pushButtonApply\_clicked()

{

if(ui->doubleSpinBoxWidth->value()\*100 >= 20)

if(ui->doubleSpinBoxHeight->value()\*100 >= 20)

if(ui->doubleSpinBoxPosX->value()\*100 >= 0 && ui->doubleSpinBoxPosX->value()\*100 <= FormWithScene->scene->sceneRect().toRect().width())

if(ui->doubleSpinBoxPosY->value()\*100 >= 0 && ui->doubleSpinBoxPosY->value()\*100 <= FormWithScene->scene->sceneRect().toRect().height())

{

FormWithScene->LastActive->Width = ui->doubleSpinBoxWidth->value()\*100;

FormWithScene->LastActive->Height = ui->doubleSpinBoxHeight->value()\*100;

FormWithScene->LastActive->setPos(ui->doubleSpinBoxPosX->value()\*100, ui->doubleSpinBoxPosY->value()\*100);

FormWithScene->LastActive->setRotation(ui->doubleSpinBoxAngle->value());

if(FormWithScene->AfterUndo)

{

FormWithScene->SaveFileInfo.DeleteHighVersions(FormWithScene->VersionOfScene);

FormWithScene->AfterUndo = false;

FormWithScene->MaximumVersionOfScene = FormWithScene->VersionOfScene;

}

FormWithScene->VersionOfScene++;

FormWithScene->MaximumVersionOfScene++;

FormWithScene->SaveFileInfo.SaveToHistory();

FormWithScene->SaveFileInfo.LoadFromHistory(FormWithScene->VersionOfScene);

this->close();

}

}

ItemEditWindow::ItemEditWindow(MainWindow \*Main, MdiChild \*Parent, QWidget \*parent) :

QWidget(parent),

ui(new Ui::ItemEditWindow)

{

ui->setupUi(this);

FormWithScene = Parent;

MainWin = Main;

ui->doubleSpinBoxWidth->setValue(Parent->LastActive->GetWidth()/100);

ui->doubleSpinBoxHeight->setValue(Parent->LastActive->GetHeight()/100);

ui->doubleSpinBoxPosX->setValue(Parent->LastActive->pos().x()/100);

ui->doubleSpinBoxPosY->setValue(Parent->LastActive->pos().y()/100);

ui->doubleSpinBoxAngle->setValue(Parent->LastActive->rotation());

QPixmap NullPicture = QPixmap(1,1);

NullPicture.fill(QColor(0,0,0,0));

setWindowIcon(QIcon(NullPicture));

}

А.7 Текст файлу fileopen.h

#ifndef FILEOPEN\_H

#define FILEOPEN\_H

#include "fileway.h"

#include "jsondata.h"

class MainWindow;

class MdiChild;

class FileOpen : public FileWay

{

private:

MainWindow \*Main;

public:

FileOpen(MainWindow \*Parent);

void SetWay();

void OpenFile();

void OpenFileByWay(QString &Way, QString &FileName);

};

#endif // FILEOPEN\_H

А.8 Текст файлу fileopen.cpp

#include "fileopen.h"

#include "mdichild.h"

#include "mainwindow.h"

FileOpen::FileOpen(MainWindow \*Parent) : FileWay()

{

Main = Parent;

FileName = "";

Way = "";

}

void FileOpen::SetWay()

{

Way = QFileDialog::getOpenFileName(nullptr, "Відкрити... | Open... | Открыть...", "D:\\", "\*.json");

FileName = GetFileNameFromWay(Way);

}

void FileOpen::OpenFile()

{

try {

SetWay();

if(FileName == "" || Way == "")

{

throw QString("File open error!");

}

MdiChild \*Child = new MdiChild(Main);

Child->setWindowTitle(FileName);

Child->setWindowIcon(QIcon(":/menu/cube"));

Child->SaveFileInfo.Way = Way;

Child->SaveFileInfo.FileName = FileName;

JsonData JsonReader;

JsonReader.inFromFile(Way, Child);

Main->loadProject(Child);

Child->SaveFileInfo.InitHistory();

Child->VersionOfScene = 0;

Child->MaximumVersionOfScene = 0;

}

catch (QString error) {

QMessageBox::critical(Main, "Warning", error);

qDebug()<<error;

}

}

void FileOpen::OpenFileByWay(QString &Way, QString &FileName)

{

MdiChild \*Child = new MdiChild(Main);

Child->setWindowTitle(FileName);

Child->setWindowIcon(QIcon(":/menu/cube"));

Child->SaveFileInfo.Way = Way;

Child->SaveFileInfo.FileName = FileName;

JsonData JsonReader;

JsonReader.inFromFile(Way, Child);

Main->loadProject(Child);

Child->SaveFileInfo.InitHistory();

Child->VersionOfScene = 0;

Child->MaximumVersionOfScene = 0;

}

А.9 Текст файлу filesave.h

#ifndef FILESAVE\_H

#define FILESAVE\_H

#include "fileway.h"

#include "database.h"

class MdiChild;

class MainWindow;

class FileSave : public FileWay

{

public:

MdiChild \*ParenWithScene;

FileSave();

~FileSave();

void Save(MainWindow \*Main);

void SaveAs(MainWindow \*Main);

void SaveToHistory();

void InitHistory();

void LoadFromHistory(int Version);

void DeleteHighVersions(int Version);

private:

void SetWay();

QString HistoryFileWay;

};

#endif // FILESAVE\_H

А.10 Текст файлу filesave.cpp

#include "filesave.h"

#include "mdichild.h"

#include "mainwindow.h"

#include "jsondata.h"

FileSave::FileSave() : FileWay()

{

ParenWithScene = NULL;

QDate DateNow = QDate::currentDate();

QTime TimeNow = QTime::currentTime();

HistoryFileWay = DateNow.toString("yyyyMMdd") + TimeNow.toString("hhmmss") + ".json";

}

void FileSave::SetWay()

{

QString adress = QFileDialog::getSaveFileName(nullptr, "Зберегти в... | Save to... | Сохранить в...","D:\\","Project files (\*.json);;JPG file (\*.jpg);;BMP file(\*.bmp);;PNG file(\*.png)");

try {

if(adress == "")

{

throw "ERROR ADRESS IS CLEAR";

}

Way = adress;

FileName = GetFileNameFromWay(adress);

}

catch (QString error) {

qDebug()<<error;

}

}

FileSave::~FileSave()

{

QFile file;

file.setFileName(HistoryFileWay);

if(file.open(QIODevice::ReadWrite|QFile::Text))

{

file.remove();

}

}

void FileSave::Save(MainWindow \*Main)

{

if(FileName != "" && Way != "")

{

JsonData JsonWriter;

JsonWriter.outToFile(Way, ParenWithScene);

QDate DateNow = QDate::currentDate();

QTime TimeNow = QTime::currentTime();

QString Date = DateNow.toString("yyyy.MM.dd ") + TimeNow.toString("hh:mm");

double Square = (ParenWithScene->scene->sceneRect().toRect().width() \* ParenWithScene->scene->sceneRect().toRect().height()) / 10000.0;

Main->database.inserIntoTable(FileName, Date, Way, Square);

}

else

{

SaveAs(Main);

}

}

void FileSave::SaveAs(MainWindow \*Main)

{

SetWay();

if(Way != "" && FileName != "")

{

ParenWithScene->setWindowTitle(FileName);

if(FileName[FileName.length()-1] == 'n')

{

JsonData JsonWriter;

JsonWriter.outToFile(Way, ParenWithScene);

ParenWithScene->setWindowTitle(FileName);

QDate DateNow = QDate::currentDate();

QTime TimeNow = QTime::currentTime();

QString Date = DateNow.toString("yyyy.MM.dd ") + TimeNow.toString("hh:mm");

double Square = (ParenWithScene->scene->sceneRect().toRect().width() \* ParenWithScene->scene->sceneRect().toRect().height()) / 10000.0;

Main->database.inserIntoTable(FileName, Date, Way, Square);

}

else

{

ParenWithScene->SaveAsImage(Way);

}

}

}

void FileSave::SaveToHistory()

{

JsonData JsonWriter;

JsonWriter.outHistoryToFile(HistoryFileWay, ParenWithScene);

}

void FileSave::InitHistory()

{

JsonData JsonWriter;

JsonWriter.outInitHistoryFile(HistoryFileWay, ParenWithScene);

}

void FileSave::LoadFromHistory(int Version)

{

JsonData JsonWriter;

JsonWriter.InitScenFromFile(HistoryFileWay, ParenWithScene, Version);

}

void FileSave::DeleteHighVersions(int Version)

{

JsonData JsonWriter;

JsonWriter.DeleteHighVersions(HistoryFileWay ,Version);

}

А.11 Текст файлу fileway.h

#ifndef FILEWAY\_H

#define FILEWAY\_H

#include <QString>

class **FileWay**

{

public:

QString Way;

QString FileName;

QString **GetFileNameFromWay**(QString &AdressFile);

**FileWay**();

**FileWay**(QString &WayToFile);

virtual ~***FileWay***();

virtual void ***SetWay***() = 0;

};

#endif // FILEWAY\_H

А.12 Текст файлу fileway.cpp

#include "fileway.h"

FileWay::**FileWay**()

{

Way = "";

FileName = "";

}

FileWay::**FileWay**(QString &WayToFile)

{

Way = WayToFile;

FileName = GetFileNameFromWay(*Way*);

}

FileWay::~***FileWay***()

{

}

QString FileWay::**GetFileNameFromWay**(QString &AdressFile)

{

QString NameFile = "";

for(int i = AdressFile.length()-1; i >= 0; i--)

{

if(AdressFile[i] == '\\' || AdressFile[i] == '/')

{

for(int j = i+1; j < AdressFile.length(); j++)

NameFile += AdressFile[j];

break;

}

}

return NameFile;

}

А.13 Текст файлу formcreate.h

#ifndef FORMCREATE\_H

#define FORMCREATE\_H

#include <QWidget>

#include "ui\_formcreate.h"

#include "mdichild.h"

#include "database.h"

namespace Ui {

class FormCreate;

}

class MainWindow;

class FormCreate : public QWidget

{

Q\_OBJECT

public:

explicit FormCreate(MainWindow \*Main, QWidget \*parent = 0);

~FormCreate();

private slots:

void on\_pushButtonCreateNew\_clicked();

void on\_pushButtonProjectName\_clicked();

void on\_pushButtonProjectTime\_clicked();

void on\_pushButtonProjectSquare\_clicked();

void SetFilesButtons();

void SetButtonsTextAlign();

void on\_pushButton\_1\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_5\_clicked();

void on\_pushButtonOpen\_clicked();

private:

InfoAboutFile \*Files;

int CountOfFiles;

int WayToOpen;

MainWindow \*MainWin;

Ui::FormCreate \*ui;

};

#endif // FORMCREATE\_H

А.14 Текст файлу formcreate.cpp

#include "formcreate.h"

#include "mainwindow.h"

FormCreate::FormCreate(MainWindow \*Main, QWidget \*parent) :

QWidget(parent),

ui(new Ui::FormCreate)

{

ui->setupUi(this);

Files = new InfoAboutFile[5];

CountOfFiles = 0;

MainWin = Main;

MainWin->database.WriteTableToArray(Files, 5, &CountOfFiles);

SetFilesButtons();

WayToOpen = -1;

QPixmap NullPicture = QPixmap(1,1);

NullPicture.fill(QColor(0,0,0,0));

setWindowIcon(QIcon(NullPicture));

}

FormCreate::~FormCreate()

{

delete ui;

}

void FormCreate::on\_pushButtonCreateNew\_clicked()

{

if(ui->doubleSpinBoxWidth->value() > 0)

if(ui->doubleSpinBoxHeight->value() > 0)

{

MdiChild \*child = new MdiChild(MainWin);

child->setWindowTitle("Untitled " + QString::number(MainWin->uniqueProjectID++));

child->SaveFileInfo.FileName = child->windowTitle();

child->setWindowIcon(QIcon(":/menu/cube"));

child->SetSceneParametrs(int(ui->doubleSpinBoxWidth->value()\*100), int(ui->doubleSpinBoxHeight->value()\*100));

MainWin->loadProject(child);

this->close();

}

}

void SwapInfoAboutFile(InfoAboutFile \*File1, InfoAboutFile \*File2)

{

InfoAboutFile temp;

temp.DateOfSave = File1->DateOfSave;

temp.FileName = File1->FileName;

temp.Way = File1->Way;

temp.Square = File1->Square;

File1->DateOfSave = File2->DateOfSave;

File1->FileName = File2->FileName;

File1->Way = File2->Way;

File1->Square = File2->Square;

File2->DateOfSave = temp.DateOfSave;

File2->FileName = temp.FileName;

File2->Way = temp.Way;

File2->Square = temp.Square;

}

void ShellSortByDate(InfoAboutFile \*File, int Count)

{

for (int gap = Count/2; gap > 0; gap /= 2)

{

for (int i = gap; i < Count; ++i)

{

for (int j = i-gap; j >= 0; j -= gap)

{

if (File[j+gap].DateOfSave <= File[j].DateOfSave) break;

else

{

SwapInfoAboutFile(&File[j], &File[j+gap]);

}

}

}

}

}

void BubbleSortBySquare(InfoAboutFile \*File, int Count)

{

for(int i = 0; i < Count-1; i++)

for(int j = 0; j < Count-1; j++)

if(File[j].Square > File[j+1].Square)

SwapInfoAboutFile(&File[j], &File[j+1]);

}

void InsertionSortByFileName(InfoAboutFile \*File, int Count)

{

int i, j;

InfoAboutFile key;

for(i = 0; i < Count; i++)

{

j = i-1;

while(j >= 0 && File[j+1].FileName < File[j].FileName)

{

SwapInfoAboutFile(&File[j], &File[j+1]);

j--;

}

}

}

void FormCreate::on\_pushButtonProjectName\_clicked()

{

InsertionSortByFileName(Files, CountOfFiles);

SetFilesButtons();

}

void FormCreate::on\_pushButtonProjectTime\_clicked()

{

ShellSortByDate(Files, CountOfFiles);

SetFilesButtons();

}

void FormCreate::on\_pushButtonProjectSquare\_clicked()

{

BubbleSortBySquare(Files, CountOfFiles);

SetFilesButtons();

}

void FormCreate::SetFilesButtons()

{

if(CountOfFiles >= 1) ui->pushButton\_1->setText((Files[0].FileName).leftJustified(16, ' ') + " " + (Files[0].DateOfSave).leftJustified(16, ' ') + " " + (QVariant(Files[0].Square).toString()).rightJustified(10, ' '));

else ui->pushButton\_1->setVisible(false);

if(CountOfFiles >= 2) ui->pushButton\_2->setText((Files[1].FileName).leftJustified(16, ' ') + " " + (Files[1].DateOfSave).leftJustified(16, ' ') + " " + (QVariant(Files[1].Square).toString()).rightJustified(10, ' '));

else ui->pushButton\_2->setVisible(false);

if(CountOfFiles >= 3) ui->pushButton\_3->setText((Files[2].FileName).leftJustified(16, ' ') + " " + (Files[2].DateOfSave).leftJustified(16, ' ') + " " + (QVariant(Files[2].Square).toString()).rightJustified(10, ' '));

else ui->pushButton\_3->setVisible(false);

if(CountOfFiles >= 4) ui->pushButton\_4->setText((Files[3].FileName).leftJustified(16, ' ') + " " + (Files[3].DateOfSave).leftJustified(16, ' ') + " " + (QVariant(Files[3].Square).toString()).rightJustified(10, ' '));

else ui->pushButton\_4->setVisible(false);

if(CountOfFiles >= 5) ui->pushButton\_5->setText((Files[4].FileName).leftJustified(16, ' ') + " " + (Files[4].DateOfSave).leftJustified(16, ' ') + " " + (QVariant(Files[4].Square).toString()).rightJustified(10, ' '));

else ui->pushButton\_5->setVisible(false);

SetButtonsTextAlign();

}

void FormCreate::SetButtonsTextAlign()

{

ui->pushButton\_1->setStyleSheet("text-align: left;");

ui->pushButton\_2->setStyleSheet("text-align: left;");

ui->pushButton\_3->setStyleSheet("text-align: left;");

ui->pushButton\_4->setStyleSheet("text-align: left;");

ui->pushButton\_5->setStyleSheet("text-align: left;");

}

void FormCreate::on\_pushButton\_1\_clicked()

{

WayToOpen = 0;

SetButtonsTextAlign();

ui->pushButton\_1->setStyleSheet("background-color: rgb(173, 255, 153); text-align: left;");

}

void FormCreate::on\_pushButton\_2\_clicked()

{

WayToOpen = 1;

SetButtonsTextAlign();

ui->pushButton\_2->setStyleSheet("background-color: rgb(173, 255, 153); text-align: left;");

}

void FormCreate::on\_pushButton\_3\_clicked()

{

WayToOpen = 2;

SetButtonsTextAlign();

ui->pushButton\_3->setStyleSheet("background-color: rgb(173, 255, 153); text-align: left;");

}

void FormCreate::on\_pushButton\_4\_clicked()

{

WayToOpen = 3;

SetButtonsTextAlign();

ui->pushButton\_4->setStyleSheet("background-color: rgb(173, 255, 153); text-align: left;");

}

void FormCreate::on\_pushButton\_5\_clicked()

{

WayToOpen = 4;

SetButtonsTextAlign();

ui->pushButton\_5->setStyleSheet("background-color: rgb(173, 255, 153); text-align: left;");

}

void FormCreate::on\_pushButtonOpen\_clicked()

{

if(WayToOpen != -1)

{

FileOpen OpenFile(MainWin);

OpenFile.OpenFileByWay(Files[WayToOpen].Way, Files[WayToOpen].FileName);

this->close();

}

}

А.15 Текст файлу graphicitem.h

#ifndef GRAPHICITEM\_H

#define GRAPHICITEM\_H

#include <QObject>

#include <QGraphicsItem>

#include <QPainter>

#include <QGraphicsSceneMouseEvent>

#include <QCursor>

#include <QtMath>

#include "points.h"

struct **GraphicItemInfo**{

QString Type;

QString Value;

double X;

double Y;

double Width;

double Height;

double Angle;

};

enum **CornerFlags**{

Top = 1,

Bottom = 2,

Left = 4,

Right = 8,

TopLeft = Top|Left,

TopRight = Top|Right,

BottomLeft = Bottom|Left,

BottomRight = Bottom|Right

};

class **MdiChild**;

class **GraphicItem** : public QObject, public QGraphicsItem

{

public:

points MidlePoints;

double Width;

double Height;

int FlagCorner;

bool IsEditNow;

bool IsResizeModNow;

int SizeOfPoint;

double MinimumWidth;

double MinimumHeight;

double CurrentAngleOfRotation;

QPointF ClickedPointOnScene;

QPointF m\_shiftMouseCoords;

MdiChild \*ParenWithScene;

bool IsSceneEdited;

**GraphicItem**(double setWidth, double setHeight, MdiChild \*MdiChildParent, QObject \*parent = 0);

~***GraphicItem***();

virtual QString ***GetType***() = 0;

double **GetHeight**() {return Height;}

double **GetWidth**() {return Width;}

double **GetRotationAngle**() {return this->rotation();}

void **SetRotationAngle**(double RotationAngleToSet) {this->setRotation(RotationAngleToSet);}

void **SetHeight**(double HeightToSet) {Height = HeightToSet > MinimumHeight ? HeightToSet : MinimumHeight;}

void **SetWidth**(double WidthToSet) {Width = WidthToSet > MinimumWidth ? WidthToSet : MinimumWidth;}

void **SetUnActive**() {IsEditNow = false;}

virtual GraphicItemInfo ***GetItemInfo***() = 0;

QRectF ***boundingRect***() const;

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) = 0;

void ***mouseMoveEvent***(QGraphicsSceneMouseEvent \*event);

void ***mousePressEvent***(QGraphicsSceneMouseEvent \*event);

void ***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event);

void ***mouseDoubleClickEvent***(QGraphicsSceneMouseEvent \*event);

void ***hoverMoveEvent***(QGraphicsSceneHoverEvent \*event);

void **ResetCursor**(double CursorX, double CursorY, bool MouseMove);

void **SetPoints**();

};

#endif // GRAPHICITEM\_H

А.16 Текст файлу graphicitem.cpp

#include "graphicitem.h"

#include "mdichild.h"

double GetAngleABC( QPointF A, QPointF B, QPointF C)

{

QPointF AB( B.x() - A.x(), B.y() - A.y() );

QPointF CB ( B.x() - C.x(), B.y() - C.y() );

double ScalarProduct = (AB.x() \* CB.x() + AB.y() \* CB.y());

double Proection = (AB.x() \* CB.y() - AB.y() \* CB.x());

double RadianAngle = qAtan2(Proection, ScalarProduct);

return -(RadianAngle \* 180. / M\_PI + 0.5);

}

GraphicItem::GraphicItem(double setWidth, double setHeight, MdiChild \*MdiChildParent, QObject \*parent) : QObject(parent), QGraphicsItem()

{

IsEditNow = false;

IsResizeModNow = false;

SizeOfPoint = 6;

MinimumWidth = 20;

MinimumHeight = 20;

CurrentAngleOfRotation = 0;

SetHeight(setHeight);

SetWidth(setWidth);

ParenWithScene = MdiChildParent;

setAcceptHoverEvents(true);

IsSceneEdited = false;

if(ParenWithScene->LastActive != NULL && ParenWithScene->LastActive != this) ParenWithScene->LastActive->SetUnActive();

ParenWithScene->LastActive = this;

}

GraphicItem::~GraphicItem()

{

}

QRectF GraphicItem::boundingRect() const

{

return QRectF (-Width/2,-Height/2,Width,Height);

}

void GraphicItem::mouseDoubleClickEvent(QGraphicsSceneMouseEvent \*event)

{

IsEditNow = !IsEditNow;

update();

}

void GraphicItem::hoverMoveEvent(QGraphicsSceneHoverEvent \*event)

{

ResetCursor(event->pos().x(), event->pos().y(), false);

}

void GraphicItem::mouseMoveEvent(QGraphicsSceneMouseEvent \*event)

{

IsSceneEdited = true;

if(IsEditNow)

{

if(IsResizeModNow)

{

double dHeight = Height, dWidth = Width;

switch(FlagCorner)

{

case Top:

{

Height -= (event->pos().y() + Height/2);

if(Height >= MinimumHeight) this->setPos(mapToScene(0, (event->pos().y() + Height/2)));

else {Height = MinimumHeight; this->setPos(MidlePoints.GetBottomMinimumCenter());}

break;

}

case Bottom:

{

Height += (event->pos().y() - Height/2);

if(Height >= MinimumHeight) this->setPos(mapToScene(0, (event->pos().y() - Height/2)));

else {Height = MinimumHeight; this->setPos(MidlePoints.GetTopMinimumCenter());}

break;

}

case Left:

{

Width -= (event->pos().x() + Width/2);

if(Width >= MinimumWidth) this->setPos(mapToScene((event->pos().x() + Width/2), 0));

else {Width = MinimumWidth; this->setPos(MidlePoints.GetRightMinimumCenter());}

break;

}

case Right:

{

Width += (event->pos().x() - Width/2);

if(Width >= MinimumWidth) this->setPos(mapToScene((event->pos().x() - Width/2), 0));

else {Width = MinimumWidth; this->setPos(MidlePoints.GetLeftMinimumCenter());}

break;

}

case TopLeft:

{

Height -= (event->pos().y() + Height/2);

Width -= (event->pos().x() + Width/2);

Height = Height < MinimumHeight ? MinimumHeight : Height;

Width = Width < MinimumWidth ? MinimumWidth : Width;

dHeight -= Height;

dWidth -= Width;

this->setPos(mapToScene(dWidth/2, dHeight/2));

break;

}

case TopRight:

{

Height -= (event->pos().y() + Height/2);

Width += (event->pos().x() - Width/2);

Height = Height < MinimumHeight ? MinimumHeight : Height;

Width = Width < MinimumWidth ? MinimumWidth : Width;

dHeight -= Height;

dWidth -= Width;

this->setPos(mapToScene(-dWidth/2, dHeight/2));

break;

}

case BottomLeft:

{

Height += (event->pos().y() - Height/2);

Width -= (event->pos().x() + Width/2);

Height = Height < MinimumHeight ? MinimumHeight : Height;

Width = Width < MinimumWidth ? MinimumWidth : Width;

dHeight -= Height;

dWidth -= Width;

this->setPos(mapToScene(dWidth/2, -dHeight/2));

break;

}

case BottomRight:

{

Height += (event->pos().y() - Height/2);

Width += (event->pos().x() - Width/2);

Height = Height < MinimumHeight ? MinimumHeight : Height;

Width = Width < MinimumWidth ? MinimumWidth : Width;

dHeight -= Height;

dWidth -= Width;

this->setPos(mapToScene(-dWidth/2, -dHeight/2));

break;

}

}

update();

}

else

{

double Angle= GetAngleABC(mapToScene(event->pos()),mapToScene(0,0),ClickedPointOnScene);

this->setRotation(CurrentAngleOfRotation+Angle);

}

}

else

{

this->setPos(mapToScene(event->pos()).x() + m\_shiftMouseCoords.x(), mapToScene(event->pos()).y()+m\_shiftMouseCoords.y());

}

if (this->pos().x() < 0) this->setPos(0, pos().y());

else if (this->pos().x() > ParenWithScene->scene->sceneRect().toRect().width()) this->setPos(ParenWithScene->scene->sceneRect().toRect().width(), pos().y());

if (this->pos().y() < 0) this->setPos(pos().x(), 0);

else if (this->pos().y() > ParenWithScene->scene->sceneRect().toRect().height()) this->setPos(pos().x(), ParenWithScene->scene->sceneRect().toRect().height());

}

void GraphicItem::mousePressEvent(QGraphicsSceneMouseEvent \*event)

{

m\_shiftMouseCoords = this->pos() - mapToScene(event->pos());

CurrentAngleOfRotation=this->rotation();

ClickedPointOnScene= mapToScene(event->pos());

ResetCursor(event->pos().x(), event->pos().y(), true);

SetPoints();

if(ParenWithScene->LastActive != NULL && ParenWithScene->LastActive != this) ParenWithScene->LastActive->SetUnActive();

ParenWithScene->LastActive = this;

ParenWithScene->UpdateScene();

}

void GraphicItem::mouseReleaseEvent(QGraphicsSceneMouseEvent \*event)

{

ResetCursor(event->pos().x(), event->pos().y(), false);

if(IsSceneEdited)

{

if(ParenWithScene->AfterUndo)

{

ParenWithScene->SaveFileInfo.DeleteHighVersions(ParenWithScene->VersionOfScene);

ParenWithScene->AfterUndo = false;

ParenWithScene->MaximumVersionOfScene = ParenWithScene->VersionOfScene;

}

ParenWithScene->VersionOfScene++;

ParenWithScene->MaximumVersionOfScene++;

ParenWithScene->SaveFileInfo.SaveToHistory();

ParenWithScene->SaveFileInfo.LoadFromHistory(ParenWithScene->VersionOfScene);

IsSceneEdited = false;

}

}

void GraphicItem::ResetCursor(double CursorX, double CursorY, bool MouseMove)

{

if(IsEditNow)

{

if(CursorX <= -Width/2+SizeOfPoint || CursorX >= Width/2-SizeOfPoint || CursorY <= -Height/2+SizeOfPoint || CursorY >= Height/2-SizeOfPoint)

{

QString NameOfCursor;

IsResizeModNow = true;

FlagCorner = 0;

if(CursorX <= (-Width/2 + SizeOfPoint)) FlagCorner |= Left;

if(CursorX >= (Width/2 - SizeOfPoint)) FlagCorner |= Right;

if(CursorY <= (-Height/2 + SizeOfPoint)) FlagCorner |= Top;

if(CursorY >= (Height/2 - SizeOfPoint)) FlagCorner |= Bottom;

switch(FlagCorner)

{

case Top:

case Bottom:

{

NameOfCursor = ":/cursors/ResizeV";

break;

}

case Left:

case Right:

{

NameOfCursor = ":/cursors/ResizeH";

break;

}

case TopLeft:

case BottomRight:

{

NameOfCursor = ":/cursors/ResizeD1";

break;

}

case TopRight:

case BottomLeft:

{

NameOfCursor = ":/cursors/ResizeD2";

break;

}

}

QPixmap Item(NameOfCursor);

QTransform trans = transform();

trans.rotate(this->rotation());

Item.transformed(trans);

this->setCursor(Item.transformed(trans).scaled(24,24,Qt::KeepAspectRatio, Qt::SmoothTransformation));

}

else

{

IsResizeModNow = false;

this->setCursor(QPixmap(":/cursors/Rotation").scaled(24,24,Qt::KeepAspectRatio, Qt::SmoothTransformation));

}

}

else

{

if(MouseMove) this->setCursor(Qt::ClosedHandCursor);

else this->setCursor(Qt::ArrowCursor);

}

}

void GraphicItem::SetPoints()

{

MidlePoints.SetTopMinimumCenter(mapToScene(0, - Height/2 + MinimumHeight/2));

MidlePoints.SetLeftMinimumCenter(mapToScene(-Width/2 + MinimumWidth/2, 0));

MidlePoints.SetRightMinimumCenter(mapToScene(Width - Width/2 - MinimumWidth/2, 0));

MidlePoints.SetBottomMinimumCenter(mapToScene(0, Height - Height/2 - MinimumHeight/2));

}

А.17 Текст файлу itembycolor.h

#ifndef ITEMBYCOLOR\_H

#define ITEMBYCOLOR\_H

#include "graphicitem.h"

class **ItemByColor** : public GraphicItem

{

public:

**ItemByColor**(double setWidth, double setHeight, QColor setColor, MdiChild \*MdiChildParent, QObject \*parent = 0);

QString ***GetType***() {return "Color";}

GraphicItemInfo ***GetItemInfo***();

private:

QColor ColorOfItem;

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget);

};

#endif // ITEMBYCOLOR\_H

А.18 Текст файлу itembycolor.cpp

#include "itembycolor.h"

ItemByColor::**ItemByColor**(double setWidth, double setHeight, QColor setColor, MdiChild \*MdiChildParent, QObject \*parent)

: GraphicItem(setWidth, setHeight, MdiChildParent, parent)

{

ColorOfItem = setColor;

}

GraphicItemInfo ItemByColor::***GetItemInfo***()

{

GraphicItemInfo Information;

Information.Angle = this->rotation();

Information.Width = this->Width;

Information.Height = this->Height;

Information.X = this->pos().x();

Information.Y = this->pos().y();

Information.Type = "Color";

Information.Value = ColorOfItem.name();

return Information;

}

void ItemByColor::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->save();

QPen pointpen(Qt::black);

painter->setPen(Qt::NoPen);

painter->setBrush(ColorOfItem);

painter->drawRect(int(-Width/2), int(-Height/2), int(Width), int(Height));

if(IsEditNow)

{

painter->setPen(Qt::DashLine);

painter->setBrush(Qt::NoBrush);

painter->drawRect(int(-Width/2), int(-Height/2), int(Width), int(Height));

pointpen.setWidth(SizeOfPoint);

painter->setPen(pointpen);

int LeftTopPointX = int(-Width/2+SizeOfPoint/2);

int LeftTopPointY = int(-Height/2+SizeOfPoint/2);

int RightBottomPointX = int(Width)%2 == 0 ? int(Width-Width/2-SizeOfPoint/2) : int(Width-Width/2-SizeOfPoint/2+1);

int RightBottomPointY = int(Height)%2 == 0 ? int(Height-Height/2-SizeOfPoint/2) : int(Height-Height/2-SizeOfPoint/2+1);

painter->drawPoint(LeftTopPointX, LeftTopPointY);

painter->drawPoint(RightBottomPointX, LeftTopPointY);

painter->drawPoint(LeftTopPointX, RightBottomPointY);

painter->drawPoint(RightBottomPointX, RightBottomPointY);

}

painter->restore();

}

А.19 Текст файлу itembypicture.h

#ifndef ITEMBYPICTURE\_H

#define ITEMBYPICTURE\_H

#include "graphicitem.h"

class **ItemByPicture** : public GraphicItem

{

public:

**ItemByPicture**(double setWidth, double setHeight, QString PictureName, MdiChild \*MdiChildParent, QObject \*parent = 0);

QString ***GetType***() {return "Picture";}

GraphicItemInfo ***GetItemInfo***();

private:

QString NameOfPicture;

QImage PictureOfItem;

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget);

};

#endif // ITEMBYPICTURE\_H

А.20 Текст файлу itembypicture.cpp

#include "itembypicture.h"

ItemByPicture::**ItemByPicture**(double setWidth, double setHeight, QString PictureName, MdiChild \*MdiChildParent, QObject \*parent)

: GraphicItem(setWidth, setHeight, MdiChildParent, parent)

{

NameOfPicture = PictureName;

PictureOfItem.load(PictureName);

}

GraphicItemInfo ItemByPicture::***GetItemInfo***()

{

GraphicItemInfo Information;

Information.Angle = this->rotation();

Information.Width = this->Width;

Information.Height = this->Height;

Information.X = this->pos().x();

Information.Y = this->pos().y();

Information.Type = "Picture";

Information.Value = NameOfPicture;

return Information;

}

void ItemByPicture::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->save();

QPen pointpen(Qt::black);

QPainter PaintPicture;

painter->drawImage(int(-Width/2), int(-Height/2), PictureOfItem.scaled(int(Width),int(Height), Qt::IgnoreAspectRatio, Qt::SmoothTransformation));

if(IsEditNow)

{

painter->setPen(Qt::DashLine);

painter->setBrush(Qt::NoBrush);

painter->drawRect(int(-Width/2), int(-Height/2), int(Width), int(Height));

pointpen.setWidth(SizeOfPoint);

painter->setPen(pointpen);

int LeftTopPointX = int(-Width/2+SizeOfPoint/2);

int LeftTopPointY = int(-Height/2+SizeOfPoint/2);

int RightBottomPointX = int(Width)%2 == 0 ? int(Width-Width/2-SizeOfPoint/2) : int(Width-Width/2-SizeOfPoint/2+1);

int RightBottomPointY = int(Height)%2 == 0 ? int(Height-Height/2-SizeOfPoint/2) : int(Height-Height/2-SizeOfPoint/2+1);

painter->drawPoint(LeftTopPointX, LeftTopPointY);

painter->drawPoint(RightBottomPointX, LeftTopPointY);

painter->drawPoint(LeftTopPointX, RightBottomPointY);

painter->drawPoint(RightBottomPointX, RightBottomPointY);

}

painter->restore();

}

А.21 Текст файлу jsondata.h

#ifndef JSONDATA\_H

#define JSONDATA\_H

#include<QJsonObject>

#include<QJsonParseError>

#include<QFile>

#include<QJsonArray>

#include "graphicitem.h"

#include "mdichild.h"

#include "itembycolor.h"

#include "itembypicture.h"

class JsonData

{

private:

QJsonDocument doc;

QJsonParseError docError;

public:

JsonData();

void inFromFile(const QString &fWay, MdiChild \*Parent);

void outToFile(const QString &fWay, MdiChild \*Parent);

void outHistoryToFile(const QString &fWay, MdiChild \*Parent);

void outInitHistoryFile(const QString &fWay, MdiChild \*Parent);

void InitializeSceneFromFile(MdiChild \*Parent,QJsonArray &docArr);

void InitScenFromFile(const QString &fWay, MdiChild \*Parent, int Version);

void DeleteHighVersions(const QString &fWay, int Version);

};

#endif // JSONDATA\_H

А.22 Текст файлу jsondata.cpp

#include "jsondata.h"

JsonData::JsonData()

{

}

void JsonData::inFromFile(const QString &fWay, MdiChild \*Parent)

{

try{

QFile file;

file.setFileName(fWay);

if (!file.open(QIODevice::ReadOnly|QFile::Text))

{

throw QString("ERROR OPEN TO READ FILE: " + fWay);

}

doc = QJsonDocument::fromJson(QByteArray(file.readAll()),&docError);

file.close();

if(docError.errorString().toInt()!=QJsonParseError::NoError)

{

throw QString("ERROR JSON PARSE");

}

QJsonValue sizeScene = QJsonValue(doc.object().value("Size scene"));

int width = sizeScene.toObject().value("width").toInt();

int height = sizeScene.toObject().value("height").toInt();

Parent->SetSceneParametrs(width,height);

QJsonArray docArr = QJsonValue(doc.object().value("figures")).toArray();

InitializeSceneFromFile(Parent,docArr);

}

catch (QString error) {

qDebug()<<error;

}

}

void JsonData::outToFile(const QString &fWay, MdiChild \*Parent)

{

try {

if(Parent == NULL)

{

throw QString("ERROR PARENT == NULL");

}

QFile file;

file.setFileName(fWay);

if(!file.open(QIODevice::WriteOnly))

{

throw QString("ERROR NOT OPEN FILE "+fWay);

}

QJsonObject mainStruct;

QJsonObject sizeScene;

sizeScene.insert("width", Parent->scene->sceneRect().toRect().width());

sizeScene.insert("height", Parent->scene->sceneRect().toRect().height());

mainStruct.insert("Size scene", sizeScene);

QJsonArray recordsArray;

Parent->DeleteGrid();

QList<QGraphicsItem \*> itm = Parent->scene->items();

for(int i = 0; i < itm.size(); i++)

{

GraphicItem \*item = qgraphicsitem\_cast<GraphicItem\*>(itm[i]);

GraphicItemInfo Iformation = item->GetItemInfo();

QJsonObject recordObject;

recordObject.insert("Type", QJsonValue::fromVariant(Iformation.Type));

recordObject.insert("Value", QJsonValue::fromVariant(Iformation.Value));

recordObject.insert("Angle", QJsonValue::fromVariant(Iformation.Angle));

QJsonObject position;

position.insert("x",Iformation.X);

position.insert("y",Iformation.Y);

recordObject.insert("Position", position);

QJsonObject size;

size.insert("width",Iformation.Width);

size.insert("height",Iformation.Height);

recordObject.insert("Size", size);

recordsArray.push\_back(recordObject);

}

Parent->DrawGrid();

mainStruct.insert("figures",recordsArray);

QJsonDocument doc(mainStruct);

file.write("\n" + doc.toJson()+"\n");

file.close();

}

catch (QString error) {

qDebug()<<error;

}

}

void JsonData::outInitHistoryFile(const QString &fWay, MdiChild \*Parent)

{

if(Parent == NULL)

{

return;

}

QFile file;

file.setFileName(fWay);

if(file.open(QIODevice::WriteOnly))

{

QJsonObject mainStruct;

QJsonArray scenesArray;

for(int i = 0; i < 1; i++)

{

QJsonObject scene;

QJsonArray recordsArray;

Parent->DeleteGrid();

QList<QGraphicsItem \*> itm = Parent->scene->items();

for(int i = 0; i < itm.size(); i++)

{

GraphicItem \*item = qgraphicsitem\_cast<GraphicItem\*>(itm[i]);

GraphicItemInfo Iformation = item->GetItemInfo();

QJsonObject recordObject;

recordObject.insert("Type", QJsonValue::fromVariant(Iformation.Type));

recordObject.insert("Value", QJsonValue::fromVariant(Iformation.Value));

recordObject.insert("Angle", QJsonValue::fromVariant(Iformation.Angle));

QJsonObject position;

position.insert("x",Iformation.X);

position.insert("y",Iformation.Y);

recordObject.insert("Position", position);

QJsonObject size;

size.insert("width",Iformation.Width);

size.insert("height",Iformation.Height);

recordObject.insert("Size", size);

recordsArray.push\_back(recordObject);

}

Parent->DrawGrid();

scene.insert("figures", recordsArray);

scenesArray.push\_back(scene);

}

mainStruct.insert("Scenes", scenesArray);

QJsonDocument doc(mainStruct);

file.write("\n" + doc.toJson()+"\n");

file.close();

}

else

{

qDebug()<<"ERROR";

}

}

void JsonData::outHistoryToFile(const QString &fWay, MdiChild \*Parent)

{

QFile file;

file.setFileName(fWay);

if(file.open(QIODevice::ReadOnly|QFile::Text))

{

doc = QJsonDocument::fromJson(QByteArray(file.readAll()),&docError);

}

file.close();

if(docError.errorString().toInt()==QJsonParseError::NoError)

{

QJsonArray scenesArray = QJsonValue(doc.object().value("Scenes")).toArray();

QJsonObject scene;

QJsonArray recordsArray;

Parent->DeleteGrid();

QList<QGraphicsItem \*> itm = Parent->scene->items();

for(int i = 0; i < itm.size(); i++)

{

GraphicItem \*item = qgraphicsitem\_cast<GraphicItem\*>(itm[i]);

GraphicItemInfo Iformation = item->GetItemInfo();

QJsonObject recordObject;

recordObject.insert("Type", QJsonValue::fromVariant(Iformation.Type));

recordObject.insert("Value", QJsonValue::fromVariant(Iformation.Value));

recordObject.insert("Angle", QJsonValue::fromVariant(Iformation.Angle));

QJsonObject position;

position.insert("x",Iformation.X);

position.insert("y",Iformation.Y);

recordObject.insert("Position", position);

QJsonObject size;

size.insert("width",Iformation.Width);

size.insert("height",Iformation.Height);

recordObject.insert("Size", size);

recordsArray.push\_back(recordObject);

}

Parent->DrawGrid();

scene.insert("figures", recordsArray);

scenesArray.push\_back(scene);

QJsonObject mainStruct;

mainStruct.insert("Scenes", scenesArray);

QFile file;

file.setFileName(fWay);

if(file.open(QIODevice::WriteOnly))

{

QJsonDocument doc(mainStruct);

file.write("\n" + doc.toJson()+"\n");

file.close();

}

else

{

qDebug()<<"ERROR";

}

}

else

{

qDebug()<<"ERROR";

}

}

void JsonData::InitializeSceneFromFile(MdiChild \*Parent, QJsonArray &docArr)

{

for(int i = 0; i<docArr.count();i++)

{

GraphicItemInfo Information;

Information.Type = docArr.at(i).toObject().value("Type").toString();

Information.Value = docArr.at(i).toObject().value("Value").toString();

QJsonValue posVal = QJsonValue(docArr.at(i).toObject().value("Position"));

Information.X = posVal.toObject().value("x").toDouble();

Information.Y = posVal.toObject().value("y").toDouble();

QJsonValue sizeVal = QJsonValue(docArr.at(i).toObject().value("Size"));

Information.Width = sizeVal.toObject().value("width").toDouble();

Information.Height = sizeVal.toObject().value("height").toDouble();

Information.Angle = docArr.at(i).toObject().value("Angle").toDouble();

if(Information.Type == "Picture")

{

ItemByPicture \*item = new ItemByPicture(Information.Width, Information.Height, Information.Value, Parent);

item->setPos(Information.X, Information.Y);

item->setRotation(Information.Angle);

Parent->scene->addItem(item);

}

else

{

ItemByColor \*item = new ItemByColor(Information.Width, Information.Height, QColor(Information.Value), Parent);

item->setPos(Information.X, Information.Y);

item->setRotation(Information.Angle);

Parent->scene->addItem(item);

}

}

}

void JsonData::InitScenFromFile(const QString &fWay, MdiChild \*Parent, int Version)

{

QFile file;

file.setFileName(fWay);

if(file.open(QIODevice::ReadOnly|QFile::Text))

{

doc = QJsonDocument::fromJson(QByteArray(file.readAll()),&docError);

}

file.close();

if(docError.errorString().toInt()==QJsonParseError::NoError)

{

QJsonArray sceneArr = QJsonValue(doc.object().value("Scenes")).toArray();

QJsonArray docArr = QJsonValue( sceneArr.at(Version).toObject().value("figures")).toArray();

Parent->LastActive = NULL;

Parent->DeleteGrid();

Parent->scene->clear();

Parent->DrawGrid();

InitializeSceneFromFile(Parent,docArr);

}

}

void JsonData::DeleteHighVersions(const QString &fWay, int Version)

{

QFile file;

file.setFileName(fWay);

if(file.open(QIODevice::ReadWrite|QFile::Text))

{

doc = QJsonDocument::fromJson(QByteArray(file.readAll()),&docError);

}

file.close();

if(docError.errorString().toInt()==QJsonParseError::NoError)

{

QJsonArray sceneArr = QJsonValue(doc.object().value("Scenes")).toArray();

for(int j = 0; j<sceneArr.count();j++)

{

while(Version+1 < sceneArr.size()){

sceneArr.removeLast();

}

}

QJsonObject mainStruct;

mainStruct.insert("Scenes",sceneArr);

if(file.open(QIODevice::ReadWrite|QFile::Text))

{

QJsonDocument doc(mainStruct);

file.resize(0);

file.write("\n" + doc.toJson()+"\n");

}

file.close();

}

}

А.23 Текст файлу mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QSettings>

#include <QTranslator>

#include <QtWidgets>

#include "ui\_mainwindow.h"

#include "jsondata.h"

#include "aboutwindow.h"

#include "mdichild.h"

#include "graphicitem.h"

#include "itembycolor.h"

#include "itembypicture.h"

#include "fileopen.h"

#include "itemeditwindow.h"

#include "formcreate.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~MainWindow();

MdiChild \*activeMdiChild() const;

void loadProject(QWidget \*widget);

DataBase database;

int uniqueProjectID = 1;

private slots:

void on\_actionNew\_triggered();

void on\_actionOpen\_triggered();

void on\_actionSave\_triggered();

void on\_actionSaveAs\_triggered();

void on\_actionClose\_triggered();

void on\_actionHelp\_triggered();

void on\_actionAbout\_triggered();

void on\_listViewBuildTools\_clicked(const QModelIndex &index);

void on\_listViewFurnitureTools\_clicked(const QModelIndex &index);

void on\_listViewDecorTools\_clicked(const QModelIndex &index);

void on\_actionUk\_triggered();

void on\_actionEn\_triggered();

void on\_actionRu\_triggered();

void on\_actionUndo\_triggered();

void on\_actionRedo\_triggered();

void on\_actionDelete\_triggered();

void on\_actionClear\_triggered();

void on\_actionZoomIn\_triggered();

void on\_actionZoomOut\_triggered();

void on\_actionSetItemSize\_triggered();

private:

void saveSettings();

QString loadSettings();

Ui::MainWindow \*ui;

QStandardItemModel \*modelBuildTools;

QSortFilterProxyModel \*proxyModelBuildTools;

QStandardItemModel \*modelFurnitureTools;

QSortFilterProxyModel \*proxyModelFurnitureTools;

QStandardItemModel \*modelDecorTools;

QSortFilterProxyModel \*proxyModelDecorTools;

int FilesCount = 0;

void setupOtherUi();

void setupBuildTools();

void setupFurnitureTools();

void setupDecorTools();

QSettings\*settings;

QTranslator\* translator;

void changeTranslator(QString postfix);

void changeEvent(QEvent \*event);

};

#endif // MAINWINDOW\_H

А.24 Текст файлу mainwindow.cpp

#include "mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

setupOtherUi();

database.connectToDataBase();

settings = new QSettings("settings.ini", QSettings::IniFormat, this);

changeTranslator(loadSettings());

}

MainWindow::~MainWindow()

{

saveSettings();

delete settings;

delete ui;

}

void MainWindow::setupOtherUi()

{

setupBuildTools();

setupFurnitureTools();

setupDecorTools();

}

void MainWindow::saveSettings()

{

if(ui->actionRu->isChecked())

{

settings->setValue("language","ru");

}

else if(ui->actionEn->isChecked())

{

settings->setValue("language","en");

}

else

{

settings->setValue("language","uk");

}

settings->setValue("geometry",geometry());

}

QString MainWindow::loadSettings()

{

setGeometry(settings->value("geometry",QRect(50,50,700,460)).toRect());

return settings->value("language","uk").toString();

}

void MainWindow::setupBuildTools()

{

modelBuildTools = new QStandardItemModel(this);

proxyModelBuildTools = new QSortFilterProxyModel(this);

modelBuildTools->insertColumn(0);

int numRows = 4;

modelBuildTools->insertRows(0, numRows);

modelBuildTools->setData(modelBuildTools->index(0, 0), QPixmap(":/list/build/wall"), Qt::DecorationRole);

modelBuildTools->setData(modelBuildTools->index(0, 0), "Стіна", Qt::EditRole);

modelBuildTools->setData(modelBuildTools->index(1, 0), QPixmap(":/list/build/window"), Qt::DecorationRole);

modelBuildTools->setData(modelBuildTools->index(1, 0), "Вікно", Qt::EditRole);

modelBuildTools->setData(modelBuildTools->index(2, 0), QPixmap(":/list/build/door"), Qt::DecorationRole);

modelBuildTools->setData(modelBuildTools->index(2, 0), "Двері", Qt::EditRole);

modelBuildTools->setData(modelBuildTools->index(3, 0), QPixmap(":/list/build/door-garage"), Qt::DecorationRole);

modelBuildTools->setData(modelBuildTools->index(3, 0), "Гаражні\nдвері", Qt::EditRole);

proxyModelBuildTools->setSourceModel(modelBuildTools);

proxyModelBuildTools->setFilterKeyColumn(0);

ui->listViewBuildTools->setModel(modelBuildTools);

}

void MainWindow::setupFurnitureTools()

{

modelFurnitureTools = new QStandardItemModel(this);

proxyModelFurnitureTools = new QSortFilterProxyModel(this);

modelFurnitureTools->insertColumn(0);

int numRows = 13;

modelFurnitureTools->insertRows(0, numRows);

modelFurnitureTools->setData(modelFurnitureTools->index(0, 0), QPixmap(":/list/furniture/sofa"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(0, 0), "Диван", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(1, 0), QPixmap(":/list/furniture/chair"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(1, 0), "Стілець", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(2, 0), QPixmap(":/list/furniture/bed-single"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(2, 0), "Ліжко", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(3, 0), QPixmap(":/list/furniture/bed-double"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(3, 0), "Двомісне\nліжко", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(4, 0), QPixmap(":/list/furniture/bedside"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(4, 0), "Тумбочка", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(5, 0), QPixmap(":/list/furniture/table-dining"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(5, 0), "Кухонний\nстіл", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(6, 0), QPixmap(":/list/furniture/table-rect"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(6, 0), "Квадратний\nстіл", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(7, 0), QPixmap(":/list/furniture/table-work"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(7, 0), "Робочий\nстіл", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(8, 0), QPixmap(":/list/furniture/table-glass"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(8, 0), "Скляний\nстіл", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(9, 0), QPixmap(":/list/furniture/table-wood"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(9, 0), "Диванний\nстолик", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(10, 0), QPixmap(":/list/furniture/oven"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(10, 0), "Духовка", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(11, 0), QPixmap(":/list/furniture/bath"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(11, 0), "Ванна", Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(12, 0), QPixmap(":/list/furniture/toilet"), Qt::DecorationRole);

modelFurnitureTools->setData(modelFurnitureTools->index(12, 0), "Туалет", Qt::EditRole);

proxyModelFurnitureTools->setSourceModel(modelFurnitureTools);

proxyModelFurnitureTools->setFilterKeyColumn(0);

ui->listViewFurnitureTools->setModel(modelFurnitureTools);

}

void MainWindow::setupDecorTools()

{

modelDecorTools = new QStandardItemModel(this);

proxyModelDecorTools = new QSortFilterProxyModel(this);

modelDecorTools->insertColumn(0);

int numRows = 2;

modelDecorTools->insertRows(0, numRows);

modelDecorTools->setData(modelDecorTools->index(0, 0), QPixmap(":/list/decor/plant"), Qt::DecorationRole);

modelDecorTools->setData(modelDecorTools->index(0, 0), "Рослина", Qt::EditRole);

modelDecorTools->setData(modelDecorTools->index(1, 0), QPixmap(":/list/decor/car"), Qt::DecorationRole);

modelDecorTools->setData(modelDecorTools->index(1, 0), "Автомобіль", Qt::EditRole);

proxyModelDecorTools->setSourceModel(modelDecorTools);

proxyModelDecorTools->setFilterKeyColumn(0);

ui->listViewDecorTools->setModel(modelDecorTools);

}

MdiChild \*MainWindow::activeMdiChild() const

{

if (QMdiSubWindow \*activeSubWindow = ui->mdiArea->activeSubWindow())

return qobject\_cast<MdiChild \*>(activeSubWindow->widget());

return 0;

}

void MainWindow::on\_actionNew\_triggered()

{

FormCreate \*Window = new FormCreate(this);

Window->show();

}

void MainWindow::on\_actionOpen\_triggered()

{

FileOpen OpenFile(this);

OpenFile.OpenFile();

}

void MainWindow::on\_actionSave\_triggered()

{

if(activeMdiChild() != NULL)

{

activeMdiChild()->SaveFileInfo.Save(this);

}

}

void MainWindow::on\_actionSaveAs\_triggered()

{

if(activeMdiChild() != NULL)

{

activeMdiChild()->SaveFileInfo.SaveAs(this);

}

}

void MainWindow::on\_actionClose\_triggered()

{

close();

}

void MainWindow::on\_actionHelp\_triggered()

{

}

void MainWindow::on\_actionAbout\_triggered()

{

AboutWindow \*about = new AboutWindow();

about->show();

}

void MainWindow::loadProject(QWidget \*widget)

{

auto window = ui->mdiArea->addSubWindow(widget);

if (ui->mdiArea->subWindowList().size() == 1) window->showMaximized();

else window->show();

}

void MainWindow::on\_listViewBuildTools\_clicked(const QModelIndex &index)

{

MdiChild \*activeChild = activeMdiChild();

if (!activeChild)

return;

int i = index.row();

switch (i) {

case 0:

activeChild->addItem(Qt::darkGray, 500, 30);

break;

case 1:

activeChild->addItem(QColor("#87CEFA"), 100, 30);

break;

case 2:

activeChild->addItem(QColor("#CB6843"), 80, 30);

break;

case 3:

activeChild->addItem(QColor("#202020"), 250, 30);

break;

}

}

void MainWindow::on\_listViewFurnitureTools\_clicked(const QModelIndex &index)

{

MdiChild \*activeChild = activeMdiChild();

if (!activeChild)

return;

int i = index.row();

switch (i) {

case 0:

activeChild->addItem(QString(":scene/sofa"), 200, 70);

break;

case 1:

activeChild->addItem(QString(":scene/chair"), 50, 60);

break;

case 2:

activeChild->addItem(QString(":scene/bed-single"), 90, 200);

break;

case 3:

activeChild->addItem(QString(":scene/bed-double"), 150, 200);

break;

case 4:

activeChild->addItem(QString(":scene/bedside"), 50, 50);

break;

case 5:

activeChild->addItem(QString(":scene/table-dining"), 180, 180);

break;

case 6:

activeChild->addItem(QString(":scene/table-rect"), 140, 140);

break;

case 7:

activeChild->addItem(QString(":scene/table-work"), 150, 90);

break;

case 8:

activeChild->addItem(QString(":scene/table-glass"), 150, 90);

break;

case 9:

activeChild->addItem(QString(":scene/table-wood"), 150, 70);

break;

case 10:

activeChild->addItem(QString(":scene/oven"), 70, 70);

break;

case 11:

activeChild->addItem(QString(":scene/bath"), 190, 80);

break;

case 12:

activeChild->addItem(QString(":scene/toilet"), 50, 80);

break;

}

}

void MainWindow::on\_listViewDecorTools\_clicked(const QModelIndex &index)

{

MdiChild \*activeChild = activeMdiChild();

if (!activeChild)

return;

int i = index.row();

switch (i) {

case 0:

activeChild->addItem(QString(":/scene/plant"), 50, 50);

break;

case 1:

activeChild->addItem(QString(":/scene/car"), 150, 350);

break;

}

}

void MainWindow::on\_actionUk\_triggered()

{

changeTranslator("uk");

}

void MainWindow::on\_actionEn\_triggered()

{

changeTranslator("en");

}

void MainWindow::on\_actionRu\_triggered()

{

changeTranslator("ru");

}

void MainWindow::changeTranslator(QString postfix)

{

if(postfix=="uk")

{

ui->actionEn->setChecked(false);

ui->actionRu->setChecked(false);

ui->actionUk->setChecked(true);

}

else if(postfix=="en")

{

ui->actionEn->setChecked(true);

ui->actionRu->setChecked(false);

ui->actionUk->setChecked(false);

}

else

{

ui->actionEn->setChecked(false);

ui->actionRu->setChecked(true);

ui->actionUk->setChecked(false);

}

QApplication::removeTranslator(translator);

translator = new QTranslator(this);

translator->load("Core\_Planner\_" + postfix);

QApplication::installTranslator(translator);

}

void MainWindow::changeEvent(QEvent \*event)

{

if (event->type() == QEvent::LanguageChange)

{

ui->retranslateUi(this);

modelBuildTools->setData(modelBuildTools->index(0, 0), tr("Стіна"), Qt::EditRole);

modelBuildTools->setData(modelBuildTools->index(1, 0), tr("Вікно"), Qt::EditRole);

modelBuildTools->setData(modelBuildTools->index(2, 0), tr("Двері"), Qt::EditRole);

modelBuildTools->setData(modelBuildTools->index(3, 0), tr("Гаражні\nдвері"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(0, 0), tr("Диван"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(1, 0), tr("Стілець"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(2, 0), tr("Ліжко"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(3, 0), tr("Двомісне\nліжко"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(4, 0), tr("Тумбочка"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(5, 0), tr("Кухонний\nстіл"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(6, 0), tr("Квадратний\nстіл"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(7, 0), tr("Робочий\nстіл"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(8, 0), tr("Скляний\nстіл"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(9, 0), tr("Диванний\nстолик"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(10, 0), tr("Духовка"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(11, 0), tr("Ванна"), Qt::EditRole);

modelFurnitureTools->setData(modelFurnitureTools->index(12, 0), tr("Туалет"), Qt::EditRole);

modelDecorTools->setData(modelDecorTools->index(0, 0), tr("Рослина"), Qt::EditRole);

modelDecorTools->setData(modelDecorTools->index(1, 0), tr("Автомобіль"), Qt::EditRole);

}

else

{

QMainWindow::changeEvent(event);

}

}

void MainWindow::on\_actionUndo\_triggered()

{

if(activeMdiChild() != NULL)

{

if(activeMdiChild()->VersionOfScene > 0)

{

activeMdiChild()->AfterUndo = true;

activeMdiChild()->VersionOfScene--;

activeMdiChild()->SaveFileInfo.LoadFromHistory(activeMdiChild()->VersionOfScene);

}

}

}

void MainWindow::on\_actionRedo\_triggered()

{

if(activeMdiChild() != NULL)

{

if(activeMdiChild()->VersionOfScene < activeMdiChild()->MaximumVersionOfScene)

{

activeMdiChild()->VersionOfScene++;

activeMdiChild()->SaveFileInfo.LoadFromHistory(activeMdiChild()->VersionOfScene);

}

}

}

void MainWindow::on\_actionDelete\_triggered()

{

if(activeMdiChild() != NULL && activeMdiChild()->LastActive != NULL)

{

delete activeMdiChild()->LastActive;

activeMdiChild()->LastActive = NULL;

if(activeMdiChild()->AfterUndo)

{

activeMdiChild()->SaveFileInfo.DeleteHighVersions(activeMdiChild()->VersionOfScene);

activeMdiChild()->AfterUndo = false;

activeMdiChild()->MaximumVersionOfScene = activeMdiChild()->VersionOfScene;

}

activeMdiChild()->VersionOfScene++;

activeMdiChild()->MaximumVersionOfScene++;

activeMdiChild()->SaveFileInfo.SaveToHistory();

activeMdiChild()->SaveFileInfo.LoadFromHistory(activeMdiChild()->VersionOfScene);

}

}

void MainWindow::on\_actionClear\_triggered()

{

if(activeMdiChild() != NULL)

{

activeMdiChild()->DeleteGrid();

activeMdiChild()->scene->clear();

activeMdiChild()->DrawGrid();

if(activeMdiChild()->AfterUndo)

{

activeMdiChild()->SaveFileInfo.DeleteHighVersions(activeMdiChild()->VersionOfScene);

activeMdiChild()->AfterUndo = false;

activeMdiChild()->MaximumVersionOfScene = activeMdiChild()->VersionOfScene;

}

activeMdiChild()->VersionOfScene++;

activeMdiChild()->MaximumVersionOfScene++;

activeMdiChild()->SaveFileInfo.SaveToHistory();

activeMdiChild()->SaveFileInfo.LoadFromHistory(activeMdiChild()->VersionOfScene);

}

}

void MainWindow::on\_actionZoomIn\_triggered()

{

if(activeMdiChild() != NULL) activeMdiChild()->setScaleToView(1.5, 1.5);

}

void MainWindow::on\_actionZoomOut\_triggered()

{

if(activeMdiChild() != NULL) activeMdiChild()->setScaleToView(0.75, 0.75);

}

void MainWindow::on\_actionSetItemSize\_triggered()

{

if(activeMdiChild() != NULL && activeMdiChild()->LastActive != NULL)

{

ItemEditWindow \*window = new ItemEditWindow(this, activeMdiChild());

window->show();

}

}

А.25 Текст файлу mdichild.h

#ifndef MDICHILD\_H

#define MDICHILD\_H

#include <QtWidgets>

#include "ui\_mdichild.h"

#include "graphicitem.h"

#include "itembypicture.h"

#include "itembycolor.h"

#include "filesave.h"

namespace Ui {

class MdiChild;

}

class FileSave;

class MdiChild : public QWidget

{

Q\_OBJECT

public:

explicit MdiChild(MainWindow \*Main, QWidget \*parent = 0);

~MdiChild();

GraphicItem \*LastActive = 0;

QGraphicsScene \*scene;

void addItem(QString Name, double setWidth, double setHeight);

void addItem(QColor ItemColor, double setWidth, double setHeight);

void UpdateScene();

void SaveAsImage(QString NameOfFile);

void setScaleToView(double X, double Y);

void DrawGrid();

void DeleteGrid();

void SetSceneParametrs(int setWidth, int setHeight);

FileSave SaveFileInfo;

int VersionOfScene;

int MaximumVersionOfScene;

bool AfterUndo;

private:

Ui::MdiChild \*ui;

MainWindow \*MainWind;

int Width, Height;

QVector<QGraphicsLineItem\*> HorizontalLines;

QVector<QGraphicsLineItem\*> VerticalLines;

void DrawHorizontalLines();

void DrawVerticalLines();

};

#endif // MDICHILD\_H

А.26 Текст файлу mdichild.cpp

#include "mdichild.h"

MdiChild::MdiChild(MainWindow \*Main, QWidget \*parent) :

QWidget(parent),

ui(new Ui::MdiChild)

{

ui->setupUi(this);

MainWind = Main;

Width = 2000; Height = 1000;

scene = new QGraphicsScene(this);

scene->setItemIndexMethod(QGraphicsScene::NoIndex);

scene->setSceneRect(0,0,Width,Height);

ui->graphicsView->setScene(scene);

SaveFileInfo.ParenWithScene = this;

SaveFileInfo.FileName = this->windowTitle();

DrawGrid();

SaveFileInfo.InitHistory();

VersionOfScene = 0;

MaximumVersionOfScene = 0;

AfterUndo = false;

}

void MdiChild::SetSceneParametrs(int setWidth, int setHeight)

{

DeleteGrid();

Width = setWidth;

Height = setHeight;

scene->setSceneRect(0,0,Width,Height);

DrawGrid();

}

MdiChild::~MdiChild()

{

QString Message = tr("Зберегти проект ");

Message += SaveFileInfo.FileName;

Message += tr(" перед закриттям?");

if(QMessageBox::Yes == QMessageBox::question(this, tr("Увага"), Message))

{

SaveFileInfo.Save(MainWind);

}

delete ui;

}

void MdiChild::addItem(QString Name, double setWidth, double setHeight)

{

if(AfterUndo)

{

SaveFileInfo.DeleteHighVersions(VersionOfScene);

AfterUndo = false;

MaximumVersionOfScene = VersionOfScene;

}

ItemByPicture \*item = new ItemByPicture(setWidth, setHeight, Name, this);

item->setPos(scene->sceneRect().toRect().width()/2, scene->sceneRect().toRect().height()/2);

scene->addItem(item);

VersionOfScene++;

MaximumVersionOfScene++;

SaveFileInfo.SaveToHistory();

SaveFileInfo.LoadFromHistory(VersionOfScene);

}

void MdiChild::addItem(QColor ItemColor, double setWidth, double setHeight)

{

if(AfterUndo)

{

SaveFileInfo.DeleteHighVersions(VersionOfScene);

AfterUndo = false;

MaximumVersionOfScene = VersionOfScene;

}

ItemByColor \*item = new ItemByColor(setWidth, setHeight, ItemColor, this);

item->setPos(scene->sceneRect().toRect().width()/2, scene->sceneRect().toRect().height()/2);

scene->addItem(item);

VersionOfScene++;

MaximumVersionOfScene++;

SaveFileInfo.SaveToHistory();

SaveFileInfo.LoadFromHistory(VersionOfScene);

}

void MdiChild::UpdateScene()

{

scene->update();

}

void MdiChild::DrawHorizontalLines()

{

for(int i = 0; i <= Height; i+=10)

{

QGraphicsLineItem \*HLine = new QGraphicsLineItem();

HLine->setLine(0,i,Width,i);

if(i%100 == 0) HLine->setPen(QPen(Qt::darkGray));else HLine->setPen(QPen(Qt::lightGray));

HorizontalLines.push\_back(HLine);

scene->addItem(HLine);

}

}

void MdiChild::DrawVerticalLines()

{

for(int i = 0; i <= Width; i+=10)

{

QGraphicsLineItem \*VLine = new QGraphicsLineItem();

VLine->setLine(i,0,i,Height);

if(i%100 == 0) VLine->setPen(QPen(Qt::darkGray));else VLine->setPen(QPen(Qt::lightGray));

VerticalLines.push\_back(VLine);

scene->addItem(VLine);

}

}

void MdiChild::DrawGrid()

{

if(VerticalLines.size() == 0) DrawVerticalLines();

if(HorizontalLines.size() == 0) DrawHorizontalLines();

}

void MdiChild::DeleteGrid()

{

while(HorizontalLines.size() > 0)

{

delete HorizontalLines.back();

HorizontalLines.pop\_back();

}

while(VerticalLines.size() > 0)

{

delete VerticalLines.back();

VerticalLines.pop\_back();

}

}

void MdiChild::SaveAsImage(QString NameOfFile)

{

DeleteGrid();

QImage image(int(scene->sceneRect().toRect().width()), int(scene->sceneRect().toRect().height()), QImage::Format\_ARGB32);

QPainter painter(&image);

painter.setRenderHint(QPainter::Antialiasing);

scene->render(&painter);

image.save(NameOfFile);

DrawGrid();

}

void MdiChild::setScaleToView(double X, double Y)

{

ui->graphicsView->scale(X, Y);

}

А.27 Текст файлу points.h

#ifndef POINTS\_H

#define POINTS\_H

#include <QPointF>

class **points**

{

private:

QPointF TopMinimumCenter;

QPointF LeftMinimumCenter;

QPointF RightMinimumCenter;

QPointF BottomMinimumCenter;

public:

**points**();

void **SetTopMinimumCenter**(QPointF point) {TopMinimumCenter = point;}

void **SetLeftMinimumCenter**(QPointF point) {LeftMinimumCenter = point;}

void **SetRightMinimumCenter**(QPointF point) {RightMinimumCenter = point;}

void **SetBottomMinimumCenter**(QPointF point) {BottomMinimumCenter = point;}

QPointF **GetTopMinimumCenter**() {return TopMinimumCenter;}

QPointF **GetLeftMinimumCenter**() {return LeftMinimumCenter;}

QPointF **GetRightMinimumCenter**() {return RightMinimumCenter;}

QPointF **GetBottomMinimumCenter**() {return BottomMinimumCenter;}

};

#endif // POINTS\_H

Додаток Б  
Інтерфейс програми

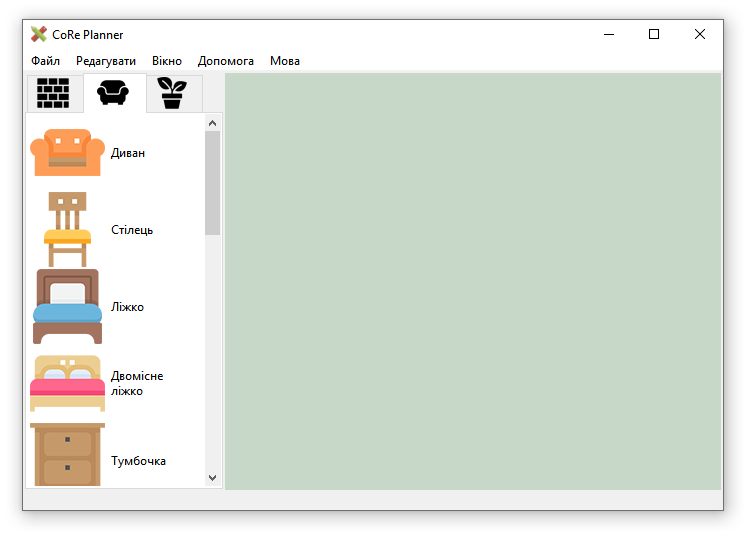


Рисунок Б.1 – Головне вікно програми

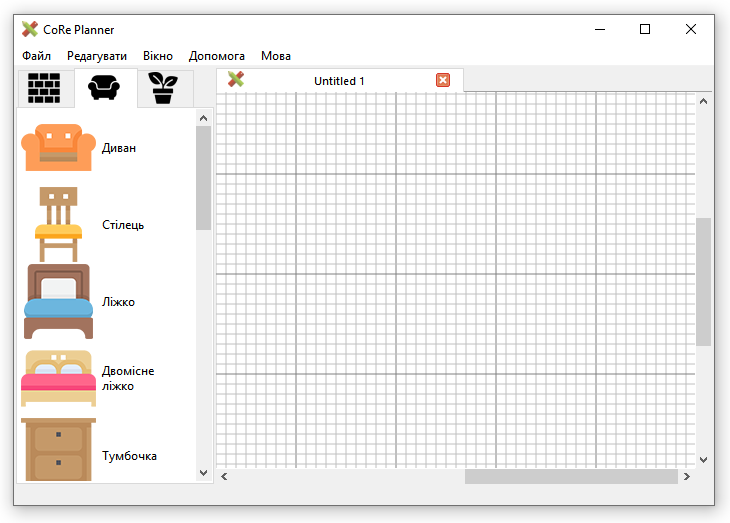


Рисунок Б.2 – Головне вікно програми з відкритим пустим проектом

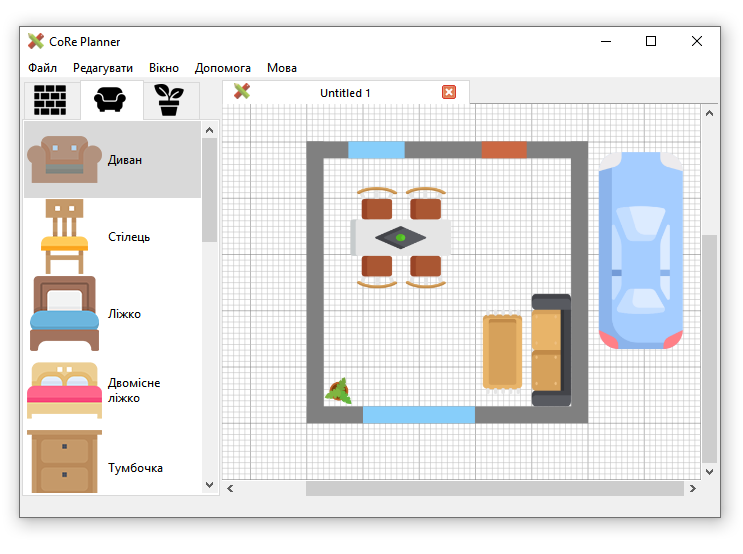


Рисунок Б.3 – Головне вікно програми з проектом у роботі

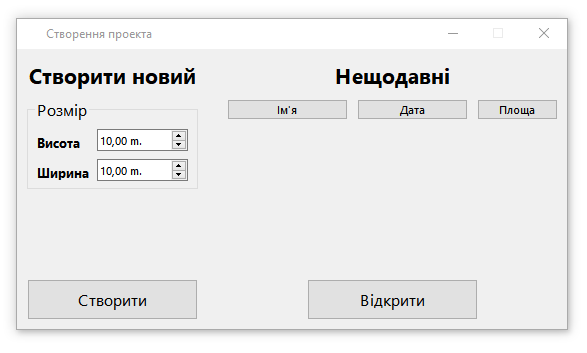


Рисунок Б.4 – Вікно створення нового проекту або відкриття нещодавнього

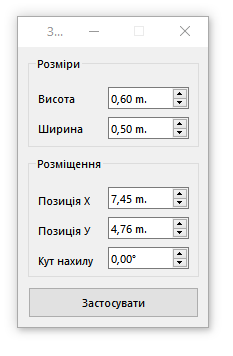


Рисунок Б.5 – Вікно редагування властивостей об’єктів

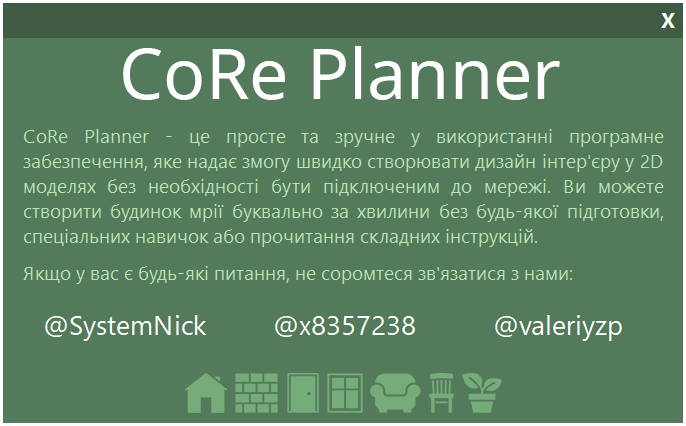


Рисунок Б.6 – Вікно з інформацією про програму та розробників

Додаток В  
Слайди презентації



Рисунок В.1 – Тема курсового проекту

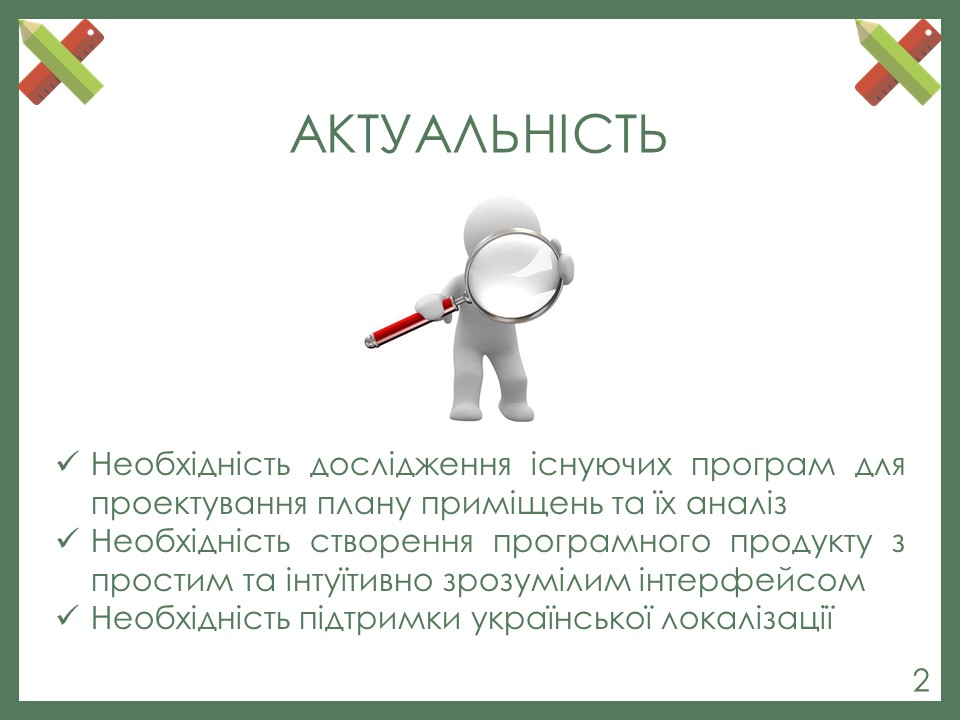


Рисунок В.2 – Актуальність



Рисунок В.3 – Мета та завдання курсової роботи

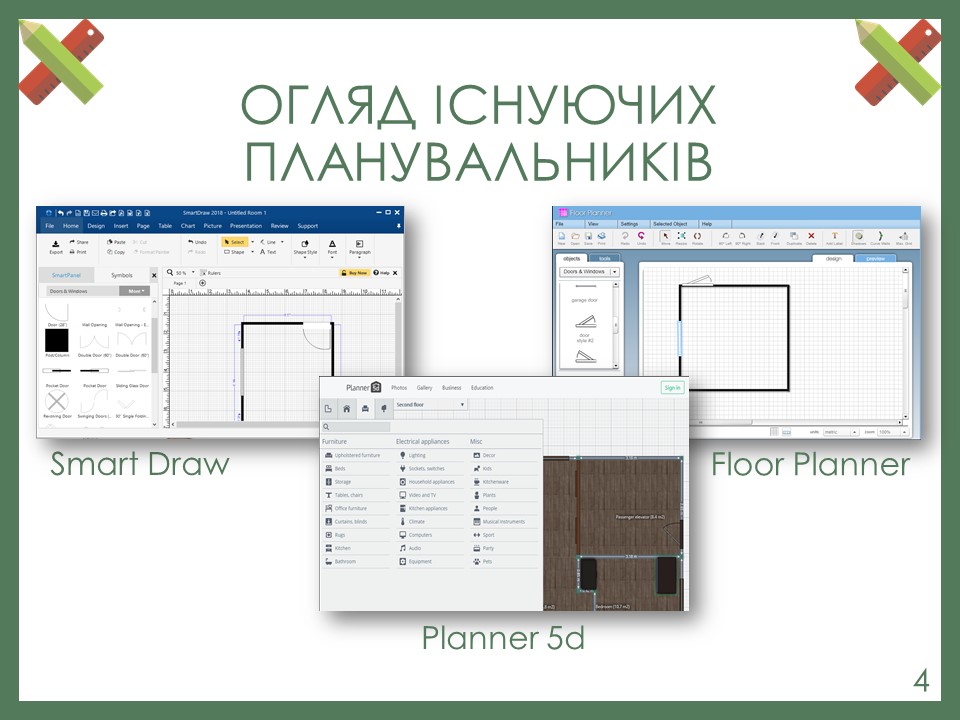


Рисунок В.4 – Огляд існуючих аналогів



Рисунок В.5 – Розроблені алгоритми



Рисунок В.6 – Сортування бульбашкою



Рисунок В.7 – Сортування вставками



Рисунок В.8 – Сортування методом Шелла

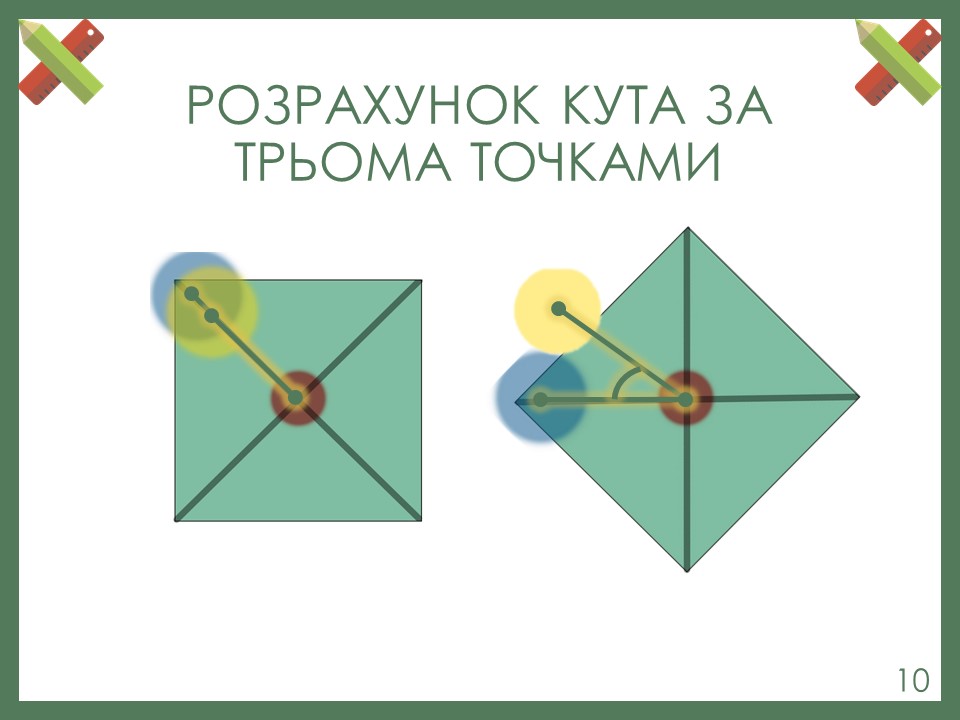


Рисунок В.9 – Розрахування кута за трьома точками



Рисунок В.10 – Робота з класами



Рисунок В.11 – Робота з даними json

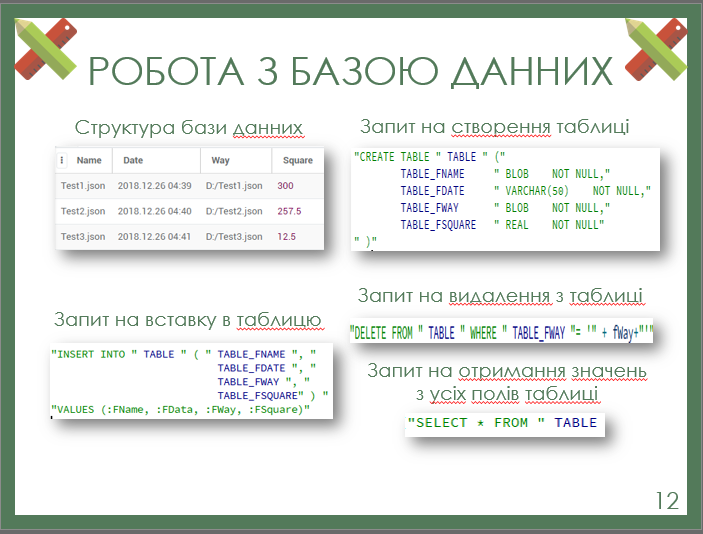


Рисунок В.12 – Робота з базою даних

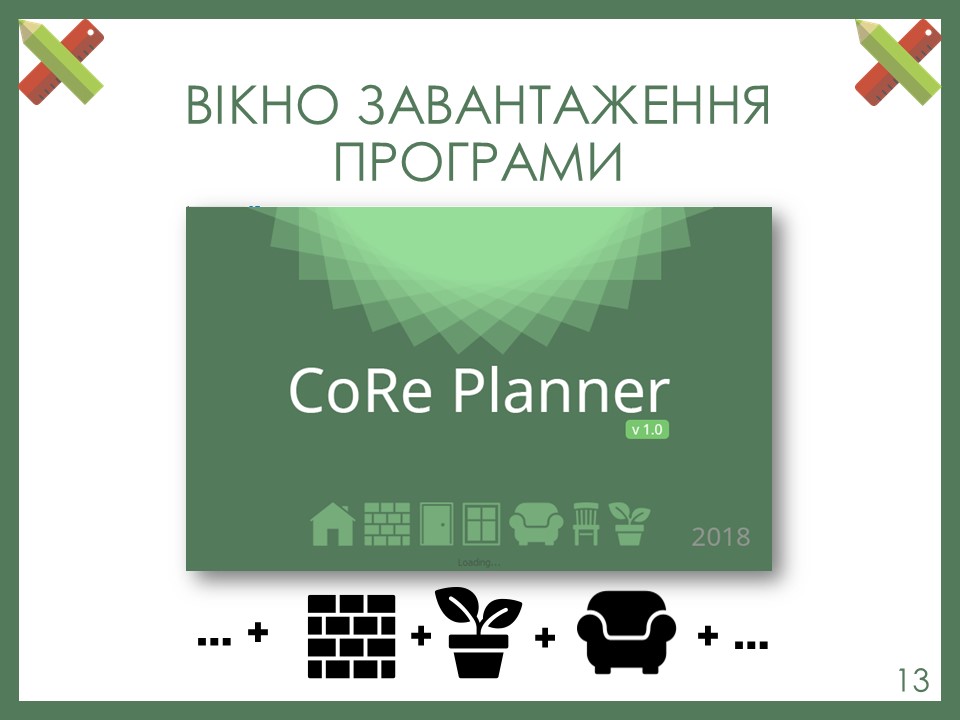


Рисунок В.13 – Вікно завантаження програми

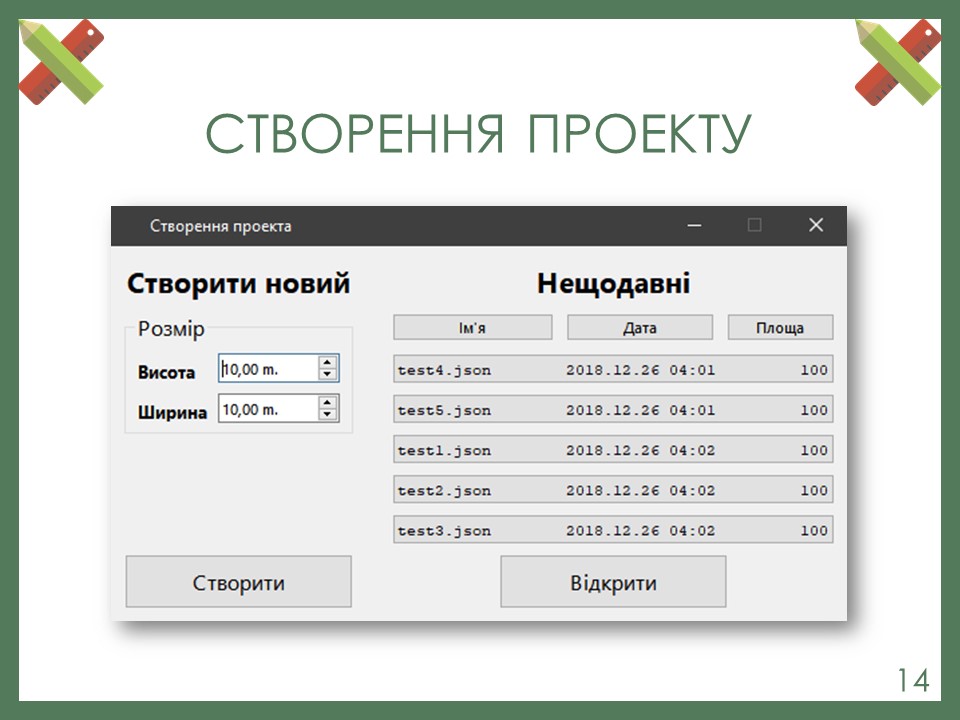


Рисунок В.14 – Створення проекту

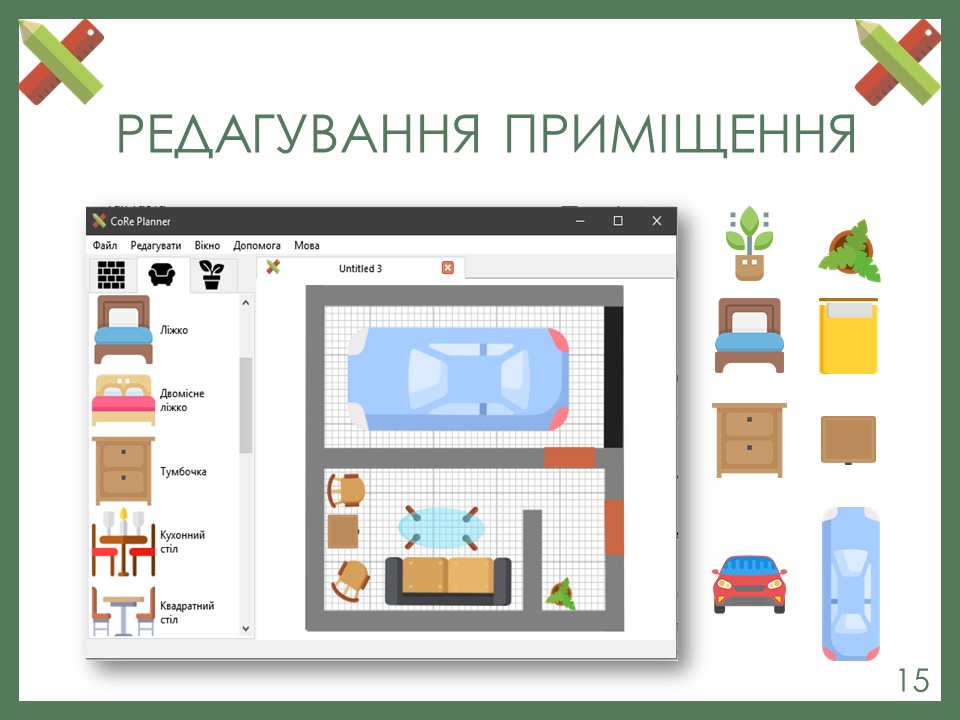


Рисунок В.15 – Редагування приміщення

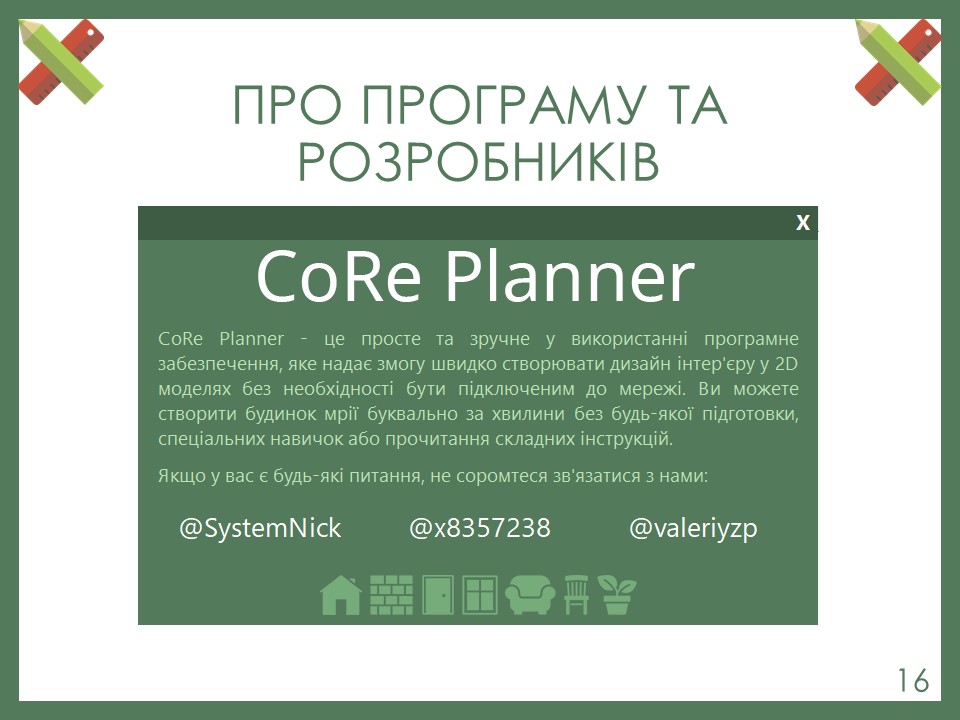


Рисунок В.16 – Про програму та розробників

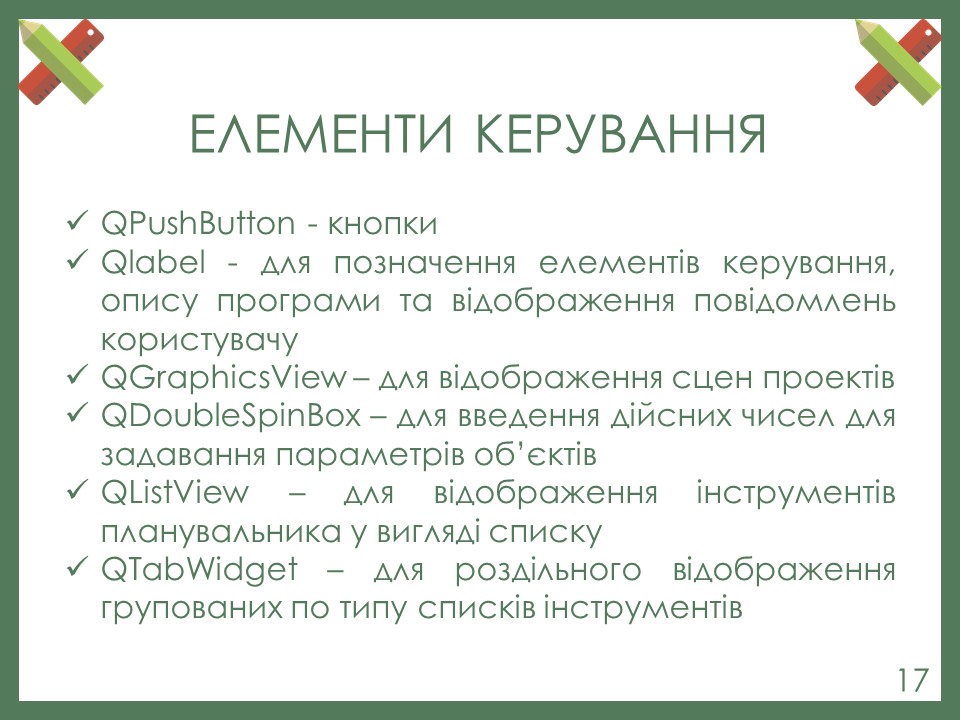


Рисунок В.17 – Елементи керування



Рисунок В.18 – Результати роботи



Рисунок В.19 – Фінальний слайд