Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка застосунку для генерації оптичних ілюзій з використанням мови Java і платформи JavaFx»

Виконав: студент 4 курсу
групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Вітовський С.М.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ЗІ Маліновський В.І.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ _____
«06» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Вітовському Сергію Михайловичу

1. Тема роботи – розробка застосунку для генерації оптичних ілюзій з використанням мови Java і платформи JavaFx.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: перелік оптичних ілюзій – статичні, динамічні; методи візуалізації – Canvas API, JavaFX GraphicsContext; режим відображення – апаратно прискорена векторна графіка; вихідні параметри – координати елементів ілюзій, кольори, швидкість анімації, кількість об'єктів; вхідні дані – вибір типу ілюзії, налаштування параметрів, вибір кольорової палітри, взаємодія з інтерфейсом користувача; методи реалізації – інкапсуляція елементів у класи, обробка подій у режимі реального часу; програмне середовище – Java, JavaFX, Maven, середовище розробки IntelliJ IDEA; вихідні дані – візуалізоване зображення ілюзії на полотні.

4. Зміст текстової частини: вступ; аналіз методів моделювання оптичних ілюзій; аналіз аналогів; аналіз методів розв'язання задачі; постановка задач; аналіз вхідних даних системи; розробка методу створення структури системи; розробка алгоритмів роботи системи; аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення; розробка основних модулів застосунку; розробка методу генерації оптичних ілюзій; тестування; висновки; список використаних джерел; додатки.

5. Перелік ілюстративної частини: зображення аналогів; блок-схеми алгоритмів роботи застосунку; схема інтерфейсу; графічний інтерфейс застосунку; тестування застосунку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Катєльніков Д.І. к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз методів моделювання оптичних ілюзій у цифровому середовищі	25.03.25 – 09.04.25	вик.
2	Розробка методу створення структури та алгоритмів роботи системи	10.04.25 – 30.04.25	вик.
3	Розробка основних модулів застосунку та методу генерації оптичних ілюзій	31.04.25 – 16.05.25	вик
4	Тестування роботи застосунку	17.05.25 – 23.05.25	вик.
5	Оформлення матеріалів до захисту БКР	24.05.25– 30.05.25	вик

Студент _____ **С.М. Вітовський**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **Д.І. Катєльніков**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 99 сторінок формату А4, на яких представлено 25 рисунків, 2 таблиці, 4 додатки. Список використаних джерел містить 18 найменувань.

У бакалаврській кваліфікаційній роботі було розроблено методи та програмний застосунок для генерації оптичних ілюзій із використанням мови програмування Java та графічної бібліотеки JavaFX. Проведено аналіз існуючих моделей побудови візуальних ілюзій у цифровому середовищі, класифікацію основних типів ілюзій та порівняння наявних програмних рішень у цій сфері. Обґрунтовано доцільність створення власного застосунку з інтерактивним інтерфейсом, налаштуванням параметрів та можливістю експорту результатів у графічні формати.

Розроблено структуру застосунку, що дозволяє модульно реалізовувати різні типи оптичних ілюзій – як статичних, так і динамічних. Створено систему налаштувань для кожної ілюзії, реалізовано механізм візуалізації в режимі реального часу, а також довідкову систему, яка надає користувачу пояснення принципу дії ілюзії.

На основі проведених досліджень створено ефективний інструмент для демонстрації зорових ефектів, що може бути використаний у сфері освіти, наукових досліджень та дизайну. Проведене тестування підтвердило стабільність роботи застосунку, правильність візуалізації та відповідність заявленим функціональним вимогам.

Ключові слова: Java, JavaFX, оптичні ілюзії, графічний інтерфейс, анімація, візуалізація.

ABSTRACT

The bachelor's qualification work consists of 99 A4 pages, which present 25 figures, 2 tables, and 4 appendices. The list of sources used contains 18 names.

In the bachelor's qualification work, methods and a software application were qualification workthe JavaFX graphics library. An analysis of existing models for constructing visual illusions in a digital environment, a classification of the main types of illusions, and a comparison of existing software solutions in this area were conducted. The feasibility of creating your own application with an interactive interface, parameter settings, and the ability to export results to graphic formats was substantiated.

The application structure was developed, which allows for modular implementation of various types of optical illusions - both static and dynamic. A system of settings for each illusion was created, a real-time visualization mechanism was implemented, and a help system was implemented that provides the user with an explanation of the principle of operation of the illusion.

Based on the research, an effective tool for demonstrating visual effects has been created, which can be used in the field of education, scientific research and design. The testing confirmed the stability of the application, the correctness of the visualization and compliance with the stated functional requirements.

Keywords: Java, JavaFX, optical illusions, graphical interface, animation, visualization.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У період активного розвитку технологій цифрова візуалізація займає центральне місце в багатьох аспектах життя сучасної людини, від освітніх програм і наукових досліджень до рекламних кампаній, мистецьких проєктів, комп'ютерних ігор та дизайну. Сприйняття візуального контенту, його аналіз і вплив на користувача відіграють ключову роль у створенні ефективного цифрового середовища. Саме в цьому контексті оптичні ілюзії стають не лише об'єктом зацікавлення з боку науковців і художників, а й потужним інструментом для вивчення та маніпуляції зоровим сприйняттям.

Оптичні ілюзії [1] – це явища, в яких спостерігається розбіжність між фізичною реальністю зображення та тим, як його інтерпретує наш мозок. Вони демонструють, що зорове сприйняття не є прямим відображенням об'єктивного світу, а складним когнітивним процесом, який залежить від багатьох факторів: контексту, контрасту, руху, кольору, освітлення, досвіду тощо. Здатність оптичних ілюзій змінювати або спотворювати сприйняття робить їх цінним матеріалом як для когнітивної психології, так і для педагогіки, дизайну та мистецтва [2].

З огляду на це, розробка цифрових інструментів, які дозволяють моделювати, досліджувати та створювати оптичні ілюзії, є надзвичайно актуальною. Такий інструмент може служити основою для наукових експериментів, навчальних демонстрацій, художніх інсталяцій або навіть терапевтичних методик. Проте на сьогоднішній день існуючі засоби генерації оптичних ілюзій часто мають вузький функціонал, застарілий підхід до інтерфейсу або орієнтовані лише на окремі типи ілюзій. Це суттєво обмежує можливості як дослідників, так і творчих користувачів у роботі з візуальними ефектами.

Зокрема, актуальними стають динамічні ілюзії, які змінюються в часі, або ті, що взаємодіють з користувачем. Вони дозволяють не лише демонструвати

ефекти, а й вивчати адаптивність зорової системи, взаємозв'язок між рухом та інтерпретацією зображення, механізми втоми рецепторів тощо. Створення інтерактивного середовища для експериментування з такими ефектами може відкрити нові підходи у візуальній освіті та наукових дослідженнях.

Також не менш важливим є потенціал оптичних ілюзій у дизайні як засобу привернення уваги, створення глибини, руху або обману перспективи [3]. У цифровому мистецтві ілюзії часто виступають центральним елементом композиції, що впливає на емоційне та естетичне сприйняття твору. Їхнє використання дозволяє художникам створювати нестандартні образи, що викликають інтерес і залученість.

Усе це свідчить про необхідність створення зручного, функціонального та універсального інструменту для роботи з оптичними ілюзіями в цифровому середовищі. Такий інструмент має відповідати потребам дослідників, викладачів, дизайнерів і художників, пропонуючи гнучкі можливості для генерації, редагування, вивчення й експорту ілюзій у різних форматах.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є розширення можливостей застосування оптичних ілюзій у навчальних, наукових, дизайнерських та експериментальних цілях шляхом розробки алгоритмів та програмного застосунку для генерації оптичних ілюзій, який дозволяє створювати як статичні, так і динамічні візуальні ефекти з можливістю інтерактивного налаштування параметрів, візуалізації результатів у реальному часі та збереження зображень у поширених графічних форматах.

Для досягнення мети потрібно виконати низку завдань:

− провести аналіз існуючих методів створення оптичних ілюзій та засобів візуального моделювання;

− розробити метод створення структури застосунку, який включає модулі генерації, візуалізації та збереження зображень;

− реалізувати метод генерації ілюзій із використанням графічних примітивів та ефектів JavaFX;

− створити графічний інтерфейс, що дозволяє користувачу взаємодіяти із застосунком у режимі реального часу;

− реалізувати функції експорту створених ілюзій та забезпечити зручну систему навігації інтерфейсом;

− протестувати функціональність застосунку та провести оцінку його ефективності й зручності.

Об’єкт дослідження – це процес розробки програмного забезпечення для створення, моделювання та демонстрації оптичних ілюзій у цифровому середовищі.

Предмет дослідження – це методи і алгоритми створення оптичних ілюзій у цифровому середовищі.

Методи дослідження. У процесі досліджень використовувались: теорія кольору і гештальтпсихологія; методи програмної інженерії зокрема методи побудови користувацьких інтерфейсів і логіки взаємодії компонентів системи; методи об'єктно-орієнтованого програмування та шаблони проектування для розробки архітектури системи; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод генерації оптичних ілюзій, який, на відміну від існуючих, поєднує динамічну візуалізацію з можливістю інтерактивного налаштування параметрів зображення, що дозволяє досягти адаптивного управління візуальними ефектами в реальному часі і, таким чином, зробити вплив більш персоніфікованим і поглибленим.

2. Подальшого розвитку отримав метод створення програмного забезпечення для генерації оптичних ілюзій, який, на відміну від існуючих, використовує модульну архітектуру, що забезпечує гнучкість у розширенні функціоналу, оскільки дозволяє легко інтегрувати нові типи ілюзій без суттєвих змін базового коду.

Практична цінність отриманих результатів полягає у тому, що в результаті було розроблено програмний засіб, який дає змогу генерувати і досліджувати оптичні ілюзії у візуальній формі. Застосунок може бути використаний у закладах освіти під час вивчення фізіології зору, візуального мистецтва, інформатики, а також у наукових експериментах для аналізу реакцій на певні типи візуальних стимулів.

Особистий внесок здобувача. Усі результати, викладені у кваліфікаційній роботі, отримані автором особисто. У друкованій праці [4], опублікованій у співавторстві, автору належать такі результати: розробки програмного забезпечення від формулювання ідеї до реалізації, тестування та оформлення результатів.

Апробація результатів роботи. Описані у дослідженні положення доповідались на конференції «LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (2025)».

Публікації. Основні результати досліджень опубліковано в матеріалах конференції «LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (2025)» [4].

1 АНАЛІЗ ПІДХОДІВ ДО МОДЕЛЮВАННЯ ОПТИЧНИХ ІЛЮЗІЙ У ЦИФРОВОМУ СЕРЕДОВИЩІ ТА ФОРМУЛЮВАННЯ ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1. Аналіз методів моделювання оптичних ілюзій у цифровому середовищі

З розвитком цифрових технологій процес моделювання оптичних ілюзій зазнав значного вдосконалення та розширення своїх можливостей у найрізноманітніших галузях – від індустрії розваг та цифрового мистецтва до медицини, психології та наукових досліджень. Оптичні ілюзії базуються на психологічних та фізіологічних особливостях зорового сприйняття, які змушують мозок інтерпретувати візуальну інформацію хибним або спотвореним чином [5]. У цифровому середовищі стало можливим точно та гнучко відтворювати ці ефекти, використовуючи як класичні методи – геометричні спотворення, контраст, гру світла і тіні, ілюзії руху [6], так і новітні технології, пов'язані з комп'ютерною графікою та алгоритмічною генерацією візуального контенту.

Значну роль у розумінні механізмів оптичних ілюзій відіграє гештальт-психологія, яка сформувала низку принципів зорового сприйняття, зокрема принцип фігури та фону, замкнутості, близькості, подібності, продовження та симетрії. Ці принципи пояснюють, чому людський мозок прагне «доповнювати» відсутню інформацію або сприймати неоднозначні зображення як цілісні структури. Врахування гештальт-принципів під час розробки цифрових моделей ілюзій дозволяє створювати глибші та переконливіші візуальні ефекти, що викликають відповідну когнітивну реакцію.

Багато сучасних ілюзій моделюються з урахуванням не лише геометричних візерунків, а й механізмів когнітивної інтерпретації, що проявляються у сприйнятті контексту, взаємозв'язків між об'єктами, напрямку руху чи освітлення. Наприклад, принцип «контекстного спотворення» або

«ілюзії впливу фону» базується саме на гештальт-теорії, згідно з якою сприйняття окремого об'єкта завжди залежить від його оточення. Таким чином, під час розробки програмного забезпечення для генерації ілюзій важливо не лише забезпечити технічну реалізацію графічного ефекту, але й врахувати закономірності зорової системи, які базуються на психологічних та когнітивних законах.

Поєднання принципів гештальту із сучасними інструментами візуалізації на базі JavaFX дозволяє створювати адаптивні інтерфейси, які динамічно реагують на зміни параметрів та забезпечують високий ступінь занурення користувача в процес сприйняття візуального ефекту. Це відкриває нові можливості для використання такого програмного забезпечення в навчанні, тестуванні когнітивних функцій, терапії зорових розладів або створенні художніх інсталяцій.

Однією з найефективніших стратегій моделювання оптичних ілюзій є використання математичних моделей та алгоритмів, що дозволяють формувати складні візуальні конфігурації. Ефект Морі, що виникає внаслідок суперпозиції періодичних структур, широко використовується для створення ілюзій псевдоруху, вібрації або глибини. Аналогічно, фрактальна геометрія використовується для побудови композицій, що порушують очікуване сприйняття пропорцій та симетрії. Завдяки математичній природі цих ілюзій, вони легко адаптуються для комп'ютерної генерації, зберігаючи високий ступінь точності.

Сучасні засоби програмування, такі як JavaFX, надають розширений функціонал для створення динамічних ілюзій за допомогою векторної графіки, анімації, прозорості та складних світлових ефектів. Це дозволяє не тільки змінювати форми та кольори в режимі реального часу, але й реалізовувати ілюзії, що реагують на дії користувача, наприклад, змінюючи зовнішній вигляд при русі миші або змінюючи масштаб. Такі інтерактивні компоненти значно підвищують ефективність сприйняття ілюзії та її адаптивність до контексту використання.

Крім того, розвиток технологій штучного інтелекту та машинного навчання створює умови для персоналізованого підходу до створення оптичних ілюзій. Наприклад, алгоритми можуть аналізувати індивідуальні характеристики зору користувача – такі як схильність до певних видів ілюзій, наявність порушень зору або колірної чутливості – та автоматично адаптувати параметри ілюзії на основі цього. Такий підхід особливо корисний у медицині, зокрема для діагностики зорових аномалій, тестування периферичного зору, реакцій на зорові подразники та відновлення після офтальмологічних операцій [7].

Інтерактивні системи, що імітують оптичні ілюзії, також знаходять застосування в дизайні інтерфейсів, ігрових технологіях, віртуальній та доповненій реальності, де можливість маніпулювати сприйняттям користувача може покращити ефекти занурення та створити унікальний користувацький досвід. Використання таких систем у цифровому мистецтві дозволяє художникам експериментувати з візуальним простором, формами та кольорами способами, які раніше були недосяжними.

В результаті цифрове моделювання оптичних ілюзій стало потужним інструментом не лише для візуальних експериментів, але й для серйозних прикладних досліджень. Сучасні програмні засоби відкривають нові горизонти для створення реалістичних, складних та налаштовуваних візуальних ефектів, які служать як інструментом для розуміння особливостей людського сприйняття, так і засобом вираження у творчих та наукових проектах.

1.2 Порівняльний аналіз аналогів

З розвитком цифрових технологій процес створення оптичних ілюзій вийшов за межі лабораторій науковців та творчих майстерень художників, ставши доступним широкому колу користувачів. Завдяки сучасним програмним засобам цей процес значно спростився та автоматизувався. Сьогодні будь-хто, хто має комп'ютер або мобільний пристрій, може створювати складні візуальні ефекти без необхідності глибоких знань фізіології зору чи алгоритмів. Програмні

засоби нового покоління забезпечують високу інтерактивність, реалістичність та адаптивність ілюзій – користувачі можуть змінювати параметри, експериментувати з кольором, формою, швидкістю та іншими характеристиками в режимі реального часу, отримуючи миттєвий візуальний результат. Це породило низку спеціалізованих додатків, здатних генерувати як статичні зображення на основі геометричних спотворень, контрастів та симетрії, так і динамічні анімаційні ефекти, що змінюються з часом або реагують на дії користувача.

Сучасний ринок цифрових технологій пропонує кілька популярних програмних продуктів, що спеціалізуються на створенні оптичних ілюзій. Кожен з них пропонує свій власний підхід до реалізації візуальних ефектів, має характерний набір функцій, графічний стиль та технічні можливості. Наприклад, «Visual Phenomena & Optical Illusions» дозволяє користувачам переглядати, змінювати та вивчати класичні та сучасні ілюзії з поясненнями механізмів їхньої дії. У свою чергу, веб-платформа «The Illusion Index» поєднує науковий підхід та базу знань про ілюзії, пропонуючи інтерактивні приклади з можливістю вивчення ефектів з точки зору когнітивної науки. Іншим прикладом є додаток «Silk – Interactive Generative Art», який дозволяє створювати візуально привабливі, симетричні анімації в художньому стилі, часто викликаючи ефект гіпнотичного сприйняття завдяки плавним рухам та грі кольорів.

Одним з прикладів використання цифрових технологій для демонстрації оптичних ілюзій є вебзастосунок «Visual Phenomena & Optical Illusions» [8], створений німецьким нейрофізіологом Міхаелем Бахом. Цей інструмент являє собою інтерактивну онлайн колекцію, яка налічує понад сто різноманітних ілюзій, класифікованих за тематичними категоріями, такими як ілюзії руху, перспективи, кольору, контрасту, яскравості та інших візуальних характеристик. Унікальність цього ресурсу полягає в його науковій орієнтації. Кожна ілюзія супроводжується докладним описом, що ґрунтується на психофізіологічних аспектах зорового сприйняття. Крім того, платформа виконує важливу функцію

популяризації знань про зорові процеси, сприяючи залученню широкої аудиторії до теми сприйняття та ілюзій (див. рисунок 1.1).

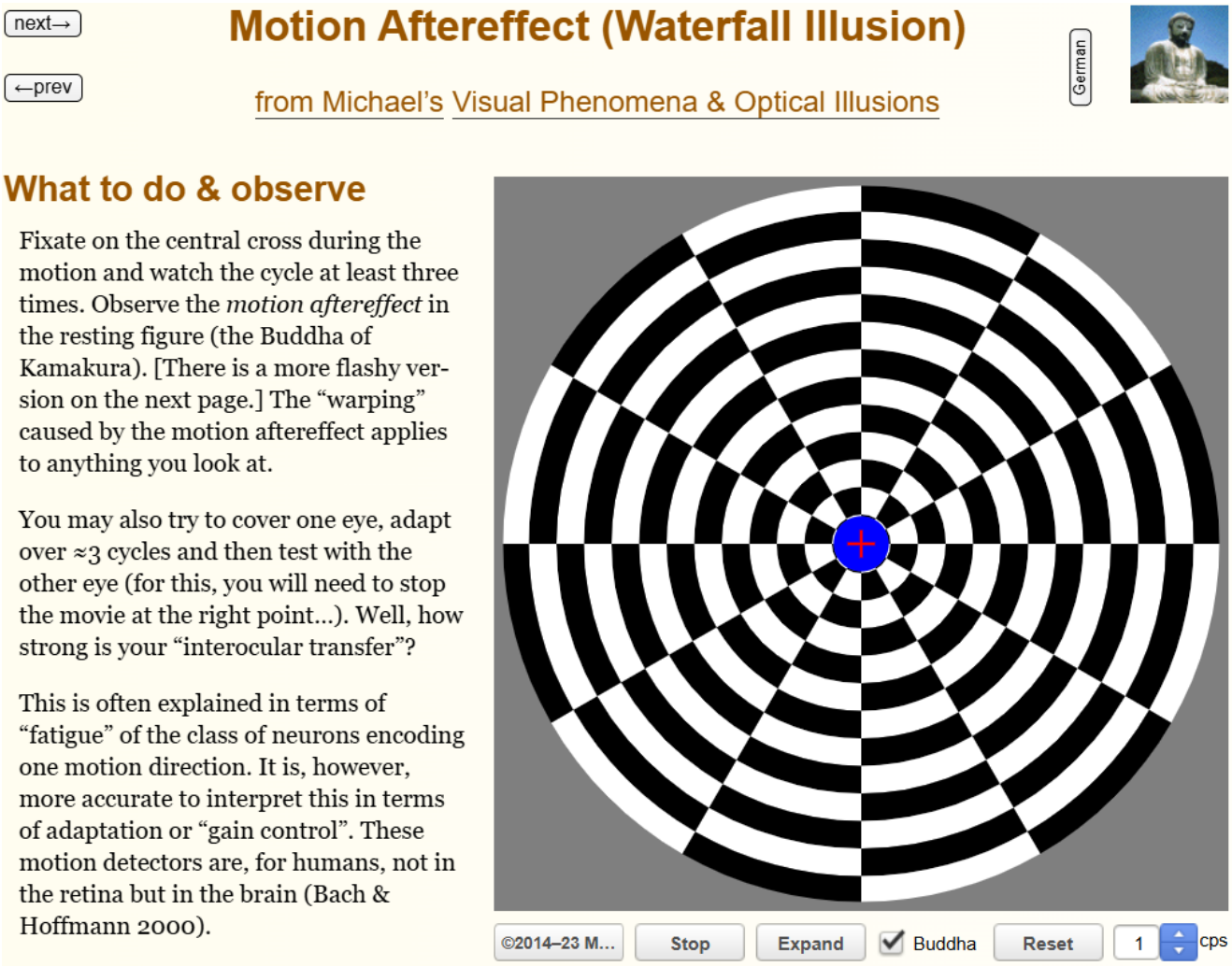

Рисунок 1.1 – Приклад роботи вебзастосунку «Visual Phenomena & Optical Illusions»

Ще одним прикладом сучасної цифрової платформи, що спеціалізується на вивченні зорового сприйняття, є «The Illusion Index» [9], система, розроблена дослідниками з Університету Ексетера. Цей онлайн ресурс надає доступ до великої бази даних оптичних ілюзій, кожна з яких супроводжується науковими поясненнями, історією відкриття, а також посиланнями на відповідні академічні публікації. Відмінною особливістю платформи є її структурований та зручний інтерфейс, який дозволяє швидко знаходити потрібні ілюзії за їх

характеристиками або механізмами впливу. Користувачі можуть фільтрувати ілюзії за такими категоріями, як сприйняття руху, контрастність, яскравість, глибина або контекст, що значно полегшує навігацію та тематичне вивчення матеріалу. Для кожного візуального прикладу надається високоякісна графіка та детальне пояснення того, як ця ілюзія впливає на когнітивне сприйняття людини. Завдяки такому науково-орієнтованому підходу, платформа стала широко використовуватися викладачами, студентами, психологами, когнітивістами та нейробіологами, які використовують її в освітніх програмах, дослідницькій діяльності або для ілюстрації впливу зорових спотворень (див. рисунок 1.2).

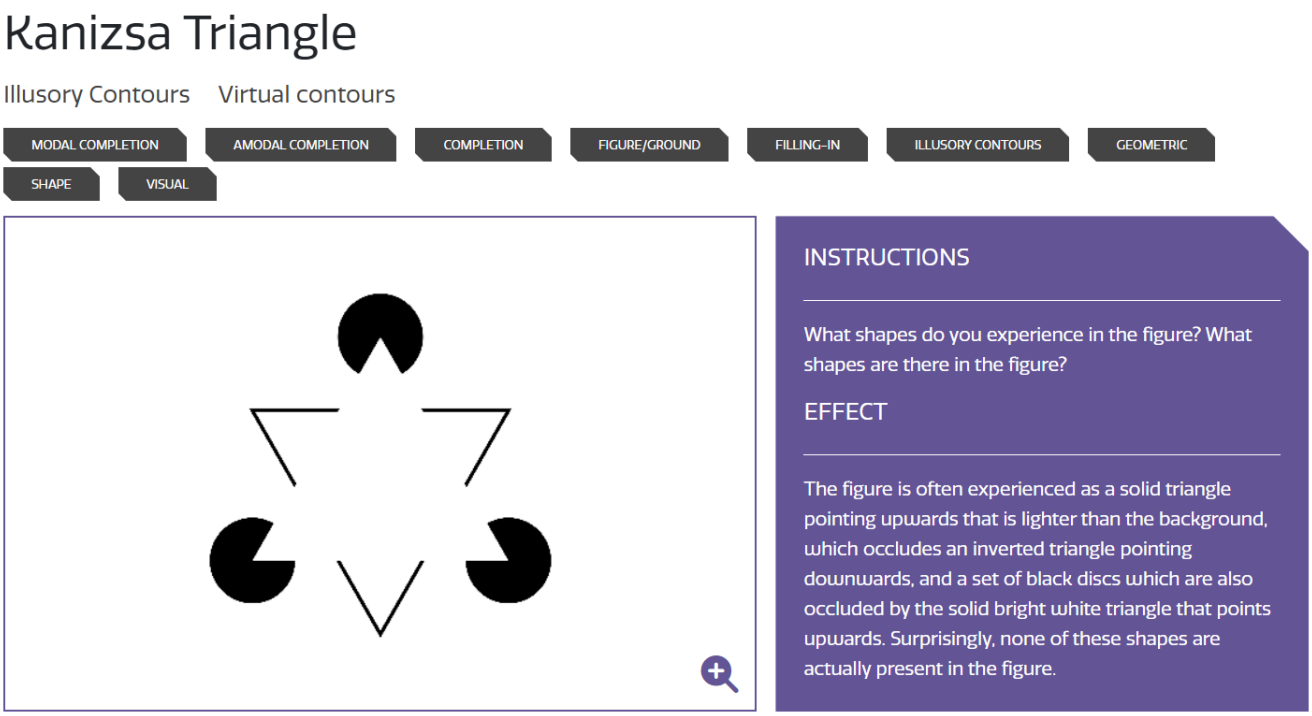

Рисунок 1.2 – Приклад роботи вебзастосунку «The Illusion Index»

Одним із найяскравіших прикладів цифрового застосунку, який, хоча й не був спеціально розроблений для вивчення оптичних ілюзій, широко використовується для їх візуального моделювання, є «Silk – Interactive Generative Art» [10]. Цей браузерний застосунок дозволяє користувачам створювати інтерактивні зображення, використовуючи симетричні візерунки, плавні лінії, ефекти світіння та переходи кольорів. Завдяки простому та інтуїтивно зрозумілому інтерфейсу навіть користувач без художнього досвіду може

створювати складні композиції, що викликають ефекти ілюзорного руху, просторової глибини або спотворення форми – ознаки типових когнітивних ілюзій. Хоча застосунок не надає наукових пояснень створеним ефектам, він є потужним інструментом для візуальних експериментів, де художнє вираження природно поєднується з явищами оптичного обману. Завдяки миттєвій візуалізації результатів та можливості експериментувати з кольором, напрямком симетрії та рівнем деталізації, Silk здобув велику популярність серед цифрових художників, дизайнерів та ентузіастів візуального мистецтва (див. рисунок 1.3).

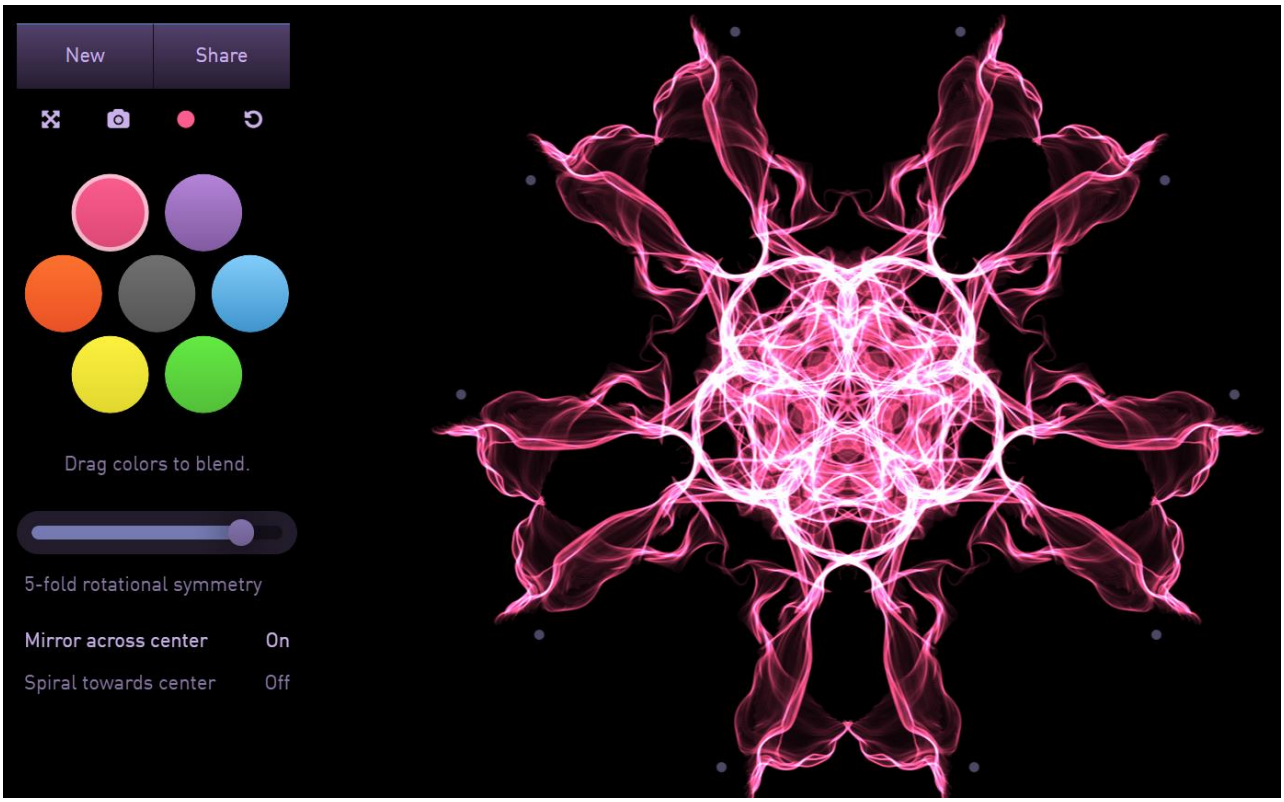

Рисунок 1.3 – Приклад роботи вебзастосунку «Silk – Interactive Generative Art»

Розглянуті програмні засоби ілюструють потенціал цифрових технологій у сфері візуалізації оптичних ілюзій, поєднуючи у собі наукову обґрунтованість, зручний інтерфейс, інтерактивність та високу якість графічного представлення. Кожен із аналізованих застосунків має свою унікальну концепцію. Подібні застосунки створюються на перетині декількох дисциплін, тож їх розробка потребує не лише глибоких знань у галузі психофізіології зору та когнітивної

психології, але й серйозної технічної підготовки у сферах комп'ютерної графіки, анімації, обробки зображень і розробки дружнього до користувача інтерфейсу. Проведений комплексний аналіз наявних програмних продуктів дозволив виявити не лише їхні сильні сторони, а й наявні обмеження. Усе це стало основою для формування функціональних вимог до власного програмного забезпечення, яке покликане об'єднати переваги наукового підходу, гнучкість творчих напрацювань і простоту у використанні. Для зручності візуального порівняння можливостей розглянутих аналогів із функціоналом власної розробки було складено таблицю 1.1, що відображає ключові характеристики та дозволяє оцінити напрямки вдосконалення.

Таблиця 1.1 – Порівняльна характеристика аналогів

Функціональні характеристики	Visual Phenomena & Optical Illusions	The Illusion Index	Silk – Interactive Generative Art	Власний застосунок
Генерація динамічних оптичних ілюзій	1	0	1	1
Можливість налаштування параметрів ілюзій (колір, форма, рух)	0	1	1	1
Збереження створених ілюзій у файл (PNG, JPEG)	0	0	0	1
Вбудоване пояснення принципу дії ілюзії	1	1	0	1
Підтримка як статичних, так і динамічних ілюзій в одному застосунку	1	0	1	1
Сумарний коефіцієнт	3	2	3	5

Сумарний коефіцієнт, отриманий за результатами порівняння для власної розробки, становить 5, що є найвищим значенням серед усіх проаналізованих

додатків та свідчить про її високий рівень універсальності, гнучкості та інноваційності. Цей показник демонструє не лише конкурентоспроможність програмного продукту, але й підтверджує доцільність обраного підходу до реалізації функціональності, орієнтованого на наукове, освітнє та творче використання.

Таким чином, створення власного додатку набуває не лише теоретичного, а й практичного значення, оскільки його можна використовувати як ефективний інструмент для вивчення механізмів зорового сприйняття, проведення навчальних демонстрацій, а також генерування нових візуальних ефектів у сферах цифрового мистецтва, дизайну, когнітивної психології та суміжних дисциплін. Інтеграція у професійну практику та освітні процеси сприяє формуванню міждисциплінарного мислення, підвищує інтерес до вивчення візуальних явищ, підтримує популяризацію когнітивної науки в сучасному інтерактивному форматі.

1.3 Аналіз методів розв’язання задачі

Один з основних і водночас найпростіших підходів до створення оптичних ілюзій – це використання готових растрових зображень, які вже містять візуальні ефекти. Цей метод особливо поширений у мобільних або вебдодатках, оскільки не вимагає складної реалізації графічного двгуна або значних обчислювальних ресурсів. Однак така стратегія має суттєві обмеження – відсутність можливості змінювати параметри ілюзії в режимі реального часу, неможливість масштабування зображення без втрати якості, а також відсутність інтерактивності, що є важливою умовою для сучасних користувацьких додатків. У зв'язку з цим такий підхід непридатний для створення гнучкої системи генерації ілюзій, орієнтованої на експериментування та навчання.

Більш ефективним рішенням є використання векторної графіки в поєднанні з анімацією. Такий підхід дозволяє реалізувати широкий спектр ефектів, включаючи ілюзії руху, зміни форми, мерехтіння, деформації простору

та перспективи. Головною перевагою є можливість гнучкого налаштування параметрів, що забезпечує високий рівень інтерактивності. Завдяки своїй векторній природі зображення можна легко масштабувати без втрати якості, а також адаптуватися до різних розмірів екрана. Однак, створення складних композицій з високим рівнем деталізації вимагає знань у галузі комп'ютерної графіки, а також певних оптимізацій для забезпечення безперебійної роботи програми.

Ще одним перспективним методом є генерація ілюзій за допомогою математичних функцій. Цей підхід базується на побудові візуальних ефектів на основі тригонометричних формул, фрактальних структур або періодичних сіток, таких як муарові візерунки. Його головною перевагою є висока гнучкість та можливість автоматичного створення складних, нестандартних візуальних зображень. Однак, цей метод вимагає значних обчислювальних ресурсів, особливо в поєднанні з динамічною анімацією, а також ускладнюється з кожною додатковою функцією, яка розширює налаштування ілюзії.

Враховуючи переваги та недоліки кожного з розглянутих підходів, найефективнішим рішенням для реалізації завдання є поєднання векторного графічного підходу з алгоритмічною генерацією візуальних ефектів на основі математичних функцій. Такий гібридний підхід забезпечує баланс між гнучкістю, точністю та високим рівнем інтерактивності. Він дозволяє створювати динамічні сцени, якими користувач може керувати в режимі реального часу через графічний інтерфейс, змінюючи параметри ілюзії, масштаб, колірну палітру або швидкість анімації. Водночас, математична основа відкриває можливості для автоматичної генерації нових ілюзій, що значно розширює потенціал програми та її подальшої модифікації. Таким чином, обраний підхід дозволяє реалізувати сучасний, розширюваний програмний інструмент для генерації оптичних ілюзій, який відповідає вимогам інтерактивності, масштабованості та наукової коректності, а також забезпечує можливість додавання нових типів ілюзій у майбутньому без необхідності радикального перероблення архітектури програми.

1.4 Постановка задач

Після ретельного аналізу існуючих програмних аналогів для генерації оптичних ілюзій, їх функціональності, переваг та обмежень, а також оцінки доцільності створення власного застосунку, було сформовано перелік основних завдань, необхідних для реалізації програмного продукту, який відповідатиме сучасним вимогам до інтерактивності, наукової точності та зручності використання.

Перш за все, необхідно визначити ключові вимоги до алгоритмів генерації оптичних ілюзій, враховуючи особливості сприйняття як статичних, так і динамічних візуальних ефектів. Це дозволить створити коректну математичну та графічну основу для моделювання різних типів ілюзій.

Наступний етап забезпечує розробку алгоритмів побудови ілюзій різної природи – геометричних, кольорових, а також тих, що базуються на русі або зміні структури зображення з часом. Для цього планується використовувати математичні функції, векторну графіку та методи комп'ютерного моделювання, що забезпечить гнучкість та масштабованість генерації.

Програмна реалізація проекту буде здійснюватися з використанням мови програмування Java, зокрема, з використанням бібліотеки JavaFX, яка надає потужні інструменти для створення графічного інтерфейсу, анімації та обробки подій, необхідних для динамічного налаштування параметрів ілюзій у режимі реального часу.

Також планується реалізувати функцію експорту створених ілюзій у популярні графічні формати, зокрема PNG та JPEG, що дозволить користувачам зберігати отримані результати для подальшого використання, аналізу або публікації.

Особлива увага буде приділена тестуванню програмного забезпечення з метою перевірки його функціональності, надійності та відповідності заданим технічним та користувацьким вимогам. Комплексне тестування охоплюватиме різні сценарії використання, налаштування параметрів, продуктивність

візуалізації та сумісність з різними платформами.

Реалізація цих завдань дозволить створити повноцінний програмний інструмент для генерації оптичних ілюзій, який поєднуватиме дослідницький потенціал з практичною цінністю в освітньому процесі, візуальному дизайні, когнітивних експериментах та цифровому мистецтві. Крім того, гнучка архітектура розробленого застосунку забезпечить можливість подальшого розширення його функціональності, адаптації до нових типів ілюзій та розширення сценаріїв використання без необхідності суттєвого перегляду структури програми.

1.5 Висновки

У першому розділі було розглянуто сучасний стан програмних засобів, призначених для генерації оптичних ілюзій, а також аналіз основних тенденцій їх розвитку. Огляд таких програмних забезпечень, як Optical Illusions & Visual Phenomena, The Illusion Index та Silk – Interactive Generative Art, дозволив нам визначити їхні характерні риси, ключові переваги та обмеження, які слід враховувати під час розробки власного програмного продукту.

У процесі аналізу методів реалізації було розглянуто кілька підходів до побудови оптичних ілюзій На основі порівняння ефективності та можливостей цих підходів було обґрунтовано доцільність використання комбінованої стратегії, яка поєднує гнучкість та точність математичного моделювання з можливостями сучасної графічної візуалізації.

В результаті аналізу було сформульовано перелік ключових завдань, які необхідно вирішити в рамках курсового проекту.

Таким чином, перший розділ слугує основою для подальшого впровадження програмного засобу, обґрунтування його актуальності, визначення технічних рекомендацій та закладення основи для розробки функціональних компонентів, які забезпечать високоякісну генерацію оптичних ілюзій у цифровому середовищі.

2 РОЗРОБКА МОДУЛЬНОЇ АРХІТЕКТУРИ СИСТЕМИ

2.1 Аналіз вхідних даних системи

Для реалізації застосунку генерації оптичних ілюзій було обрано мову програмування Java [11] у поєднанні з платформою JavaFX [12], середовищем розробки IntelliJ IDEA [13], інструментом управління проектами Maven [14] та бібліотекою JavaFX Animation API. Кожен з обраних інструментів та технологій має своє обґрунтоване призначення та переваги порівняно з альтернативами, що пояснює їх вибір у рамках розробки застосунку.

Вибір мови програмування Java зумовлений її широким використанням, високим рівнем стабільності, зручністю для розробки об'єктно-орієнтованих систем та підтримкою численних бібліотек та фреймворків. Завдяки кросплатформеності за рахунок використання віртуальної машини JVM, додаток, написаний на Java, може бути легко розгорнутий на різних операційних системах без необхідності модифікації коду. Крім того, Java має довгу історію успішного використання в освітніх, комерційних та дослідницьких додатках, що підтверджує її надійність та зрілість як технології для створення складних інтерактивних програм.

Для побудови інтерфейсу користувача було обрано JavaFX, яка стала найкращою альтернативою бібліотекам AWT та Swing. JavaFX пропонує сучасну архітектуру графічного інтерфейсу з підтримкою стилізації через CSS, використання FXML для розділення логіки та розмітки, вбудовані інструменти анімації, а також адаптивну графіку, що значно полегшує реалізацію складних візуальних ефектів. На відміну від AWT та Swing, JavaFX дозволяє створювати гнучкі, масштабовані та привабливі інтерфейси із сучасним дизайном та високою продуктивністю, що критично важливо при генерації динамічних ілюзій.

Середовищем розробки було обрано IntelliJ IDEA. Це IDE забезпечує високу швидкість, має інтуїтивно зрозумілий інтерфейс, підтримку розширень, інтелектуальне автодоповнення, навігацію коду, вбудований клієнт Git та тісну

інтеграцію з Maven, JavaFX та багатьма іншими технологіями. Порівняно з Eclipse або NetBeans, IntelliJ IDEA пропонує кращу підтримку сучасних фреймворків, вимагає менше налаштування та є більш стабільним при роботі з великими проектами. Він також має вбудовані інструменти для зручної роботи з FXML, Scene Builder та налагодження JavaFX-додатків.

Maven було обрано для управління залежностями та автоматизації збірки, оскільки це стандартна та добре документована система збірки в екосистемі Java. Maven має просту декларативну структуру на основі XML, яка дозволяє легко визначати конфігурацію проекту, залежності, плагіни та фази життєвого циклу. Хоча Gradle також є сучасною альтернативою з більш гнучкими можливостями налаштування, для малого або середнього проекту з типовими потребами Maven виявився оптимальним рішенням завдяки своїй простоті, великій спільноті, наявності прикладів та високій сумісності з більшістю бібліотек Java, включаючи JavaFX.

Розробка програмного продукту забезпечує створення інтерактивного графічного інтерфейсу, де користувач може змінювати параметри оптичних ілюзій у режимі реального часу, запускати та зупиняти анімацію, а також зберігати створені зображення у поширених графічних форматах. Основною платформою розробки є IntelliJ IDEA, яка забезпечує зручне управління проектами, інтеграцію з JavaFX, підтримку Maven та функціональні інструменти для налагодження коду. Для візуалізації ілюзій використовується Canvas API [15] з JavaFX – інструмент для створення векторної графіки, який дозволяє динамічно малювати графічні елементи, керувати їхньою поведінкою та забезпечувати високу точність і швидкість візуального відтворення.

Особливу роль у створенні динамічних ефектів відіграє бібліотека JavaFX Animation API [16], яка надає гнучкі можливості керування анімацією, включаючи використання таймерів, інтерполяцію значень, обробку циклів відтворення та зупинок. Це дозволяє реалізувати ключові компоненти ілюзій, зокрема, ефекти плавного руху, обертання, зміни розміру та кольорів, які є основою багатьох когнітивних ефектів. Програмний інструмент буде

реалізовано як настільний додаток з кросплатформною підтримкою – Windows, Linux та macOS.

Таким чином, обраний набір технологій та інструментів розробки дозволить створити ефективну, гнучку та інтуїтивно зрозумілу систему для моделювання, візуалізації та дослідження оптичних ілюзій. Застосунок забезпечить повний функціонал для створення як статичних, так і динамічних візуальних ефектів, а також дозволить користувачам взаємодіяти з елементами в режимі реального часу, змінюючи параметри ілюзій за допомогою зручного інтерфейсу. Процес розробки охоплює цілий спектр завдань: побудову графічного інтерфейсу, обробку дій користувача, реалізацію логіки генерації ілюзій, підтримку анімації та можливість експорту результатів. Всі компоненти системи будуть реалізовані з урахуванням вимог до продуктивності, адаптивності та зручності використання. Тому створення цього програмного засобу є складним, але цілісним завданням, що поєднує методи графічного програмування, алгоритмічного моделювання та сучасного дизайну інтерфейсів.

2.2 Розробка методу створення структури системи

Під час розробки створення структури програмної системи основна увага приділялася виокремленню логічно завершених підсистем та блоків відповідно до функціональних вимог та реального коду, сформованого в процесі створення застосунку. Одним із ключових компонентів була графічна підсистема, яка реалізована за допомогою абстрактного класу OpticalIllusion та низки його конкретних реалізацій таких як HeringIllusion, ZollnerIllusion тощо. Ця підсистема визначає основу на якій базується створення всіх ілюзій, включаючи методи draw, getSettingsPane, resetParameters та getHelpText, що забезпечує модульність та розширюваність системи.

На основі цього класу сформовано блок генерації статичних зображень, що охоплює всі класи, що реалізують нерухомі візуальні ілюзії. Вони генерують зображення без використання анімації та без необхідності часової мінливості. У

класах ZollnerIllusion або CafeWallIllusion метод draw() малює відповідні шаблони без циклів анімації, а лише один статичний результат на основі параметрів користувача.

Для анімованих ілюзій створено окремий блок генерації динамічних зображень, який реалізує ілюзії з рухом або зміною часу, такі як PhiPhenomenonIllusion, ChoppyRotationIllusion або MotionBindingIllusion. Ці класи використовують таймери AnimationTimer або Timeline для безперервного оновлення Canvas, де елементи перераховуються та перемальовуються відповідно до частоти кадрів. Таким чином, динамічні ілюзії – це окремі об'єкти, які взаємодіють з графічним полотном у режимі реального часу.

Виділено окремий блок графічного інтерфейсу користувача, реалізований у класі MainController та пов'язаний з FXML-файлом. Цей блок містить усі елементи керування ComboBox, Button, ColorPicker, TextArea та забезпечує зв'язок між користувачем та графічною підсистемою. Саме тут реалізовано вибір типу ілюзії, застосування параметрів, перемикання теми, відображення довідки та експорт результату в зображення.

Ще однією важливою частиною є система визначення параметрів ілюзії, яка реалізована через індивідуальні налаштування для кожного класу ілюзій. Кожен з них створює власний VBox з пов'язаними з ним елементами інтерфейсу Slider, ColorPicker , які динамічно відображаються у правій частині вікна. Через обробники подій будь-яка зміна параметра автоматично викликає метод fireChange() та оновлює зображення, що гарантує інтерактивність та точне налаштування ефектів.

Для збереження створених зображень, зокрема у форматах PNG та JPEG, реалізовано блок серіалізації. У методі onExport класу MainController, використовуючи клас FileChooser, користувач вибирає місце та формат для збереження, після чого програма створює окреме полотно, на якому малюється лише ілюзія без фону для PNG, або з користувацьким фоном для JPEG, після чого зображення зберігається через ImageIO.write. У випадку PNG повна прозорість фону забезпечується завдяки параметрам SnapshotParameters.

У процесі розробки програмного застосунку для генерації оптичних ілюзій було створено діаграму діяльності UML [17], яка демонструє послідовність дій користувача від моменту запуску застосунку до завершення сеансу з ілюзіями. Ця діаграма дозволяє структурувати логіку програми, визначити ключові блоки керування, виявити можливі ситуації, пов'язані з вибором або відсутністю вибору, та надати цілісне уявлення про цикл програми (див. рисунок 2.1).

Згідно з побудованою діаграмою, взаємодія користувача з застосунком починається з ініціалізації програми. Після завершення процесу ініціалізації автоматично відображається головне вікно з елементами керування, що включає випадаючі списки для вибору типу ілюзії, а також кнопки для очищення, експорту, скидання параметрів та виклику довідки.

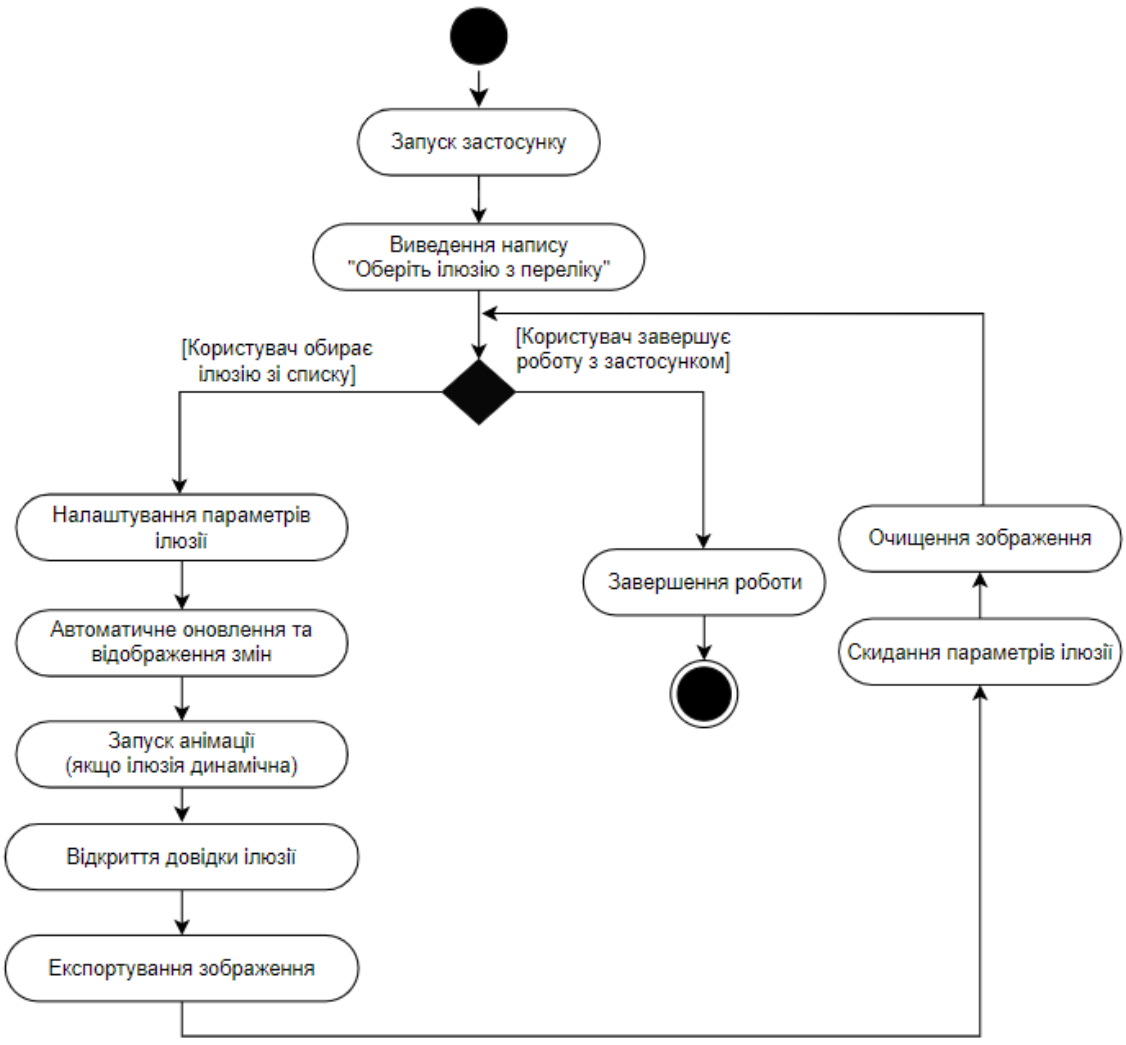

Рисунок 2.1 – Діаграма діяльності програмного застосунку

Усі параметри оновлюються в режимі реального часу завдяки інтеграції з платформою JavaFX та використанню спеціалізованого механізму циклу анімації – AnimationTimer, який дозволяє безперервно перераховувати значення та відображати їх на екрані. Завдяки цьому зміни миттєво відображаються на полотні без необхідності додаткового підтвердження або перезавантаження сцени, що значно підвищує динамічність та інтерактивність роботи. У випадку динамічних ілюзій користувач отримує доступ до елементів керування, які дозволяють запускати, зупиняти або змінювати анімацію.

Крім того, у програмі реалізована можливість відкриття довідки для кожної ілюзії, яка містить коротке пояснення принципу її роботи та історичного контексту. Після завершення роботи з ілюзією користувач може експортувати результат у графічний формат PNG або JPEG, що дозволяє зберегти зображення для подальшого використання. У будь-який час користувач також може очистити полотно та скинути параметри вибраної ілюзії до початкового стану.

2.3 Розробка алгоритмів роботи системи

Для реалізації взаємодії користувача з програмним застосунком для генерації оптичних ілюзій було створено блок-схему, яка відображає логіку роботи системи від початку до кінця сеансу (див. рисунок 2.2). Алгоритм починається із запуску програми та завантаження графічного середовища, після чого система очікує дії користувача – вибору однієї з доступних ілюзій. Якщо вибір не зроблено, жодних дій не відбувається, програма залишається в стані очікування. Після вибору ілюзії коригуються її параметри, специфічні для обраного типу. Якщо ілюзія динамічна, система автоматично активує цикл анімації.

Подальший хід подій залежить від дій користувача. Він може змінювати параметри, адаптуючи ілюзію до власних потреб: змінювати кольори, кількість елементів, розміри або швидкість. Будь-яке оновлення значень миттєво відображається на графічному полотні без необхідності перезавантаження сцени.

Якщо потрібна довідкова інформація, користувач може скористатися кнопкою виклику пояснення – відкриється вікно з коротким описом та принципом роботи ілюзії. Якщо користувач хоче зберегти зображення, він натискає кнопку «Експорт», після чого результат зберігається у вигляді графічного файлу.

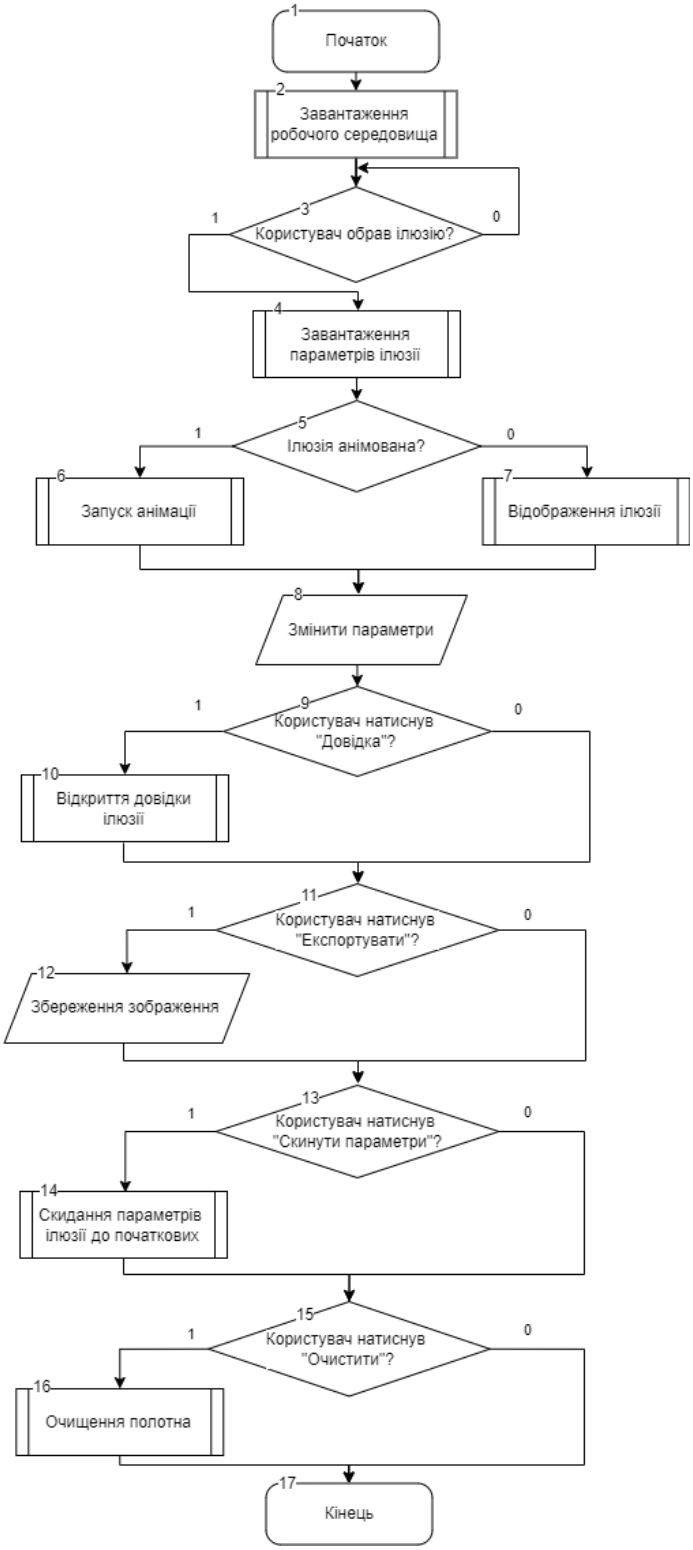

Рисунок 2.2 – Блок-схема роботи програмного застосунку

Також є допоміжні функції – скидання параметрів до початкових значень та очищення полотна. Перший варіант дозволяє швидко повернутися до стандартного вигляду ілюзії, другий – повністю видалити зображення з екрану. Всі дії реалізовані як окремі логічні гілки, що дозволяє підтримувати чисту архітектуру програми та уникати конфліктів між взаємодіями. Після виконання основних дій користувач може або завершити роботу, або повернутися до вибору нової ілюзії – таким чином, цикл взаємодії легко повторюється без необхідності перезапуску програми.

Однією з основних функцій, що визначає поведінку інтерфейсу в програмі для генерації оптичних ілюзій, є метод drawPreview(), який відповідає за побудову зображення на полотні відповідно до вибору користувача. Цей метод викликається щоразу, коли користувач змінює значення у списку доступних ілюзій, та виконує кілька ключових дій: очищення полотна, вибір відповідної ілюзії, виклик її методу малювання, оновлення опису та очищення або оновлення панелі опцій. Для чіткого розуміння роботи методу була побудована блок-схема (див. рисунок 2.3), яка демонструє покрокову логіку виконання коду та обробки умовних ситуацій.

Алгоритм починається з ініціалізації графічних елементів, що дозволяє працювати безпосередньо з полотном через методи JavaFX. Далі зчитується поточне значення зі списку, що випадає — тобто назва ілюзії, яку вибрав користувач. Якщо користувач не зробив вибір (значення null або порожнє), програма очищає полотно, встановлює колір фону на чорний та відображає текстову підказку «Виберіть ілюзію зі списку» в центрі полотна. Одночасно панель параметрів очищається, щоб уникнути відображення несумісних налаштувань. Це гарантує, що інтерфейс завжди залишатиметься чітким, навіть якщо вибір ще не зроблено.

Якщо користувач дійсно вибрав ілюзію, програма переходить до пошуку відповідного об'єкта в колекції ілюзій, яка містить реалізації всіх доступних ілюзій. Якщо знайдено відповідну ілюзію, програма готує полотно до

відображення: зчитується колір фону (або встановлюється прозорий фон, якщо активовано відповідний режим експорту), полотно очищається, і для вибраної ілюзії викликається метод draw() з усіма необхідними параметрами.

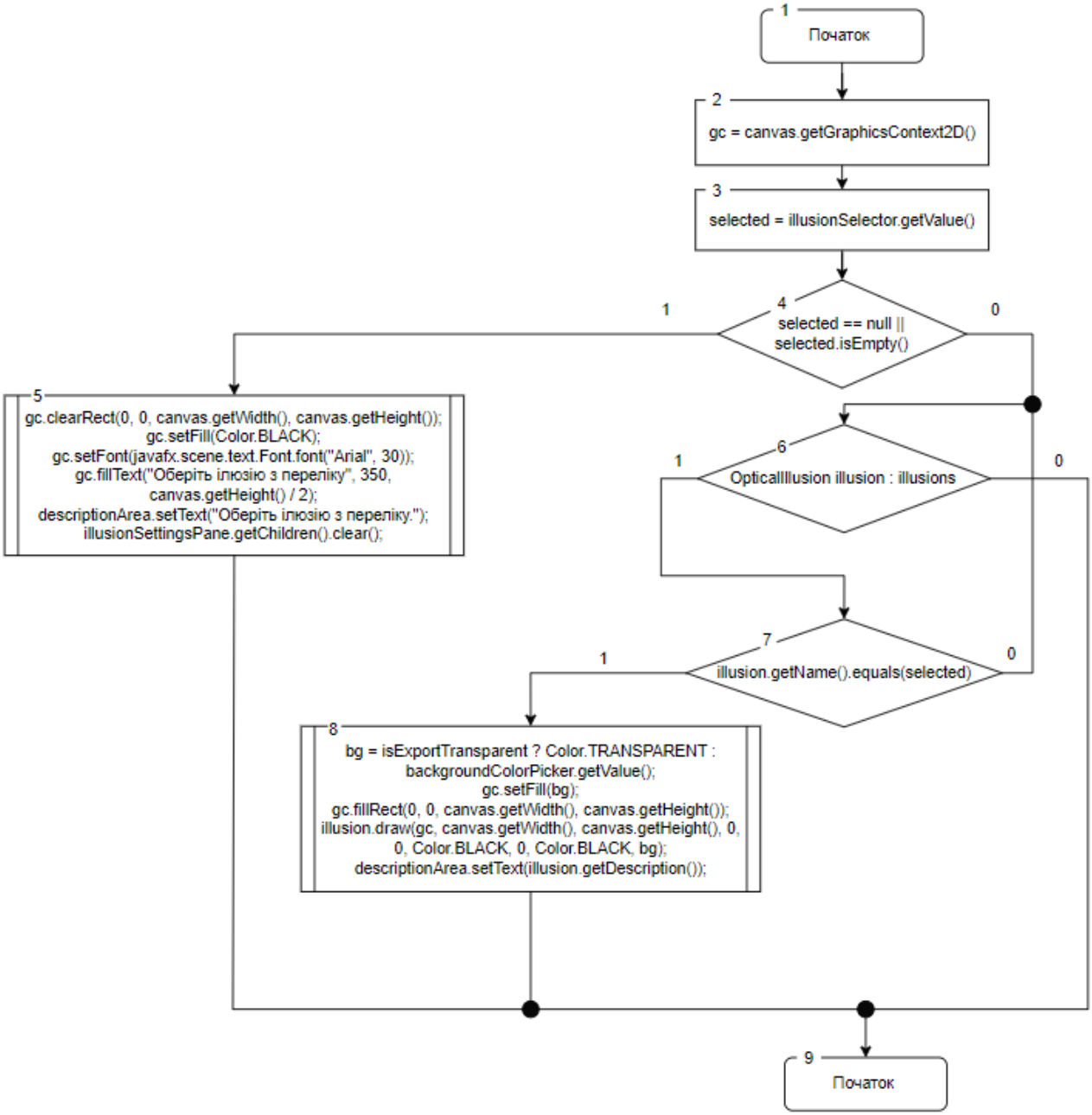

Рисунок 2.3 – Блок-схема роботи методу drawPreview()

Якщо жодна ілюзія не відповідає вибраному значенню, програма не переходить у стан помилки, а просто очищає полотно та заповнює його кольором фону, не викликаючи жодних методів малювання. Таким чином, навіть за відсутності збігів, інтерфейс поводиться передбачувано та стабільно.

Таким чином, метод drawPreview() відіграє центральну роль у механізмі оновлення візуального інтерфейсу, забезпечуючи зв'язок між вибором користувача та відображенням результату. Блок-схема, що ілюструє цей процес, дозволяє легко простежити всі можливі шляхи виконання та дає чітке уявлення про структуру обробки подій, пов'язаних з побудовою зображення на полотні.

2.4 Висновки

У другому розділі було проведено поглиблений аналіз архітектури програмного застосунку для генерації оптичних ілюзій, розглянуто принципи його функціонування та реалізації ключових алгоритмів. Детально проаналізовано логіку роботи методу drawPreview(), який відповідає за побудову зображення на канвасі відповідно до вибору користувача, а також забезпечує оновлення опису та налаштувань поточної ілюзії. Крім того, для кращого розуміння послідовності виконання дій у застосунку було створено низку UML-діаграм, що відображають логіку переходів, умовні розгалуження, реакції на події та взаємодію між компонентами графічного інтерфейсу користувача.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ТА МЕТОДІВ РЕАЛІЗАЦІЇ ПРОГРАМИ ДЛЯ ГЕНЕРАЦІЇ ОПТИЧНИХ ІЛЮЗІЙ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Під час створення програмного застосунку для генерації оптичних ілюзій особлива увага приділялася вибору технологічного стеку, який би забезпечував зручність реалізації графічного інтерфейсу, ефективну побудову візуальних ефектів та легкість подальшого розширення функціональності. З урахуванням цих вимог, основною мовою програмування була обрана Java, а для розробки графічної частини – платформа JavaFX. Java – це потужна об'єктно-орієнтована мова з кросплатформною підтримкою, яка дозволяє створювати стабільні настільні додатки з широкими можливостями налаштування, масштабованості та інтеграції. JavaFX, у свою чергу, є офіційним фреймворком для побудови графічного інтерфейсу в середовищі Java, який забезпечує не тільки стандартні елементи керування, але й підтримку 2D-графіки, анімації та ефектів, що дуже важливо при реалізації візуальних ілюзій.

Особливістю JavaFX є можливість створення інтерфейсів за допомогою FXML – розмітки, яка відокремлює логіку від візуального представлення, та CSS – для гнучкого стилістичного оформлення. Для спрощення роботи з FXML-файлами було використано SceneBuilder – зручний графічний редактор, який дозволяє створювати макети вікон шляхом візуального розміщення компонентів, без необхідності ручного написання XML-коду [18]. Це значно пришвидшило процес проектування інтерфейсу, забезпечило точне позиціонування елементів та швидке внесення змін до структури програми на будь-якому етапі розробки.

Розробка відбувалася в IntelliJ IDEA – одному з найпотужніших інтегрованих середовищ розробки для Java. IntelliJ забезпечує повну підтримку JavaFX, зручну навігацію по проектах, автоматизовані інструменти перевірки коду, а також інструменти для налагодження та профілювання. Крім того, IntelliJ

IDEA підтримує інтеграцію з Maven – інструментом, який був обраний для управління залежностями та автоматизації процесу збірки програми. Maven дозволяє централізовано підключати сторонні бібліотеки, включаючи JavaFX SDK, структурувати код відповідно до стандартів та забезпечувати відтворюваність збірки на різних платформах.

Таким чином, поєднання Java, JavaFX, SceneBuilder, IntelliJ IDEA та Maven сформувало ефективне середовище розробки, яке дозволяє не тільки створювати графічно насичений та продуктивний додаток, але й забезпечувати його подальшу підтримку, розширюваність та адаптацію до нових завдань. Обрані інструменти є взаємосумісними, мають активну підтримку спільноти, регулярні оновлення та великий обсяг документації, що зробило їх ідеальним вибором для реалізації проекту візуалізації оптичних ілюзій.

3.2 Розробка методу генерації оптичних ілюзій

Для розробки методу генерації оптичних ілюзій реалізацію застосунку було розбито на окремі частини. Основним структурним елементом є клас MainController, який виконує роль центрального модуля керування всім графічним інтерфейсом. Саме цей клас відповідає за організацію інтерфейсу користувача, координацію взаємодії між різними компонентами програми та реалізацію логіки, пов'язаної з побудовою ілюзій, реагуванням на події та обробкою параметрів. Усі ключові елементи інтерфейсу об'єднані в графічне полотно для малювання, елементи керування вибором ілюзій, панель налаштувань, кнопки очищення та експорту, а також поле для відображення опису вибраного ефекту. Клас безпосередньо обробляє дії користувача, оновлюючи зображення з кожним вибором або зміною параметрів, ініціюючи перемалювання полотна та оновлюючи опис.

MainController реалізує логіку відображення зображення на основі даних, введених користувачем. При цьому враховується як колір фону, вибраний з палітри, так і конкретні параметри, встановлені для кожної конкретної ілюзії —

наприклад, кількість елементів, колірні схеми або швидкість анімації. Клас реалізує набір методів, що охоплюють повний цикл роботи інтерфейсу Крім того, MainController забезпечує інтеграцію з внутрішніми класами, що представляють окремі ілюзії, викликаючи методи їх малювання та обробки параметрів. Завдяки цьому він виступає єдиним координаційним центром, через який відбуваються всі ключові взаємодії між інтерфейсом та логікою побудови оптичних ефектів.

Також одним із ключових модулів програмного застосунку для створення оптичних ілюзій, є абстрактний клас OpticalIllusion. Цей клас виступає базовим шаблоном для всіх реалізацій конкретних ілюзій та визначає загальний інтерфейс, якого повинні дотримуватися всі підкласи. Такий підхід дозволяє легко додавати нові ілюзії до застосунку, не змінюючи базову логіку контролера, реалізуючи принцип відкритості/закритості в об'єктно-орієнтованому програмуванні.

Для чіткої візуалізації структури класу та його зв'язків з підкласами була створена діаграма класів (див. рисунок 3.1). Вона відображає абстрактні методи, типи повернення, поля класу та очікувані реалізації в нащадках. Така діаграма відіграє важливу роль у плануванні системи, дозволяє швидко зрозуміти необхідні точки розширення та забезпечує єдиний підхід до побудови модулів ілюзій.

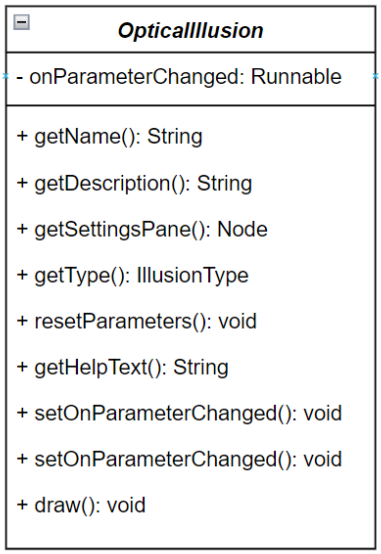

Рисунок 3.1 – Діаграма класу OpticalIllusion

Для реалізації методу генерації ілюзії Герінга було використано як візуальний, так і математичний підхід, заснований на особливостях зорового сприйняття перспективи людиною. Суть ілюзії полягає в тому, що дві вертикальні або майже вертикальні прямі лінії, розміщені перед фоном радіальних ліній (променів, що розходяться від центру), здаються вигнутими назовні, хоча насправді вони залишаються абсолютно прямими. Цей ефект виникає завдяки інтерпретації мозком просторової глибини: розбіжні лінії створюють враження перспективи, подібної до тунелю або тривимірного простору, і на цьому фоні вертикальні прямі лінії сприймаються як спотворені.

Математично фон цієї ілюзії реалізується як сукупність променів, що розходяться від центру полотна. Якщо позначити координати центру полотна як x_0 і y_0 то кожна лінія побудована від центру під певним кутом θ, який змінюється з фіксованим кроком. Кожен промінь має початкову точку в центрі та кінцеву точку, яка визначається тригонометричними функціями (3.1) і (3.2):

$$x=x_0+R\cdot\cos(\theta) \tag{3.1}$$

та

$$y=y_0+R\cdot\sin(\theta) \tag{3.2}$$

де R – довжина лінії (радіус до краю полотна), $\theta\in[0,2\pi]$ – кут повороту з кроком, залежним від заданої кількості променів N.

На тлі цих радіальних ліній розміщуються дві вертикальні прямі. Вони задаються як паралельні прямі на рівній відстані від центра полотна (3.3).

$$x=x_0\pm d,\ y\in[0,H] \tag{3.3}$$

де d – половина відстані між прямими, H – висота полотна.

У програмній реалізації цей метод представлений у класі HeringIllusion, який реалізує інтерфейс OpticalIllusion. Метод draw() є основним моментом побудови, в якому спочатку всі промені генеруються за допомогою циклу та

тригонометричних функцій, а потім накладаються дві вертикальні лінії. Через те, що вертикальні лінії накладаються на фон з променями, око інтерпретує їх кривизну через спотворене сприйняття просторової перспективи, хоча насправді кривих немає.

Ілюзія Cafe Wall яскравий приклад того, як взаємодія контрастних елементів та зміщення рядів можуть призвести до спотвореного сприйняття прямолінійних меж. На перший погляд, ця ілюзія складається з рядів чергуючихся чорних та білих прямокутників. Між ними розміщені вузькі лінії, і кожен наступний ряд трохи зміщений убік. В результаті горизонтальні розділові лінії здаються вигнутими або хвилястими, хоча насправді вони залишаються повністю прямими та паралельними. Цей ефект пов'язаний з контрастною взаємодією між світлими та темними ділянками та особливостями нейронної обробки меж у людському зорі.

Математично ілюзія реалізується шляхом побудови кількох рядів однакових плиток з чергуваним горизонтальним зміщенням. Нехай w – ширина плитки, h – її висота, s – горизонтальне зміщення кожного непарного ряду, а p – висота проміжної лінії між рядами. Координати початку кожної плитки в ряду можна визначити за формулою для парного ряду (3.4) і для непарного ряду зі зміщенням (3.5)

$$x = i \cdot w,\ y = r \cdot (h + p) \tag{3.4}$$

та

$$x = i \cdot w + \frac{s}{2},\ y = r \cdot (h + p) \tag{3.5}$$

де i — індекс плитки в ряду, r — номер ряду.

Кожна плитка чергується між чорним і білим кольором, а між рядами малюється горизонтальна лінія нейтрального кольору, яка підсилює ефект викривлення.

У програмному коді ця ілюзія реалізована в класі CafeWallIllusion, що

також наслідує інтерфейс OpticalIllusion. Метод draw(.) реалізує побудову всієї сітки плиток: за допомогою вкладених циклів проходиться по рядках та стовпцях, чергуючи кольори плиток та зміщуючи непарні рядки відповідно до параметра зміщення. Між кожними двома рядками малюється роздільна лінія заданої товщини, колір якої задається користувачем або встановлюється за замовчуванням.

Ілюзія Мюллера-Лайєра одна з найвідоміших оптичних ілюзій, яка демонструє вплив контекстного дизайну на сприйняття довжини лінії. Вона складається з двох горизонтальних ліній однакової довжини, до кінців яких додані короткі штрихи, що утворюють «стрілки», спрямовані або всередину, або назовні. Незважаючи на однакову фізичну довжину, лінія з «відкритими» кінцями сприймається довшою за лінію із «закритими» кінцями.

Математично ця ілюзія реалізується шляхом побудови двох відрізків однакової довжини L, до яких під певним кутом θ додаються допоміжні лінії, що утворюють стрілки. Координати кінців таких стрілок можна розрахувати, враховуючи довжину відрізків «стрілок» d та кут їх нахилу (3.6).

$$(x_2,y_2) = (x + d\cdot\cos(\theta),\ y \pm d\cdot\sin(\theta)) \qquad (3.6)$$

де x — координата кінця основної лінії, d — довжина штрихів, а знак ± залежить від напрямку стрілки.

У застосунку ця ілюзія реалізована в окремому класі MullerLyerIllusion, який реалізує інтерфейс OpticalIllusion. Метод draw() відповідає за графічну побудову всіх елементів. Спочатку малюються горизонтальні лінії однакової довжини, а потім стрілки з обох боків. Користувач може налаштувати параметри ілюзії, такі як довжина основної лінії, розмір стрілок, кут стрілок, колір ліній, а також кількість повторень та вертикальний інтервал між екземплярами ілюзії.

Спіральна ілюзія Фрейзера, також відома як «хибна спіраль», є яскравим прикладом того, як конфлікт між локальними та глобальними особливостями зображення може ввести в оману зорову систему. На перший погляд здається,

що глядачеві представлена спіраль, яка постійно закручується до центру. Насправді фігура складається з концентричних кіл, які залишаються замкнутими, але завдяки взаємодії з кутовими елементами виникає ілюзія спіралі.

Математично, основна структура ілюзії - це набір концентричних кіл з однаковим кроком r. Кожне коло розділене на сектори з фіксованою кількістю сегментів n. Для кожного сегмента будується елемент у вигляді дужки або сегмента, зміщеного у фазі на кут ϕ, який поступово збільшується від кола до кола, створюючи псевдоспіральну траєкторію.

Координати країв кожного сегмента обчислюються за допомогою полярної системи координат (3.7).

$$x = r \cdot \cos(\theta + i \cdot \phi), \quad y = r \cdot \sin(\theta + i \cdot \phi) \tag{3.7}$$

де r – радіус поточного кола, θ – початковий кут сегмента, i – індекс сегмента на колі, ϕ – зміщення фази між сусідніми сегментами.

У програмному застосунку ілюзія реалізована в класі FraserSpiralIllusion, який також реалізує інтерфейс OpticalIllusion. Метод draw() відповідає за побудову кількох кіл та накладання на них сегментів або штрихів з поступовим зсувом фази. Принцип роботи базується на зміні напрямку кожного сегмента всередині кільця, що викликає візуальний зсув і зрештою створює ефект спіралі.

Ілюзія RotatingCircles типовий приклад динамічної оптичної ілюзії, заснованої на конфлікті між реальним двовимірним зображенням та сприйняттям глядача, що створює враження просторової глибини та об'ємного обертання. Цей ефект досягається шляхом накладання кількох концентричних кіл, кожне з яких складається з кольорових секторів які чергуються, що обертаються в різних напрямках та з різною швидкістю. Через це зорове сприйняття зазнає сенсорного перевантаження, мозок намагається інтерпретувати рух як просторову зміну положення, і в результаті виникає гіпнотичний ефект.

Математично кожне кільце задається радіусом R_i, кількістю секторів n та

початковою фазою обертання $\alpha_i(t)$, яка змінюється з часом. Кожен сектор описується як частина кола з кутовим охопленням θ. Для забезпечення анімації кожен сектор обертається зі швидкістю ω_i, яка може бути позитивною або негативною (3.8).

$$\alpha_i(t)=\alpha_{i0} + \omega_i \cdot t \tag{3.8}$$

де α_{i0} – початковий кут обертання, ω_i – кутова швидкість обертання кільця i, t_i – час.

Координати центру кожного сектора обчислюються у полярній системі координат (3.9).

$$x(\theta) = R_i \cdot \cos(\theta + \alpha_i(t)), \quad y(\theta) = R_i \cdot \sin(\theta + \alpha_i(t)) \tag{3.9}$$

де (x,y) – координати центру сектора, R_i – радіус кільця, θ – кут сектора в межах кільця, $\alpha_i(t)$ – кут повороту кільця у часі.

Програмно ілюзія реалізована в класі RotatingCirclesIllusion, який успадковує абстрактний клас OpticalIllusion. Метод draw() відповідає за відображення початкового зображення, а метод startAnimation() реалізує безперервне оновлення кутів повороту для кожного кільця за допомогою AnimationTimer.

Феномен Фі – це класична динамічна ілюзія руху, при якій спостерігачеві створюється враження переміщення світлового об'єкта між двома або більше точками, хоча об'єкти фізично не рухаються. Цей ефект є основою кінематографа, де послідовна зміна статичних зображень створює враження безперервного руху.

Реалізація Фі-ефекту базується на поступовому миготінні кількох об'єктів, розташованих вздовж кола або прямої лінії. Спостерігач сприймає не зміну положень, а ілюзорний рух одного об'єкта між кількома положеннями.

Положення кожного з об'єктів N кіл, розташованих вздовж кола радіуса R

задається за допомогою полярних координат (3.10).

$$x_k = R \cdot \cos(\frac{2\pi k}{N}),\ y_k = R \cdot \sin(\frac{2\pi k}{N}) \quad (3.10)$$

де $k \in \{0,1,...,N-1\}$ – індекс кола, (x_k, y_k) – координати центру кола, R – радіус кола.

Кожне коло періодично "мигає" з певною частотою. Для створення ілюзії плавного руху між позиціями використовується ефект затухання хвоста, при якому попередні положення відображаються з меншою яскравістю або прозорістю. Таким чином формується візуальний слід.

Прозорість для кожного з кіл задається експоненційною функцією (3.11).

$$\alpha_k(t) = \exp(-\lambda \cdot (t - t_k)^2) \quad (3.11)$$

де t – поточний час, t_k – момент останньої появи кола k, λ – параметр, що визначає швидкість затухання.

Ілюзія реалізована у класі PhiPhenomenonIllusion, який є підкласом OpticalIllusion і використовує методи JavaFX-анімації. Метод draw() відповідає за початкову побудову точок, а метод startAnimation() за періодичне оновлення екрану, чергуючи активні та затухаючі об’єкти.

Цікавою динамічною ілюзією є зв'язування руху, візуальне явище, в якому взаємозалежний рух кількох об'єктів змінює наше сприйняття окремих рухів. Хоча кожен об'єкт рухається незалежно, їхній скоординований рух створює враження єдиної, пов'язаної структури. Найпоширенішим прикладом є мережа точок, кожна з яких рухається вертикально, але вся структура сприймається як рухома по діагоналі або по колу.

Основна ідея цієї ілюзії полягає в тому, щоб надати кожному об'єкту незалежний локальний рух, але зробити його синхронізованим у фазі або з невеликим зсувом, що створює глобальне враження спільного руху.

Математично положення кожної точки (x,y) у момент часу t описується формулою (3.12).

$$x_i(t) = x_{0i}, \qquad y_i(t) = y_{0i} + A \cdot \sin(2\pi ft + \phi_i) \tag{3.12}$$

де x_{0i}, y_{0i} – початкова позиція точки A – амплітуда вертикальних коливань, f – частота руху (Hz), ϕ_i – фазовий зсув для кожної точки.

Щоб створити ефект прив’язки руху, точкам призначаються однакові частота та амплітуда, а фазовий зсув ϕ_i визначає початок коливання. Коли $\phi_i = 0$ для всіх точок, рух є повністю синхронізованим. Але навіть незначні варіації фаз можуть порушити це сприйняття.

При синхронному русі з певним розташуванням точок спостерігач сприймає не вертикальну осциляцію, а спільний рух форми, хоча фізично цього немає.

У програмному застосунку ілюзія реалізується у класі MotionBindingIllusion, що розширює базовий клас OpticalIllusion. Основний метод draw() ініціалізує об’єкти, а в методі startAnimation() використовується AnimationTimer, який оновлює позицію точок на основі синусоїдальної функції з фіксованою частотою.

3.3 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача є невід'ємною та надзвичайно важливою складовою створення програмного застосунку для генерації оптичних ілюзій, оскільки саме інтерфейс визначає зручність, зрозумілість та ефективність взаємодії користувача з усіма функціями програми. Головною метою інтерфейсу є створення інтуїтивно зрозумілого середовища, в якому користувач може легко вибрати тип ілюзії, змінити її параметри, керувати відображенням анімації, зберегти результати роботи та отримати додаткову інформацію про ілюзію. Добре розроблений інтерфейс значно покращує якість взаємодії з програмою,

скорочує криву навчання та дозволяє зосередитися безпосередньо на процесі створення або аналізу візуальних ефектів. Для досягнення цієї мети була створена структурна схема інтерфейсу користувача, яка відображає основні елементи управління та їх взаємозв’язки (див. рисунок 3.2).

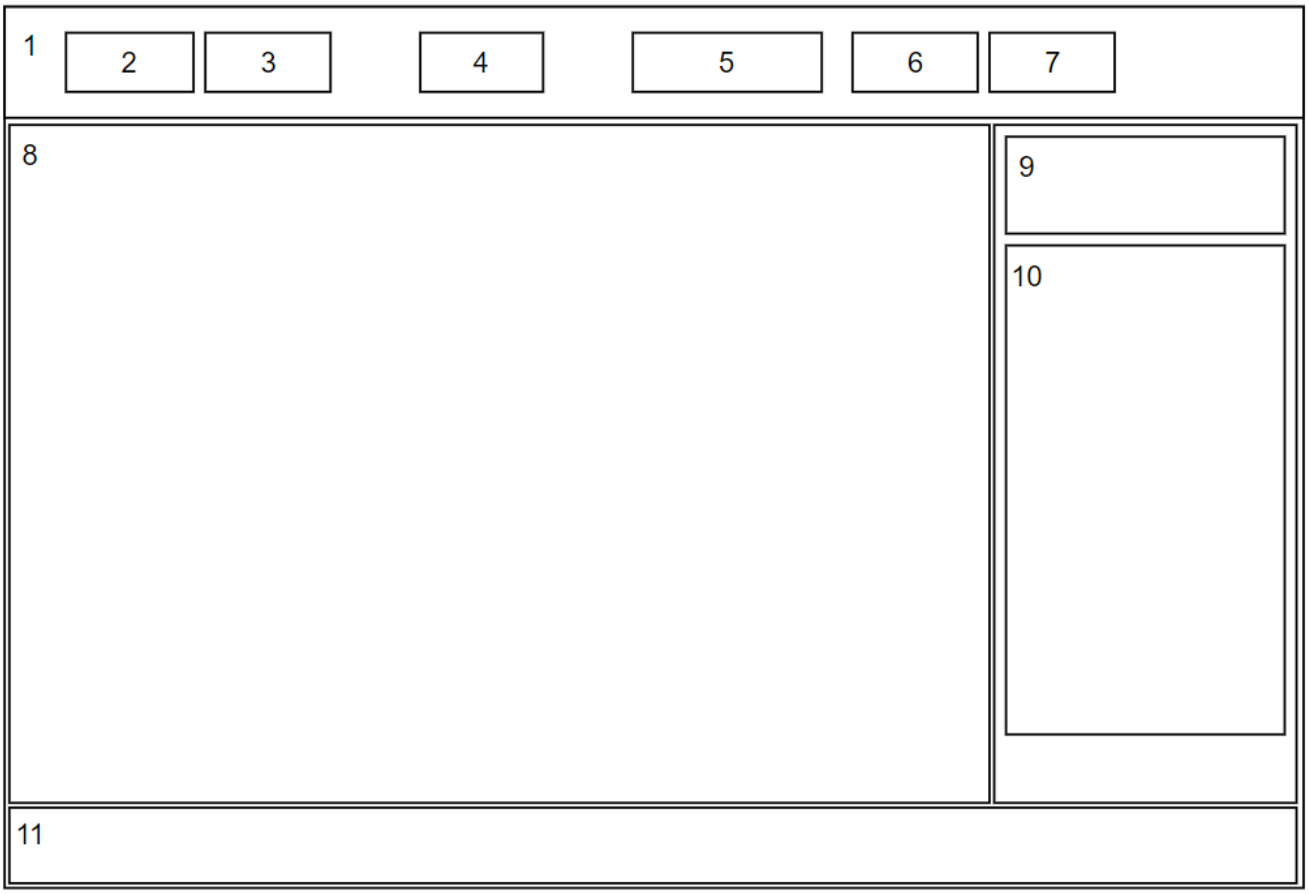

Рисунок 3.2 – Структурна схема інтерфейсу

Основні елементи інтерфейсу користувача:

1. Панель інструментів.
2. Кнопка для збереження зображення.
3. Кнопка для очищення для полотна.
4. Випадаючий список для вибору типу ілюзії.
5. Випадаючий список для вибору ілюзії.
6. Кнопка для скидання параметрів ілюзії.
7. Кнопка для відкриття довідки.
8. Полотно для відображення ілюзії.

9. Область з загальними налаштуваннями.

10. Область з налаштуваннями вибраної ілюзії.

11. Короткий опис ілюзії.

Після запуску застосунку користувач потрапляє на стартовий екран, який складається з полотна для візуалізації та текстового повідомлення з підказкою «Оберіть ілюзію з переліку» (див. рисунок 3.3).

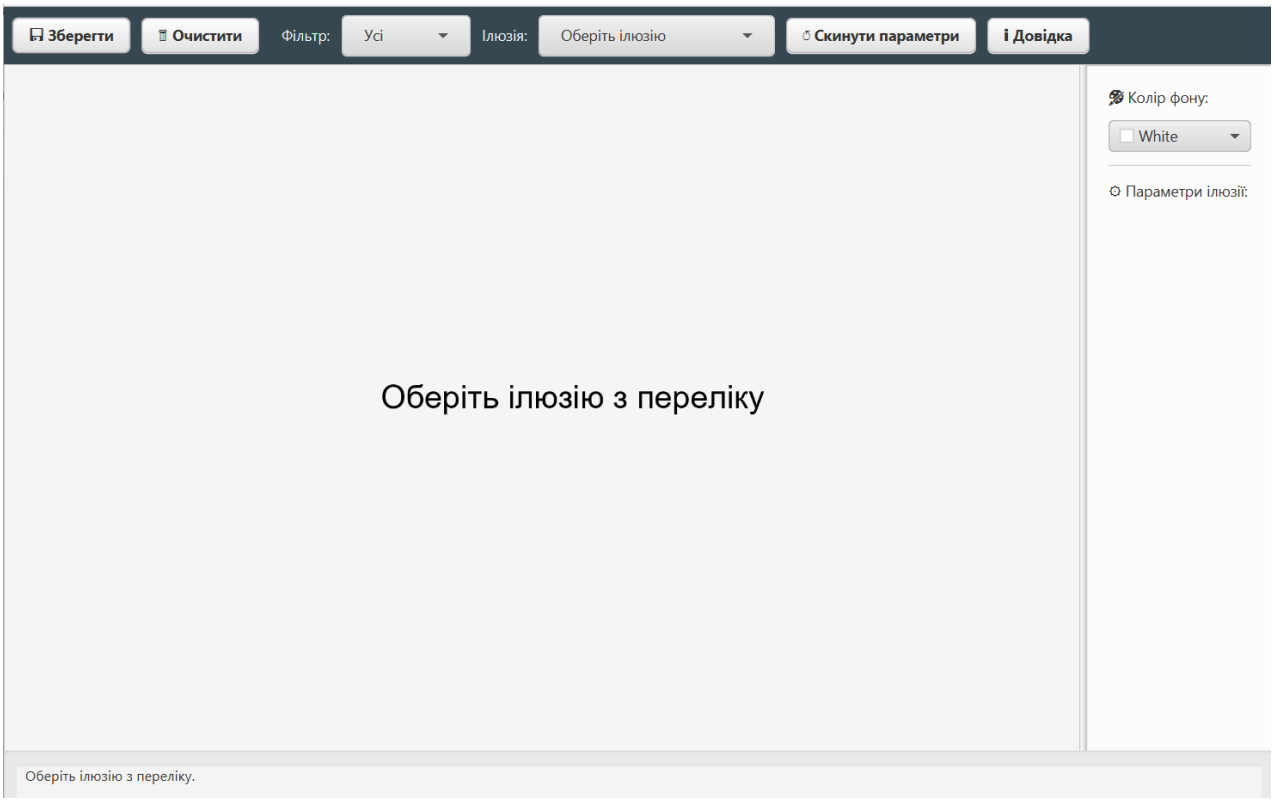

Рисунок 3.3 – Стартове вікно застосунку

Основним елементом керування програмним застосунком є верхня панель інструментів, розташована у верхній частині вікна, яка забезпечує доступ до ключових функцій програми та дозволяє швидко взаємодіяти з ілюзіями. Панель містить ряд інтуїтивно зрозумілих кнопок і випадаючих списків, кожен з яких відповідає за окремий аспект керування. Кнопка «Зберегти» дозволяє експортувати згенеровану ілюзію у форматі JPEG або PNG, зберігаючи результат роботи користувача у вигляді графічного файлу. Кнопка «Очистити» виконує функцію очищення полотна, видаляючи поточне зображення та дозволяючи

розпочати нову побудову з нуля. Поруч із цими елементами розташовані два випадаючі списки. Перший — із підписом «Фільтр» — надає можливість обрати категорію ілюзій («Усі», «Статичні», «Динамічні»), а другий — із написом «Ілюзія» — дозволяє вибрати конкретну оптичну ілюзію зі списку. Для зручності додано кнопку «Скинути параметри», яка повертає значення налаштувань до початкових. Остання кнопка «Довідка» відкриває вікно з інформацією про ілюзію (див. рисунок 3.4).

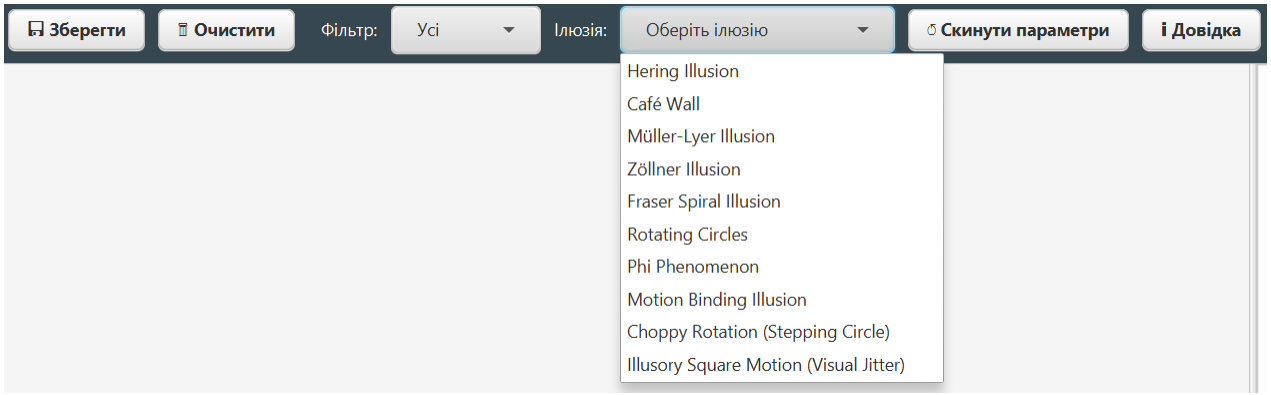

Рисунок 3.14 – Панель інструментів

Центральну частину інтерфейсу займає графічне полотно, яке виконує функцію основної області виведення згенерованих оптичних ілюзій у режимі реального часу. Саме на цьому полотні відображаються всі візуальні ефекти відповідно до обраної ілюзії та встановлених параметрів (див. рисунок 3.5).

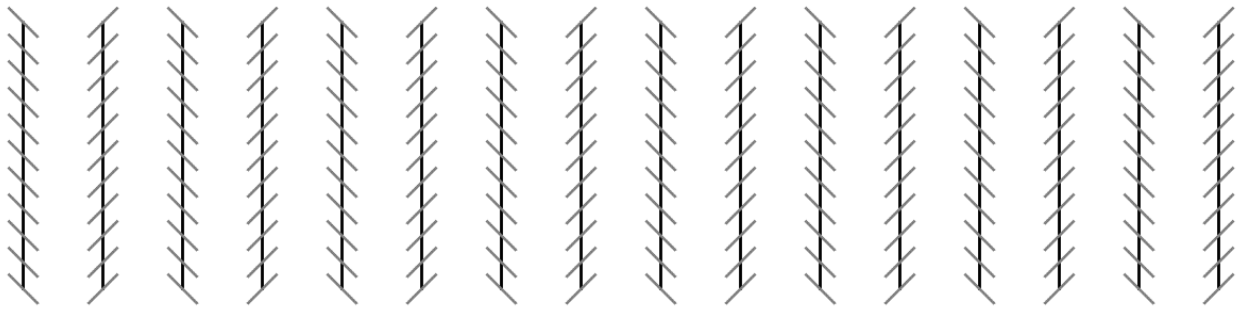

Рисунок 3.5 – Відображення ілюзії на полотні

Праву частину інтерфейсу займає панель налаштувань, яка забезпечує користувачеві зручний доступ до параметрів ілюзії та дозволяє тонко керувати її візуальними характеристиками. Цей блок поділено на дві функціональні секції. Перша секція містить загальні параметри, що є універсальними для всіх типів ілюзій – зокрема, вибір кольору фону, який визначає базовий вигляд полотна. Друга секція є динамічною – вона автоматично оновлюється залежно від того, яку ілюзію обрав користувач (див. рисунок 3.6).

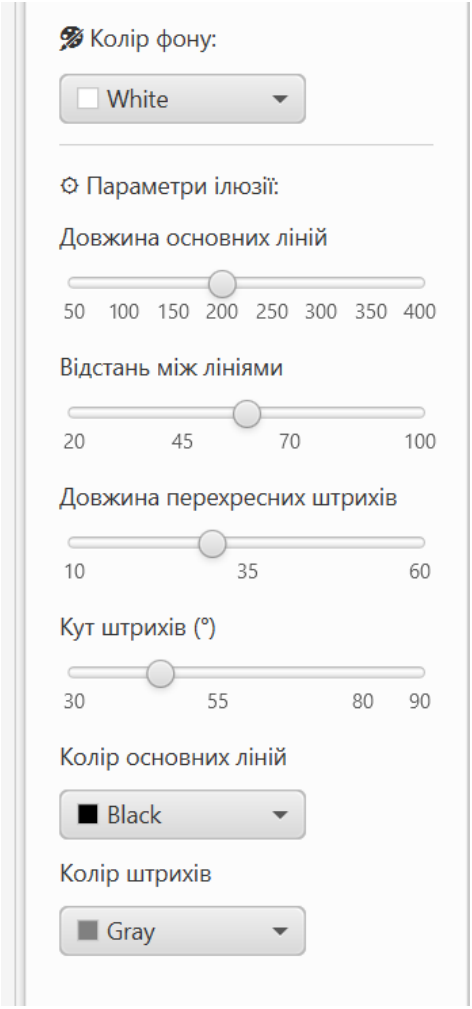

Рисунок 3.6 – Панель налаштувань

У нижній частині вікна розташоване спеціальне текстове поле, призначене для відображення опису обраної ілюзії. Це поле виконує інформативну функцію та дозволяє користувачеві ознайомитися з принципом дії ілюзії, її особливостями та механізмом візуального сприйняття (див. рисунок 3.7).

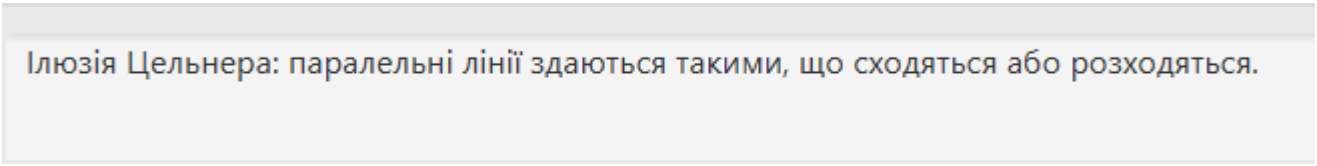

Рисунок 3.7 – Текстове поле з поясненням принципу дії ілюзії

Передбачено також спеціальне вікно довідки, яке містить структурований опис обраної ілюзії та слугує джерелом довідкової інформації, що допомагає користувачеві краще зрозуміти принцип її дії. Інтерфейс довідки реалізований у вигляді окремого діалогового вікна з текстовим полем, що забезпечує зручне ознайомлення навіть з великими обсягами інформації (див. рисунок 3.8).

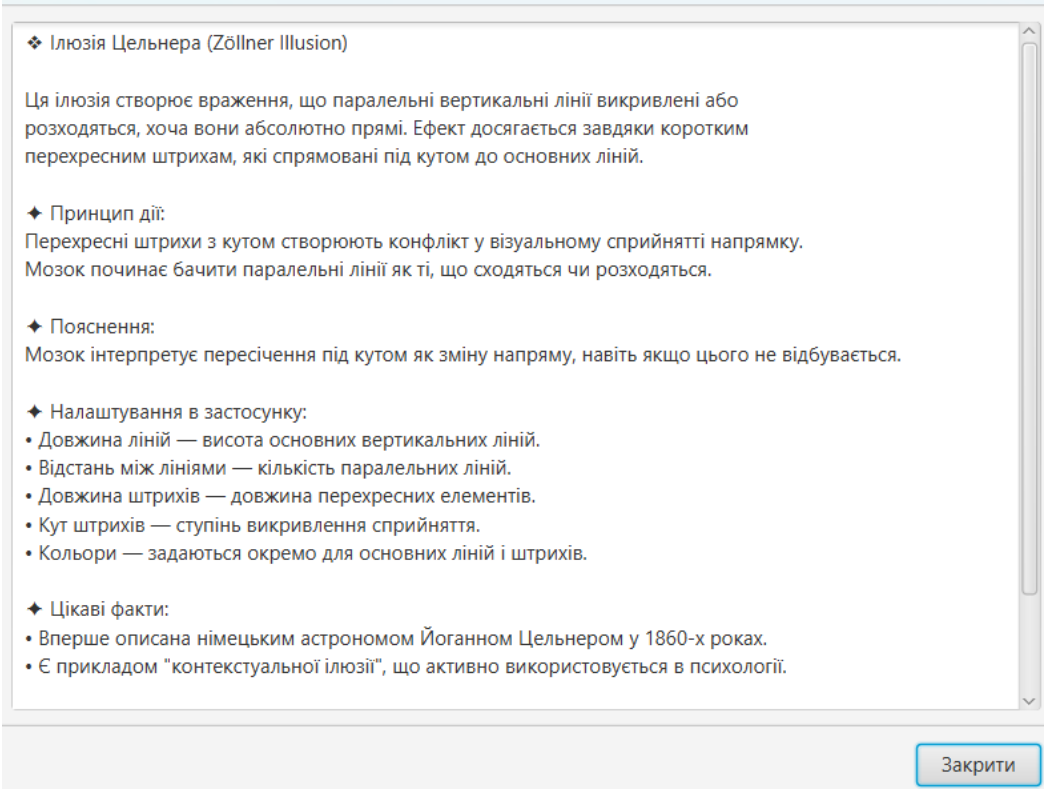

Рисунок 3.8 – Вікно довідки

3.4 Висновки

У третьому розділі обґрунтовано вибір технологій, мов програмування та інструментів, що використовуються для реалізації програмного забезпечення, призначеного для генерації оптичних ілюзій. Основна увага приділялася

поєднанню мови Java, платформи JavaFX, системи збірки проектів Maven та середовища розробки IntelliJ IDEA. Детально описано процес розробки ключових модулів програмного продукту, включаючи модулі для керування відображенням ілюзій, налаштування параметрів та автоматичного оновлення графіки у відповідь на дії користувача.

Окремо розглянуто архітектуру графічного інтерфейсу користувача, яка націлена на чіткий розподіл функціональних областей, верхньої панелі інструментів, центрального полотна для візуалізації, правої панелі параметрів та нижнього текстового поля для опису ілюзії. Була продемонстрована підтримка динамічної зміни елементів керування, завдяки чому кожна ілюзія має свій адаптивний набір налаштувань, які зручно змінюються користувачем у режимі реального часу. Було створено довідку, яка відкривається у вигляді окремого вікна та містить структурований опис обраної ілюзії, включаючи принцип її роботи, налаштування та цікаві факти. Це не тільки сприяє кращому розумінню механізмів ілюзії, але й підвищує освітню цінність програми. В результаті вдалося створити ефективне, інтуїтивно зрозуміле середовище, яке поєднує графічну складність візуалізації з гнучкістю керування, забезпечуючи високий рівень взаємодії та адаптивність до потреб користувача.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

У процесі розробки програмного забезпечення для генерації оптичних ілюзій важливу роль відіграє етап тестування, який дозволяє оцінити якість реалізованих функцій, стабільність програми та її відповідність технічним та функціональним вимогам. Тестування – це складний процес, що включає різні види тестів, кожен з яких має своє призначення. функціональне тестування перевіряє правильність реалізації функцій. Тестування продуктивності перевіряє швидкість та реакцію системи на навантаження. Тестування зручності використання перевіряє зручність та логіку взаємодії користувача з інтерфейсом, а тестування надійності перевіряє стабільність програми в умовах помилок або некоректного введення. Додатково можуть проводитися модульні та інтеграційні тести, які дозволяють тестувати окремі компоненти ізольовано або в складі всієї системи.

Враховуючи особливості програмного продукту, основна увага приділялася функціональному тестуванню, оскільки ключовим завданням програми є правильне створення графічних ілюзій відповідно до обраного типу, заданих параметрів та дій користувача. Важливо було переконатися, що всі алгоритми генерації ілюзій працюють без збоїв, параметри змінюються в режимі реального часу, а візуалізація оновлюється плавно та без зависань. Тестування включало перевірку генерації як статичних, так і динамічних ілюзій, вибору ілюзії зі списку, зміни кольорів, кількості елементів, розміру, швидкості анімації, а також стабільності функцій очищення полотна, скидання параметрів та експорту зображень.

Функціональне тестування дозволило виявити та усунути критичні помилки. В рамках тестування було протестовано всі ключові сценарії використання, а саме вибір ілюзії, її генерація на полотні, взаємодія з налаштуваннями через слайдери, випадаючі списки, колірні палітри, запуск та

зупинка анімації, виклик довідки та експорт результатів.

Для початку тестування було обрано ілюзію Café Wall. Після запуску цієї ілюзії програма правильно згенерувала чорно-білий візерунок, що складається з чергування горизонтальних рядів квадратів зі зміщенням кожного наступного ряду, що призвело до характерного ефекту нахилу горизонтальних ліній, які насправді є ідеально прямими. Важливим аспектом тестування було перевірити, чи система за замовчуванням встановлює правильні значення для зміщення рядка, кількості рядків і стовпців, а також кольору фону, який у цій ілюзії зазвичай заповнює проміжки між квадратами (див. рисунок 4.1).

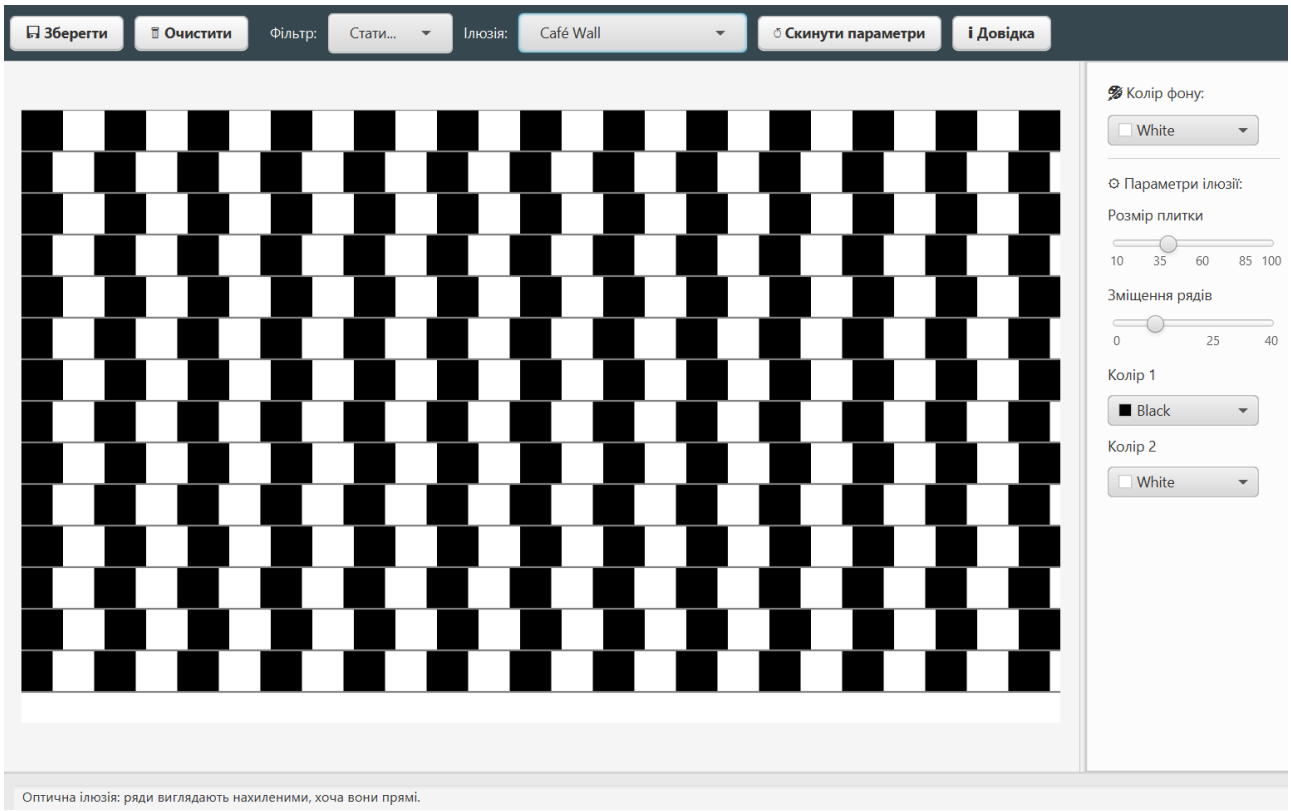

Рисунок 4.1 – Стандартний вигляд ілюзії "Café Wall"

Далі було змінено розмір плитки, що є одним з ключових параметрів ілюзії. Зменшення розміру плитки до мінімально допустимого значення дозволило перевірити здатність програми коректно відображати велику кількість дрібних графічних елементів без втрати чіткості та якості зображення. В процесі тестування було підтверджено, що система правильно адаптує кількість плиток до поточного розміру полотна, забезпечуючи рівномірне заповнення простору.

При цьому програма дотримується правильного чергування кольорів (чорного та білого), зберігаючи ілюзію кривизни горизонтальних ліній незалежно від масштабу. Це свідчить про точність реалізації алгоритму генерації сітки плиток та її масштабування (див. рисунок 4.2).

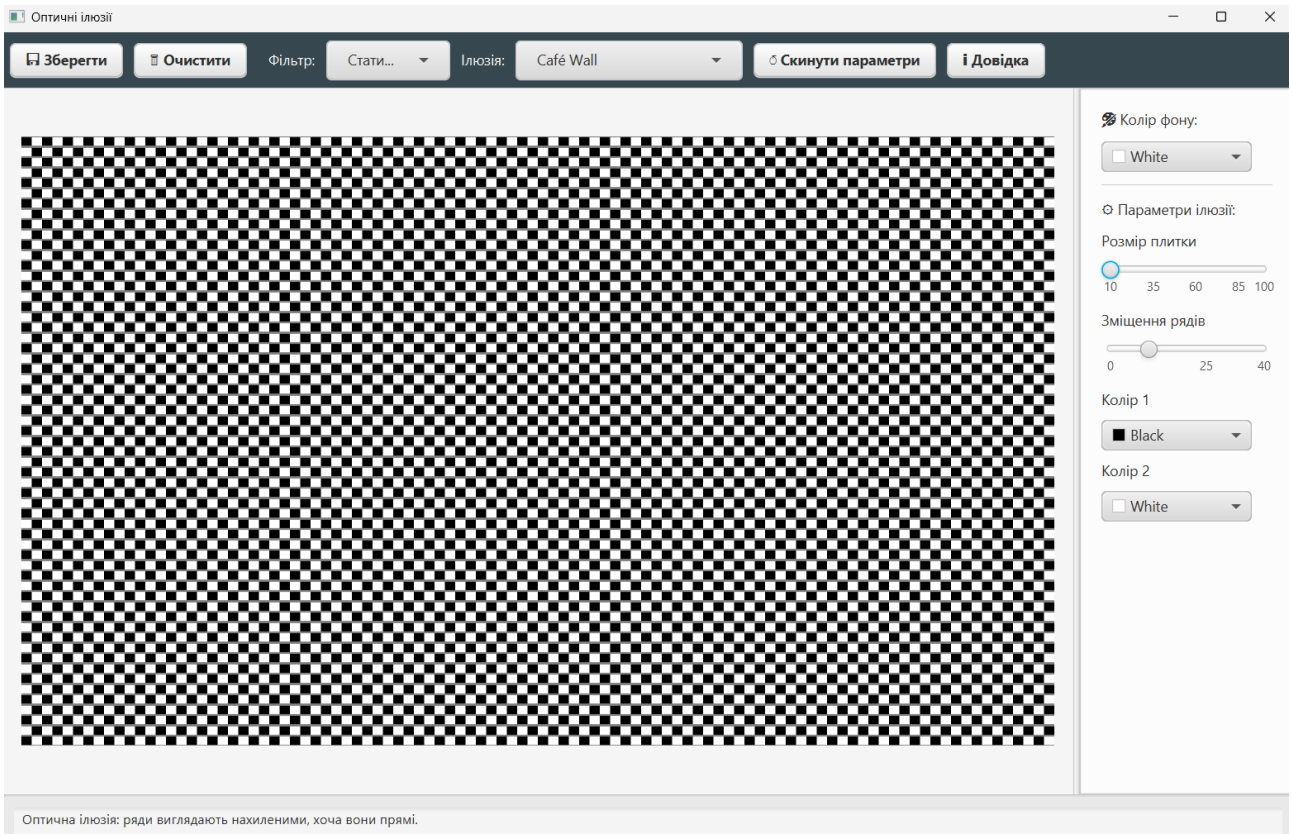

Рисунок 4.2 – "Café Wall" із мінімальним розміром плитки

Після цього було протестовано функцію зміни колірної гами. Для перевірки ефективності механізму та якості візуалізації навмисно були обрані кольори з високою контрастністю – яскраво-жовтий для світлих плиток і темно-синій для темних. Таке поєднання дозволяє чітко оцінити вплив кольору на сприйняття ілюзії, особливо в контексті його здатності викликати ефект викривлення ліній при зміні візуального середовища. Зміна кольорів здійснювалася за допомогою стандартних елементів керування – палітр ColorPicker, інтегрованих у праву панель налаштувань. Після вибору нових значень програма миттєво оновлювала зображення на полотні, перемальовуючи плитки без затримок та збоїв, що підтвердило стабільність та коректну роботу

системи в режимі реального часу при зміні параметрів. Результати тестування підтвердили, що обробка кольорів здійснюється правильно (див. рисунок 4.3).

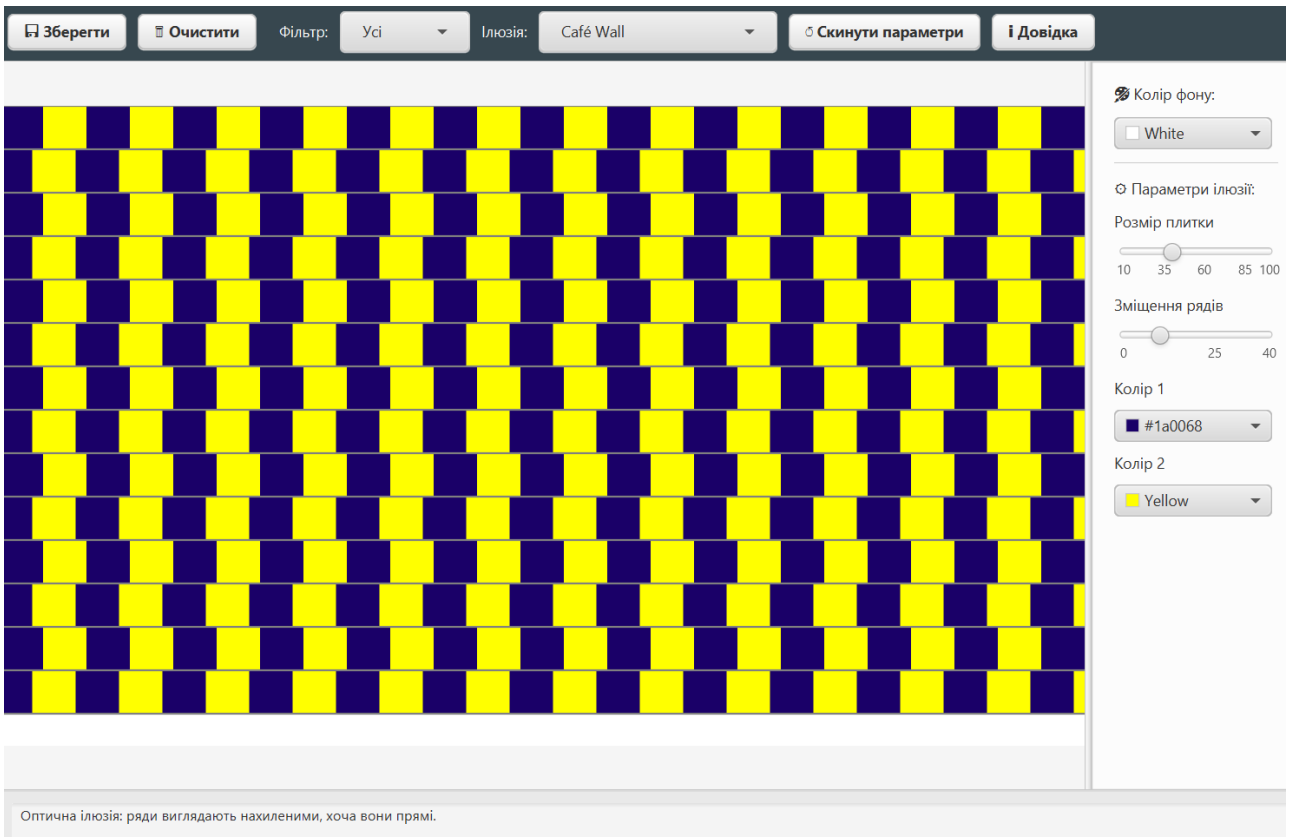

Рисунок 4.3 – "Café Wall" із зміненими кольорами плиток

В рамках функціонального тестування також було протестовано роботу модуля довідки для ілюзії «Café Wall». Після вибору відповідної ілюзії та натискання кнопки « Довідка» на верхній панелі інструментів відкривалося окреме вікно, яке містить структуровану текстову інформацію про ілюзію: загальний опис, принцип роботи, пояснення ефекту, список параметрів налаштування, а також цікаві факти (див. рисунок 4.4). Особливу увагу було приділено перевірці автоматичної адаптації контенту до розміру вікна, а також наявності вертикальної прокрутки для перегляду повного тексту при великій кількості інформації. Також було протестовано функціональність кнопки «Закрити» – після її натискання вікно довідки успішно закривається, повертаючи користувача до основного інтерфейсу без втрати даних та зміни поточних налаштувань ілюзії. Тестування підтвердило, що модуль довідки працює

стабільно, забезпечує доступ до корисної інформації та виконує свою інформаційно-освітню функцію без збоїв.

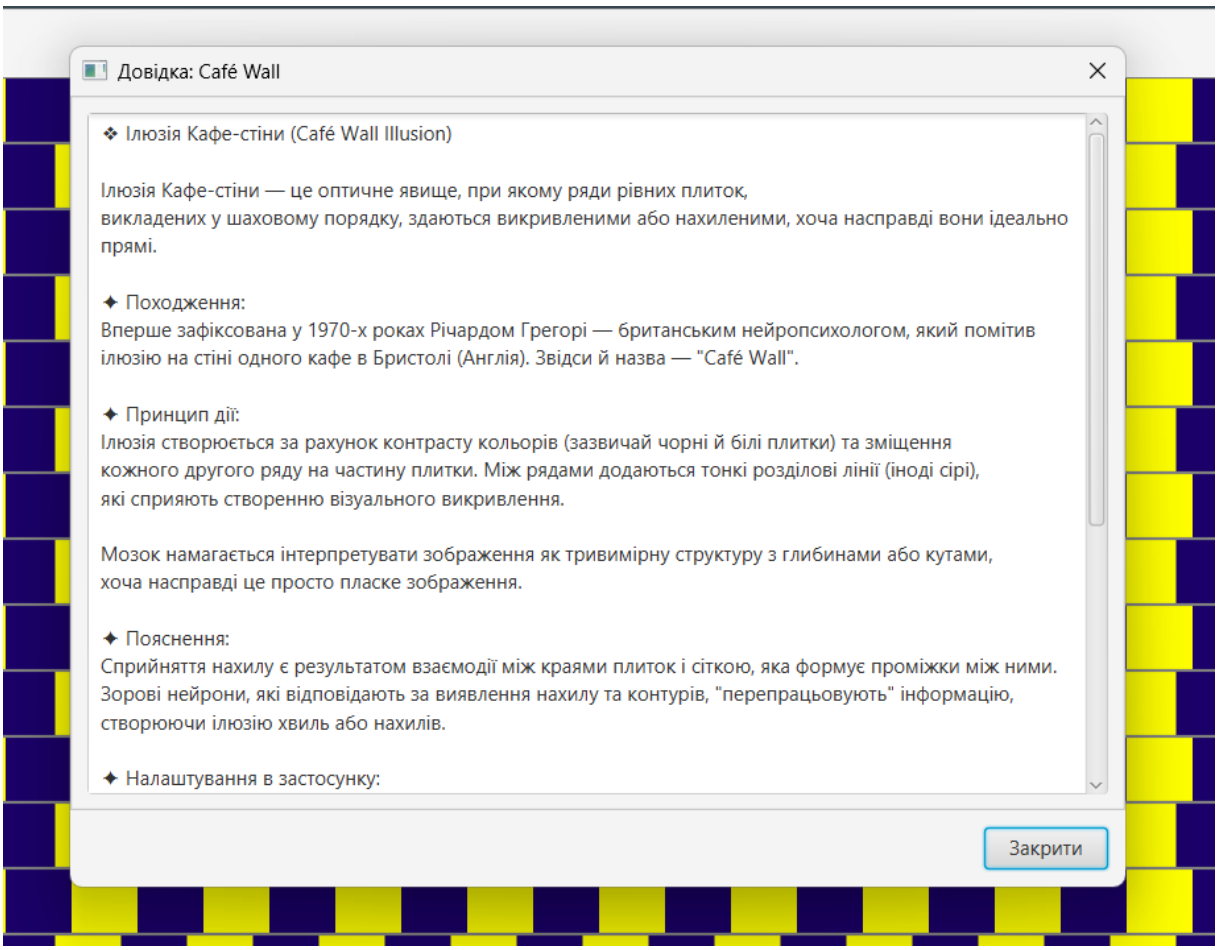

Рисунок 4.4 – Довідка для ілюзії "Café Wall"

Було протестовано роботу кнопок «Скинути параметри» та «Очистити». Після натискання кнопки «Скинути параметри» було перевірено, що всі значення автоматично повернулися до початкових, а графічне полотно миттєво оновлювалося відповідно до основних параметрів. Кнопка «Очистити» була протестована на здатність повністю очистити графічне полотно від зображення поточної ілюзії. Після її активації програма успішно видалила вміст полотна та повідомлення «Виберіть ілюзію зі списку».

В межах функціонального тестування наступною було протестовано ілюзію Fraser Spiral, яка належить до категорії статичних ілюзій, що створюють хибне враження обертання спіралі. Після запуску ілюзії програма коректно

згенерувала концентричні кільця, які візуально виглядають як спіраль завдяки чергуванню кольорових секторів з певним зміщенням. Було перевірено, що система встановлює оптимальні значення параметрів за замовчуванням, кількість кіл, кількість секторів та коефіцієнт спіральності – які забезпечують чіткий та виразний ілюзорний ефект (див. рисунок 4.5).

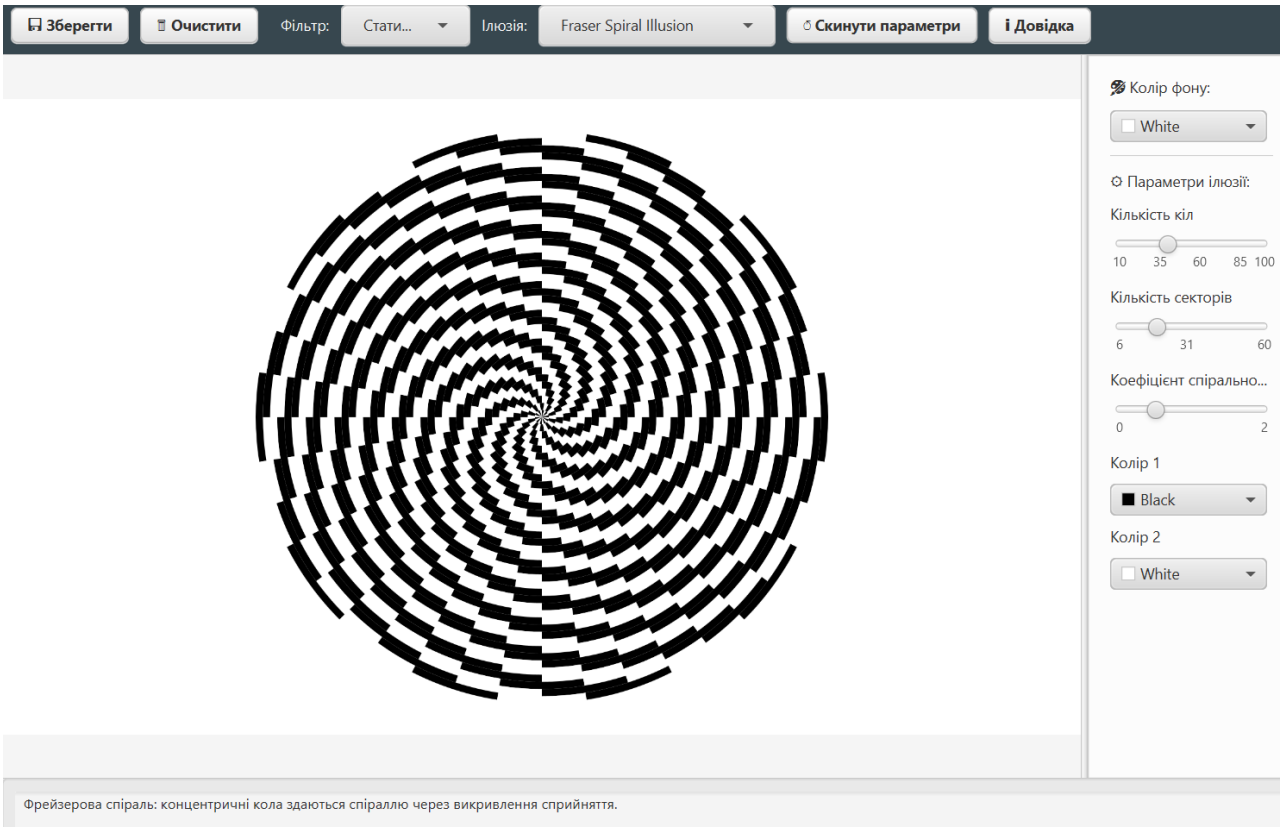

Рисунок 4.5 – Стандартний вигляд ілюзії "Fraser Spiral"

Тестування включало зміну значень усіх доступних параметрів. Повзунки для регулювання кількості кіл та секторів працювали стабільно, дозволяючи динамічно змінювати складність структури. Збільшення коефіцієнта спіралі посилювало ілюзію скручування, що підтвердило правильність візуальної реакції на зміну параметра. Колірна палітра також працювала без помилок – після вибору нових кольорів у ColorPicker зображення миттєво оновлювалося, зберігаючи геометричну цілісність та візуальний ефект. Всі зміни параметрів відображалися в режимі реального часу без затримок та графічних артефактів (див. рисунок 4.6).

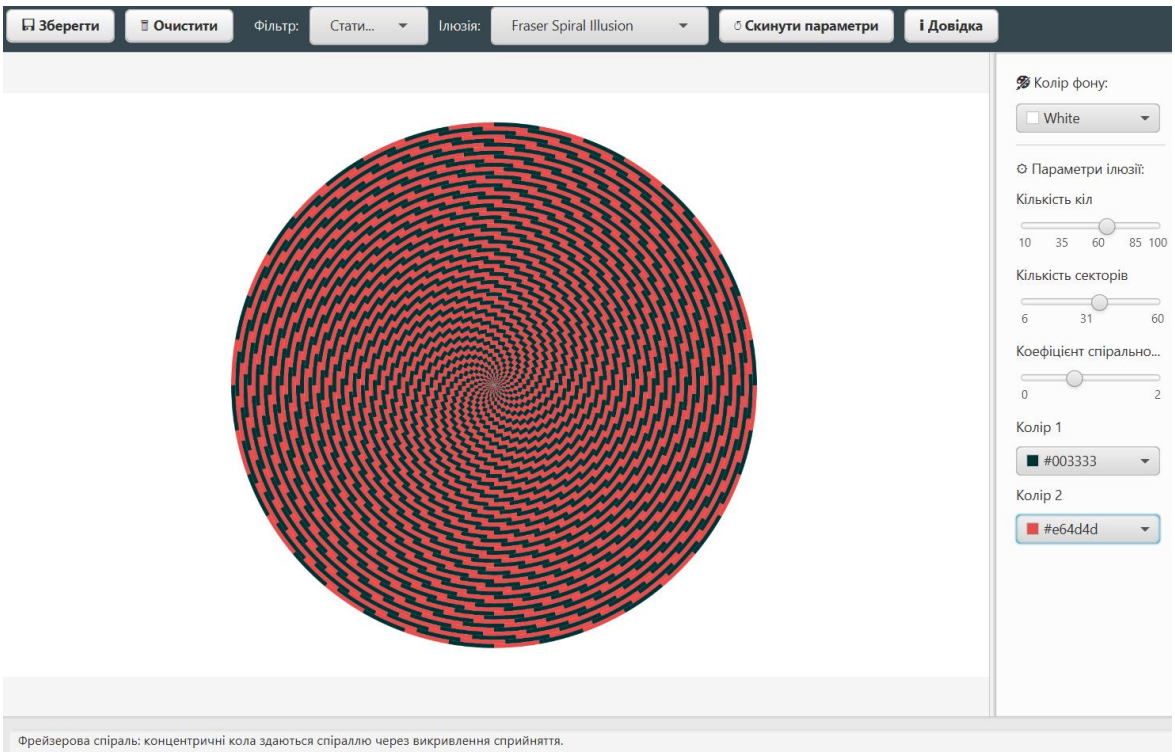

Рисунок 4.6 – "Fraser Spiral" із зміненими параметрами

Під час тестування функції збереження зображень було перевірено роботу кнопки «Зберегти», розташованої на панелі інструментів. Після натискання кнопки відкривалося стандартне діалогове вікно збереження файлу, в якому користувач міг вказати ім'я файлу, вибрати місце для збереження, а також вибрати потрібний формат зображення JPEG або PNG зі списку доступних (див. рисунок 4.7).

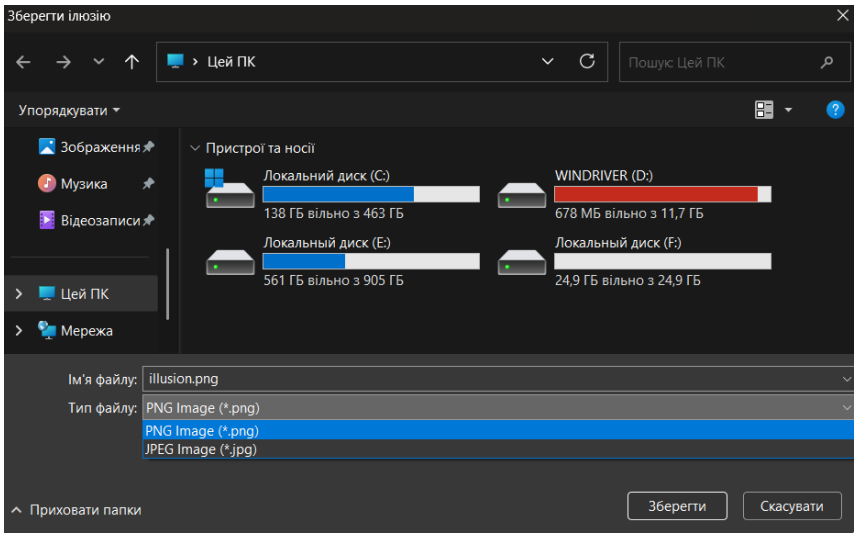

Рисунок 4.7 – Вікно збереження зображення ілюзії

Після збереження зображення у форматі JPEG, отриманий файл було перевірено на якість графіки, відтворення кольорів та загальну цілісність зображення (див. рисунок 4.8).

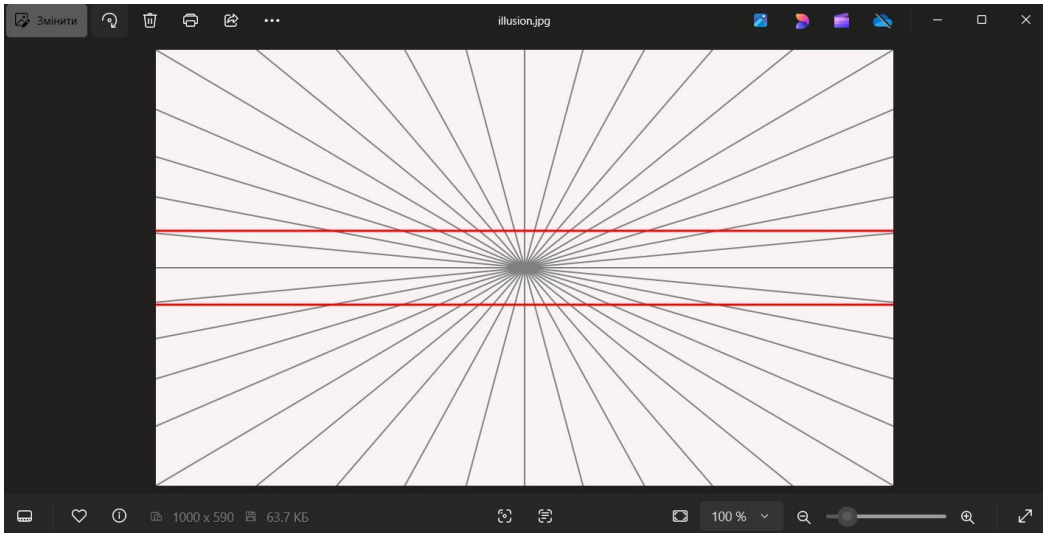

Рисунок 4.8 – Збережена ілюзія в форматі JPEG

Збереження у форматі PNG також було виконано коректно. У цьому випадку було перевірено підтримку прозорих фонів. Це дозволяє надалі використовувати зображення для композицій або в дизайні без додаткової обробки. Ілюзії були збережені з високою точністю, без втрати різкості або кольору, що робить формат PNG зручним для подальшого професійного використання (див. рисунок 4.9).

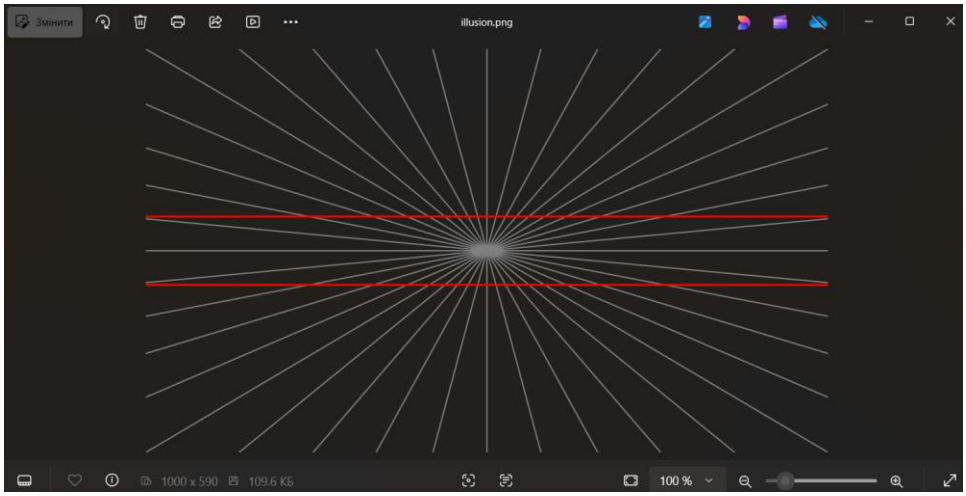

Рисунок 4.9 – Збережена ілюзія в форматі PNG

Для тестування динамічної ілюзії Motion Binding було перевірено коректність запуску, зупинки та анімації рухомих елементів. Після вибору відповідної ілюзії зі списку та натискання кнопки «Старт» система ініціювала анімацію вертикальних ліній та рухомих точок. Було підтверджено, що точки рухаються синхронно з вертикальними лініями, утворюючи оптичний ефект зв'язування з фоном – саме це створює характерне сприйняття комбінованого руху, на якому базується ілюзія (див. рисунок 4.10).

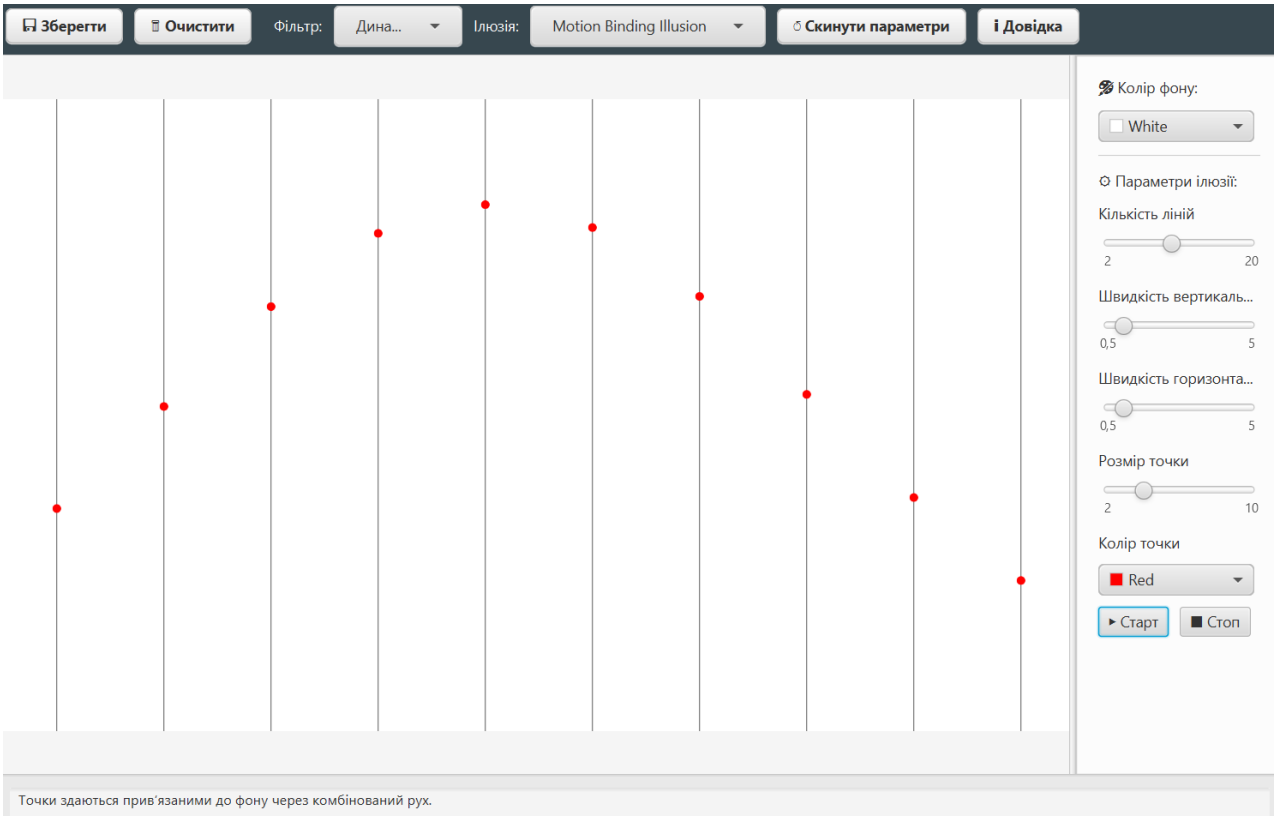

Рисунок 4.10 – Тестування запуску анімації ілюзії «Motion Binding»

Окремо було протестовано зміну параметрів ілюзії в режимі реального часу: кількість ліній, швидкість вертикального та горизонтального руху, розмір точок та їх колір. Зміни, внесені за допомогою слайдерів та випадаючих списків, одразу впливали на графічне зображення без перезапуску анімації. Особливу увагу було приділено перевірці плавності руху на різних швидкостях, а також чіткості точок навіть при їх мінімальному розмірі. Візуальний ефект не

втрачається незалежно від комбінації налаштувань, що свідчить про правильну реалізацію логіки руху та стабільність модуля анімації (див. рисунок 4.11).

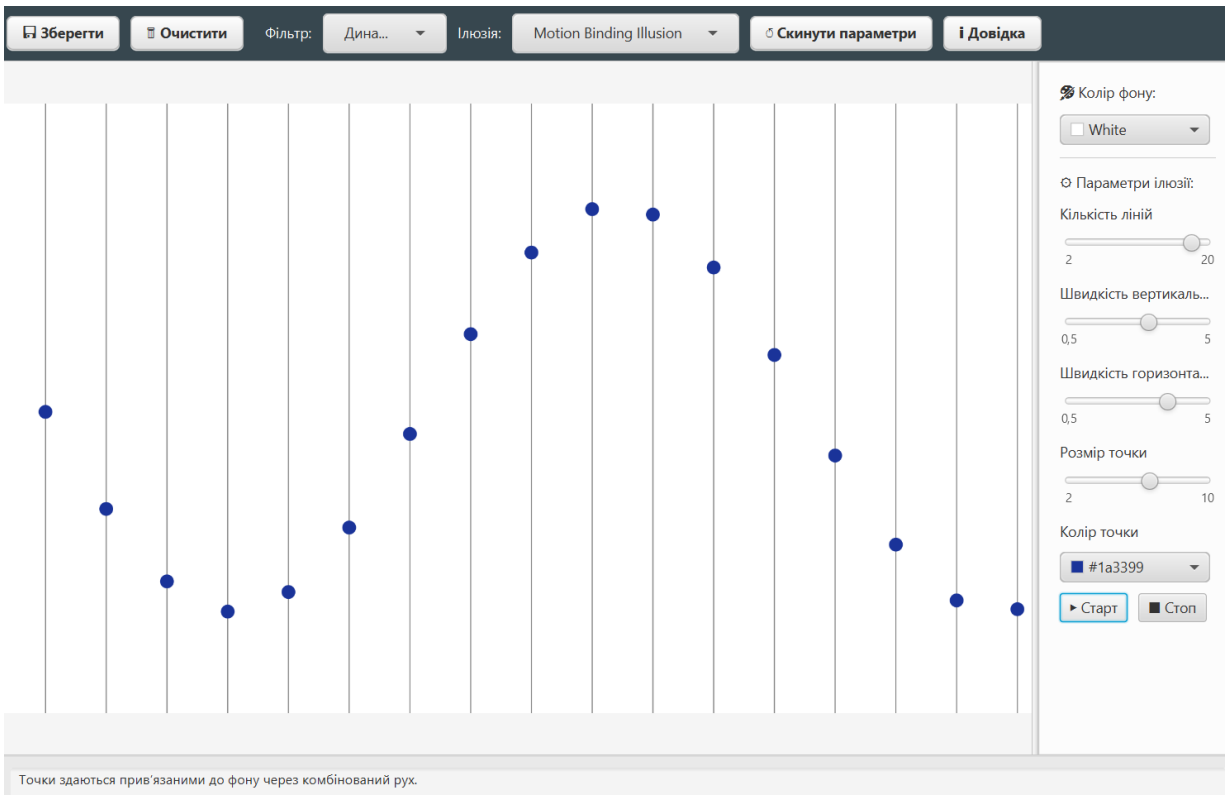

Рисунок 4.11 – Тестування зміни параметрів ілюзії «Motion Binding»

Результати тестування показали стабільну та надійну роботу програмного забезпечення в умовах взаємодії з різними видами оптичних ілюзій, як статичними, так і динамічними. Усі функціональні компоненти працюють злагоджено, забезпечуючи користувачеві зручний та зрозумілий інтерфейс для взаємодії з графічними об'єктами. Параметри ілюзій можна змінювати в режимі реального часу за допомогою інтерактивних елементів керування, при цьому візуальний результат оновлюється миттєво без затримок та збоїв.

4.2 Розробка інструкції користувача

Інструкція користувача є важливим компонентом програмного забезпечення, оскільки вона надає повний огляд можливостей програми,

процедури її встановлення, запуску, використання, а також технічних вимог до комп'ютера. Головна мета – допомогти користувачеві швидко розпочати роботу з програмою, ефективно використовувати її функціональність та уникнути можливих труднощів під час роботи.

Оскільки програма створена з використанням сучасних технологій JavaFX для обробки графіки та анімації, для її стабільної та комфортної роботи потрібна відповідна апаратна конфігурація. Програма використовує системні ресурси для обробки ефектів, включаючи графічне полотно, анімацію в реальному часі та параметричне оновлення візуальних елементів. Тому комп'ютер користувача повинен підтримувати апаратне прискорення та мати актуальну версію середовища виконання Java.

У таблиці 4.1 подано рекомендовані та мінімальні вимоги до апаратного та програмного забезпечення для забезпечення коректної роботи застосунку:

Таблиця 4.1 – Вимоги до апаратного та програмного забезпечення

Вимоги до	Мінімально	Рекомендовано
Операційна система	Windows 7 / Linux / macOS	Windows 10 / 11 / Linux / macOS
Процесор	2-ядерний, 1.8 ГГц	4-ядерний, 2.5 ГГц і вище
Оперативна пам’ять	2 ГБ	4 ГБ і вище
Відеокарта	Вбудована	Дискретна з апаратним прискоренням
Java Runtime Environment (JRE)	Входить до складу програми	Входить до складу програми
Розширення екрана	1280×720	1920×1080
Вільне місце на диску	150 MB	300 MB

Процес встановлення застосунку максимально спрощений і не потребує інсталяції сторонніх бібліотек або додаткових програм. Програма йде в комплекті у вигляді зібраного пакету у форматі .jar, а також разом із необхідними

файлами ресурсів. Для початку роботи користувачу потрібно виконати лише кілька простих кроків:

1. Завантажити архів із програмою з наданого джерела.
2. Розпакувати архів у зручне місце на локальному диску.
3. Запустити файл OpticalIllusionsApp.jar подвійним кліком.
4. Програма відкриється у новому вікні з графічним інтерфейсом.

До складу розпакованого архіву також входить текстовий файл README.txt, який містить базову довідкову інформацію щодо програми. У ньому подано короткий опис застосунку, основні технічні вимоги, інструкції щодо запуску, а також контактну інформацію для зворотного зв’язку чи повідомлення про помилки. Цей файл рекомендовано переглянути перед першим запуском програми.

4.3 Висновки

У четвертому розділі було протестовано основні функції програмного засобу для генерації оптичних ілюзій. Перевірено коректність графічного інтерфейсу, динамічну зміну параметрів ілюзії, реакцію на відсутність вибору та нестандартні дії користувача. Особливу увагу було приділено коректності відображення результатів у режимі реального часу та збереженню ілюстрацій у форматах JPEG та PNG, включаючи підтримку прозорого фону.

Також було протестовано роботу кнопок «Очистити» та «Скинути параметри», анімаційні ефекти, специфічні параметри для кожної ілюзії та відкриття вікна довідки. Усі компоненти працювали стабільно, без збоїв та затримок.

Було створено інструкцію користувача, що містить опис встановлення, запуску та роботи з програмою, що дозволяє забезпечити зручність і зрозумілість використання програмного продукту для кінцевого користувача без необхідності звертатися до технічної документації.

ВИСНОВКИ

У бакалаврській роботі було розроблено програмний застосунок для генерації оптичних ілюзій з використанням мови програмування Java та платформи JavaFX. Дослідження охоплювало як теоретичні аспекти візуального сприйняття, так і практичні засоби реалізації оптичних ефектів у цифровому середовищі. Було проведено детальний аналіз сучасних підходів до створення ілюзій, включаючи класифікацію візуальних ефектів, принципи їх побудови та технічні засоби візуалізації. Було виявлено, що поєднання математичних моделей з векторною графікою є найефективнішим підходом до реалізації як статичних, так і динамічних ілюзій у комп'ютерних додатках.

В ході проекту було створено програмний засіб на основі модульної архітектури, який забезпечує незалежну реалізацію кожної ілюзії як окремого класу. Це дозволяє легко розширювати функціональність програми, додаючи нові ілюзії без необхідності зміни базової логіки роботи додатку. Візуалізація реалізована через Canvas API, який забезпечує точне та продуктивне відображення графічних елементів у режимі реального часу. Окрім базового функціоналу, реалізовано підтримку інтерактивного налаштування параметрів кожної ілюзії: колірної палітри, розміру елемента, кількості структурних компонентів, швидкості анімації тощо. Кожна зміна параметрів супроводжується негайним оновленням зображення на полотні.

Особливу увагу було приділено створенню зручного та інтуїтивно зрозумілого інтерфейсу користувача. Програма містить верхню панель інструментів для базових операцій (збереження, очищення, скидання), панель налаштувань праворуч, яка динамічно змінюється залежно від типу обраної ілюзії, центральне полотно для відображення зображення та нижню текстову панель з коротким описом ілюзії. Додатково реалізовано систему довідки, яка відкривається в окремому вікні та містить повну інформацію про ілюзію, її принцип роботи, налаштування та приклади використання.

Практичне значення результатів полягає у створенні універсального програмного засобу, який можна використовувати в навчальному процесі для демонстрації візуальних ефектів, у дослідницькій діяльності з аналізу особливостей візуального сприйняття, а також у цифровому мистецтві та дизайні. Програма легко запускається з виконуваного файлу .jar та сумісна з основними операційними системами, що робить її доступною для широкого кола користувачів.

Тестування охопило всі основні функціональні компоненти: запуск ілюзій, динамічну зміну параметрів, роботу з анімацією, збереження зображень у форматах PNG та JPEG та відображення довідкової інформації. Усі протестовані сценарії підтвердили стабільність роботи, коректну обробку дій користувача та відповідність програмного продукту заявленим функціональним вимогам.

Отримані результати свідчать про успішне виконання поставлених завдань та можуть бути використані як основа для подальшого розширення функціональності програмного засобу, зокрема, додавання нових ілюзій, інтеграції з сенсорними пристроями або використання в наукових експериментах з візуального сприйняття.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gregory R. L. Eye and Brain: The Psychology of Seeing / R. L. Gregory. – Princeton: Princeton University Press, 2015. – 288 p.

2. Hoffman D. D. Visual Intelligence: How We Create What We See / D. D. Hoffman. – New York: W. W. Norton & Company, 2000. – 294 p.

3. Purves D., Lotto R. B., Nundy S. Why We See What We Do Redux: A Wholly Empirical Theory of Vision / D. Purves, R. B. Lotto, S. Nundy. – Sunderland: Sinauer Associates, 2011. – 400 p.

4. Катєльніков Д.І., Вітовський С.М. Алгоритмічні підходи до моделювання оптичних ілюзій у програмних системах / LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025) / Уклад. Д.І. Катєльніков, С.М. Вітовський [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24381 (дата звернення: 15.04.2025).

5. Gregory R. L. Perception and Illusions: Advances in Cognitive Neuroscience / R. L. Gregory. – Cambridge: Cambridge University Press, 2020. – 336 p.

6. Tyler C. W. Visual Perception: The Influence of Geometrical and Contrast Illusions / C. W. Tyler / Journal of Vision Science. – 2019. – 215 p.

7. Westheimer G. Moiré Patterns and Their Role in Visual Perception / G. Westheimer / Vision Research. – 2021. – 181 p.

8. Visual Phenomena & Optical Illusions [Електронний ресурс] – Режим доступу до ресурсу: https://michaelbach.de/ot/ (дата звернення: 17.04.2025).

9. The Illusion Index [Електронний ресурс] – Режим доступу до ресурсу: https://www.illusionsindex.org (дата звернення: 17.04.2025).

10. Silk – Interactive Generative Art [Електронний ресурс] – Режим доступу до ресурсу: http://weavesilk.com (дата звернення: 17.04.2025).

11. Java | Oracle [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/en/ (дата звернення: 23.04.2025).

12. JavaFX [Електронний ресурс] – Режим доступу до ресурсу: https://openjfx.io (дата звернення: 23.04.2025).

13. IntelliJ IDEA – the IDE for Pro Java and Kotlin Development [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/ (дата звернення: 23.04.2025)

14. Welcome to Apache Maven – Maven [Електронний ресурс] – Режим доступу до ресурсу: https://maven.apache.org (дата звернення: 23.04.2025)

15. Canvas (JavaFX 8) [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/Canvas.html (дата звер-нення: 23.04.2025).

16. Animation (JavaFX 8) [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/8/javafx/javafx/animation/Animation.html (дата зверне-ння: 23.04.2025).

17. What is a UML Diagram? | Different Types and Benefits [Електронний ресурс] – Режим доступу до ресурсу: https://miro.com/diagramming/what-is-a-uml-diagram/ (дата звернення: 25.04.2025).

18. Scene Builder [Електронний ресурс] – Режим доступу до ресурсу: https://gluonhq.com/products/scene-builder/ (дата звернення: 27.04.2025).

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
"21" березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка застосунку для генерації оптичних ілюзій з використанням мови Java і платформи JavaFx»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ к.т.н., доцент Д.І. Катєльніков
"21" 03 2025 р.
Виконав:
_____ студент гр. 4ПІ-21б С.М. Вітовський
"21" 03 2025 р.

ВНТУ – 2025

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка застосунку для генерації оптичних ілюзій з використанням мови Java і платформи JavaFx».

Галузь застосування – освітні, дослідницькі та демонстраційні сфери, пов’язані з вивченням зорового сприйняття та графічного моделювання

2. Підстава для розробки.

Підставою для розробки бакалаврської кваліфікаційної роботи є рішення засідання кафедри програмного забезпечення.

3. Мета та призначення розробки.

Метою бакалаврської кваліфікаційної роботи є розширення можливостей застосування оптичних ілюзій у навчальних, наукових, дизайнерських та експериментальних цілях шляхом розробки застосунку для генерації оптичних ілюзій з використанням мови Java і платформи JavaFx.

Призначення роботи – створення застосунку, який може бути використаний у навчальних, дослідницьких або демонстраційних цілях для аналізу особливостей зорового сприйняття.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Gregory R. L. Eye and Brain: The Psychology of Seeing / R. L. Gregory. – Princeton: Princeton University Press, 2015. – 288 p.

2. Hoffman D. D. Visual Intelligence: How We Create What We See / D. D. Hoffman. – New York: W. W. Norton & Company, 2000. – 294 p.

3. Purves D., Lotto R. B., Nundy S. Why We See What We Do Redux: A Wholly Empirical Theory of Vision / D. Purves, R. B. Lotto, S. Nundy. – Sunderland: Sinauer Associates, 2011. – 400 p.

5. Технічні вимоги

Вихідні дані до роботи: перелік оптичних ілюзій – статичні, динамічні; методи візуалізації – Canvas API, JavaFX GraphicsContext; режим відображення – апаратно прискорена векторна графіка; вихідні параметри – координати елементів ілюзій, кольори, швидкість анімації, кількість об’єктів; вхідні дані – вибір типу ілюзії, налаштування параметрів, вибір кольорової палітри, взаємодія з інтерфейсом користувача; методи реалізації – інкапсуляція елементів у класи, обробка подій у режимі реального часу; програмне середовище – Java, JavaFX, Maven, середовище розробки IntelliJ IDEA.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

1. Пояснювальна записка до БКР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз методів моделювання оптичних ілюзій у цифровому середовищі	25.03.25 – 09.04.25
2	Розробка методу створення структури та алгоритмів роботи системи	10.04.25 – 30.04.25
3	Розробка основних модулів застосунку та методу генерації оптичних ілюзій	31.04.25 – 16.05.25
4	Тестування роботи застосунку	17.05.25 – 23.05.25
5	Оформлення матеріалів до захисту БКР	24.05.25– 30.05.25

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка застосунку для генерації оптичних ілюзій з використанням мови Java і платформи JavaFx

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра програмного забезпечення, ФІТКІ, 4ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 2,1 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Катєльніков Д.І.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Вітовський С.М.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

MainApp.java:

```
package com.example;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class MainApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/main-
view.fxml"));
        Parent root = loader.load();

        Scene scene = new Scene(root);

scene.getStylesheets().add(getClass().getResource("/style/style.css").toExternal
Form());

        primaryStage.setTitle("Оптичні ілюзії");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

MainController.java:

```
package com.example.controller;

import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.SnapshotParameters;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.control.Dialog;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.image.WritableImage;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import com.example.illusion.*;
import javafx.embed.swing.SwingFXUtils;
import javafx.stage.FileChooser;

import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
```

```
import java.io.File;
import java.io.IOException;
import java.util.List;

public class MainController {

    @FXML private Canvas canvas;
    @FXML private ComboBox<String> illusionSelector;
    @FXML private ComboBox<String> illusionTypeFilter;
    @FXML private ColorPicker backgroundColorPicker;
    @FXML private TextArea descriptionArea;
    @FXML private VBox illusionSettingsPane;
    private boolean isExportTransparent = false;
    private final List<OpticalIllusion> illusions = List.of(
            new HeringIllusion(),
            new CafeWallIllusion(),
            new MullerLyerIllusion(),
            new ZollnerIllusion(),
            new FraserSpiralIllusion(),
            new RotatingCirclesIllusion(),
            new PhiPhenomenonIllusion(),
            new MotionBindingIllusion(),
            new ChoppyRotationIllusion(),
            new IllusorySquareMotionIllusion()
    );

    @FXML
    public void initialize() {
        illusionTypeFilter.getItems().addAll("Усі", "Статичні", "Динамічні");
        illusionTypeFilter.getSelectionModel().selectFirst();

illusionTypeFilter.getSelectionModel().selectedItemProperty().addListener((obs,
oldVal,
                            newVal) -> updateIllusionList());
        for (OpticalIllusion illusion : illusions) {
            illusion.setOnParameterChanged(this::drawPreview);
        }

illusionSelector.getSelectionModel().selectedItemProperty().addListener((obs,
oldVal, newVal)
                -> {
            drawPreview();
            updateUI();
        });
        backgroundColorPicker.setValue(Color.WHITE);
        backgroundColorPicker.valueProperty().addListener((obs, oldVal, newVal)
-> drawPreview());
        updateIllusionList();
    }

    private void updateIllusionList() {
        illusionSelector.getItems().clear();
        String filter = illusionTypeFilter.getValue();

        for (OpticalIllusion illusion : illusions) {
            if ("Статичні".equals(filter) && illusion.getType() !=
OpticalIllusion.IllusionType.
                    STATIC) continue;
            if ("Динамічні".equals(filter) && illusion.getType() !=
OpticalIllusion.IllusionType.
                    ANIMATED) continue;
            illusionSelector.getItems().add(illusion.getName());
        }
```

```
        illusionSelector.getSelectionModel().clearSelection();
        drawPreview();
        illusionSettingsPane.getChildren().clear();
    }

    private void updateUI() {
        String selected = illusionSelector.getValue();
        for (OpticalIllusion illusion : illusions) {
            if (illusion.getName().equals(selected)) {

illusionSettingsPane.getChildren().setAll(illusion.getSettingsPane());
                return;
            }
        }
    }

    private void drawPreview() {
        GraphicsContext gc = canvas.getGraphicsContext2D();
        String selected = illusionSelector.getValue();
        if (selected == null || selected.isEmpty()) {
            gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
            gc.setFill(Color.BLACK);
            gc.setFont(javafx.scene.text.Font.font("Arial", 30));
            gc.fillText("Оберіть ілюзію з переліку", 350, canvas.getHeight() /
2);
            descriptionArea.setText("Оберіть ілюзію з переліку.");
            illusionSettingsPane.getChildren().clear();
            return;
        }
        for (OpticalIllusion illusion : illusions) {
            if (illusion.getName().equals(selected)) {
                Color bg = isExportTransparent ? Color.TRANSPARENT :
backgroundColorPicker.getValue();
                gc.setFill(bg);
                gc.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());
                illusion.draw(gc, canvas.getWidth(), canvas.getHeight(), 0, 0,
                        Color.BLACK, 0, Color.BLACK, bg);
                descriptionArea.setText(illusion.getDescription());
                return;
            }
        }
    }



    @FXML
    private void onClear() {
        illusionSelector.getSelectionModel().clearSelection();
        descriptionArea.setText("Оберіть ілюзію з переліку.");
        illusionSettingsPane.getChildren().clear();
        drawPreview();
    }

    @FXML
    private void onExport() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Зберегти ілюзію");
        FileChooser.ExtensionFilter pngFilter = new
FileChooser.ExtensionFilter("PNG Image (*.png)", "*.png");
        FileChooser.ExtensionFilter jpgFilter = new
FileChooser.ExtensionFilter("JPEG Image (*.jpg)", "*.jpg");
        fileChooser.getExtensionFilters().addAll(pngFilter, jpgFilter);
        fileChooser.setSelectedExtensionFilter(pngFilter);
        fileChooser.setInitialFileName("illusion");
```

```
        File file = fileChooser.showSaveDialog(canvas.getScene().getWindow());
        if (file != null) {
            try {
                double width = canvas.getWidth();
                double height = canvas.getHeight();
                String extension = getFileExtension(file);
                if (extension == null) extension = "png";
                WritableImage image = new WritableImage((int) width, (int)
height);
                Canvas exportCanvas = new Canvas(width, height);
                GraphicsContext gc = exportCanvas.getGraphicsContext2D();
                String selected = illusionSelector.getValue();
                if (selected == null || selected.isEmpty()) return;
                Color bgColor = extension.equalsIgnoreCase("png")
                        ? Color.TRANSPARENT
                        : backgroundColorPicker.getValue();
                for (OpticalIllusion illusion : illusions) {
                    if (illusion.getName().equals(selected)) {
                        illusion.draw(gc, width, height,
                                0, 0, Color.BLACK, 0, Color.BLACK,
                                bgColor);
                        break;
                    }
                }
                SnapshotParameters params = new SnapshotParameters();
                params.setFill(bgColor);
                exportCanvas.snapshot(params, image);
                if (extension.equalsIgnoreCase("jpg") ||
extension.equalsIgnoreCase("jpeg")) {
                    BufferedImage fxImage = SwingFXUtils.fromFXImage(image,
null);
                    BufferedImage rgbImage = new BufferedImage((int) width,
(int) height, BufferedImage.TYPE_INT_RGB);
                    Graphics2D g2d = rgbImage.createGraphics();

                    java.awt.Color awtColor = new java.awt.Color(
                            (float) bgColor.getRed(),
                            (float) bgColor.getGreen(),
                            (float) bgColor.getBlue()
                    );
                    g2d.setColor(awtColor);
                    g2d.fillRect(0, 0, rgbImage.getWidth(),
rgbImage.getHeight());
                    g2d.drawImage(fxImage, 0, 0, null);
                    g2d.dispose();
                    ImageIO.write(rgbImage, "jpg", file);
                } else {
                    ImageIO.write(SwingFXUtils.fromFXImage(image, null), "png",
file);
                }
                System.out.println("Збережено: " + file.getAbsolutePath());

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private String getFileExtension(File file) {
        String name = file.getName();
        int dotIndex = name.lastIndexOf('.');
        return (dotIndex > 0) ? name.substring(dotIndex + 1).toLowerCase() :
null;
    }
```

```
    private String getExtension(FileChooser.ExtensionFilter filter) {
        String description = filter.getDescription();
        if (description.contains("*.png")) return ".png";
        if (description.contains("*.jpg")) return ".jpg";
        return ".jpg";
    }



    @FXML
    private void onResetIllusion() {
        String selected = illusionSelector.getValue();
        if (selected == null) return;

        for (OpticalIllusion illusion : illusions) {
            if (illusion.getName().equals(selected)) {
                illusion.resetParameters();
                drawPreview();
                return;
            }
        }
    }

    @FXML
    private void onShowHelp() {
        String selected = illusionSelector.getValue();
        for (OpticalIllusion illusion : illusions) {
            if (illusion.getName().equals(selected)) {
                showHelpWindow(illusion.getName(), illusion.getHelpText());
                return;
            }
        }
    }

    private void showHelpWindow(String title, String content) {
        TextArea textArea = new TextArea(content);
        textArea.setWrapText(true);
        textArea.setEditable(false);
        textArea.setPrefWidth(600);
        textArea.setPrefHeight(400);
        textArea.setMaxWidth(Double.MAX_VALUE);
        textArea.setMaxHeight(Double.MAX_VALUE);

        ScrollPane scrollPane = new ScrollPane(textArea);
        scrollPane.setFitToWidth(true);
        scrollPane.setFitToHeight(true);

        Dialog<Void> dialog = new Dialog<>();
        dialog.setTitle("Довідка: " + title);
        dialog.getDialogPane().setContent(scrollPane);

        ButtonType closeButtonType = new ButtonType("Закрити",
ButtonBar.ButtonData.CANCEL_CLOSE);
        dialog.getDialogPane().getButtonTypes().add(closeButtonType);

        dialog.getDialogPane().setPrefSize(640, 480);
        dialog.getDialogPane().setMinSize(400, 300);
        dialog.showAndWait();
    }
}
```

CafeWallIllusion.java:

```
package com.example.illusion;

import javafx.scene.Node;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.ColorPicker;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;

public class CafeWallIllusion extends OpticalIllusion {

    private final Slider tileSizeSlider = new Slider(10, 100, 40);
    private final Slider offsetSlider = new Slider(0, 40, 10);
    private final ColorPicker color1Picker = new ColorPicker(Color.BLACK);
    private final ColorPicker color2Picker = new ColorPicker(Color.WHITE);

    private final VBox settingsPane;

    public CafeWallIllusion() {
        tileSizeSlider.setShowTickLabels(true);
        offsetSlider.setShowTickLabels(true);

        settingsPane = new VBox(10,
                new Label("Розмір плитки"), tileSizeSlider,
                new Label("Зміщення рядів"), offsetSlider,
                new Label("Колір 1"), color1Picker,
                new Label("Колір 2"), color2Picker
        );

        tileSizeSlider.valueProperty().addListener((obs, o, n) -> fireChange());
        offsetSlider.valueProperty().addListener((obs, o, n) -> fireChange());
        color1Picker.valueProperty().addListener((obs, o, n) -> fireChange());
        color2Picker.valueProperty().addListener((obs, o, n) -> fireChange());
    }

    @Override
    public IllusionType getType() {
        return IllusionType.STATIC;
    }
    @Override
    public String getName() {
        return "Café Wall";
    }

    @Override
    public String getDescription() {
        return "Оптична ілюзія: ряди виглядають нахиленими, хоча вони прямі.";
    }

    @Override
    public Node getSettingsPane() {
        return settingsPane;
    }

    @Override
    public void draw(GraphicsContext gc,
                     double width,
                     double height,
                     double elementSize,
                     double lineWidth,
```

```
                    Color lineColor,
                    double illusionLineWidth,
                    Color illusionLineColor,
                    Color backgroundColor) {

        gc.setFill(backgroundColor);
        gc.fillRect(0, 0, width, height);

        int tileSize = (int) tileSizeSlider.getValue();
        int offset = (int) offsetSlider.getValue();
        Color color1 = color1Picker.getValue();
        Color color2 = color2Picker.getValue();

        int rows = (int) (height / tileSize);
        int cols = (int) (width / tileSize) + 2;

        for (int row = 0; row < rows; row++) {
            boolean even = row % 2 == 0;
            double y = row * tileSize;
            double rowOffset = even ? 0 : offset;

            for (int col = 0; col < cols; col++) {
                double x = col * tileSize - rowOffset;
                gc.setFill((col % 2 == 0) ? color1 : color2);
                gc.fillRect(x, y, tileSize, tileSize);
            }
        }

        gc.setStroke(Color.GRAY);
        gc.setLineWidth(2);
        for (int row = 0; row <= rows; row++) {
            double y = row * tileSize;
            gc.strokeLine(0, y, width, y);
        }
    }

    @Override
    public void resetParameters() {
        tileSizeSlider.setValue(40);
        offsetSlider.setValue(10);
        color1Picker.setValue(Color.BLACK);
        color2Picker.setValue(Color.WHITE);
    }

    @Override
    public String getHelpText() {
        return """
    ❖ Ілюзія Кафе-стіни (Café Wall Illusion)

    Ілюзія Кафе-стіни — це оптичне явище, при якому ряди рівних плиток,
    викладених у шаховому порядку, здаються викривленими або нахиленими, хоча
насправді вони ідеально прямі.

    ✦ Походження:
    Вперше зафіксована у 1970-х роках Річардом Грегорі — британським
нейропсихологом, який помітив
    ілюзію на стіні одного кафе в Бристолі (Англія). Звідси й назва — "Café
Wall".

    ✦ Принцип дії:
    Ілюзія створюється за рахунок контрасту кольорів (зазвичай чорні й білі
плитки) та зміщення
    кожного другого ряду на частину плитки. Між рядами додаються тонкі розділові
```

```
лінії (іноді сірі),
    які сприяють створенню візуального викривлення.

    Мозок намагається інтерпретувати зображення як тривимірну структуру з
глибинами або кутами,
    хоча насправді це просто пласке зображення.

    ✦ Пояснення:
    Сприйняття нахилу є результатом взаємодії між краями плиток і сіткою, яка
формує проміжки між ними.
    Зорові нейрони, які відповідають за виявлення нахилу та контурів,
"перепрацьовують" інформацію,
    створюючи ілюзію хвиль або нахилів.

    ✦ Налаштування в застосунку:
    • Розмір плитки — визначає ширину і висоту кожного блоку (чим більше, тим
сильніший ефект).
    • Зміщення рядів — наскільки зсунуті непарні ряди відносно парних.
    • Колір 1 — перший основний колір плитки (наприклад, чорний).
    • Колір 2 — другий колір плитки (наприклад, білий).

    ✦ Цікаві факти:
    • Ілюзія працює навіть без контрастних кольорів, хоча ефект слабшає.
    • Візуальне викривлення посилюється за умови сильного контрасту між плитками
та проміжками.
    • Часто використовується в демонстраціях когнітивного сприйняття в школах та
наукових музеях.

    ✦ Практичне застосування:
    Ця ілюзія є чудовим прикладом для дослідження контекстуального впливу на
сприйняття та
    використовується в психології, дизайні й візуальній ергономіці для
тестування сприйняття форм.

    """;
    }

}
```

ChoppyRotationIllusion.java:

```
package com.example.illusion;

import javafx.animation.AnimationTimer;
import javafx.scene.Node;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;

public class ChoppyRotationIllusion extends OpticalIllusion {

    private final Slider sectorCountSlider = new Slider(4, 40, 12);
    private final Slider stepDelaySlider = new Slider(100, 1000, 300); // мс між
кроками
    private final ColorPicker color1Picker = new ColorPicker(Color.DARKBLUE);
    private final ColorPicker color2Picker = new ColorPicker(Color.LIGHTGRAY);
    private final Button startButton = new Button("▶ Старт");
    private final Button stopButton = new Button("■ Стоп");
```

```
    private final VBox settingsPane;

    private int currentStep = 0;
    private long lastStepTime = 0;
    private boolean running = false;
    private AnimationTimer timer;

    public ChoppyRotationIllusion() {
        sectorCountSlider.setShowTickLabels(true);
        stepDelaySlider.setShowTickLabels(true);
        stepDelaySlider.setShowTickMarks(true);

        settingsPane = new VBox(10,
                new Label("Кількість секторів"), sectorCountSlider,
                new Label("Затримка між кроками (мс)"), stepDelaySlider,
                new Label("Колір 1"), color1Picker,
                new Label("Колір 2"), color2Picker,
                new HBox(10, startButton, stopButton)
        );

        startButton.setOnAction(e -> start());
        stopButton.setOnAction(e -> stop());

        sectorCountSlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        stepDelaySlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        color1Picker.valueProperty().addListener((obs, o, n) -> fireChange());
        color2Picker.valueProperty().addListener((obs, o, n) -> fireChange());

        initTimer();
    }

    private void initTimer() {
        timer = new AnimationTimer() {
            @Override
            public void handle(long now) {
                long currentTime = System.currentTimeMillis();
                long delay = (long) stepDelaySlider.getValue();
                if (currentTime - lastStepTime > delay) {
                    currentStep++;
                    lastStepTime = currentTime;
                    fireChange();
                }
            }
        };
    }

    private void start() {
        if (!running) {
            running = true;
            lastStepTime = System.currentTimeMillis();
            timer.start();
        }
    }

    private void stop() {
        running = false;
        timer.stop();
    }

    @Override
    public String getName() {
```

```
        return "Choppy Rotation (Stepping Circle)";
    }

    @Override
    public String getDescription() {
        return "Ілюзія обертання, де сектори змінюють положення поштовхами,
створюючи ефект 'ступінчатого' руху.";
    }

    @Override
    public IllusionType getType() {
        return IllusionType.ANIMATED;
    }

    @Override
    public Node getSettingsPane() {
        return settingsPane;
    }

    @Override
    public void draw(GraphicsContext gc,
                     double width,
                     double height,
                     double elementSize,
                     double lineWidth,
                     Color lineColor,
                     double illusionLineWidth,
                     Color illusionLineColor,
                     Color backgroundColor) {
        gc.setFill(backgroundColor);
        gc.fillRect(0, 0, width, height);

        int sectors = (int) sectorCountSlider.getValue();
        double anglePerSector = 360.0 / sectors;
        double radius = Math.min(width, height) / 2 - 20;
        double cx = width / 2;
        double cy = height / 2;

        for (int i = 0; i < sectors; i++) {
            double angle = anglePerSector * i + currentStep * anglePerSector;
            Color fill = (i % 2 == 0) ? color1Picker.getValue() :
color2Picker.getValue();
            gc.setFill(fill);
            gc.beginPath();
            gc.moveTo(cx, cy);
            gc.arc(cx, cy, radius, radius, angle, anglePerSector);
            gc.closePath();
            gc.fill();
        }
    }

    @Override
    public void resetParameters() {
        sectorCountSlider.setValue(12);
        stepDelaySlider.setValue(300);
        color1Picker.setValue(Color.DARKBLUE);
        color2Picker.setValue(Color.LIGHTGRAY);
    }

    @Override
    public String getHelpText() {
        return """
            ❖ Ілюзія Поштовхового Обертання (Choppy Rotation / Stepping Circle)
```

```
        Ця ілюзія створює ефект обертання круга, хоча сектори змінюють положення
        ривками (поштовхами), що створює незвичне відчуття дискретного руху.

        ✦ Принцип дії:
        Через зміну фаз обертання по фіксованим крокам (а не плавно), мозок
        інтерпретує зображення як обертання з "перескакуванням".

        ✦ Пояснення:
        Поштовхи в положенні секторів провокують ілюзію дискретного обертання,
        хоча жодного плавного руху насправді не відбувається.

        ✦ Налаштування в застосунку:
        • Кількість секторів — на скільки частин поділено коло.
        • Затримка між кроками — час між кожним обертанням на один сектор.
        • Колір 1 і Колір 2 — кольори чергуючих секторів.
        • Кнопки ▶ Старт і ■ Стоп — для керування анімацією.

        ✦ Цікаві факти:
        • Цей тип ілюзії використовується для тестування сприйняття часу та ритму.
        • Аналогічні ефекти можна зустріти в ігрових інтерфейсах та UX-анімаціях.

        ✦ Практичне застосування:
        • Вивчення часової обробки візуальної інформації.
        • Декоративні або гіпнотичні візуальні ефекти в мультимедіа.
                """;
        }
}
```

FraserSpiralIllusion.java:

```
package com.example.illusion;

import javafx.scene.Node;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;

public class FraserSpiralIllusion extends OpticalIllusion {

    private final Slider ringsSlider = new Slider(10, 100, 40);
    private final Slider sectorsSlider = new Slider(6, 60, 20);
    private final Slider twistSlider = new Slider(0, 2, 0.5);
    private final ColorPicker color1Picker = new ColorPicker(Color.BLACK);
    private final ColorPicker color2Picker = new ColorPicker(Color.WHITE);

    private final VBox settingsPane;

    public FraserSpiralIllusion() {
        ringsSlider.setShowTickLabels(true);
        sectorsSlider.setShowTickLabels(true);
        twistSlider.setShowTickLabels(true);

        settingsPane = new VBox(10,
                new Label("Кількість кіл"), ringsSlider,
                new Label("Кількість секторів"), sectorsSlider,
                new Label("Коефіцієнт спіральності"), twistSlider,
                new Label("Колір 1"), color1Picker,
                new Label("Колір 2"), color2Picker
        );
```

```
        ringsSlider.valueProperty().addListener((obs, o, n) -> fireChange());
        sectorsSlider.valueProperty().addListener((obs, o, n) -> fireChange());
        twistSlider.valueProperty().addListener((obs, o, n) -> fireChange());
        color1Picker.valueProperty().addListener((obs, o, n) -> fireChange());
        color2Picker.valueProperty().addListener((obs, o, n) -> fireChange());
    }

    @Override
    public String getName() {
        return "Fraser Spiral Illusion";
    }

    @Override
    public String getDescription() {
        return "Фрейзерова спіраль: концентричні кола здаються спіраллю через
викривлення сприйняття.";
    }

    @Override
    public IllusionType getType() {
        return IllusionType.STATIC;
    }

    @Override
    public Node getSettingsPane() {
        return settingsPane;
    }

    @Override
    public void draw(GraphicsContext gc, double width, double height, double
elementSize,
                     double lineWidth,
                     Color lineColor, double illusionLineWidth, Color
illusionLineColor,
                     Color backgroundColor) {
        double centerX = width / 2;
        double centerY = height / 2;
        int rings = (int) ringsSlider.getValue();
        int sectors = (int) sectorsSlider.getValue();
        double twist = twistSlider.getValue();
        Color color1 = color1Picker.getValue();
        Color color2 = color2Picker.getValue();
        gc.fillRect(0, 0, width, height);
        double maxRadius = Math.min(width, height) * 0.45;
        double ringStep = maxRadius / rings;
        for (int r = 0; r < rings; r++) {
            double innerR = r * ringStep;
            double outerR = (r + 1) * ringStep;
            for (int s = 0; s < sectors; s++) {
                double angle = 360.0 / sectors * s + r * twist * 360.0 /
sectors;
                double arc = 360.0 / sectors;
                gc.setFill((s % 2 == 0) ? color1 : color2);
                gc.beginPath();
                gc.arc(centerX, centerY, outerR, outerR, angle, arc);
                gc.arc(centerX, centerY, innerR, innerR, angle + arc, -arc);
                gc.closePath();
                gc.fill();
            }
        }
    }

    @Override
    public void resetParameters() {
```

```
        ringsSlider.setValue(40);
        sectorsSlider.setValue(20);
        twistSlider.setValue(0.5);
        color1Picker.setValue(Color.BLACK);
        color2Picker.setValue(Color.WHITE);
    }

    @Override
    public String getHelpText() {
        return """
 ❖ Ілюзія Фрейзера (Fraser Spiral Illusion)

Ця оптична ілюзія створює враження безперервної спіралі, хоча насправді
складається з концентричних кіл із секторів.

✦ Принцип дії:
Через зміщення кутів секторів у кожному кільці, сприйняття викривлюється, і кола
здаються спіраллю.

✦ Пояснення:
Мозок намагається об’єднати викривлені елементи в єдину фігуру, інтерпретуючи їх
як безперервну спіраль.

✦ Налаштування в застосунку:
• Кількість кіл — кількість концентричних рядів.
• Кількість секторів — кількість розділів у кожному кільці.
• Коефіцієнт спіральності — рівень фазового зсуву між кільцями.
• Колір 1 та Колір 2 — чергування кольорів секторів.
• Колір фону — базовий колір сцени.

✦ Цікаві факти:
• Вперше описана британським психологом Джеймсом Фрейзером у 1908 році.
• Схожий ефект використовується в мистецтві та графічному дизайні.

✦ Практичне застосування:
• Використовується для вивчення візуального сприйняття форм і руху.
• Часто застосовується в інтерактивних візуалізаціях та демонстраціях ілюзій.
        """;
    }
}
```

HeringIllusion.java:

```
package com.example.illusion;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.Node;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;

public class HeringIllusion extends OpticalIllusion {

    private final Slider lineCountSlider = new Slider(10, 100, 40);
    private final Slider raysThicknessSlider = new Slider(0.5, 10, 2);
    private final ColorPicker raysColorPicker = new ColorPicker(Color.GRAY);
    private final Slider illusionLineWidthSlider = new Slider(1, 10, 3);
    private final ColorPicker illusionLineColorPicker = new
ColorPicker(Color.RED);

    private final VBox settingsPane;
```

```
    public HeringIllusion() {
        lineCountSlider.setShowTickLabels(true);
        raysThicknessSlider.setShowTickLabels(true);
        illusionLineWidthSlider.setShowTickLabels(true);

        settingsPane = new VBox(10,
                new Label("Кількість променів"), lineCountSlider,
                new Label("Товщина променів"), raysThicknessSlider,
                new Label("Колір променів"), raysColorPicker,
                new Label("Товщина червоних ліній"), illusionLineWidthSlider,
                new Label("Колір червоних ліній"), illusionLineColorPicker
        );

        lineCountSlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        raysThicknessSlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        raysColorPicker.valueProperty().addListener((obs, o, n) ->
fireChange());
        illusionLineWidthSlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        illusionLineColorPicker.valueProperty().addListener((obs, o, n) ->
fireChange());
    }

    @Override
    public IllusionType getType() {
        return IllusionType.STATIC;
    }
    @Override
    public String getName() {
        return "Hering Illusion";
    }
    @Override
    public String getDescription() {
        return "Ілюзія Герінга: прямі здаються вигнутими через фон із
променів.";
    }
    @Override
    public Node getSettingsPane() {
        return settingsPane;
    }

    @Override
    public void draw(GraphicsContext gc,
                     double width,
                     double height,
                     double elementSize,
                     double lineWidth,
                     Color lineColor,
                     double illusionLineWidth,
                     Color illusionLineColor,
                     Color backgroundColor) {
        gc.setFill(backgroundColor);
        gc.fillRect(0, 0, width, height);
        gc.setStroke(raysColorPicker.getValue());
        gc.setLineWidth(raysThicknessSlider.getValue());
        int lines = (int) lineCountSlider.getValue();
        double centerX = width / 2;
        double centerY = height / 2;
        for (int i = 0; i < lines; i++) {
            double angle = Math.toRadians((360.0 / lines) * i);
            double x = centerX + Math.cos(angle) * width;
```

```
            double y = centerY + Math.sin(angle) * height;
            gc.strokeLine(centerX, centerY, x, y);
        }
        gc.setStroke(illusionLineColorPicker.getValue());
        gc.setLineWidth(illusionLineWidthSlider.getValue());
        gc.strokeLine(0, centerY - 50, width, centerY - 50);
        gc.strokeLine(0, centerY + 50, width, centerY + 50);


    }
    @Override
    public void resetParameters() {
        lineCountSlider.setValue(40);
        raysThicknessSlider.setValue(2);
        raysColorPicker.setValue(Color.GRAY);
        illusionLineWidthSlider.setValue(3);
        illusionLineColorPicker.setValue(Color.RED);
    }

    @Override
    public String getHelpText() {
        return """
    ❖ Ілюзія Герінга (Hering Illusion)

    Ілюзія Герінга — це класичне оптичне явище, при якому дві прямі
горизонтальні лінії здаються
    викривленими назовні (випуклими), коли на фоні зображено віялоподібний
візерунок з променів,
    які виходять із однієї точки або центру зображення.

    ✦ Історія:
    Ілюзію вперше описав німецький фізіолог Евальд Герінг у 1861 році. Його
дослідження стосувалися
    сприйняття глибини і просторового викривлення. Це явище стало одним із
найвідоміших прикладів
    того, як фон здатен спотворювати наше візуальне сприйняття простих форм.

    ✦ Принцип дії:
    В основі ілюзії лежить взаємодія між фоном (променями, які сходяться в
центрі) та основними лініями
    (двома горизонтальними). Мозок сприймає промені як ознаку перспективи —
глибини простору,
    схожої на залізничні колії, що сходяться на горизонті. Через це прямі лінії
здаються вигнутими,
    ніби вони прогинаються під впливом цього глибокого простору.

    ✦ Пояснення з точки зору нейропсихології:
    Зорові системи мозку обробляють контекстні підказки з фону, що приводить до
неправильного
    сприйняття напрямку ліній. Це демонструє, як зорове сприйняття не є
абсолютно об'єктивним
    — воно тісно пов'язане з інтерпретацією простору, яку здійснює мозок.

    ✦ Налаштування в застосунку:
    • Кількість променів — збільшує або зменшує кількість ліній фону, що йдуть
із центру.
    • Товщина променів — змінює візуальну вагу фону (чим товстіші, тим сильніший
ефект).
    • Колір променів — вибір кольору для віялоподібного фону.
    • Товщина червоних ліній — задає товщину двох горизонтальних ліній, що
створюють ефект ілюзії.
    • Колір червоних ліній — дозволяє налаштувати колір основного елемента
ілюзії.
```

```
    ✦ Практичне застосування:
    Цей тип ілюзії часто використовується у когнітивних науках, психології та
тестах на сприйняття.
    Вона ілюструє важливість контексту при аналізі інформації візуальною
системою людини.

    ✦ Цікаві факти:
    • Подібна ілюзія виникає і в інших типах фонового контрасту (наприклад, з
концентричними колами).
    • Дослідження показують, що інтенсивність ефекту залежить від кута нахилу
променів та
      їхнього кольору відносно центральних ліній.

    """;
    }
}
```

IllusorySquareMotionIllusion.java:

```
package com.example.illusion;

import javafx.animation.AnimationTimer;
import javafx.scene.Node;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;

import java.util.Random;

public class IllusorySquareMotionIllusion extends OpticalIllusion {

    private final Slider squareCountSlider = new Slider(4, 100, 30);
    private final Slider squareSizeSlider = new Slider(10, 80, 30);
    private final ColorPicker color1Picker = new ColorPicker(Color.BLACK);
    private final ColorPicker color2Picker = new ColorPicker(Color.WHITE);
    private final Button startButton = new Button("▶ Старт");
    private final Button stopButton = new Button("■ Стоп");
    private final VBox settingsPane;

    private final Random random = new Random();
    private boolean running = false;
    private AnimationTimer timer;

    public IllusorySquareMotionIllusion() {
        squareCountSlider.setShowTickLabels(true);
        squareSizeSlider.setShowTickLabels(true);

        settingsPane = new VBox(10,
                new Label("Кількість квадратів"), squareCountSlider,
                new Label("Розмір квадратів"), squareSizeSlider,
                new Label("Колір 1"), color1Picker,
                new Label("Колір 2"), color2Picker,
                new HBox(10, startButton, stopButton)
        );

        startButton.setOnAction(e -> startAnimation());
        stopButton.setOnAction(e -> stopAnimation());

        squareCountSlider.valueProperty().addListener((obs, o, n) ->
```

```
fireChange());
        squareSizeSlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        color1Picker.valueProperty().addListener((obs, o, n) -> fireChange());
        color2Picker.valueProperty().addListener((obs, o, n) -> fireChange());

        initTimer();
    }

    private void initTimer() {
        timer = new AnimationTimer() {
            @Override
            public void handle(long now) {
                fireChange();
            }
        };
    }

    private void startAnimation() {
        running = true;
        timer.start();
    }

    private void stopAnimation() {
        running = false;
        timer.stop();
    }

    @Override
    public String getName() {
        return "Illusory Square Motion (Visual Jitter)";
    }

    @Override
    public String getDescription() {
        return "Ілюзія мерехтливих квадратів, що створюють враження хаотичного
руху.";
    }

    @Override
    public IllusionType getType() {
        return IllusionType.ANIMATED;
    }

    @Override
    public Node getSettingsPane() {
        return settingsPane;
    }

    @Override
    public void resetParameters() {
        squareCountSlider.setValue(30);
        squareSizeSlider.setValue(30);
        color1Picker.setValue(Color.BLACK);
        color2Picker.setValue(Color.WHITE);
    }

    @Override
    public String getHelpText() {
        return """
        ❖ Illusory Square Motion (Visual Jitter)

        Ілюзія створює враження, що статичні квадрати рухаються або тремтять,
хоча
```

```
        вони не змінюють позицію. Це досягається за рахунок хаотичного
мерехтіння кольору.

        ✦ Принцип дії:
        Мозок сприймає швидке чергування кольорів у випадковому порядку як рух,
        хоча координати об'єктів не змінюються.

        ✦ Пояснення:
        Це приклад ілюзії руху, викликаної лише чергуванням контрасту та
позиційної нестабільності.

        ✦ Налаштування в застосунку:
        • Кількість квадратів — визначає щільність сітки.
        • Розмір квадратів — встановлює розміри елементів.
        • Колір 1 та Колір 2 — чергуються для створення ефекту мерехтіння.
        • Кнопки ▶ Старт і ■ Стоп — керують анімацією.

        ✦ Практичне застосування:
        • Вивчення візуальної нестабільності.
        • Демонстрації в когнітивних науках та психології.
        """;
    }

    @Override
    public void draw(GraphicsContext gc, double width, double height, double
elementSize,
                     double lineWidth, Color lineColor, double
illusionLineWidth,
                     Color illusionLineColor, Color backgroundColor) {
        gc.setFill(backgroundColor);
        gc.fillRect(0, 0, width, height);

        int count = (int) squareCountSlider.getValue();
        int size = (int) squareSizeSlider.getValue();

        for (int i = 0; i < count; i++) {
            double x = random.nextDouble() * (width - size);
            double y = random.nextDouble() * (height - size);
            gc.setFill(random.nextBoolean() ? color1Picker.getValue() :
color2Picker.getValue());
            gc.fillRect(x, y, size, size);
        }
    }
}
```

MotionBindingIllusion.java:

```
package com.example.illusion;

import javafx.animation.AnimationTimer;
import javafx.scene.Node;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;

public class MotionBindingIllusion extends OpticalIllusion {

    private final Slider lineCountSlider = new Slider(2, 20, 10);
    private final Slider verticalSpeedSlider = new Slider(0.5, 5, 1);
```

```
    private final Slider horizontalSpeedSlider = new Slider(0.5, 5, 1);
    private final Slider dotRadiusSlider = new Slider(2, 10, 4);
    private final ColorPicker dotColorPicker = new ColorPicker(Color.RED);

    private final VBox settingsPane;
    private final Button startButton = new Button("▶ Старт");
    private final Button stopButton = new Button("■ Стоп");

    private boolean running = false;
    private AnimationTimer timer;
    private double phase = 0;
    private double offsetX = 0;

    public MotionBindingIllusion() {
        lineCountSlider.setShowTickLabels(true);
        verticalSpeedSlider.setShowTickLabels(true);
        horizontalSpeedSlider.setShowTickLabels(true);
        dotRadiusSlider.setShowTickLabels(true);

        settingsPane = new VBox(10,
                new Label("Кількість ліній"), lineCountSlider,
                new Label("Швидкість вертикального руху"), verticalSpeedSlider,
                new Label("Швидкість горизонтального руху"),
horizontalSpeedSlider,
                new Label("Розмір точки"), dotRadiusSlider,
                new Label("Колір точки"), dotColorPicker,
                new HBox(10, startButton, stopButton)
        );
        startButton.setOnAction(e -> startAnimation());
        stopButton.setOnAction(e -> stopAnimation());

        lineCountSlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        verticalSpeedSlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        horizontalSpeedSlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        dotRadiusSlider.valueProperty().addListener((obs, o, n) ->
fireChange());
        dotColorPicker.valueProperty().addListener((obs, o, n) -> fireChange());

        initTimer();
    }

    private void initTimer() {
        timer = new AnimationTimer() {
            @Override
            public void handle(long now) {
                phase += verticalSpeedSlider.getValue();
                offsetX += horizontalSpeedSlider.getValue();
                fireChange();
            }
        };
    }

    private void startAnimation() {
        if (!running) {
            running = true;
            timer.start();
        }
    }

    private void stopAnimation() {
```

```
        running = false;
        timer.stop();
    }

    @Override
    public String getName() {
        return "Motion Binding Illusion";
    }

    @Override
    public String getDescription() {
        return "Точки здаються прив’язаними до фону через комбінований рух.";
    }

    @Override
    public IllusionType getType() {
        return IllusionType.ANIMATED;
    }

    @Override
    public Node getSettingsPane() {
        return settingsPane;
    }

    @Override
    public void draw(GraphicsContext gc, double width, double height, double e,
double lw, Color lc, double ilw, Color ilc, Color bg) {
        gc.setFill(bg);
        gc.fillRect(0, 0, width, height);

        int lines = (int) lineCountSlider.getValue();
        double spacing = width / lines;
        double dotRadius = dotRadiusSlider.getValue();
        Color dotColor = dotColorPicker.getValue();

        for (int i = 0; i < lines; i++) {
            double x = i * spacing + spacing / 2 + offsetX % spacing;
            gc.setStroke(Color.GRAY);
            gc.setLineWidth(1);
            gc.strokeLine(x, 0, x, height);

            double y = height / 2 + Math.sin((i * 0.5 + phase * 0.05)) * (height
/ 3);
            gc.setFill(dotColor);
            gc.fillOval(x - dotRadius, y - dotRadius, dotRadius * 2, dotRadius *
2);
        }
    }

    @Override
    public void resetParameters() {
        lineCountSlider.setValue(10);
        verticalSpeedSlider.setValue(1);
        horizontalSpeedSlider.setValue(1);
        dotRadiusSlider.setValue(4);
        dotColorPicker.setValue(Color.RED);
        phase = 0;
        offsetX = 0;
    }

    @Override
    public String getHelpText() {
        return """
        ❖ Motion Binding Illusion
```

```
        У цій ілюзії рух точок виглядає пов’язаним із рухом усієї структури.
Коли точки рухаються
        вертикально, а вся група — горизонтально, здається, що точки слідують
траєкторії
        зсередини структури.

        ✦ Принцип дії:
        Візуальна система інтерпретує рух не окремо, а в контексті оточення.
Комбіновані
        траєкторії створюють ефект «зв’язаного руху».

        ✦ Пояснення:
        Мозок намагається об’єднати візуальні сигнали в єдину систему координат.
        Цей ефект вивчається в психології для розуміння механізмів зорової
інтеграції.

        ✦ Налаштування в застосунку:
        • Кількість вертикальних ліній — задає структуру.
        • Швидкість вертикального руху точок — визначає рух кожної точки.
        • Швидкість горизонтального руху структури — створює ілюзію прив’язки.
        • Розмір і колір точки — естетичні налаштування.

        ✦ Практичне застосування:
        • Аналіз зорових конфліктів.
        • Дослідження інтеграції сенсорної інформації в мозку.
        """;
    }
}
```

Додаток Г – Графічна частина

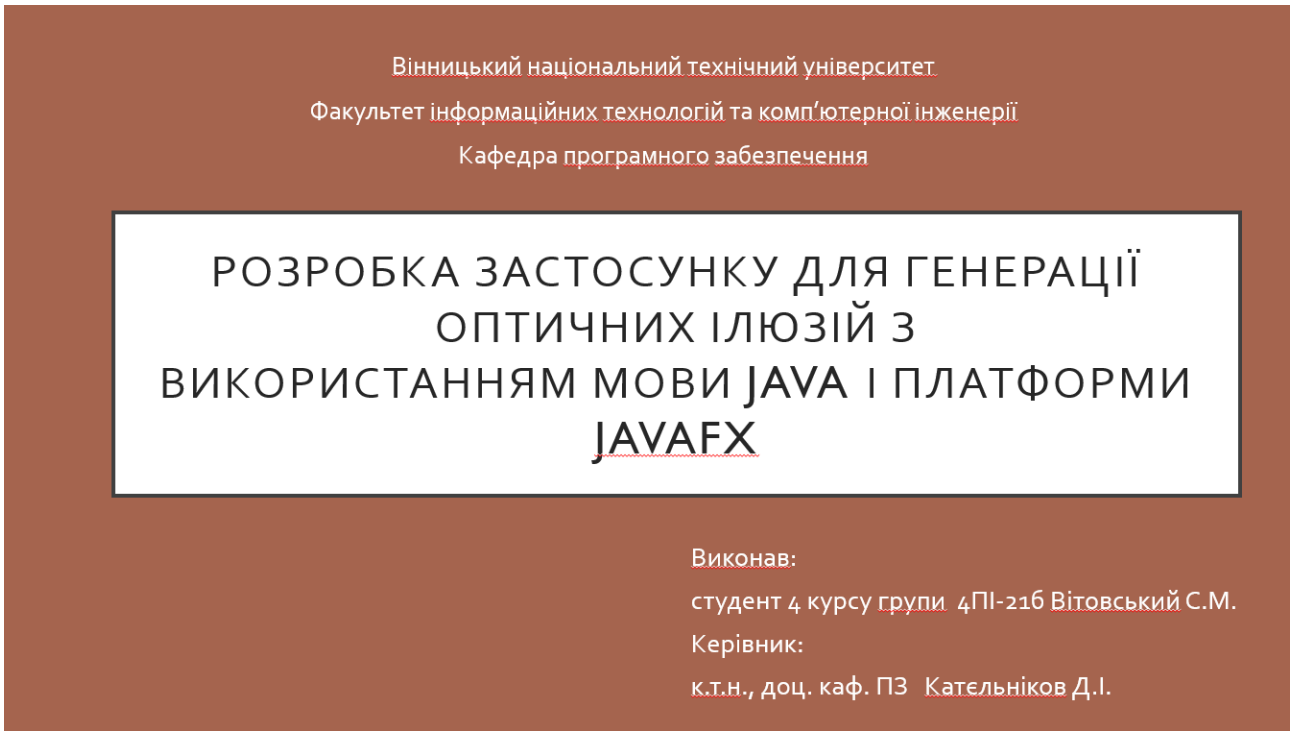

Рисунок Г.1 – Титульний слайд

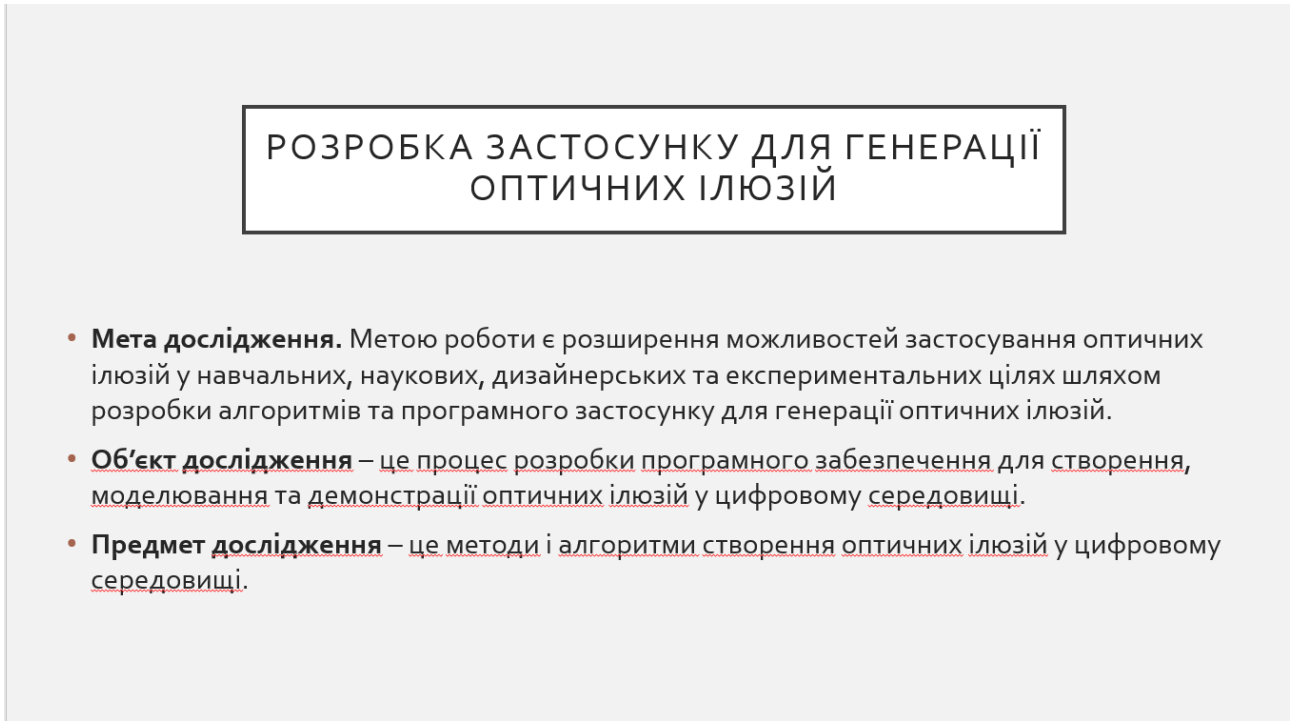

Рисунок Г.2 – Мета, об'єкт і предмет дослідження

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ГЕНЕРАЦІЇ ОПТИЧНИХ ІЛЮЗІЙ

- **Новизна отриманих результатів**.

  1. Подальшого розвитку отримав метод генерації оптичних ілюзій, який, на відміну від існуючих, поєднує динамічну візуалізацію з можливістю інтерактивного налаштування параметрів зображення, що дозволяє досягти адаптивного управління візуальними ефектами в реальному часі і, таким чином, зробити вплив більш персоніфікованим і поглибленим.

  2. Подальшого розвитку отримав метод створення програмного забезпечення для генерації оптичних ілюзій, який, на відміну від існуючих, використовує модульну архітектуру, що забезпечує гнучкість у розширенні функціоналу, оскільки дозволяє легко інтегрувати нові типи ілюзій без суттєвих змін базового коду.

  Практична цінність отриманих результатів. полягає у тому, що в результаті було розроблено програмний засіб, який дає змогу генерувати і досліджувати оптичні ілюзії у візуальній формі. Застосунок може бути використаний у закладах освіти під час вивчення фізіології зору, візуального мистецтва, інформатики, а також у наукових експериментах для аналізу реакцій на певні типи візуальних стимулів.

Рисунок Г.3 – Наукова новизна та практична цінність

ЗАДАЧІ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

- провести аналіз існуючих методів створення оптичних ілюзій та засобів візуального моделювання;
- розробити метод створення структури застосунку, який включає модулі генерації, візуалізації та збереження зображень;
- реалізувати метод генерації ілюзій із використанням графічних примітивів та ефектів JavaFX;
- створити графічний інтерфейс, що дозволяє користувачу взаємодіяти із застосунком у режимі реального часу;
- реалізувати функції експорту створених ілюзій та забезпечити зручну систему навігації інтерфейсом;
- протестувати функціональність застосунку та провести оцінку його ефективності й зручності.

Рисунок Г.4 – Задачі бакалаврської кваліфікаційної роботи

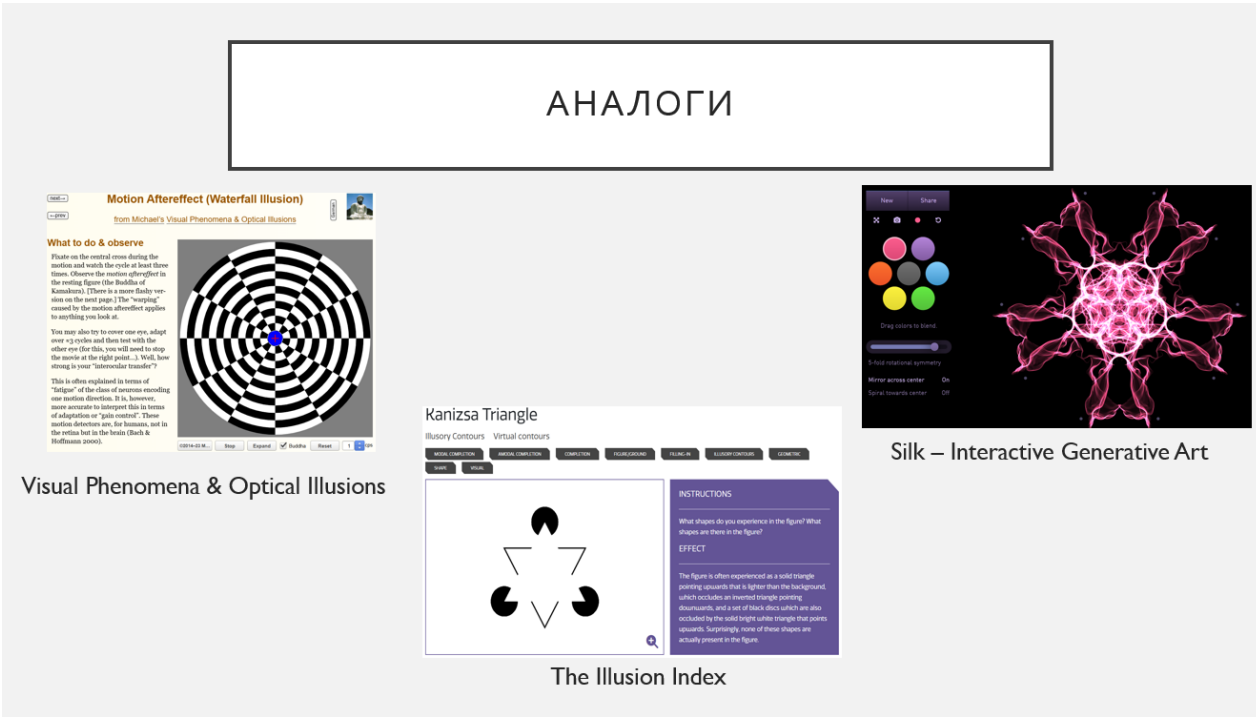

Рисунок Г.5 – Аналоги

Функціональні характеристики	Visual Phenomena & Optical Illusions	The Illusion Index	Silk – Interactive Generative Art	Власний застосунок
Генерація динамічних оптичних ілюзій	1	0	1	1
Можливість налаштування параметрів ілюзій (колір, форма, рух)	0	1	1	1
Збереження створених ілюзій у файл (PNG, JPEG)	0	0	0	1
Вбудоване пояснення принципу дії ілюзії	1	1	0	1
Підтримка як статичних, так і динамічних ілюзій в одному застосунку	1	0	1	1
Сумарний коефіцієнт	3	2	3	5

Рисунок Г.6 – Порівняльний аналіз аналогів

ДІАГРАМА ДІЯЛЬНОСТІ ПРОГРАМНОГО ЗАСТОСУНКУ

У процесі розробки програмного застосунку для генерації оптичних ілюзій було створено діаграму діяльності UML, яка демонструє послідовність дій користувача від моменту запуску застосунку до завершення сеансу з ілюзіями. Ця діаграма дозволяє структурувати логіку програми, визначити ключові блоки керування, виявити можливі ситуації, пов'язані з вибором або відсутністю вибору, та надати цілісне уявлення про цикл програми.

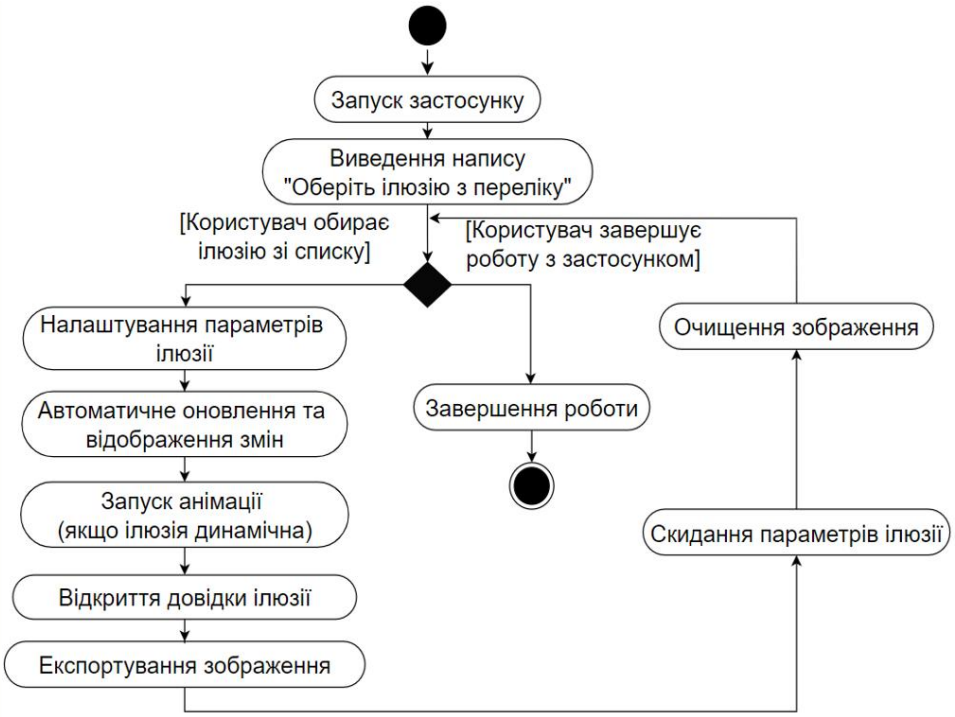

Рисунок Г.7 – Діаграма діяльності програмного застосунку

БЛОК-СХЕМА РОБОТИ ПРОГРАМНОГО ЗАСТОСУНКУ

Для реалізації взаємодії користувача з програмним застосунком для генерації оптичних ілюзій було створено блок-схему, яка відображає логіку роботи системи від початку до кінця сеансу.

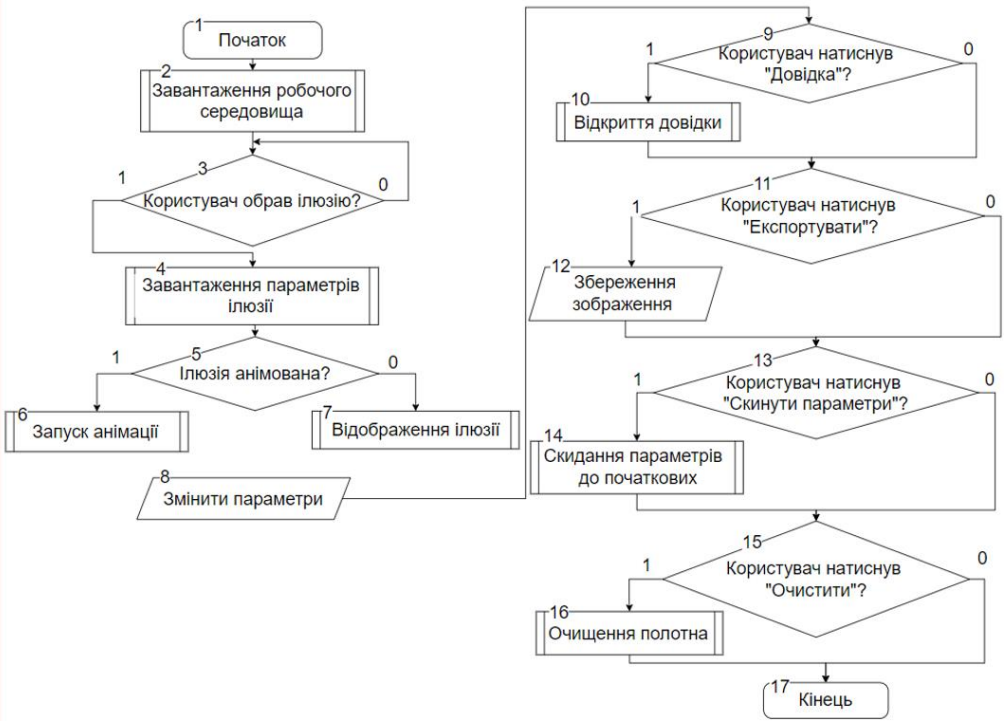

Рисунок Г.8 – Блок-схема роботи програмного застосунку

БЛОК-СХЕМА РОБОТИ МЕТОДУ DRAWPREVIEW()

Однією з основних функцій, що визначає поведінку інтерфейсу в програмі для генерації оптичних ілюзій, є метод drawPreview(), який відповідає за побудову зображення на полотні відповідно до вибору користувача. Цей метод викликається щоразу, коли користувач змінює значення у списку доступних ілюзій, та виконує кілька ключових дій: очищення полотна, вибір відповідної ілюзії, виклик її методу малювання, оновлення опису та очищення або оновлення панелі опцій.

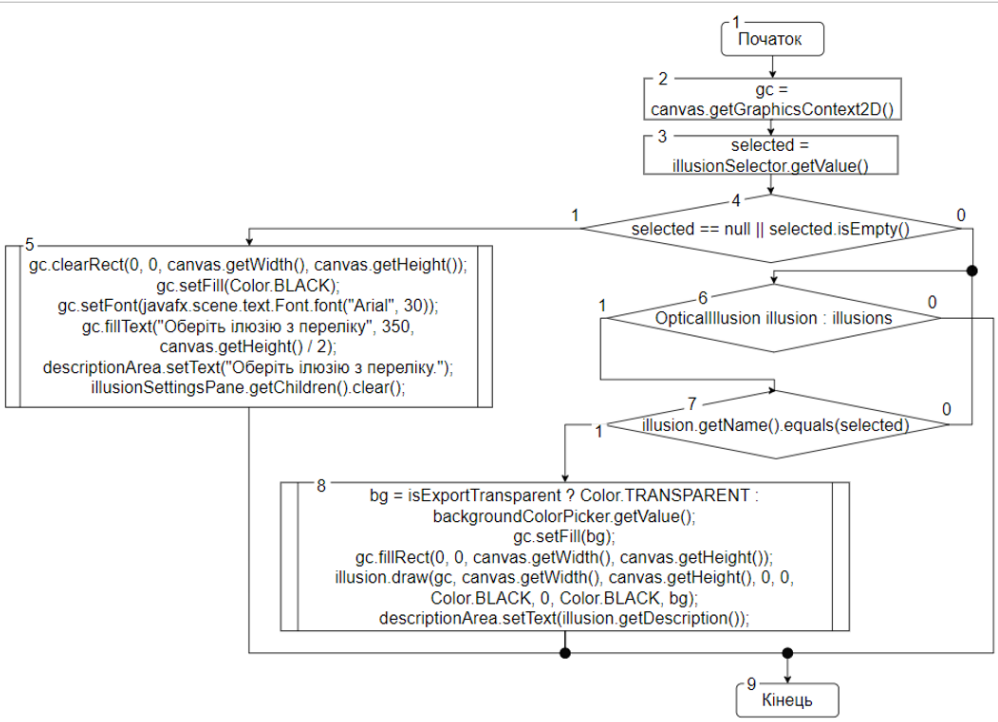

Рисунок Г.9 – Блок-схема роботи методу drawPreview()

ДІАГРАМА КЛАСУ OPTICALILLUSION

Для чіткої візуалізації структури класу та його зв'язків з підкласами була створена діаграма класів. Вона відображає абстрактні методи, типи повернення, поля класу та очікувані реалізації в нащадках. Така діаграма відіграє важливу роль у плануванні системи, дозволяє швидко зрозуміти необхідні точки розширення та забезпечує єдиний підхід до побудови модулів ілюзій.

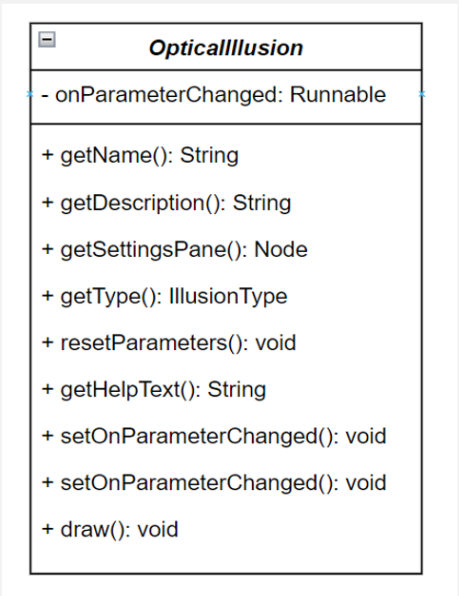

Рисунок Г.10 – Діаграма класу OpticalIllusion

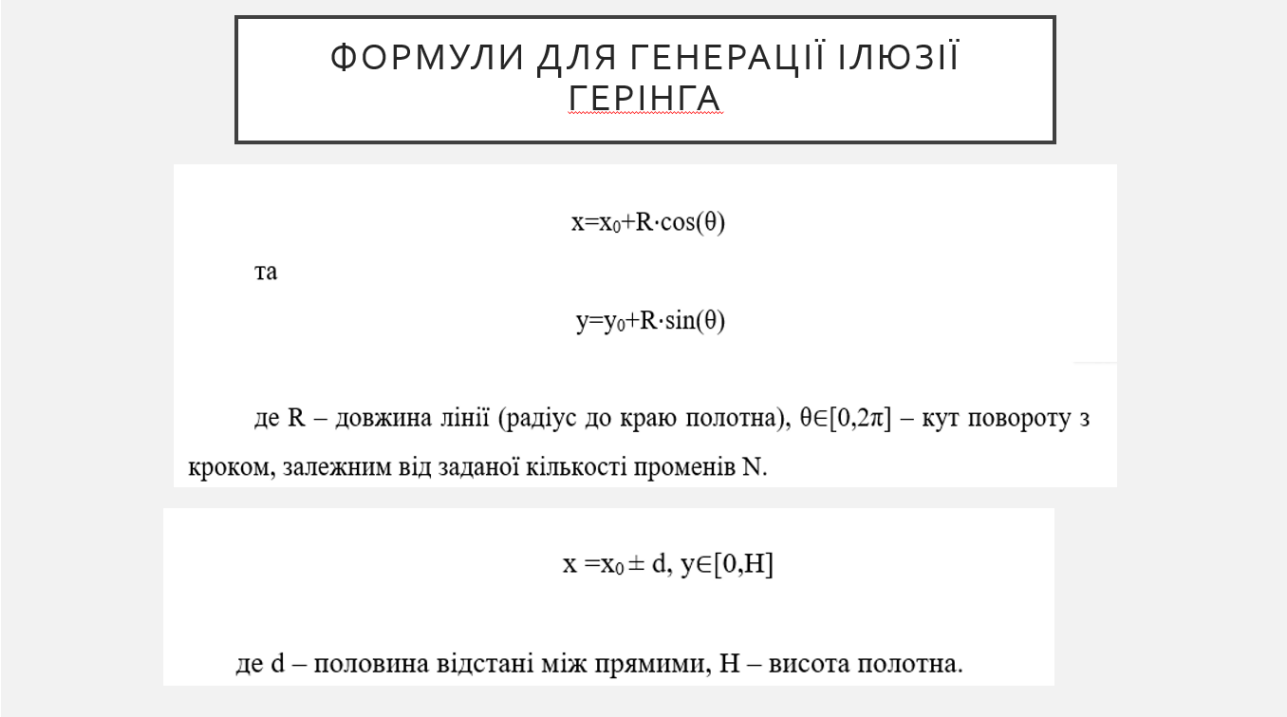

Рисунок Г.11 – Формули для генерації ілюзії Герінга

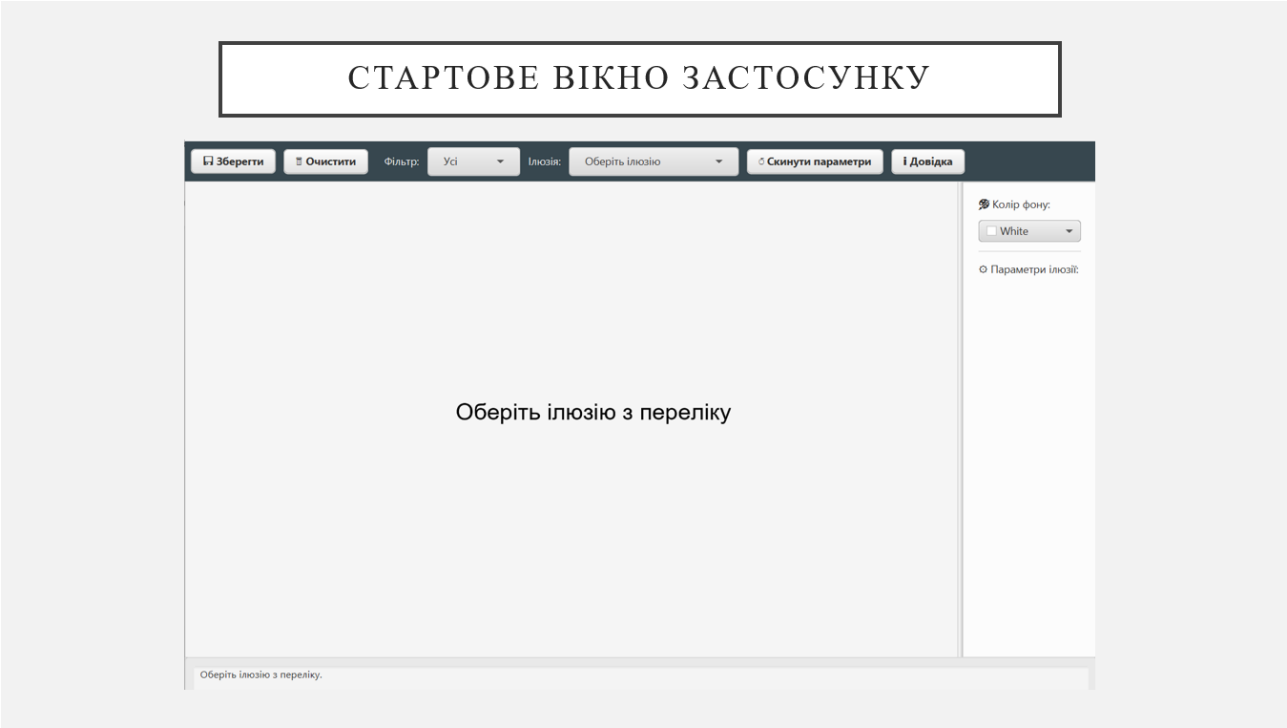

Рисунок Г.12 – Стартове вікно застосунку

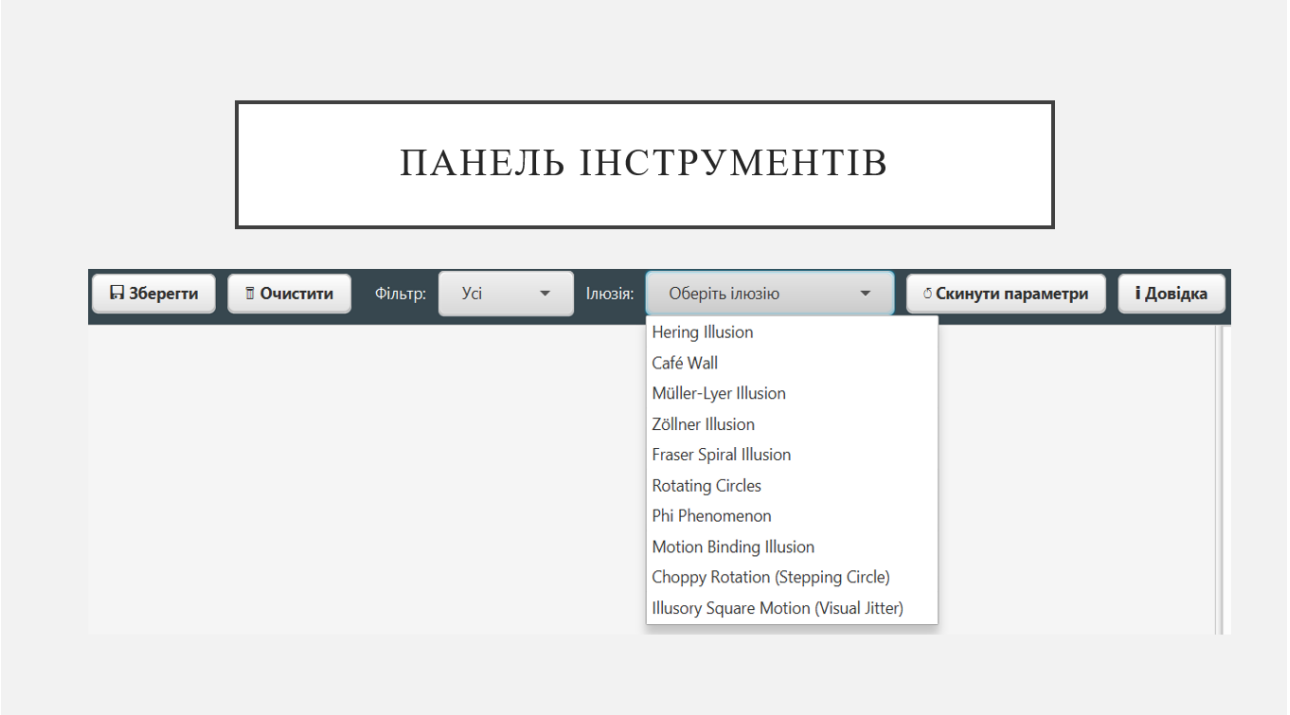

Рисунок Г.13 – Панель інструментів

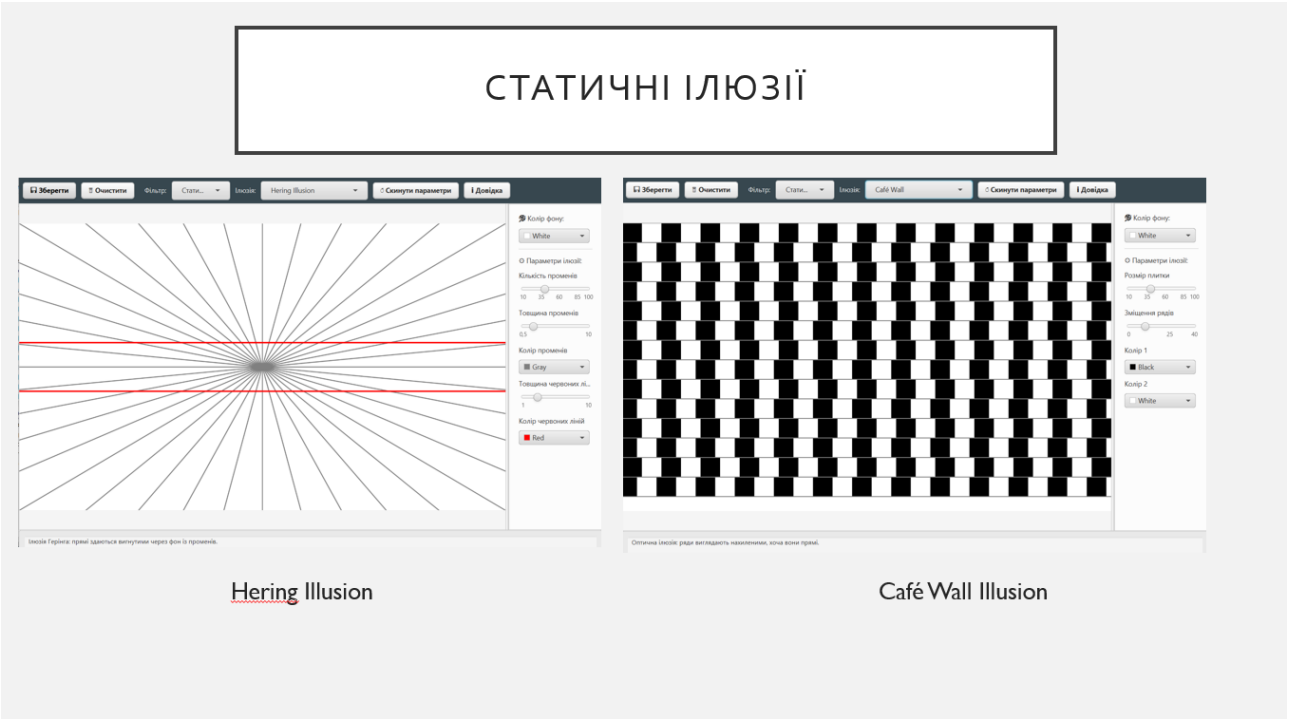

Рисунок Г.14 – Статичні ілюзії

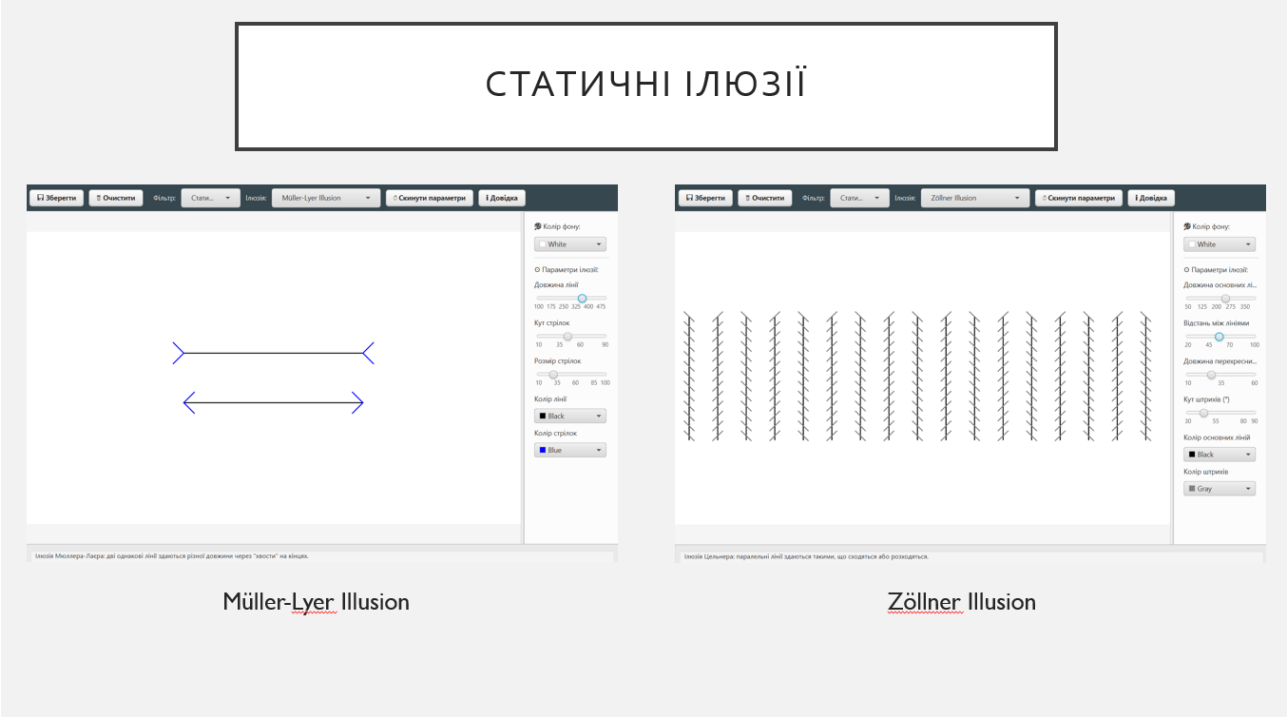

Рисунок Г.15 – Статичні ілюзії

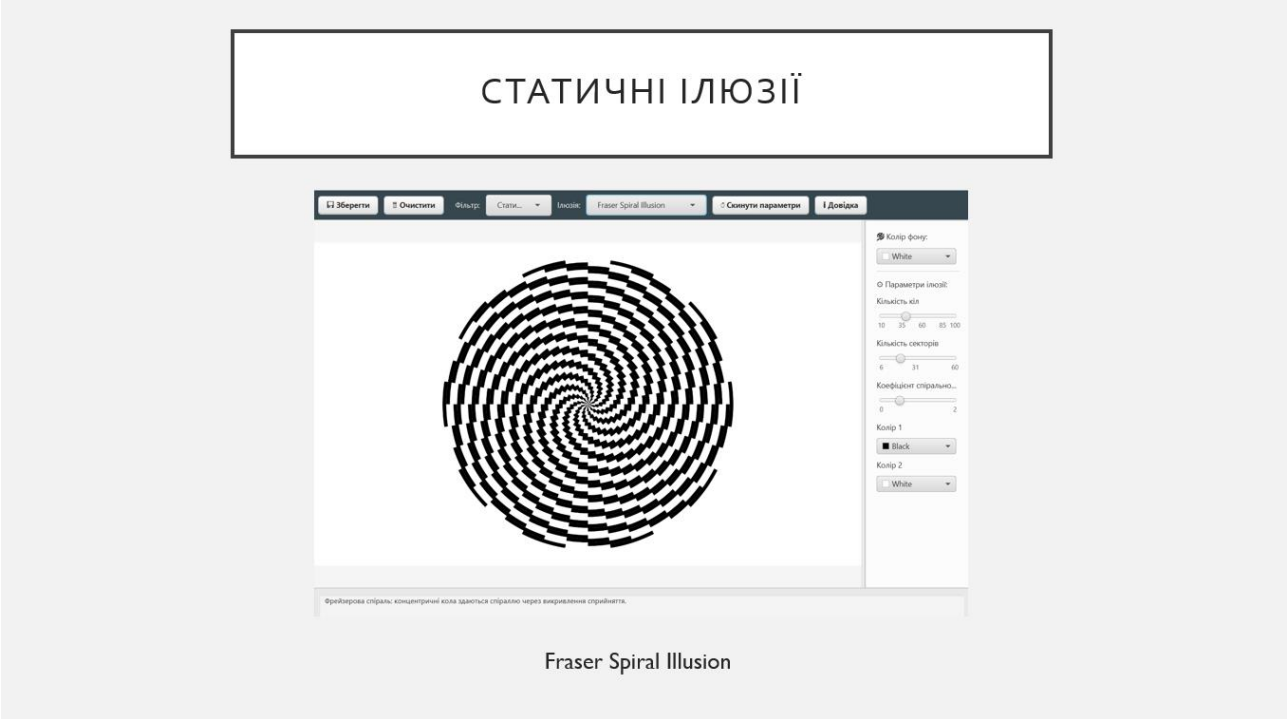

Рисунок Г.16 – Статичні ілюзії

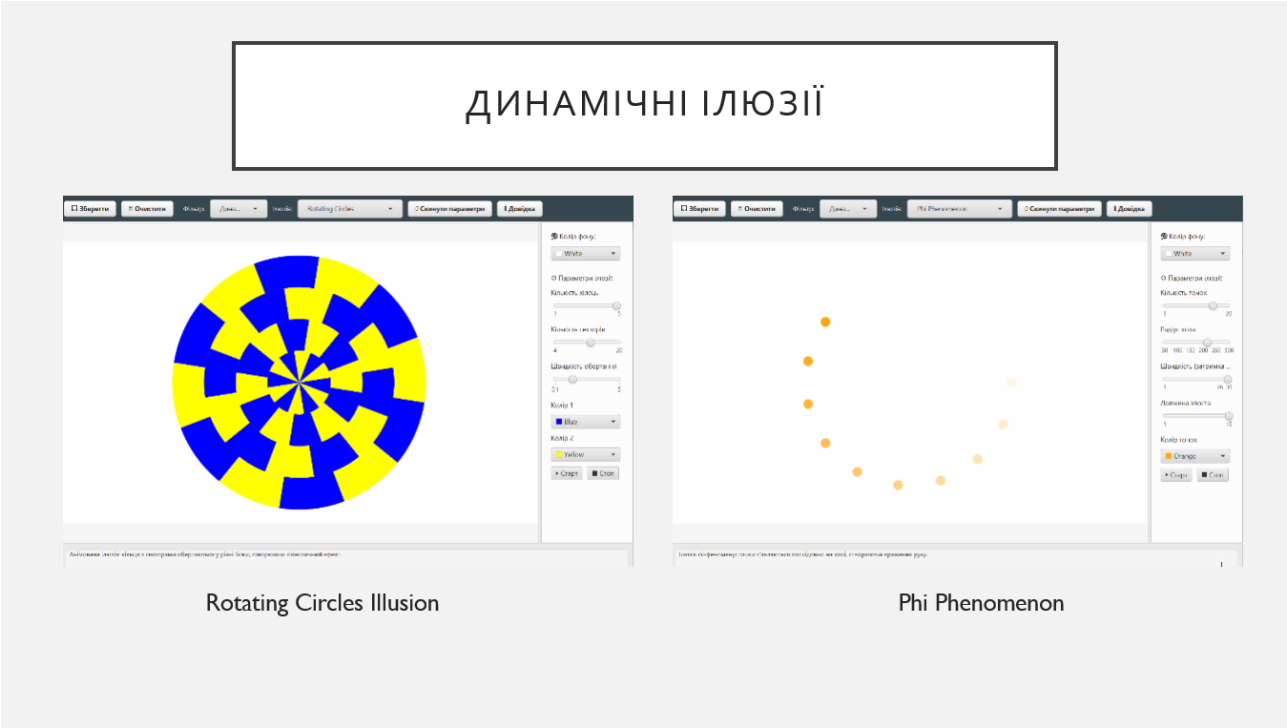

Рисунок Г.17 – Динамічні ілюзії

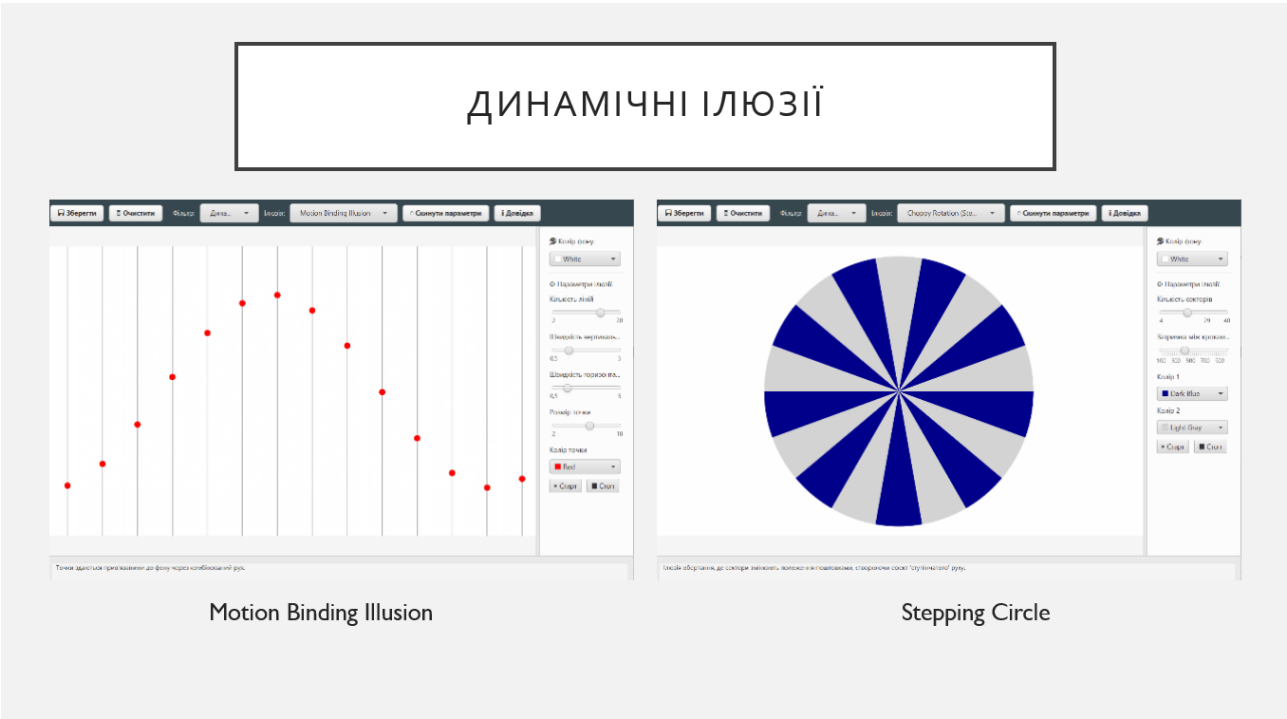

Рисунок Г.18 – Динамічні ілюзії

Рисунок Г.18 – Динамічні ілюзії

Рисунок Г.19 – Ілюзія зі зміненими параметрами

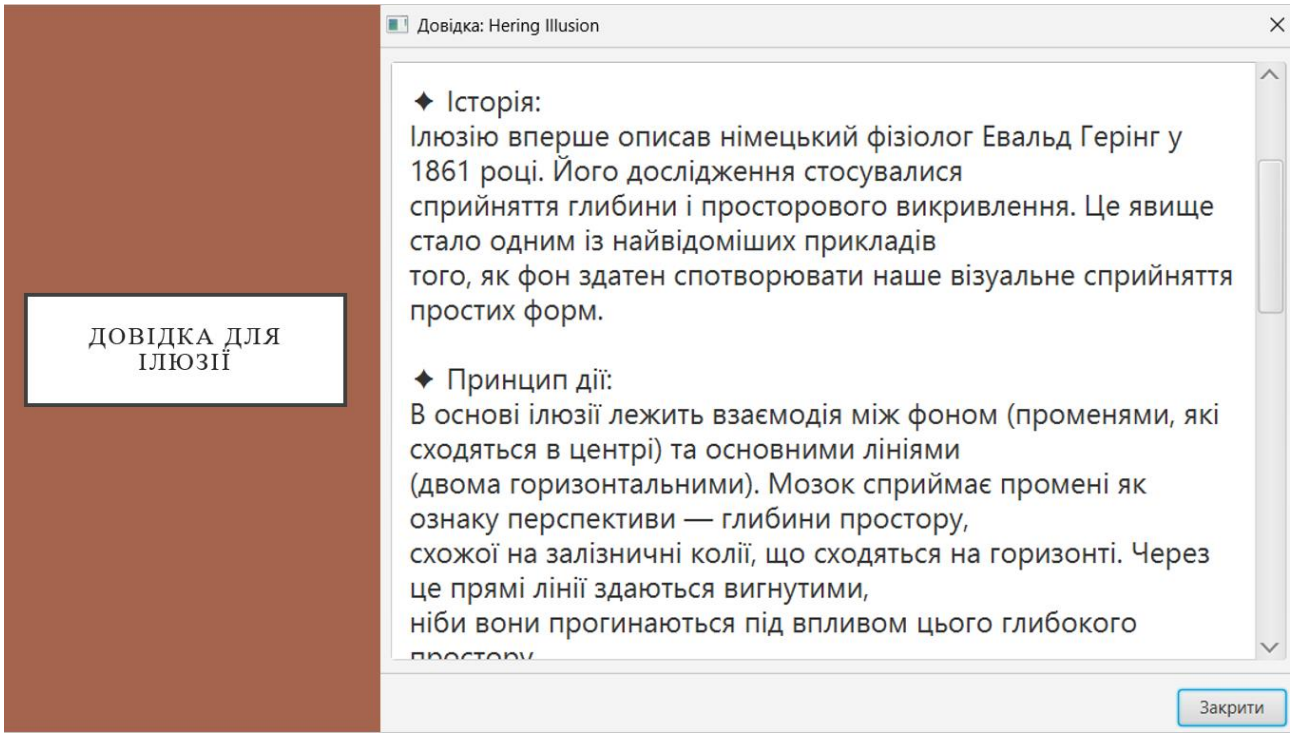

Рисунок Г.20 – Довідка для ілюзії

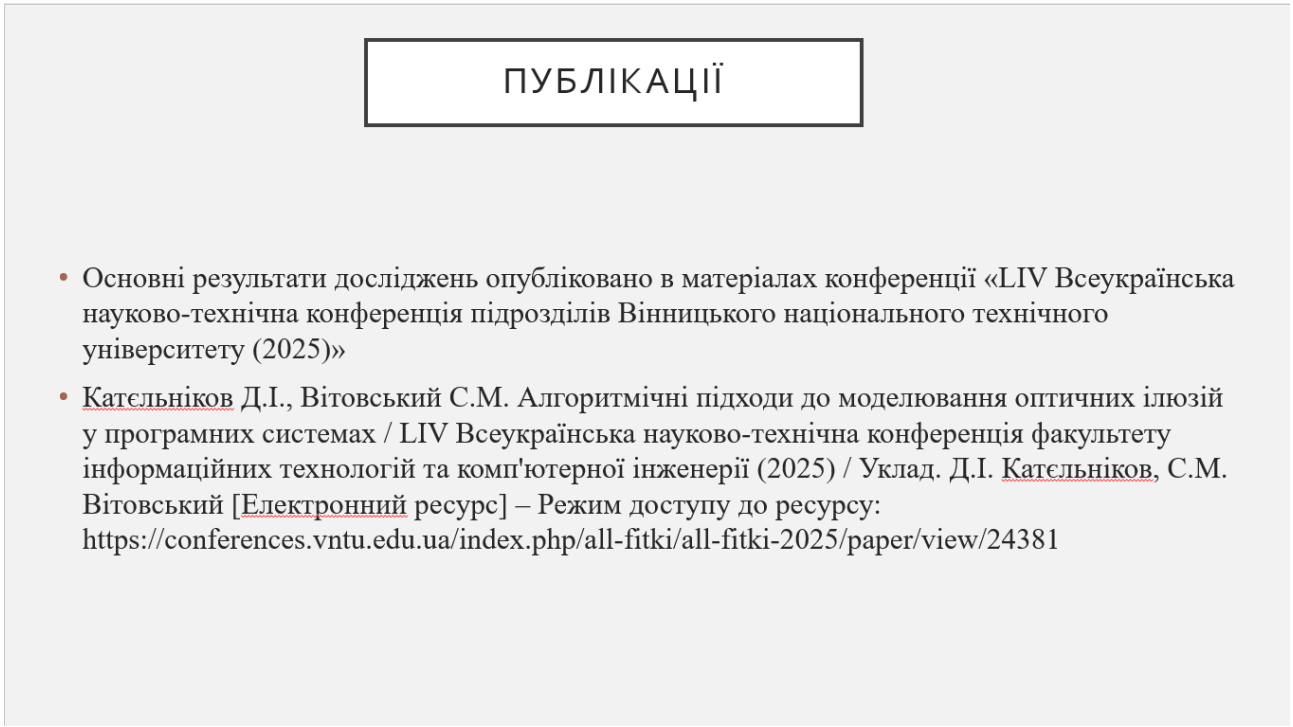

Рисунок Г.21 – Публікації

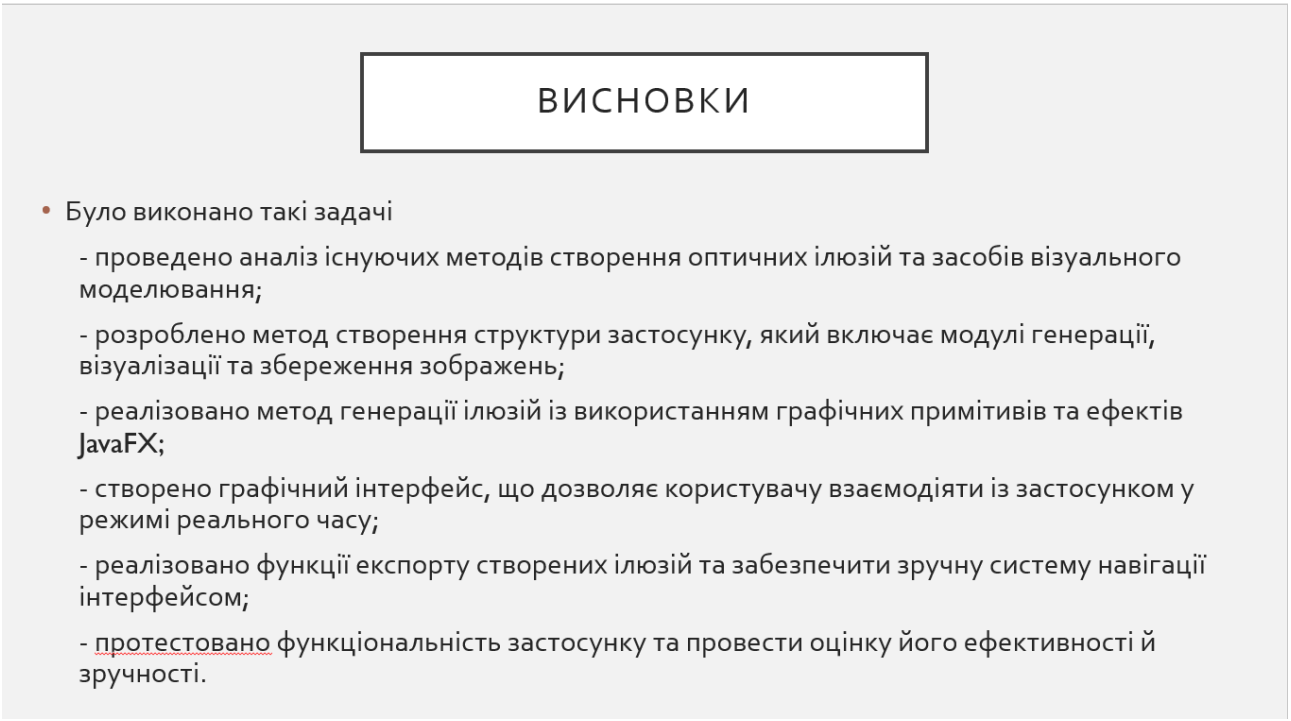

Рисунок Г.22 – Висновки