НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра автоматизованих систем обробки інформації і управління

До захисту допущено:
В.о. завідувачакафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2021р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення комп'ютеризованих систем»
спеціальності 121 «Інженерія програмного забезпечення»

на тему: «*Програмне забезпечення для планування інтер'єру у 3D»*

Виконав: *студент IV курсу, групи ІП-з71*

Богдан Валентин Вікторович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *ас. Іванова Лідія Андріївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук Катерина Ігорівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут ім. І.Сікорського”

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра автоматизованих систем обробки інформації і управління
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення комп’ютеризованих систем*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“___” _____ 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Богдану Валентину Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Програмне забезпечення для планування інтер’єру у 3D*»
керівник проєкту *Іванова Лідія Андріївна, ассистент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «11» *травня* 2021 р. №1139-с

2. Термін подання студентом проєкту «07» *червня 2021 року*

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення,

архітектура прыограмного забезпечення

3) Аналіз якості та тестування програмного забезпечення

4) Впровадження та супровід програмного забезпечення

5. Перелік графічного матеріалу

1) Схема структурна варіантів використання

2) Схема бізнес-процесу редагування

3) Схема структурна класів програмного забезпечення

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «14» *березня 2021 року*

КАЛЕНДАРНИЙ ПЛАН

з/п	**Назва етапів виконання дипломного проєкту**	**Термін виконання етапів проєкту**	**Примітка**
1.	*Вивчення рекомендованої літератури*	*05.04.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*10.04.2021*	
3.	*Постановка та формалізація задачі*	*12.04.2021*	
4.	*Аналіз вимог до програмного забезпечення*	*15.04.2021*	
5.	*Алгоритмізація задачі*	*17.04.2021*	
6.	*Моделювання програмного забезпечення*	*18.04.2021*	
7.	*Обґрунтування використовуваних технічних засобів*	*20.04.2021*	
8.	*Розробка архітектури програмного забезпечення*	*21.04.2021*	
9.	*Розробка програмного забезпечення*	*26.04.2021*	
10.	*Налагодження програми*	*30.04.2021*	
11.	*Виконання графічних документів*	*03.05.2021*	
12.	*Оформлення пояснювальної записки*	*03.05.2021*	
13.	*Подання ДП на попередній захист*	*24.05.2021*	
14.	*Подання ДП рецензенту*	*06.06.2021*	
15.	*Подання ДП на основний захист*	*08.06.2021*	

Студент _____ Валентин Богдан

(підпис)

Керівник _____ Лідія Іванова
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
	4		Завдання на дипломний проект	2	
	4	КПІ.ІП-з7109.045490.02.81	Пояснювальна записка	50	
	4	КПІ.ІП-з7109.045490.03.91	Технічне завдання	10	
	4	КПІ.ІП-з7109.045490.04.13	Опис програми	38	
	3	КПІ.ІП-з7109.045490.05.51	Програма та методика тестування	6	
	3	КПІ.ІП-з7109.045490.06.34	Керівництво користувача	4	
	3	КПІ.ІП-з7109.045490.07.99.СС	Схема структурна варіантів використання	1	
	3	КПІ.ІП-з7109.045490.07.99.СБП	Схема бізнес-процесу редагування	1	
	3	КПІ.ІП-з7109.045490.07.99.СС	Схема структурна класів програмного забезпечення	1	

Зм.	Арк.	ПІБ	Підп.	Дата				
Розробн.		Богдан В.В.			Відомість дипломного проєкту	ім.	ист	Листів
Керівн.		Іванова Л.А.						1
Консульт.						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІПз-71		
Н/контр.		Ліщук К.І.						
В.о.зав.каф.		Павлов О.А.						

АНОТАЦІЯ

Пояснювальна записка складається з чотирьох розділів, містить 16 рисунків, 29 таблиць, 2 додатки та 11 джерел — загалом 145 сторінок. Об'єкт дослідження — програмне забезпечення для планування інтер’єру у 3D.

Мета дипломного проєкту — спростити проектування інтер’єру для користувачів-початківців за допомогою створення програмного забезпечення із спрощеним досвідом користувача.

У першому розділі проведено проаналізовано існуючі рішення та створено функціональні та нефункціональні вимоги до розроблюваного рішення.

У другому розділі було описано архітектуру компонентів програмного забезпечення і технологічні рішення для конструювання програмного забезпечення.

У третьому розділі описано та проведено процеси автоматизованого та ручного тестування програмного забезпечення, описано контрольні приклади тестувань.

У четвертому розділі описано процеси розгортання, впровадження та способи користування програмним забезпеченням.

КЛЮЧОВІ СЛОВА: КРОСПЛАТФОРМЕНИЙ ЗАСТОСУНОК, ПЛАНУВАННЯ ІНТЕР’ЄРУ, 3D.

ABSTRACT

The explanatory note consists of four sections, contains 16 figures, 29 tables, 2 additional elements and 11 sources - a total of 145 pages. The object of study - software for interior planning in 3D.

The purpose of the thesis project is to design interior design for novice users by creating software with user messages.

In the first section the existing solutions are analyzed and functional and non-functional requirements to the developed solution are created.

Another section described the architecture of software components and technological solutions for software design.

The third section describes and conducts the processes of automated and manual software testing, describes control examples of testing.

The fourth section describes the deployment, implementation, and membership program processes.

KEY WORDS: CROSSPLATFORM APPLIANCE, INTERIOR PLANNING, 3D.

Пояснювальна записка
до дипломного проєкту

на тему: *Програмне забезпечення для планування інтер'єру у 3D*

Київ – 2021 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

Слайдер – компонент графічного інтерфейсу користувача, що використовується для зміни значень.

Сетінг – середовище, в якому відбувається дія. UI – користувацький інтерфейс.

Скрол-бар – смуга прокрутки, елемент управління інтерфейсу.

View – представлення. Model – модель.

Controller – контролер.

MVC – Model-View-Controller.

Singleton – патерн одинак.

Factory – патерн будівник.

ВСТУП

У сучасному світі, на фоні зростання населення зростає і потреба в новому житлі. Так останні 5 років ринок новобудов Києва та України переживає вибухове зростання. Наприкінці 2019 року з'явилися цікаві цифри щодо загального стану будівництва в Україні. Усього в нашій країні будується 2 100 нових комплексів. Водночас наприкінці 2018 року житлових комплексів по всій країні налічувалося всього 1850.

Не кожен може дозволити собі користуватися послугами професійного архітектора та дизайнера для планування майбутнього житла, також багато хто хоче самостійно продумати житло своєї мрії. Звідси виникає потреба у зручному додатку для планування майбутньої оселі/офісу як для власника майбутньої квартири, так і для забудовника чи просто для людини яка планує ремонт.

Плани поверхів дають повітряну, зменшену ілюстрацію простору. Вони можуть бути комплексними, як проект будинку, що відображають весь внутрішній і зовнішній простір, або гранульованими, як план поверху квартири, візуалізуючи, як може виглядати простір, коли його закінчать і обставлять. Плани поверхів зазвичай розробляються архітекторами і використовуються будівельниками та підрядниками, дизайнерами інтер'єрів та агентами з нерухомості. Призначення плану поверху - дати уявлення про те, як облаштовано простір з точки зору світильників, розмірів та просторових співвідношень. Вони допомагають людям зрозуміти, чи придатні ділянки за призначенням. Наприклад, спільні офісні планувальники робочих приміщень повинні враховувати як функціональність, так і натхнення, щоб випередити конкурентів, пропонуючи відчутні, бажані переваги коворкінгу. Як ви можете собі уявити, створення планів поверхів може бути трудомістким.

Існуюче програмне забезпечення для планування інтер'єру має високу складність користування і не є безкошновним.

Завданням роботи є створення та розгортання програмного забезпечення, яке дозволить користувачеві створювати, редагувати, зберігати та загружати проект дому, та подивитися кінцевий результат у 3D.

Як результат створенно додаток HousePlanner. Додаток дозволяє користувачеві створити проект, розмістити всі необхідні елементи інтер’єру, рухати та видаляти їх, зберігати готовий чи проміжний проект, та завантажувати їх.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

План будинку – це сукупність будівельних або робочих креслень (які іноді називають кресленнями), що визначають усі будівельні характеристики житлового будинку, такі як розміри, матеріали, плани, методи та техніки монтажу. Кінець двадцятого і початок XXI століття заслужено називають періодом становлення і розвитку загальної інформаційної системи. Дійсно, на зміну газетам і журналам і навіть більш солідним (товстим) друкованим продуктам - книгам, приходять електронні аналоги. Це не тільки дешевше, але і зручніше. Інформація стає універсальною і легко засвоюється.

План поверху (Рисунок 1.1) – вид зверху на завершений будинок. На плані ви побачите паралельні лінії, які масштабуються на будь-яку ширину стін, яка повинна бути. Розміри зазвичай проводяться між стінами, щоб вказати розміри приміщень та довжини стін. На планах поверхів також будуть вказані кімнати, всі двері та вікна та будь-які вбудовані елементи, такі як сантехнічні прилади, шафи, водонагрівачі, печі тощо. На планах поверхів будуть примітки із зазначенням обробки, способів будівництва або символів для електричних елементів.

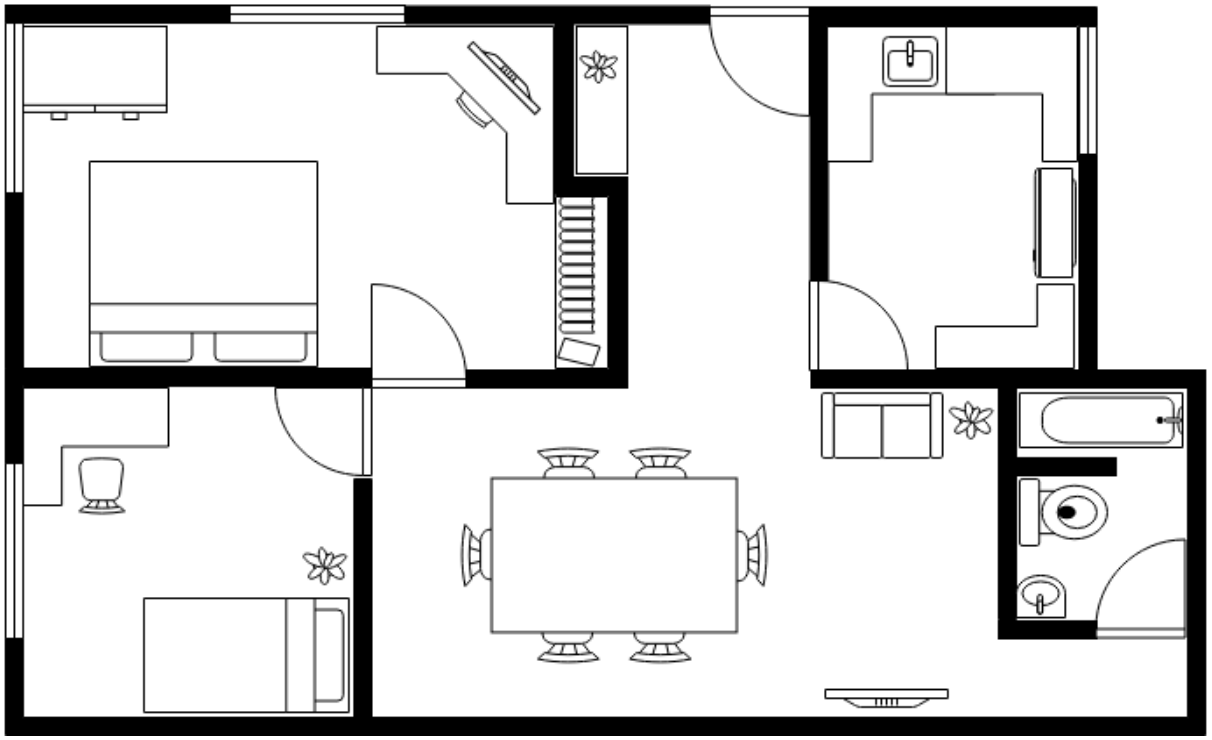

Рисунок 1.1 – Приклад плану поверху

У планах будинків використовуються такі рядки та символи для передачі взаємозв’язку між об’єктами:

1) розмірні лінії, які складаються з суцільної лінії з позначкою на кожному кінці; простір між двома позначками дорівнює відстані, зазначеній поруч із лінією;

2) стіна - товсті суцільні лінії між кімнатами використовуються для позначення стіни;

3) стіна з частковою висотою – низька стіна, яка не тягнеться аж до стелі;

4) вбудовані конструкції, описуються за допомогою тонких суцільних ліній.

1.2 Змістовний опис і аналіз предметної області

Сучасне програмне забезпечення для планування будинків має дуже широкий функціонал, проте найважливіші інструменти це:

А) маштаб та сітка – функціонал який дозволяє зробити план точним, та допоможе орієнтуватися у розмірах плану;

Б) стіни – основний функціонал додатку. Їх додавання, редагування та видалення є обов'язковим;

В) облаштування – можливість облаштовувати кімнати, та зручність цього функціоналу є однією з основних особливостей сучасних планувальників;

Г) 3D відображення – кожен користувач бажає подивитись кінцевий результат перед затвердженням плану. Відстуність такого функціоналу може бути суттєвим фактором при виборі програмного забезпечення.

1.3 Аналіз відомих програмних продуктів

У результаті аналізу існуючих програмних продуктів були знайдені такі рішення:

- AutoCAD Architecture;
- SketchUp;
- Civil 3D.

AutoCAD Architecture (Рисунок 1.2) - це версія флагманського продукту Autodesk, AutoCAD, з інструментами та функціями, спеціально придатними для архітектурних робіт. Архітектурні об'єкти мають взаємозв'язок і розумно взаємодіють між собою. Наприклад, вікно має відношення до стіни, яка його містить. Якщо ви переміщуєте або видаляєте стіну, вікно реагує відповідним чином. Об'єкти можуть бути представлені як у 2D, так і в 3D. Крім того, інтелектуальні архітектурні об'єкти підтримують динамічні зв'язки з будівельними документами та технічними умовами, що призводить до більш точних результатів проекту. Наприклад, коли хтось видаляє або модифікує двері, розклад дверей може бути автоматично оновлений. Пробіли та площі

оновлюються автоматично при зміні певних елементів, розрахунки, такі як квадратні метри, завжди оновлені.

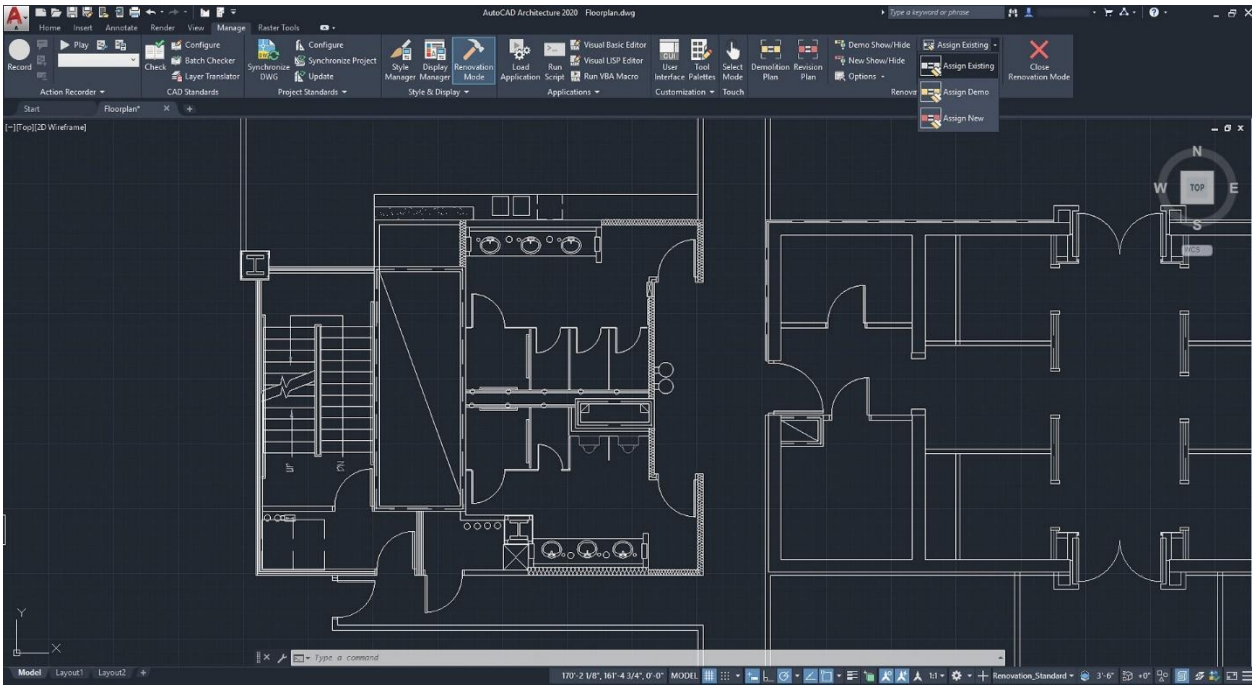

Рисунок 1.2 – Скріншот AutoCAD Architecture

Autodesk® Civil 3D® (Рисунок 1.3) - це рішення для проектування та документації в цивільному будівництві, яке підтримує робочі процеси інформаційного моделювання будівель (BIM) на проектах цивільної інфраструктури, включаючи дороги та автомагістралі, залізницю, розробку ділянок, аеропорти та воду. Civil 3D допомагає професіоналам цивільної інфраструктури покращити реалізацію проектів, зменшити ризик, помилки та упущення та швидше реагувати на зміни проекту. Крім того, користувачі можуть впорядкувати трудомісткі завдання, такі як дизайн коридору (автомобільний та залізничний), дизайн перехрестя, компонування ділянок, класифікація ділянок та дизайн трубопроводів за допомогою конкретних інструментів та настроюваних стандартів дизайну.

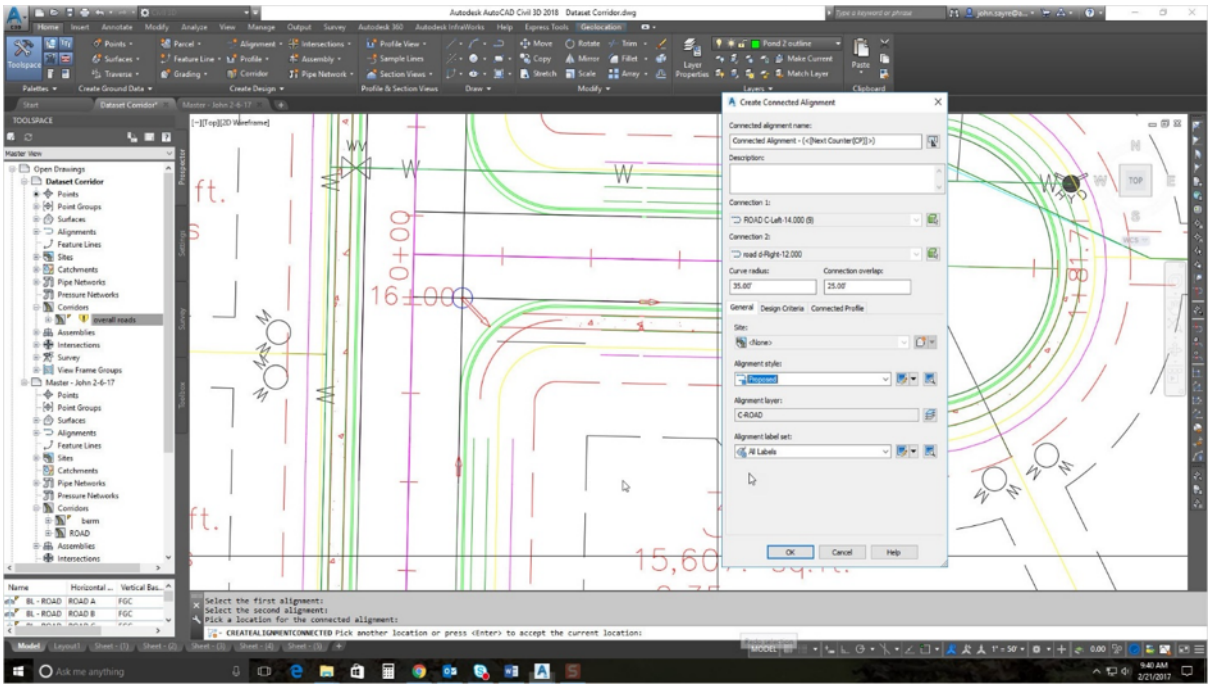

Рисунок 1.3 – Скріншот Autodesk Civil 3D

SketchUp (Рисунок 1.4) - це комп’ютерна програма для тривимірного моделювання для широкого кола програм для малювання, таких як архітектура, дизайн інтер’єру, ландшафтна архітектура, цивільне та машинобудування, дизайн кіно та відеоігор. Він доступний як веб-додаток, SketchUp Free, та платна версія з додатковою функціональністю, SketchUp Pro. Раніше також була доступна безкоштовна версія SketchUp Make. SketchUp належить Trimble Inc., що займається картографічним маркшейдерським та навігаційним обладнанням Існує Інтернет-бібліотека безкоштовних модельних збірок (наприклад, вікна, двері, автомобілі), 3D Warehouse, до якої користувачі можуть внести свої моделі. Програма включає функціонал макета креслення, дозволяє повертати поверхню у змінних "стилях", підтримує сторонні програми "плагіни", розміщені на сайті, що називається Extension Warehouse, для надання інших можливостей (наприклад, біля фотореалістичного візуалізації) і дозволяє розміщення моделі в Google Планета Земля.

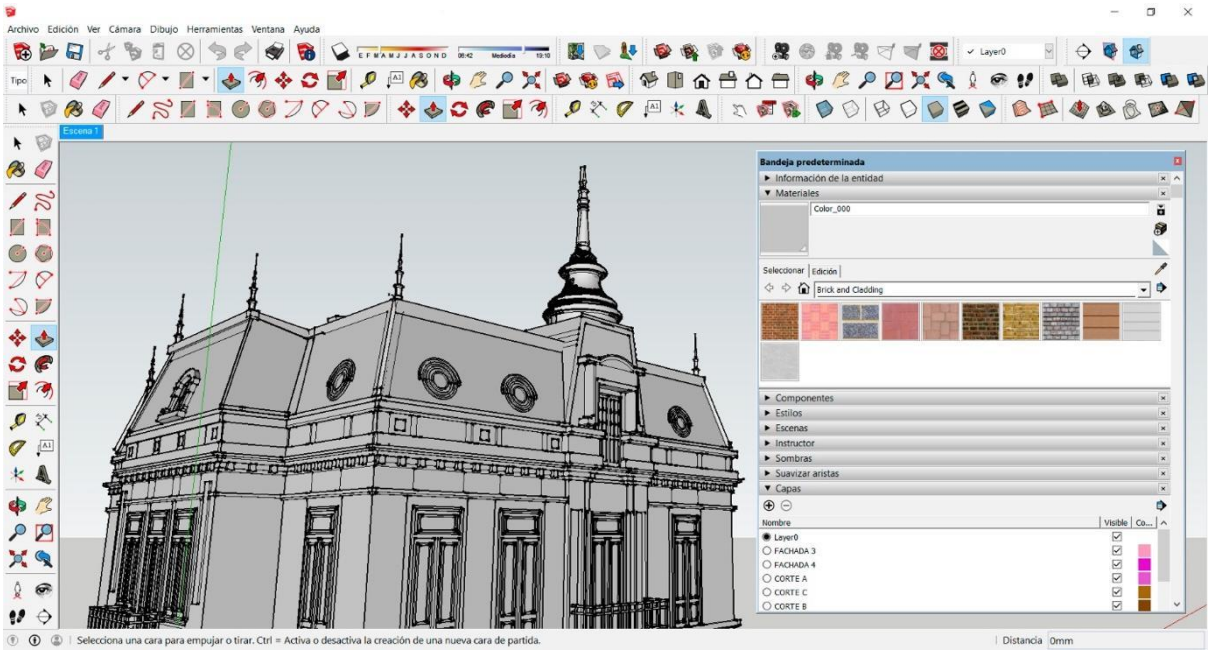

Рисунок 1.4 - Скріншот SketchUp

Усі перераховані програмні продукти мають достатньо складний інтерфейс для початківця і вимагають навичок для створення навіть простих планів інтер'єру.

1.4 Аналіз вимог до програмного забезпечення

1.4.1 Розроблення функціональних вимог

Для створення use-case діаграми потрібно визначити акторів, що взаємодіятимуть з програмним забезпеченням. Оскільки застосунок є десктопним додатком, єдиним учасником системи є користувач. Схему варіантів використання застосунку представлено у Додатку «Графічний матеріал».

Додаток може бути використаний за нижченавединими варіантами використання(use cases) у таблицях 1.1 – 1.7

Таблиця 1.1 – Варіант використання UC01 «додати елемент»

Назва	Додати елемент
Опис	Користувач може додати новий елемент до плану стіну, мебель, тощо.
Актор	Користувач
Передумови	-
Постумови	Новий елемент доданий до проекту
Сценарій	Користувач активує інструмент додавання, та додає елемент до плану в бажаному місці за допомогою мишки
Розширені сценарії	-

Таблиця 1.2 – Варіант використання UC02 «Редагувати елемент»

Назва	Редагувати елемент
Опис	Користувач може змінити доданий елемент за бажанням.
Актор	Користувач
Передумови	Користувач додав елемент, чи загрузив готовий проект
Постумови	Елемент має інші властивості
Сценарій	Користувач активує інструмент зміни, та змінює елемент за допомогою мишки
Розширені сценарії	Користувач обирає елемент за допомогую мишки, або в дереві об'єктів та змінює його властивості у вікні властивостей

Таблиця 1.3 – Варіант використання UC03 «Видалити елемент»

Назва	Видалити елемент
Опис	Користувач може видалити доданий елемент.
Актор	Користувач
Передумови	Користувач додав елемент, чи загрузив готовий проект
Постумови	Елемент видалений з проекту
Сценарій	Користувач активує інструмент видалення, та видаляє елемент за допомогою мишки
Розширені сценарії	Користувач обирає елемент за допомогую мишки, або в дереві об'єктів та видаляє його за допомогою кнопки "Delete"

Таблиця 1.4 Варіант використання UC04 «Зберегти проект»

Назва	Зберегти проект
Опис	Користувач може зберігти проект до файлу
Актор	Користувач
Передумови	-
Постумови	Проект збережений до файлу
Сценарій	Користувач переходить до контекстного меню, натискає "Save", пише ім'я файлу, та підтверджує операцію
Розширені сценарії	-

Таблиця 1.5 – Варіант використання UC05 «Завантажити проект»

Назва	Загрузити проект
Опис	Користувач може загрузити проект за файлу
Актор	Користувач
Передумови	-
Постумови	Проект загруженний з файлу
Сценарій	Користувач переходить до контекстного меню, натискає "Load", вибирає файл з файлової системи, та підтверджує операцію
Розширені сценарії	-

Таблиця 1.6 – Варіант використання UC06 «Відмінити останню дію»

Назва	Відмінити останню дію
Опис	Користувач може відмінити останню виконану дію
Актор	Користувач
Передумови	Користувач виконав якусь зміну до проекту
Постумови	Остання виконана дія була відмінена
Сценарій	Користувач натискає на стрілочку "undo", або використовує хот-кей ctrl+z
Розширені сценарії	-

Таблиця 1.7 – Варіант використання UC07 «Подивитися план на 3д сцені»

Назва	Подивитися план на 3д сцені

Опис	Користувач може подивитися план у 3д

Продовження таблиці 1.7

Актор	Користувач
Передумови	-
Постумови	Відкрите нове вікно з 3д-сценою
Сценарій	Користувач натискає на кнопу "3д-сцена"
Розширені сценарії	-

Функціональні вимоги описані в таблицях 1.8 – 1.22

Таблиця 1.8 – Опис функціональної вимоги REQ1

Назва	Додавання стін на план
Опис	Додаток має надавати можливість додати стіну на план

Таблиця 1.9 – Опис функціональної вимоги REQ2

Назва	Додавання меблей на план
Опис	Додаток має надавати можливість додати меблі на план

Таблиця 1.10 – Опис функціональної вимоги REQ3

Назва	Додавання кімнат
Опис	Додаток має надавати можливість додати кімнату

Таблиця 1.11 – Опис функціональної вимоги REQ4

Назва	Вибирання елементів
Опис	Додаток має надавати можливість вибирати елементи на плані

Таблиця 1.12 – Опис функціональної вимоги REQ5

Назва	Видалення елементів
Опис	Додаток має надавати можливість видалити вибрані елементи з плану

Таблиця 1.13 – Опис функціональної вимоги REQ6

Назва	Пересування елементів
Опис	Додаток має надавати можливість пересувати вибрані елементи на плані

Таблиця 1.14 – Опис функціональної вимоги REQ7

Назва	Обертання елементів
Опис	Додаток має надавати можливість обертати вибрані елементи на плані

Таблиця 1.15 – Опис функціональної вимоги REQ8

Назва	Відміна останньої дії
Опис	Додаток має надавати можливість відмінити останню виконану дію

Таблиця 1.16 – Опис функціональної вимоги REQ9

Назва	Повернутись на дію вперед
Опис	Додаток має надавати можливість повернути відмінену дію

Таблиця 1.17 – Опис функціональної вимоги REQ10

Назва	Додавання дверей та вікон
Опис	Додаток має надавати можливість додавати на стіни вікна та двері

Таблиця 1.18 – Опис функціональної вимоги REQ11

Назва	Маштаб на сітка
Опис	Додаток має містити маштаб та маштабовану сітку для більш точного розміщення елементів

Таблиця 1.19 – Опис функціональної вимоги REQ12

Назва	Збереження плану
Опис	Додаток має надавати можливість зберегти план

Таблиця 1.20 – Опис функціональної вимоги REQ13

Назва	Завантаження плану
Опис	Додаток має надавати можливість завантажити план

Таблиця 1.21 – Опис функціональної вимоги REQ14

Назва	Редагування властивостей елементів
Опис	Додаток має надавати можливість редагувати властивості елементів

Таблиця 1.22 – Опис функціональної вимоги REQ15

Назва	Перегляд плану у 3D
Опис	Додаток має надавати можливість переглядати план у 3D

Таблиця 1.23 – Матриця залежностей вимог та варіантів використання

	UC01	UC02	UC03	UC04	UC05	UC06	UC07
R01	■						
R02	■						
R03	■						
R04		■	■				
R05			■				
R06		■					
R07		■					
R08						■	
R09						■	
R10	■						
R11	■						
R12				■			
R13					■		
R14		■					
R15							■

1.4.2 Розроблення нефункціональних вимог

До нефункціональних вимог відносяться:

- кросс-платформеність додатку, тобто можливість використання додатку як на Windows так і на Unix системах;
- головна мова додатку – англійська.

1.4.3 Постановка завдання

Метою розробки є створення зручного та інтуїтивного інструменту для планнування майбутньої оселі/офісу. Для того, щоб мета вважалась досягнутою, проєкт має відповідати ряду технічних задач, а саме:

- створення плану кімнат;
- додавання облаштування кімнат;
- можливість редагування існуючого плану;
- можливість збереження та завантаження проекту;
- можливість перегляду кінцевого результату.

1.5 Висновки до розділу

Зі зростанням будівельного бізнесу у світі та зокрема в Україні виникла потреба у софті для планування. У данному розділі описано предметну область розробки, зокрема проаналізовано існуючі продукти, та перерахований функціонал, який повинен бути присутній.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

З метою створення високоякісного додатку для початку необхідно описати процеси взаємодії користувача з програмним забезпеченням. Детальний бізнес-процес редагування описано у Додатку «Графічний матеріал».

Процеси взаємодії користувача і додатку :

- користувач додає елемент, программа у відповідь додає елемент до проекту, записує команду до стеку undo, та очищає стек команд redo;
- користувач видаляє елемент, программа у відповідь видаляє елемент з проекту, записує команду до стеку undo, та очищає стек команд redo;
- користувач редагує елемент, программа у відповідь змінює проект, записує команду до стеку undo, та очищає стек команд redo;
- користувач завантажує проект, программа у відповідь замінює дійсний проект на новий з файлу, та очищає стек команд undo та redo;
- користувач зберігає, программа у відповідь серіалізує та зберігає проект до файлу;
- користувач відміняє останню дію, программа у відповідь прибирає останню зміну до проекту, та видаляє останню команду з стеку undo, та добавляю команду до стеку redo;
- користувач повертається на дію вперед, программа у відповідь змінює проект, записує команду до стеку undo, та видаляє останню команду з стеку redo;
- користувач відкриває 3д сцену, программа у відповідь відкриває нове вікно з 3д сценою, та рендерить усі об’єкти з проекту.

2.2 Архітектура програмного забезпечення

Програмне забезпечення реалізовано за допомогою MVC архітектури, підлаштнованої під особливості десктопного додатку.

QT дозволяє зручно зв'язувати елементи UI та елементами классу

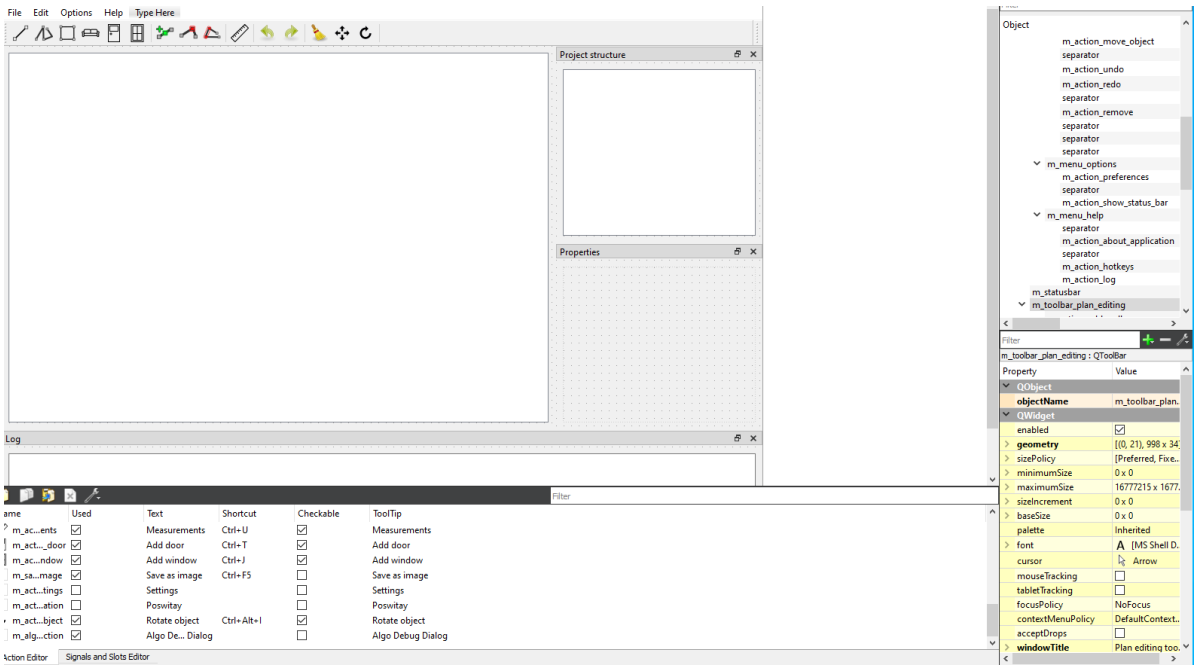

Рисунок 2.1 – Вигляд вікна редагування UI

```
connect(m_ui.m_action_new_plan,          &QAction::triggered, this, &MainWindow::NewPlan);
connect(m_ui.m_action_save_file,         &QAction::triggered, this, &MainWindow::SavePlan);
connect(m_ui.m_action_save_as,           &QAction::triggered, this, &MainWindow::SaveAsPlan);
connect(m_ui.m_action_open_file,         &QAction::triggered, this, &MainWindow::OpenPlan);
connect(m_ui.m_action_show_status_bar,   &QAction::changed,   this, &MainWindow::ToggleShowStatusBarAction);
connect(m_ui.m_action_close_application, &QAction::triggered, this, &MainWindow::close);
connect(m_ui.m_action_hotkeys,           &QAction::triggered, this, &MainWindow::ShowHotkeysHelp);
connect(m_ui.m_action_preferences,       &QAction::triggered, this, &MainWindow::OpenPreferences);
```

Рисунок 2.2 – Зв’язування елементу UI з методом у коді

View відображає дані та реагує на взаємодію користувача з UI. View не знає про модель чи контроле при конкретних діях користувача (натисканні кнопки, зміні значення поля тощо).

Контролер містить посилання на відповідну йому View та класи моделі, він може змінювати View та даны моделы, а також ініціювати відкриття нових вікон. В конструкторі контролер отримує посилання на ві інструменті, з якими він має працювати та вибирає з пулу потрібну View. Тут же відбувається ініціалізація данних та конект слотів та подій. Моделі відповідають за збереження на підрахунки данних.

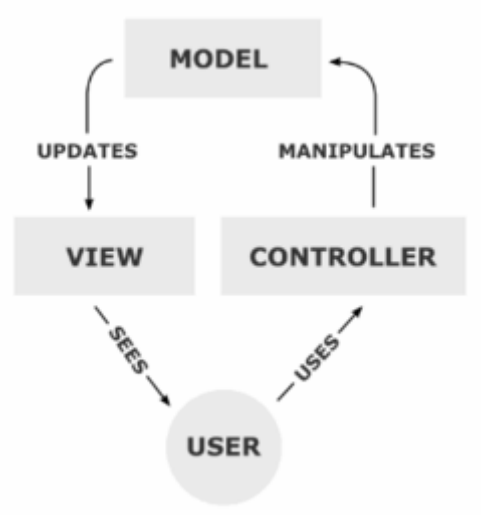

Рисунок 2.3 – Схема роботи патерна MVC

У ході розробки булі використані такі патерни проектування як синглтон (Singleton) та фабрика (Factory).

Синглтон - це шаблон дизайну програмного забезпечення, який обмежує інстанціювання класу одним "єдиним" екземпляром. Це корисно, коли для координації дій у системі потрібен рівно один об’єкт. Цей термін походить від математичного поняття синглтона.

Шаблон проектування singleton є одним із двадцяти трьох добре відомих шаблонів дизайну " Gang of Four ", що описують, як вирішувати повторювані дизайнерські завдання для проектування гнучкого та багаторазового об'єктно-орієнтованого програмного забезпечення, тобто об'єктів, які легше впровадити, змінити, протестувати та повторно використати.

Синглтон дозволяє визначте загальнодоступну статичну операцію (getInstance ()), яка повертає єдиний екземпляр класу.

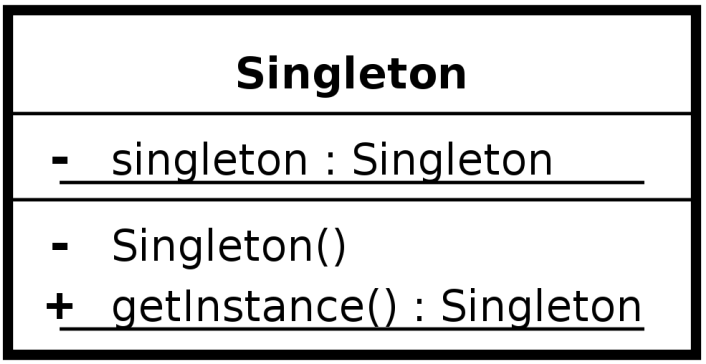

Рисунок 2.4 – Опис шаблону «синглтон» у UML

Шаблон проектування Factory - це творчий шаблон, який використовує фабричні методи для вирішення проблеми створення об'єктів без необхідності вказувати точний клас об'єкта, який буде створений. Це робиться шляхом створення об'єктів шляхом виклику фабричного методу - або зазначеного в інтерфейсі та реалізованого дочірніми класами, або реалізованого в базовому класі і необов'язково заміненого похідними класами, а не за допомогою конструктора.

Фабрика - один із шаблонів проектування "Gang of Four", який описує, як вирішувати повторювані дизайнерські завдання для проектування гнучкого та багаторазово використовуваного об’єктно-орієнтованого програмного забезпечення, тобто об’єктів, які легше впровадити, змінити, протестувати та повторне використання.

Шаблон проектування Factory Factory використовується замість конструктора звичайного класу для дотримання принципів програмування SOLID, роз'єднання конструкції об'єктів з самими об'єктами.

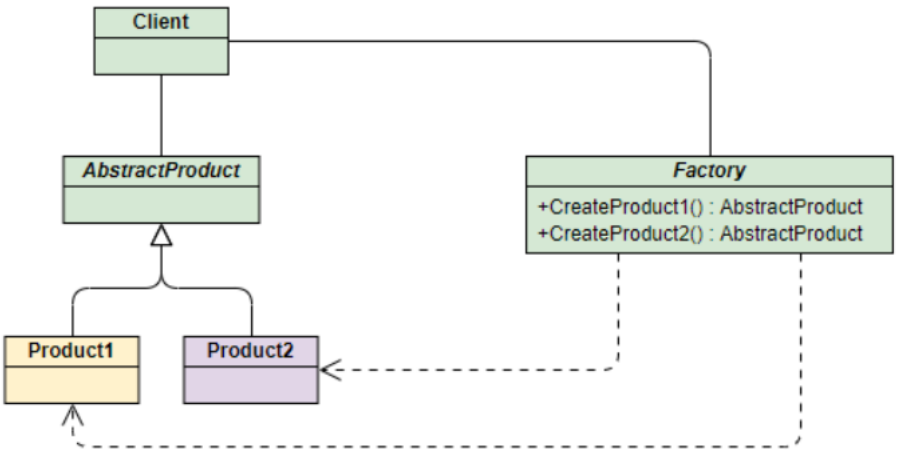

Рисунок 2.5 – Опис патерну «фабрика» у UML

2.3 Конструювання програмного забезпечення

2.3.1 Вибір технологій розробки програмного забезпечення

Для забезпечення швидкодії та оптимальної роботи додатку було обрано мову програмування C++.

C++ - це мова програмування загального призначення, створена Бьярном Страструпом як розширення мови програмування C, або "C with Classes". З плином часу мова значно розширилася, і сучасний C++ тепер має об'єктно-орієнтовані, загальні та функціональні функції, на додаток до засобів для маніпуляцій з пам'яттю низького рівня. Він майже завжди реалізується як компільована мова, і багато постачальників надають компілятори C ++, зокрема Free Software Foundation, LLVM, Microsoft, Intel, Oracle та IBM, тому він доступний на багатьох платформах. [11]

C ++ добре підходить до різноманітних графічних інтерфейсів, що забезпечує більшу універсальність дизайну. Наприклад, комбінація QML та фреймворку Qt (фреймворк на основі C ++) надає розробникам можливість створювати привабливі дизайни графічного інтерфейсу.

C ++ був розроблений з орієнтацією на системне програмування та вбудоване програмне забезпечення, обмежене ресурсами, та великі системи, що характеризує продуктивність, ефективність та гнучкість

використання. C++ також виявився корисним у багатьох інших контекстах, головними перевагами якого є програмна інфраструктура та обмежені ресурси, включаючи настільні програми, відеоігри, сервери (наприклад, електронну комерцію, веб-пошук або бази даних) та критичні для продуктивності програми (наприклад, телефонні комутатори або космічні зонди).

```
Matrix3Df operator*(const Matrix3Df& i_mat1, const Matrix3Df& i_mat2)
  {
  Matrix3Df result;
  for (size_t i = 0; i < 3; i++)
    result.m_cols[i] = Vector3Df(
      DotProduct(i_mat1.Row(0), i_mat2.Col(i)),
      DotProduct(i_mat1.Row(1), i_mat2.Col(i)),
      DotProduct(i_mat1.Row(2), i_mat2.Col(i)));
  return result;
  }

Matrix3Df Transpose(const Matrix3Df& i_mat)
  {
  return Matrix3Df(
    i_mat.Row(0),
    i_mat.Row(1),
    i_mat.Row(2));
  }

Matrix3Df Inverse(const Matrix3Df& i_mat)
  {
  //this code is taken from GLM library
  Matrix3Df m = i_mat;
  long double one_over_determinant = 1.0L / (
    +m[0][0] * (m[1][1] * m[2][2] - m[2][1] * m[1][2])
    - m[1][0] * (m[0][1] * m[2][2] - m[2][1] * m[0][2])
    + m[2][0] * (m[0][1] * m[1][2] - m[1][1] * m[0][2]));

  Matrix3Df inverse;
  inverse[0][0] = +(m[1][1] * m[2][2] - m[2][1] * m[1][2]) * one_over_determinant;
  inverse[1][0] = -(m[1][0] * m[2][2] - m[2][0] * m[1][2]) * one_over_determinant;
  inverse[2][0] = +(m[1][0] * m[2][1] - m[2][0] * m[1][1]) * one_over_determinant;
  inverse[0][1] = -(m[0][1] * m[2][2] - m[2][1] * m[0][2]) * one_over_determinant;
  inverse[1][1] = +(m[0][0] * m[2][2] - m[2][0] * m[0][2]) * one_over_determinant;
  inverse[2][1] = -(m[0][0] * m[2][1] - m[2][0] * m[0][1]) * one_over_determinant;
  inverse[0][2] = +(m[0][1] * m[1][2] - m[1][1] * m[0][2]) * one_over_determinant;
  inverse[1][2] = -(m[0][0] * m[1][2] - m[1][0] * m[0][2]) * one_over_determinant;
  inverse[2][2] = +(m[0][0] * m[1][1] - m[1][0] * m[0][1]) * one_over_determinant;

  return inverse;
  }
}
```

Рисунок 2.6 – Приклад коду написаного на C++

Оскільки однією з нефункціональних вимог була кросс-платформеність, було обрано бібліотеку Qt.

Qt – це набір інструментів віджетів для створення графічних користувальницьких інтерфейсів, а також міжплатформенних додатків, що працюють на різних програмних та апаратних платформах, таких як Linux,

Windows, macOS, Android або вбудовані системи, з незначними або відсутніми змінами в базовій кодовій базі. рідна програма з рідними можливостями та швидкістю.

В даний час Qt розробляється компанією The Qt, яка публічно котирується, та проектом Qt під управлінням з відкритим кодом, залучаючи окремих розробників та організації, що працюють над просуванням Qt. Qt доступний як за комерційними ліцензіями, так і за ліцензіями GPL 2.0, GPL 3.0 та LGPL 3.0 з відкритим кодом.

Qt використовується для розробки графічних користувальницьких інтерфейсів (GUI) та мультиплатформенних додатків, що працюють на всіх основних настільних платформах та більшості мобільних або вбудованих платформ. Більшість програм графічного інтерфейсу, створені за допомогою Qt, мають власний інтерфейс, і в цьому випадку Qt класифікується як набір віджетів. Також можуть бути розроблені програми, що не належать до графічного інтерфейсу, такі як інструменти командного рядка та консолі для серверів. Прикладом такої програми, що не стосується графічного інтерфейсу, використовує Qt, є веб-фреймворк Cutelyst.

Qt підтримує різні компілятори, включаючи компілятор GCC C ++, набір Visual Studio, PHP через розширення для PHP5 та має широку підтримку інтернаціоналізації. Qt також надає Qt Quick, що включає декларативну мову сценаріїв під назвою QML, що дозволяє використовувати JavaScript для забезпечення логіки. Завдяки Qt Quick стала можливою швидка розробка додатків для мобільних пристроїв, тоді як логіка все ще може бути записана за допомогою власного коду для досягнення найкращої можливої продуктивності.

Інші функції включають доступ до бази даних SQL, синтаксичний аналіз XML, синтаксичний аналіз JSON, управління потоками та підтримку мережі.

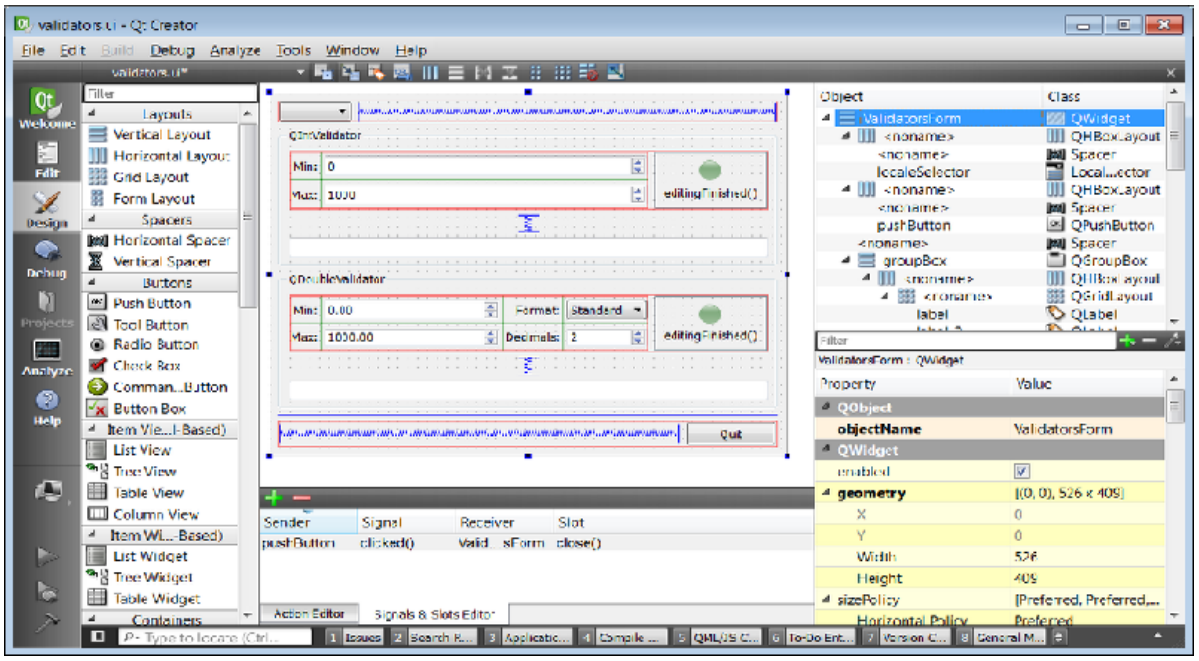

Рисунок 2.7 – Приклад розробки програмного інтерфейсу з використанням фреймворку Qt

Як інструмент зборки проекту було CMake, бо він є дефолтним інструментом для роботи з Qt.

CMake - це безкоштовне міжплатформене програмне забезпечення з відкритим кодом для автоматизації збірки, тестування, упаковки та встановлення програмного забезпечення за допомогою незалежного від компілятора методу. CMake не є системою збірки, а скоріше генерує файли збірки іншої системи. Він підтримує ієрархію каталогів та програми, які залежать від декількох бібліотек. Він використовується у поєднанні з рідними середовищами збірки, такими як Make, Qt Creator, Ninja, Android Studio, Apple Xcode та Microsoft Visual Studio. Він має мінімальні залежності, вимагаючи лише компілятора C ++ у власній системі складання.

CMake поширюється як програмне забезпечення з відкритим кодом за дозвольною новою ліцензією BSD.

```
add_precompiled_header(HousePlanner "stdh.h" "stdh.cpp")
list(APPEND GEN_SRC ${QRC_SRC} ${MOC_SRC})
if(MSVC)
    foreach(src_file ${GEN_SRC})
        set_source_files_properties(${src_file} PROPERTIES COMPILE_FLAGS "/Yustdh.h /FIstdh.h")
    endforeach()
endif(MSVC)

#This function is located in external/CMakeLists.txt
assign_source_group(${SRC_LIST} ${MOC_SRC} ${UIS})

add_dependencies(HousePlanner
    CoreData
    CoreMath
    CoreLogic
    Viewer3D
)

target_link_libraries(
    HousePlanner
    CoreData
    CoreMath
    CoreLogic
    Viewer3D
    Qt5::Core
    Qt5::Widgets
    Qt5::WinMain
    Qt5::Gui
)


set_target_properties(HousePlanner PROPERTIES LINK_FLAGS_RELEASE "/SUBSYSTEM:WINDOWS")
set_target_properties(HousePlanner PROPERTIES LINK_FLAGS_DEBUG "/SUBSYSTEM:WINDOWS")
```

Рисунок 2.8 – Лінковка проекту за допомогою CMake

2.3.2 Модулі програмного забезпечення

Код програми був розбитий на модулі для полегшення розробки та структурування програмного коду.

Всього додаток розбитий на 5 модулів:

1. “CoreData” – модуль з усіма ключовими структурами для збереження данних;
2. “CoreMath” – модуль з усією математикою та алгоритмічними складовими;
3. “ImportExport” – модуль з логікою для імпорту та експорту проекту;
4. “HousePlanner” – основний модуль, який містить UI та контроллер;
5. “Viewer3D” – модуль якій відповідає за 3Д сцену.

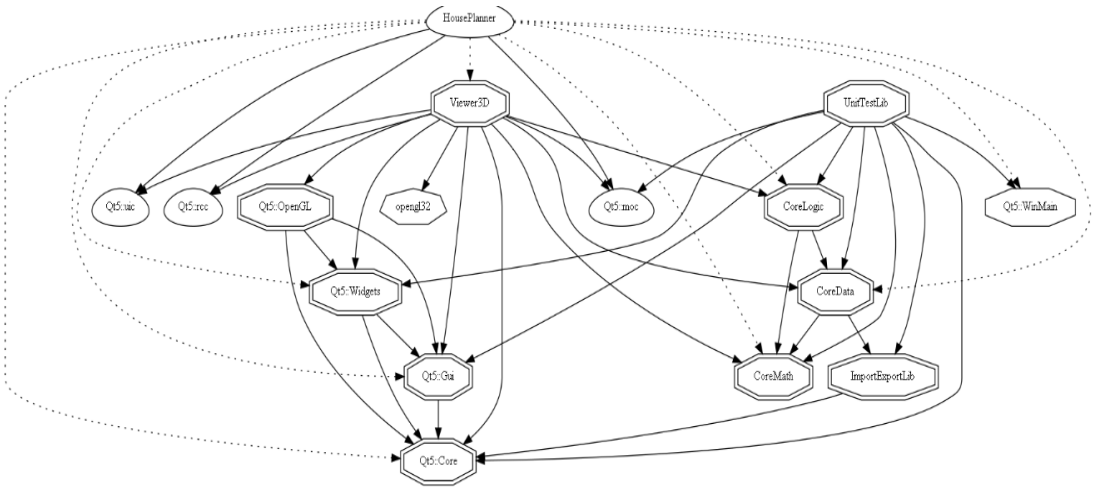

Рисунок 2.7 – Схема залежностей між модулями

Розбиття на модулі полегшує тестування, та процесс розробки, дозволяючи працювати над окремими маленькими модулями, покривати їх тестами, тощо. Також полегшує знаходженя багів у коді.

Класи які відповідають за збереження та серіалізацію можно побачити в таблиці 3.1.

Таблиця 3.1 – Опис классів модулю "CoreData"

Класс	Опис
Door	Займается збереження та обробкою данних о дверях, а також інформацією про читання та запис
Window	Займается збереження та обробкою данних о вікнах, а також інформацією про читання та запис
Furniture	Займается збереження та обробкою данних о меблях, а також інформацією про читання та запис
Wall	Займается збереження та обробкою данних о стінах, а також інформацією про читання та запис
WallNode	Займается збереження та обробкою данних о перетинаї стін, а також інформацією про читання та запис

Продовження таблиці 3.1

Plan	Займається збереження та обробкою данних о всіх елементах плану, а також інформацією про читання та запис проекту

Класи які відповідають за збереження та математичну логіку можно побачити в таблиці 3.2

Таблиця 3.2 – Опис класів модулю "CoreMath"

Класс	Опис
DirectedSegment	Займається збереженням відрізків, та містить частину математичної логіки
GraphVertex	Займається збереженням нодів графу та містить частину математичної логіки
Matrix3Df	Займається збереженням 3д матриць, та містить частину математичної логіки
Vector2D	Займається збереженням 2д точок, та алгоритмами пов'язаними з ними
Vector3D	Займається збереженням 3д точок, та алгоритмами пов'язаними з ними

Класи які відповідають за імпорт та експорт можно побачити в таблиці 3.3

Таблиця 3.3 – Опис классів модулю "ImportExport"

Класс	Опис
XmlSerializer	Відповідає за збереження проекту
XmlDeserializer	Відповідає за зчитування проекту

Класи які відповідають за UI та Контролер описано в таблиці 3.4.

Таблиця 3.4 - Опис классів модулю "HousePlanner"

Класс	Опис
Commands	Відповідають за логіку undo-redo(додавання, редагування, тощо)
Canvas	Відповідає за обробку сигналів, та відображені елементів на канвасі
CanvasView	Відповідає за обробку сигналів, та відображені елементів навколо канвасу (рулетка)
Logger	Відповідає за записання логів
Ruller	Відповідає за роботи рулетки
ScaleManager	Відповідає за скейлінг рулетки
Settings	Відповідає за збереження налаштувань
SettingsDialog	Відповідає за обробку сигналів діалогу налаштувань
UndoRedoManager	Відповідає за обробку сигналів undo-redo, та збереження команд

Продовження таблиці 3.4

MainWindow	Відповідає за головне вікно, та підключення усіх елементів інтерфейсу до відповідних контрорелів
Project	Відповідає за усю логіку бек енду та взаємодію головного вікна з логікою.
AddDoorComand	Відповідає за збереження інформації про додавання дверей до плану
AddFurnitureCommand	Відповідає за збереження інформації про додавання меблей до плану
AddVertexCommand	Відповідає за збереження інформації про додавання ноди стінки до плану
AddWallCommand	Відповідає за збереження інформації про додавання стінки до плану
AddWindowCommand	Відповідає за збереження інформації про додавання вікна до плану
MoveRotateObjectCommand	Відповідає за збереження інформації про передвигання та повертання обьектів
MoveVerticesCommand	Відповідає за збереження інформації про передвигання нод стінок
RemoveDoorCommand	Відповідає за збереження інформації про видалення дверей
RemoveFurnitureCommand	Відповідає за збереження інформації про видалення меблей

Продовження таблиці 3.4

RemoveWallCommand	Відповідає за збереження інформації про видалення стінок
RemoveWindowCommand	Відповідає за збереження інформації про видалення вікон
AddDoorMouse	Відповідає за інструмент додавання дверей
AddFurnitureMouse	Відповідає за інструмент додавання меблей
AddPolylineMouse	Відповідає за інструмент додавання декількох стін
AddRectangleMouse	Відповідає за інструмент додавання прямокутних кімнат
AddVertexMouse	Відповідає за інструмент додавання ноди стінки
AddWallMouse	Відповідає за інструмент додавання стінки
AddWindowMouse	Відповідає за інструмент додавання вікон
DoorGraphicsPreview	Відповідає за відображення графічного превью дверей
FurnitureGraphicPreview	Відповідає за відображення графічного превью дверей

Продовження таблиці 3.4

Measurement Mouse	Відповідає за інструмент лінійки
MoveObjectMouse	Відповідає за інструмент передвигання об'єктів
MoveVertexMouse	Відповідає за інструмент передвигання вертексів стінок
RotateObjectMouse	Відповідає за інструмент передвигання вертексів стінок
SelectionMouse	Відповідає за інструмент вибору
WallGraphicsPreview	Відповідає за відображення графічного превью дверей
WindowGraphicPreview	Відповідає за відображення графічного превью дверей
UIDoor	Відповідає за графічне відображення дверей на канвасі
UIFurniture	Відповідає за графічне відображення меблів на канвасі
UIWall	Відповідає за графічне відображення стінок на канвасі
UIWindow	Відповідає за графічне відображення вікон на канвасі

Продовження таблиці 3.4

Canvas	Відповідає за відображення всіх графічних елемнтів, та оброку кліків юзера
CanvasMouse Controller	Відповідає за оброку кліків юзера відповідним інструментом
FurnitureProperties	Відповідає за відображення властивостей меблів
Logger	Відповідає за логіювання усіх дій

Класси які відповідають за вікно з 3D сценою можно побачити в таблиці 3.5.

Таблиця 3.5 - Опис классів модулю "Viewer3D"

Класс	Опис
Viewer3D	Відповідають за вікно 3д сцени
ViewCamera	Відповідає за 3д камеру
MainWidget	Відповідає за рендерінг 3д сцени
3d Objects	Відповідають за рендерінг окремих об'єктів

2.4 Аналіз безпеки даних

Для імпорту та експорту плану використовується .xml формат. Тому є можливим випадок, коли файл був змінений поза додатком і може містити данні, які зломають программу. Для цього при зчитуванні відбувається валідація зчитуванних данних. Данні проходять перевірку на тип, та на валідність значень.

2.5 Висновки до розділу

У розділі було розглянуто алгоритм дій користувача програмного забезпечення та відповіді програми. Представлено перелік архітектурних рішень. Також детально описано класси та реалізацію застосунку.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Коли триває проект з розробки програмного забезпечення, потрібно знати, що помилки можуть з’являтися в будь-яку фазу життєвого циклу. Як відомо, мало хто з них є невиявленими. Таким чином, важливість забезпечення якості не можна ігнорувати. Існує велика ймовірність того, що остаточний код має помилки у функціональності та дизайні. Для виявлення проблем до появи в критичному середовищі є необхідною умовою проведення тестування програмного забезпечення. Це буває невід’ємною частиною процесу. Потрібно пам’ятати, що ціна через несправність програмного забезпечення може бути дійсно високою.

Баги у процесі розробки сповільняють команду та вимагають додаткових ресурсів на вирішеня проблем. Аби тестування було максимально евективним, варто ще до початку розробки продумати тест-план. Це дозволяє зважити ризки які можуть виникнути в процесі розробки, та разробити чіткі функціональні вимоги.

3.2 Опис процесів тестування

Було здійснено мануальне тестування розробником, а також розроблено автоматичні юніт-тести.

3.3 Опис контрольного прикладу

3.3.1 Мануальне тестування

Детальний план мануального тестування наведено у таблиці 3.1.

Таблиця 3.1 - Детальний план тестування

Номер	Опис	Кроки вітворення	Очікуваний результат
1	Додавання стінки	Вибрати інструмент додавання стін. Вибрати першу точку на плані, та натиснути ЛКМ вибрати другу точку на плані натиснути ЛКМ	На плані додана стінка
2	Створеня кімнати	Вибрати інструмент додавання стін. Вибрати першу точку на плані, та натиснути ЛКМ вибрати другу точку на плані натиснути ЛКМ	На плані додана кімната
3	Видалення стінки	Додати стінку за допомогою інструменту додавання стін. Вибрати стінку, та натиснути видалити	Стінка видалена з плану

Продовження таблиці 3.1

4	Видалення декількох стінов	Додати кімнату за допомогою інструменту додавання кімнат. Вибрати всі стінки, та натиснути видалити	Стінки видалені з плану
5	Додавання меблів	Вибрати інструмент додавання меблів Вибрати першу точку на плані, та натиснути ЛКМ.	На план додані меблі
6	Видалення меблів	Додати стінку за допомогою інструменту додавання стін. Вибрати мебель, та натиснути видалити	Мебель видалена з плану

Продовження таблиці 3.1

7	Рухання стінки	Додати стінку за допомогою інструменту додавання стін. Вибрати інструмент руху стінок, вибрати стінку та перетягнути її	Стінка змінила поло ження
8	Рухання меблів	Додати мебель за допомогою інструмента додавання меблів. Вибрати інструмент руху. Перетягнути мебель.	Мебель змінила положення
9	Рухання нодів	Додати стінку за допомогою інструменту додавання стін. Вибрати інструмент руху нодів стінки, вибрати ноду та перетягнути її	Одна з нод стінки змінила положення

Продовження таблиці 3.1

10	Лінійка	Вибрати інструмент лінійка. Вибрати першу точку на плані, та натиснути ЛКМ вибрати другу точку на плані натиснути ЛКМ	На екрані присутня відстань між двома точками
11	Додавання дверей	Додати стінку за допомогою інструменту додавання стін.Вибрати інструмент додавання дверей Вибрати місце на стіні, та натиснути ЛКМ.	Двері додані до стіни
12	Додавання вікон	Додати стінку за допомогою інструменту додавання стін.Вибрати інструмент додавання вікон. Вибрати місце на стіні, та натиснути ЛКМ.	Вікно додане до стіни

Продовження таблиці 3.1

13	Undo додавання стінки	Додати стінку за допомогою інструменту додавання стін. Натиснути "Undo".	Стінка видалена з плану
14	Undo додавання меблів	Додати меблі за допомогою інструменту додавання меблів. Натиснути "Undo".	Мебель видалена з плану
15	Undo додавання кімнат	Додати кімнату за допомогою інструменту додавання кімнат. Натиснути "Undo".	Кімната видалена з плану

Продовження таблиці 3.1

16	Додавання вікон	Додати стінку за допомогою інструменту додавання стін.Вибрати інструмент додавання вікон. Вибрати місце на стіні, та натиснути ЛКМ.	Вікно додане до стіни
17	Undo додавання стінки	Додати стінку за допомогою інструменту додавання стін. Натиснути "Undo".	Стінка видалена з плану
18	Undo додавання меблів	Додати меблі за допомогою інструменту додавання меблів. Натиснути "Undo".	Мебель видалена з плану

Продовження таблиці 3.1

19	Undo додавання кімнат	Додати кімнату за допомогою інструменту додавання кімнат. Натиснути "Undo".	Кімната видалена з плану
20	Redo додавання меблів	Додати меблі за допомогою інструменту додавання меблів. Натиснути "Undo". Потім натиснути "Redo".	На план додані меблі
21	Redo додавання кімнат	Додати кімнату за допомогою інструменту додавання кімнат. Натиснути "Undo". Потім натиснути "Redo".	На плані додана кімната

Продовження таблиці 3.1

22	Redo додавання вікон	Додати стінку за допомогою інструменту додавання стін. Додати вікно за допомогою інструменту додавання вікон. Натиснути "Undo" двічі. Потім натиснути "Redo".	Вікно додане до стіни
23	Redo додавання дверей	Додати стінку за допомогою інструменту додавання стін. Додати двері за допомогою інструменту додавання дверей. Натиснути "Undo" двічі. Потім натиснути "Redo".	Двері додані до стіни
24	Відображення стіни у 3D	Додати стінку за допомогою інструменту додавання стін. Встановити шлях до текстури у властивостях стіни. Натиснути кнопку "3D".	Стіна присутня на 3D сцені з встановленою текстурою

Продовження таблиці 3.1

25	Відображення меблів у 3D	Додати мебель за допомогою інструменту додавання меблів. Встановити шлях до текстури у властивостях мебелі. Натиснути кнопку "3D".	Мебель присутня на 3D сцені з встановленою текстурою
26	Відображення вікон у 3D	Додати стінку за допомогою інструменту додавання стін. Додати вікно за допомогою інструменту додавання вікон. Встановити шлях до текстури у властивостях вікна. Натиснути кнопку "3D".	Вікно присутнє на 3D сцені з встановленою текстурою
27	Відображення дверей у 3D	Додати стінку за допомогою інструменту додавання стін. Додати двері за допомогою інструменту додавання дверей. Встановити шлях до текстури у властивостях дверей. Натиснути кнопку "3D".	Двері присутні на 3D сцені з встановленою текстурою

Продовження таблиці 3.1

28	Додавання ноди стінки	Додати стінку за допомогою інструменту додавання стін. Додати ноду за допомогою інструменту додавання нод.	Нова нода додалась на стінку
29	Undo додавання ноди стінки	Додати стінку за допомогою інструменту додавання стін. Додати ноду за допомогою інструменту додавання нод. Натиснути "Undo".	Додана нода зникла зі стінки
30	Redo додавання ноди стінки	Додати стінку за допомогою інструменту додавання стін. Додати ноду за допомогою інструменту додавання нод. Натиснути "Undo". Натиснути "Redo".	Нова нода додалась на стінку

Продовження таблиці 3.1

31	Undo рухання стінки	Додати стінку за допомогою інструменту додавання стін. Вибрати інструмент руху стінок, вибрати стінку та перетягнути її. Натиснути "Undo".	Стінка повернулася на початкове положення
32	Undo рухання меблів	Додати мебель за допомогою інструмента додавання меблів. Вибрати інструмент руху. Перетягнути мебель. Натиснути "Undo".	Мебель повернулася на початкове положення
33	Undo рухання нодів	Додати стінку за допомогою інструменту додавання стін. Вибрати інструмент руху нодів стінки, вибрати ноду та перетягнути її. Натиснути "Undo".	Нода повернулася на початкове положення

Продовження таблиці 3.1

34	Redo рухання стінки	Додати стінку за допомогою інструменту додавання стін. Вибрати інструмент руху стінок, вибрати стінку та перетягнути її. Натиснути "Undo". Натиснути "Redo".	Стінка змінила положення
35	Redo рухання меблів	Додати мебель за допомогою інструмента додавання меблів. Вибрати інструмент руху. Перетягнути мебель. Натиснути "Undo". Натиснути "Redo".	Мебель змінила положення.
36	Redo рухання нодів	Додати стінку за допомогою інструменту додавання стін. Вибрати інструмент руху нодів стінки, вибрати ноду та перетягнути її. Натиснути "Undo". Натиснути "Redo".	Нода змінила положення

Продовження таблиці 3.1

37	Збереження проекту	Додати стінку за допомогою інструменту додавання стін. Натиснути “File”. Натиснути “Save as”. У діалоговому вікні вказати шлях та назву файла. Натиснути “Save”.	За вказаним шляхом з'явився файл з вказаною назвою
38	Завантаження проекту	Додати стінку за допомогою інструменту додавання стін. Зберегти проект. Натиснути “File”. Натиснути “Open”. В діалоговому вікні вказати шлях до збереженого проекту. Натиснути “Open”.	Відкрився план з доданою на ньому стінкою.
39	Валідація властивостей стінки	Додати стінку за допомогою інструменту додавання стін. Встановити від'ємну висоту у властивостях стінки.	Значення висоти стінки не змінилося

Продовження таблиці 3.1

40	Валідація розташування меблів	Додати мебель за допомогою інструменту додавання меблів. Додати стінку за допомогою інструменту додавання стін, так щоб вона перетинала раніше додану мебель.	Мебель змінила свій колір на червоний.

3.3.2 Юніт-тестування

Також внутрішня логіка була протестована за допомогою бібліотеки від Майкрософт UnitTestCpp.

```
TEST_CLASS(WallPlan)
  {
  public:
    TEST_METHOD(CompareWall_ShouldTrue_WhenEqual)
      {
      auto node1 = std::make_shared<CoreData::WallNode>(CoreMath::Vector2D(2, 2), -1);
      auto node2 = std::make_shared<CoreData::WallNode>(CoreMath::Vector2D(7, 7), -1);
      CoreData::Wall wall1(node1, node2, 10, 0);
      CoreData::Wall wall2(node1, node2, 5,  1);

      Assert::IsTrue(wall1 == wall2);
      }

    TEST_METHOD(CompareWall_ShouldFalse_WhenNotEqual)
      {
      auto node1 = std::make_shared<CoreData::WallNode>(CoreMath::Vector2D(2, 2), -1);
      auto node2 = std::make_shared<CoreData::WallNode>(CoreMath::Vector2D(7, 7), -1);
      auto node3 = std::make_shared<CoreData::WallNode>(CoreMath::Vector2D(1, 2), -1);
      auto node4 = std::make_shared<CoreData::WallNode>(CoreMath::Vector2D(35, 15), -1);
      CoreData::Wall wall1(node1, node2, 10, 0);
      CoreData::Wall wall2(node3, node4, 10, 1);

      Assert::IsFalse(wall1 == wall2);
      }

    TEST_METHOD(CompareWall_ShouldFalse_WhenWallsHaveDifferentHeight)
      {
      auto node1 = std::make_shared<CoreData::WallNode>(CoreMath::Vector2D(2, 2), -1);
      auto node2 = std::make_shared<CoreData::WallNode>(CoreMath::Vector2D(7, 7), -1);
      CoreData::Wall wall1(node1, node2, 10, 0);
      CoreData::Wall wall2(node2, node1, 10, 1);

      Assert::IsTrue(wall1 == wall2);
      }
```

Рисунок 3.1 – Приклад юніт тестів з використанням UnitTestCpp.

3.4 Висновки до розділу

У данному розділі було описано тест план, згідно з якім має тестуватись програмне забезпечення. Наведено компоненти, що маютть бути протестовані, обов’язки тестувальника та можливі ризики.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для розгортання даного застосунку під платформу Windows спершу потрібно створити збірку. Це має відбуватись наступним чином:

- з офіційних сайтів завантажити та встановити: CMake, Qt, Visual Studio;
- зібрати проект за допомогою консолі, використовуючи консоль чи CMake GUI;
- відкрити проект в Visual studio та скомпілювати у .exe файл.

4.2 Робота з програмним забезпеченням

Способи та послідовності роботи з програмним забезпеченням описані більш детально у розділі "Керівництво користувача".

4.3 Висновки до розділу

У даному розділі наведено алгоритми створення збірки та її встановлення, наведено детальну інструкцію користувача.

ВИСНОВКИ

В рамках дипломного проекту розроблено програмне забезпечення для планування інтер’єру у 3D.

Тема дипломного проекту є актуальною, оскільки присвячена розробці програмного забезпечення для планування інтер’єру. Розробка може бути використана для швидкого прототипування та оцінки вигляду майбутнього житла. Також програмне забезпечення дозволяє задати кастомний вигляд будь-якого з елементів інтер’єру.

У розділі «Аналіз вимог до програмного забезпечення» було оглянуто предметну область, а саме існуюче програмне забезпечення конкурентів, та було досліджено Їх на предмет переваг та недоліків. Спираючись на отриману інформацію, було сформовано функціональні вимоги до програмного забезпечення, створено приблизний UI майбутнього додатку, також поставлено цілі, що мають бути досягнуті на момент завршення розробки.

У розділі «Моделювання та конструювання програмного забезпечення» було подано алгоритм дій користувача та реакцію програми на них. Було описано архітектуру програми, структуру даних, що формують параметри задач, зазначено допоміжні фреймворки, використані у процесі розробки. Було зазначено використані під час розробки мову програмування та фреймворки. Також проаналізовано переваги даних технологій, що роблять їх зручними для розробки данного програмного забезпечення.

У розділі «Аналіз якості та тестування програмного забезпечення» було описанно тестування розробленого програмного забепечення.

У розділі «Впровадження та супровід програмного забезпечення» було описано як отримати виконуваний файл з вихідного коду, а також інструкція користувача.

Результатом даного дипломного проекту є програмне забезпечення для планування будинку, що складається з вікна для редагування плану будівлі, та вікна для перегляду його у 3D.

Таким чином, розроблене програмне забезпечення вирішує всі поставлені перед ним задачі та відповідає поставленим вимогам.

ПЕРЕЛІК ПОСИЛАНЬ

1) AutoCad Archtecture [Електронний ресурс] – Режим доступу: https://autodesk.com

2) CMake [Електронний ресурс] – Режим доступу: https://cmake.org

3) Factory pattern [Електронний ресурс] – Режим доступу: https://tutorialspoint.com/design_pattern/factory_pattern.htm

4) Singleton pattern [Електронний ресурс] – Режим доступу: https://tutorialspoint.com/design_pattern/singleton_pattern.htm

5) Smart pointers [Електронний ресурс] – Режим доступу: https://en.cppreference.com/book/into/smart_pointers

6) Transformation matrix [Електронний ресурс] – Режим доступу: https://forum.patagames.com/posts/t501-What-Is-Transformation-Matrix-and-How-to-Use-It

7) Vector [Електронний ресурс] – Режим доступу: https://mathinsight.org/vector_introduction

8) Visual studio [Електронний ресурс] – Режим доступу: https://visualstudio.microsoft.com

9) Qt [Електронний ресурс] – Режим доступу: https://qt.io

10) MVC [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/mvc-design-pattern/

11) C++ [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/C%2B%2B

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1008308259

Дата перевірки:
16.06.2021 05:08:13 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
16.06.2021 06:55:16 EEST

ID користувача:
77149

Назва документа: **BogdanVV_bachelor_isz71_2**

Кількість сторінок: **152** Кількість слів: **21266** Кількість символів: **163086** Розмір файлу: **2.53 MB** ID файлу: **1008376190**

5.16%
Схожість

Найбільша схожість: **0.95%** з джерелом з Бібліотеки (ID файлу: **1008262349**)

1.16% Джерела з Інтернету	157	Сторінка 154
4.78% Джерела з Бібліотеки	332	Сторінка 155

14% Цитат

Цитати	56	Сторінка 156

Вилучення списку бібліографічних посилань вимкнене

80.4%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

Немає вилучених Інтернет-джерел

80.4% Вилученого тексту з Бібліотеки	1	Сторінка 156

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	14

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПЛАНУВАННЯ ІНТЕР’ЄРУ У 3D

Опис програми

КПІ.ІП-з7109.045490.04.13

“ПОГОДЖЕНО”

Керівник проєкту:

Л.А. Іванова

Нормоконтроль:

К.І. Ліщук

Виконавець:

В.В. Богдан

Київ – 2021 року

Тексти програмного коду

Програмне забезпечення для планування інтер'єру у 3D

(Найменування програми (документа))

DVD-R

(Вид носія даних)

90 арк, 90 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2021

Door.h:

```
#pragma once
#include "CoreDataApi.h"
#include "IDBObject.h"


#include "ParameterObjects\DoorParameters.h"

namespace CoreData
  {
  class CORE_DATA_API Door : public IDBObject
    {
    public:
      Door(long long i_shift, long long i_width, long long i_height, bool i_is_direction_up,
long long i_id, bool i_is_valid = true);
      Door(long long i_shift, long long i_width, long long i_height, bool i_is_direction_up,
bool i_is_valid = true);

      Door() = default;

      Door(const Door&) = delete;
      Door& operator=(const Door&) = delete;

      long long GetShift() const;
      long long GetWidth() const;
      long long GetHeight() const;
      bool GetIsDirectionUp() const;
      bool IsValid() const;

      DoorParameters getParameters() const;

      void SetShift(long long i_shift);
      void SetWidth(long long i_width);
      void SetHeight(long long i_height);
      void SetIsDirectionUp(bool i_direction);
      void SetValidity(bool i_is_valid) const;

      void Serialize(Serializer::ISerializer& i_serializer) override final;
      void Deserialize(Serializer::IDeserializer& i_serializer) override final;

      bool operator == (const Door &i_right);

    private:
      long long m_shift;
      long long m_width;
      long long m_height = 7;
      bool m_is_direction_up;
      mutable bool m_is_valid;
    };
  }
   },
  },
};
```

Door.cpp:

```
#include "stdh.h"
#include "Door.h"
#include "ParameterObjects\DoorParameters.h"

namespace CoreData
  {
```

```
Door::Door(long long i_shift, long long i_width, long long i_height, bool
i_is_direction_up, long long i_id, bool i_is_valid)
    : IDBObject(i_id)
    , m_shift(i_shift)
    , m_width(i_width)
    , m_height(i_height)
    , m_is_direction_up(i_is_direction_up)
    , m_is_valid(i_is_valid)
    {}

Door::Door(long long i_shift, long long i_width, long long i_height, bool
i_is_direction_up, bool i_is_valid)
    : m_shift(i_shift)
    , m_width(i_width)
    , m_height(i_height)
    , m_is_direction_up(i_is_direction_up)
    , m_is_valid(i_is_valid)
    {}

long long Door::GetShift() const
    {
    return m_shift;
    }

long long Door::GetWidth() const
    {
    return m_width;
    }

long long Door::GetHeight() const
    {
    return m_height;
    }

bool Door::GetIsDirectionUp() const
    {
    return m_is_direction_up;
    }

bool Door::IsValid() const
    {
    return m_is_valid;
    }

void Door::SetShift(long long i_shift)
    {
    m_shift = i_shift;
    }

void Door::SetWidth(long long i_width)
    {
    m_width = i_width;
    }


void Door::SetHeight(long long i_height)
    {
    m_height = i_height;
    }


void Door::SetIsDirectionUp(bool i_direction)
    {
    m_is_direction_up = i_direction;
    }
```

```
  void Door::SetValidity(bool i_is_valid) const
    {
    m_is_valid = i_is_valid;
    }

  void Door::Serialize(Serializer::ISerializer& i_serializer)
    {
    i_serializer.WriteStartTitle("Door");

    i_serializer.WriteLongLong("Shift", m_shift);
    i_serializer.WriteLongLong("Width", m_width);
    i_serializer.WriteLongLong("Height", m_height);
    i_serializer.WriteBool("IsDirectionUp", m_is_direction_up);
    i_serializer.WriteLongLong("ID", m_id);
    i_serializer.WriteBool("Is_Valid", m_is_valid);

    i_serializer.WriteEndTitle();
    }

  void Door::Deserialize(Serializer::IDeserializer& i_serializer)
    {
    m_shift = i_serializer.ReadLongLong("Shift");
    m_width = i_serializer.ReadLongLong("Width");
    m_height = i_serializer.ReadLongLong("Height");
    m_is_direction_up = i_serializer.ReadBool("IsDirectionUp");
    m_id = i_serializer.ReadLongLong("ID");
    m_is_valid = i_serializer.ReadBool("Is_Valid");
    }

  bool Door::operator==(const Door & i_right)
    {
    return m_shift == i_right.m_shift
      && m_width == i_right.m_width
      && m_height == i_right.m_height ;
    }

  DoorParameters Door::getParameters() const
    {
    return DoorParameters(m_shift, m_width, m_height, m_is_direction_up, m_is_valid, m_id);
    }
  }}
```

Furniture.h:

```
#pragma once
#include "CoreDataApi.h"
#include "IDBObject.h"
#include "IMoveRotate.h"
#include "FurnitureEnum.h"

#include <CoreMath/Vector2D.h>

namespace CoreData
  {
  class CORE_DATA_API Furniture : public IDBObject, public IMoveRotate
    {
    public:
     Furniture() = default;
     Furniture(const CoreMath::Vector2D& i_pos, double i_direction_angle, long long i_width,
long long i_length, long long i_height, FurnitureType i_type, bool i_valid);
     Furniture(const CoreMath::Vector2D& i_pos, double i_direction_angle, long long i_width,
long long i_length, long long i_height, FurnitureType i_type, bool i_valid, long long i_id);
```

```
    Furniture(const Furniture&) = delete;
    Furniture& operator=(const Furniture&) = delete;

    void Serialize(Serializer::ISerializer& i_serializer) override final;
    void Deserialize(Serializer::IDeserializer& i_serializer) override final;

    void SetWidth(long long i_width);
    void SetLength(long long i_length);
    void SetHeight(long long i_height);
    void SetType(FurnitureType i_type);
    void SetValid(const bool i_valid);
    void UpdateValid(const bool i_valid);

    long long GetWidth() const;
    long long GetLength() const;
    long long GetHeight() const;
    std::vector<CoreMath::Vector2D> GetEdgePoints() const;
    FurnitureType GetType() const;
    bool GetValid() const;

  private:
    long long           m_width;
    long long           m_length;
    long long           m_height;
    FurnitureType       m_type;
    bool                m_is_valid;
  };

  bool CORE_DATA_API operator==(const Furniture &i_furniture1, const Furniture
&i_furniture2);
  }
```

Furniture.cpp:

```
#include "stdh.h"
#include "Furniture.h"

namespace CoreData
  {
  Furniture::Furniture
  (
    const CoreMath::Vector2D& i_pos,
    double                    i_direction_angle,
    long long                 i_width,
    long long                 i_length,
    long long                 i_height,
    FurnitureType             i_type,
    bool                      i_valid
  )
    : IMoveRotate(i_pos, i_direction_angle)
    , m_width(i_width)
    , m_length(i_length)
    , m_height(i_height)
    , m_type(i_type)
    , m_is_valid(i_valid)
    {
    }

  Furniture::Furniture
  (
    const CoreMath::Vector2D& i_pos,
    double                    i_direction_angle,
    long long                 i_width,
```

```
    long long                   i_length,
    long long                   i_height,
    FurnitureType               i_type,
    bool                        i_valid,
    long long                   i_id
)
    : IDBObject(i_id)
    , IMoveRotate(i_pos, i_direction_angle)
    , m_width(i_width)
    , m_length(i_length)
    , m_height(i_height)
    , m_type(i_type)
    , m_is_valid(i_valid)
    {
    }

long long Furniture::GetWidth() const
    {
    return m_width;
    }

long long Furniture::GetLength() const
    {
    return m_length;
    }

long long Furniture::GetHeight() const
    {
    return m_height;
    }

std::vector<CoreMath::Vector2D> Furniture::GetEdgePoints() const
    {
    std::vector<CoreMath::Vector2D> points
      {
      m_position + CoreMath::RotateVec(CoreMath::Vector2D(-m_width / 2, -m_length / 2),
-m_direction_angle),
      m_position + CoreMath::RotateVec(CoreMath::Vector2D(m_width / 2, -m_length / 2),
-m_direction_angle),
      m_position + CoreMath::RotateVec(CoreMath::Vector2D(m_width / 2, m_length / 2),
-m_direction_angle),
      m_position + CoreMath::RotateVec(CoreMath::Vector2D(-m_width / 2, m_length / 2),
-m_direction_angle)
      };
    return points;
    }

FurnitureType Furniture::GetType() const
    {
    return m_type;
    }
bool Furniture::GetValid() const
    {
    return m_is_valid;
    }

void Furniture::SetWidth(long long i_width)
    {
    m_width = i_width;
    }

void Furniture::SetLength(long long i_lenght)
    {
    m_length = i_lenght;
    }
```

```
void Furniture::SetHeight(long long i_height)
  {
  m_height = i_height;
  }

void Furniture::SetType(FurnitureType i_type)
  {
  m_type = i_type;
  }

void Furniture::SetValid(bool i_valid)
  {
  m_is_valid = i_valid;
  }

void Furniture::UpdateValid(const bool i_valid)
  {
  m_is_valid &= i_valid;
  }

void Furniture::Serialize(Serializer::ISerializer& i_serializer)
  {
  i_serializer.WriteStartTitle("Furniture");
  i_serializer.WriteLongLong("Coor_x", m_position.m_x);
  i_serializer.WriteLongLong("Coor_y", m_position.m_y);
  i_serializer.WriteDouble("Direction_angle", m_direction_angle);
  i_serializer.WriteLongLong("Width", m_width);
  i_serializer.WriteLongLong("Lenght", m_length);
  i_serializer.WriteLongLong("Height", m_height);
  i_serializer.WriteInt("Type", static_cast<int>(m_type));
  i_serializer.WriteBool("Valid", m_is_valid);
  i_serializer.WriteLongLong("ID", m_id);
  i_serializer.WriteEndTitle();
  }

void Furniture::Deserialize(Serializer::IDeserializer& i_serializer)
  {
  m_position.m_x = i_serializer.ReadLongLong("Coor_x");
  m_position.m_y = i_serializer.ReadLongLong("Coor_y");
  m_direction_angle = i_serializer.ReadDouble("Direction_angle");
  m_width = i_serializer.ReadLongLong("Width");
  m_length = i_serializer.ReadLongLong("Lenght");
  m_height = i_serializer.ReadLongLong("Height");
  m_type = static_cast<FurnitureType>(i_serializer.ReadInt("Type"));
  m_is_valid = i_serializer.ReadBool("Valid");
  m_id = i_serializer.ReadLongLong("ID");
  }

bool operator==(const Furniture &i_furniture1, const Furniture &i_furniture2)
  {
  return i_furniture1.GetPosition() == i_furniture2.GetPosition();
  }
}
```

IDatabase.h:

```
#pragma once
#include "CoreDataApi.h"

#include <memory>

namespace CoreData
  {
```

```
  class IDBObject;
  class Wall;
  class Furniture;
  class WallNode;
  class Door;
  class Window;
  class Room;
  class CORE_DATA_API IDatabase
    {
    public:
      IDatabase() = default;
      virtual ~IDatabase() = default;

      virtual void Register(Wall*) = 0;
      virtual void Register(Furniture* ) = 0;
      virtual void Register(Room*) = 0;
      virtual void Register(
        Door*,
        std::weak_ptr<WallNode>,
        std::weak_ptr<WallNode>
      ) = 0;
      virtual void Register(
        Window*,
        std::weak_ptr<WallNode>,
        std::weak_ptr<WallNode>
      ) = 0;

      virtual bool Modify(long long i_id) = 0;
      virtual bool Unregister(long long i_id) = 0;
    };
  }
```

IDBObject.h:

```
#pragma once
#include "CoreDataApi.h"
#include "IDatabase.h"

#include <ImportExportLib/ISerializer.h>
#include <string>

#pragma warning(push)
#pragma warning(disable:4251)

namespace CoreData
  {
  class CORE_DATA_API IDBObject
    {
    public:
      IDBObject(long long i_id = -1);
      IDBObject(IDatabase *ip_owner, long long i_id = -1);
      virtual ~IDBObject();
      IDBObject(const IDBObject&) = delete;
      IDBObject& operator = (const IDBObject&) = delete;

      virtual void Serialize(Serializer::ISerializer& i_serializer) {};
      virtual void Deserialize(Serializer::IDeserializer& i_serializer) {};

      void SetOwner(IDatabase *ip_owner);
      IDatabase *GetOwner() const;
      long long GetId() const;

    protected:
```

```
      long long m_id;
      IDatabase *mp_owner;
      std::string m_texture_path;
    };
  }

#pragma warning(pop)
```

IDBObject.cpp:

```
#include "stdh.h"
#include "IDBObject.h"

namespace CoreData
  {
  IDBObject::IDBObject(long long i_id)
    : m_id(i_id)
    , mp_owner(nullptr)
    {}

  IDBObject::IDBObject(IDatabase *ip_owner, long long i_id)
    : mp_owner(ip_owner)
    , m_id(i_id)
    {}

  IDBObject::~IDBObject()
    {
    if(mp_owner)
      mp_owner->Unregister(m_id);
    }

  void CoreData::IDBObject::SetOwner(IDatabase *ip_owner)
    {
    mp_owner = ip_owner;
    }

  IDatabase* IDBObject::GetOwner() const
    {
    return mp_owner;
    }

  long long IDBObject::GetId() const
    {
    return m_id;
    }
  }
```

ObjectFactory.h:

```
#pragma once
#include "ParameterObjects/IParameterObject.h"
#include "FurnitureEnum.h"

#include <CoreMath/Vector2D.h>
#include "Wall.h"
#include "WallNode.h"
#include "Furniture.h"
#include "Door.h"
#include "Window.h"
#include "IDatabase.h"
#include "Room.h"
//#include <CoreData/ParameterObjects/RoomParameters.h>

namespace CoreData
  {
```

```
class ObjectFactory
  {
  public:
    std::shared_ptr<Wall> CreateWall(
      std::weak_ptr<CoreData::WallNode> i_begin,
      std::weak_ptr<CoreData::WallNode> i_end,
      long long i_height,
      long long i_id = -1);

    std::shared_ptr<WallNode> CreateWallNode(
      const CoreMath::Vector2D & i_position,
      long long i_id = -1);

    std::shared_ptr<Room> CreateRoom(
      const std::vector<CoreMath::Vector2D> &i_room_nodes,
      const std::vector <std::vector<CoreMath::Vector2D>> &i_internal_rooms,
      CoreData::ZoneType i_zone_type,
      long long i_id = -1);

    std::shared_ptr<Door> CreateDoor(
      long long i_shift,
      long long i_width,
      long long i_height,
      bool i_is_direction_up,
      long long i_id = -1,
      bool i_is_valid = true);

    std::shared_ptr<Window> CreateWindowOnPlan(
      long long i_left_shift,
      long long i_bottom_shift,
      long long i_width,
      long long i_height,
      long long i_begin_node_id,
      long long i_end_node_id,
      long long i_id = -1,
      bool i_is_valid = true);

    std::shared_ptr<Furniture> CreateFurniture(
      const CoreMath::Vector2D &i_pos,
      double i_direction_angle,
      long long i_width,
      long long i_length,
      long long i_height,
      FurnitureType i_type,
      bool i_valid,
      long long i_id = -1);

    void SetCurrentId(long long i_id);

  private:
    long long _GetFreeID();
  private:
    long long m_current_id = 0;
  };
}
```

ObjectFactory.cpp:

```
#include "stdh.h"
#include "ObjectFactory.h"
#include "Wall.h"
#include "WallNode.h"
#include "Furniture.h"
#include "Door.h"
#include "Window.h"
#include "Room.h"
```

```
#include "ParameterObjects\RoomParameters.h"

namespace CoreData
  {
  std::shared_ptr<Wall> ObjectFactory::CreateWall(
    std::weak_ptr<CoreData::WallNode> i_begin,
    std::weak_ptr<CoreData::WallNode> i_end,
    long long i_height,
    long long i_id)
    {
    std::shared_ptr<CoreData::Wall> p_wall = std::make_shared<Wall>(i_begin, i_end,
i_height, i_id == -1 ? _GetFreeID() : i_id);
    auto p_begin = p_wall->GetBeginNode();
    auto p_end = p_wall->GetEndNode();
    if (p_begin)
      p_begin->AppendWall(p_wall);
    if (p_end)
      p_end->AppendWall(p_wall);
    return p_wall;
    }

  std::shared_ptr<WallNode> ObjectFactory::CreateWallNode(
    const CoreMath::Vector2D & i_position,
    long long i_id)
    {
    return std::make_shared<WallNode>(i_position, i_id == -1 ? _GetFreeID() : i_id);
    }

  std::shared_ptr<Room> ObjectFactory::CreateRoom(
    const std::vector<CoreMath::Vector2D>& i_room_nodes,
    const std::vector<std::vector<CoreMath::Vector2D>>& i_internal_rooms,
    CoreData::ZoneType i_zone_type,
    long long i_id)
    {
    return std::make_shared<Room>(i_room_nodes, i_internal_rooms, i_zone_type, i_id == -1 ?
_GetFreeID() : i_id);
    }

  std::shared_ptr<Furniture> ObjectFactory::CreateFurniture(
    const CoreMath::Vector2D &i_pos,
    double i_direction_angle,
    long long i_width,
    long long i_length,
    long long i_height,
    FurnitureType i_type,
    bool i_valid,
    long long i_id)
    {
    return std::make_shared<Furniture>(i_pos, i_direction_angle, i_width, i_length,
i_height, i_type, i_valid, i_id == -1 ? _GetFreeID() : i_id);
    }

  void ObjectFactory::SetCurrentId(long long i_id)
    {
    m_current_id = i_id;
    }

  std::shared_ptr<Door> ObjectFactory::CreateDoor(
    long long i_shift,
    long long i_width,
    long long i_height,
    bool i_is_direction_up,
    long long i_id,
    bool i_is_valid)
    {
```

```
    return std::make_shared<Door>(i_shift, i_width, i_height, i_is_direction_up, i_id == -1
? _GetFreeID() : i_id, i_is_valid);
    }

  std::shared_ptr<Window> ObjectFactory::CreateWindowOnPlan(
    long long i_left_shift,
    long long i_bottom_shift,
    long long i_width,
    long long i_height,
    long long i_begin_node_id,
    long long i_end_node_id,
    long long i_id,
    bool i_is_valid)
    {
    return std::make_shared<Window>(i_bottom_shift, i_left_shift, i_width, i_height,
i_end_node_id, i_begin_node_id, i_id == -1 ? _GetFreeID() : i_id, i_is_valid);
    }

  long long ObjectFactory::_GetFreeID()
    {
    return m_current_id++;
    }
  }
```

Plan.h:

```
#pragma once
#include "CoreDataApi.h"
#include "IDBObject.h"
#include "ObjectFactory.h"
#include "FurnitureEnum.h"

#include <CoreMath/Vector2D.h>

#include "Wall.h"
#include "WallNode.h"
#include "Furniture.h"
#include "Door.h"
#include "Window.h"
#include "Room.h"
#include "ParameterObjects/RoomParameters.h"

#pragma warning(push)
#pragma warning(disable:4251)

namespace CoreData
  {
  class CORE_DATA_API Plan : public IDBObject
    {
    public:
      Plan() = default;
      Plan(const Plan&) = delete;
      void  operator=(const Plan&) = delete;

      long long AddNode(
        CoreMath::Vector2D i_position,
        long long i_id);

      long long AddWall(
        std::weak_ptr<WallNode> i_begin,
        std::weak_ptr<WallNode> i_end,
        long long i_height,
        long long i_id = -1);
```

```
long long AddWall(
  long long i_begin_id,
  long long i_end_id,
  long long i_height,
  long long i_id = -1);

long long AddWall(
  const CoreMath::Vector2D& i_begin,
  const CoreMath::Vector2D& i_end,
  long long i_height,
  long long i_snap_tolerance,
  long long i_id = -1);

long long AddDoor(
  long long i_shift,
  long long i_width,
  long long i_height,
  bool i_is_direction_up,
  long long i_begin_node_id,
  long long i_end_node_id,
  long long i_id = -1,
  bool i_is_valid = true);

long long AddWindow(
  long long i_left_shift,
  long long i_bottom_shift,
  long long i_width,
  long long i_height,
  long long i_begin_node_id,
  long long i_end_node_id,
  long long i_id = -1,
  bool i_is_valid = true);

long long AddFurniture(
  const CoreMath::Vector2D& i_pos,
  double i_direction_angle,
  long long m_width,
  long long m_length,
  long long m_height,
  FurnitureType i_type,
  bool i_valid,
  long long i_id = -1);

long long AddRoom(
  const std::vector < CoreMath::Vector2D > & i_room_nodes,
  const std::vector < std::vector < CoreMath::Vector2D > > &i_internal_rooms,
  CoreData::ZoneType i_zone_type,
  long long i_id = -1);

const std::vector<std::shared_ptr<WallNode>>&  GetWallNodes() const;
const std::vector<std::shared_ptr<Wall>>&      GetWalls() const;
const std::vector<std::shared_ptr<Furniture>>& GetFurniture() const;
std::vector<std::shared_ptr<Window>>      GetWindows() const;
const std::vector<std::shared_ptr<Room>>& GetRooms() const;

void SetIdToFactory(long long i_new_id);

void Clear();

void Serialize(Serializer::ISerializer& i_serializer) override final;
void Deserialize(Serializer::IDeserializer& i_serializer)  override final;

bool DeleteItemById(long long i_id);
```

```
        CoreMath::Vector2D GetPositionNodeByID(long long i_id) const;

        bool CanNodeBeAdded(
          CoreMath::Vector2D i_position,
          long long i_id) const;


      private:
        std::shared_ptr<WallNode> _GetNodeByID(long long i_id);
        std::shared_ptr<WallNode> _GetNodeUnderCursor(const CoreMath::Vector2D& i_position,
long long i_tolerance) const;

        ObjectFactory                           m_object_factory;
        std::vector<std::shared_ptr<WallNode>>  m_wall_nodes;
        std::vector<std::shared_ptr<Wall>>      m_walls;
        std::vector<std::shared_ptr<Furniture>> m_furniture;
        std::vector<std::shared_ptr<Room>>      m_rooms;
      };
    }

#pragma warning(pop)
```

Plan.cpp:

```
#include "stdh.h"
#include "Plan.h"
#include "Wall.h"
#include "WallNode.h"
#include "Furniture.h"
#include "Door.h"
#include "Window.h"
#include "Room.h"

namespace CoreData
  {

  bool Plan::CanNodeBeAdded(
    CoreMath::Vector2D i_position,
    long long i_id) const
    {
    for (auto& wall_node : m_wall_nodes)
      {
      if (wall_node->GetId() == i_id || wall_node->GetPosition() == i_position)
        return false;
      }
    return true;
    }

  long long Plan::AddNode(CoreMath::Vector2D i_position, long long i_id)
    {
    for (auto& wall_node : m_wall_nodes)
      {
```

```
    if (wall_node->GetId() == i_id || wall_node->GetPosition() == i_position)
      return wall_node->GetId();
    }
  auto node = m_object_factory.CreateWallNode(i_position, i_id);
  m_wall_nodes.push_back(node);
  return node->GetId();
  }

long long Plan::AddWall(
  long long i_begin_id,
  long long i_end_id,
  long long i_height,
  long long i_id)
  {
  auto begin_node = _GetNodeByID(i_begin_id);
  auto end_node = _GetNodeByID(i_end_id);
  if (!begin_node || !end_node || begin_node == end_node)
    return -1;

  for (auto &wall : begin_node->GetLinkedWalls())
    {
    if (wall.lock()->GetBeginNode()->GetId() == end_node->GetId() ||
      wall.lock()->GetEndNode()->GetId() == end_node->GetId())
      return -1;
    }

  return AddWall(begin_node, end_node, i_height, i_id);
  }

long long Plan::AddWall(
  std::weak_ptr<WallNode> i_begin,
  std::weak_ptr<WallNode> i_end,
  long long i_height,
  long long i_id)
  {
    std::shared_ptr<Wall> new_wall = m_object_factory.CreateWall(i_begin, i_end,
i_height, i_id);
  const long long id = new_wall->GetId();

  m_walls.push_back(new_wall);
  new_wall->SetOwner(GetOwner());
  GetOwner()->Register(new_wall.get());
  return id;
  }
```

```
long long Plan::AddWall(
  const CoreMath::Vector2D &i_begin,
  const CoreMath::Vector2D &i_end,
  long long i_height,
  long long i_snap_tolerance,
  long long i_id)
  {
        std::shared_ptr<WallNode> start_node = _GetNodeUnderCursor(i_begin,
i_snap_tolerance);
         std::shared_ptr<WallNode> end_node = _GetNodeUnderCursor(i_end,
i_snap_tolerance);

  if (!start_node)
   {
   start_node = m_object_factory.CreateWallNode(i_begin);
   m_wall_nodes.push_back(start_node);
   }
  if (!end_node)
   {
   end_node = m_object_factory.CreateWallNode(i_end);
   m_wall_nodes.push_back(end_node);
   }

  return AddWall(start_node, end_node, i_height, i_id);
  }

   long long Plan::AddFurniture(const CoreMath::Vector2D& i_pos, double
i_direction_angle, long long i_width, long long i_length, long long i_height,
FurnitureType i_type, bool i_valid, long long i_id)
  {
                              std::shared_ptr<Furniture> new_furniture =
m_object_factory.CreateFurniture(i_pos, i_direction_angle, i_width, i_length,
i_height, i_type, i_valid, i_id);
  m_furniture.push_back(new_furniture);
  new_furniture->SetOwner(GetOwner());
  GetOwner()->Register(new_furniture.get());
  return new_furniture->GetId();
  }


long long Plan::AddRoom(
  const std::vector<CoreMath::Vector2D>& i_room_nodes,
  const std::vector<std::vector<CoreMath::Vector2D>>& i_internal_rooms,
  CoreData::ZoneType i_zone_type,
  long long i_id)
```

```
    {
                                        std::shared_ptr<Room>          new_room          =
m_object_factory.CreateRoom(i_room_nodes, i_internal_rooms, i_zone_type, i_id);
    m_rooms.push_back(new_room);
    new_room->SetOwner(GetOwner());
    GetOwner()->Register(new_room.get());
    return new_room->GetId();
    }

  long long Plan::AddDoor(
    long long i_shift,
    long long i_width,
    long long i_height,
    bool i_is_direction_up,
    long long i_begin_node_id,
    long long i_end_node_id,
    long long i_id,
    bool i_is_valid)
    {
     std::shared_ptr<Door> new_door = m_object_factory.CreateDoor(i_shift, i_width,
i_height, i_is_direction_up, i_id, i_is_valid);
    new_door->SetOwner(GetOwner());

    for (auto &wall : m_walls)
      if (wall->GetWallNodesId() == std::make_pair(i_begin_node_id, i_end_node_id)
        || wall->GetWallNodesId() == std::make_pair(i_end_node_id, i_begin_node_id))
        {
        wall->AddDoor(new_door);
        break;
        }

            GetOwner()->Register(new_door.get(),    _GetNodeByID(i_begin_node_id),
_GetNodeByID(i_end_node_id));
    return new_door->GetId();
    }

  long long Plan::AddWindow(
    long long i_left_shift,
    long long i_bottom_shift,
    long long i_width,
    long long i_height,
    long long i_begin_node_id,
    long long i_end_node_id,
    long long i_id,
    bool i_is_valid)
```

```
  {
    {
          auto new_window = m_object_factory.CreateWindowOnPlan(i_left_shift,
i_bottom_shift, i_width, i_height, i_begin_node_id, i_end_node_id, i_id, i_is_valid);
    new_window->SetOwner(GetOwner());

    for (auto &wall : m_walls)
                  if (wall->GetWallNodesId() == std::make_pair(i_begin_node_id,
i_end_node_id)
                      || wall->GetWallNodesId() == std::make_pair(i_end_node_id,
i_begin_node_id))
        {
        wall->AddWindow(new_window);
        break;
        }

         GetOwner()->Register(new_window.get(), _GetNodeByID(i_begin_node_id),
_GetNodeByID(i_end_node_id));
    return new_window->GetId();
    }
  }


  void Plan::Clear()
    {
   m_rooms.clear();
   m_walls.clear();
   m_wall_nodes.clear();
   m_furniture.clear();
   }

  const std::vector<std::shared_ptr<WallNode>>& Plan::GetWallNodes() const
    {
   return m_wall_nodes;
   }

  const std::vector<std::shared_ptr<CoreData::Wall>>& Plan::GetWalls() const
    {
   return m_walls;
   }

    const std::vector<std::shared_ptr<CoreData::Furniture>>& Plan::GetFurniture()
const
    {
   return m_furniture;
```

```
}

std::vector<std::shared_ptr<Window>> Plan::GetWindows() const
 {
 std::vector<std::shared_ptr<CoreData::Window>> result;

 for (const auto& wall: GetWalls())
  {
  auto windows = wall->GetWindows();
  result.insert(result.end(), windows.begin(), windows.end());
  }
 return result;
 }

const std::vector<std::shared_ptr<Room>>& Plan::GetRooms() const
 {
 return m_rooms;
 }

void Plan::SetIdToFactory(long long i_new_id)
 {
 m_object_factory.SetCurrentId(i_new_id);
 }

void Plan::Serialize(Serializer::ISerializer& i_serializer)
 {
 i_serializer.WriteStartTitle("Plan");

 i_serializer.WriteStartTitle("WallNodes");
 for (auto& wall_node : m_wall_nodes)
  {
  wall_node->Serialize(i_serializer);
  }
 i_serializer.WriteEndTitle();

 i_serializer.WriteStartTitle("Walls");
 for (auto& wall : m_walls)
  {
  i_serializer.WriteStartTitle("Wall");

  i_serializer.WriteLongLong("Node_begin_id", wall->GetBeginNode()->GetId());
  i_serializer.WriteLongLong("Node_end_id", wall->GetEndNode()->GetId());
  i_serializer.WriteLongLong("Height", wall->GetHeight());
  i_serializer.WriteLongLong("ID", wall->GetId());
```

```
    i_serializer.WriteStartTitle("Windows");
    for (auto window: wall->GetWindows())
      {
      window->Serialize(i_serializer);
      }
    i_serializer.WriteEndTitle();

    i_serializer.WriteStartTitle("Doors");
    for (auto &door : wall->GetDoors())
      {
      door->Serialize(i_serializer);
      }
    i_serializer.WriteEndTitle();

    i_serializer.WriteEndTitle();
    }
  i_serializer.WriteEndTitle();

  i_serializer.WriteStartTitle("Furnitures");
  for (auto& furniture : m_furniture)
    {
    furniture->Serialize(i_serializer);
    }
  i_serializer.WriteEndTitle();

  i_serializer.WriteStartTitle("Rooms");
  for (auto& room : m_rooms)
    {
    room->Serialize(i_serializer);
    }
  i_serializer.WriteEndTitle();

  i_serializer.WriteEndTitle();
  }

void Plan::Deserialize(Serializer::IDeserializer& i_serializer)
  {
  std::map<long long, std::weak_ptr<WallNode>> nodes_by_id;

  if (i_serializer.GetTitle() == "Plan")
    {
    while (!i_serializer.atEnd())
      {
      if (i_serializer.GetTitle() == "WallNodes")
        {
```

```
        while (i_serializer.GetTitle() == "WallNode")
         {

m_wall_nodes.push_back(m_object_factory.CreateWallNode(CoreMath::Vector2D(0,
0)));
         m_wall_nodes.back()->Deserialize(i_serializer);
         nodes_by_id[m_wall_nodes.back()->GetId()] = m_wall_nodes.back();
         i_serializer.GetTitle();
         }
        }
       if (i_serializer.GetTitle() == "Walls")
        {
        while (i_serializer.GetTitle() == "Wall")
         {
                                              const long long wall_begin_id =
i_serializer.ReadLongLong("Node_begin_id");
         const long long wall_end_id = i_serializer.ReadLongLong("Node_end_id");
         const long long wall_height = i_serializer.ReadLongLong("Height");
         const long long wall_ID = i_serializer.ReadLongLong("ID");

         if (nodes_by_id.find(wall_begin_id) == nodes_by_id.end() ||
            nodes_by_id.find(wall_end_id)   == nodes_by_id.end())
          continue;

                     AddWall(nodes_by_id[wall_begin_id],  nodes_by_id[wall_end_id],
wall_height, wall_ID);

         if (i_serializer.GetTitle() == "Windows")
          {
          while (i_serializer.GetTitle() == "Window")
           {
           Window window;
           window.Deserialize(i_serializer);
           AddWindow(
            window.GetLeftShift(),
            window.GetBottomShift(),
            window.GetWidth(),
            window.GetHeight(),
            wall_begin_id,
            wall_end_id,
            window.GetId(),
            window.IsValid());
           i_serializer.GetTitle(); // /Window
           }
          }
```

```
    if (i_serializer.GetTitle() == "Doors")
      {
      while (i_serializer.GetTitle() == "Door")
        {
        Door door;
        door.Deserialize(i_serializer);
        AddDoor(
          door.GetShift(),
          door.GetWidth(),
          door.GetHeight(),
          door.GetIsDirectionUp(),
          wall_begin_id,
          wall_end_id,
          door.GetId(),
          door.IsValid());
        i_serializer.GetTitle(); // /Door
        }
      }
    i_serializer.GetTitle(); // /Wall
    }
  }
if (i_serializer.GetTitle() == "Furnitures")
  {
  while (i_serializer.GetTitle() == "Furniture")
    {
    Furniture furniture;
    furniture.Deserialize(i_serializer);
    AddFurniture(
      furniture.GetPosition(),
      furniture.GetDirectionAngle(),
      furniture.GetWidth(),
      furniture.GetLength(),
      furniture.GetHeight(),
      furniture.GetType(),
      furniture.GetValid(),
      furniture.GetId()
    );
    i_serializer.GetTitle();
    }
  }
if (i_serializer.GetTitle() == "Rooms")
  {
  while (i_serializer.GetTitle() == "Room")
    {
```

```
            Room room;
            room.Deserialize(i_serializer);
            AddRoom(
              room.GetRoomNodes(),
              room.GetInternalRoomNodes(),
              room.GetType(),
              room.GetId()
            );
            i_serializer.GetTitle(); //room
            }
          }
        }
      }
    else
      {
      throw std::invalid_argument("File isn`t valid! Please open another file!");
      }
    }

  bool Plan::DeleteItemById(long long i_id)
    {

          auto room_it = std::find_if(m_rooms.begin(), m_rooms.end(), [&](const
std::shared_ptr<Room> & ip_room)
      {
      return i_id == ip_room->GetId();
      });

    if (room_it != m_rooms.end())
      {
      m_rooms.erase(room_it);
      return true;
      }

    for (auto &wall : m_walls)
      {
                        auto door_it = std::find_if(wall.get()->GetDoors().begin(),
wall.get()->GetDoors().end(), [&](const std::shared_ptr<Door>& ip_door)
        {
        return i_id == ip_door->GetId();
        });
      if (door_it != wall->GetDoors().end())
        if (wall->EraseDoorById(door_it->get()->GetId()))
          return true;
```

```
                auto window_it = std::find_if(wall.get()->GetWindows().begin(),
wall.get()->GetWindows().end(), [&](const std::shared_ptr<Window>& ip_window)
      {
      return i_id == ip_window->GetId();
      });
    if (window_it != wall->GetWindows().end())
      if (wall->EraseWindowById(window_it->get()->GetId()))
        return true;

    }

          auto wall_it = std::find_if(m_walls.begin(), m_walls.end(), [&](const
std::shared_ptr<Wall>& ip_wall)
    {
    return i_id == ip_wall->GetId();
    });

   if (wall_it != m_walls.end())
    {
    std::shared_ptr<WallNode> wall_begin_node = (*wall_it)->GetBeginNode();
    std::shared_ptr<WallNode> wall_end_node = (*wall_it)->GetEndNode();

    for (int index = 0; index < m_rooms.size(); ++index)
      {
      auto &room = m_rooms[index];
                      auto node_it = std::find_if(room->GetRoomNodes().begin(),
room->GetRoomNodes().end(), [&](const CoreMath::Vector2D& ip_wall_node)
        {
         return (ip_wall_node == wall_begin_node->GetPosition()) || (ip_wall_node ==
wall_end_node->GetPosition());
        });
      if (node_it != room->GetRoomNodes().end())
        {
        --index;
        DeleteItemById(room->GetId());
        }
      }


    m_walls.erase(wall_it);

    if (wall_begin_node->GetLinkedWalls().size() == 0)
      DeleteItemById(wall_begin_node->GetId());

    if (wall_end_node->GetLinkedWalls().size() == 0)
```

```
        DeleteItemById(wall_end_node->GetId());
      return true;
      }

     auto node_it = std::find_if(m_wall_nodes.begin(), m_wall_nodes.end(), [&](const
std::shared_ptr<WallNode>& ip_wall_node)
      {
      return i_id == ip_wall_node->GetId();
      });
    if (node_it != m_wall_nodes.end())
      {
      m_wall_nodes.erase(node_it);
      return true;
      }

      auto furniture_it = std::find_if(m_furniture.begin(), m_furniture.end(), [&](const
std::shared_ptr<Furniture>& ip_furniture)
      {
      return i_id == ip_furniture->GetId();
      });
    if (furniture_it != m_furniture.end())
      {
      m_furniture.erase(furniture_it);
      return true;
      }

    return false;
    }

  std::shared_ptr<WallNode> Plan::_GetNodeByID(long long i_id)
    {
    for (const auto& wall_node : m_wall_nodes)
      {
      if (wall_node->GetId() == i_id)
        return wall_node;
      }
    return nullptr;
    }

                    std::shared_ptr<WallNode>          Plan::_GetNodeUnderCursor(const
CoreMath::Vector2D & i_position, long long i_tolerance) const
    {
    for (const auto& wall_node : m_wall_nodes)
      {
```

```
            if (CoreMath::DistanceSqr(wall_node->GetPosition(), i_position) <=
i_tolerance*i_tolerance)
        return wall_node;
      }
    return nullptr;
    }

  CoreMath::Vector2D Plan::GetPositionNodeByID(long long i_id) const
    {
    for (const auto& wall_node : m_wall_nodes)
      {
      if (wall_node->GetId() == i_id)
        return wall_node.get()->GetPosition();
      }
    return CoreMath::Vector2D();
    }
  }
```

Room.h:

```
#pragma once
#include "CoreDataApi.h"
#include "IDBObject.h"
#include "CoreMath/Vector2D.h"
#include "ParameterObjects\RoomParameters.h"

#pragma warning(push)
#pragma warning(disable:4251)

#include "Wall.h"
#include "WallNode.h"


namespace CoreMath
  {
  struct Vector2D;
  }

namespace CoreData
  {
  class CORE_DATA_API Room : public IDBObject
    {
    public:

      Room();

      Room(
        const std::vector<CoreMath::Vector2D> i_room_contour,
        const std::vector<std::vector<CoreMath::Vector2D>> i_internal_contours,
        ZoneType i_zone_type,
        long long i_id);

      void Serialize(Serializer::ISerializer& i_serializer) override final;
      void Deserialize(Serializer::IDeserializer& i_serializer) override final;

      const std::vector<CoreMath::Vector2D>& GetRoomNodes() const;
```

```
      const std::vector<std::vector<CoreMath::Vector2D>>& GetInternalRoomNodes() const;
      RoomParameters GetParameters() const;
      ZoneType GetType() const;
      void SetType(ZoneType i_type);

    private:

      void _WriteRoomInFile(Serializer::ISerializer& i_serializer, const
std::vector<CoreMath::Vector2D> i_room);
      std::vector<CoreMath::Vector2D> _ReadRoomFromFile(Serializer::IDeserializer&
i_serializer);

      std::vector<CoreMath::Vector2D>              m_outer_contour;
      std::vector<std::vector<CoreMath::Vector2D>>  m_internal_contours;
      ZoneType                                     m_zone_type;

    };
  }
```

Room.cpp:

```
#include "stdh.h"
#include "Room.h"
#include "Wall.h"
#include "WallNode.h"
#include "CoreMath/Vector2D.h"
#include "CoreLogic/Utility.h"

namespace CoreData
  {
  Room::Room()
    {
    m_zone_type = ZoneType::ECommonZone;
    }

  Room::Room(
    const std::vector<CoreMath::Vector2D> i_room_contour,
    const std::vector<std::vector<CoreMath::Vector2D>> i_internal_contours,
    ZoneType i_zone_type,
    long long i_id)
    : IDBObject(i_id)
    , m_outer_contour(i_room_contour)
    , m_internal_contours(i_internal_contours)
    , m_zone_type(i_zone_type)
    {}

  void Room::Serialize(Serializer::ISerializer& i_serializer)
    {
    i_serializer.WriteStartTitle("Room");
    i_serializer.WriteInt("Type", static_cast<int>(m_zone_type));

    _WriteRoomInFile(i_serializer, m_outer_contour);

    i_serializer.WriteStartTitle("Internal_rooms");
    i_serializer.WriteLongLong("Internal_rooms_count", m_internal_contours.size());
    for (const auto& internal_room : m_internal_contours)
      _WriteRoomInFile(i_serializer, internal_room);
    i_serializer.WriteEndTitle();

    i_serializer.WriteLongLong("ID", m_id);
    i_serializer.WriteEndTitle();
    }
```

```
  void Room::Deserialize(Serializer::IDeserializer& i_serializer)
    {
    m_zone_type = static_cast<ZoneType>(i_serializer.ReadInt("Type"));

    m_outer_contour.clear();
    m_outer_contour = _ReadRoomFromFile(i_serializer);

    if (i_serializer.GetTitle() == "Internal_rooms")
      {
      m_internal_contours.clear();
      auto internal_rooms_count = i_serializer.ReadLongLong("Internal_rooms_count");
      for (int i = 0; i < internal_rooms_count; i++)
        m_internal_contours.push_back(_ReadRoomFromFile(i_serializer));
      i_serializer.GetTitle();
      }

    m_id = i_serializer.ReadLongLong("ID");
    }

  const std::vector<CoreMath::Vector2D>& Room::GetRoomNodes() const
    {
    return m_outer_contour;
    }

  const std::vector<std::vector<CoreMath::Vector2D>>& Room::GetInternalRoomNodes() const
    {
    return m_internal_contours;
    }

  RoomParameters Room::GetParameters() const
    {
    return RoomParameters(m_outer_contour, m_internal_contours, m_zone_type, m_id);
    }

  ZoneType Room::GetType() const
    {
    return m_zone_type;
    }

  void Room::SetType(ZoneType i_type)
    {
    m_zone_type = i_type;
    }

  void Room::_WriteRoomInFile(Serializer::ISerializer& i_serializer, const
std::vector<CoreMath::Vector2D> i_room)
    {
    i_serializer.WriteLongLong("room_nodes_count", i_room.size());
    for (const auto& node:i_room)
      {
      i_serializer.WriteLongLong("node_x", node.m_x);
      i_serializer.WriteLongLong("node_y", node.m_y);
      }
    }

  std::vector<CoreMath::Vector2D> Room::_ReadRoomFromFile(Serializer::IDeserializer&
i_serializer)
    {
    std::vector<CoreMath::Vector2D> result;

    long long nodes_count = i_serializer.ReadLongLong("room_nodes_count");

    for (int i = 0; i < nodes_count; i++)
      {
      CoreMath::Vector2D node;
```

```
        node.m_x = i_serializer.ReadLongLong("node_x");
        node.m_y = i_serializer.ReadLongLong("node_y");
        result.emplace_back(node);
        }
      return (result);
      }

}
```

Wall.h:

```
#pragma once
#include "CoreDataApi.h"
#include "IDBObject.h"

#include <CoreMath/Vector2D.h>

#include "Door.h"
#include "Window.h"
#include "WallNode.h"

#pragma warning(push)
#pragma warning(disable:4251)

namespace CoreData
  {
  class CORE_DATA_API Wall : public IDBObject
    {
    public:
      Wall(std::weak_ptr<WallNode> i_node_1, std::weak_ptr<WallNode> i_node_2, long long
i_height, long long i_id);
      ~Wall();

      Wall(const Wall&) = delete;
      Wall& operator=(const Wall&) = delete;

      std::shared_ptr<WallNode> GetBeginNode() const;
      std::shared_ptr<WallNode> GetEndNode() const;
      CoreMath::Vector2D GetBegin() const;
      CoreMath::Vector2D GetEnd() const;
      CoreMath::Vector2D GetDirection() const;
      long long GetHeight() const;
      void SetHeight(long long i_height);

      void Serialize(Serializer::ISerializer& i_serializer) override final;
      void Deserialize(Serializer::IDeserializer& i_serializer) override final;

      std::pair<long long, long long> GetWallNodesId() const;
      void AddDoor(std::shared_ptr<Door> i_door);
      void AddWindow(std::shared_ptr<Window> i_window);

      bool EraseDoorById(long long i_id);
      bool EraseWindowById(long long i_id);

      const std::vector<std::shared_ptr<Door>> &GetDoors() const;
      const std::vector<std::shared_ptr<Window>> &GetWindows() const;



    private:
      std::weak_ptr<WallNode> m_begin_node;
      std::weak_ptr<WallNode> m_end_node;
      long long               m_height = 3000;
      std::vector<std::shared_ptr<Door>> m_doors;
```

```
    std::vector<std::shared_ptr<Window>> m_windows;
  };

  bool CORE_DATA_API operator==(const Wall &i_wall1, const Wall &i_wall2);
  bool CORE_DATA_API operator!=(const Wall &i_wall1, const Wall &i_wall2);
  }

#pragma warning(pop)
```

Wall.cpp:

```
#include "stdh.h"
#include "Wall.h"
#include "Door.h"
#include "Window.h"
#include "WallNode.h"

namespace CoreData
  {
  Wall::Wall(std::weak_ptr<WallNode> i_node_1, std::weak_ptr<WallNode> i_node_2, long long
i_height, long long i_id)
    : IDBObject(i_id)
    , m_begin_node(i_node_1)
    , m_end_node(i_node_2)
    , m_height(i_height)
    {
    }

  Wall::~Wall()
    {
    auto begin_node = GetBeginNode();
    if (begin_node)
      begin_node->UnlinkDeadWalls();

    auto end_node = GetEndNode();
    if (end_node)
      end_node->UnlinkDeadWalls();
    }

  std::shared_ptr<WallNode> Wall::GetBeginNode() const
    {
    return m_begin_node.lock();
    }

  std::shared_ptr<WallNode> Wall::GetEndNode() const
    {
    return m_end_node.lock();
    }

  CoreMath::Vector2D Wall::GetBegin() const
    {
    auto p_begin = m_begin_node.lock();
    if (!p_begin)
      return CoreMath::Vector2D(0, 0);
    return p_begin->GetPosition();
    }

  CoreMath::Vector2D Wall::GetEnd() const
    {
    auto p_end = m_end_node.lock();
    if (!p_end)
      return CoreMath::Vector2D(0, 0);
    return p_end->GetPosition();
```

```
  }

CoreMath::Vector2D Wall::GetDirection() const
  {
  return GetEnd() - GetBegin();
  }

long long Wall::GetHeight() const
  {
  return m_height;
  }

void CoreData::Wall::SetHeight(long long i_height)
  {
  m_height = i_height;
  }

const std::vector<std::shared_ptr<Window>> &Wall::GetWindows() const
  {
  return m_windows;
  }

  }

bool operator==(const Wall &i_wall1, const Wall &i_wall2)
  {
  return i_wall1.GetBegin() == i_wall2.GetBegin() &&
    i_wall1.GetEnd() == i_wall2.GetEnd() ||
    i_wall1.GetBegin() == i_wall2.GetEnd() &&
    i_wall1.GetEnd() == i_wall2.GetBegin();
  }

bool operator!=(const Wall &i_wall1, const Wall &i_wall2)
  {
  return !(i_wall1 == i_wall2);
  }

std::pair<long long, long long> Wall::GetWallNodesId() const
  {
  return std::make_pair(
    m_begin_node.lock().get()->GetId(),
    m_end_node.lock().get()->GetId());
  }

void Wall::AddDoor(std::shared_ptr<Door> i_door)
  {
  m_doors.push_back(i_door);
  }


void Wall::AddWindow(std::shared_ptr<Window> i_window)
  {
  m_windows.push_back(i_window);
  }

const std::vector<std::shared_ptr<Door>> &Wall::GetDoors() const
  {
  return m_doors;
  }

bool Wall::EraseDoorById(long long i_id)
  {
  auto is_exist = [&](std::shared_ptr<Door> &i_door)
    {
    return i_door->GetId() == i_id;
```

```
      };

    bool result = std::any_of(m_doors.begin(), m_doors.end(), is_exist);

    m_doors.erase(
      std::remove_if(m_doors.begin(), m_doors.end(), is_exist),
      m_doors.end());

    return result;
    }

  bool Wall::EraseWindowById(long long i_id)
    {
    auto is_exist = [&](std::shared_ptr<Window> &i_window)
      {
      return i_window->GetId() == i_id;
      };

    bool result = std::any_of(m_windows.begin(), m_windows.end(), is_exist);

    m_windows.erase(
      std::remove_if(m_windows.begin(), m_windows.end(), is_exist),
      m_windows.end());

    return result;
    }
  }
```

WallNode.h:

```
#pragma once
#include "CoreDataApi.h"
#include "IDBObject.h"

#include <CoreMath/Vector2D.h>

#include "Wall.h"

#pragma warning(push)
#pragma warning(disable:4251)

namespace CoreData
  {
  class CORE_DATA_API WallNode : public IDBObject
    {
    public:
      WallNode(const CoreMath::Vector2D& i_position, long long i_id);

      bool operator==(const WallNode& i_other) const;

      void AppendWall(std::weak_ptr<Wall> i_wall);
      void UnlinkDeadWalls();
      void SetPosition(const CoreMath::Vector2D& i_position);

      CoreMath::Vector2D GetPosition() const;
      const std::vector<std::weak_ptr<Wall>>& GetLinkedWalls() const;

      void Serialize(Serializer::ISerializer& i_serializer) override final;
      void Deserialize(Serializer::IDeserializer& i_serializer) override final;

    private:
      CoreMath::Vector2D m_position;
      std::vector<std::weak_ptr<Wall>> m_walls;
        };
  }
```

```
#pragma warning(pop)
```

WallNode.cpp:

```
#include "stdh.h"
#include "WallNode.h"

namespace CoreData
  {
  WallNode::WallNode(const CoreMath::Vector2D& i_position, long long i_id)
    : IDBObject(i_id)
    , m_position(i_position)
    {
    }

  void WallNode::AppendWall(std::weak_ptr<Wall> i_wall)
    {
    auto p_wall_to_add = i_wall.lock();
    if (!p_wall_to_add)
      return;

    for (auto& wall : m_walls)
      {
      auto p_wall = wall.lock();
      if (!p_wall)
        continue;
      if (p_wall == p_wall_to_add)
        return;
      }

    m_walls.push_back(i_wall);
    }

  void WallNode::UnlinkDeadWalls()
    {
    bool erased = false;
    do
      {
      erased = false;
      for (auto wall_it = m_walls.begin(); wall_it != m_walls.end(); wall_it++)
        {
        auto p_wall = (*wall_it).lock();
        if (p_wall)
          continue;

        erased = true;
        m_walls.erase(wall_it);
        break;
        }
      } while (erased);
    }

  void WallNode::SetPosition(const CoreMath::Vector2D & i_position)
    {
    m_position = i_position;
    }

  CoreMath::Vector2D WallNode::GetPosition() const
    {
    return m_position;
    }

  const std::vector<std::weak_ptr<Wall>>& WallNode::GetLinkedWalls() const
    {
```

```
    return m_walls;
    }

  bool WallNode::operator==(const WallNode& i_other) const
    {
    return &i_other == this;
    }

  void WallNode::Serialize(Serializer::ISerializer& i_serializer)
    {
    i_serializer.WriteStartTitle("WallNode");

    i_serializer.WriteLongLong("Coor_x", m_position.m_x);
    i_serializer.WriteLongLong("Coor_y", m_position.m_y);
    i_serializer.WriteLongLong("Node_id", GetId());

    i_serializer.WriteEndTitle();
    }

  void WallNode::Deserialize(Serializer::IDeserializer& i_serializer)
    {
    m_position.m_x = i_serializer.ReadLongLong("Coor_x");
    m_position.m_y = i_serializer.ReadLongLong("Coor_y");
    m_id = i_serializer.ReadLongLong("Node_id");
    }
  }
```

Window.h:

```
#pragma once
#include "CoreDataApi.h"
#include "IDBObject.h"

namespace CoreData
  {
  class WindowParameters;

  class CORE_DATA_API Window : public IDBObject
    {
    public:
      Window(long long i_bottom_shift, long long i_left_shift, long long i_width, long long
i_height, long long i_end_node_id, long long i_begin_node_id, long long i_id, bool is_valid
= true);
      Window(long long i_bottom_shift, long long i_left_shift, long long i_width, long long
i_height, long long i_end_node_id, long long i_begin_node_id, bool is_valid = true);

      Window() = default;

      Window(const Window&) = delete;
      Window& operator=(const Window&) = delete;

      long long GetLeftShift() const;
      long long GetBottomShift() const;
      long long GetWidth() const;
      long long GetHeight() const;
      long long GetBeginNodeId() const;
      long long GetEndNodeId() const;
      bool IsValid() const;
      WindowParameters getParameters() const;

      void SetLeftShift(long long i_left_shift);
      void SetBottomShift(long long i_bottom_shift);
      void SetWidth(long long i_width);
```

```
      void SetHeight(long long i_height);
      void SetBeginNodeId(long i_begin_node_id);
      void SetEndNodeId(long i_end_node_id);
      void SetValidity(bool i_is_valid) const;

      void Serialize(Serializer::ISerializer& i_serializer) override final;
      void Deserialize(Serializer::IDeserializer& i_serializer) override final;

      bool operator == (const Window &i_right);

    private:
      long long m_left_shift;
      long long m_bottom_shift;
      long long m_width;
      long long m_height;
      long long m_end_node_id;
      long long m_begin_node_id;
      mutable bool m_is_valid;
    };
  }
```

Window.cpp:

```
#include "stdh.h"
#include "Window.h"
#include "ParameterObjects\WindowParameters.h"

namespace CoreData
  {
  Window::Window(long long i_bottom_shift, long long i_left_shift, long long i_width, long
long i_height, long long i_end_node_id, long long i_begin_node_id, long long i_id, bool
i_is_valid)
    : IDBObject(i_id)
    , m_left_shift(i_left_shift)
    , m_bottom_shift(i_bottom_shift)
    , m_width(i_width)
    , m_height(i_height)
    , m_begin_node_id(i_begin_node_id)
    , m_end_node_id(i_end_node_id)
    , m_is_valid(i_is_valid)
    {}

  Window::Window(long long i_bottom_shift, long long i_left_shift, long long i_width, long
long i_height, long long i_end_node_id, long long i_begin_node_id, bool i_is_valid)
    : m_left_shift(i_left_shift)
    , m_bottom_shift(i_bottom_shift)
    , m_width(i_width)
    , m_height(i_height)
    , m_begin_node_id(i_begin_node_id)
    , m_end_node_id(i_end_node_id)
    , m_is_valid(i_is_valid)
    {}

  long long Window::GetLeftShift() const
    {
    return m_left_shift;
    }

  long long Window::GetBottomShift() const
    {
    return m_bottom_shift;
    }

  long long Window::GetWidth() const
```

```
  {
  return m_width;
  }

long long Window::GetHeight() const
  {
  return m_height;
  }

long long Window::GetBeginNodeId() const
  {
  return  m_begin_node_id;
  }

long long Window::GetEndNodeId() const
  {
  return m_end_node_id;
  }

bool Window::IsValid() const
  {
  return m_is_valid;
  }

void Window::SetLeftShift(long long i_left_shift)
  {
  m_left_shift = i_left_shift;
  }

void Window::SetBottomShift(long long i_bottom_shift)
  {
  m_bottom_shift = i_bottom_shift;
  }

void Window::SetWidth(long long i_width)
  {
  m_width = i_width;
  }


void Window::SetHeight(long long i_height)
  {
  m_height = i_height;
  }

void Window::SetBeginNodeId(long i_begin_node_id)
  {
  m_begin_node_id = i_begin_node_id;
  }

void Window::SetEndNodeId(long i_end_node_id)
  {
  m_end_node_id = i_end_node_id;
  }

void Window::SetValidity(bool i_is_valid) const
  {
  m_is_valid = i_is_valid;
  }

void Window::Serialize(Serializer::ISerializer& i_serializer)
  {
  i_serializer.WriteStartTitle("Window");

  i_serializer.WriteLongLong("Left_Shift", m_left_shift);
```

```
    i_serializer.WriteLongLong("Bottom_Shift", m_bottom_shift);
    i_serializer.WriteLongLong("Width", m_width);
    i_serializer.WriteLongLong("Height", m_height);
    i_serializer.WriteLongLong("Begin_Node_Id", m_begin_node_id);
    i_serializer.WriteLongLong("End_Node_Id", m_end_node_id);
    i_serializer.WriteLongLong("Window_Id", m_id);
    i_serializer.WriteBool("Is_Valid", m_is_valid);

    i_serializer.WriteEndTitle();
    }

  void Window::Deserialize(Serializer::IDeserializer& i_serializer)
    {
    m_left_shift = i_serializer.ReadLongLong("Left_Shift");
    m_bottom_shift = i_serializer.ReadLongLong("Bottom_Shift");
    m_width = i_serializer.ReadLongLong("Width");
    m_height = i_serializer.ReadLongLong("Height");
    m_begin_node_id = i_serializer.ReadLongLong("Begin_Node_Id");
    m_end_node_id = i_serializer.ReadLongLong("End_Node_Id");
    m_id = i_serializer.ReadLongLong("Window_Id");
    m_is_valid = i_serializer.ReadBool("Is_Valid");
    }

  bool Window::operator==(const Window & i_right)
    {
    return m_left_shift == i_right.m_left_shift
      && m_bottom_shift == i_right.m_bottom_shift
      && m_width == i_right.m_width
      && m_height == i_right.m_height;
    }

  WindowParameters Window::getParameters() const
    {
    return WindowParameters(m_width, m_height, m_left_shift, m_bottom_shift, m_is_valid,
m_id);
    }
  }
```

Canvas.h:

```
#pragma once

class QGraphicsSceneMouseEvent;
class QGraphicsPixmapItem;

class Canvas : public QGraphicsScene
  {
  Q_OBJECT
  public:
    Canvas(QObject *ip_parent = nullptr);

    void mousePressEvent(QGraphicsSceneMouseEvent* ip_mouse_event) override final;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent* ip_mouse_event) override final;
    void mouseMoveEvent(QGraphicsSceneMouseEvent* ip_mouse_event) override final;
    void mouseDoubleClickEvent(QGraphicsSceneMouseEvent* ip_mouse_event) override final;

    void drawBackground(QPainter* ip_painter, const QRectF &i_rect) override final;

  private:
    QGraphicsPixmapItem *mp_background = nullptr;
  };
```

Canvas.cpp:

```
#include "stdh.h"
```

```
#include "Canvas.h"
#include "CanvasMouseController.h"
#include "ScaleManager.h"
#include "Settings.h"

Canvas::Canvas(QObject* ip_parent)
  : QGraphicsScene(ip_parent)
  {}

void Canvas::mousePressEvent(QGraphicsSceneMouseEvent* ip_mouse_event)
  {
  CanvasMouseController::GetInstance().ProcessMousePressEvent(*ip_mouse_event);
  }

void Canvas::mouseReleaseEvent(QGraphicsSceneMouseEvent* ip_mouse_event)
  {
  CanvasMouseController::GetInstance().ProcessMouseReleaseEvent(*ip_mouse_event);
  }

void Canvas::mouseMoveEvent(QGraphicsSceneMouseEvent* ip_mouse_event)
  {
  CanvasMouseController::GetInstance().ProcessMouseMoveEvent(*ip_mouse_event);
  }

void Canvas::mouseDoubleClickEvent(QGraphicsSceneMouseEvent* ip_mouse_event)
  {
  CanvasMouseController::GetInstance().ProcessMouseDoubleClickEvent(*ip_mouse_event);
  }

void Canvas::drawBackground(QPainter *ip_painter, const QRectF &i_rect)
  {
  QGraphicsScene::drawBackground(ip_painter, i_rect);

  ip_painter->fillRect(sceneRect(), Qt::white);

  QPixmap image = backgroundBrush().texture();
  if (!image.isNull())
    {
    image = image.transformed(QTransform(1, 0, 0, -1, 0, 0));
    ip_painter->drawPixmap(-image.width() / 2, -image.height() / 2, image);
    }

  // Drawing Grid
  int grid_step = Settings::GetInstance().GetGridStep();
  qreal scale_value = ScaleManager::GetInstance().GetScaleValue();

  long long left = i_rect.left() - static_cast<long long>(i_rect.left()) % grid_step;
  long long top = i_rect.top() - static_cast<long long>(i_rect.top()) % grid_step;
  long long right = i_rect.right();
  long long bottom = i_rect.bottom();

  QPen grid_pen(Qt::darkGray);
  grid_pen.setWidthF(scale_value);
  ip_painter->setPen(grid_pen);

  if (grid_step / scale_value >= 5)
    {
    QVector<QPoint> grid_points;
    for (long long x = left; x <= right; x += grid_step)
      for (long long y = top; y <= bottom; y += grid_step)
        grid_points.append(QPoint(x, y));

    ip_painter->drawPoints(grid_points.data(), grid_points.size());
    }
```

```
    // Drawing Major Grid
    int grid_step_major = grid_step * Settings::GetInstance().GetGridStepMajor();
    left = i_rect.left() - static_cast<long long>(i_rect.left()) % grid_step_major;
    top = i_rect.top() - static_cast<long long>(i_rect.top()) % grid_step_major;

    QPen grid_major_pen(Qt::black);
    grid_major_pen.setWidthF(scale_value * 2);
    if (grid_step_major / scale_value < 10)
      grid_major_pen.setWidthF(scale_value);
    ip_painter->setPen(grid_major_pen);

    if (grid_step_major / scale_value >= 5)
      {
      QVector<QPoint> grid_points;
      for (long long x = left; x <= right; x += grid_step_major)
        for (long long y = top; y <= bottom; y += grid_step_major)
          grid_points.append(QPoint(x, y));

      ip_painter->drawPoints(grid_points.data(), grid_points.size());
      }

    grid_pen.setColor(Qt::green);
    ip_painter->setPen(grid_pen);
    ip_painter->drawRect(sceneRect());

    grid_pen.setColor(Qt::red);
    ip_painter->setPen(grid_pen);
    ip_painter->drawLine(QLineF(0, 0, width() / 2, 0));
    grid_pen.setColor(Qt::blue);

    ip_painter->setPen(grid_pen);
    ip_painter->drawLine(QLineF(0, 0, 0, height() * 2));
    }
```

```
#pragma once
#include "ui_MainWindow.h"
#include "Canvas.h"

class QTimer;
class BaseProperties;
class MainWidget;

class MainWindow : public QMainWindow
  {
  Q_OBJECT

  public:
    MainWindow(QMainWindow *ip_parent = nullptr);
    ~MainWindow();

    void closeEvent(QCloseEvent *ip_event) override;
    void keyPressEvent(QKeyEvent *i_event) override;

  public slots:
    void ToggleShowStatusBarAction();
    void SaveAsPlan();
    void OpenPlan();
    void SavePlan();
    void NewPlan();
    void OpenPreferences();
    void ShowHotkeysHelp();
    void ShowLog();
    void SavePlanAsImage();
```

```
  private:
    void _SetupActionTooltips();
    void _ConnectTools();
    void _ConnectActions();
    void _InitializeProperties();
    void _ConnectTreeView();
    void _OpenPlan();
    void _ClearUndoRedo();
    void _UncheckAllButtons();
    void _OpenWindow3D();
    void _EditLog();
    void _ReselectObjects();
    int _GetExitDecision();
    std::string _GetPathToOpenFileFromDialog();

  private:

    Ui::MainWindow m_ui;
    Canvas m_canvas;
    std::string m_file_name;

    std::string m_file_name_for_auto_save;
  };
```

MainWindow.cpp:

```
#include "stdh.h"
#include "MainWindow.h"
#include "MainWindowController.h"
#include "CanvasMouseController.h"
#include "Mouses/IMouse.h"
#include "Settings.h"
#include "SettingsDialog.h"
#include "Logger.h"
#include "PropertiesController.h"
#include "UndoRedoManagerController.h"
#include "Project.h"

#include <Viewer3D/Viewer3D.h>
#include <CoreData/Plan.h>
#include <CoreData/Room.h>

MainWindow::MainWindow(QMainWindow *ip_parent)
  : QMainWindow(ip_parent)
  , m_canvas(this)
  , m_file_name("")
  {
  m_ui.setupUi(this);
  Project::GetInstance().SetScene(&m_canvas);

  m_ui.m_canvas_view->setScene(&m_canvas);
```

```
  m_canvas.setSceneRect(-100'000'000, -100'000'000, 200'000'000, 200'000'000);
  m_ui.m_canvas_view->setAlignment(Qt::AlignTop | Qt::AlignLeft);


CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);

  _ConnectTools();
  _ConnectActions();
  connect(&MainWindowController::GetInstance(),
    &MainWindowController::StatusBarUpdateSignal,
    m_ui.m_statusbar,
    &QStatusBar::showMessage);
                connect(m_ui.m_action_view_3d,          &QAction::triggered,          this,
&MainWindow::_OpenWindow3D);

  _ClearUndoRedo();
  m_ui.m_dock_log->hide();
  _InitializeProperties();
  _ConnectTreeView();
  _SetupActionTooltips();

  restoreState(Settings::GetInstance().GetDocksState());
  }

MainWindow::~MainWindow()
  {
  Settings::GetInstance().SetDocksState(saveState());

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ENon
e);
  Project::GetInstance().ClearPlan();
  }

void MainWindow::_SetupActionTooltips()
  {
  for (auto& action : findChildren<QAction*>())
    {
    if (action->shortcut().toString() != "")
      action->setToolTip(action->toolTip() + " (" + action->shortcut().toString() + ")");
    }
  }

void MainWindow::closeEvent(QCloseEvent *ip_event)
```

```
  {
  if (Project::GetInstance().IsPlanEmpty())
    {
    ip_event->accept();
    }
  else
    {
    int pressed = _GetExitDecision();
    if (pressed == QMessageBox::Cancel)
      {
      ip_event->ignore();
      }
    else
      {
      if(pressed== QMessageBox::Yes)
        SavePlan();
      ip_event->accept();
      }
    }
  }

void MainWindow::keyPressEvent(QKeyEvent *i_event)
  {
  if (i_event->key() == Qt::Key_Escape)
    {
    _UncheckAllButtons();
    PropertiesController::GetInstance().Deactivate();

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    }
  }

void MainWindow::_ConnectTools()
  {
  connect(m_ui.m_action_add_vertex,
    &QAction::triggered,
    [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
      m_ui.m_action_add_vertex->setChecked(i_checked);
```

```
CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EAdd
Vertex);
      }
    else
      {

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
      }
    });

  connect(m_ui.m_action_move_vertex,
    &QAction::triggered,
    [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
     m_ui.m_action_move_vertex->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EMov
eVertex);
      }
    else
      {

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
      }
    });

  connect(m_ui.m_action_move_wall,
    &QAction::triggered,
    [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
     m_ui.m_action_move_wall->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EMov
eWall);
      }
    else
```

```
      {

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
      }
    });

  connect(m_ui.m_action_add_wall,
    &QAction::triggered,
    [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
     m_ui.m_action_add_wall->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EAdd
Wall);
      }
    else

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    });

  connect(m_ui.m_action_add_polywall,
    &QAction::triggered,
    [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
     m_ui.m_action_add_polywall->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EAdd
Polyline);
      }
    else

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    });

  connect(m_ui.m_action_add_rectangle,
    &QAction::triggered,
```

```
    [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
      m_ui.m_action_add_rectangle->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EAdd
Rectangle);
      }
    else

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    });

  connect(m_ui.m_action_remove, &QAction::triggered, [this]
    {
    _UncheckAllButtons();
    Project::GetInstance().EraseObjects();

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    });

  connect(m_ui.m_action_add_furniture,
    &QAction::triggered,
    [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
      m_ui.m_action_add_furniture->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EAdd
Furniture);
      }
    else

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    });

  connect(m_ui.m_action_add_door,
    &QAction::triggered,
```

```
    [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
      m_ui.m_action_add_door->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EAdd
Door);
      }
    else

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    });

    connect(m_ui.m_action_move_object,
      &QAction::triggered,
      [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
      m_ui.m_action_move_object->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EMov
eObject);
      }
    else

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    });

  connect(m_ui.m_action_rotate_object,
    &QAction::triggered,
    [this](bool i_checked)
    {
    _UncheckAllButtons();
    if (i_checked)
      {
      m_ui.m_action_rotate_object->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ERota
teObject);
```

```
    }
   else

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
   });

  connect(m_ui.m_action_add_window,
   &QAction::triggered,
   [this](bool i_checked)
   {
   _UncheckAllButtons();
   if (i_checked)
    {
    m_ui.m_action_add_window->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EAdd
Window);
    }
   else

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
   });

  connect(m_ui.m_action_measurements,
   &QAction::triggered,
   [this](bool i_checked)
   {
   _UncheckAllButtons();
   if (i_checked)
    {
    m_ui.m_action_measurements->setChecked(i_checked);

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::EMea
surement);
    }
   else

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
   });
```

```
        connect(m_ui.m_action_show_status_bar, &QAction::changed, this,
&MainWindow::ToggleShowStatusBarAction);

  }

void MainWindow::_ConnectActions()
  {

     connect(m_ui.m_action_new_plan,                &QAction::triggered,  this,
&MainWindow::NewPlan);
      connect(m_ui.m_action_save_file,              &QAction::triggered,  this,
&MainWindow::SavePlan);
     connect(m_ui.m_action_save_as,                 &QAction::triggered,  this,
&MainWindow::SaveAsPlan);
      connect(m_ui.m_action_open_file,              &QAction::triggered,  this,
&MainWindow::OpenPlan);
      connect(m_ui.m_action_show_status_bar,        &QAction::changed,    this,
&MainWindow::ToggleShowStatusBarAction);
        connect(m_ui.m_action_close_application,    &QAction::triggered,  this,
&MainWindow::close);
     connect(m_ui.m_action_hotkeys,                 &QAction::triggered,  this,
&MainWindow::ShowHotkeysHelp);
      connect(m_ui.m_action_preferences,            &QAction::triggered,  this,
&MainWindow::OpenPreferences);
     connect(m_ui.m_action_log,                     &QAction::triggered,  this,
&MainWindow::ShowLog);
     connect(m_ui.m_save_as_image,                  &QAction::triggered,  this,
&MainWindow::SavePlanAsImage);
     connect(&Logger::GetInstance(),                &Logger::changed,     this,
&MainWindow::_EditLog);
  connect(m_ui.m_action_undo,          &QAction::triggered, [&]()
    {
    UndoRedoManagerController::GetInstance().Undo();
    MainWindowController::GetInstance().UpdateObjectTree();
    });
  connect(m_ui.m_action_redo,          &QAction::triggered, [&]()
    {
    UndoRedoManagerController::GetInstance().Redo();
    MainWindowController::GetInstance().UpdateObjectTree();
    });

                                connect(&UndoRedoManagerController::GetInstance(),
&UndoRedoManager::UndoChangedSignal,
    [this](bool i_new_state)
    {
```

```
    m_ui.m_action_undo->setEnabled(i_new_state);
    });

                                    connect(&UndoRedoManagerController::GetInstance(),
&UndoRedoManager::RedoChangedSignal,
    [this](bool i_new_state)
    {
    m_ui.m_action_redo->setEnabled(i_new_state);
    });
  }

void MainWindow::_InitializeProperties()
  {
  PropertiesController::GetInstance().SetPropertiesFrame(m_ui.m_prop_container);
  }

void MainWindow::_ConnectTreeView()
  {
  MainWindowController::GetInstance().UpdateObjectTree();

m_ui.m_tree_view_project->setModel(MainWindowController::GetInstance().GetTre
eModel().get());

m_ui.m_tree_view_project->setSelectionModel(MainWindowController::GetInstance
().GetSelectionModel().get());

m_ui.m_tree_view_project->setSelectionBehavior(QAbstractItemView::SelectRows)
;

m_ui.m_tree_view_project->setSelectionMode(QAbstractItemView::ExtendedSelecti
on);
  m_ui.m_tree_view_project->expandAll();
  connect(m_ui.m_tree_view_project->selectionModel(),
                                    &QItemSelectionModel::selectionChanged,  this,
&MainWindow::_ReselectObjects);
  }

void MainWindow::ToggleShowStatusBarAction()
  {
  if (m_ui.m_action_show_status_bar->isChecked())
    m_ui.m_statusbar->show();
  else
    m_ui.m_statusbar->hide();
  }
```

```
void MainWindow::NewPlan()
  {
  Project::GetInstance().ClearPlan();
  m_file_name.clear();
  m_file_name_for_auto_save.clear();
  _ClearUndoRedo();
  MainWindowController::GetInstance().UpdateObjectTree();
  Logger::GetInstance().WriteToLog("\tNew plan created");
  }

void MainWindow::SaveAsPlan()
  {
  QString file_name = QFileDialog::getSaveFileName(this,
    tr("Save Plan"), "",
    tr("PLN Files (*.pln);;All Files (*)"));
  m_file_name = file_name.toLocal8Bit().constData();
  if (!m_file_name.empty())
    {
    Project::GetInstance().SavePlan(m_file_name);
    m_ui.m_statusbar->showMessage("Saved");
    _ClearUndoRedo();
    }
  }

void MainWindow::SavePlan()
  {
  if (!m_file_name.empty())
    {
    Project::GetInstance().SavePlan(m_file_name);
    m_ui.m_statusbar->showMessage("Saved");
    _ClearUndoRedo();
    }
  else
    SaveAsPlan();
  }


void MainWindow::OpenPlan()
  {
  if (Project::GetInstance().IsPlanEmpty())
    {
    _OpenPlan();
    }
  else
    {
```

```
    int pressed = _GetExitDecision();
    if (!(pressed == QMessageBox::Cancel))
      {
      if (pressed == QMessageBox::Yes)
        SavePlan();
      _OpenPlan();
      }
    }
  }

void MainWindow::_OpenPlan()
  {
  QString file_name = QFileDialog::getOpenFileName(this,
    tr("Open Plan"), "",
    tr("PLN Files (*.pln);;All Files (*)"));
  std::string path_to_file = file_name.toLocal8Bit().constData();
  if (path_to_file.empty()) return;
  m_file_name = path_to_file;
  try
    {
    _UncheckAllButtons();

CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    _ClearUndoRedo();
    Project::GetInstance().OpenPlan(path_to_file);
    }
  catch (std::invalid_argument &err)
    {
    m_file_name = "";
    QString i_message = err.what();
    QMessageBox message;
    message.setText(i_message);
    message.exec();
    }
  }

void MainWindow::_UncheckAllButtons()
  {
  m_ui.m_action_add_wall->setChecked(false);
  m_ui.m_action_add_polywall->setChecked(false);
  m_ui.m_action_add_rectangle->setChecked(false);
  m_ui.m_action_move_vertex->setChecked(false);
  m_ui.m_action_move_wall->setChecked(false);
  m_ui.m_action_add_vertex->setChecked(false);
```

```
  m_ui.m_action_add_furniture->setChecked(false);
  m_ui.m_action_add_window->setChecked(false);
  m_ui.m_action_measurements->setChecked(false);
  m_ui.m_action_add_door->setChecked(false);
  m_ui.m_action_move_object->setChecked(false);
  m_ui.m_action_rotate_object->setChecked(false);
  }

void MainWindow::_OpenWindow3D()
  {
  Project& project = Project::GetInstance();
  long long selected_cam_id = -1;

  try
    {
    Viewer3D viewer3D(project.GetPlan(), this);
    viewer3D.SetWallTexturePath(Settings::GetInstance().GetPathToWallTexture());
    viewer3D.SetFloorTexturePath(Settings::GetInstance().GetPathToFloorTexture());
    viewer3D.SetDoorTexturePath(Settings::GetInstance().GetPathToDoorTexture());
    Logger::GetInstance().WriteToLog("\tViewer3D opened");
    viewer3D.exec();
    }
  catch (std::exception& err)
    {
    m_file_name = "";
    QString i_message = err.what();
    QMessageBox message;
    message.setWindowTitle("Error");
    message.setText(i_message);
    message.exec();
    }

  if(selected_cam_id > -1)
    project.Modify(selected_cam_id);
  Logger::GetInstance().WriteToLog("\tViewer3D closed");
  }

void MainWindow::OpenPreferences()
  {
  SettingsDialog settings_dialog;
  settings_dialog.exec();
  if (settings_dialog.result() == QDialog::Accepted)
    {
```

```
CanvasMouseController::GetInstance().SetActiveMouse(IMouse::MouseType::ESele
ction);
    _UncheckAllButtons();
    PropertiesController::GetInstance().Deactivate();

    m_ui.m_canvas_view->UpdateGrid();
    }
  }

void MainWindow::ShowHotkeysHelp()
  {
  QString hotkeys_text = "Hotkeys list:\n";
  std::vector<QString> hotkeys_vector;

  int longest_action_name = 0;
  for (auto& action : findChildren<QAction*>())
    {
    if (action->shortcut().toString() != "")
      longest_action_name = std::max(longest_action_name, action->text().length());
    }

  for (auto& action : findChildren<QAction*>())
    {
    if (action->shortcut().toString() != "")
      {
      QString hotkey_string = action->text();
      hotkey_string += QString(longest_action_name - action->text().length() + 10, ' ');
      hotkey_string += action->shortcut().toString();
      hotkey_string += QString('\n');
      hotkeys_vector.emplace_back(hotkey_string);
      }
    }
  std::sort(hotkeys_vector.begin(), hotkeys_vector.end());

  int index = 0;
  for (auto& hotkey : hotkeys_vector)
    {
    if ((index % 5) == 0)
      hotkeys_text += QString('\n');
    hotkeys_text += hotkey;
    index++;
    }
  QFont monospace_font("Consolas");
  monospace_font.setStyleHint(QFont::Monospace);
```

```
  QMessageBox message;
  message.setWindowIcon(QIcon(":/icons/Help-icon.png"));
  message.setWindowTitle("Hotkeys Help");
  message.setText(hotkeys_text);
  message.setFont(monospace_font);
  message.exec();
  }

void MainWindow::_ClearUndoRedo()
  {
  UndoRedoManagerController::GetInstance().Clear();
  m_ui.m_action_undo->setEnabled(false);
  m_ui.m_action_redo->setEnabled(false);
  }

void MainWindow::ShowLog()
  {
  m_ui.m_dock_log->show();
  }

void MainWindow::_EditLog()
  {

m_ui.m_log_text_browser->appendPlainText(QString(Logger::GetInstance().GetLast
Changes().c_str()));
  }

void MainWindow::_ReselectObjects()
  {
  auto selected_ids=MainWindowController::GetInstance().GetSelectedIDs();
  Project::GetInstance().ReselectObjectsByIDs(selected_ids);
  }


void MainWindow::SavePlanAsImage()
  {
  QString file_name = QFileDialog::getSaveFileName(this,
    "Save plan as image", "",
    "Image Files(*.png *.jpg *.jpeg *.bmp);;All Files (*)");
  if (file_name.isEmpty())
    return;
                                    QSize                scene_size                =
Project::GetInstance().GetScene()->itemsBoundingRect().toRect().size();
            qreal      jackal_coef      =      (scene_size.width()>scene_size.height())      ?
(2048.0f/scene_size.width()) : (2048.0f/scene_size.height());
```

```
                                                                    QPixmap
pixmap(Project::GetInstance().GetScene()->itemsBoundingRect().toRect().size()*jac
kal_coef);
  QPainter painter(&pixmap);
                        std::unique_ptr<QGraphicsScene>          save_scene          =
std::make_unique<QGraphicsScene>();


save_scene->setSceneRect(Project::GetInstance().GetScene()->itemsBoundingRect()
);
  for (auto item : Project::GetInstance().GetScene()->items())
    {
    save_scene->addItem(item);
    }

  QGraphicsView tmp_view(save_scene.get());
  tmp_view.setBackgroundBrush(Qt::white);
  tmp_view.scale(1.0f,-1.0f);
  tmp_view.fitInView(save_scene->itemsBoundingRect().toRect());
  painter.setRenderHint(QPainter::Antialiasing);
  tmp_view.render(&painter, pixmap.rect(), tmp_view.rect(), Qt::IgnoreAspectRatio);
  pixmap.save(file_name);

  for (auto item : save_scene->items())
    {
    Project::GetInstance().GetScene()->addItem(item);
    }
  }

int MainWindow::_GetExitDecision()
  {
  QMessageBox close_box;
  close_box.setWindowTitle("Editor2D");
  close_box.setText("Do you want to save plan?");
  close_box.addButton(QMessageBox::Yes);
  close_box.setDefaultButton(QMessageBox::Yes);
  close_box.addButton(QMessageBox::No);
  close_box.setDefaultButton(QMessageBox::No);
  close_box.addButton(QMessageBox::Cancel);
  close_box.setDefaultButton(QMessageBox::Cancel);
  return close_box.exec();
  }
```

MainWindowController.h:

```
#pragma once

class QStandardItemModel;
class QStandardItem;

class MainWindowController : public  QObject
  {
  Q_OBJECT

  public :
    static MainWindowController &GetInstance();
    std::shared_ptr<QStandardItemModel> GetTreeModel();
    std::shared_ptr<QItemSelectionModel> GetSelectionModel();
    void ChangeStatusBarText(QString i_message);
    void UpdateObjectTree();
    void UpdateObjectTreeWalls();
    void UpdateObjectTreeFurnitures();

    std::vector<long long> GetSelectedIDs();
  signals :
    void StatusBarUpdateSignal(QString i_message, int i_time_out = 0);
  private:
    MainWindowController();
    MainWindowController(const MainWindowController&) = delete;
    MainWindowController& operator=(const MainWindowController&) = delete;
  private:
    std::shared_ptr<QStandardItemModel> mp_object_tree_model;
    std::shared_ptr<QItemSelectionModel> mp_selection;
    QStandardItem* mp_root_furniture;
    QStandardItem* mp_root_walls;
    QStandardItem* mp_plan;
    std::map<std::string, QIcon> m_icons;
  };
```

MainWindowController.cpp:

```
#include "stdh.h"
#include "MainWindowController.h"
#include "Project.h"
#include "UIObjects/UIWall.h"

#include <CoreData/Wall.h>
#include <CoreData/Plan.h>
#include <CoreData/Furniture.h>
#include <CoreData/Door.h>
#include <CoreData/Window.h>

MainWindowController& MainWindowController::GetInstance()
  {
  static MainWindowController main_window_controller_instance;
  return main_window_controller_instance;
  }

std::shared_ptr<QStandardItemModel> MainWindowController::GetTreeModel()
  {
  return mp_object_tree_model;
  }

std::shared_ptr<QItemSelectionModel> MainWindowController::GetSelectionModel()
  {
  return mp_selection;
  }

void MainWindowController::ChangeStatusBarText(QString i_message)
```

```
  {
  emit StatusBarUpdateSignal(i_message);
  }


std::vector<long long> MainWindowController::GetSelectedIDs()
  {
  auto& selected_items = mp_selection->selectedRows();
  std::vector<long long> selected_ids;
  for (auto& item : selected_items)
    {
    selected_ids.push_back(mp_object_tree_model->itemFromIndex(item)->data().toLongLong());
    }
  return selected_ids;
  }


void MainWindowController::UpdateObjectTree()
  {
  UpdateObjectTreeFurnitures();
  UpdateObjectTreeWalls();
  }

void MainWindowController::UpdateObjectTreeWalls()
  {
  mp_selection->blockSignals(true);
  mp_root_walls->removeRows(0, mp_root_walls->rowCount());
  long long object_count = 0;
  std::vector<QStandardItem*> wall_items;
  auto walls = Project::GetInstance().GetPlan().GetWalls();
  auto selected_ids = Project::GetInstance().GetIdSelectedObjects();
  for (const auto& elem : walls)
    {
    wall_items.emplace_back(new QStandardItem(m_icons["wall"],
      QString("Wall ") + std::to_string(elem->GetId()).c_str()));
    wall_items.back()->setData(elem->GetId());
    wall_items.back()->setFlags(Qt::ItemIsSelectable| Qt::ItemIsEnabled);
    long long object_index_door = 0;
    for (auto& door : elem->GetDoors())
      {
      wall_items.back()->appendRow((new QStandardItem(m_icons["door"],
        QString("Door ") + std::to_string(door->GetId()).c_str())));
      wall_items.back()->child(object_index_door)->setData(door->GetId());
      wall_items.back()->child(object_index_door)->setFlags(Qt::ItemIsSelectable|
Qt::ItemIsEnabled);
      object_index_door++;
      }

    long long object_index_window = object_index_door;
    for (auto& window : elem->GetWindows())
      {
      wall_items.back()->appendRow((new QStandardItem(m_icons["window"],
        QString("Window ") + std::to_string(window->GetId()).c_str())));
      wall_items.back()->child(object_index_window)->setData(window->GetId());
      wall_items.back()->child(object_index_window)->setFlags(Qt::ItemIsSelectable|
Qt::ItemIsEnabled);
      object_index_window++;
      }

    }
  for (const auto& wall : wall_items)
    {
    mp_root_walls->appendRow(wall);
    if (std::find(selected_ids.begin(), selected_ids.end(), wall->data()) !=
selected_ids.end())
```

```
      {
        mp_selection->select(mp_object_tree_model->indexFromItem(wall),
QItemSelectionModel::Select
                                                                    |
QItemSelectionModel::Rows);
      }


    if (wall->hasChildren())
      {
      for (int i = 0; i < wall->rowCount(); ++i)
        {
        if (std::find(selected_ids.begin(), selected_ids.end(), wall->child(i)->data()) !=
selected_ids.end())
          {
          mp_selection->select(mp_object_tree_model->indexFromItem(wall->child(i)),
QItemSelectionModel::Select
                                                                        |
QItemSelectionModel::Rows);
          }
        }
      }
    }
  mp_selection->blockSignals(false);
  }

void MainWindowController::UpdateObjectTreeFurnitures()
  {
  mp_selection->blockSignals(true);
  mp_root_furniture->removeRows(0, mp_root_furniture->rowCount());
  auto selected_ids = Project::GetInstance().GetIdSelectedObjects();
  std::vector<QStandardItem*> furniture_items;
  auto furnitures = Project::GetInstance().GetPlan().GetFurniture();
  for (const auto& elem : furnitures)
    {
    std::string furniture_type;
    switch (elem->GetType())
      {
      case CoreData::FurnitureType::EFurnitureSofa:
        furniture_type = "Sofa ";
        break;
      case CoreData::FurnitureType::EFurnitureSafe:
        furniture_type = "Safe ";
        break;
      case CoreData::FurnitureType::EFurnitureCabinet:
        furniture_type = "Cabinet ";
        break;
      case CoreData::FurnitureType::EFurnitureChair:
        furniture_type = "Chair ";
        break;
      case CoreData::FurnitureType::EFurnitureTable:
        furniture_type = "Table ";
        break;
      default:
        furniture_type = "Unknown furniture ";
        break;
      }
    furniture_items.emplace_back(new QStandardItem(m_icons["furniture"],
      QString(furniture_type.c_str()) + std::to_string(elem->GetId()).c_str()));
    furniture_items.back()->setData(elem->GetId());
    furniture_items.back()->setFlags(Qt::ItemIsSelectable| Qt::ItemIsEnabled);
    }

  for (const auto& furniture : furniture_items)
    {
```

```
    mp_root_furniture->appendRow(furniture);
    if (std::find(selected_ids.begin(), selected_ids.end(), furniture->data()) !=
selected_ids.end())
      {
      mp_selection->select(mp_object_tree_model->indexFromItem(furniture),
QItemSelectionModel::Select
                                                                          |
QItemSelectionModel::Rows);
      }
    }
  mp_selection->blockSignals(false);
  }


MainWindowController::MainWindowController()
  {
  QIcon icon_line;
  icon_line.addFile(QStringLiteral(":/icons/vector-path-line-128x128.png"), QSize(),
QIcon::Normal, QIcon::Off);
  QIcon icon_furniture;
  icon_furniture.addFile(QStringLiteral(":/icons/furniture.png"), QSize(), QIcon::Normal,
QIcon::Off);
  QIcon icon_door;
  icon_door.addFile(QStringLiteral(":/icons/Iconsmind-Outline-Door.ico"), QSize(),
QIcon::Normal, QIcon::Off);
  QIcon icon_window;
  icon_window.addFile(QStringLiteral(":/icons/Window.png"), QSize(), QIcon::Normal,
QIcon::Off);

  m_icons.insert(std::pair<std::string, QIcon>("wall", icon_line));
  m_icons.insert(std::pair<std::string, QIcon>("furniture", icon_furniture));
  m_icons.insert(std::pair<std::string, QIcon>("door", icon_door));
  m_icons.insert(std::pair<std::string, QIcon>("window", icon_window));

  mp_object_tree_model = std::make_shared<QStandardItemModel>();
  mp_selection = std::make_shared<QItemSelectionModel>(mp_object_tree_model.get());

  mp_root_walls = new QStandardItem(m_icons["wall"], "Walls");
  mp_root_walls->setSelectable(false);
  mp_root_furniture = new QStandardItem(m_icons["furniture"], "Furniture");
  mp_root_furniture->setSelectable(false);
  mp_plan = new QStandardItem(m_icons["rectangle"], "Plan");
  mp_plan->setSelectable(false);

  mp_object_tree_model->appendRow(mp_plan);

  mp_plan->appendRow(mp_root_walls);
  mp_plan->appendRow(mp_root_furniture);
  }
```

Project.h:

```
#pragma once
#include "UIObjects/IUIOBject.h"
#include "ScaleManager.h"

#include <CoreData/IDatabase.h>
#include <CoreData/FurnitureEnum.h>
#include <CoreData/ParameterObjects/RoomParameters.h>

class QGraphicsScene;
class QPointF;
```

```
namespace Serializer
  {
  class XmlSerializerWriter;
  }

namespace CoreData
  {
  class Plan;
  class Wall;
  class Furniture;
  class Door;
  class IParameterObject;
  class DoorParameters;
  class WindowParameters;
  class IMoveRotate;
  enum class ZoneType;
  }

#pragma warning(push)
#pragma warning(disable:4251)

  class Project : CoreData::IDatabase
    {
    public:
      struct NodeSelectionInfo;

      Project(const Project&) = delete;
      Project(Project&&) = delete;
      Project& operator= (const Project&) = delete;
      Project& operator= (Project&&) = delete;

      static Project& GetInstance();

      void OpenPlan(std::string i_filename);
      void SavePlan(std::string i_filename);
      void ClearPlan();

      bool IsPlanEmpty();
      void SetScene(QGraphicsScene *ip_scene);
      QGraphicsScene *GetScene();

      void GetWallParamById(
        long long i_id,
        long long& o_begin_id,
        long long& o_end_id,
        CoreMath::Vector2D &o_begin,
        CoreMath::Vector2D &o_end,
        long long &o_height) const;

      const std::shared_ptr<CoreData::Wall>     GetWallById(long long i_id) const;
      const std::shared_ptr<CoreData::WallNode> GetWallNodeById(long long i_id) const;
      const std::shared_ptr<CoreData::Furniture>GetFurnitureById(long long i_id) const;

      const std::shared_ptr<CoreData::IMoveRotate> GetObjectById(long long i_id) const;
      IUIObject* GetUIObjectById(long long i_id) const;

      void GetFurnitureParamById(
        long long i_id,
        CoreMath::Vector2D& o_pos,
        double &o_direction_angle,
        long long &o_width,
        long long &o_length,
        long long &o_height,
        CoreData::FurnitureType &o_type,
        bool &o_valid) const;
```

```
void GetDoorParamById(
  long long i_id,
  long long &o_shift,
  long long &o_width,
  long long &o_height,
  bool &o_is_direction_up,
  long long &o_begin_node_id,
  long long &o_end_node_id) const;

void GetWindowParamById(
  long long i_id,
  long long& o_width,
  long long& o_height,
  long long& o_left_shift,
  long long& o_bottom_shift,
  long long& o_begin_node_id,
  long long& o_end_node_id);

long long AddNodeToPlan(
  CoreMath::Vector2D i_position,
  long long i_id = -1);

long long AddWallToPlan(
  long long i_begin_id,
  long long i_end_id,
  long long i_height,
  long long i_id = -1);


long long AddFurnitureToPlan(
  const CoreMath::Vector2D& i_pos,
  double i_direction_angle,
  long long i_width,
  long long i_length,
  long long i_height,
  CoreData::FurnitureType i_type,
  bool i_valid,
  long long i_id = -1);

void UpdateFurniture();
void CheckFurnitureValid();

CoreData::RoomParameters GetRoomParamsById(long long i_id);
std::vector<CoreData::RoomParameters> GetAllRoomsParams() const;

long long AddDoorToPlan(
  long long i_shift,
  long long i_width,
  long long i_height,
  bool i_is_direction_up,
  long long i_begin_node_id,
  long long i_end_node_id,
  long long i_id = -1);


long long AddDoorToPlan(
  const CoreData::DoorParameters &i_param,
  std::pair<long long, long long> i_wall_ids);

long long AddWindowToPlan(
  const CoreData::WindowParameters &i_param,
  std::pair<long long, long long> i_wall_ids);
```

```
        long long AddWindowToPlan(
          long long i_left_shift,
          long long i_bottom_shift,
          long long i_width,
          long long i_height,
          long long i_begin_node_id,
          long long i_end_node_id,
          long long i_id = -1);

        void EraseObjects();
        long long GetWallUnderCursor(const CoreMath::Vector2D& i_coordinates) const;
        bool TrySelect(const CoreMath::Vector2D& i_coordinates, bool i_cumulative);
        long long TrySelect(const CoreMath::Vector2D& i_left_top, const CoreMath::Vector2D&
i_right_bottom, bool i_cumulative);
        bool TrySelectNodes(const CoreMath::Vector2D& i_coordinates, bool i_cumulative);
        bool TrySelectWallNodes();
        void UnselectAll();

        long long TrySelectObject(const CoreMath::Vector2D& i_coordinates);
        bool TryMoveObject(const CoreMath::Vector2D &i_shift_position, double
i_shift_direction_angle, long long i_id);

        bool TryMoveNodesRelative(const CoreMath::Vector2D& i_shift);
        void SetNodeSelection(const std::vector<NodeSelectionInfo>& i_selection_info);
        std::vector<NodeSelectionInfo> GetNodeSelectionInfo() const;
        void EraseObjectById(long long i_id);
        void EraseRooms();

        CoreData::RoomParameters TrySelectRoom(
          const CoreMath::Vector2D &i_user_click_position,
          std::vector<CoreData::RoomParameters> &o_deleted_rooms);

        const CoreData::Plan& GetPlan() const;
        std::vector<long long> GetIdSelectedObjects() const;
        void ReselectObjectsByIDs(std::vector<long long>);

        void Register(CoreData::Wall* ip_camera) override final;
        void Register(CoreData::Furniture *ip_furniture) override final;
        void Register(CoreData::Room* ip_room) override final;
        void Register(
          CoreData::Door* ip_door,
          std::weak_ptr<CoreData::WallNode> i_begin_node,
          std::weak_ptr<CoreData::WallNode> i_end_node) override final;

        void Register(
          CoreData::Window* ip_window,
          std::weak_ptr<CoreData::WallNode> i_begin_node,
          std::weak_ptr<CoreData::WallNode> i_end_node) override final;


        bool HasSurveillanceArea() const ;

        bool Unregister(long long i_id) override final;

        bool Modify(long long i_id) override final;

        bool ModifyWallByNodesIds(long long i_node1, long long i_node2);

        template<typename IType> std::vector<IType*> GetItems();

        struct NodeSelectionInfo
          {
          long long m_wall_node_id;
          };
```

```
    private:
      Project();
      ~Project() = default;
      void _UpdateSelectedObjectsProperties();

    private:
      std::vector<std::weak_ptr<CoreData::WallNode>> m_selected_nodes;
      std::map<long long, std::unique_ptr<IUIObject>> m_items_map;
      std::unique_ptr<CoreData::Plan> mp_plan;
      QGraphicsScene *mp_scene;
      const double eps = 0.0001;
    };

  template<typename IType> std::vector<IType*> Project::GetItems()
    {
    std::vector<IType*> items;

    for (auto &elem : m_items_map)
      if (IType *item = dynamic_cast<IType*>(elem.second.get()))
        items.push_back(item);

    return items;
    }


#pragma warning(pop)
```

Project.cpp:

```
#include "stdh.h"
#include "Project.h"
#include "UIPropertiesBase.h"
#include "UIObjects/UIWall.h"
#include "MainWindowController.h"
#include "PropertiesController.h"
#include "UIObjects/UIFurniture.h"
#include "UIObjects/UIDoor.h"
#include "UIObjects/UIWindow.h"
#include "UIObjects/UIRoom.h"
#include "Logger.h"
#include "Settings.h"

#include "UndoRedoManagerController.h"
#include "Commands/RemoveWallCommand.h"
#include "Commands/RemoveFurnitureCommand.h"
#include "Commands/RemoveDoorCommand.h"
#include "Commands/RemoveWindowCommand.h"
#include "Commands\MoveRotateObjectCommand.h"

#include <CoreLogic/RoomInternalLogic.h>
#include <CoreLogic/Utility.h>
#include <CoreData/IMoveRotate.h>
```

```
#include <CoreData/Door.h>
#include <CoreData/Wall.h>
#include <CoreData/WallNode.h>
#include <CoreData/Window.h>
#include <CoreData/Room.h>
#include <CoreData/Plan.h>
#include <CoreData\ParameterObjects\DoorParameters.h>
#include <CoreData\ParameterObjects\WindowParameters.h>
#include <CoreMath/CompGeom.h>
#include <ImportExportLib/XmlSerializer.h>
#include <ImportExportLib/XmlDeserializer.h>



Project::Project() : mp_plan(std::make_unique<CoreData::Plan>())
   {
   mp_plan->SetOwner(this);
   }

  Project& Project::GetInstance()
   {
   static Project project;
   return project;
   }

  void Project::SetScene(QGraphicsScene *ip_scene)
   {
   mp_scene = ip_scene;
   }

  QGraphicsScene *Project::GetScene()
   {
   return mp_scene;
   }

  void Project::OpenPlan(std::string i_filename)
   {
   Serializer::XmlDeserializer xmlreader(i_filename);
   ClearPlan();
   mp_plan->Deserialize(xmlreader);

   //Setting id of last element in opened file
   if (!m_items_map.empty())
     mp_plan->SetIdToFactory(m_items_map.rbegin()->first + 1);
   MainWindowController::GetInstance().UpdateObjectTree();
```

```
  Logger::GetInstance().WriteToLog("\tPlan loaded from  " + i_filename );
  }

 void Project::SavePlan(std::string i_filename)
  {
  Serializer::XmlSerializer xmlwritter(i_filename);
  mp_plan->Serialize(xmlwritter);
  Logger::GetInstance().WriteToLog("\tPlan saved to  " + i_filename );
  }

 void Project::ClearPlan()
  {
  mp_plan->Clear();
  }


 long long Project::AddNodeToPlan(CoreMath::Vector2D i_position, long long i_id)
  {
  auto rooms = mp_plan->GetRooms();
  for (int index = 0; index < rooms.size(); index++)
                                        if      (CoreMath::PointInPolygon(i_position,
CoreLogic::ConvertRoomNodesToPolygon(rooms[index]->GetRoomNodes())))
     EraseObjectById(rooms[index]->GetId());
  return mp_plan->AddNode(i_position, i_id);
  }

  long long Project::AddWallToPlan(long long i_begin_id, long long i_end_id, long
long i_height, long long i_id)
  {
  long long new_wall_id = mp_plan->AddWall(i_begin_id, i_end_id, i_height, i_id);
  return new_wall_id;
  }


 long long Project::AddDoorToPlan(
  long long i_shift,
  long long i_width,
  long long i_height,
  bool i_is_direction_up,
  long long i_begin_node_id,
  long long i_end_node_id,
  long long i_id)
  {
        return  mp_plan->AddDoor(i_shift,  i_width,  i_height,  i_is_direction_up,
i_begin_node_id, i_end_node_id, i_id);
```

```
    }


  long long Project::AddDoorToPlan(
    const CoreData::DoorParameters &i_param,
    std::pair<long long, long long> i_wall_ids)
    {
    return mp_plan->AddDoor(
                                i_param.m_shift,      i_param.m_width,      i_param.m_height,
i_param.m_is_direction_up,
      i_wall_ids.first, i_wall_ids.second, i_param.m_preferable_id);
    }

  long long Project::AddWindowToPlan(
    const CoreData::WindowParameters &i_param,
    std::pair<long long, long long> i_wall_ids)
    {
        return  mp_plan->AddWindow(i_param.m_left_shift,  i_param.m_bottom_shift,
i_param.m_width, i_param.m_height,
      i_wall_ids.first, i_wall_ids.second, i_param.m_preferable_id);
    }


    long long Project::AddWindowToPlan(
    long long i_left_shift,
    long long i_bottom_shift,
    long long i_width,
    long long i_height,
    long long i_begin_node_id,
    long long i_end_node_id,
    long long i_id)
    {
        return  mp_plan->AddWindow(i_left_shift,  i_bottom_shift,  i_width,  i_height,
i_begin_node_id, i_end_node_id, i_id);
    }


std::vector<long long> Project::GetIdSelectedObjects() const
    {
    std::vector<long long> id_for_delete;

    for (const auto &pair : m_items_map)
      {
      if (pair.second->IsSelected())
```

```
        id_for_delete.push_back(pair.first);
      }
    return id_for_delete;
    }

  long long Project::AddFurnitureToPlan(
    const CoreMath::Vector2D& i_pos,
    double i_direction_angle,
    long long i_width,
    long long i_length,
    long long i_height,
    CoreData::FurnitureType i_type,
    bool i_valid,
    long long i_id)
    {
        return mp_plan->AddFurniture(i_pos, i_direction_angle, i_width, i_length,
i_height, i_type, i_valid, i_id);
    }

  CoreData::RoomParameters Project::GetRoomParamsById(long long i_id)
    {
    for (const auto &room : mp_plan->GetRooms())
      if (room->GetId() == i_id)
        return room->GetParameters();
    return CoreData::RoomParameters();
    }

  std::vector<CoreData::RoomParameters> Project::GetAllRoomsParams() const
    {
    std::vector<CoreData::RoomParameters> rooms;

    for (const auto &room : mp_plan->GetRooms())
      rooms.push_back(room->GetParameters());
    return rooms;
    }


  void Project::EraseObjects()
    {

    std::vector<long long> id_for_delete = GetIdSelectedObjects();


    std::vector<QUndoCommand *> commands;
```

```
    for (auto wall_obj_pointer : Project::GetInstance().GetPlan().GetWalls())
     {
     //erase doors
     for (size_t i = 0; i < wall_obj_pointer->GetDoors().size(); i++)
      {
                                if    (std::find(id_for_delete.begin(),    id_for_delete.end(),
wall_obj_pointer->GetDoors()[i]->GetId()) != id_for_delete.end())
                                                                commands.push_back(new
RemoveDoorCommand(wall_obj_pointer->GetDoors()[i]->GetId()));
       if (wall_obj_pointer->GetDoors().empty())
        break;
       }

     //erase windows
     for (size_t i = 0; i < wall_obj_pointer->GetWindows().size(); i++)
      {
                                if    (std::find(id_for_delete.begin(),    id_for_delete.end(),
wall_obj_pointer->GetWindows()[i]->GetId()) != id_for_delete.end())
                                                                commands.push_back(new
RemoveWindowCommand(wall_obj_pointer->GetWindows()[i]->GetId()));
       if (wall_obj_pointer->GetWindows().empty())
        break;
       }

                              if     (std::find(id_for_delete.begin(),     id_for_delete.end(),
wall_obj_pointer->GetId()) != id_for_delete.end())
       {
       for (const auto &door : wall_obj_pointer->GetDoors())
        commands.push_back(new RemoveDoorCommand(door->GetId()));
       for (const auto &window : wall_obj_pointer->GetWindows())
        commands.push_back(new RemoveWindowCommand(window->GetId()));
       commands.push_back(new RemoveWallCommand(wall_obj_pointer->GetId()));
       }
     }

    for (auto furniture_obj_pointer : Project::GetInstance().GetPlan().GetFurniture())
     {
                              if     (std::find(id_for_delete.begin(),     id_for_delete.end(),
furniture_obj_pointer->GetId()) != id_for_delete.end())
                                                                commands.push_back(new
RemoveFurnitureCommand(furniture_obj_pointer->GetId()));
     }

    UndoRedoManagerController::GetInstance().PushGroup(commands);
```

```
    MainWindowController::GetInstance().UpdateObjectTree();
    }

        long long Project::GetWallUnderCursor(const CoreMath::Vector2D &
i_coordinates) const
    {
                                                double selection_tolerance =
ScaleManager::GetInstance().GetSelectionTolerance();
     const long long select_tolerance_sqr = selection_tolerance * selection_tolerance *
Settings::GetInstance().GetGridStep() / 1000;
            std::pair<long long, long long> nearest_wall = std::make_pair(-1LL,
select_tolerance_sqr);
    for (const auto& wall : mp_plan->GetWalls())
      {
              const long long wall_dist = CoreMath::DistanceSqr(wall->GetBegin(),
wall->GetEnd(), i_coordinates);
      if (wall_dist < nearest_wall.second)
        {
       nearest_wall.first = wall->GetId();
       nearest_wall.second = wall_dist;
        }
      }

    return nearest_wall.first;
    }

  void Project::EraseObjectById(long long i_id)
    {
    mp_plan->DeleteItemById(i_id);
    }

  void Project::EraseRooms()
    {
      auto rooms = mp_plan->GetRooms();
      for (int index = 0; index < rooms.size(); ++index)
        {
       EraseObjectById(rooms[index]->GetId());
        }
    }

   CoreData::RoomParameters Project::TrySelectRoom(const CoreMath::Vector2D &
i_user_click_position, std::vector<CoreData::RoomParameters> &o_deleted_rooms)
    {
```

```
                std::vector<std::shared_ptr<CoreData::WallNode>>  room_nodes  =
CoreLogic::TryGetRoomNodes(
    i_user_click_position,
    mp_plan->GetWalls());

   std::vector<CoreMath::Vector2D> room_points;

   for (auto& node : room_nodes)
    room_points.push_back(node->GetPosition());

   if (room_nodes.size() != 0)
    {
          auto const internal_rooms = CoreLogic::GetRoomInternals(room_nodes,
mp_plan->GetWalls());
    std::vector <std::vector<CoreMath::Vector2D>> internal_rooms_points;

    for (auto &internal_room : internal_rooms)
     {
     std::vector<CoreMath::Vector2D> internal_room_points;
     for (auto &node : internal_room)
      internal_room_points.push_back(node->GetPosition());

     internal_rooms_points.emplace_back(internal_room_points);
     }

                    std::vector<CoreMath::DirectedSegment>  segments_union  =
CoreLogic::ConvertRoomNodesToPolygon(room_points);
    for (auto internal_room_points : internal_rooms_points)
     {
                  segments_union  =  CoreMath::SubtractPolygon(segments_union,
CoreLogic::ConvertRoomNodesToPolygon(internal_room_points));
     }

    std::vector<long long> deleting_rooms_id;

    for (auto & zone : mp_plan->GetRooms())
     {
     if (zone->GetType() == CoreData::ZoneType::ECommonZone)
      {
                    std::vector<CoreMath::DirectedSegment>  checking_zone  =
CoreLogic::ConvertRoomNodesToPolygon(zone->GetRoomNodes());
       for (auto internal_room_points : zone->GetInternalRoomNodes())
        {
                    checking_zone  =  CoreMath::SubtractPolygon(checking_zone,
CoreLogic::ConvertRoomNodesToPolygon(internal_room_points));
```

```
        }
        if (CoreMath::IntersectPolygons({ segments_union,checking_zone }).size() !=
0)
        {
                                segments_union = CoreMath::UnionPolygons({
segments_union,checking_zone });
        deleting_rooms_id.emplace_back(zone->GetId());
        }

       }
      }

    if (segments_union.empty())
     return CoreData::RoomParameters();

    for (auto deleting_room_id : deleting_rooms_id)
     {
     o_deleted_rooms.push_back(GetRoomParamsById(deleting_room_id));
     mp_plan->DeleteItemById(deleting_room_id);
     }
          CoreLogic::ConvertPolygonToRoom(room_points, internal_rooms_points,
segments_union);

                    long long room_id = mp_plan->AddRoom(room_points,
internal_rooms_points,CoreData::ZoneType::ECommonZone);
            return CoreData::RoomParameters(room_points, internal_rooms_points,
CoreData::ZoneType::ECommonZone, room_id);
    }
  return CoreData::RoomParameters();
  }

     bool Project::TrySelect(const CoreMath::Vector2D& i_coordinates, bool
i_cumulative)
  {
  if (!i_cumulative)
   UnselectAll();

  PropertiesController::GetInstance().DeletePropertiesWindow();
  MainWindowController::GetInstance().UpdateObjectTree();
  for (auto r_iter = m_items_map.rbegin(); r_iter != m_items_map.rend(); r_iter++)
   if (r_iter->second->TrySelect(i_coordinates))
    {
    MainWindowController::GetInstance().UpdateObjectTree();
    _UpdateSelectedObjectsProperties();
    return true;
```

```
        }
    return false;
    }

  long long Project::TrySelectObject(const CoreMath::Vector2D& i_coordinates)
    {
                                                double          selection_tolerance          =
ScaleManager::GetInstance().GetSelectionTolerance();
      for (auto& furniture : mp_plan->GetFurniture())
        if(CoreMath::IsPointInsideOfRectObject(
             i_coordinates,
             furniture->GetWidth(),
             furniture->GetLength(),
             furniture->GetPosition(),
             furniture->GetDirectionAngle())
          )
         {
          return furniture->GetId();
         }
      return -1;
    }

   const std::shared_ptr<CoreData::IMoveRotate> Project::GetObjectById(long long
i_id) const
    {
    for (const auto &furniture : mp_plan->GetFurniture())
      if(furniture->GetId() == i_id)
        {
        return furniture;
        }
    return nullptr;
    }

  IUIObject* Project::GetUIObjectById(long long i_id) const
    {
    if (m_items_map.find(i_id) != m_items_map.end())
      {
      return m_items_map.at(i_id).get();
      }
    return nullptr;
    }

      long long Project::TrySelect(const CoreMath::Vector2D& i_left_top, const
CoreMath::Vector2D& i_right_bottom, bool i_cumulative)
    {
```

```
    if (!i_cumulative)
      UnselectAll();

    long long selected_items = 0;
          CoreMath::Vector2D  left_top(std::min(i_left_top.m_x,  i_right_bottom.m_x),
std::max(i_left_top.m_y, i_right_bottom.m_y));
     CoreMath::Vector2D right_bottom(std::max(i_left_top.m_x, i_right_bottom.m_x),
std::min(i_left_top.m_y, i_right_bottom.m_y));

    for (auto r_iter = m_items_map.rbegin(); r_iter != m_items_map.rend(); r_iter++)
      if (r_iter->second->TrySelect(left_top, right_bottom))
        ++selected_items;
    PropertiesController::GetInstance().DeletePropertiesWindow();
    MainWindowController::GetInstance().UpdateObjectTree();
    _UpdateSelectedObjectsProperties();
    return selected_items;
    }

     bool  Project::TrySelectNodes(const  CoreMath::Vector2D&  i_coordinates,  bool
i_cumulative)
    {
                                              double          selection_tolerance          =
ScaleManager::GetInstance().GetSelectionTolerance();

    if(!i_cumulative)
      m_selected_nodes.clear();

    bool selected = false;

    for (auto& wall_node : mp_plan->GetWallNodes())
      {
             if  (CoreMath::DistanceSqr(wall_node->GetPosition(),  i_coordinates)  <=
selection_tolerance * selection_tolerance)
        {
        bool found = false;
        for (auto& selected_node : m_selected_nodes)
          {
          auto p_node = selected_node.lock();
          if (!p_node)
            continue;
          if (p_node == wall_node)
            {
            found = true;
            break;
            }
```

```
        }
      if (!found)
        {
        m_selected_nodes.push_back(wall_node);
        selected = true;
        }
      }
    }

  return selected;
  }

bool Project::TrySelectWallNodes()
  {
  m_selected_nodes.clear();

  auto& walls = GetPlan().GetWalls();

  std::vector<long long> selected_ids;
  for (auto& pair : m_items_map)
    {
    if (pair.second->IsSelected())
      selected_ids.push_back(pair.first);
    }

  for (long long id : selected_ids)
    {
    for (auto& wall : walls)
      {
      if (wall->GetId() == id)
        {
        for(auto& wall_node : {wall->GetBeginNode(), wall->GetEndNode()})
          {
          bool found = false;
          for (auto& selected_node : m_selected_nodes)
            {
            auto p_node = selected_node.lock();
            if (!p_node)
              continue;
            if (p_node == wall_node)
              {
              found = true;
              break;
              }
            }
```

```
            if (!found)
              {
              m_selected_nodes.push_back(wall_node);
              }
             }
            break;
            }
          }
        }

    return !m_selected_nodes.empty();
    }

  void Project::UnselectAll()
    {
    for (auto& pair : m_items_map)
      pair.second->Unselect();
    PropertiesController::GetInstance().DeletePropertiesWindow();
    MainWindowController::GetInstance().UpdateObjectTree();
    }

  void Project::_UpdateSelectedObjectsProperties()
    {
    if (GetIdSelectedObjects().size() == 1)
      {
      auto p_selected = m_items_map[GetIdSelectedObjects().front()].get();
      if (dynamic_cast<UIPropertiesBase*>(p_selected))
        {
        PropertiesController::GetInstance().AddPropertiesWindow(
          dynamic_cast<UIPropertiesBase*>(p_selected));
        PropertiesController::GetInstance().Activate();
        }
      }
    }

    bool Project::TryMoveObject(const CoreMath::Vector2D &i_shift_position, double
i_new_direction_angle, long long i_id)
    {
    bool changed = false;

    auto p_object = GetObjectById(i_id);
    if (!p_object)
      return false;

    if (i_shift_position != CoreMath::Vector2D(0, 0))
```

```
    {
    changed = true;
    p_object->SetPosition(i_shift_position + p_object->GetPosition());
    }

  if (fabs(i_new_direction_angle - p_object->GetDirectionAngle()) > eps)
    {
    changed = true;
    p_object->SetDirectionAngle(i_new_direction_angle);
    }
  if (changed)
    Modify(i_id);

  return changed;
  }

bool Project::TryMoveNodesRelative(const CoreMath::Vector2D& i_shift)
  {
  if (i_shift == CoreMath::Vector2D(0, 0))
    return false;

  EraseRooms();

  bool roll_back = false;

  auto move_verts_fn = [this](const CoreMath::Vector2D& i_shift)
    {
    for (auto& vertex : m_selected_nodes)
      {
      auto p_wall_node = vertex.lock();
      if (!p_wall_node)
        continue;

      p_wall_node->SetPosition(p_wall_node->GetPosition() + i_shift);

      for (auto& wall : p_wall_node->GetLinkedWalls())
        {
        auto p_wall = wall.lock();
        if (p_wall)
          {
          Modify(p_wall->GetId());
          for (const auto& window : p_wall->GetWindows())
            Modify(window->GetId());
          for (const auto& door : p_wall->GetDoors())
            Modify(door->GetId());
```

```
            }
          }
        }
      };

    move_verts_fn(i_shift);

    std::unordered_set<CoreData::Wall*> selected_walls;
    for (auto& node : m_selected_nodes)
     {
     auto p_wall_node = node.lock();
     if (!p_wall_node)
       continue;
     auto linked_walls = p_wall_node->GetLinkedWalls();
     for (auto& wall : linked_walls)
       {
       auto p_wall = wall.lock();
       if (p_wall)
         selected_walls.insert(p_wall.get());
       }
     }


    for (auto& selected_wall : selected_walls)
     {
     for (const auto& wall : GetPlan().GetWalls())
       {
       if (wall.get() != selected_wall &&
           CoreMath::EdgesIntersectionType(
             CoreLogic::ConvertWallToSegment(*selected_wall),
                                             CoreLogic::ConvertWallToSegment(*wall))    !=
CoreMath::EEdgeIntersectionType::NO_INTERSECTION)
         {

         if(wall->GetBeginNode() == selected_wall->GetBeginNode() ||
           wall->GetEndNode()   == selected_wall->GetEndNode()   ||
           wall->GetBeginNode() == selected_wall->GetEndNode()   ||
           wall->GetEndNode()   == selected_wall->GetBeginNode())
          continue;

         roll_back = true;
         break;
         }
       }
      if (roll_back)
```

```
        break;
      }


    if (!roll_back)
      {
      return true;
      }

    move_verts_fn(-i_shift);

    return false;
    }

  void Project::UpdateFurniture()
    {
    for (const auto& furniture : GetPlan().GetFurniture())
      {
      Modify(furniture->GetId());
      }
    }

  void Project::CheckFurnitureValid()
    {
      for (const auto &furniture : Project::GetInstance().GetPlan().GetFurniture())
        {
        bool change = false;
        for (const auto &wall : Project::GetInstance().GetPlan().GetWalls())
          {
          bool intersect = CoreMath::IntersectionLineWithRectangle(
            furniture.get()->GetPosition(),
            furniture.get()->GetWidth(),
            furniture.get()->GetLength(),
            furniture.get()->GetDirectionAngle(),
            wall.get()->GetBegin(),
            wall.get()->GetEnd());

          change |= intersect;
          }

                                        for (const auto &furniture_second :
Project::GetInstance().GetPlan().GetFurniture())
          {
          if (furniture.get()->GetId() != furniture_second.get()->GetId())
            {
```

```
            bool intersect = CoreMath::IntersectionRectangleWithRectangle(
              furniture.get()->GetPosition(),
              furniture.get()->GetWidth(),
              furniture.get()->GetLength(),
              furniture.get()->GetDirectionAngle(),
              furniture_second.get()->GetPosition(),
              furniture_second.get()->GetWidth(),
              furniture_second.get()->GetLength(),
              furniture_second.get()->GetDirectionAngle()
            );
            change |= intersect;
            }
          }

          furniture.get()->SetValid(!change);
        }
      Project::GetInstance().UpdateFurniture();
    }

void Project::SetNodeSelection(const std::vector<Project::NodeSelectionInfo>&
i_selection_info)
    {
    m_selected_nodes.clear();

    for (auto& node_info : i_selection_info)
      {
      for (auto& wall_node : GetPlan().GetWallNodes())
        {
        if (wall_node->GetId() == node_info.m_wall_node_id)
          {
          bool found = false;
          for (auto& selected_node : m_selected_nodes)
            {
            auto p_node = selected_node.lock();
            if (!p_node)
              continue;
            if (p_node == wall_node)
              {
              found = true;
              break;
              }
            }
          if (!found)
            m_selected_nodes.push_back(wall_node);
          break;
```

```
        }
      }
    }
  }

std::vector<Project::NodeSelectionInfo> Project::GetNodeSelectionInfo() const
  {
  std::vector<NodeSelectionInfo> selection_info;
  for (auto& node : m_selected_nodes)
    {
    auto p_wall_node = node.lock();
    if (!p_wall_node)
      continue;

    NodeSelectionInfo node_info;
    node_info.m_wall_node_id = p_wall_node->GetId();
    selection_info.push_back(node_info);
    }
  return selection_info;
  }

const CoreData::Plan& Project::GetPlan() const
  {
  return *mp_plan;
  }

void Project::Register(CoreData::Wall *ip_wall)
  {
  m_items_map.insert(std::pair<long long, std::unique_ptr<IUIObject>>(
    ip_wall->GetId(),
    std::make_unique<UIWall>(*ip_wall)));
  MainWindowController::GetInstance().UpdateObjectTreeWalls();
  }

void Project::Register(CoreData::Furniture *ip_furniture)
  {
  m_items_map.insert(std::pair<long long, std::unique_ptr<IUIObject>>(
    ip_furniture->GetId(),
    std::make_unique<UIFurniture>(*ip_furniture)));
  MainWindowController::GetInstance().UpdateObjectTreeFurnitures();
  }

void Project::Register(CoreData::Room * ip_room)
  {
  m_items_map.insert(std::pair<long long, std::unique_ptr<IUIObject>>(
```

```
    ip_room->GetId(),
    std::make_unique<UIRoom>(*ip_room)));
  }

void Project::Register(
  CoreData::Door* ip_door,
  std::weak_ptr<CoreData::WallNode> i_begin_node,
  std::weak_ptr<CoreData::WallNode> i_end_node)
  {
  m_items_map.insert(std::pair<long long, std::unique_ptr<IUIObject>>(
    ip_door->GetId(),
    std::make_unique<UIDoor>(*ip_door, i_begin_node, i_end_node)));
  MainWindowController::GetInstance().UpdateObjectTreeWalls();
  }

void Project::Register(
  CoreData::Window* ip_window,
  std::weak_ptr<CoreData::WallNode> i_begin_node,
  std::weak_ptr<CoreData::WallNode> i_end_node)
  {
  m_items_map.insert(std::pair<long long, std::unique_ptr<IUIObject>>(
    ip_window->GetId(),
    std::make_unique<UIWindow>(*ip_window, i_begin_node, i_end_node)));
  MainWindowController::GetInstance().UpdateObjectTreeWalls();
  }

bool Project::HasSurveillanceArea() const
  {
  return mp_plan->GetRooms().size()>0;
  }

bool Project::Unregister(long long i_id)
  {
  return m_items_map.erase(i_id) != 0;
  }

bool Project::Modify(long long i_id)
  {
  if (m_items_map.find(i_id) == m_items_map.end())
    return false;
  m_items_map.find(i_id)->second->UpdateGraphicsItem();
  return true;
  }

bool Project::ModifyWallByNodesIds(long long i_node1, long long i_node2)
```

```
  {
  for (const auto& wall : mp_plan->GetWalls())
    {
     if (wall->GetBeginNode()->GetId() == i_node1 || wall->GetBeginNode()->GetId()
== i_node2)
        if (wall->GetEndNode()->GetId() == i_node1 || wall->GetEndNode()->GetId()
== i_node2)
      {
       Modify(wall->GetId());
       return true;
      }
    }
  return false;
  }

void Project::GetWallParamById(
    long long i_id,
    long long& o_begin_id,
    long long& o_end_id,
    CoreMath::Vector2D &o_begin,
    CoreMath::Vector2D &o_end,
    long long &o_height) const
    {
    for (const auto &wall : mp_plan->GetWalls())
      if (wall->GetId() == i_id)
        {
        o_begin_id = wall->GetBeginNode()->GetId();
        o_end_id = wall->GetEndNode()->GetId();
        o_begin = wall->GetBegin();
        o_end = wall->GetEnd();
        o_height = wall->GetHeight();
        return;
        }
    }


  const std::shared_ptr<CoreData::Wall> Project::GetWallById(long long i_id) const
    {
    for (const auto &wall : mp_plan->GetWalls())
      if (wall->GetId() == i_id)
        {
        return wall;
        }
    return nullptr;
    }
```

```
const std::shared_ptr<CoreData::WallNode> Project::GetWallNodeById(long long
i_id) const
  {
  for (const auto& node : mp_plan->GetWallNodes())
    {
    if (node->GetId() == i_id)
      {
      return node;
      }
    }
  return nullptr;
  }

  const std::shared_ptr<CoreData::Furniture> Project::GetFurnitureById(long long
i_id) const
  {
  for (const auto& furniture : mp_plan->GetFurniture())
    {
    if (furniture->GetId() == i_id)
      {
      return furniture;
      }
    }
  return nullptr;
  }

 void Project::GetFurnitureParamById(
  long long i_id,
  CoreMath::Vector2D &o_pos,
  double &o_direction_angle,
  long long &o_width,
  long long &o_length,
  long long &o_height,
  CoreData::FurnitureType &o_type,
  bool &o_valid) const
  {
  for (const auto &furniture : mp_plan->GetFurniture())
    if (furniture.get()->GetId() == i_id)
      {
      o_pos = furniture.get()->GetPosition();
      o_direction_angle = furniture.get()->GetDirectionAngle();
      o_width   = furniture.get()->GetWidth();
      o_length  = furniture.get()->GetLength();
      o_height  = furniture.get()->GetHeight();
```

```
      o_type = furniture.get()->GetType();
      o_valid = furniture.get()->GetValid();
      return;
      }
  }

void Project::GetDoorParamById(
  long long i_id,
  long long &o_shift,
  long long &o_width,
  long long &o_height,
  bool &o_is_direction_up,
  long long &o_begin_node_id,
  long long &o_end_node_id) const
  {
  for (const auto &wall : mp_plan->GetWalls())
    for (const auto &door : wall->GetDoors())
      if (door->GetId() == i_id)
        {
        o_shift = door->GetShift();
        o_width = door->GetWidth();
        o_height = door->GetHeight();
        o_is_direction_up = door->GetIsDirectionUp();
        o_begin_node_id = wall->GetBeginNode()->GetId();
        o_end_node_id = wall->GetEndNode()->GetId();
        return;
        }
  }

void Project::GetWindowParamById(
  long long i_id,
  long long& o_width,
  long long& o_height,
  long long& o_left_shift,
  long long& o_bottom_shift,
  long long& o_begin_node_id,
  long long& o_end_node_id)
  {
  for (const auto &window : mp_plan->GetWindows())
    if (window.get()->GetId() == i_id)
      {
      o_width = window->GetWidth();
      o_height = window->GetHeight();
      o_left_shift = window->GetLeftShift();
      o_bottom_shift = window->GetBottomShift();
```

```
        o_begin_node_id = window->GetBeginNodeId();
        o_end_node_id = window->GetEndNodeId();
        return;
        }
    }

  void Project::ReselectObjectsByIDs(std::vector<long long> i_ids)
    {
    UnselectAll();

    PropertiesController::GetInstance().DeletePropertiesWindow();
    for (auto r_iter = m_items_map.rbegin(); r_iter != m_items_map.rend(); r_iter++)
     if (std::find(i_ids.begin(), i_ids.end(), r_iter->first) != i_ids.end())
       {
       r_iter->second->Select();
       }

    _UpdateSelectedObjectsProperties();
    }

  bool Project::IsPlanEmpty()
    {
    return (mp_plan->GetWalls().empty() && mp_plan->GetFurniture().empty());
    }
```

Viewer3D.h:

```
#pragma once
#include "_Library.h"

#include <Editor2D/Project.h>

#include <QDialog>
#include <thread>
#include <mutex>

namespace Ui
  {
  class Viewer3D;
  }

class MainWidget;

class VIEWER_3D_API Viewer3D : public QDialog
  {
    Q_OBJECT

  public:
    explicit Viewer3D(const CoreData::Plan& i_plan, QWidget* ip_parent = nullptr);
    ~Viewer3D();

    void SetWallTexturePath(const QString& i_wall_texture);
    void SetFloorTexturePath(const QString& i_floor_texture);
```

```
    void SetDoorTexturePath(const QString& i_door_texture);

  protected:
    bool event(QEvent* ip_event) override;

    void mousePressEvent(QMouseEvent* ip_event) override;
    void mouseMoveEvent(QMouseEvent* ip_event) override;
    void keyPressEvent(QKeyEvent* ip_event) override;
    void keyReleaseEvent(QKeyEvent* ip_event) override;

  private:
    const int m_update_event_type;
    bool m_is_alive;
    std::mutex m_alive_mutex;
    std::thread m_update_event_thread;
    std::unique_ptr<Ui::Viewer3D> mp_ui;
    MainWidget* mp_widget3D;
  };
```

Viewer3D.cpp:

```
#include "stdh.h"
#include "Viewer3D.h"
#include "ui_Viewer3D.h"
#include "mainwidget.h"

#include "mainplane.h"
#include "wall3D.h"
#include "Door3D.h"
#include "window3D.h"


Viewer3D::Viewer3D(const CoreData::Plan& i_plan, QWidget* ip_parent)
  : QDialog(ip_parent)
  , m_update_event_type(QEvent::registerEventType())
  , m_is_alive(true)
  , mp_ui(std::make_unique<Ui::Viewer3D>())
  {
  mp_ui->setupUi(this);
  setWindowFlags(windowFlags() | Qt::WindowMinMaxButtonsHint);
  mp_widget3D = new MainWidget(i_plan, mp_ui->frame);
  QVBoxLayout* layout = new QVBoxLayout(mp_ui->frame);
  layout->addWidget(mp_widget3D);
  mp_ui->frame->setLayout(layout);

  m_update_event_thread = std::thread([this]()
    {
    while (true)
      {
      m_alive_mutex.lock();
      if (m_is_alive)
        {
        m_alive_mutex.unlock();
        QEvent* update_event = new QEvent(static_cast<QEvent::Type>(m_update_event_type));
        QCoreApplication::postEvent(this, update_event);
        std::this_thread::sleep_for(std::chrono::milliseconds(8));
        }
      else
        {
        m_alive_mutex.unlock();
        break;
        }
```

```
        }
      });
    }

Viewer3D::~Viewer3D()
  {
  m_alive_mutex.lock();
  m_is_alive = false;
  m_alive_mutex.unlock();
  if(m_update_event_thread.joinable())
    m_update_event_thread.join();
  }

bool Viewer3D::event(QEvent* ip_event)
  {
  if (ip_event->type() == static_cast<QEvent::Type>(m_update_event_type))
    {
    mp_widget3D->UpdateMovement();
    return true;
    }

  return QDialog::event(ip_event);
  }

void Viewer3D::SetWallTexturePath(const QString& i_wall_texture)
  {
  Wall3D::mg_wall_texture = i_wall_texture;
  }

void Viewer3D::SetFloorTexturePath(const QString& i_floor_texture)
  {
  Plane::mg_floor_texture = i_floor_texture;
  }

void Viewer3D::SetDoorTexturePath(const QString& i_door_texture)
  {
  Door3D::mg_door_texture = i_door_texture;
  }

void Viewer3D::mousePressEvent(QMouseEvent* ip_event)
  {
  mp_widget3D->MousePressEvent(ip_event);
  if (!ip_event->isAccepted())
    QDialog::mousePressEvent(ip_event);
  }

void Viewer3D::mouseMoveEvent(QMouseEvent* ip_event)
  {
  mp_widget3D->MouseMoveEvent(ip_event);
  if (!ip_event->isAccepted())
    QDialog::mouseMoveEvent(ip_event);
  }

void Viewer3D::keyPressEvent(QKeyEvent* ip_event)
  {
  mp_widget3D->KeyPressEvent(ip_event);
  if (!ip_event->isAccepted())
    QDialog::keyPressEvent(ip_event);
  }

void Viewer3D::keyReleaseEvent(QKeyEvent* ip_event)
  {
  mp_widget3D->KeyReleaseEvent(ip_event);
  if (!ip_event->isAccepted())
    QDialog::keyReleaseEvent(ip_event);
```

```
  }
```

Model3D.h:

```
#pragma once
#include "geometryengine.h"

class Model3D : public GeometryEngine
  {
  public:
    Model3D();
    Model3D(const QString& i_model_filename, const QString& i_texture_filename);

    void SetModel(const QString& i_model_filename, const QString& i_texture_filename);
  };
```

Model3D.cpp:

```
#include "stdh.h"
#include "Model3D.h"

namespace
  {
  void LoadModelFromObj(const QString &obj, QVector<VertexData> &o_vertices, QVector<GLuint>
&o_indices)
    {
    QFile objfile(obj);
    if(!objfile.exists())
      return;

    objfile.open(QIODevice::ReadOnly);

    QVector<QVector3D> coords;
    QVector<QVector2D> textures;
    QVector<QVector3D> normals;

    QTextStream input(&objfile);

    while (!input.atEnd())
      {
      QString str = input.readLine();
      QStringList list = str.split(" ");
      if (list[0] == "v")
        {
        coords.append(QVector3D(list[1].toFloat(), list[2].toFloat(), list[3].toFloat()));
        }
      else if (list[0] == "vt")
        {
        textures.append(QVector2D(list[1].toFloat(), list[2].toFloat()));
        }
      else if (list[0] == "vn")
        {
        normals.append(QVector3D(list[1].toFloat(), list[2].toFloat(), list[3].toFloat()));
        }
      else if (list[0] == "f")
        {
        for(int i : {1, 3, 2})
          {
          QStringList vert = list[i].split("/");
          o_vertices.append(VertexData(coords[vert[0].toLong() - 1],
textures[vert[1].toLong() - 1], normals[vert[2].toLong() - 1]));
          o_indices.append(o_indices.size());
```

```
            }
          }
        }
      objfile.close();
      }
    }

Model3D::Model3D()
  {
  }

Model3D::Model3D(const QString& i_model_filename, const QString& i_texture_filename)
  {
  SetModel(i_model_filename, i_texture_filename);
  }

void Model3D::SetModel(const QString& i_model_filename, const QString& i_texture_filename)
  {
  QVector<VertexData> model_vertices;
  QVector<GLuint> model_indices;
  LoadModelFromObj(i_model_filename, model_vertices, model_indices);
  SetData(model_vertices, model_indices);
  SetTexture(i_texture_filename);
  }
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПЛАНУВАННЯ ІНТЕР’ЄРУ У 3D

Графічний матеріал

КПІ.ІП-з7109.045490.07.99

“ПОГОДЖЕНО”

Керівник проєкту:

Л. А. Іванова

Нормоконтроль:

К.І. Ліщук

Виконавець:

В. В. Богдан

Київ — 2021 року

3. Hide logo

Result	Error	Duration
OK	-	00:00:01.1413902

▼ Screenshots

Expected	Actual