Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»
Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка додатку для виконання лабораторних робіт у режимі інтерактивної симуляції на тему "Дослідження магнітного поля"»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1812»

_____ /Андрій СЕРБЕНЮК/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /доц. Олена КУРОП'ЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Олена КУРОП'ЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань
Студент _____
(підпис)

Дніпро 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Development of an application for use in laboratory assignments in interactive mode on the topic of "Magnetic field exploration"»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1812: /Andrii SERBENIUK/

Scientific Supervisor: /Olena KUROPIATNYK/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Факультет «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)

Дата 09.12.2021

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра
студенту Сербенюку Андрію Васильовичу
1. Тема роботи: «Розробка додатку для виконання лабораторних робіт у режимі інтерактивної симуляції на тему "Дослідження магнітного поля"»
Керівник роботи: Куроп'ятник Олена Сергіївна, доцент
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: 28.06.2022 р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки:
4.1 Збір та аналіз вимог;
4.2 Зовнішнє проєктування;
4.3 Внутрішнє проєктування;
4.4 Розробка програми;
4.5 Тестування.
5. Перелік графічного матеріалу:
5.1. Опис предметної області;
5.2. Опис побудови симуляції;
5.3. Демонстрація виконання програми.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка технічного завдання	31.01.2022 – 18.02.2022	
2	Проєктування системи	19.02.2022 – 11.03.2022	
3	Написання програмного коду	14.03.2022 – 01.05.2022	
4	Тестування та налагодження програми	02.05.2022 – 24.05.2022	
5	Розробка, узгодження, та затвердження програмної документації	25.05.2022 – 10.06.2022	
6	Постановка технічного завдання	31.01.2022 – 18.02.2022	
7	Подання кваліфікаційної роботи до кафедри	08.06.2022	
8	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	22.06.2022	

Студент _____ (підпис) Андрій СЕРБЕНЮК (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ (підпис) доц. Олена КУРОП'ЯТНИК (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка складається з 9 розділів:

- вступ – в даному розділі описується сутність розробки, її актуальність. Складається з 2 сторінок;
- збір вимог до програмного забезпечення – в цьому розділі проводиться аналіз аналогів та літератури по даній предметній області, а також проводиться опитування зацікавлених сторін для формування найбільш повних вимог до програмного забезпечення. Складається з 6 сторінок;
- зовнішнє проєктування – в цьому розділі проведений огляд вхідних і вихідних даних, формалізація задачі. Складається з 8 сторінок;
- внутрішнє проєктування – в цьому розділі приводиться опис об’єктно-орієнтованого проектування, проектування інтерфейсу користувача, ескізи форм, аналіз проекту, проектування динаміки системи. Складається з 13 сторінок;
- розробка програми – розробка фізичного проекту, пояснюються обгрунтовуються рішення, прийняті під час розробки. Складається з 8 сторінок;
- тестування та налагодження – включає в себе вибір стратегії тестування, опис тестів методами «чорного» та «білого» ящиків. Складається з 9 сторінок;
- висновки – підсумки процесу розробки та її результатів. Складається з 1 сторінки;
- список літератури – включає в себе бібліографічний список використаної літератури. Складається з 1 сторінки;
- додатки – містить лист затвердження, специфікації, опис програми, текст програми, керівництво з використання.

Кількість таблиць: 6 штук.

Кількість рисунків: 15 штук.

Ключові слова: індукція, магнітне поле, струм.

ЗМІСТ

УМОВНІ ПОЗНАЧЕННЯ

А – ампер, одиниця вимірювання сили струму.

Тл – тесла, одиниця вимірювання сили індукції магнітного поля.

R – радіус одного витка, по якому проходить струм.

Гн – генрі, одиниця вимірювання індуктивності електричного контуру, що збуджує магнітний потік в один вебер за сили постійного струму у ньому в один ампер.

ВСТУП

Актуальність виконання цієї роботи полягає у відсутності великої кількості аналогів з такою ж тематикою. Хоча фізичні симуляції це досить відома галузь наукових досліджень, тем для цих симуляцій значно більше, ніж на сьогоднішній день проводиться досліджень.

Сьогодень, розробка фізичних симуляцій в основному направлена на галузі механіки, термодинаміки, фізики полімерів, та біофізики. Вони популярні через те, що теми їх досліджень досить актуальні в нашому повсякденному житті. Теми цих досліджень направлені на покращення наших фактичних умов життя, покращення нашого розуміння всесвіту в цілому, шляхом симуляцій реального світу в віртуальних умовах. З відносно постійною періодичністю з’являються нові матеріали досліджень на теми фізичних симуляцій, в яких покращуються відповідність симуляції до реального світу, час виконання симуляції, потреби в системних ресурсах пристрою, на якому виконується симуляція.

Тематика електромагнетизму в фізичних симуляціях розвинута не дуже сильно, і якщо і можна знайти деякі симуляції на цю тему, вони здебільшого обмежуються тільки конкретним експериментом. В таких випадках, ці симуляції не можна використовувати для демонстрації інших явищ.

Доцільність цієї роботи полягає в тому, що для фізичного експерименту, який є темою цієї розробки, немає актуальних симуляцій.

Метою цієї роботи є розробка додатку, який дозволив би проводити симуляції експерименту на тему визначення магнітного поля в центрі колового витка зі струмом.

Розроблений додаток надасть користувачам можливість проводити вказаний експеримент у віртуальній лабораторії. В тому числі, він дозволятиме проводити експерименти без необхідності в фізичному обладнанні. Звісно, окрім машини, на якому запущено додаток. Також, наприклад, для віртуальної симуляції можна задавати параметри, яких важко досягнути при проведенні експерименту в реальному світі.

Розроблений додаток можна використовувати, наприклад, в навчальних закладах із дистанційним навчанням. Також, завдяки простоті проведення віртуальних симуляцій, порівняно з реальним світом, брати участь в їх проведенні зможуть навіть люди без попереднього досвіду роботи з приладдям, необхідним для його проведення.

Також, експеримент передбачає роботу зі струмом. При його проведенні в віртуальній лабораторії виключається можливість виникнення травм від чинників які можуть виникнути під час його проведення в фізичній лабораторії.

1 ЗБІР ТА АНАЛІЗ ВИМОГ

1.1 Функціональні вимоги

Методом бесіди із замовником, було отримано функціональні вимоги.

Програма повинна дозволяти проводити симуляції експерименту визначення індукції магнітного поля в центрі колового витка зі струмом. Умова експерименту наступна.

Електромагнітна індукція – явище створення в просторі вихрового електричного поля змінним магнітним потоком. Одним із наслідків електромагнітної індукції є зв'язок між змінними електричним та магнітними полями в електромагнітній хвилі, інший наслідок, практично важливий для генерації електричного струму – виникнення електрорушійної сили в провідному контурі, магнітний потік через який змінюється [1].

В центрі колового витка зі струмом виникає магнітне поле, і силу його індукції можна визначити. Якщо в центрі витка розмістити магнітну стрілку, то кут повороту буде визначатися дією магнітного поля Землі і магнітного поля струму.

Порядок виконання роботи:

1 Підключити тангенс-гальванометр до джерела струму. Кількість витків N, які підключені до джерела занести до таблиці;

2 Встановити силу струму за вказівкою викладача;

3 Виміряти кут повороту магнітної стрілки;

4 Обрахувати індукцію магнітного поля, створюваного витками;

5 Визначити індукцію магнітного поля, створюваного одним витком;

6 Для кожного значення сили струму обрахувати теоретичну індукцію магнітного поля.

Завдання програми в цьому процесі – замінити собою лабораторне обладнання та надати середовище для проведення даного експерименту. Вона повинна дозволяти проводити симуляції даного експерименту, визначати силу індукції, кут повороту магнітної стрілки всередині витка.

1.2 Дослідження предметної області

Магнітна індукція – це векторна фізична величина, яка позначає силу дії магнітного поля на заряджені частинки і тіла, які мають магнітний момент і рухаються відносно даного магнітного поля. Одиницею вимірювання магнітної індукції є тесла (Тл)

Індукція магнітного поля в центрі колового витка зі струмом визначається за формулою:

$$B_в = \mu\mu_0 \frac{I}{2R}, \quad (1.1)$$

де:

μ – відносна проникність середовища (для повітря μ=1);

μ_0 – магнітна стала ($\mu_0=4\cdot\pi\cdot10^{-7}$ Гн/м);

I – сила струму у витку;

R – радіус витка.

Якщо в центрі витка розмістити магнітну стрілку, то кут повороту буде визначатися дією магнітного поля Землі і магнітного поля струму. Під дією магнітного поля Землі та магнітного поля витка магнітна стрілка повернеться на кут α:

$$\mathrm{tg}\alpha = \frac{B_в}{B_З}, \quad (1.2)$$

де:

$B_З$ – горизонтальна складова індукції магнітного поля Землі ($B_З=0{,}5\cdot10^{-5}$ Тл)

Індукцію магнітного поля, створюваного тільки одним витком, можна визначити за формулою:

$$B_1 = B_в/N. \quad (1.3)$$

1.3 Огляд аналогів

В даній галузі досить багато однотипних аналогів. Велику кількість інтерактивних фізичних симуляцій можна знайти, наприклад, на інтернет ресурсі phet.colorado.edu.

Density: Зображення інтерфейсу даного аналогу можна побачити на рис.1.1.

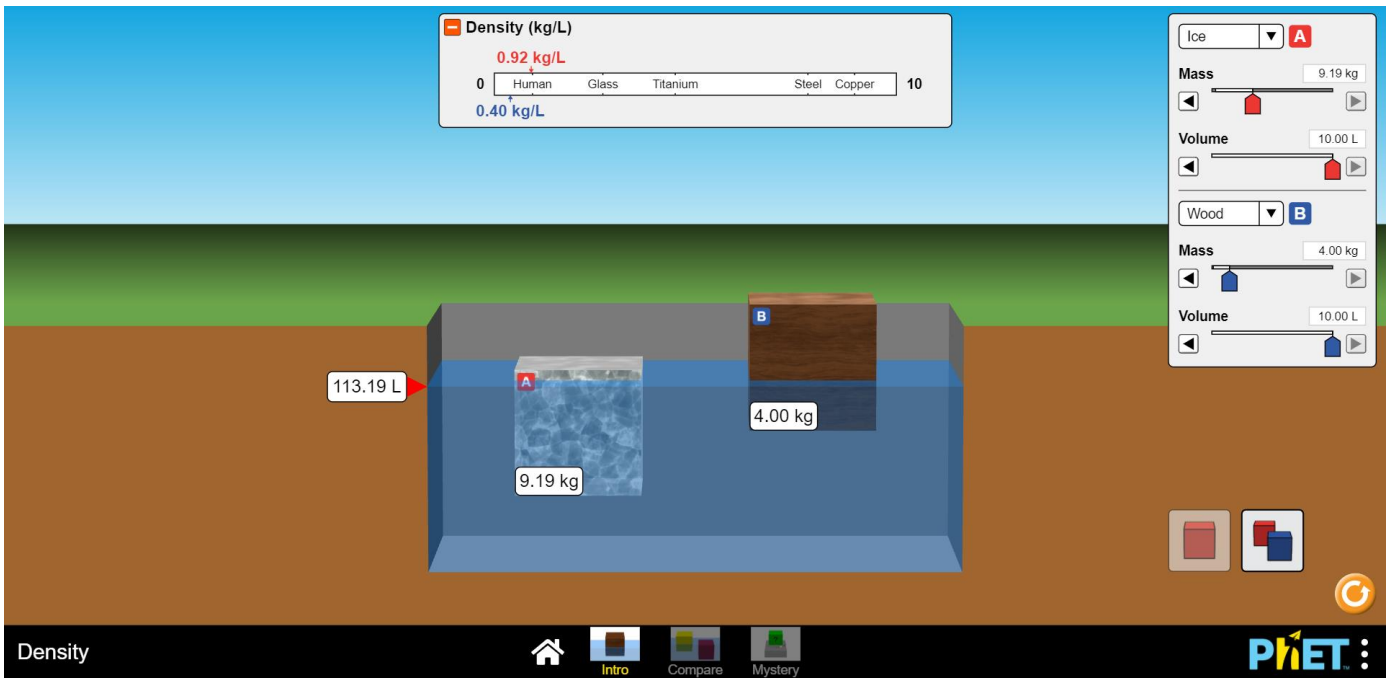

Рисунок 1.1 – Головне вікно аналогу «Density»

Даний аналог дозволяє проводити інтерактивні симуляції експериментів, пов’язаних із визначенням густини матеріалів.

Переваги:

- використання графічного інтерфейсу наглядно демонструє хід експерименту;
- можливість змінювати параметри експерименту, такі як розмір, маса, густина матеріалів;
- можливість взаємодії із об’єктами симуляції з допомогою миші;
- можливість перезапуску симуляції із початковими даними.

Недоліки:

- відсутність можливості зміни параметрів симуляції шляхом введення числових даних.

Висновок – використання графічного інтерфейсу та взаємодія із ним є дуже зручним та наглядним способом демонстрації ходу експерименту;

Circuit Construction Kit: DC: Зображення інтерфейсу даного аналогу можна побачити на рис.1.2.

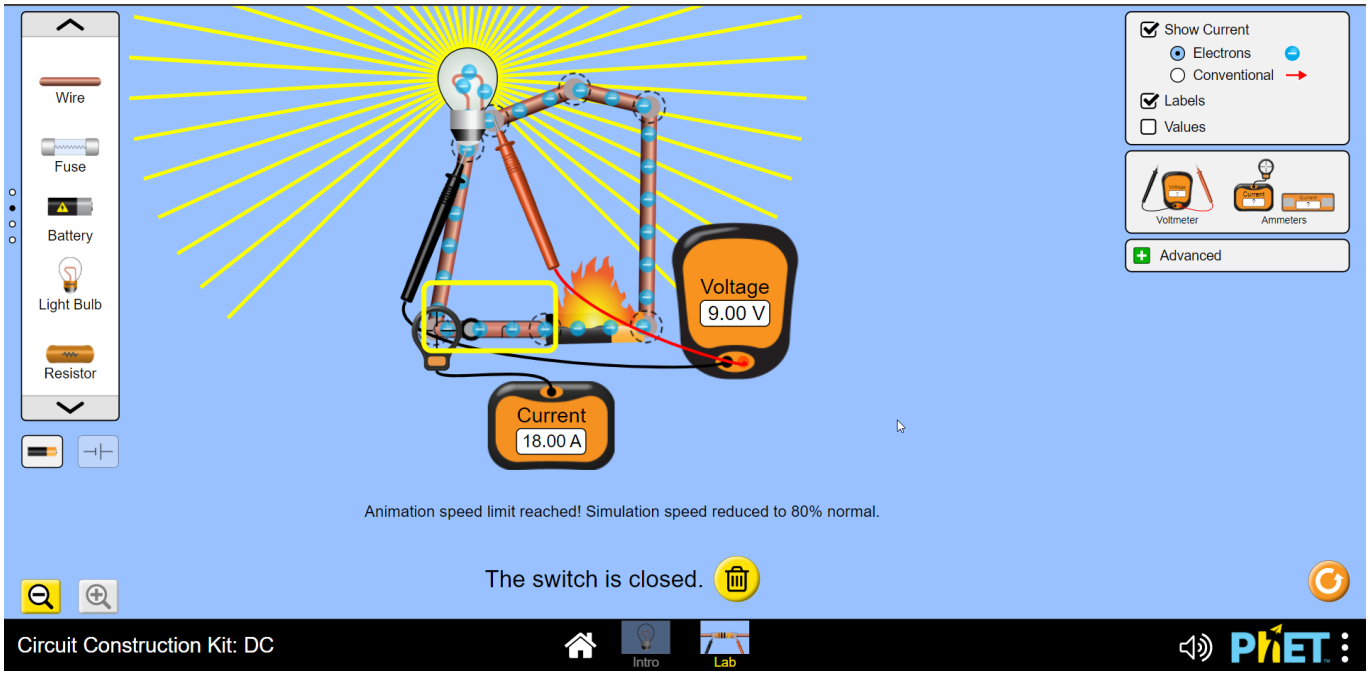

Рисунок 1.2 – Головне вікно аналогу «Circuit Constructor Kit: DC»

Даний аналог дозволяє проводити інтерактивні симуляції створення та редагування електричних схем.

Переваги:

- використання графічного інтерфейсу наглядно демонструє хід експерименту;
- можливість змінювати параметри експерименту;
- можливість взаємодії із об’єктами симуляції з допомогою миші;
- можливість перезапуску симуляції із початковими даними.

Недоліки:

- відсутність можливості зміни параметрів симуляції шляхом введення числових даних.

Висновок – як і в попередньому випадку, використання графічного інтерфейсу та взаємодія із ним є дуже зручним та наглядним способом демонстрації ходу експерименту. Однак, для числових даних не вистачає можливості їх ручного введення з клавіатури. Тому, в процесі розробки потрбно забезпечити можливість введення даних з допомогою клавіатури.

Висновки до розділу 1

Було зібрано функціональні вимоги до програмного забезпечення та проведено огляд аналогів.

Необхідно розробити програму для проведення експерименту із дослідження магнітного поля у віртуаьній лабораторії.

В галузі фізичних симуляцій досить багато аналогів, однак аналогу, з допомогою якого можна було б провести експеримент із функціональних вимог так і не знайшлось. Тому, прямих аналогів – немає.

Аналоги здебільшого використовують графічний інтерфейс із можливістю взаємодії із елементами симуляції з допомогою миші. Подібний спосіб дуже наглядний та інтуїтивно зрозумілий користувачу.

Однак, в переглянутих аналогах не було можливості введення параметрів симуляції з клавіатури, а тільки з допомогою повзунків та стрілок для інкременту та декременту параметрів. Такий спосіб модифікації параметрів симуляції менш контрольований, ніж ручне введення з клавіатури. Якщо користувачу необхідно ввести конкретне значення параметру, то існує можливість, що він не зможе ввести його без змін, оскільки крок інкременту може не дозволяти вводити значення деякої точності. Тому – при проєктуванні потрібно врахувати можливість введення параметрів симуляції з допомогою клавіатури.

2 ЗОВНІШНЄ ПРОЄКТУВАННЯ

2.1 Функціональне призначення

Програмне забезпечення, що розроблюється, повинне надавати можливість проводити інтерактивні симуляції експерименту із дослідження магнітного поля – визначати індукцію магнітного поля в центрі колового витка зі струмом. Програма повинна симулювати фізику реального світу: під час експерименту, при зміні радіусу витка, кількості витків, або сили струму – кут повороту магнітної стрілки в центрі витка змінюється. Повинна бути можливість вводити власні параметри симуляції.

2.2 Експлуатаційне призначення

Призначенням цієї програми є заміна собою справжнього фізичного устаткування для проведення даного експерименту. Це надасть користувачу можливість проводити даний експеримент, навіть за відсутності потрібного фізичного устаткування.

Часто бувають випадки, коли лабораторне приладдя зламане, не відкаліброване, або якимось іншим чином непридатне до використання. Все що потрібно для проведення експерименту у віртуальній лабораторії – це комп’ютер для запуску програми, та сама програма.

Також, як радикальний варіант цього випадку, може приладдя взагалі немає. Таке буває, в тому числі, під час дистанційного навчання. Не у кожної людини вдома є необхідне приладдя для проведення експериментів із фізики. Однак, при наявності віртуальної лабораторії, учні зможуть проводити експерименти і не перебуваючи безпосередньо в лабораторії.

2.3 Функціональні вимоги

Програмне забезпечення повинне являти собою віртуальну лабораторію для проведення експерименту із визначення сили індукції магнітного поля в центрі колового витка зі струмом.

Хід експерименту у віртуальній лабораторії:

1 Запуск програми – після запуску програми у параметрів симуляції вже є деякі початкові значення параметрів;

2 Введення параметрів симуляції;

3 Спостереження результатів симуляції;

4 Перехід на крок 2 або крок 5;

5 Вихід з програми.

Виходячи із даного сценарію, програма повинна:

- забезпечити введення параметрів симуляції:
  - з допомогою клавіатури, для більш точного значення;
  - за допомогою інтерактивних елементів графічного інтерфейсу, для тих значень, для яких це має сенс;
- проводити симуляцію – підраховувати значення параметрів симуляції, виходячи із заданих значень інших параметрів;
- показувати користувачу результати симуляції:
  - числові значення результатів;
  - графічне відображення в графічному інтерфейсі – деякий ескіз фізичного середовища;
- надавати можливість повторного проведення симуляції із новими параметрами:
  - мати деякі стандартні початкові значення параметрів симуляції.

Сценарії використання функцій програми можна побачити на діаграмі прецедентів [6] (рис. 2.1).

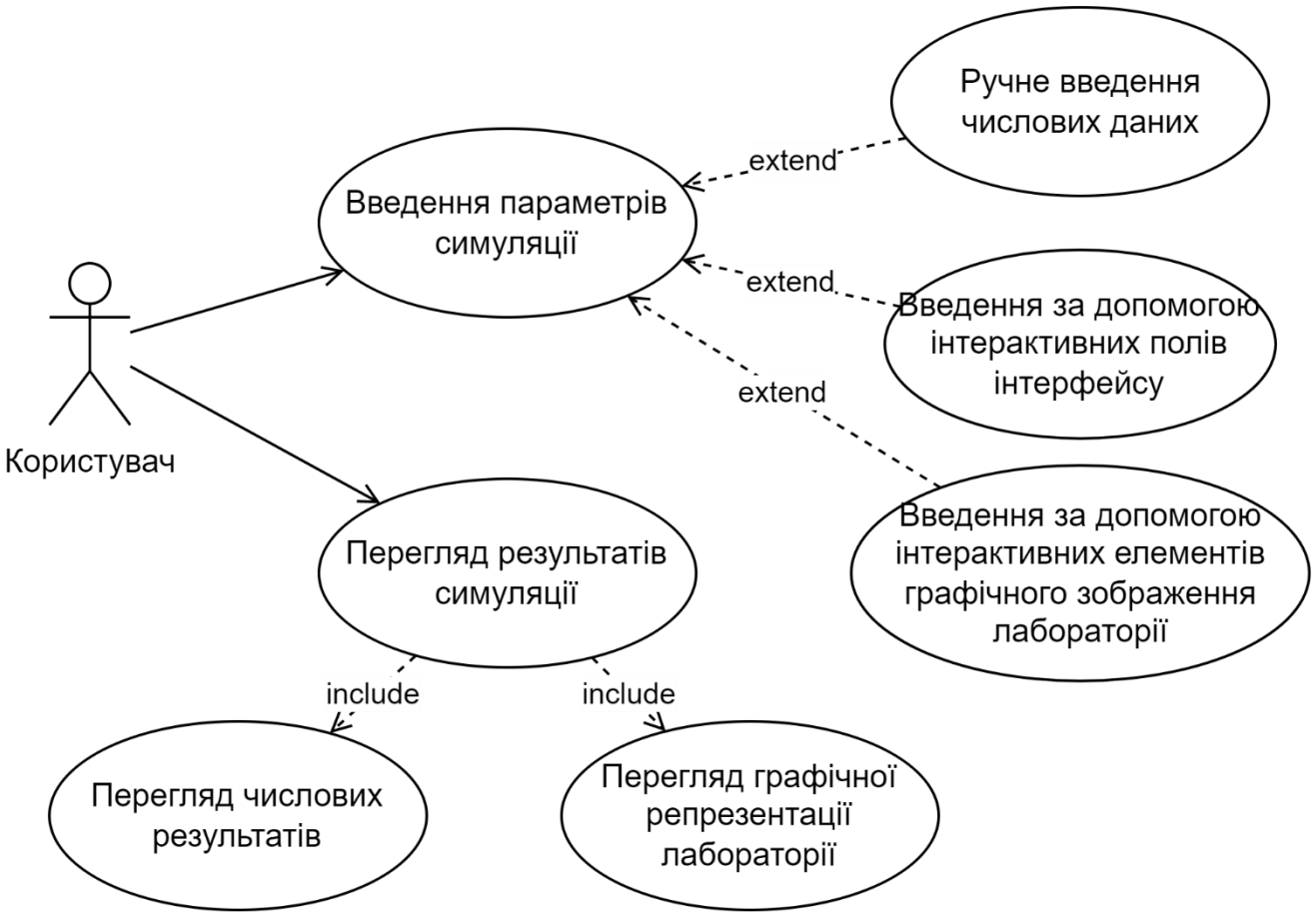

Рисунок 2.1 – Діаграма прецедентів

2.4 Вхідні дані

Вхідними даними програми є:

- Параметри симуляції:
  - coil_radius – радіус одного витка: ціле число в діапазоні від 1 до 100, крок 1;
  - coil_number – кількість витків: ціле число від 1 до 30, крок 1;
  - el_current – сила струму: дійсне число в діапазоні від 0 до 200 із точністю 10^{-3}, тобто три числа після коми.

2.5 Вихідні дані

Вихідними даними програми є:

- числові дані:
  - induction – сила індукції магнітного поля, утвореного витками: дійсне число із точністю 10^{-10}, тобто десять чисел після коми;

- compass_angle – кут повороту магнітної стрілки всередині колового витка: дійсне число із точністю 10^{-4}, тобто чотири числа після коми, в діапазоні від 0 до 359,9999. Одиниця вимірювання – градуси;

- графічне відображення:
  - кут повороту магнітної стрілки всередині колового витка: зображення магнітної стрілки, повернутої на числове значення цього з параметру;
  - стрілочний амперметр: зображення стрілочного амперметра, яке відображає числове значення сили струму, що протікає по коловому витку.

Приклад графічного зображення магнітної стрілки, та векторів дії інших магнітних полів на неї можна побачити на рис 2.2.

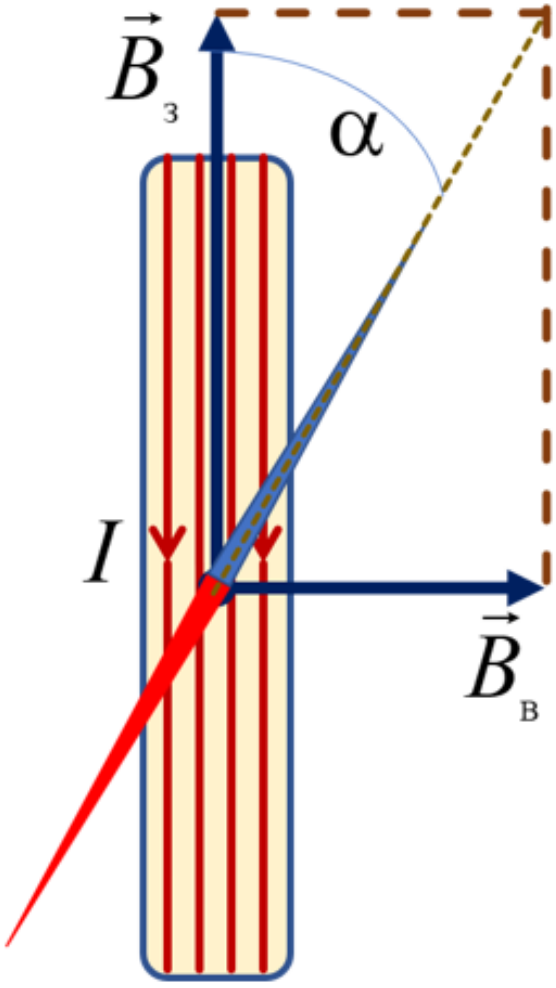

Рисунок 2.2 – Приклад зображення магнітної стрілки

2.6 Опис зовнішнього інформаційного середовища

Дані coil_radius, coil_number, el_current вводяться з клавіатури в поля з контролем введення. Для даних el_current є сенс введення з допомогою елементу інтерфейсу із шкалою, оскільки це частково нагадує зміну сили електричного струму на аналогових пристроях.

Закінчений продукт для свого функціонування іншого особливого програмного забезпечення не потребує.

Висновки до розділу 2

В процесі зовнішнього проектування програмного продукту було визначено функціональні вимоги до програми, вхідні та вихідні дані, було визначено функціональне та експлуатаційне призначення.

Оскільки програма розрахована на масове використання різними групами користувачів, можна зробити висновок, що для роботи з програмою користувач повинен володіти комп’ютером на хоча б мінімальному рівні. Розробку інтерфейсу користувача та схему взаємодії із програмою варто розробляти, виходячи з припущення, що нею буде користуватись людина з мінімальними знаннями по користуванню комп’ютером. Тобто, інтерфейс та взаємодія з програмою повинні бути максимально простими та зрозумілими.

На цьому етапі можна зазначити, що програмі для функціонування потрібен графічний інтерфейс. Якщо для роботи з тільки числовими даними симуляції було б достатньо і консольного інтерфейсу, він не підходить для графічної візуалізації результату. Крім того, програма розрахована на користувачів з хоча б мінімальними навичками користування комп’ютером, а консольний інтерфейс досить складний у використанні. Робота із ним здійснюється за допомогою команд, які потрібно запам’ятовувати, або в діалоговому режимі. Недосвідченому користувачу буде важко запам’ятовувати команди для роботи із консольною програмою, або навіть мати їх список і постійно дивитись в нього для виконання елементарних дій. Графічний інтерфейс, в якому чітко і структуровано викладено все з чим працює користувач, підходить в цій ситуації набагато краще.

Основним способом задання параметрів симуляції буде введення з клавіатури в поля із контролем значень, оскільки таким чином можна легко, швидко, і що найголовніше – точно, задати значення параметру.

Також було визначено, що для деяких параметрів є сенс задавати значення з допомогою інтерактивних елементів інтерфейсу, оскільки так симулюється керування цим параметром на аналогових пристроях, і тому – більш зрозуміле користувачу.

Вихідні дані буде представлено в двох форматах – в текстовому та графічному.

Подання результатів у текстовому вигляді надає можливість користувачу їх безпосереднього виділення та використання в інших програмах, в який ці дані можуть бути потрібні.

Графічне представлення результатів допомагає користувачу самому краще усвідомлювати їх, що робить покращує розуміння результатів симуляції.

3 ВНУТРІШНЄ ПРОЄКТУВАННЯ

3.1 Статична архітектура системи

Для моделювання словника системи виконаємо аналіз зовнішніх специфікацій системи.

Оскільки фізичні симуляції, які є ціллю цієї розробки, досить складні в своїй природі, необхідно сконструювати систему, яка була б достатньо гнучкою у використанні, зрозумілою в заданні її частин, та, за необхідності, могла підходити для нових способів проведення симуляції.

Тобто, від ядра системи вимагається:

- легкість в заданні екосистеми симуляції;
- незалежність працездатності всієї системи від математичних обчислень, які проводяться в процесі проведення симуляції – можливість розширення, та/або зміни самої симуляції.

На цьому етапі можна природно виділити одну з бажаних поведінок системи – виставлення залежностей параметрів симуляції одне від одного, оновлення їх значень при зміні параметрів, від яких вони залежать.

Найкращим прикладом суті цієї вимоги є принцип дії концепту «формули», а саме – значення залежить від значень інших аргументів. Якщо змінюється значення одного із аргументів – змінюється і значення функції.

Для реалізації подібної поведінки можна скористатись шаблоном проектування «наглядач» (англ. «Observer») [2].

Також, оскільки механізм створення формул повинен бути гнучким, основувати його на створенні окремих типів для кожної формули недоцільно. Тому – об’єкт формули повинен містити в собі саму функцію, яка безпосередньо виконує обчислення. Для цього відмінно підходить механізм зворотніх викликів [3].

Ідентифіковані сутності:

- симуляція – Simulation;
- параметр симуляції – SimParameter;
- формула параметру – Formula;
- залежності параметру – ParamDependencies;

- графічний інтерфейс користувача – MainGUI;
- графічне поле параметру симуляції – GUIDataUnit.

Ідентифіковані обов’язки:

- Simulation – контроль над проведенням симуляції, значень параметрів;
- SimParameter – збереження значення параметру симуляції, способів його обчислення, та запуск обчислення;
- Formula – безпосереднє обчислення значення параметру;
- ParamDependencies – параметри симуляції, які залежать від даного параметру. Відповідає за їх оновлення при зміні параметру;
- MainGUI – головна форма програми. Відповідає за організацію взаємодії із користувачем, відображення всіх графічних елементів програми;
- GUIDataUnit – елемент інтерфейсу програми, який відповдає за роботу із певним параметром симуляції. Має доступ до значення параметру, безпосередньо взаємодіє із графічним інтерфейсом для роботи з параметром.

Атрибути та операції, необхідні для виконання обов'язків кожної сутності-класу наведено в таблиці 3.1.

Таблиця 3.1 – Сутності системи, їх арибути та методи

Сутність	Атрибути	Методи
1	2	3
Simulation	parameters – параметри симуляції.	simulate – провести симуляцію
SimParameter	value – числове значення параметру; formulae – набір формул для обчислення параметру.	calculate – обчислити значення параметру за доступними фомулами

Продовження таблиці 3.1

1	2	3
Formula	args – аргуметри, необхідні формулі для обчислення; function – безпосередньо математична формула, яка обчислює числове значення параметру	calculate – безпосереднє обчислня числового значення параметру
ParamDependents	dependents – параметри, значення яких залежать від даного параметру.	notify – сповістити залежні параметра про необхідність перерахувати їх значення; add_dependent – додати залежний параметр.
MainGUI	gui_elements – елементи інтерфейсу; data_elements – контролери елементів інтерфейсу, які безпосередньо взаємодіють із параметрами симуляції; simulation – об’єкт симуляції.	gui_events – обробники подій графічного інтерфейсу; start_simulation – запуск симуляції.
GUIDataUnit	parameter – параметр симуляції, за який відповідає об’єкт сутності; gui_element – елемент інтерфейсу, з яким взаємодіє об’єкт сутності.	get_data – отримати значення параметру з графічного інтерфейсу; set_data – оновити значення параметру на графічному інтерфейсі.

Судячи із призначення сутностей, із можна поділити на функціональні групи, залежно від того що є їх відповідальністю.

Множинами класів, які працюють спільно для досягнення деякої поведінки є:

- MainGUI, GUIDataUnit – взаємодія із користувам з допомогою графічного інтерфейсу, представлення на графічному інтерфейсі параметрів симуляції, робота із їх заданням;
- Simulation, SimParameter, Formula – проведення математичних обчислень симуляції;
- SimParameter, Formula, ParamDependents – контроль значень параметрів симуляції, їх взаємодія одні з одними.

Структурні зв’язки між класами представлені на таблиці 3.2.

Таблиця 3.2 – Моделювання залежностей

Клас, який зв’язується	Клас, з яким зв’язуються	Зв’язок
MainGUI	GUIDataUnit	Композиція
	Simulation	Композиція
GUIDataUnit	SimParameter	Агрегація
Simulation	SimParameter	Композиція
SimParameter	ParamDependents	Композиця
	Formula	Асоціація
ParamDependents	SimParameter	Агрегація
Formula	SimParameter	Агрегація

Моделювання непрограмних сутностей для даної системи виконувати не потрібно, оскільки особливості апаратної частини не використовується в роботі системи, всі інші сутності вже представлено у вигляді класів.

Визначення примітивних типів виконаємо на етапі побудови діаграми класів.

Кінцевий результат проектування статичної архітектури системи представимо у вигляді діаграми класів [6] (рис. 3.1).

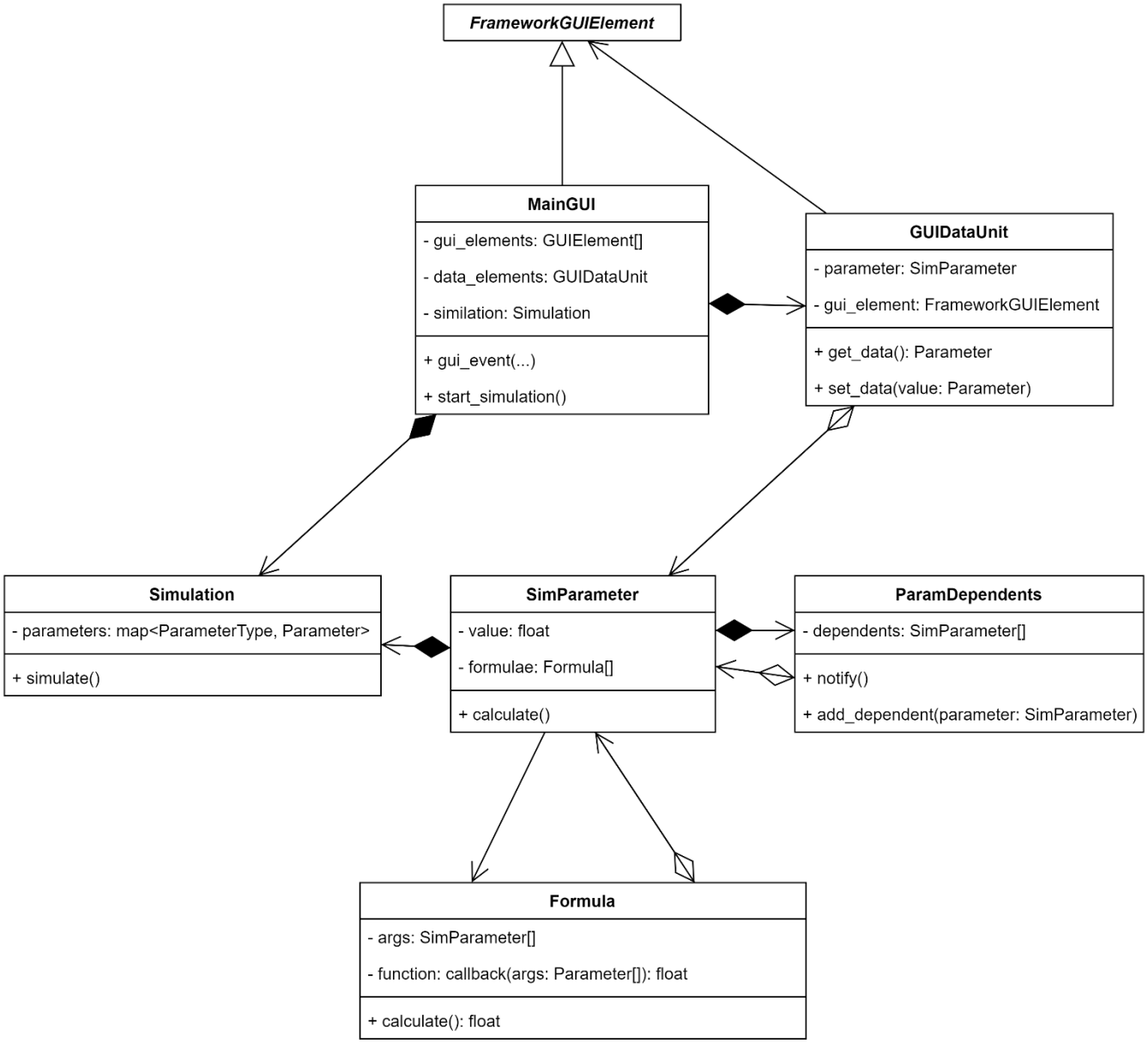

Рисунок 3.1 – Діаграма класів

3.2 Моделювання поведінки системи

Проаналізувавши діаграму прецедентів (див. рис. 2.1), можна визначити, що кінцевим та ключовим прецедентом завжди є перегляд результатів симуляції. Тому, опрацювавши його, можна розглянути поведінку всієї системи.

Ініціатором прецеденту є користувач, як зовнішній об’єкт системи. Хід роботи завжди прямолінійний: коли користувач змінює значення деякого з параметрів – симуляція перераховується та інші параметри змінюються.

Опис поведінки системи можна побачити на рис. 3.2.

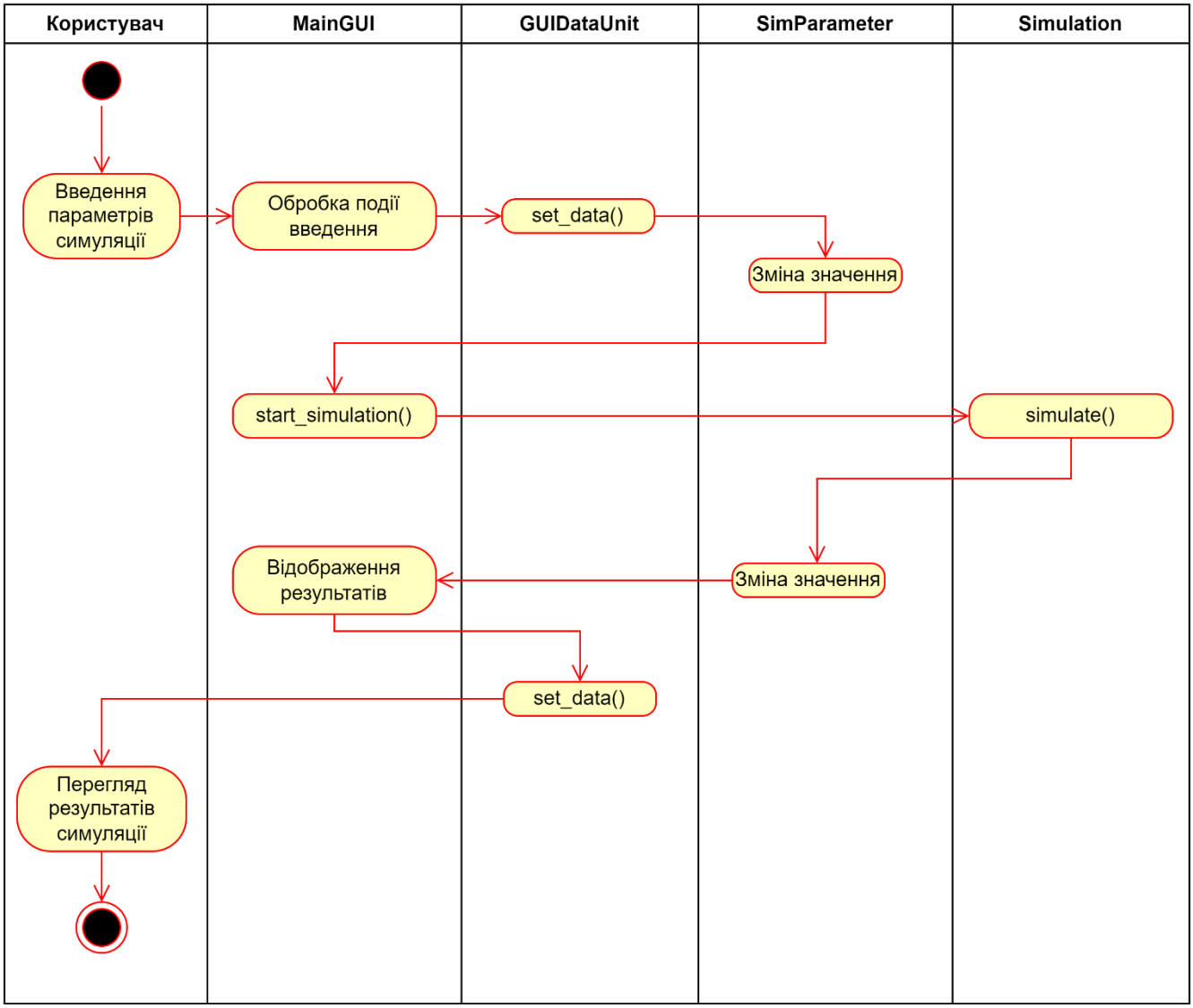

Рисунок 3.2 – Діаграма діяльності для прецеденту «Перегляд результатів симуляції»

3.3 Топологія системи

Передбачається, що система буде розроблена із використанням мови C++ та платформи QT.

Файли із кінцевим кодом матимуть розширення .h та .cc.

Артефактом розміщення для даної системи буде .exe файл, який можна створити із допомогою інтструменту CMake[4] із .h та .cc файлів. Зокрема, використання цього інструменту можливе із середовища розробки QT Creator, який рекомендовано використовувати при розробці програм із використанням платформи QT.

Для визначення топології системи та фізичного розміщення артефактів доцільним є використання діаграми компонентів.

Спроектовану діаграму компонентів даної системи зображено на рис. 3.3.

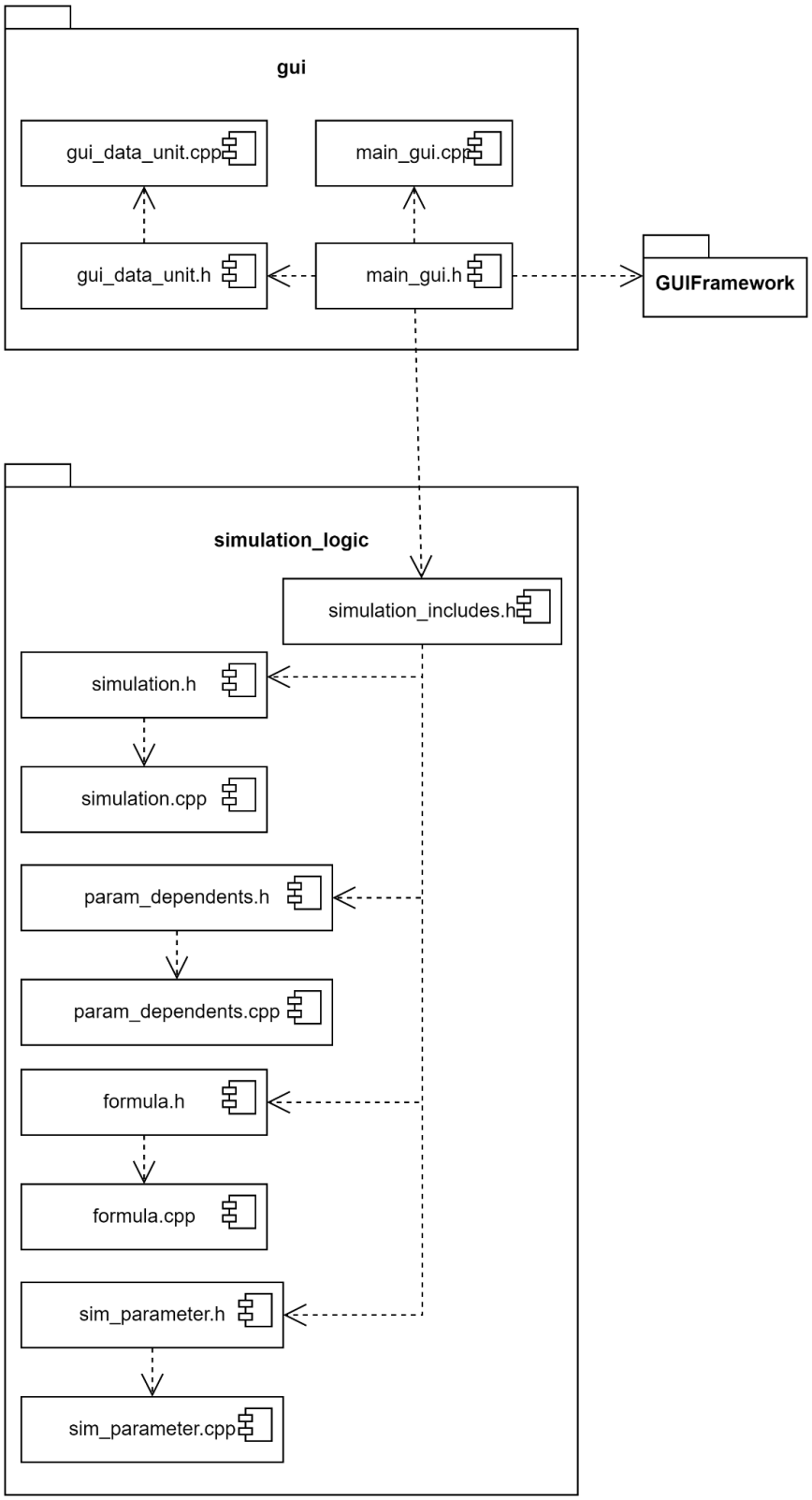

Рисунок 3.3 – Діаграма компонентів

Діаграма розгортання для цієї системи не буде інформативною, оскільки система не є розподіленою. Зовнішній вигляд діаграми в даному випадку зводиться до одного вузла-процесора.

3.4 Проєктування інтерфейсу користувача

3.4.1 Проєктування станів графічного інтерфейсу

Розглянувши діаграму прецедентів (див. рис. 2.1), можна зробити висновок, що єдині дії, які може виконувати користувач – це зміна параметрів симуляції, та перегляд її результату. Симуляція перезапускається автоматично при зміні параметрів, тому користувачу не потрібно робити це вручну.

На основі цих вимог була розроблена діаграма станів користувача програми, яка представлена на рис. 3.4.

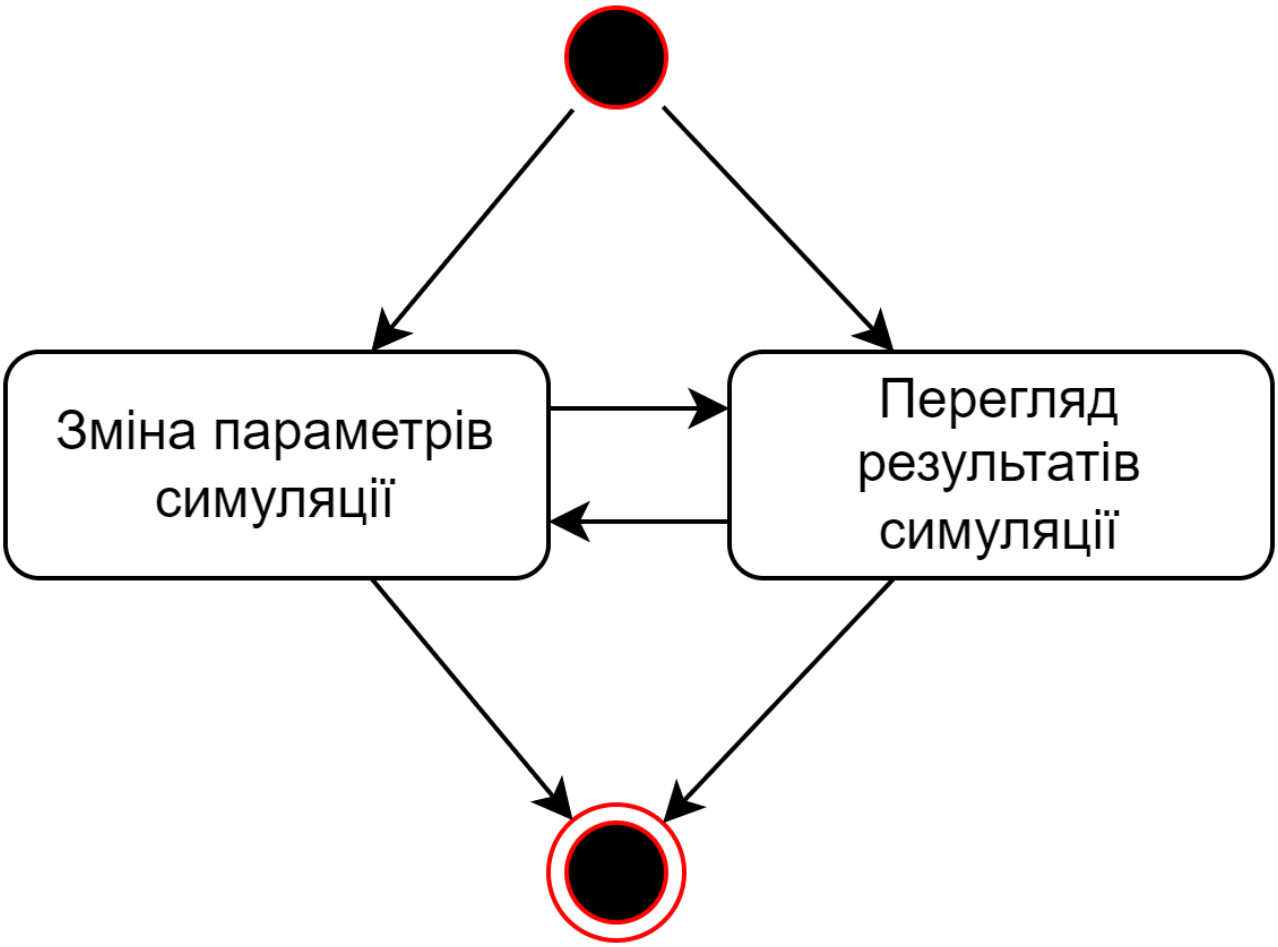

Рисунок 3.4 – Діаграма станів користувача програми

Оскільки кількість дій, які може виконувати користувач така обмежена, то потреби в обширному меню для роботи із користувачем не потрібно. Єдиним пунктом в меню користувача буде вихід з програми.

3.4.2 Проєктування ескізу графічного інтерфейсу

Графічний інтерфейс повинен містити:

1 Поля для введення параметрів:

1.1. З можливістю текстового введення;

1.2. З можливістю заданням з допомогою шкали;

2 Поля для відображення результатів симуляції:

2.1. У текстовому вигляді із можливістю їх копіювання;

2.2. У графічному вигляді;

3 Меню із пунктом «Вихід».

Окрім цього, поля із даними повинні мати випливаючі підказки, які пояснювали б користувачу призначення цих даних для симуляції.

На основі цього, можна спроєктувати ескіз графічного інтерфейсу. Він представлений на рис 3.5.

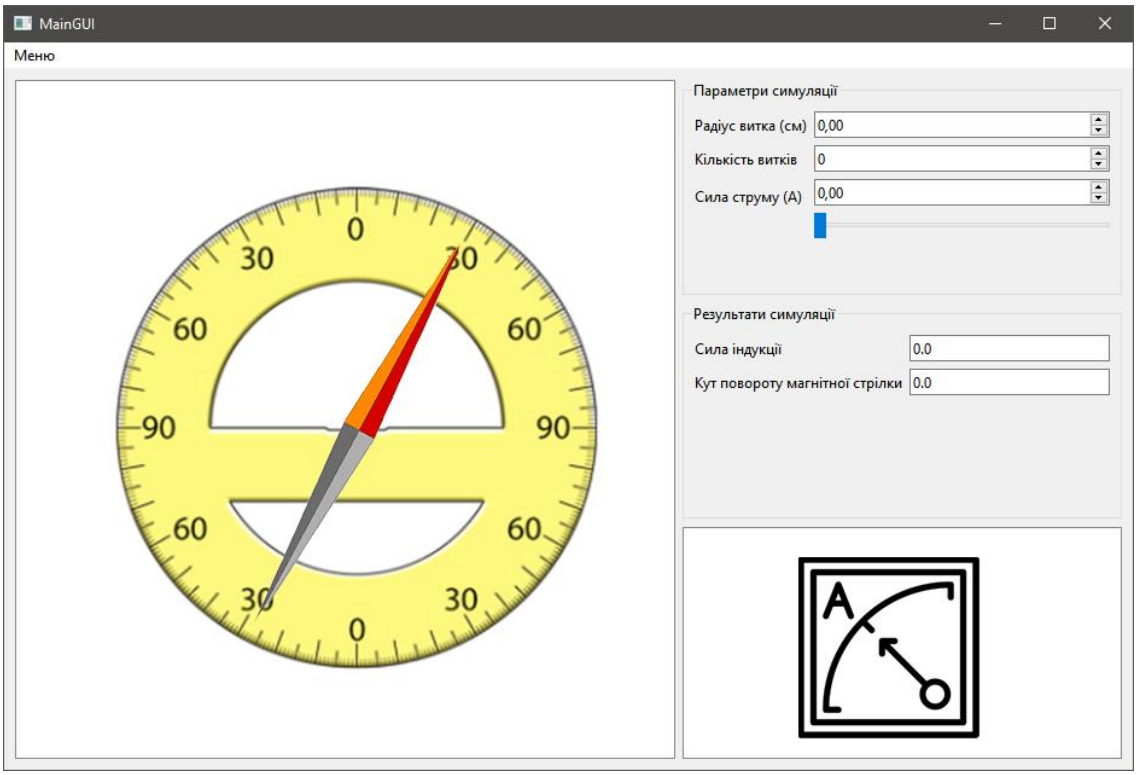

Рисунок 3.5 – Прототип графічного інтерфейсу

Висновки до розділу 3

В процесі внутрішнього проєктування було розроблено внутрішню архітектуру системи.

Було спроєктовано статичну архітектуру системи. В процесі проєктування було вирішено, що для представлення формул краще скористатись механізмом зворотних викликів, ніж наслідування. Це пов’язано з тим, що наслідування формул від деякого спільного інтерфейсу і перегрузка методів підрахунку для кожного з них більш затратна на час реалізації та ресурси системи під час роботи програми, та, як наслідок, менш гнучка в можливостях задання симуляції. Так, наприклад, потенційно можливо задавати формули для симуляції зі стороннього файлу в зрозумілій людині мові, яка могла б транслюватись у виконуваний код на етапі запуску програми, та надавати функцію-інтерфейс, яку можна використати як функцію зворотного виклику для симуляції. Таке рішення надає системі потенціал до майбутніх розширень і покращень.

До поведінки системи досить мало вимог, тому і її функціонал із точки зору користувача досить невеликий – все, що доступно користувачу, це зміна параметрів симуляції, та перегляд її результату. Спроєктований ескіз інтерфейсу користувача теж це відображає – єдині інтерактивні елементи на ньому використовуються для зміни параметрів симуляції.

Результатів проєктування достатньо, щоб розпочати розробку програмного коду.

4 РОЗРОБКА ПРОГРАМИ

Мовою програмування, вибраною для розробки програми, стала C++. Причиною вибору стала її швидкість виконання, оскільки фізичні симуляції можуть бути досить ресурсозатратними.

4.1 Графічний інтерфейс користувача

В якості основи для розробки графічного інтерфейсу було вирішено скористатись програмним каркасом QT [5], оскільки він надає простий і зручний спосіб створення графічних інтерфейсів із використанням мови програмування C++.

Оскільки програмі потрібно виводити результати в різних форматах(наприклад, графічне та числове відображення), необхідна стандартизація способу присвоєння значень елементам графічного інтерфейсу. Цю роль виконує клас GUIDataUnit. Вона є узагальненою та базовою для інших сутностей, які вже реалізують функціонал для конкретних елементів інтерфейсу. Цими сутностями є: SliderUnit, DoubleSpinBoxUnit, SpinBoxUnit, DoubleLineEditUnit, SceneUnit, MagFieldSceneUnit, AmpermeterSceneUnit. Окрім цього, сутність SceneUnit є базовою для сутностей, які реалізують графічне відображення значень.

Для запобігання циклічних викликів методів класу GUIDataUnit, до нього було додано атрибут should_update – він вказує, чи відбувається оновлення значення елемнту інтерфейсу на даний момент.

Також, від сутності GUIDataUnit було вирішено відділити окрему сутність – GUIParameter. Вона виконує роль обгортки для параметру симуляції, і забезпечує його взаємодію із GUIDataUnit.

4.2 Логіка симуляції

Перш за все, необхідно обрати тип для представлення числових значень симуляції. Оскільки фізичні значення це переважно дійсні числа, то потрібно вибрати тип із плаваютою комою. Також, в контектсті даної розробки, ці значення будуть активно використовуватись елементами графічного інтерфейсу, тому потрібен тип, який було б легко інтерпретувати елементам графічного інтерфейсу. Цим вимогам цілком відповідає тип double – він достатньо великий щоб вміщати

великі значення, і достатньо поширений щоб не виконувати конвертацій при його передачі в елементи інтерфейсу.

В процесі розробки було прийняте рішення обєднати сутність ParamDependents із сутністю SimParameter, оскільки ParamDependents не має достатньо чітких, відмінних від SimParameter обов'язків. Вона тепер являє собою звичайний масив, як атрибут сутності SimParameter.

Для задання формул обрахунку використовується сутність Formula. В об'єктах цього класу потрібно вказувати функцію зворотнього виклику, яка власне і обраховує значення параметру.

Для передачі та зберігання результатів обрахунку формул буде використовуватись тип ParamResult, який являє собою пару із числового значення, та прапорця, правильне це значення чи ні. Така міра потрібна, адже не завжди вхідні значення можуть бути правильними для обрахунку параметру. Наприклад, коли потрібне значення тільки в певному діапазоні, або у випадку ділення на нуль.

Оскільки симуляція задається у вигляді деякого графу залежностей одних параметрів симуляції від інших, в ньому можливі циклічні залежності. В найгіршому випадку це може привести до того, що значення параметру ніколи не закінчить обрахунок.

Для запобігання циклічним залежностям у сутності параметру є прапорець recalc_in_progress, який вказує, чи відбувається на даний момент перерахунок параметру чи ні. Якщо перерахунок уже відбувається – інший перерахунок того ж параметру в той самий момент починатись не буде.

Алгоритм перерахунку значення параметру можна представити з допомогою блок-схеми (див. рис. 4.1)

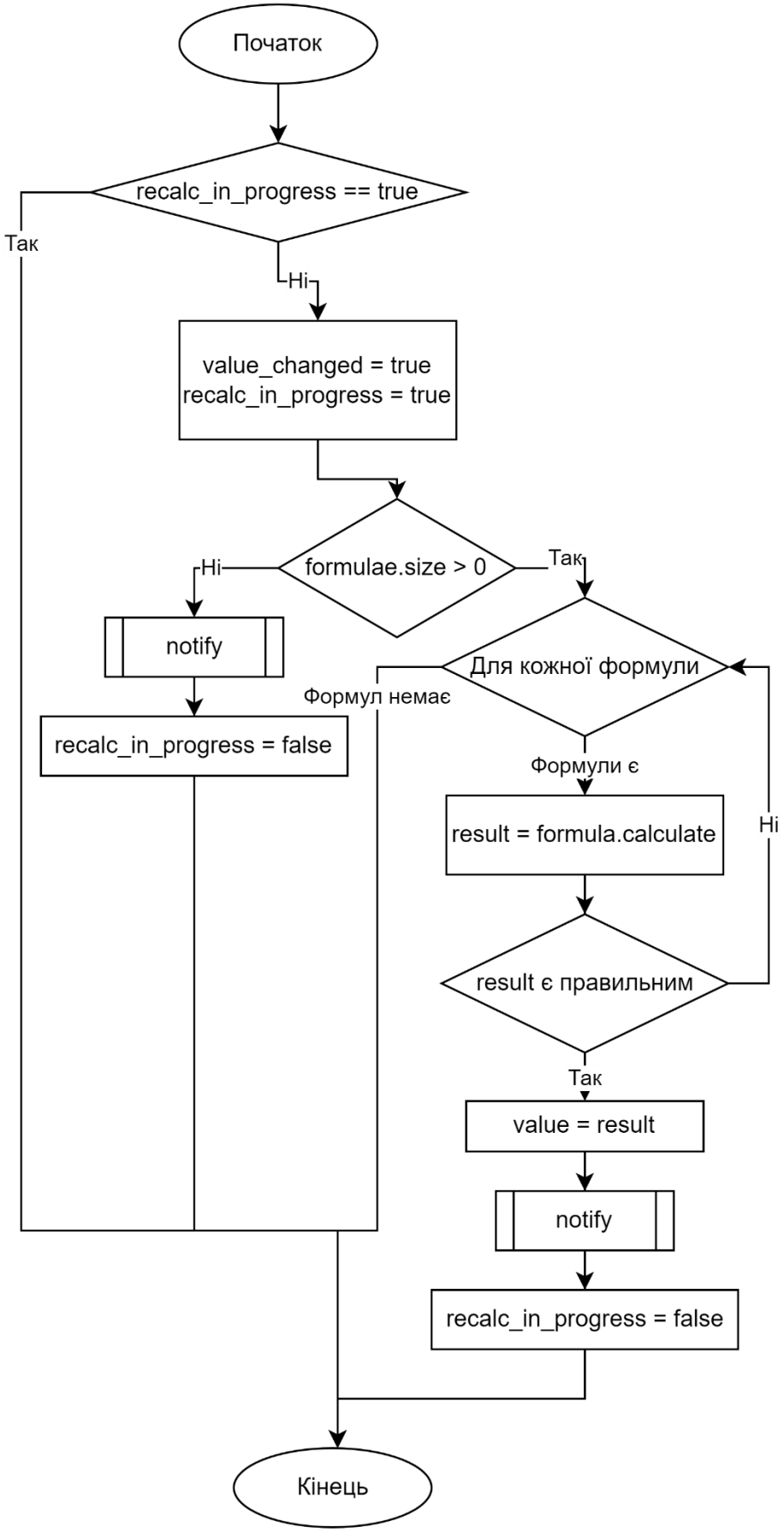

Рисунок 4.1 – Блок-схема алгоритму перерахунку параметру симуляції

Також, на блок-схемі вище можна побачити, що коли у значення немає формул з допомогою яких його можна було б порахувати – воно буде вважатись перерахованим.

Коли перерахунок значення вважається закінченим – перерахований параметр сповіщає залежні від нього параметри про зміну свого значення, щоб вони також перерахували свої. Алгоритм виконання цих сповіщень можна побачити на рис 4.2.

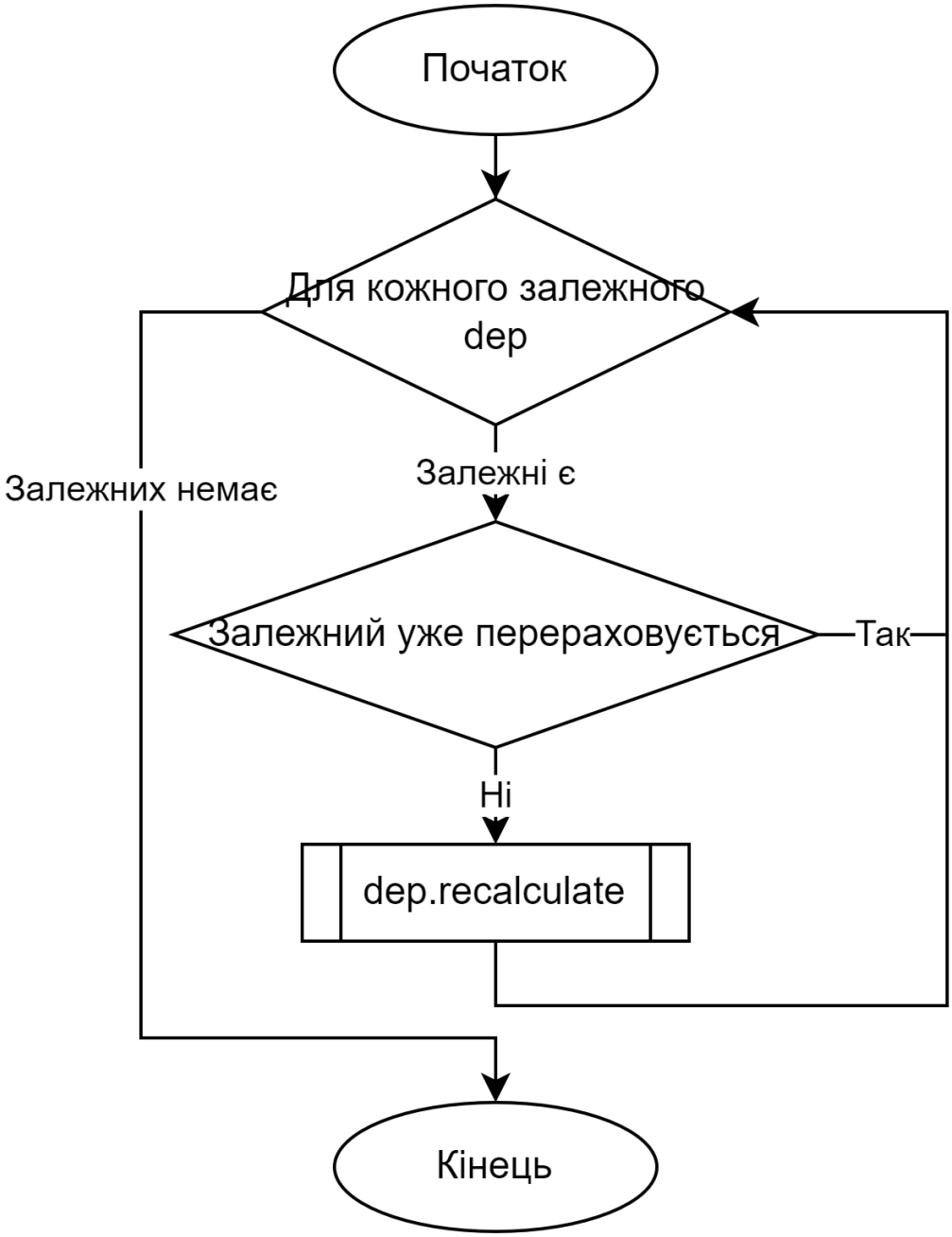

Рисунок 4.2 – Блок-схема сповіщень залежних параметрів

4.3 Інтеграція симуляції в графічний інтерфейс

Перш за все потрібно звернути увагу на те, що було вирішено прибрати сутність Simulation. Оскільки при роботі із параметрами симуляції відбувається взаємодія із ними безпосередньо, присутність посередника тільки ускладнює процес. Замість нього, в класі графічного інтерфейсу тепер містяться самі параметри, а не об’єкт симуляції.

Для інтеграції симуляції в графічний інтерфейс, потрібно, для початку, описати саму симуляцію. Для цього створюються об’єкти параметрів симуляції, описуються їх формули та залежності один від одного. В кінці вони прив’язуються до об’єктів типу GUIParameter для їх відображення в інтерфейсі.

Формули, використані для реалізації експерименту даної розробки, це формули 1.1, 1.2, та 1.3, згадані в розділі 1.2.

4.4 Зміни діаграми класів

Як було описано вище, в процесі розробки діаграма класів зазнала досить значних змін.

Було переглянуто необхідність у використанні деяких класів, необхідність в їх відокремленості від інших класів. Так, наприклад, було об’єднано класи SimParameter та ParamDependents.

Також, під час розробки виникла необхідність підлаштування під архітектуру програмного каркасу графічного інтерфейсу. Через це виникла необхідність у відділенні класу GUIParameter із класу GUIDataUnit, наслідування класу GUIDataUnit для взаємодії із різноманітними елементами інтерфейсу. Також, виникла необхідність в додаванні нових полів у ці класи, для забезпечення можливості їх прямої взаємодії із елементами графічного інтерфейсу.

Оновлену діаграму класів можна побачити на рис. 4.3.

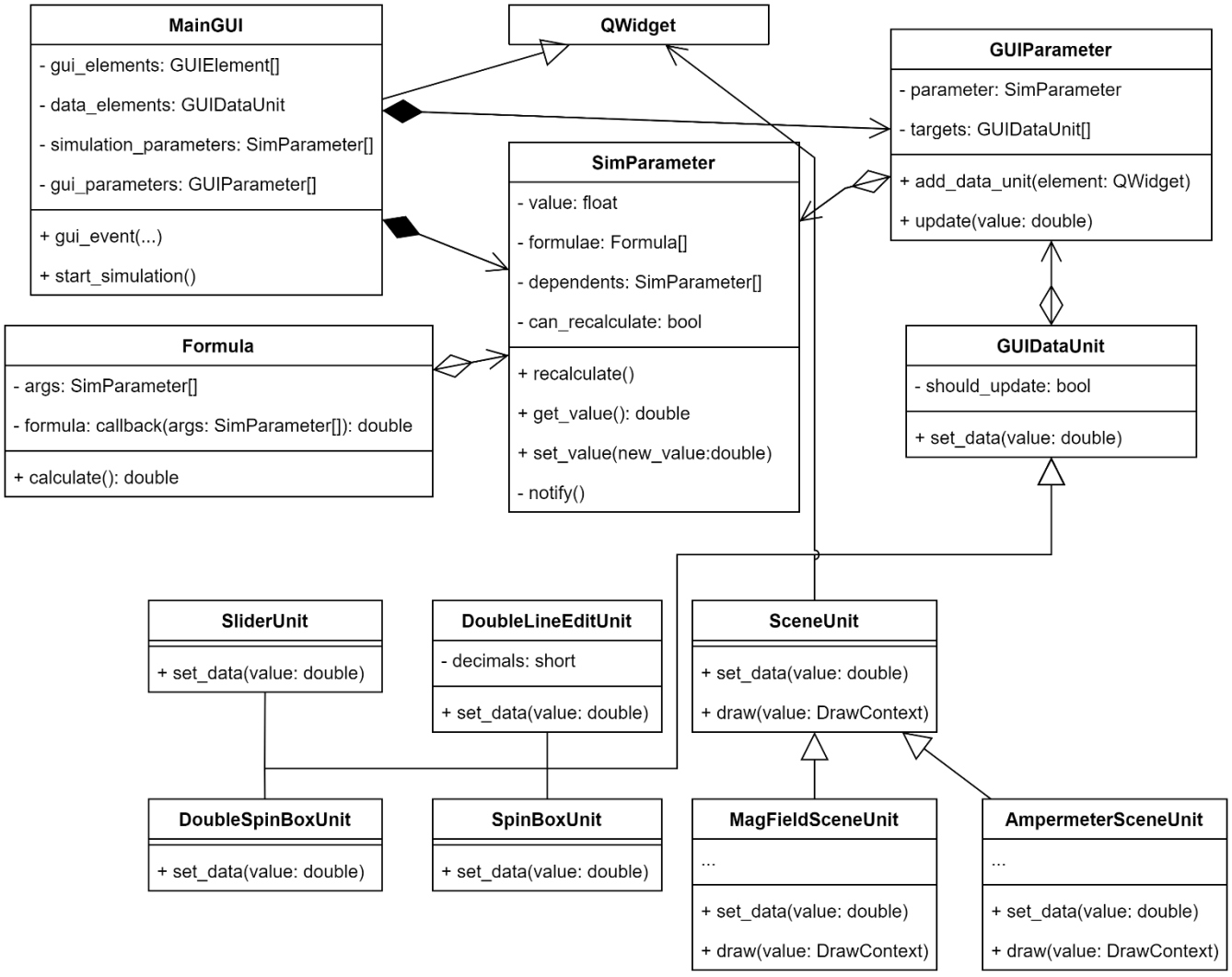

Рисунок 4.3 – Оновлена діаграма класів

Висновки до розділу 4

Було виконано розробку програмного продукту. В результаті було отримано програмний продукт, який відповідає заданим специфікаціям.

В процесі розробки було досить сильно змінено статичну структуру програми: видалено деякі сутності, об’єднано з іншими сутностями, додано нові.

Розроблена система проведення симуляції основується на залежностях параметрів одне від одного та не є прямо розрахована на деякий конкретний експеримент. Це дозволить, за необхідності, розширити, або повторно використати її для іншого експерименту в майбутньому. Завдяки своїй модульності та незалежності від інтерфейсу користувача, логіку симуляції можна виділити окремо та за необхідності поширювати для інших проектів у вигляді бібліотеки.

Розроблений графічний інтерфейс є швидким та інтерактивним. Розроблені абстракції для керування значеннями елементів інтерфейсу дозволяють без додаткових зусиль під’єднувати параметри симуляції до елементів інтерфейсу.

5 ТЕСТУВАННЯ

5.1 Модульне тестування

Для перевірки працезнатності внутрішніх модулів програми, було проведено модульне тестування. Програмним каркасом для модульного тестування став qTest, який є частиною інфраструктури згаданого раніше каркасу QT.

Структурно програму можна поділити на дві складові: графічний інтерфейс, та логіка симуляції.

Розроблені модульні тести для модулів графічного інтерфейсу можна побачити на таблиці 5.1.

Таблиця 5.1 – Модульні тести модулів графічного інтерфейсу

№	Назва тесту	Вхідні дані	Очікуваний результат
1	2	3	4
1	Працездатність класу SliderUnit	data = 10	slider.value() = 10
2	Працездатність класу SpinBoxUnit	data = 10	spin_box.value() = 10
3	Працездатність класу DoubleSpinBoxUnit	data = 10	double_spin_box.value() = 10
4	Працездатність класу DoubleLineEditUnit	data = 10	line_edit.text() = “10”
5	Працездатність можливості задання кількості чисел після коми класу DoubleLineEditUnit	data = 10.123456 decimals = -1	line_edit.text() = “10.1235”
		data = 10.123456 decimals = 0	line_edit.text() = “10”
		data = 10.123456 decimals = 3 (> 0)	line_edit.text() = “10.123”

Продовження таблиці 5.1

1	2	3	4
6	Оновлення об’єктів класів GUIDataUnit об’єктами класу GUIParameter, при зміні значення параметру симуляції	Об’єкт класу GUIParameter з даними: value = 5, для якого викликається метод update(double), який в свою чергу викликає методи update(double) об’єктів класів GUIDataUnit, для зміни внутрішніх значень об’єктів класів QWidget	slider.value() = 10 spin_box.value() = 10 double_spin_box.value() = 10 line_edit.text() = “10”
7	Оновлення об’єктів класів GUIDataUnit об’єктами класу GUIParameter, при зміні значення іншого параметру симуляції	Об’єкт класу GUIParameter з даними: value = 10, для якого викликається метод refresh(), який в свою чергу викликає методи update(double) об’єктів класів GUIDataUnit, для зміни внутрішніх значень об’єктів класів QWidget	slider.value() = 5

Розроблені модульні тести для модулів логіки симуляції можна побачити на таблиці 5.2.

Таблиця 5.2 – Модульні тести модулів логіки симуляції

№	Назва тесту	Вхідні дані	Очікуваний результат
1	2	3	4

Продовження таблиці 5.2

1	2	3	4
1	Перевірка працездатності класу Formula	Параметр із формулою, що рахує суму всіх використаних параметрів; параметри, що будуть використовуватись у формулі, зі значеннями -30, 20,5, 10.	Результат виконання формули, присвоєний параметру, який нею володіє – 0,5.
2	Перевірка працездатності перерахунку параметрів, від яких залежить даний	Параметр із формулою, яка використовує для обрахунку інші параметри; Параметр, значення якого копіює значення першого параметру.	При обрахунку параметру із формулою копіювання, змінюється значення параметру, який він копіює – воно буде сумою значень двох інших параметрів.
3	Перевірка працездатності перерахунку параметрів, які залежать від даного	Параметр із формулою, яка копіює значення іншого параметру; другий параметр, який використовується у формулі першого	При перерахунку значення другого параметру, перераховується значення і першого

Продовження таблиці 5.2

1	2	3	4
4	Перевірка стійкості до циклічних залежностях від параметрів, від яких залежить даний	Набір із декількох параметрів, які використовують одне одного у своїх формулах. При перерахунку одного параметру – перераховується значення параметру із формули, і так по кругу.	Перерахунок не безкінечний, в результаті повертається недійсне значення
5	Перевірка стійкості до циклічних залежностях від параметрів, які залежать від даного	Набір із декількох параметрів, які вказують одне одного як залежні від них. При перерахунку одного параметру – обраховується залежний від нього параметр, і так по кругу.	Перерахунок не безкінечний, в результаті повертається недійсне значення

Результати проведення модульного тестування графічного інтерфейсу можна побачити на рис. 5.1.

Рисунок 5.1 – Результати модульного тестування графічного інтерфейсу

Результати проведення модульного тестування логіки симуляції можна побачити на рис. 5.2.

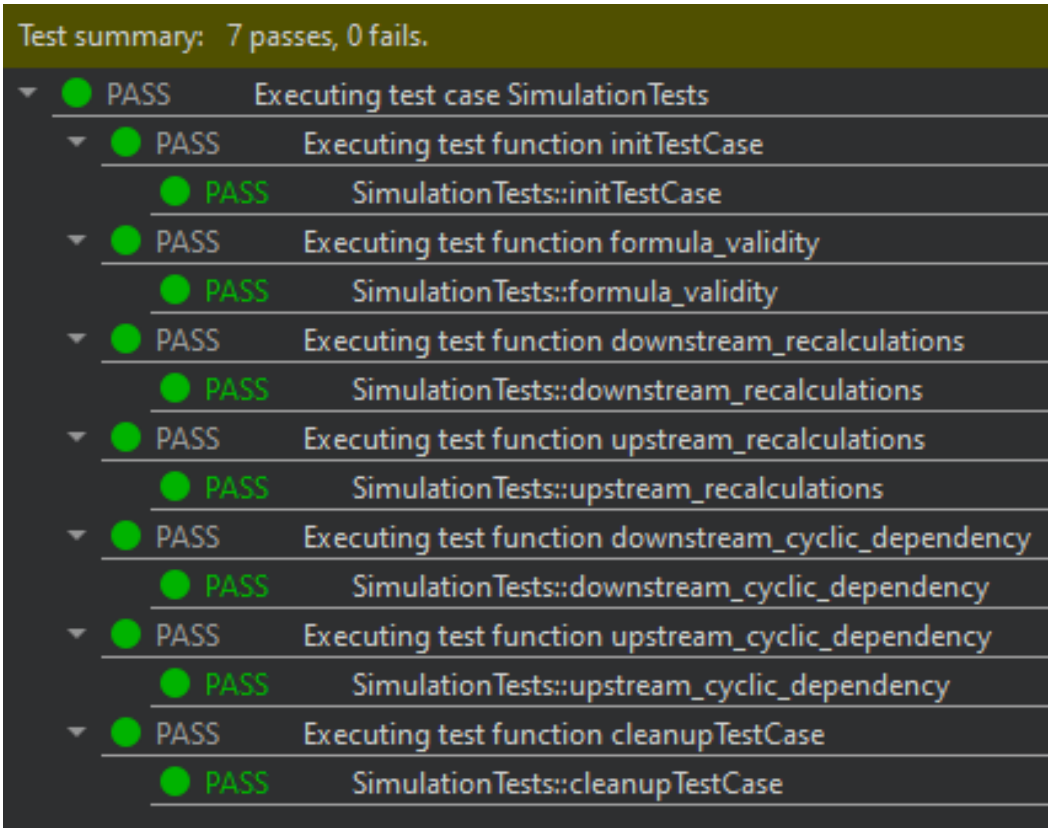

Рисунок 5.2 – Результати модульного тестування логіки симуляції

5.2 Системне тестування

Загальну працездатність програми та відповідність функціональним вимогам виконано з допомогою системного тестування. Розроблені тести можна побачити в таблиці 5.3.

Таблиця 5.3 – Системні тести

№	Назва тесту	Кроки проведення	Очікуваний результат
1	2	3	4

Продовження таблиці 5.3

1	2	3	4
1	Синхронізація елементів інтерфейсу, що відображають один і той же параметр	Зміна значення сили струму	Значення сили струму синхронізується між всіма елементами інтерфейсу, що його відображають: поле для введення з числовим значенням, повзунок, який представляє шкалу, графічне зображення амперметра, на якому відбувається поворот стрілки.
		Зміна результату куту повороту магнітної стрілки, шляхом зміни значень вхідних параметрів	Значення повороту магнітної стрілки синхронізується між всіма елементами інтерфейсу, що його відображають: текстове поле, графічне зображення магнітної стрілки, яке повертається на вказаний кут

Продовження таблиці 5.3

1	2	3	4
2	Перевірка правильності обрахунку результатів симуляції	Нехай сила струму дорівнює 20А, радіус витка дорівнює 7см, кількість витків дорівнює 3	Сила індукції лорівнює $1,79 * 10^{-6}$ Тл; Кут повороту стрілки дорівнює 19,75 градусів; Сила індукції одного витка дорівнює $5,98 * 10^{-8}$.
3	Перевірка правильності графічного відображення параметрів симуляції	Нехай сила струму дорівнює 20А, радіус витка дорівнює 7см	Стрілка амперметра повернута на 10% від свого максимального кута повороту; Магнітна стрілка вказує на значення 19,75 (~20) градусів на транспортирі, який теж зображений на малюнку.

Результати проведення системного тестування можна побачити в таблиці 5.4.

Таблиця 5.4 – Результати системного тестування

№	Назва тесту	Результат
1	Синхронізація елементів інтерфейсу, що відображають один і той же параметр	Тест пройдено (див. рис. 5.3)
2	Перевірка правильності обрахунку результатів симуляції	Тест пройдено (див. рис. 5.3)
3	Перевірка правильності графічного відображення параметрів симуляції	Тест пройдено (див. рис. 5.3)

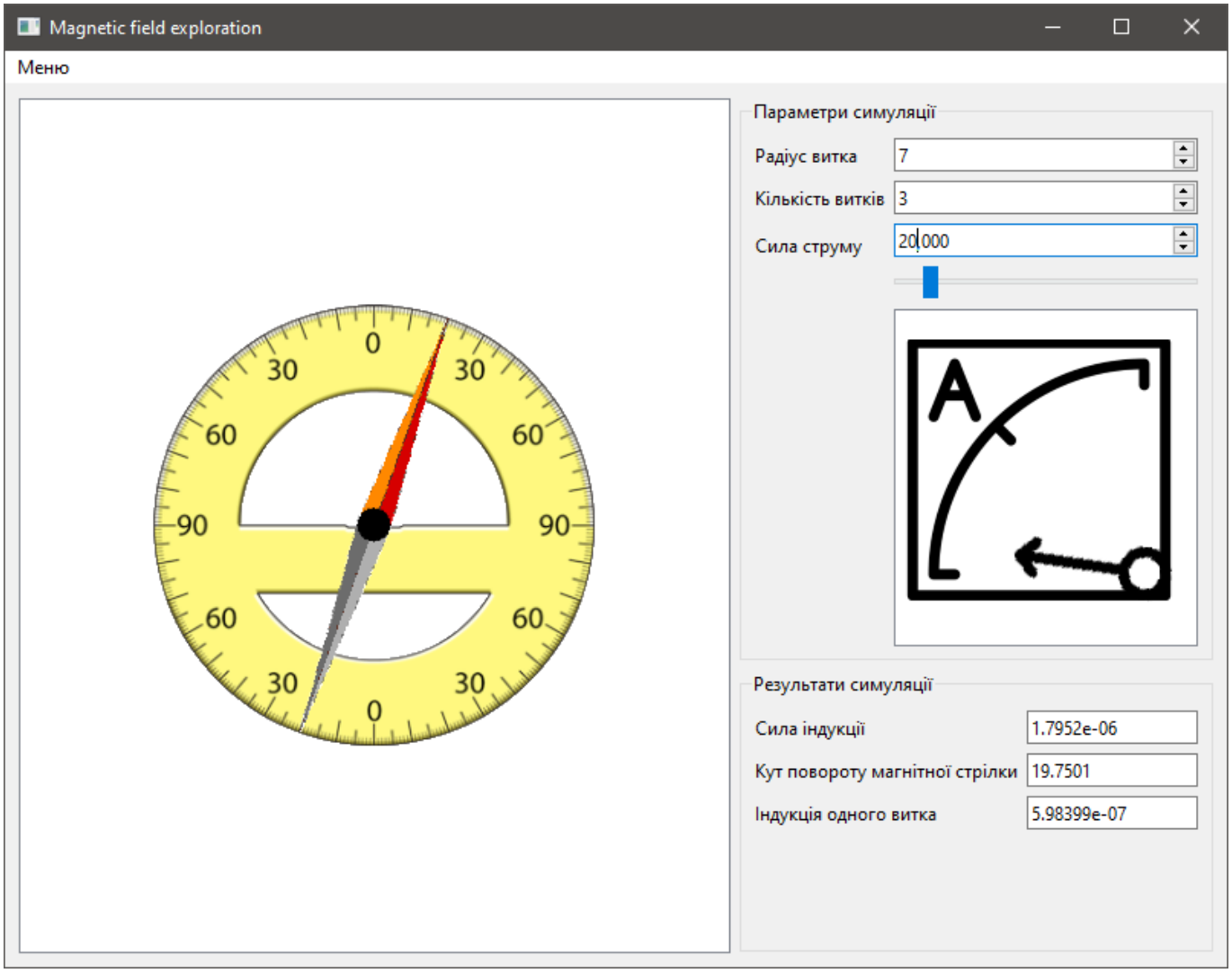

Рисунок 5.3 – Результат системного тестування

Висновки до розділу 5

Було проведено тестування програми. Судячи із результатів, можна зробити висновок, що програма працює коректно та відповідає заданим вимогам.

Усі графічні поля, які відображають одне і те ж значення, синхронізуються між собою, що забезпечує інтерактивність інтерфейсу. Графічні відображення параметрів симуляції зображені коректно.

Логіка симуляції працює коректно:

- задані формули обчислюються правильно;
- значення параметру перераховується тільки при необхідності;
- при перерахунку значення параметру, перераховуються значення всіх залежних від нього параметрів, і всіх параметрів, від яких залежить він.

ВИСНОВКИ

Результатом роботи став розроблений програмний продукт. Судячи із результатів розробки та тестування, програмний продукт відповідає заданим експлуатаційним та функціональним вимогам.

В процесі проєктування та розробки було використано шаблони проєктування наглядач, посередник. Засобами мови програмування було використано механізми наслідування, поліморфізму, зворотних викликів. Їх використання дозволило розробити гнучку, модульну архітектуру програми, яка дозволяє легко керувати елементами інтерфейсу програми, додавати нові елементи інтерфейсу і підтримку для них.

При реалізації розрахунків симуляції, головка увага приділялась модульності системи. Симуляція представляє собою граф залежностей між параметрами симуляції, та способи їх обчислення. Через це, при розробці було зустрінуто та розв’язано проблеми циклічних залежностей параметрів симуляції одне від одного.

В результаті було отримано систему представлення симуляції, яка дозволяє вручну задавати залежності між параметрами симуляції, тобто – будувати будь-які симуляції, і не тільки на тему, описану в завданні розробки. Такий підхід універсальний – тобто дозволить полегшити майбутні розробки симуляцій будь-яких систем, при його повторному використанні. Також, використання механізму зворотних викликів для представлення формул обчислення параметрів полегшує створення параметру симуляції, оскільки не вимагає перевантаження методів обчислення. Також, це відкриває можливість до автоматизації створення параметрів симуляції, що, потенційно, могло б дозволити коистувачу самому створювати власні симуляції із власними параметрами.

В підсумку можна дійти до висновку, що розробка пройшла успішно, та принесла бажаний результат.

ВИКОРИСТАНА ЛІТЕРАТУРА

1 В. Чабан. Електромагнетні кола: навч. посіб. для електротехн. фахів / В. Чабан. — 7-е вид., доповн. — Львів, Україна 2013;

2 Refactoring.Guru. Observer / Refactoring.Guru – Access mode: URL: https://refactoring.guru/design-patterns/observer. – Date of access 28.05.2022;

3 Vanmechelen K. Interfacing C++ member functions with C libraries / Kurt Vanmechelen, Jan Broeckhove – Antwerp, Belgium 2004

4 Mastering Cmake – Access Mode: URL: https://cmake.org/cmake/help/book/mastering-cmake – Date of access: 28.05.2022;

5 Getting Started with Qt – Access Mode: URL: https://doc.qt.io/qt-6/gettingstarted.html – Date of access: 28.05.2022;

6 Unified Modeling Language 2.5.1 / Object Management Group 2017.

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
18.02.22

ДОСЛІДЖЕННЯ МАГНІТНОГО ПОЛЯ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01249-01-ЛЗ

Представники
підприємства-розробника

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
18.02.22

Керівник розробки
Олена КУРОП'ЯТНИК
18.02.22

Виконавець
Андрій СЕРБЕНЮК
18.02.22

Нормоконтролер
Олена КУРОП'ЯТНИК
18.02.22

2022

ЗАТВЕРДЖЕНО
44165850.01249-01

ДОСЛІДЖЕННЯ МАГНІТНОГО ПОЛЯ

Технічне завдання

44165850.01249-01

Листів 11

2022

ЗМІСТ

ВСТУП

Програмний продукт «Дослідження магнітного поля» призначений для проведення симуляцій фізичних явищ.

Причина виникнення необхідності розробки програмного забезпечення в тому, що в сучасних реаліях, у викладачів не завжди є можливість проводити лабораторні заняття із студентами очно в лабораторії. Тому, як додатковий варіант проведення лабораторних робіт, це можна робити дистанційно з допомогою програмного забезпечення, яке б наглядно демонструвало фізичні процеси, які є темою лабораторних.

Області, в яких передбачається застосовування програмного продукту – навчальні заклади із дистанційним навчанням, домашнє навчання.

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ Українського державного університету науки і технологій № 77 ст від 08.12.2021.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – симуляції експерименту визначення індукції магнітного поля в центрі колового витка зі струмом. Для симуляції задаються власні параметри. Програма повинна симулювати фізику реального світу: під час експерименту, при зміні радіусу витка, кількості витків, або сили струму – кут повороту магнітної стрілки в центрі витка змінюється.

Експлуатаційне призначення – можливість проведення експериментів під час дистанційних занять та/або за відсутності можливості їх проведення в фізичній лабораторії.

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Програма повинна дозволяти проводити симуляції експерименту визначення індукції магнітного поля в центрі колового витка зі струмом.

Програма повинна симулювати фізику реального світу: під час експерименту, при зміні радіусу витка, кількості витків, або сили струму – кут повороту магнітної стрілки в центрі витка змінюється.

Програма повинна виконувати наступні функції:

– введення вхідних даних користувачем;
– здійснення фізичної симуляції (експеримент із вимірювання магнітної індукції колового витка зі струмом);
– виведення результатів симуляції користувачу.

Вхідні дані:

– радіус витка: ціле число від 5 до 25, крок 1. Одиниця вимірювання – сантиметри (см);

– кількість витків: ціле число від 1 до 30;

– сила струму: дійсне число в діапазоні від 0 до 2. Одиниця вимірювання – ампери (А).

Вхідні дані надаються користувачем шляхом введення їх з клавіатури у спеціально відведені поля на графічному інтерфейсі, або шляхом взаємодії із елементами графічного інтерфейсу.

Вихідні дані:

– сила індукції магнітного поля, твореного витками. Одиниця вимірювання – Тесла (Тл).

– кут повороту магнітної стрілки всередині колового витка. Одиниця вимірювання – градуси.

Виведення результатів роботи програми відбувається у вигляді їх демонстрації в інтерфейсі користувача.

3.2 Вимоги до надійності

Для забезпечення надійного функціонування необхідно забезпечити наявність архівного коду тексту програми на зовнішньому носії.

3.3 Умови експлуатації

Для забезпечення надійної роботи програмного продукту користувачу необхідно дотримуватись таких умов:

– програмний засіб повинен використовуватись у приміщеннях, призначених для роботи з ЕОМ з відповідними кліматичними умовами;

– працювати з програмним засобом може людина, що має навички роботи з ПК та ознайомлена з посібником користувача.

3.4 Вимоги до складу і параметрів технічних засобів

Програмний продукт розрахований для функціонування на пристроях, що мають такі мінімальні характеристики:

– процесор: Intel Core 2 Duo;
– місце на диску: 256 Mb;
– об’єм оперативної пам’яті: 2 Gb;
– маніпулятори: клавіатура, миша;
– інше устаткування: монітор.

3.5 Вимоги до інформаційної і програмної сумісності

Для роботи програмного продукту на ЕОМ має бути встановлене наступне програмне забезпечення:

– операційна система: Windows XP/Vista/8/8.1/10/11;
– програмне середовище: QT Creator.

3.6 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію, повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). На упаковці повинна бути присутня повна назва продукту, номер версії, мінімальні системні вимоги. На зворотній стороні упаковки вказується розробник та його юридична адреса.

Макет маркування можна побачити на рис. 4.1.

Програмне забезпечення «Симуляція магнітного поля» Мінімальні системні вимоги: – OS: Windows XP/Vista/7/8/8.1/10/11; – RAM: 2 Gb; – Пам’ять на диску: 256 Mb.	Розробник: Сербенюк А. В. Кафедра КІТ УДУНТ м. Дніпро, вул. Лазаряна, 2 2022

Рисунок 4.1 – Макет маркування

3.7 Вимоги до транспортування і зберігання

Програмне забезпечення міститься на фізичному носії даних, тому при транспортуванні повинен мати відповідну упаковку, уникати механічних ушкоджень.

Також програмне забезпечення може поширюватись мережею, тому для його зберігання та поширення доречно скористатись хмарними сервісами.

4 ЕКОНОМІЧНІ ПОКАЗНИКИ

Розрахунки економічних показників для цієї розробки не ведуться, оскільки вона не є комерціійною.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

– специфікація;
– опис програми;
– текст програми.

Вся документація до програмного забезпечення повинна задовольняти вимоги державного стандарту до оформлення програмної документації ГОСТ 19.101-77 [1].

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Стадії та етапи розробки програми приведені в табл. 6.1.

Таблиця 6.1 Стадії та етапи розробки

Стадія	Зміст робіт	Термін виконання
Технічне завдання	Постановка завдання, збір інформації, викладення та обґрунтування критеріїв розробки; попередній вибір методів вирішення задач; визначення вимог до технічних засобів; узгодження та затвердження технічного завдання.	31.01.2022 – 18.02.2022
Робочий проект	Проєктування програми та написання програмного коду	19.02.2022 – 01.05.2022
	Тестування програми	02.05.2022 – 24.05.2022
	Розробка, узгодження, та затвердження програмної документації	25.05.2022 – 10.06.2022

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙОМУ

Контроль за виконанням роботи здійснює керівник розробки Куроп’ятник О. С.

Прийом здійснюється комісією у складі, визначеному університетом.

8 БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
10.06.22

ДОСЛІДЖЕННЯ МАГНІТНОГО ПОЛЯ

Специфікації
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01249-01-ЛЗ

Представники
підприємства-розробника

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
10.06.22

Керівник розробки
Олена КУРОП'ЯТНИК
10.06.22

Виконавець
Андрій СЕРБЕНЮК
10.06.22

Нормоконтролер
Олена КУРОП'ЯТНИК
10.06.22

2022

ЗАТВЕРДЖЕНО
44165850.01249-01

ДОСЛІДЖЕННЯ МАГНІТНОГО ПОЛЯ

Специфікації

44165850.01249-01

Листів 2

2022

Позначення	Найменування	Примітки
	Документація	
44165850.01249-01-ЛЗ	Лист затвердження	
44165850.01249-01 12 01-ЛЗ	Лист затвердження	
44165850.01249-01 12 01	Текст програми	
44165850.01249-01 13 01-ЛЗ	Лист затвердження	
44165850.01249-01 13 01	Опис програми	

ДОДАТОК В

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
18.02.22

ДОСЛІДЖЕННЯ МАГНІТНОГО ПОЛЯ

Опис програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01249-01 13 01 ЛЗ

Представники
підприємства-розробника

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
18.02.22

Керівник розробки
Олена КУРОП'ЯТНИК
18.02.22

Виконавець
Андрій СЕРБЕНЮК
18.02.22

Нормоконтролер
Олена КУРОП'ЯТНИК
18.02.22

2022

ЗАТВЕРДЖЕНО
44165850.01249-01 13 01

ДОСЛІДЖЕННЯ МАГНІТНОГО ПОЛЯ

Опис програми

44165850.01249-01 13 01

Листів 14

2022

АНОТАЦІЯ

Документ 44165850.01249-01 13 01 «Дослідження магнітного поля. Опис програми» входить до складу програмної документації на програму, що реалізовує фізичні симуляції.

У даному документі представлений опис функціоналу програми. Програма написана на мові C++.

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

Програмний продукт «Дослідження магнітного поля» призначений для проведення симуляцій фізичних явищ.

Програма написана на мові програмування C++. Середовищем розробки є QT Creator.

Завдяки вибору саме цієї мови програмування, розробленій програмі для функціонування не потрібне додаткове програмне забезпечення, окрім операційної системи. Програма поширюється у вигляді попередньо збудованих пакетів для конкретних платформ.

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Функціональне призначення – симуляції експерименту визначення індукції магнітного поля в центрі колового витка зі струмом. Для симуляції задаються власні параметри. Програма повинна симулювати фізику реального світу: під час експерименту, при зміні радіусу витка, кількості витків, або сили струму – кут повороту магнітної стрілки в центрі витка змінюється.

Функціональним обмеженням програми є те, що для симуляції не можна задати значення результату симуляції, щоб замість нього обрахувати з якими вхідними даними він можливий.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1 Алгоритм програми

Алгоритм програми представлено на рис. 1.

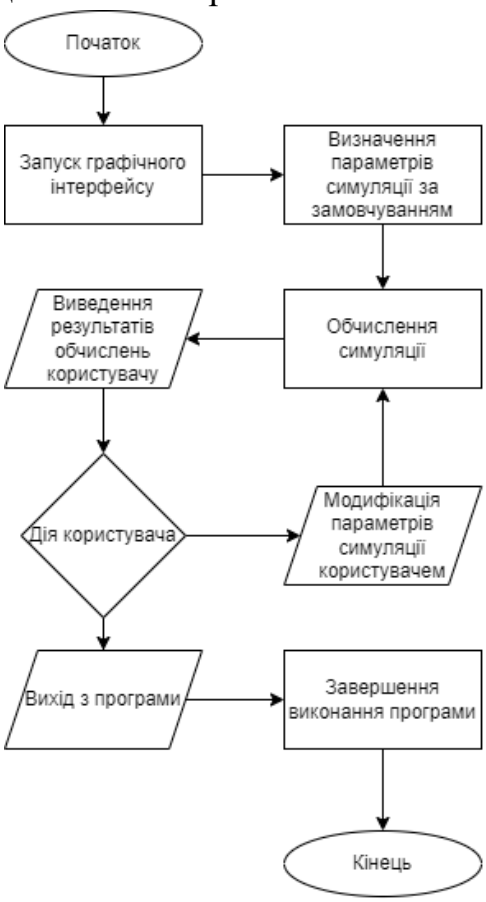

Рис. 1. Алгоритм виконання програми

3.2 Використані методи

При плануванні структури програми був використаний об’єтно-орієнтований підхід. Він має наступні переваги:

– опис системи у вигляді об'єктів що відповідає змістовному змістом предметної області;

– сутності реального світу, як правило, володіють поведінкою, що в об'єктно-орієнтованому проектуванні відбивається за допомогою визначення методів класу. У структурному підході дані (атрибути) і алгоритми (методи) існують окремо один від одного;

– об'єднання атрибутів і методів в об'єкті (класі), а також інкапсуляція призводить до більшої внутрішньої і меншою зовнішньої зв'язності між компонентами системи. Це у свою чергу полегшує вирішення проблем адаптації системи до зміни існуючих або появи нових вимог, супроводу системи на різних стадіях життєвого циклу, повторного використання компонентів;

– дозволяє легше організувати паралельні обчислення, так як кожен об'єкт володіє власними значеннями характеристик (атрибутів) і поведінкою, за рахунок чого можна домогтися його автономної роботи.

3.3 Структура програми

Структурно програму можна поділити на наступні модулі:

– модуль user_interaction – відповідає за взаємодію з користувачем, виведення йому повідомлень, отримання і обробку даних від нього;
– модуль simulation_logic – відповідає за обчислення симуляції на основі деяких вхідних даних;
– модуль interface_display – відповідає за відображення результатів симуляції на графічному інтерфейсі.

3.4 Зв’язки програми з іншими програмами

Окрім операційної системи пристрою, програма не має зв’язків з іншими програмами.

4 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Програмний продукт розрахований для функціонування на пристроях, що мають такі мінімальні характеристики:

– процесор: Intel Core 2 Duo;

– місце на диску: 256 Mb;

– об’єм оперативної пам’яті: 2 Gb;

– операційна система: Windows XP/Vista/8/8.1/10/11;

5 ВИКЛИК І ЗАВАНТАЖЕННЯ

Передбачається поширення програми у вигляді попередньо збудованих пакетів для конкретних платформ. Для запуску додатку потрібно отримати пакет з версією програми для платформи користувача на сам пристрій. Самі пакети можуть поширюватись як на фізичних носіях даних, так і через Інтернет.

Для запуску програми, потрібно запустити відповідний файл із розширенням .exe (magnetic_field_exploration.exe).

6 ВХІДНІ ДАНІ

Вхідними даними, потрібними для симуляції, є:

– радіус витка: ціле число від 5 до 100, крок 1. Одиниця вимірювання – сантиметри (см);

– кількість витків: ціле число від 1 до 30;

– сила струму: дійсне число в діапазоні від 0 до 200. Одиниця вимірювання – ампери (А).

Дані надаються користувачем шляхом заповнення полів у графічному інтерфейсі програми.

7 ВИХІДНІ ДАНІ

Вихідними даними роботи програми є:

– сила індукції магнітного поля, твореного витками – дійсне число. Одиниця вимірювання – Тесла (Тл).

– кут повороту магнітної стрілки всередині колового витка – дійсне число в діапазоні від 0 до 360. Одиниця вимірювання – градуси.

Виведення результатів роботи програми відбувається у вигляді їх демонстрації в інтерфейсі користувача.

8 ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА

Після завантаження програми (launch.exe) на екрані з’являється інтерфейс користувача (рис. 8.1).

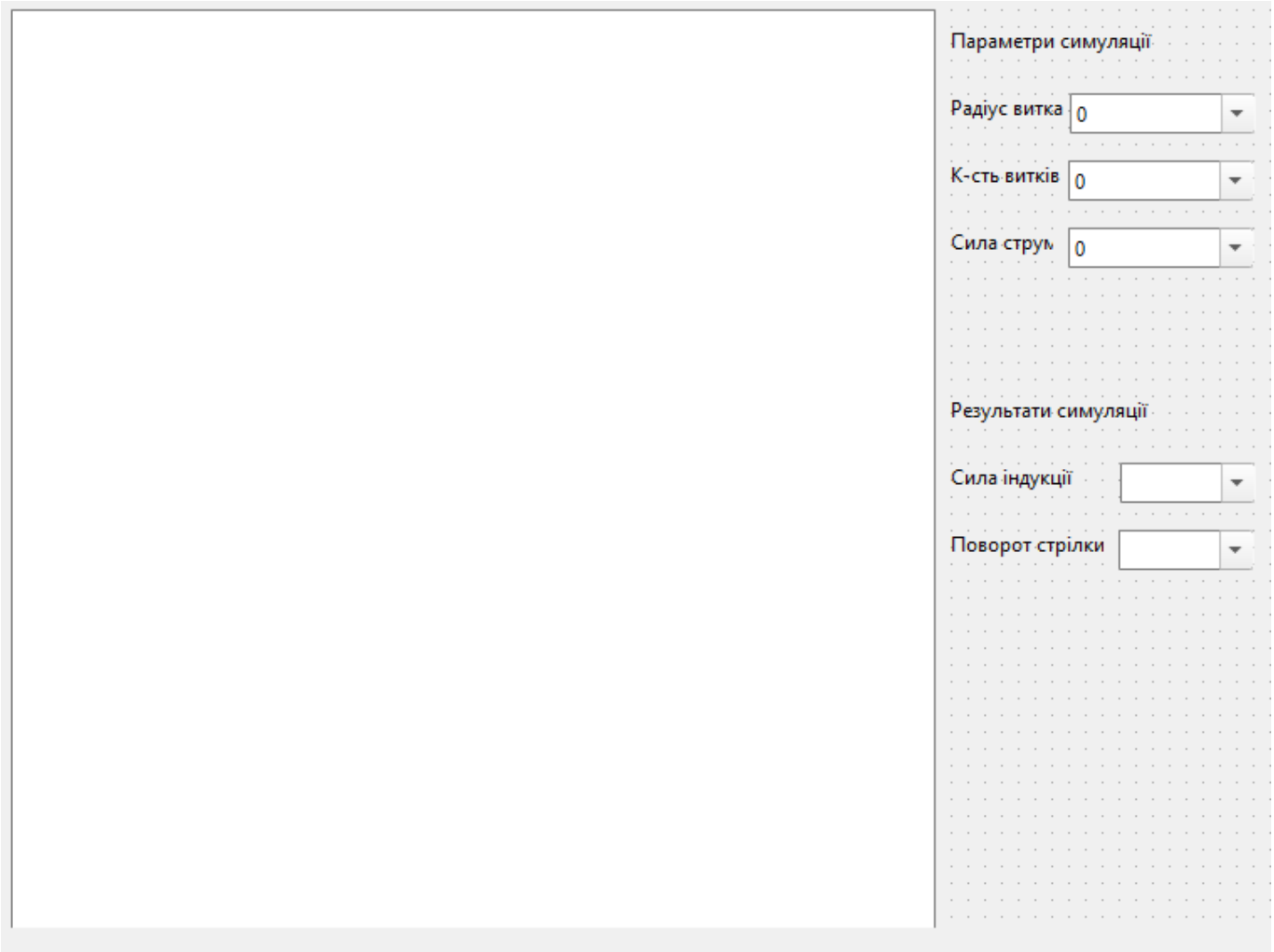

Рис. 8.1 Прототип графічного інтерфейсу користувача

При введенні параметру симуляції вона одразу запускається в реальному часі, тому механізму її ручного запуску не передбачено.

Для завершення програми потрібно натиснути на кнопку закриття у верхній правій частині вікна (рис. 8.1).

9 ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

Робота із програмою відбувається наступним чином:

1) запуск програми;
2) введення параметрів симуляції;
3) отримання результатів симуляції;
4) можливий перехід на крок 2;
5) завершення роботи програми.

10 ПОВІДОМЛЕННЯ

Можливі повідомлення програми користувачу наведено в табл. 10.1.

Таблиця 10.1 Повідомлення користувачу

Текст повідомлення	Опис	Рекомендовані дії
Введені дані некоректні	Користувач ввів дані, які неможливо використовувати для симуляції	Ввести дані коректного формату, відповідно до фізичної величини
На основі введених даних провести симуляцію фізично неможливо	Користувач ввів дані, використання яких неможливе з фізичної точки зору	Ввести коректні дані

ДОДАТОК Г

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
18.02.22

ДОСЛІДЖЕННЯ МАГНІТНОГО ПОЛЯ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01249-01 12 01 ЛЗ

Представники
підприємства-розробника

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
18.02.22

Керівник розробки
Олена КУРОП'ЯТНИК
18.02.22

Виконавець
Андрій СЕРБЕНЮК
18.02.22

Нормоконтролер
Олена КУРОП'ЯТНИК
18.02.22

2022

ЗАТВЕРДЖЕНО
44165850.01249-01 12 01

ДОСЛІДЖЕННЯ МАГНІТНОГО ПОЛЯ

Текст програми

44165850.01249-01 12 01

Листів 6

2022

АНОТАЦІЯ

Документ 44165850.01249-01 12 01 «Дослідження магнітного поля. Текст програми».

Об’єкт дослідження – симуляція фізичних явищ.

Мета роботи – Розробка додатку для симуляції фізичних явищ, для її подальшого використання в навчальних лабораторіях.

Метод дослідження – Практична розробка додатку з підтримкою графічного інтерфейсу.

ЗМІСТ

1 ТЕКСТ ПРОГРАМИ

1.1 Класи інтерфейсу користувача

1.1.1 main.cpp

```
#include "gui/main_gui.h"

#include <QApplication>

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

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

1.1.2 main_gui.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainGUI</class>
 <widget class="QMainWindow" name="MainGUI">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>935</width>
    <height>602</height>
   </rect>
  </property>
  <property name="minimumSize">
   <size>
    <width>800</width>
    <height>600</height>
   </size>
  </property>
  <property name="windowTitle">
   <string>Magnetic field exploration</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <property name="sizePolicy">
    <sizepolicy hsizetype="Preferred" vsizetype="Preferred">
     <horstretch>2</horstretch>
     <verstretch>0</verstretch>
    </sizepolicy>
   </property>
   <layout class="QHBoxLayout" name="horizontalLayout">
    <item>
     <layout class="QHBoxLayout" name="MainLayout" stretch="3,2">
      <item>
       <widget class="QGraphicsView" name="mag_field_view">
        <property name="verticalScrollBarPolicy">
         <enum>Qt::ScrollBarAlwaysOff</enum>
        </property>
        <property name="horizontalScrollBarPolicy">
         <enum>Qt::ScrollBarAlwaysOff</enum>
        </property>
        <property name="sizeAdjustPolicy">
         <enum>QAbstractScrollArea::AdjustToContents</enum>
        </property>
        <property name="interactive">
         <bool>false</bool>
        </property>
        <property name="resizeAnchor">
         <enum>QGraphicsView::AnchorViewCenter</enum>
        </property>
        <property name="viewportUpdateMode">
         <enum>QGraphicsView::SmartViewportUpdate</enum>
        </property>
        <property name="rubberBandSelectionMode">
         <enum>Qt::ContainsItemBoundingRect</enum>
        </property>
       </widget>
      </item>
      <item>
       <layout class="QVBoxLayout" name="ParametersLayout" stretch="2,1">
        <item>
         <widget class="QGroupBox" name="InputsGroupBox">
```

```
                <property name="title">
                 <string>Параметри симуляції</string>
                </property>
                <property name="flat">
                 <bool>false</bool>
                </property>
                <layout class="QHBoxLayout" name="horizontalLayout_2">
                 <item>
                  <layout class="QFormLayout" name="InputsLayout">
                   <item row="0" column="0">
                    <widget class="QLabel" name="coil_radius_label">
                     <property name="toolTip">
                      <string>Радіус витка установки (R)</string>
                     </property>
                     <property name="text">
                      <string>Радіус витка</string>
                     </property>
                    </widget>
                   </item>
                   <item row="0" column="1">
                    <widget class="QDoubleSpinBox" name="coil_radius_box">
                     <property name="toolTip">
                      <string>Сантиметри (см)</string>
                     </property>
                     <property name="decimals">
                      <number>0</number>
                     </property>
                     <property name="minimum">
                      <double>1.000000000000000</double>
                     </property>
                     <property name="maximum">
                      <double>100.000000000000000</double>
                     </property>
                    </widget>
                   </item>
                   <item row="1" column="0">
                    <widget class="QLabel" name="coil_number_label">
                     <property name="toolTip">
                      <string>Кількість витків установки (N)</string>
                     </property>
                     <property name="text">
                      <string>Кількість витків</string>
                     </property>
                    </widget>
                   </item>
                   <item row="1" column="1">
                    <widget class="QSpinBox" name="coil_number_box">
                     <property name="toolTip">
                      <string>Кількість</string>
                     </property>
                     <property name="minimum">
                      <number>1</number>
                     </property>
                     <property name="maximum">
                      <number>30</number>
                     </property>
                    </widget>
                   </item>
                   <item row="2" column="0">
                    <widget class="QLabel" name="current_label">
                     <property name="toolTip">
                      <string>Сила струму, що протікає через витки установки (I)</string>
                     </property>
                     <property name="text">
                      <string>Сила струму</string>
                     </property>
                    </widget>
                   </item>
                   <item row="2" column="1">
                    <layout class="QVBoxLayout" name="CurrentForceLayout">
                     <item>
                      <widget class="QDoubleSpinBox" name="current_spin_box">
                       <property name="toolTip">
                        <string>Ампери (A)</string>
                       </property>
                       <property name="decimals">
                        <number>3</number>
                       </property>
                       <property name="maximum">
                        <double>200.000000000000000</double>
                       </property>
                       <property name="singleStep">
                        <double>0.100000000000000</double>
                       </property>
                      </widget>
                     </item>
                     <item>
                      <widget class="QSlider" name="current_slider">
                       <property name="toolTip">
                        <string>Ампери (A)</string>
                       </property>
                       <property name="maximum">
                        <number>2000</number>
                       </property>
```

```
                    <property name="singleStep">
                     <number>10</number>
                    </property>
                    <property name="pageStep">
                     <number>100</number>
                    </property>
                    <property name="orientation">
                     <enum>Qt::Horizontal</enum>
                    </property>
                   </widget>
                  </item>
                  <item>
                   <widget class="QGraphicsView" name="ampermeter_view">
                    <property name="toolTip">
                     <string>Ампери (A)</string>
                    </property>
                    <property name="verticalScrollBarPolicy">
                     <enum>Qt::ScrollBarAlwaysOff</enum>
                    </property>
                    <property name="horizontalScrollBarPolicy">
                     <enum>Qt::ScrollBarAlwaysOff</enum>
                    </property>
                    <property name="sizeAdjustPolicy">
                     <enum>QAbstractScrollArea::AdjustToContents</enum>
                    </property>
                    <property name="interactive">
                     <bool>false</bool>
                    </property>
                    <property name="resizeAnchor">
                     <enum>QGraphicsView::AnchorViewCenter</enum>
                    </property>
                    <property name="viewportUpdateMode">
                     <enum>QGraphicsView::SmartViewportUpdate</enum>
                    </property>
                    <property name="rubberBandSelectionMode">
                     <enum>Qt::ContainsItemBoundingRect</enum>
                    </property>
                   </widget>
                  </item>
                 </layout>
                </item>
               </layout>
              </item>
             </layout>
            </widget>
           </item>
           <item>
            <widget class="QGroupBox" name="OutputsGroupBox">
             <property name="title">
              <string>Результати симуляції</string>
             </property>
             <property name="flat">
              <bool>false</bool>
             </property>
             <layout class="QHBoxLayout" name="horizontalLayout_3">
              <item>
               <layout class="QFormLayout" name="OutputsLayout">
                <item row="0" column="0">
                 <widget class="QLabel" name="induction_label">
                  <property name="toolTip">
                   <string>Сила магнітної індукції, створюваної витками установки (B)</string>
                  </property>
                  <property name="text">
                   <string>Сила індукції</string>
                  </property>
                 </widget>
                </item>
                <item row="1" column="0">
                 <widget class="QLabel" name="angle_label">
                  <property name="toolTip">
                   <string>Кут повороту магнітної стрілки від магнітного поля Землі</string>
                  </property>
                  <property name="text">
                   <string>Кут повороту магнітної стрілки</string>
                  </property>
                 </widget>
                </item>
                <item row="0" column="1">
                 <widget class="QLineEdit" name="induction_box">
                  <property name="toolTip">
                   <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;Тесла (Тл)&lt;/p&gt;&lt;p&gt;&lt;img src=&quot;:/ind_formula&quot;/&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
                  </property>
                  <property name="text">
                   <string>0.0</string>
                  </property>
                  <property name="readOnly">
                   <bool>true</bool>
                  </property>
```

```
                    </widget>
                   </item>
                   <item row="1" column="1">
                    <widget class="QLineEdit" name="angle_box">
                     <property name="toolTip">

<string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;Градуси&lt;/p&gt;&lt;p&gt;&lt;img src=&quot;:/angle_formula&quot;/&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
                     </property>
                     <property name="text">
                      <string>0.0</string>
                     </property>
                     <property name="readOnly">
                      <bool>true</bool>
                     </property>
                    </widget>
                   </item>
                   <item row="2" column="0">
                    <widget class="QLabel" name="coil_induction_label">
                     <property name="toolTip">
                      <string>Індукція, створювана одним витком установки (B1)</string>
                     </property>
                     <property name="text">
                      <string>Індукція одного витка</string>
                     </property>
                    </widget>
                   </item>
                   <item row="2" column="1">
                    <widget class="QLineEdit" name="coil_induction_box">
                     <property name="toolTip">

<string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;Тесла (Тл)&lt;/p&gt;&lt;p&gt;&lt;img src=&quot;:/coil_ind_formula&quot;/&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
                     </property>
                     <property name="text">
                      <string>0.0</string>
                     </property>
                     <property name="readOnly">
                      <bool>true</bool>
                     </property>
                    </widget>
                   </item>
                  </layout>
                 </item>
                </layout>
               </widget>
              </item>
             </layout>
            </item>
           </layout>
          </item>
         </layout>
        </item>
       </layout>
      </item>
     </layout>
    </widget>
    <widget class="QMenuBar" name="menubar">
     <property name="geometry">
      <rect>
       <x>0</x>
       <y>0</y>
       <width>935</width>
       <height>21</height>
      </rect>
     </property>
     <widget class="QMenu" name="menuMenu">
      <property name="title">
       <string>Меню</string>
      </property>
      <addaction name="actionExit"/>
     </widget>
     <addaction name="menuMenu"/>
    </widget>
    <action name="actionExit">
     <property name="text">
      <string>Вихід</string>
     </property>
    </action>
   </widget>
   <resources/>
   <connections/>
  </ui>
```

1.1.3 main_gui.h

```
#ifndef MAIN_GUI_H
#define MAIN_GUI_H

#include <QMainWindow>
#include "../simulation_logic/sim_parameter.h"
#include "gui_parameter.h"
#include "scene_unit.h"

QT_BEGIN_NAMESPACE
namespace Ui
{
class MainGUI;
```

```
}
QT_END_NAMESPACE

namespace sl = simulation;

class MainGUI : public QMainWindow
{
   Q_OBJECT

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

private:
   Ui::MainGUI* m_ui;
   QGraphicsScene m_current_scene;
   QGraphicsScene m_mag_field_scene;

   // Simulation parameters
   sl::SimParameter m_el_current;      // Electric current
   sl::SimParameter m_coil_number_par; // Quantity of coils
   sl::SimParameter m_coil_radius_par; // Radius of a single
coil
   sl::SimParameter    m_induction_par;          //    Resulting
magnetic induction
   sl::SimParameter m_angle_par;         // Rotation angle of
the magnetic arrow
   sl::SimParameter m_coil_induction_par;

   // Input simulation parameters
   ui::GUIParameter* m_current;        // Electric current
   ui::GUIParameter* m_coil_number;    // Quantity of coils
   ui::GUIParameter* m_coil_radius;    // Radius of a single
coil

   ui::DoubleSpinBoxUnit* m_current_box;
   ui::SliderUnit* m_current_slider;
   ui::AmpermeterSceneUnit* m_current_painter;
   ui::SpinBoxUnit* m_coil_num_box;
   ui::DoubleSpinBoxUnit* m_coil_radius_box;

   // Output simulation parameters
   ui::GUIParameter* m_induction;
   ui::GUIParameter* m_angle;
   ui::GUIParameter* m_coil_induction;

   ui::DoubleLineEditUnit* m_induction_box;
   ui::DoubleLineEditUnit* m_angle_box;
   ui::MagFieldSceneUnit* m_mag_field_painter;
   ui::DoubleLineEditUnit* m_coil_inductiom_box;

   void init_sim_params();
   void init_connections();
};

#endif // MAINGUI_H
```

1.1.4 main_gui.cpp

```
#include <limits>
#include <math.h>
#include "main_gui.h"
#include "./ui_main_gui.h"

#include "QGraphicsEllipseItem"
#include "QGraphicsRectItem"
#include "QPixmap"

#define INDUCTION_DECIMALS 20
#define ANGLE_DECIMALS 20

// Simulation constants
constexpr double PI = 3.141593;
constexpr double MU = 1;

constexpr double MU_0()
{
   return 4 * PI * std::pow<double>(10, -7);
}

constexpr double IND_EARTH()
{
   return 0.5 * pow<double>(10, -5);
}

// MainGUI methods
void MainGUI::init_sim_params()
{
   m_induction_par.add_formula(sl::Formula(
      {&m_el_current, &m_coil_radius_par},
      [](std::vector<sl::SimParameter*>&          args)          ->
sl::ParamResult
      {
         if (args.size() < 2)
         {
            return sl::ParamResult(false, 0);
         }

         return sl::ParamResult(true, MU * MU_0() *
            (args[0]->get_value().second /
            (double)(2 * args[1]->get_value().second)));
      }));
```

```
m_angle_par.add_formula(sl::Formula({&m_induction_par},
                    [](std::vector<sl::SimParameter*>&  args)  ->
sl::ParamResult
                    {
                      if (args.size() < 1)
                      {
                        return sl::ParamResult(false, 0);
                      }

                      return  sl::ParamResult(true,  atan(args[0]-
>get_value().second
                        / IND_EARTH()) * (180 / PI));
                    }));

                m_coil_induction_par.add_formula(sl::Formula(
                  {&m_induction_par, &m_coil_number_par},
                  [](std::vector<sl::SimParameter*>&  args)  ->
sl::ParamResult
                  {
                    if (args.size() < 2 or args[1]->get_value().second ==
0)
                    {
                      return sl::ParamResult(false, 0);
                    }

                    return  sl::ParamResult(true,  args[0]-
>get_value().second
                      / args[1]->get_value().second);
                  }));

                // Dependents
                m_el_current.add_dependent(&m_induction_par);
                m_el_current.add_dependent(&m_angle_par);


m_coil_number_par.add_dependent(&m_coil_induction_par);

                m_coil_radius_par.add_dependent(&m_induction_par);
                m_coil_radius_par.add_dependent(&m_angle_par);

                m_induction_par.add_dependent(&m_angle_par);

m_induction_par.add_dependent(&m_coil_induction_par);
              }

              void MainGUI::init_connections()
              {
                // Electric current
                connect(m_ui->current_spin_box,
                  SIGNAL(valueChanged(double)),
                  m_current,
                  SLOT(update(double)));
                connect(m_ui->current_slider,
                  SIGNAL(valueChanged(int)),
                  m_current,
                  SLOT(update(int)));

                // Coils quantity
                connect(m_ui->coil_number_box,
                  SIGNAL(valueChanged(int)),
                  m_coil_number,
                  SLOT(update(int)));

                // Coils radius
                connect(m_ui->coil_radius_box,
                  SIGNAL(valueChanged(double)),
                  m_coil_radius,
                  SLOT(update(double)));

                // Electromagnetic induction
                connect(m_current,
                  SIGNAL(updated()),
                  m_induction,
                  SLOT(refresh()));

                connect(m_coil_radius,
                  SIGNAL(updated()),
                  m_induction,
                  SLOT(refresh()));

                // Magnetic arrow turn angle
                connect(m_induction,
                  SIGNAL(updated()),
                  m_angle,
                  SLOT(refresh()));

                // Single coil induction
                connect(m_coil_number,
                  SIGNAL(updated()),
                  m_coil_induction,
                  SLOT(refresh()));

                connect(m_induction,
                  SIGNAL(updated()),
                  m_coil_induction,
                  SLOT(refresh()));
              }

              MainGUI::MainGUI(QWidget *parent)
                : QMainWindow(parent),
                m_ui(new  Ui::MainGUI),  m_current_scene(),
m_mag_field_scene()
              {
```

```
    m_ui->setupUi(this);

    double current_min = m_ui->current_spin_box->minimum();
    double current_max = m_ui->current_spin_box->maximum();

    double coils_n_min = m_ui->coil_number_box->minimum();
    double coils_n_max = m_ui->coil_number_box->maximum();

    double coils_r_min = m_ui->coil_radius_box->minimum();
    double coils_r_max = m_ui->coil_radius_box->maximum();

    // Simulation parameters
    m_el_current = sl::SimParameter(current_min);
    m_coil_number_par = sl::SimParameter(coils_n_min);
    m_coil_radius_par = sl::SimParameter(coils_r_min);
    m_induction_par = sl::SimParameter(0);
    m_angle_par = sl::SimParameter(0);
    m_coil_induction_par =sl::SimParameter(0);

    init_sim_params();

    // GUIParameters
    // Electric current
    double current_int_factor = (current_max - current_min) /
      static_cast<double>(m_ui->current_slider->maximum() -
      m_ui->current_slider->minimum());

    m_current = new ui::GUIParameter(m_el_current, current_min, current_max,
      current_int_factor);

    // Coils quantity
    m_coil_number = new ui::GUIParameter(m_coil_number_par, coils_n_min,
      coils_n_max);

    // Coils radius
    m_coil_radius = new ui::GUIParameter(m_coil_radius_par, coils_r_min,
      coils_r_max);

    // Electromagnetic induction
    m_induction = new ui::GUIParameter(m_induction_par,
      std::numeric_limits<double>::min(),
      std::numeric_limits<double>::max());

    // Magnetic arrow turn angle
    m_angle = new ui::GUIParameter(m_angle_par,
      std::numeric_limits<double>::min(),
      std::numeric_limits<double>::max());

    // Single coil induction
    m_coil_induction = new ui::GUIParameter(m_coil_induction_par,
      std::numeric_limits<double>::min(),
      std::numeric_limits<double>::max());

    // GUIDataUnits
    // Electric current
    m_current_box = new ui::DoubleSpinBoxUnit(m_ui->current_spin_box);
    m_current_slider = new ui::SliderUnit(m_ui->current_slider,
      current_int_factor);
    m_current_painter = new ui::AmpermeterSceneUnit(m_current_scene);
    m_current_painter->set_min_value(current_min);
    m_current_painter->set_max_value(current_max);

    m_current->add_data_unit(m_current_box);
    m_current->add_data_unit(m_current_slider);
    m_current->add_data_unit(m_current_painter);

    m_ui->ampermeter_view->setScene(&m_current_scene);

    // Coils quantity
    m_coil_num_box = new ui::SpinBoxUnit(m_ui->coil_number_box);
    m_coil_number->add_data_unit(m_coil_num_box);

    // Coils radius
    m_coil_radius_box = new ui::DoubleSpinBoxUnit(m_ui->coil_radius_box);
    m_coil_radius->add_data_unit(m_coil_radius_box);

    // Electromagnetic induction
    m_induction_box = new ui::DoubleLineEditUnit(m_ui->induction_box,
      INDUCTION_DECIMALS);

    m_induction->add_data_unit(m_induction_box);

    // Magnetic arrow turn angle
    m_angle_box = new ui::DoubleLineEditUnit(m_ui->angle_box, ANGLE_DECIMALS);
    m_angle->add_data_unit(m_angle_box);
```

```
        m_ui->mag_field_view->setScene(&m_mag_field_scene);
        m_mag_field_painter = new ui::MagFieldSceneUnit(m_mag_field_scene);
        m_angle->add_data_unit(m_mag_field_painter);

        // Single coil induction
        m_coil_inductiom_box = new ui::DoubleLineEditUnit(m_ui->coil_induction_box,
            INDUCTION_DECIMALS);
        m_coil_induction->add_data_unit(m_coil_inductiom_box);

        // Slots bindings
        init_connections();

        m_current->refresh();
    }

    MainGUI::~MainGUI()
    {
        delete m_ui;

        // GUIParameters
        delete m_current;
        delete m_coil_number;
        delete m_coil_radius;

        delete m_induction;
        delete m_angle;
        delete m_coil_induction;

        // GUIDataUnits
        delete m_current_box;
        delete m_current_slider;
        delete m_current_painter;
        delete m_coil_num_box;
        delete m_coil_radius_box;

        delete m_induction_box;
        delete m_mag_field_painter;
        delete m_angle_box;
        delete m_coil_inductiom_box;
    }
```

1.1.5 gui_parameter.h

```
    #ifndef GUI_PARAMETER_H
    #define GUI_PARAMETER_H

    #include <vector>
    #include "QObject"
    #include "../simulation_logic/sim_parameter.h"
    #include "gui_data_unit.h"

    namespace ui
    {

    namespace sl = simulation;

    // A wrapper for SimParameter objects, that functions as a mediator for
    // GUIDataUnit objects
    class GUIParameter : public QObject
    {
        Q_OBJECT

    public:
        GUIParameter() = delete;
        GUIParameter(const GUIParameter& other) = delete;
        GUIParameter(sl::SimParameter& parameter, double min, double max,
            double int_factor = 1)
            : QObject(), m_parameter(parameter), m_ui_elements(),
              m_update_in_progress(false), m_min(min), m_max(max),
              m_int_factor(int_factor)
        {}
        ~GUIParameter() = default;

        inline void add_data_unit(GUIDataUnit* unit)
        {
            m_ui_elements.push_back(unit);
        }

        inline double get_min() const
        {
            return m_min;
        }

        inline double get_max() const
        {
            return m_max;
        }

    signals:
        inline void updated();

    public slots:

        inline void update(double value)
        {
```

```
            if (!m_update_in_progress)
            {
                m_update_in_progress = true;
                m_parameter.set_value(value);
                update_others();
                m_update_in_progress = false;
                emit updated();
            }
        }

        inline void update(int value)
        {
            if (!m_update_in_progress)
            {
                m_update_in_progress = true;
                m_parameter.set_value(static_cast<double>(value) *
m_int_factor);
                update_others();
                m_update_in_progress = false;
                emit updated();
            }
        }

        inline void refresh()
        {
            if (!m_update_in_progress)
            {
                m_update_in_progress = true;
                update_others();
                m_update_in_progress = false;
                emit updated();
            }
        }

    private:
        bool m_update_in_progress;
        // A value, by which an input int value has to be
multiplied by,
        // in order to make it a proper simulation parameter value
        double m_int_factor;
        double m_min;
        double m_max;

        sl::SimParameter& m_parameter;
        std::vector<GUIDataUnit*> m_ui_elements;

        inline void update_others()
        {
            for (auto el : m_ui_elements)
            {
                if (el->should_update())
                {
                    el->set_data(m_parameter.get_value().second);
                }
            }
        }
    };
    } // namespace ui

    #endif // GUI_PARAMETER_H
```

1.1.6 gui_data_unit.h

```
    #ifndef GUI_DATA_UNIT_H
    #define GUI_DATA_UNIT_H

    #include "QObject"
    #include "QSlider"
    #include "QDoubleSpinBox"
    #include "QSpinBox"
    #include "QLineEdit"
    #include "../simulation_logic/sim_parameter.h"

    namespace ui
    {
    namespace sl = simulation;

    // A QWidget wrapper, that provides a unified interface to
set values to
    // QWidget objects
    // Inheritors must have their own pointers to QWidget
objects of specific
    // types, and override setting logic, because it may differ
greatly for
    // several widgets
    class GUIDataUnit
    {
    public:
        GUIDataUnit() : m_should_update(true)
        {}
        GUIDataUnit(const GUIDataUnit& other) = default;
        virtual ~GUIDataUnit() = default;

        virtual void set_data(double data) = 0;

        inline bool should_update() const
        {
            return m_should_update;
        }

        inline void should_update(bool should)
        {
```

```
        m_should_update = should;
    }

protected:
    bool m_should_update;
};

class SliderUnit : public GUIDataUnit
{
public:
    SliderUnit(QSlider* element, double int_factor = 1);
    SliderUnit(const SliderUnit& other) = default;
    ~SliderUnit() = default;

    void set_data(double data) override;

private:
    double m_int_factor;
    QSlider* m_slider;
};

class DoubleSpinBoxUnit : public GUIDataUnit
{
public:
    DoubleSpinBoxUnit(QDoubleSpinBox* element);
    DoubleSpinBoxUnit(const DoubleSpinBoxUnit& other)
= default;
    ~DoubleSpinBoxUnit() = default;

    void set_data(double data) override;

private:
    QDoubleSpinBox* m_spin_box;
};

class SpinBoxUnit : public GUIDataUnit
{
public:
    SpinBoxUnit(QSpinBox* element);
    SpinBoxUnit(const SpinBoxUnit& other) = default;
    ~SpinBoxUnit() = default;

    void set_data(double data) override;

private:
    QSpinBox* m_spin_box;
};

class DoubleLineEditUnit : public GUIDataUnit
{
public:
    DoubleLineEditUnit(QLineEdit* element, short decimals
= -1);
    DoubleLineEditUnit(const DoubleLineEditUnit& other)
= default;
    ~DoubleLineEditUnit() = default;

    void set_data(double data) override;
    void set_decimals(short decimals);

private:
    short m_decimals;
    QLineEdit* m_edit_box;
};
} // namespace ui

#endif // GUI_DATA_UNIT
```

1.1.7 gui_data_unit.cpp

```
#include <sstream>
#include "gui_data_unit.h"

namespace ui
{
// SliderUnit
SliderUnit::SliderUnit(QSlider* element, double int_factor)
    :                                          GUIDataUnit(),
m_slider(static_cast<QSlider*>(element)),
    m_int_factor(int_factor)
{}

void SliderUnit::set_data(double data)
{
    int slider_value = static_cast<int>(data / m_int_factor);
    int min = m_slider->minimum();
    int max = m_slider->maximum();
    slider_value = slider_value >= min ? slider_value : min;
    slider_value = slider_value <= max ? slider_value : max;

    should_update(false);
    m_slider->setValue(slider_value);
    should_update(true);
}

// DoubleSpinBoxUnit
DoubleSpinBoxUnit::DoubleSpinBoxUnit(QDoubleSpinBo
x* element)
    :                                          GUIDataUnit(),
m_spin_box(static_cast<QDoubleSpinBox*>(element))
{}
```

```
void DoubleSpinBoxUnit::set_data(double data)
{
   double box_value = data;
   double min = m_spin_box->minimum();
   double max = m_spin_box->maximum();
   box_value = box_value >= min ? box_value : min;
   box_value = box_value <= max ? box_value : max;

   should_update(false);
   m_spin_box->setValue(box_value);
   should_update(true);
}

// SpinBoxUnit
SpinBoxUnit::SpinBoxUnit(QSpinBox* element)
   :                                    GUIDataUnit(),
m_spin_box(static_cast<QSpinBox*>(element))
{}

void SpinBoxUnit::set_data(double data)
{
   int box_value = static_cast<int>(data);
   int min = m_spin_box->minimum();
   int max = m_spin_box->maximum();
   box_value = box_value >= min ? box_value : min;
   box_value = box_value <= max ? box_value : max;

   should_update(false);
   m_spin_box->setValue(box_value);
   should_update(true);
}

// DoubleLineEditUnit
DoubleLineEditUnit::DoubleLineEditUnit(QLineEdit*
element,
   short decimals)
   :                                    GUIDataUnit(),
m_edit_box(static_cast<QLineEdit*>(element)),
   m_decimals(decimals)
{}

void DoubleLineEditUnit::set_decimals(short decimals)
{
   m_decimals = decimals;
}

void DoubleLineEditUnit::set_data(double data)
{
   std::stringstream ss;
   ss << data;
   std::string data_str;
   ss >> data_str;

   QString new_value;
   size_t comma_index;

   if (m_decimals < 0 or
      (comma_index = data_str.find(".")) ==
std::string::npos)
   {
      new_value = data_str.c_str();
   }
   else if (m_decimals == 0 and comma_index !=
std::string::npos)
   {
      new_value = data_str.substr(0, comma_index).c_str();
   }
   else
   {
      new_value = data_str.substr(0, comma_index +
m_decimals + 1).c_str();
   }

   should_update(false);
   m_edit_box->setText(new_value);
   should_update(true);
}

}
```

1.1.8 scene_unit.h

```
#ifndef SCENE_UNIT_H
#define SCENE_UNIT_H
#include "gui_data_unit.h"
#include <QGraphicsScene>
#include <QGraphicsPixmapItem>
#include "QPixmap"

namespace ui
{
struct DrawContext
{
   double value = 0;
};

// A subtype of GUIDataUnit, that is used with
QGraphicsScene
class SceneUnit : public GUIDataUnit
{
public:
   SceneUnit() = delete;
```

```
    SceneUnit(const SceneUnit& other) = default;
    SceneUnit(QGraphicsScene& scene)
      : GUIDataUnit(), m_scene(scene)
    {}
    virtual ~SceneUnit() = default;

  protected:
    QGraphicsScene& m_scene;

    virtual void draw(DrawContext context) = 0;
  };

  class MagFieldSceneUnit : public SceneUnit
  {
  public:
    MagFieldSceneUnit() = delete;
    MagFieldSceneUnit(const MagFieldSceneUnit& other) =
default;
    MagFieldSceneUnit(QGraphicsScene& scene);
    ~MagFieldSceneUnit() = default;

    void set_data(double data) override;

  private:
    QPixmap m_arrow_img;
    QPixmap m_protr_img;

    QGraphicsPixmapItem* m_arrow_item;
    QGraphicsPixmapItem* m_protr_item;

    void draw(DrawContext context) override;
  };

  class AmpermeterSceneUnit : public SceneUnit
  {
  public:
    AmpermeterSceneUnit() = delete;
    AmpermeterSceneUnit(const AmpermeterSceneUnit&
other) = default;
    AmpermeterSceneUnit(QGraphicsScene& scene);
    ~AmpermeterSceneUnit() = default;

    void set_data(double data) override;
    inline void set_min_value(double value)
    {
      m_value_min = value;
    }

    inline void set_max_value(double value)
    {
      m_value_max = value;
    }

  private:
    double m_angle_min;
    double m_angle_max;

    double m_value_min;
    double m_value_max;

    QPixmap m_amp_scale_img;
    QPixmap m_amp_arrow_img;
    QGraphicsPixmapItem* m_amp_scale_item;
    QGraphicsPixmapItem* m_amp_arrow_item;

    void draw(DrawContext context) override;
    double translate_to_scene(double value);
  };

  } // namespace ui

  #endif // SCENE_UNIT_H
```

1.1.9 scene_unit.cpp

```
  #include "scene_unit.h"

  namespace ui
  {
  // MagFieldSceneUnit
  MagFieldSceneUnit::MagFieldSceneUnit(QGraphicsScene
& scene)
    : SceneUnit(scene), m_arrow_img(":mag_arrow"),
    m_protr_img(":protr"),
    m_protr_item(scene.addPixmap(m_protr_img)),
    m_arrow_item(scene.addPixmap(
      m_arrow_img.scaledToHeight(m_protr_img.height())))
  {
    m_arrow_item->setZValue(1);
    m_arrow_item-
>setTransformOriginPoint(m_arrow_item->boundingRect().center());
  }

  void MagFieldSceneUnit::set_data(double data)
  {
    draw({ data });
  }

  void MagFieldSceneUnit::draw(DrawContext context)
  {
    m_arrow_item->setRotation(context.value);
  }
```

```
// AmpermeterSceneUnit
AmpermeterSceneUnit::AmpermeterSceneUnit(QGraphicsScene& scene)
    : SceneUnit(scene),
    m_amp_scale_img(":amp_scale"), m_amp_arrow_img(":amp_arrow"),

m_amp_scale_item(m_scene.addPixmap(m_amp_scale_img)),

m_amp_arrow_item(m_scene.addPixmap(m_amp_arrow_img)),
    m_angle_min(0), m_angle_max(90)
{
    QPointF scale_rect_br = m_amp_scale_item->boundingRect().bottomRight();
    QRectF arrow_rect = m_amp_arrow_item->boundingRect();
    m_amp_arrow_item->setOffset(scale_rect_br.x() - arrow_rect.width(),
      scale_rect_br.y() - arrow_rect.height());

    // Arrow transformations origin point
    QPointF origin_point;
    QPointF arrow_rect_br = m_amp_arrow_item->boundingRect().bottomRight();
    float offset = arrow_rect.height() / 2;
    origin_point.setX(arrow_rect_br.x() - offset);
    origin_point.setY(arrow_rect_br.y() - offset);
    m_amp_arrow_item->setTransformOriginPoint(origin_point);
}

void AmpermeterSceneUnit::set_data(double data)
{
  draw({ data });
}

void AmpermeterSceneUnit::draw(DrawContext context)
{
  m_amp_arrow_item->setRotation(translate_to_scene(context.value));
}

double AmpermeterSceneUnit::translate_to_scene(double value)
{
  if (value < m_value_min)
    return m_value_min;

  if (value > m_value_max)
    return m_value_max;

  return ((m_value_min + value) * m_angle_max)/(m_value_max) - m_angle_min;
}
}
```

1.2 Класи фізичних обчислень

1.2.1 simulation_includes.h

```
#ifndef SIMULATION_INCLUDES_H
#define SIMULATION_INCLUDES_H

#include <vector>

namespace simulation
{
// bool first - viability flag, double second - value
typedef std::pair<bool, double> ParamResult;
} // namespace simulation

#endif // SIMULATION_INCLUDES_H
```

1.2.2 sim_parameter.h

```
#ifndef SIM_PARAMETER_H
#define SIM_PARAMETER_H

#include <vector>
#include "simulation_includes.h"
#include "formula.h"

namespace simulation
{
class Formula;

// Simulation parameter
```

```
        // Represents a graph node with a value, can be linked to other SimParameter instances
        // Performs recalculations of other SimParameter instances, when they are:
        // - used in formulas of the parameter (downstream);
        // - are explicitly linked to the object as dependent objects (upstream).
        class SimParameter
        {
        public:
            SimParameter() = default;
            SimParameter(double value, std::vector<Formula> formulae = {},
                std::vector<SimParameter*> dependents = {})
                : m_value(true, value), m_formulae(formulae), m_value_changed(true),
                m_dependents(dependents), m_recalc_in_progress(false)
            {}
            SimParameter(const SimParameter& other) = default;
            ~SimParameter() = default;

            // Starts recalculation when the value has changed - subsequently
            // recalculates linked parameters as well
            inline ParamResult get_value()
            {
                if (m_value_changed and !m_recalc_in_progress)
                {
                    recalculate();
                }

                return m_value;
            }

            void recalculate();

            inline void set_value(double value)
            {
                m_value_changed = true;
                m_value.first = true;
                m_value.second = value;
            }

            inline void add_dependent(SimParameter* dependent)
            {
                m_dependents.emplace_back(dependent);
            }

            inline void add_formula(Formula formula)
            {
                m_formulae.emplace_back(formula);
            }

        private:
            bool m_recalc_in_progress;
            bool m_value_changed;
            ParamResult m_value;
            std::vector<Formula> m_formulae;
            std::vector<SimParameter*> m_dependents;

            void notify();
        };
        } // namespace simulation

        #endif // SIM_PARAMETER_H
```

1.2.3 sim_parameter.cpp

```
        #include "sim_parameter.h"
        #include <string>

        namespace simulation
        {
        void SimParameter::recalculate()
        {
            if (m_recalc_in_progress)
            {
                return;
            }

            m_value_changed = false;
            m_recalc_in_progress = true;

            if (!m_formulae.size())
            {
                notify();
                m_recalc_in_progress = false;
                return;
            }
            m_value.first = false;
            ParamResult result;

            for (auto formula : m_formulae)
            {
                result = formula.calculate();
                if (result.first)
                {
                    m_value = result;
                    notify();
                    m_recalc_in_progress = false;
                    return;
                }
```

```
    }

    m_recalc_in_progress = false;
}

void SimParameter::notify()
{
    for (auto dep : m_dependents)
    {
        if (!dep->m_recalc_in_progress)
            dep->recalculate();
    }
}

}
```

1.2.4 formula.h

```
#ifndef FORMULA_H
#define FORMULA_H

#include "simulation_includes.h"
#include "sim_parameter.h"

namespace simulation
{
class SimParameter;

// Formula, that SimParameter objects use to recalculate their values.
// Formula is represented as a sequence of arguments, and a callback function
// that uses them.
// The callback function must return ParamResult object and take
// std::vector<SimParameter*>& argument.
class Formula
{
    typedef ParamResult (*formula)(std::vector<SimParameter*>& args);

public:
    Formula(std::vector<SimParameter*> args, formula formula)
        : m_args(args), m_formula(formula)
    {}
    Formula(const Formula& other) = default;
    ~Formula() = default;

    inline ParamResult calculate()
    {
        return m_formula(m_args);
    }

private:
    std::vector<SimParameter*> m_args;
    formula m_formula;
};
} // namespace simulation

#endif // FORMULA_H
```

1.3 Класи модульного тестування

1.3.1 gui_tests.cpp

```
#include <QtTest/QtTest>
#include <QtTestWidgets>
#include "QSlider"
#include "QDoubleSpinBox"
#include "QSpinBox"
#include "QLineEdit"
#include "../src/gui/gui_parameter.h"

using namespace ui;

class GUITests : public QObject
{
    Q_OBJECT

private slots:
    void gui_slider_update()
    {
        QSlider slider;
        slider.setValue(0);
        SliderUnit slider_unit(&slider);

        QVERIFY(slider.value() == 0);
        slider_unit.set_data(10);
        QVERIFY(slider.value() == 10);
    }

    void gui_spin_box_update()
    {
        QSpinBox spin_box;
        spin_box.setValue(0);
        SpinBoxUnit spin_box_unit(&spin_box);

        QVERIFY(spin_box.value() == 0);
        spin_box_unit.set_data(10);
        QVERIFY(spin_box.value() == 10);
    }
```

```
void gui_double_spin_box_update()
{
    QDoubleSpinBox d_spin_box;
    d_spin_box.setValue(0);
    DoubleSpinBoxUnit d_spin_box_unit(&d_spin_box);

    QVERIFY(d_spin_box.value() == 0);
    d_spin_box_unit.set_data(10);
    QVERIFY(d_spin_box.value() == 10);
}

void gui_double_line_edit_update()
{
    QLineEdit line_edit;
    line_edit.setText("0");
    DoubleLineEditUnit line_edit_unit(&line_edit);

    QVERIFY(line_edit.text() == "0");
    line_edit_unit.set_data(10);
    QVERIFY(line_edit.text() == "10");
}

void gui_double_line_edit_decimals_data()
{
    QTest::addColumn<short>("decimals");
    QTest::addColumn<double>("data");
    QTest::addColumn<QString>("expected");

    double common_data = 10.123456;

    QTest::newRow("dec = -1") << (short)-1 <<
common_data << "10.1235";
    QTest::newRow("dec = 0") << (short)0 <<
common_data << "10";
    QTest::newRow("dec > 0") << (short)3 <<
common_data << "10.123";
}

void gui_double_line_edit_decimals()
{
    QLineEdit line_edit;
    line_edit.setText("0");
    DoubleLineEditUnit line_edit_unit(&line_edit);
    QVERIFY(line_edit.text() == "0");

    QFETCH(short, decimals);
    QFETCH(double, data);
    QFETCH(QString, expected);

    line_edit_unit.set_decimals(decimals);
    line_edit_unit.set_data(data);
    QVERIFY(line_edit.text() == expected);
}

void gui_param_update_units()
{
    sl::SimParameter raw_param;
    GUIParameter param(raw_param, 0, 10);

    QSlider slider;
    slider.setValue(0);
    QDoubleSpinBox d_spin_box;
    d_spin_box.setValue(0);
    QSpinBox spin_box;
    spin_box.setValue(0);
    QLineEdit line_edit;
    line_edit.setText("0");

    SliderUnit slider_unit(&slider);
    DoubleSpinBoxUnit d_spin_box_unit(&d_spin_box);
    SpinBoxUnit spin_box_unit(&spin_box);
    DoubleLineEditUnit line_edit_unit(&line_edit);

    param.add_data_unit(&slider_unit);
    param.add_data_unit(&d_spin_box_unit);
    param.add_data_unit(&spin_box_unit);
    param.add_data_unit(&line_edit_unit);

    QVERIFY(slider.value() == 0);
    QVERIFY(d_spin_box.value() == 0);
    QVERIFY(spin_box.value() == 0);
    QVERIFY(line_edit.text() == "0");

    param.update(10);

    QVERIFY(slider.value() == 10);
    QVERIFY(d_spin_box.value() == 10);
    QVERIFY(spin_box.value() == 10);
    QVERIFY(line_edit.text() == "10");
}

void gui_param_refresh()
{
    sl::SimParameter raw_param;
    GUIParameter param(raw_param, 0, 10);

    QSlider slider;
    slider.setValue(0);
    SliderUnit slider_unit(&slider);
    param.add_data_unit(&slider_unit);
    QVERIFY(slider.value() == 0);

    raw_param.set_value(5);
```

```
        param.refresh();
        QVERIFY(slider.value() == 5);
    }
};
```

1.3.2 simulation_tests.cpp

```
#include <QtTest/QtTest>
#include <vector>
#include "../src/simulation_logic/sim_parameter.h"

using namespace simulation;

class SimulationTests : public QObject
{
    Q_OBJECT

private:
    Formula copy_formula(SimParameter* param)
    {
        return Formula(
            { param },
            [] (std::vector<SimParameter*>& args) -> ParamResult
            {
                return args[0]->get_value();
            });
    }

    Formula sum_formula(std::vector<SimParameter*> params)
    {
        return Formula(
            params,
            [] (std::vector<SimParameter*>& args) -> ParamResult
            {
                double sum = 0;
                ParamResult ret;

                for (auto arg : args)
                {
                    ret = arg->get_value();
                    if (!ret.first)
                    {
                        return ret;
                    }
                    sum += ret.second;
                }

                return { true, sum };
            });
    }

private slots:
    void formula_validity()
    {
        // Leaf params
        SimParameter s1(-30), s2(20.5), s3(10);
        // Sum of s1 and s2
        SimParameter p1(0, { sum_formula({ &s1, &s2, &s3 }) });

        ParamResult res = p1.get_value();
        QVERIFY(res.first and res.second == 0.5);
    }

    void downstream_recalculations()
    {
        // Leaf params
        SimParameter s1(1), s2(2);
        // Sum of s1 and s2
        SimParameter p1(0, { sum_formula({ &s1, &s2 }) });
        // Copies value of p1, making it recalculate itself
        SimParameter p0(0, { copy_formula(&p1) });

        ParamResult p0_res = p0.get_value();
        ParamResult p1_res = p1.get_value();

        QVERIFY(p1_res.first and p1_res.second == 3);
        QVERIFY(p0_res.first and p0_res.second == 3);
    }

    void upstream_recalculations()
    {
        // Leaf param
        SimParameter p1(0);
        // Upstream param
        SimParameter p0(0, { copy_formula(&p1) });

        // Recalculate and update them
        ParamResult p0_res = p0.get_value();
        ParamResult p1_res = p1.get_value();

        QVERIFY(p1_res.first and p1_res.second == 0);
        QVERIFY(p0_res.first and p0_res.second == 0);
    }

QTEST_MAIN(GUITests)
#include "gui_tests.moc"
```

```
    // Update p0 when recalculating p1
    p1.add_dependent(&p0);
    // Update and recalculate p1
    p1.set_value(10);
    p1.recalculate();
    // p0 won't be recalculated in get_value() because
    // it wasn't updated with set().
    // But the value should change because of p1's recalculation
    p0_res = p0.get_value();

    QVERIFY(p0_res.first and p0_res.second == 10);
    QVERIFY(true);
  }

  void downstream_cyclic_dependency()
  {
    SimParameter p1(1), p2(2), p3(3);
    // p1<-p2<-p3<-p1...
    p1.add_formula(copy_formula(&p2));
    p2.add_formula(copy_formula(&p3));
    p3.add_formula(copy_formula(&p1));

    // Should be { false, 1 }
    ParamResult result = p1.get_value();
    QVERIFY(!result.first);
  }

  void upstream_cyclic_dependency()
  {
    // Upstream recalculations just notify dependent params about the
    // change and don;t return anything. The purpose of the test is to
    // verify that the recalculation isn't infinite.
    SimParameter p1(1), p2(2), p3(3);
    // p1->p2->p3->p1...
    p1.add_dependent(&p2);
    p2.add_dependent(&p3);
    p3.add_dependent(&p1);

    ParamResult result = p1.get_value();  // Should be { true, 1 }
    QVERIFY(result.first);
  }
};

QTEST_MAIN(SimulationTests)
#include "simulation_tests.moc"
```

ДОДАТОК Д

КЕРІВНИЦТВО З ПРОВЕДЕННЯ ДОСЛІДІВ

«magnetic_field_exploration» – програма проведення фізичних симуляцій експерименту з лослідження магнітного поля.

В експериментів використовуються:

- коловий виток, площина якого розташована вздовж магнітного меридіану Землі;
- магнітна стрілка, розташована всередині колового витка;
- джерело струму.

Якщо по коловому витку подати струм, то кут повороту стрілки буде визначатися дією магнітного поля Землі і магнітного поля витка.

При користуванні програмою, для зміни параметрів симуляції можна скористатись наданими полями в графічному інтерфейсі. З їх допомогою, значення параметрів можна задавати в текстовому вигляді. Також, параметр сили струму можна змінювати з допомогою повзунка.

Одразу після зміни значення параметру, результати симуляції оновлюються в реальному часі. Їх можна переглядати в текстовому вигляді, з можливістю копіювання до буферу обміну, або в графічному вигляді. В графічному вигляді можна побачити репрезентацію значення сили струму, у вигляді зображення амперметра, та магнітну стрілку із транспортиром.