Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача

Виконав: студент 4 курсу
групи 1ПІ-21Б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Козаченко М. Р.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Чехместрук Р.Ю.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ОТ Дудник О.В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ д.т.н., проф. О. Н. Романюк
«06» червня 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

Козаченку Михайлу Руслановичу

1. Тема роботи – розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача.

Керівник роботи: Чехместрук Роман Юрійович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025р. №97

2. Строк подання студентом роботи 2 червня 2025р.

3. Вихідні дані до роботи: значення поточних налаштувань яскравості, контрасту, колірних каналів RGB, системна інформація про час доби, дані про активне вікно та запущені програми, таймер активності користувача, профілі режимів створені користувачем та їх параметри, системні функції Windows API, зокрема gammaRamp.

4. Зміст текстової частини: аналіз сучасних технологій адаптивного керування екраном, порівняння сучасних програм аналогів, дослідження механізмів регулювання зображення в ОС Windows, принципи створення та використання профілів зображення, методи визначення активного вікна програми та його асоціація з профілем, моделювання поведінки адаптивного алгоритму, залежність яскравості та кольору від часу доби, розробка методу алгоритму зміни основних параметрів екрану та створення режимів роботи екрану, створення користувацьких налаштувань та профілювання із заданими параметрами, розробка адаптивних режимів зміни параметрів та режимів екрану, робота програми у фоновому режимі, розробка інтерфейсу програмного застосунку, розробка інструкції користувача, тестування програмного застосунку.

5. Перелік ілюстративного матеріалу: приклади інтерфейсу програм аналогів, таблиця порівнянь характеристик аналогів, таблиці результатів проведення експериментального дослідження, гістограма зорового

навантаження, модель режиму 24/7, таблиця порівняльного аналізу м
програмування, ілюстрації інтерфейсу середовищ розробки, табл
порівняння середовищ розробки, блок схема методу UpdateGammaR
таблиця характеристик режимів роботи екрану, ілюстрації роботи режи
екрану, діаграма роботи режиму 24/7, ілюстрації сповіщень програмно
застосунку, ілюстрації інтерфейсу програмного застосунку, результа
тестування, інструкція користувача та технічні вимоги до пристроїв.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	к.т.н., доц. каф. ПЗ Чехместрук Р.Ю.	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025р.

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз існуючих рішень і проблематики	25.03.2025 – 05.04.2025	Вик.
2	Моделювання та експериментальне дослідження	06.04.2025 – 20.04.2025	Вик.
3	Теоретична частина	21.04.2025 – 01.05.2025	Вик.
4	Розробка програмного застосунку	02.05.2025 – 19.05.2025	Вик.
5	Розробка інтерфейсу та тестування	20.05.2025 – 30.05.2025	Вик.

Студент _____ (підпис) **М. Р. Козаченко** (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Р. Ю. Чехместрук** (ініціали та прізвище)

АНОТАЦІЯ

УДК 681.004.58

Козаченко М. Р. Розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 111 с.

На укр. мові. Бібліограф. 24 назв; рис. : 44; табл. 12.

У бакалаврській кваліфікаційній роботі розроблено інтелектуальний програмний застосунок для адаптивного керування режимами роботи екрану на основі аналізу активності користувача. Проєкт включає розробку режимів роботи екрану, зміну параметрів екрану (яскравості, контрасту та колірних каналів RGB), створення користувацьких профілів із заданими параметрами екрану, автоматичне регулювання параметрів екрану відповідно до часу доби, адаптивну зміну режимів роботи відповідно до ввімкненої програми, систему обчислення безперервного користування режимами та сповіщення про необхідність перерви, інтеграцію із системним треєм та роботою у фоновому режимі для підвищення зорового комфорту, зменшення навантаження на зір та покращення взаємодії користувачів із екраном. Програмний застосунок розроблено засобами та можливостями мови програмування C++, у середовищі розробки Microsoft Visual Studio.

Ключові слова: режими екрану, зміна параметрів, gammaRanp, користувацьке налаштування, адаптивна зміна.

ABSTRACT

Kozachenko M. R. Development of an intelligent software application for adaptive control of screen operation modes based on user activity analysis. Vinnytsia: VNTU, 2025. 111 p.

In Ukrainian language. Bibliographer: 24 titles; fig. : 44; tabl. 12.

In the bachelor's qualification work, an intelligent software application for adaptive control of screen operating modes based on user activity analysis was developed. The project includes the development of screen operating modes, changing screen parameters (brightness, contrast, and RGB color channels), creating user profiles with specified screen parameters, automatic adjustment of screen parameters according to the time of day, adaptive change of operating modes according to the enabled program, a system for calculating continuous use of modes and notifications about the need for a break, integration with the system tray and work in the background mode to increase visual comfort, reduce eye strain, and improve user interaction with the screen. The software application was developed using the tools and capabilities of the C++ programming language, in the Microsoft Visual Studio development environment.

Keywords: screen modes, changing parameters, gammaRanp, user settings, adaptive change.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. На сучасний етапі розвитку комп’ютерних технологій супроводжується стрімким зростанням тривалості часу взаємодії з екраном комп’ютера. Це призводить до різкого підвищення навантаження на зір, зниження концентрації уваги, порушень сну, головного болю, а також розвитку так званого синдрому комп’ютерного зору. Мінімізувати цей вплив можливо створенням інтелектуального програмного застосунку, здатного автоматично адаптувати параметри екрану відповідно до потреб користувача, поточних умов взаємодії з екраном, активності користувача та контексту використання. Актуальність роботи зумовлена потребою в інструменті, який не лише покращить взаємодію користувача із екраном пристрою, а й збереженню зорового здоров’я.

Одним із важливих аспектів у сучасному світі є оптимізація використання електроенергії. Ефективне використання різних режимів роботи екрану може сприяти зменшенню споживання електроенергії.

Постійний розвиток технологій інтерфейсу користувача вимагає постійного оновлення та вдосконалення інтерфейсу зі зручними, швидкими, простими, інтуїтивно зрозумілими та ефективними інструментами управління.

Існуючі вбудовані інструменти операційних систем, такі як нічне світло в Windows чи режим «Eye Comfort» у деяких пристроях, мають дуже обмежений функціонал здатен покрити лише невеликі потреби користувачів, і не враховують індивідуальні особливості користувачів або контекст використання комп’ютера.

Значна частина розроблених програмних застосунків має обмежені можливості налаштування режимів екрану, часто не підтримують створення гнучких профілів або не враховують активність користувачів.

У той час, як людський фактор усе частіше стає центром уваги в розробці програмного забезпечення, зростає потреба в адаптивних, «розумних» інструментах, що динамічно підлаштовуються до потреб користувача. Тема інтелектуального керування режимами екрану є актуальною, оскільки дозволяє поєднувати користувацький комфорт, здоров’я та технологічну гнучкість.

Розробка програмного застосунку, який адаптивно змінює параметри екрану відповідно до часу доби, активної програми, поведінки користувача та надає рекомендації щодо перерв – є відповіддю на запит часу.

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

Мета та завдання дослідження. Метою роботи є покращення та полегшення взаємодії користувачів із екраном за рахунок зміни режимів роботи екрану.

Основними задачами дослідження є:

- провести аналіз існуючих рішень у сфері зміни режимів роботи екрану;
- визначити вимоги до інтелектуального адаптивного програмного застосунку;
- реалізувати методи зміни яскравості, контрасту, кольорів RGB;
- створити систему профілів з можливостю призначення для окремих програм;
- розробити модуль добової автоматизованої зміни режимів;
- впровадити таймер активності з рекомендаціями про перерви;
- Забезпечити роботу програми у фоновому режимі через системний трей.

Об'єкт дослідження – процес зміни налаштування екрану для адаптивної зміни режимів роботи екрану.

Предмет дослідження – модулі та засоби, що забезпечать зміни основних налаштувань екрану під кожну потребу.

Методи дослідження – у процесі розробки застосовувались методи об'єктно-орієнтованого програмування, техніки роботи з API Windows, моделювання поведінки користувача, а також експериментальна перевірка ефективності адаптивних режимів.

Наукова новизна отриманих результатів:

1. Вперше запропоновано метод інтелектуального, адаптивного автоматичного керування параметрами екрану (яскравість, контраст, кольорові

канали RGB) з урахуванням часу доби, активності користувача та контексту активних програм, що дозволяє підвищити продуктивність роботи та комфорт користувача протягом дня.

2. Розроблено новий підхід до побудови динамічних профілів екрану, який забезпечує можливість створення, збереження та автоматичного застосування до п’яти індивідуальних режимів через інтеграцію із системним треєм або на основі контекстного аналізу активної програми, що підвищує ергономіку користувацької взаємодії.

3. Дістала подальшого розвитку система рекомендацій щодо організації перерв у роботі з екраном, яка базується на тривалості використання поточного режиму та реалізована через систему попереджувальних повідомлень, що сприяє збереженню зорового здоров’я користувача.

Практичне значення отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритм та реалізовано програмний застосунок, який забезпечує адаптивне керування режимами роботи екрану комп’ютера. Розроблені програмні засоби дозволяють автоматизувати зміну параметрів екрану відповідно до часу доби, активної програми та поведінки користувача.

Особистий внесок здобувача. Усі наукові результати, викладені у БКР, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: Інтелектуальний програмний застосунок для адаптивного керування режимами роботи екрану на основі аналізу активності користувача [1]. Аналіз вимог до інтелектуального програмного застосунку для адаптивного керування параметрами екрана [2].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення БКР доповідалися та обговорювалися на Міжнародних і всеукраїнських конференціях: LIV Всеукраїнська науково-технічна конференція підрозділів ВНТУ «Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії» ВНТКП ВНТУ-2025 (Вінниця 2025), IV Міжнародна науково-практична інтернет-конференція

«Молодь в науці: дослідження, проблеми, перспективи» МН-2025 (Вінниця 2025).

Публікації. Основні результати досліджень опубліковано в 2 наукових працях.

Структура та обсяг БКР. БКР складається зі вступу, п’яти розділів, висновків, списку літератури що містить 24 найменувань, 4 додатків. Робота містить 39 ілюстрацій, 12 таблиць. Загальний обсяг роботи складає 111 сторінок, основний зміст викладено на 69 сторінках друкованого тексту.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПРОБЛЕМАТИКИ

1.1 Актуальність розробки програмного застосунку керування режимами роботи екрану

Неможливо уявити сучасний світ без швидкісних інформаційних потоків та постійного використання комп'ютерних технологій у різних сферах діяльності людини. З кожним роком зростає кількість користувачів персональних комп'ютерів, мобільних пристроїв, ноутбуків що у свою чергу підвищує потребу у легкому, ефективному та гнучкому керуванню параметрами дисплеїв. Одним з ключових елементів в цьому контексті є ефективне керування та налаштування екрану під власні потреби. Особливо у операційній системі Windows, яка займає найбільшу частину з усіх ОС в Україні та 2 місце у світі [3].

Користувачі все більше часу проводять сидячи перед екраном, що негативно впливає на стан очей. Відповідне керування режимами роботи екрану зможе зменшити негативний вплив шкідливого синього світла вночі та підвищити комфорт користування екраном пристрою [4].

Попри наявність базових засобів регулювання екрану в операційних система, більшість із них не покривають усіх потреб користувачів ,таких як: адаптивна зміна налаштувань екрану відповідно до відкритої програми, створення профілів із користувацьким налаштуванням екрану. Зазвичай користувачам доводиться змінювати налаштування вручну, що є неефективним і знижує загальну продуктивність.

Наявні доступні аналоги не відповідають вимогам користувачів, багато з них мають обмежений функціонал, що не дозволяє точно налаштовувати параметри зображення під різні сценарії використання. Деякі програми пропонують лише базові функції, такі як регулювання яскравості та контрасту, але не мають можливостей зміни колірної температури та великого вибору режимів роботи екрану. Крім того, у багатьох аналогах інтерфейс є складним або незручним у використанні, що відвертає користувачів від активного, постійного користування програмним застосунком. Це підкреслює необхідність створення

якісного та конкурентоспроможного рішення, яке забезпечить ширший набір функцій і зручність взаємодії.

Таким чином, розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану відповідає сучасним потребам користувачів у комфортній, безпечній для здоров'я та ефективній роботі з комп'ютером. Це визначає актуальність теми дослідження та її практичну значущість.

1.2 Аналіз методів розв'язання задач

Для розробки інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача, використовуються різні методи, які можна поділити на кілька категорій:

- ручне регулювання;
- статичне профілювання;
- контекстно-залежна адаптація;
- інтелектуальна система керування.

Ручне регулювання параметрів екрану є найпростішим і найпоширенішим методом. Користувач самостійно змінює значення яскравості, контрасту, кольорову температури, нічний режим через вбудовані в операційну систему інструменти. Основним недоліком даного підходу є те, що відсутня автоматизація, додатковий функціонал та погана взаємодія з інструментами через інтерфейс користувача.

Статичне профілювання полягає у створенні заздалегідь визначених режимів (профілів), які налаштовуються відповідно до певних сценаріїв таких, як перегляд кіно, робота з екраном вночі та ін. Перемикання між профілями може бути частково автоматизоване або залишитись на розсуд користувача. Подібна концепція реалізована в деяких сторонніх програмах таких як Asus Armoury crate та Iris. Проте недоліком є обмеження перемиканнями конкретних режимів та профілів та не підтримують користувацьке та адаптивне налаштування.

Контекстно-залежна адаптація використовує інформацію про зовнішні чинники або активність користувача для зміни параметрів екрану. Наприклад

деякі програми можуть змінювати яскравість залежно від освітлення та часу доби. Такий підхід дозволяє покращити ергономіку, але він не враховує тип виконуваної задачі програмою з якою працює користувач.

Інтелектуальні системи адаптації передбачають глибший аналіз поведінки користувача, його взаємодію з програмами, та автоматичну зміну параметрів екрану на основі виявлених шаблонів або за допомогою правил, заданих користувачем. Такі програмні застосунки можуть реалізовувати призначення режимів окремим програмам та регулювати параметри екрану в реальному часі.

Окрім зміни налаштувань екрану, є також актуальні методи, пов’язані із аналізом тривалості користування екраном та відповідними налаштуваннями. Вони надають змогу бачити скільки часу користувач проводить за монітором і формує рекомендації щодо перерв для запобіганню погіршень здоров’я [5].

Таким чином актуальним у реалізації програмного застосунку є комплексне поєднання усіх методів. Саме такий підхід реалізовано в розробці інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача.

1.3 Порівняльний аналіз аналогів

У процесі розробки інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача важливо враховувати існуючі рішення, їх можливості та обмеження. Це дозволить виявити функціональні прогалини та вдосконалити власну реалізацію на основі практичного досвіду розробників аналогічних програм.

Нижче наведено огляд найбільш релевантних програмних рішень, які виконують схожі завдання. Відповідно до теми знайдено аналоги, програми:

- f.lux;
- Eye Saver;
- Armoury Create;

Програма f.lux – популярна кросплатформна програма, яка регулює колірну температуру дисплея відповідно до місця розташування та часу доби, пропонуючи функціональний відпочинок для очей. (див. рисунок 1.1) [6].

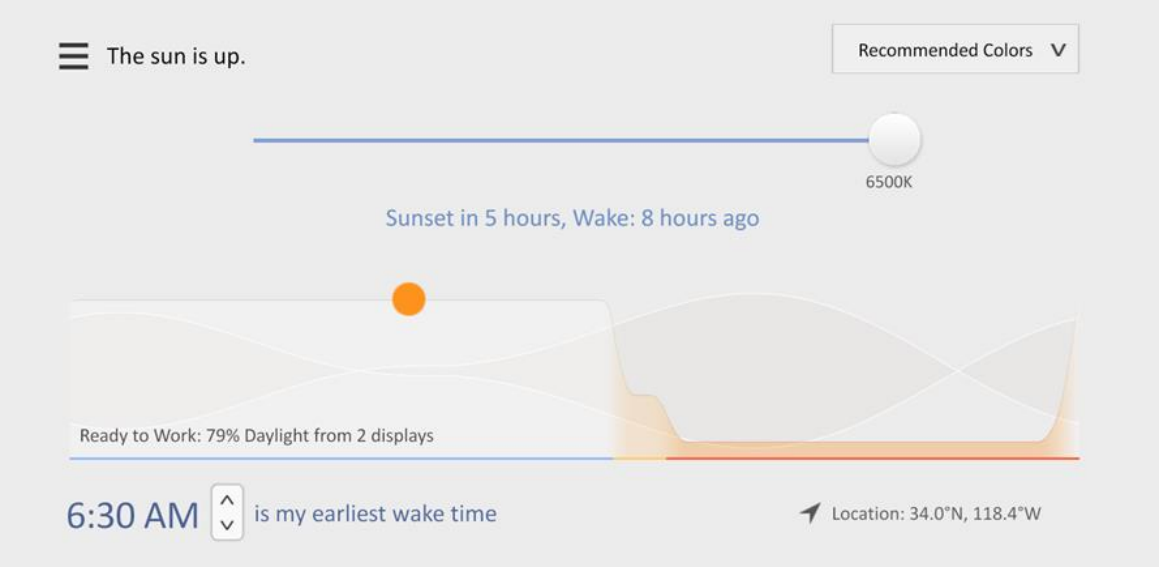

Рисунок 1.1 – Головне вікно програми f.lux

Програма розроблена компанією "f.lux Software LLC". Остання версія: 4.119.

Переваги програми:

- регулювання температури кольору;
- користувацьке налаштування;
- автоматична зміна налаштувань на основі часу доби.

Недоліки програми:

- можливість впливу на колір інших відображень;
- потреба у налаштуванні;
- можливі проблеми зі сумісністю.

Armoury Crate – це програма розроблена компанією Asus, орієнтована на геймерську аудиторію. Використовується для керування та моніторингу параметрів роботи ігрових ноутбуків та ПК [7]. Однією з можливостей є зміна режимів роботи екрана. Ця функція дозволяє налаштувати екран відповідно до потреб

Доступні 8 режимів роботи екрану: Default; Racing; Scenery; RTS/RPG; FPS; Cinema; Eyecare; Vivid (див. рисунок 1.2).

Рисунок 1.2 – Вікно зміни режимів екрану програми Armorury Crate

Переваги програми:

- можливість зміни режимів роботи екрану;
- моніторинг параметрів роботи системи;
- створення профілів.

Недоліки програми:

- доступна лише для пристроїв Asus;
- складна для нових користувачів;
- орієнтована на ігрові задачі;
- високе навантаження системи.

Eye Sever – програма для Windows, яка використовує різні методи для зменшення напруження очей, що виникає через довготривале користування комп’ютером. Фільтрує шкідливе синє світло, яке випромінює дисплей, і робить кольори теплішими та приємнішими для очей. Усуває малопомітне оку мерехтіння, підсвічування дисплея, причину напруги очей і головного болю. (див. рисунок 1.3) [8].

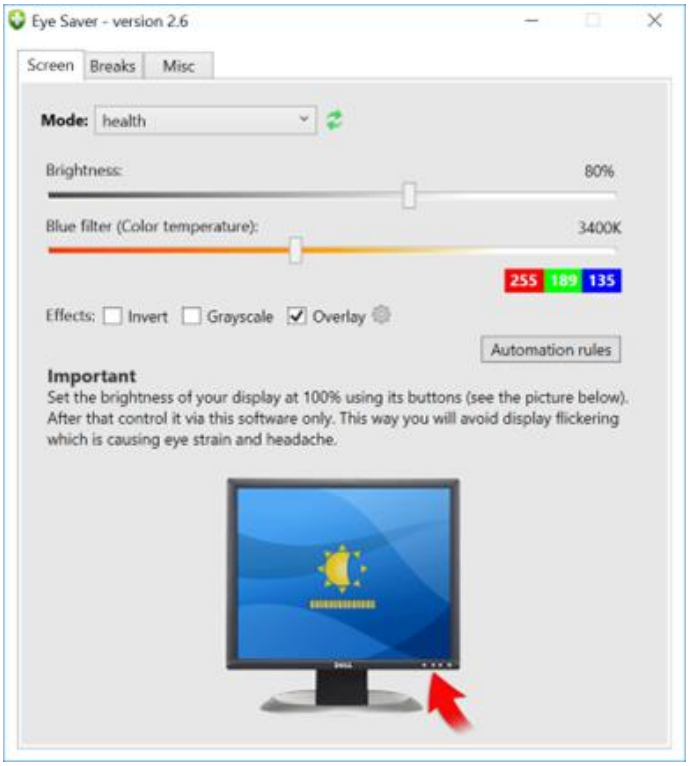

Рисунок 1.3 – Головне вікно програми Eye Saver

Програма розроблена компанією: Leosoft.ltd., остання версія 2.48.

Переваги програми:

- зменшення напруги очей;
- підвищення продуктивності;
- користувацьке налаштування.

Недоліки програми:

- деякі функції доступні лише в платній версії.

Жодна з представлених програм не надає комплексного рішення, що охоплює 6 важливих функцій одночасно:

- зміну основних параметрів екрану;
- створення користувацьких профілів;
- зміна налаштувань відповідно до часу доби;
- адаптивне перемикання режимів відповідно до програми;
- нагадування про необхідність перерви;
- зручне керування у фоновому режимі через меню у треї.

Результати порівняння аналогів та власного програмного застосунку зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	f.lux	Armoury Crate	Eye Saver	Власна розробка
Зміна режимів роботи екрану відповідно до потреб	-	+	-	+
Користувацька зміна параметрів (яскравість, контраст, RGB)	-	-	+	+
Регулювання температури кольору	+	-	+	+
Створення користувацьких профілів	+	-	+	+
Автоматична зміна за часом доби	+	-	-	+
Адаптивна автоматична зміна режимів до програм	-	-	-	+
Керування у фоновому режимі через меню у треї	+	+	+	+
Нагадування про необхідність перерви	-	-	-	+
Загальна оцінка	50%	25%	50%	100%

Відповідно до таблиці порівнянь розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача є доцільною. Отриманий продукт зможе покрити недоліки аналогів.

1.4 Постановка задач проєкту

На основі проведеного аналізу існуючих рішень, методів та інструментів адаптивного керування режимами роботи екрану, можна зробити висновки, що наявні інструменти частково вирішують проблему зорового навантаження та ергономіки, однак не забезпечують комплексного, інтелектуального, автоматичного та контекстно-залежного підходу до налаштування параметрів відповідно до потреб користувачів. Зокрема, відсутні рішення, які б поєднували одночасно:

- гнучке налаштування основних параметрів екрану (яскравості, контрасту та колірних каналів RGB);
- створення та використання користувацьких профілів;
- автоматичне адаптивне перемикання режимів відповідно до увімкнених програм;
- адаптацію налаштувань екрану відповідно до часу доби;
- систему рекомендацій про необхідність перерви у використанні екрану, задля запобігання погіршення здоров’я;
- інтеграцію програмного застосунку з роботою у фоновому режимі та меню в треї.

З огляду на це, формується мета проєкту – розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача, для покращення та полегшення взаємодії користувачів з екраном.

Для досягнення цієї мети необхідно виконати наступні задачі:

- провести аналіз існуючих програмних рішень;
- обґрунтувати вибір теоретичних та технічних рішень, необхідних для реалізації функцій;
- спроектувати архітектуру програмного застосунку;
- реалізувати режими роботи екрану відповідно до найпоширеніших потреб користувача;
- реалізувати користувацьке налаштування та створення власних профілів із налаштуваннями;

- забезпечити функцію адаптивної зміни параметрів відповідно до часу доби;
- розробити функцію для адаптивної зміни параметрів відповідно до відкритої програми;
- створити модуль, що контролюватиме тривалість безперервної роботи режимів та сповіщень про перерву;
- забезпечити роботу програми у фоновому режимі та інтегрувати із системним треєм;
- розробити зручний, простий та інтуїтивно зрозумілий інтерфейс користувача;
- провести тестування розробленого програмного застосунку та оцінити ефективність реалізованих функцій.

Успішне виконання поставлених дозволить створити програмний застосунок, який немає прямих аналогів та конкурентів за функціональністю та здатен покращити та полегшити взаємодію користувачів з екраном.

1.5 Висновки

У першому розділі БКР було проведено аналіз актуальності проблеми адаптивного керування режимами роботи екрану та обґрунтовано необхідність створення інтелектуального програмного застосунку. Розглянуто основні методи, що застосовуються в існуючих рішеннях, від ручного керування налаштуваннями до часткової автоматизації процесів. Проведено детальний аналіз розроблених програм аналогів, виявлено їх переваги та недоліки. Основним недоліком є те, що жодна програма не пропонує комплексного рішення а орієнтується під одну конкретну задачу. На основі цього сформульовано завдання, які необхідно успішно виконати при розробці програмного застосунку, задля отримання продукту, аналогів та конкурентів якому не розроблено.

2 ТЕОРЕТИЧНА ОСНОВА

2.1 Огляд стану технологій зміни налаштувань екрану

У контексті розвитку програмних засобів, спрямованих на підвищення комфорту користувачів при роботі з комп’ютерним екраном, технології керування параметрами екрану займають важливе місце. До таких параметрів: належать яскравість, контраст, кольорова температура та інтенсивність складових кольору (RGB). Сучасні методи регулювання цих характеристик реалізуються як на апаратному рівні (через монітори та відеокарту), так і на програмному – за допомогою драйверів, API операційної системи та сторонніх утиліт.

Головною перевагою технологій адаптивного налаштування екрану є їх позитивний вплив на здоров’я користувачів. Дослідження підтверджують, що надмірне випромінювання синього світла у вечірній час може призводити до порушення циркадних ритмів і проблем зі сном. Використання спеціалізованого програмного забезпечення для зменшення цього впливу може значно покращити якість життя людей, які проводять багато часу перед монітором.

В більшості випадків, апаратури та драйверів моніторів, зміна параметрів відбувається через вбудоване меню монітора (OSD – On-Screen Display). Таке налаштування потребує ручного регулювання користувачем та не передбачає автоматизації, після декількох ручних змін параметрів у користувачів зникає бажання знову вручну змінювати параметри. Деякі сучасні монітори підтримують керування через USB або DDC/CI-протоколи, що відкриває можливість програмного доступу до налаштувань монітора. Однак підтримка DDC/CI є неповна та обмежена, або навіть нестабільна у межах одного виробника

На рівні ОС (особливо Windows) існують інструменти та інтерфейси, що дозволяють змінювати значення кольоропередачі – насамперед через механізм gammaRamp. Цей механізм дозволяє застосовувати табличні значення для кожного з трьох колірних каналів RGB, R – червоний, G – зелений, B – синій, які змінюють візуальне відображення екрану без втручання у фізичні характеристики монітора [9].

Існує низка програмних застосунків, таких як Armoury Crate, f.lux, Eye saver та інші, що частково реалізують зміну налаштувань екрану, здебільшого пропонуючи декілька режимів роботи екрану орієнтованих під конкретну задачу, зміну параметру теплоти екрану шляхом накладання фільтру синього кольору, та зміну теплоти відповідно до часу доби, не пропонуючи комплексного, гнучкого рішення, яке надає користувачу можливість зміни конкретних параметрів, адаптивного, автоматичного налаштування параметрів.

Незважаючи на наявність великої кількості інструментів, утиліт та програм, більшість з них не дозволяють здійснювати гнучке налаштування, профілювання, не підтримують автоматизації. Це створює передумови для розробки нового програмного застосунку, який поєднає функціональність, автоматизацію та зручність користування в одному цілісному рішенні.

2.2 Принципи регулювання параметрів дисплея в середовища Windows

Операційна система Windows надає основні засоби керування параметрами виводу зображення на дисплей, зокрема такими як яскравість, контраст, кольорова температура, а також значення червоного(R), зеленого(G) та синього(B) кольорових каналів (RGB). Дані параметри можуть змінюватись користувачем через графічний інтерфейс або програмно – з використанням відповідних системних функцій та API.

Регулювання яскравість дисплея у Windows можна двома основними способами:

- апаратно, через драйвер відеокарти та інтерфейс монітора;
- програмно, через зміну значень кольорової гами через API.

Програмне регулювання яскравості не змінює фактичної потужності підсвічування дисплею, а просто темнішає відображення. Тому воно доступне навіть на зовнішніх моніторах, де апаратна яскравість недоступна для контролю з боку ОС.

Для регулювання контрасту, операційна система Windows не надає прямого API, як у випадку з яскравістю. Однак контраст можна змінювати

опосередковано, за допомогою модифікації LUT (Look-Up Table) у таблиці гами, що впливає на співвідношення світлих і темних пікселів. Для прикладу, збільшення контрасту досягається шляхом розширення динамічного діапазону каналів RGB у gammaRamp, що зменшує кількість півтонів, але посилює візуальні відмінності.

Кожен з трьох кольорових каналів може регулюватись окремо, що дозволяє зробити доволі гнучко коригувати кольоровий баланс екрану. Зміни застосовуються через побудову нової таблиці гами, яка визначає, як інтенсивність кожного кольору перетворюється при виведені зображення. Це дозволяє створювати фільтри, наприклад для зменшення синього випромінення у вечірній час або для балансу кольорових відхилень конкретного монітору .

У середовищі Windows керування кольоровими параметрами дисплея реалізується через такі функції:

- SetDeviceGammaRamp(hdc, *lpRamp) – задає таблицю гами (гамма-криву) для дисплея;
- GetDeviceGammaRamp(hdc, *lpRamp) – отримує поточну таблицю гами;
- EnumDisplayDevices та EnumDisplaySettings – дозволяють визначити активні дисплеї та їх параметри;
- GetForegroundWindow та GetWindowThreadProcessId – використовуються для виявлення активного вікна, що необхідне для автоматично застосування параметрів відповідно до відкритого вікна.

Ці функції вимагають доступу до дескриптору контексту (HDC) та потребують певного рівня прав, особливо при роботі з декількома дисплеями одночасно, або при використанні в 64-бітній системі.

Windows підтримує керування кількома дисплеями одночасно, однак таблиця гами за замовчування змінюється для основного дисплею. Для керування іншими моніторами необхідно вказати дескриптор відповідного пристрою, що ускладнює програмну реалізацію та вимагає використання додаткових бібліотек та сторонніх API [10].

Таким чином ОС Windows надає можливості для програмного керування параметрами виводу зображення на дисплей через відповідні API, зокрема gammaRamp, що дозволяє реалізувати інтелектуальну зміну режимів роботи екрана. Дані можливості лягли в основу розроблених алгоритмів адаптивного керування режимами роботи екрану, реалізованих у межах БКР проєкту.

2.3 Особливості роботи з gammaRamp та API моніторів

Таблиця gammaRamp складається з трьох масивів по 256 значень. Кожне значення відповідає конкретному рівню інтенсивності пікселя, який модифікується перед його виведенням на екран. Значення таблиці гами зберігається у вигляді 16-бітних чисел (від 0 до 65535), що дозволяє досягти високої точності регулювання.

Модифікація GammaRamp дозволяє:

- затемнити або освітлити зображення;
- знизити або підвищити контраст;
- змінити колірний баланс;
- накласти фільтри.

Для програмного доступу до gammaRamp у Windows використовується функція: BOOL SetDeviceGammaRamp(HDC hDC, LVOID lpRamp); у якій:

- hDC – дескриптор контексту екрану;
- lpRamp – вказівник на структуру, яка містить три масиви по 256 елементів.

Перед використанням цієї функції необхідно отримати HDC для екрану за допомогою функції GetDC(NULL) або CreateDC().

Для отримання поточних налаштувань гами використовується функція: BOOL GetDeviceGammaRamp(HDC hDC, LPVOID lpRamp);

Хоча gammaRamp є універсальним і стандартним механізмом, її використання має кілька обмежень і проблем:

- у більшості реалізацій змінює гаму лише для основного монітору;

- деякі графічні драйвери (особливо нові версії NVIDIA або AMD) можуть блокувати зміну gammaRamp або перезаписувати її підчас перемикання профілю;
- після режиму сон, блокування екрану чи режиму енергозбереження система може скинути значення таблиці gammaRamp, що потребує додаткового контрою з боку програмного застосунку;
- gammaRamp не забезпечує апаратної зміни яскравості а лише модифікує виведене зображення.

Окрім стандартного API, для розширення можливостей управляння дисплеями можуть використовуватись додаткові засоби:

- DDC/CI – протокол апаратного доступу до налаштувань монітора;
- WWI – для отримання системної інформації;
- Win32 API – для роботи з багатомоніторними конфігураціями.

Механізм gammaRamp у ОС Windows надає ефективний спосіб програмної зміни параметрів колірних каналів, яскравості, контрасту. У межах БКР цей інструмент активно використовується для реалізації основних функцій програмного застосунку. Проте через його обмеження доцільно також використовувати додаткові інструменти [11].

2.4 Структура роботи режимів: яскравість, контраст, RGB

У розробленому програмному застосунку реалізовано систему керування режимами екрана, які базуються на регулюванні трьох основних параметрів зображення: яскравість, контраст та кольоровий баланс через канали R, G, B. Кожен режим являє собою набір конкретних значень цих параметрів, що дозволяє адаптувати зображення до певного типу діяльності користувача, часу доби або персональних уподобань.

Яскравість – це візуальна інтенсивність освітлення, яка визначає, наскільки світлим виглядає зображення на екрані. У контексті програмного керування, яскравість не змінює фізичну силу підсвітки, а регулює відображення кольорових компонентів через таблицю gammaRamp. У розробленому програмному застосунку яскравість змінюється шляхом множення значень компонентів R, G,

B на певний коефіцієнт (від 0.0 до 1.0), з дотриманням межі 65535 у 16-бітному представленні значень гами.

Контраст відповідає за різкість переходів між темними і світлими ділянками зображення. З точки зору програмного керування, контрастність реалізується шляхом нелінійного перетворення значень gammaRamp.

У програмному застосунку передбачено параметр, який дозволяє підвищити або зменшити контрастність кожного каналу окремо або одночасно. Це дозволяє, зокрема, створити режими з м'яким зображенням (для читання) чи висококонтрастним (для роботи з відео або графікою).

Зображення на дисплеї формується шляхом комбінування червоного, зеленого та синього кольорових каналів. Регулювання інтенсивності кожного з них дозволяє досягати різних ефектів – від «теплого» нічного режиму (зниження синього) до «холодного» денного [12].

У структурі режимів, які реалізовано в програмному застосунку, Кожен режим визначається набором параметрів:

- current_brightness, яскравість (%);
- current_contrast, контрастність (%);
- current_red, current_green, current_blue, кольорові коефіцієнти каналів R, G, B (%).

Ці значення масштабуються до діапазону від 0 до 1 та використовуються для обчислення гамма-кривої. Окремо для кожного з 256 рівнів інтенсивності кольору обчислюється нове значення, яке враховує:

- зміщення яскравості;
- розтягнення або стиск динамічного діапазону (контраст);
- корекцію кольорових каналів через насиченість та кольорові коефіцієнти.

Формула в середині циклу for (int i = 0; i < 256; ++i): value = ((value - 0.5f) * contrast + 0.5f) * brightness. визначає нормалізовану яскравість із контрастною модуляцією.

Після цього для кожного каналу:

- gammaRamp[0][i] = gray + s * (baseValue * rFactor - gray); // Червоний;
- gammaRamp[1][i] = gray + s * (baseValue * gFactor - gray); // Зелений;
- gammaRamp[2][i] = gray + s * (baseValue * bFactor - gray); // Синій.

Обчислюється зміщене значення, де: gray – нейтральне значення (основа), s – насиченість, rFactor, gFactor, bFactor – коефіцієнти RGB.

Таким чином, кожне значення формується з урахуванням відхилення від сірого (тобто балансу кольорів), що дозволяє створити ефекти "теплого", "холодного", "нейтрального" зображення.

Програмний застосунок підтримує створення режимів, кожен з яких відповідає певному набору значень згаданих параметрів. Наприклад, у методі Shooter() викликається: SetDisplayProfile(100, 100, 100, 105, 105). Це означає: R, G, B = 100%, яскравість = 105%, контраст = 105% (посилення яскравості та контрасту для чіткої картинки у грі). Таким чином, режим «Shooter» посилює яскравість і контраст зберігаючи стандартні значення колірних каналів.

Технічні переваги такого методу реалізації:

- висока точність регулювання (16-бітна гама);
- миттєве застосування налаштувань без перезапуску програм;
- гнучкість формування користувацьких налаштувань, профілювання змінених параметрів, включно з тонким налаштуванням колірного балансу;
- легка інтеграція з подіями інтерфейсу через кнопки або системний трей.

Реалізований підхід до зміни режимів роботи екрану заснований на гнучкій побудові таблиці гами (gammaRamp), яка модифікується в реальному часі з урахуванням заданих параметрів. Це забезпечує адаптивну візуалізацію, підвищує комфорт роботи за комп’ютером та дозволяє тонко налаштовувати зображення відповідно до індивідуальних, конкретних потреб користувача.

2.5 Висновки

У другому розділі БКР було розглянуто теоретичну частину реалізації інтелектуального програмного застосунку для адаптивного керування параметрів екрану на основі аналізу активності користувача. Оглянуто сучасний стан технологій зміни параметрів екрану в ОС Windows, та визначено засоби та методи, що дозволяють реалізувати програмне регулювання зображення.

Виявлено, що найбільш універсальним і сумісним методом зміни параметрів екрану у Windows є використання гама таблиці, яка надає доступ до зміни значень яскравості, контрасту та колірних каналів. Описано структуру роботи режимів екрану.

Результати теоретичного аналізу лягли в основу подальшої розробки програмного застосунку.

3 МОДЕЛЮВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Постановка експерименту та критерії оцінювання

Метою проведення експерименту є перевірка ефективності реалізованих модулів програмного застосунку в умовах реального використання.

Передбачається оцінити наступне:

- відповідність режимів задачам користувачів;
- зручність перемикання між профілями;
- доцільність та вплив системи сповіщень на зменшення втоми;
- стабільність роботи програми в режимі 24/7.

Умови проведення експерименту:

- ноутбук ASUS із IPS дисплеєм 1920×1080, Windows 10;
- 5 користувачів віком 19-40 років;
- тривалість тестування – 3 сесії по 1,5 години (1 режим – 1 сесія);
- параметри дисплея задаються програмним застосунком.

Критерії оцінювання:

Суб’єктивна оцінка комфорту – кожен користувач після проведеної сесії оцінює ступінь зорового навантаження за 5-бальною формулою:

$$\mathrm{C} = \frac{1}{n}\sum_{i=1}^{n} c_i$$

де C – середній рівень комфорту, c_i – оцінка і-го користувача.

Частота використання режимів:

$$F_p = \frac{N_p}{N}$$

де F_p – частка використання певного режиму, N_p – кількість перемикань на нього, N – загальна кількість змін.

Час реакції системи на перемикання (затримка від моменту вибору до фактичного застосування параметрів), фіксується вручну і автоматично. Визначається середній час за формулою:

$$T_{avg} = \frac{1}{m}\sum_{j=1}^{m} t_j$$

де t_j – час спрацювання при j-in активації, m – кількість вимірювань.

Результативність сповіщень:

- % користувачів, які зробили перерву після сповіщення;
- час реакції після появи сповіщення;
- кількість ігнорованих повідомлень.

Засоби фіксації результатів:

- анкетування користувача;
- лог-файли програми;
- знімки екрану інтерфейсу під час активного користування.

3.2 Проведення експериментальних досліджень

Згідно з методикою описаною у розділі 3.1, проведено 3 сесії експериментів досліджень для оцінки функціональних можливостей розробленого програмного застосунку, 7 основних режимів:

- шутер;
- декорації;
- книга;
- кіно;
- захист очей;
- ніч;
- за замовчування.

Для кожного режиму проводилось тестування п’ятьма користувачами протягом сесії 90 хвилин. Фіксувались наступні показники:

- суб’єктивна оцінка зорового комфорту;
- середній час реакції системи на перемикання режимів;
- загальна частота використання кожного режиму.

Після проведення сесії експерименту кожен з 5 учасників надав оцінки зорового комфорту по 5-бальній шкалі, обчислено у таблицю (див. таблиця 3.1).

Таблиця 3.1 – Оцінки комфорту

Режим	Середня оцінка
Шутер	3.8
Декорації	4.2
Книга	4.6
Кіно	4.3
Захист очей	4.8
Ніч	4.7
За замовчування	3.5

Такі режими як «Захист очей» та «Ніч» отримали найвищі оцінки з боку користувачів завдяки зменшенню яскравості, теплому відтінку кольорової гами екрану та загальному зниженню візуального навантаження. Найнижчу оцінку отримав стандартний режим, який не має додаткових змін параметрів дисплею.

Часу при перемиканні з будь-якого режиму на інший (див. таблиця 3.2)

Таблиця 3.2 – Час реакції системи на перемикання між режимами

Режим	Середній час реакції T_{avg}, мс
Шутер	310
Декорації	300
Книга	295
Кіно	305
Захист очей	315
Ніч	290
За замовчування	280

Усі режими демонструють швидкий час реакції а саме <350, що забезпечує комфортне перемикання без помітних затримок.

Упродовж 3 сесій, по 90 хвилин кожна, було зафіксовано кількість увімкнення кожного режиму (див. таблиця 3.3).

Таблиця 3.3 – Частота використання режимів

Режим	Кількість активацій	Частота використання F_p
Шутер	17	12%
Декорації	23	16%
Книга	28	19%
Кіно	25	17%
Захист очей	31	21%
Ніч	21	15%
За замовчування	5	<4%

Режим «Захист очей» став найпопулярнішим, що свідчить про актуальність функцій, орієнтованих на турботу про зір користувача.

Експериментальне дослідження підтвердило доцільність реалізації спеціалізованих режимів у програмному застосунку. Найвищі оцінки отримали режими «Книга», «Захист очей» та «Ніч», що вказує на потребу користувачів у адаптивному керування екраном відповідно до типу завдання.

3.3 Моделювання впливу режимів на зорове навантаження

Одним із головних критеріїв ефективності програмного застосунку, що керує параметрами дисплею, є його вплив на стан зору користувача. У даному розділі розглянуто моделювання впливу реалізованих режимів на показник втоми очей, зорового дискомфорту та оцінки загального зорового навантаження.

Передбачається, що режими зниженої яскравості, контрасту та зменшеним синім колірним каналом мають менший негативний вплив на зорову систему, ніж

стандартні, ігрові та кіно-режими, які орієнтовані на яскраве та насичене зображення.

Моделювання впливу здійснювалось за такими змінними:

- L, рівень яскравості (%);
- B, насиченість синього колірного каналу (%);
- C, контраст (%);
- T, тривалість сесії (хв);
- Z, інтегральний коефіцієнт зорового навантаження.

Використано формулу, яка умовно відображає зорове навантаження з урахуванням впливових факторів:

$$Z = a \cdot \frac{L \cdot B \cdot C}{100^3} \cdot T$$

де a = 1.2 – коефіцієнт вагомості, $Z \in |0,1|$ – нормалізований показник навантаження (чим ближче до 1, тим вища ймовірність перевтоми).

Результати обчислень занесено у таблицю 3.4.

Таблиця 3.4 – Результати обчислень

Режим	L(%)	B(%)	C(%)	T(хв.)	Z
За замовчування	100	100	100	90	1.08
Шутер	105	100	105	90	1.33
Декорації	90	100	120	90	1.17
Книга	90	80	80	90	0.63
Кіно	90	100	110	90	1.19
Захист очей	90	60	90	90	0.70
Ніч	70	90	70	90	0.53

При обчислені розраховано інтегральний коефіцієнт зорового навантаження – Z, дані зорового навантаження кожного режиму представлені у гістограмі (див. рисунок 3.1).

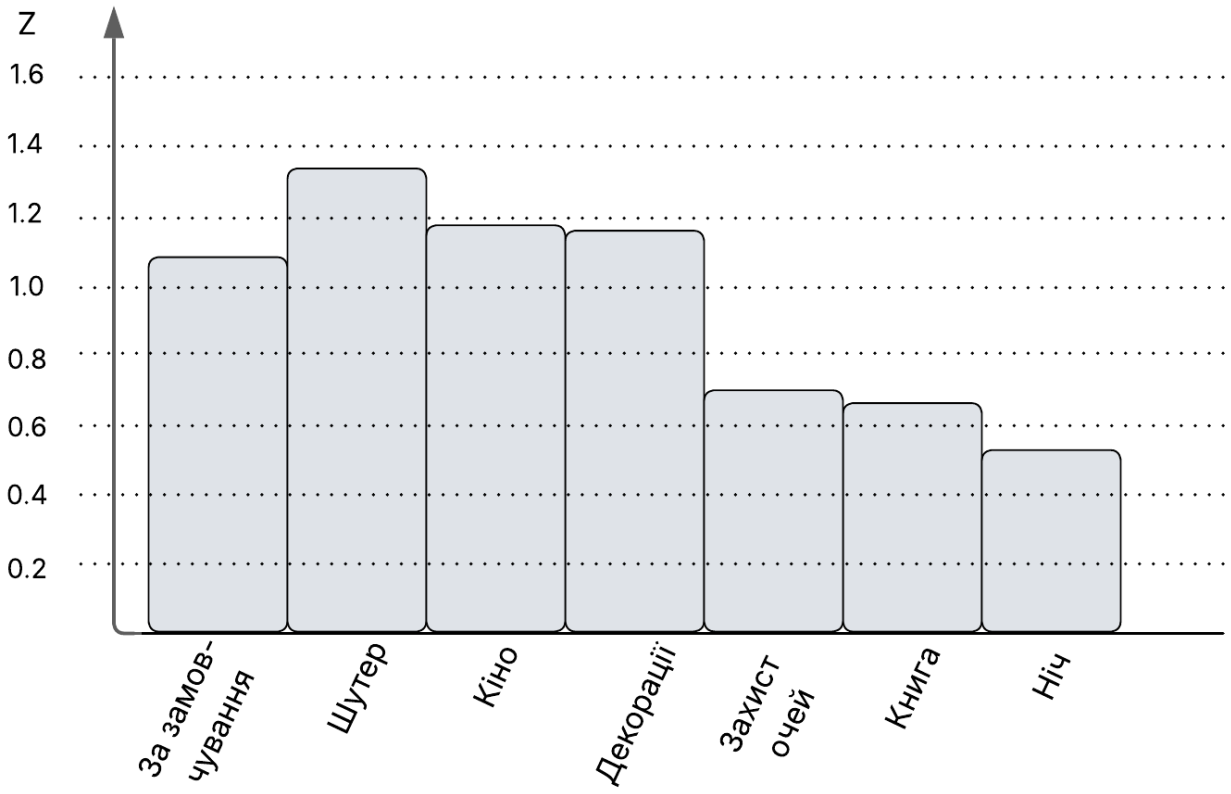

Рисунок 3.1 – Гістограма зорового навантаження

Пороговим значенням показника Z вважається 1.0, за якого рекомендовано обов’язкову перерву. Режими «Шутер», «Кіно» та «Стандартний» перевищили цей поріг, що вказує на потребу в обмеженому часі використання або автоматичному сповіщенні.

Моделювання показало, що вибір режиму екрану має прямий вплив на рівень зорового навантаження користувача. Режими зниженої яскравості й контрасту суттєво знижують ризик перевтоми очей, особливо при тривалому користуванні. Отримані результати стануть підґрунтям для моделювання системи сповіщень рекомендацій, що адаптується до поточного режиму та часу використання.

3.4 Моделювання системи сповіщень та 24/7 режиму

Збереження здоров’я користувача є важливою складовою при розробці програмного застосунку, що має тривалий вплив на зорову систему людини. У

зв’язку з цим, у програмному застосунку реалізовано дві ключові функції профілактичного характеру:

- система сповіщень, рекомендацій про необхідність перерви при тривалій безперервній роботі за комп’ютером;
- режим «24/7» адаптивний, інтелектуальний режим, що автоматично змінює параметри зображення залежно від часу доби.

Мета сповіщень – попередити зорове та когнітивне перевантаження, спричинене тривалим безперервним переглядом екрана. В основі системи – ідея регулярних нагадувань, що спонукають зробити коротку перерву.

Медичні джерела рекомендують дотримуватися правила 60-60-60: кожні 60 хвилин дивитися на об’єкт, розташований на відстані 6 метрів, протягом 60 секунд. Це зменшує зорове напруження та сприяє зволоженню очей. Запропонована модель сповіщень зображена у таблиці 3.5.

Таблиця 3.5 – Модель сповіщень

Інтервал (хв.)	Мета сповіщення	Обґрунтування часу
30	Перший попереджувальний сигнал	Після пів години активної концентрації на зображенні дисплею
60	Посилене нагадування	Після однієї години безперервної роботи
120	Критичний рівень – рекомендована пауза	Межа безперервного навантаження на очі

Після кожного пройденого порогу часу, користувач отримуватиме сповіщення у вигляді повідомлення, рекомендації на екрані. Модель розрахована на те, що принаймні 2 з 3 нагадувань користувач сприйме серйозно і зробить коротку перерву. Це дозволить зменшити кумулятивне навантаження на зір до 40–50% за тривалий період активності.

24/7 – режим автоматичного регулювання параметрів зображення впродовж доби, базується на добовому (циркадному) ритмі людини. Протягом

дня рівень чутливості сітківки, гормональна активність та продуктивність зору змінюються. Програма адаптує дисплей до біологічного циклу користувача.

Модель розподілу параметрів упродовж доби представлено у таблиці 3.6.

Таблиця 3.6 – Модель режиму 24/7

Час	Яскравість	Контрас	Фільтр синього	Обґрунтування
06:00 – 08:00	75	75	75	Ранок, поступове пробудження
08:00 – 10:00	90	90	90	Перехід до активної денного користування дисплеєм
10:00 – 12:00	100	100	-	Активна фаза, максимальна продуктивність
12:00 – 16:00	120	120	-	День, пікова концентрація та освітлення
16:00 – 18:00	100	100	-	Активна фаза, максимальна продуктивність
18:00 – 20:00	90	90	90	Перехід до вечірнього режиму
20:00 – 22:00	75	75	75	Підготовка до сну, приглушене світло
22:00 – 06:00	50	50	50	Пізній вечір і ніч, мінімальне навантаження, мінімум синього світла

Моделювання добового навантаження з використанням 24/7 режиму надає такі переваги:

- зменшення яскравості та синього спектру у вечірній час дозволяє уникнути пригнічення мелатоніну – гормону сну;

- плавні переходи параметрів запобігають зоровому шоку, який виникає при різкій зміні яскравості;
- у поєднанні з системою сповіщень забезпечується рівномірне навантаження на очі протягом дня.

Моделювання системи сповіщень і 24/7 режиму показує, що комплексний підхід до збереження зорового комфорту є ефективним. Чітко визначені часові пороги, біологічно обґрунтовані параметри відображення та поведінкова модель користувача створюють надійну основу для автоматизованої турботи про здоров’я.

3.5 Висновки

У даному розділі було проведено моделювання та попереднє експериментальне дослідження основних функціональних компонентів програмного застосунку, зокрема – режимів роботи екрану, системи сповіщень про необхідність перерви та режиму автоматичного регулювання параметрів дисплея відповідно до часу доби (режим 24/7).

Проведено експериментальне дослідження семи режимів керування екраном, групою користувачів з оцінкою зорового комфорту, швидкості реакції системи та частоти використання.

Змодельована логіка роботи системи сповіщень, яка базується на рекомендованих інтервалах активності й передбачає поступове інформування користувача про необхідність зробити перерву;

Розроблено добову модель роботи режиму 24/7, що враховує добовий біоритм людини та забезпечує адаптивне зниження навантаження у вечірній і нічний час.

4 РОЗРОБКА ПРОГРМАНОГО ЗАСТОСУНКУ

4.1 Обґрунтування вибору мови програмування та середовища розробки

Вибір мови програмування для розробки програми під Windows залежить від різноманітних факторів, зокрема від цілей проєкту, вимог до продуктивності, особливостей інтеграції з іншими компонентами, а також від рівня підготовки та досвіду команди розробників. Крім того, враховується наявність необхідних бібліотек та інструментів для спрощення процесу розробки.

Проведено огляд таких мов програмування :

- C#;
- C++;
- Python:
- Java.

C# – об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Поточна версія і дата останнього випуску: C# 13.0 (вересень 2024)[13].

Переваги:

- чистий синтаксис та висока швидкість розробки;
- широке використання в екосистемі Microsoft;
- інтеграція з .NET Framework або .NET Core.

Недоліки:

- менша продуктивність порівняно з мовами, що компілюються;
- обмежена переносимість коду через залежність від платформи .NET.

Python – інтерпретована, високорівнева мова програмування загального призначення. Поточна версія і дата останнього випуску: Python 3.13.3 (квітень 2025). Розробник або власник: Python Software Foundation [14].

Переваги:

- простий синтаксис та висока читабельність коду;
- велика кількість сторонніх бібліотек і фреймворків;
- підтримується на багатьох платформах.

Недоліки:

- зазвичай менша швидкодія в порівнянні з компільованими мовами.

Java – об'єктно-орієнтована мова програмування, яка використовується для розробки багатоплатформних додатків. Поточна версія і дата останнього випуску: Java 24 (березень 2025). Розробник або власник: Oracle Corporation [15].

Переваги:

- велика переносимість коду завдяки віртуальній машині Java (JVM);
- широке використання в корпоративному середовищі;
- багата екосистема фреймворків та інструментів для розробки.

Недоліки:

- зазвичай менша швидкодія порівняно з мовами, що компілюються;
- великі вимоги до пам'яті та ресурсів.

C++ – загальною компільованою мовою програмування, яка поєднує в собі властивості низькорівневої мови програмування (наприклад, доступ до пам'яті) і високорівневої мови програмування (наприклад, об'єктно-орієнтованого програмування). Поточна версія і дата останнього випуску: C++23 (жовтень 2024). Мова C++ була створена Б'ярном Страуструпом у 1983 році. Зараз стандарт розробляється ISO/IEC JTC1/SC22/WG21 [16];

Переваги:

- висока продуктивність і швидкодія;
- прямий доступ до системних ресурсів і API;
- розширені можливості оптимізації та контролю над ресурсами пам'яті;
- широкий вибір бібліотек і фреймворків.

Недоліки:

- вимагає більшого обсягу коду для досягнення тих самих результатів, що може призвести до більш тривалого розроблення;

Порівняльний аналіз мов програмування представлено у таблиці 4.1.

Таблиця 4.1 – Порівняльний аналіз мов програмування

	C#	Python	Java	C++
Прямий доступ до API	1	1	1	1
Широкий вибір бібліотек і фреймворків	1	1	1	1
Переносимість коду	0	1	1	1
Статична типізація	1	0	1	1
Інструкція goto	1	0	0	1
Інтеграція з .Net	1	0	1	0
Простий синтаксис	0	1	0	1
Сумарний коефіцієнт	5	4	5	6

Провівши детальний огляд мов програмування для розробки програмного застосунку, було обрано мову C++, оскільки вона найкраще відповідає всім поставленим вимогам, забезпечує високу продуктивність, гнучкість та ефективність розробки, перевершуючи інші альтернативні рішення.

Середовище розробки (IDE) для мови програмування C++ – це інтегрована система, яка забезпечує розробника необхідними інструментами для написання, налагодження, тестування та підтримки програмного застосунку. У межах роботи було проведено аналіз та опис чотирьох найпопулярніших середовищ розробки для мови C++, кожне з яких має свої особливості, переваги та сфери застосування.

Оглянуто такі IDE:

- Visual Studio;
- CLion;
- Code::Blocks;
- Eclipse CDT.

Visual Studio – інтегроване середовище розробки (IDE), створене компанією Microsoft. Воно підтримує розробку різноманітних програмних застосунків, зокрема для платформи Windows, а також для інших середовищ, таких як мобільні пристрої та веб-платформи. Visual Studio надає розробникам широкий спектр інструментів для написання, налагодження, тестування та розгортання програмного забезпечення. Інтерфейс середовища є зручним та функціональним, що робить процес розробки ефективним (див. рисунок 4.1) [17].

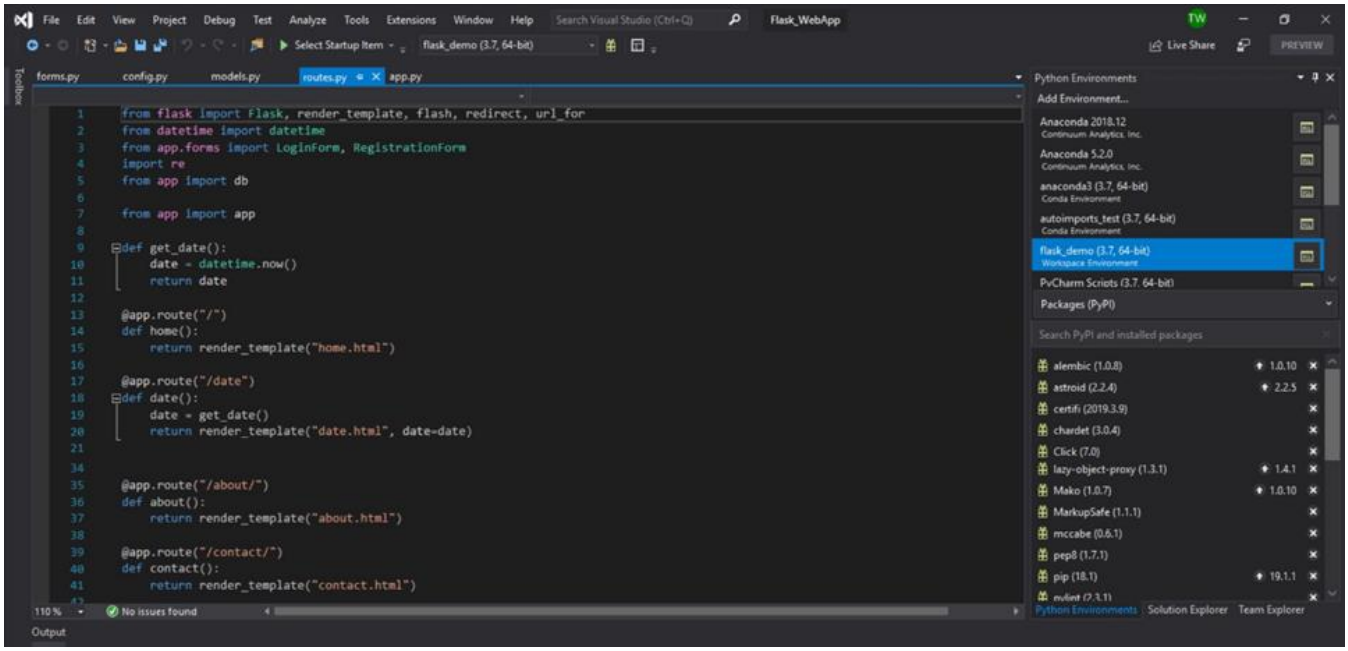

Рисунок 4.1 – Середовище розробки Microsoft Visual Studio

Переваги:

- широкий набір інструментів для розробки, відлагодження та профілювання програм на C++;
- інтеграція з платформою Windows і підтримка широкого спектру технологій;
- велика спільнота користувачів та доступність різноманітних ресурсів.

Недоліки:

- великі вимоги до ресурсів системи, що може призвести до повільної роботи на старих або менш потужних комп'ютерах.

CLion – інтегроване середовище розробки, розроблене компанією JetBrains, яке спеціалізується на розробці на мові C++ (див. рисунок 4.2) [18].

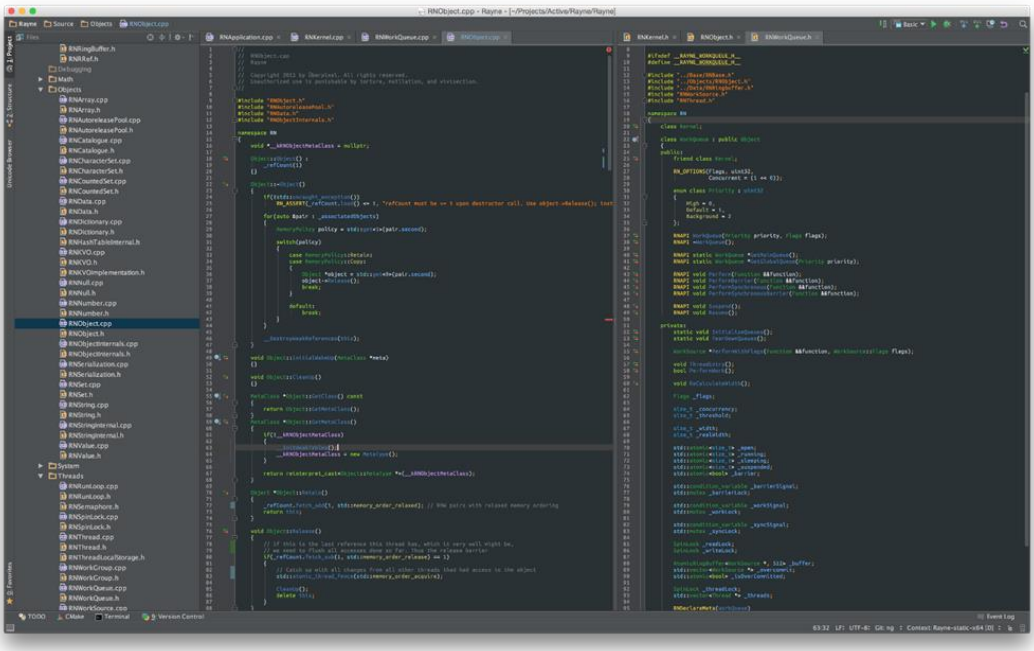

Рисунок 4.2 – Середовище розробки CLion

Переваги:

– підтримка різних платформ, включаючи Windows, Linux і macOS;
– інтелектуальний аналіз коду, автодоповнення і автоматичне відстеження помилок;
– інтеграція з різними системами контролю версій та іншими інструментами розробки.

Недоліки:

– відносно висока вартість ліцензії для комерційного використання.

Code::Blocks – вільне та відкрите інтегроване середовище розробки, доступне для Windows, Linux і macOS (див. рисунок 4.3) [19].

Переваги:

– простий у використанні та налаштуванні, підтримка багатьох плагінів;
– широкі можливості налаштування та розширення;
– підтримка компіляторів GCC, Clang, MSVC.

Недоліки:

– обмежені можливості для великих та складних проектів порівняно з іншими IDE.

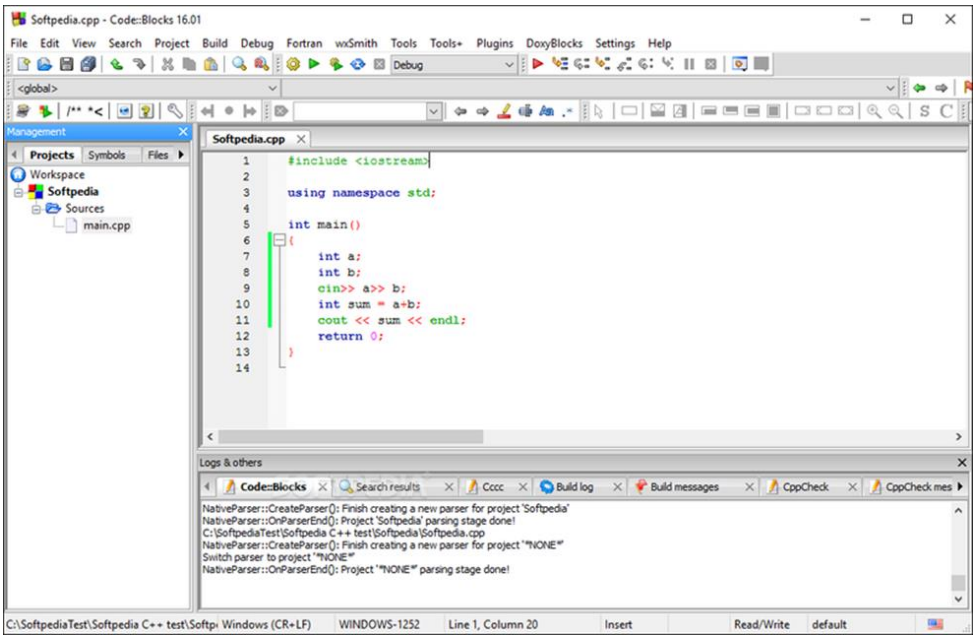

Рисунок 4.3 – Середовище розробки Code::Blocks

Eclipse CDT – це інтегроване середовище розробки, яке підтримує розробку мовою C++ за допомогою плагіну C/C++ Development Tooling (див. рисунок 4.4) [20].

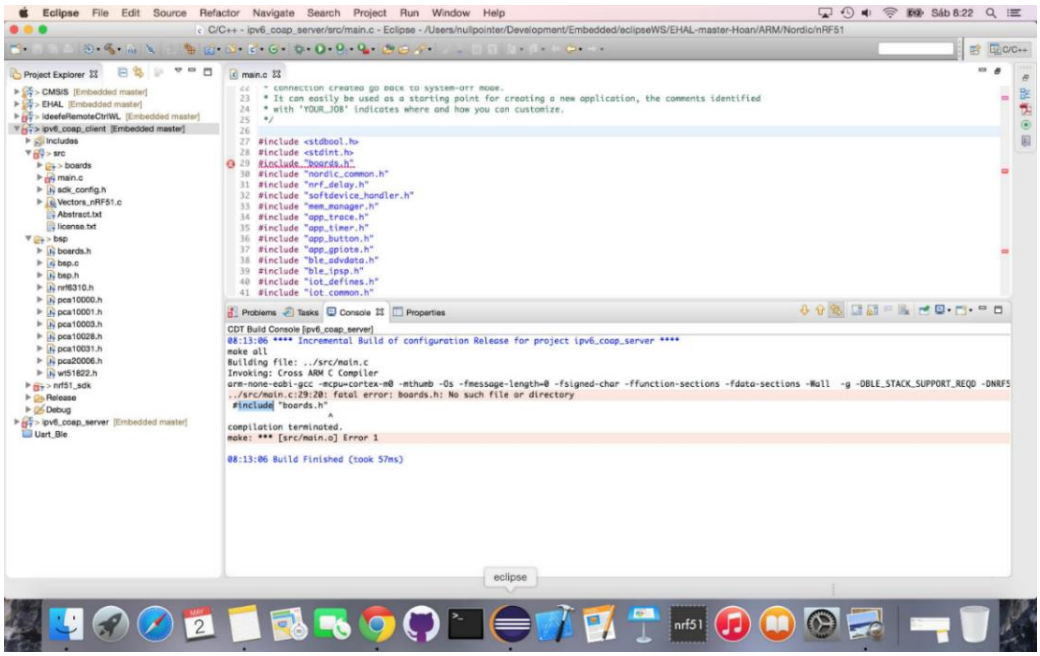

Рисунок 4.4 – Середовище розробки «Eclipse CDT»

Переваги:

- відкрите джерело та велика спільнота користувачів;
- підтримка багатьох функцій, таких як автодоповнення, відлагодження, підтримка системи контролю версій тощо.

Недоліки:

- менш інтуїтивний і менш потужний порівняно з іншими IDE, такими як Visual Studio або CLion.

Порівняльний аналіз середовищ розробки представлено у таблиці 4.2.

Таблиця 4.2 – Порівняння середовищ розробки

IDE	MS Visual Studio	CLion	Code::Blocks	Eclipse CDT
Браузер класу	1	1	1	1
Статичний аналіз коду	1	1	0	1
Debugger	1	1	1	1
Покриття коду	1	1	0	1
Конструктор GUI	1	0	1	1
Профайлер	1	0	1	1
Сумарний коефіцієнт	6	4	4	6

Оглянувши доступні середовища розробки для програмного застосунку з використанням мови C++, обрано середовище Microsoft Visual Studio, адже воно найбільш підходить для розробки програмного застосунку

4.2 Режими роботи екрану та користувацьке налаштування

Однією з головних функцій програмного застосунку є зміна режимів роботи екрану, що дозволяє адаптувати його до параметрів різних умов використання, покращити зоровий комфорт та знизити навантаження на зір. Зміна режимів виконується як автоматично так і вручну через інтерфейс програмного застосунку.

Користувач має можливість змінювати такі характеристики зображення:

- яскравість (current_brightness), загальний рівень світності зображення;
- контраст (current_contrast), різкість переходів між темними і світлими областями;
- кольоровий баланс (current_red, current_green, current_blue), регулювання окремих каналів RGB.

Усі параметри виражаються у відсотках (від 0 до 200) зі стандартним значенням 100, яке відповідає нейтральному режиму. Ці значення зберігають поточний стан параметрів екрану у відсотках, та оновлюються під час перемикання режимів або при ручному користувацькому налаштуванні.

Всі зміни режимів виконуються через основну функцію UpdateGammaRamp(), яка обчислює нову таблицю гами для кожного кольорового каналу на основі поточних налаштувань. Застосування нового вигляду таблиці гами до дисплея здійснюється функцією SetDeviceGammaRamp, що змінює вивід на екрані в реальному часі.

Ключовими етапами розрахунку є:

- обчислення базової яскравості з урахуванням контрасту;
- зміна кольору через насиченість і масштабування RGB;
- приведення значень до 16-бітної шкали (0–65535) для кожного з 256 рівнів.

Блок-схема зміни налаштувань параметрів екрану відповідно до режимів представлено на рисунку 4.5

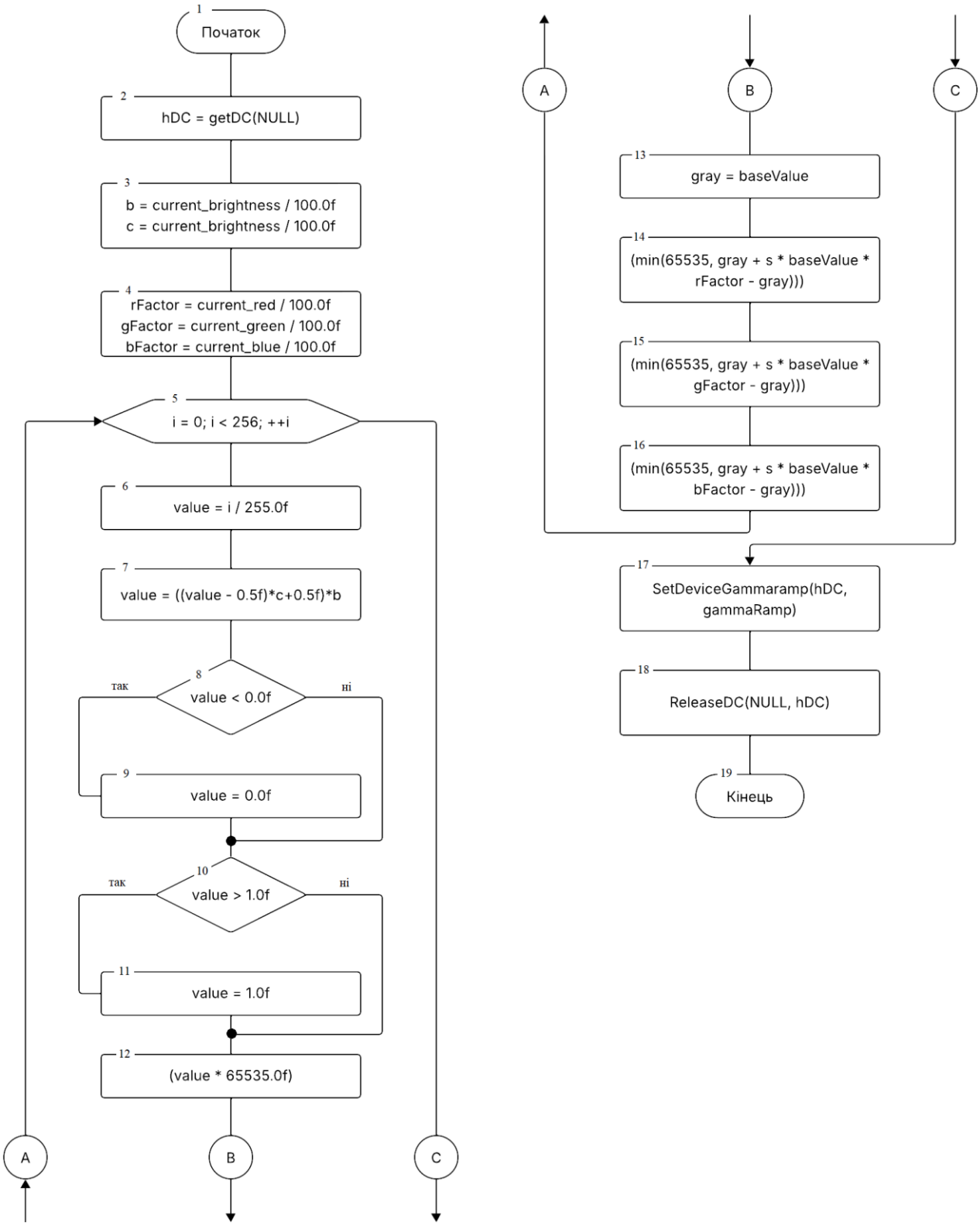

Рисунок 4.5 – Блок-схема методу UpdateGammaRamp

Для користувачів реалізовано 6 унікальних режимів роботи екрану: Шутер, Декорації, Книга, Кіно, Захист очей, Ніч та стандартний режим. Кожен з яких реалізується відповідною кнопкою інтерфейсу. Характеристики режимів:

Таблиця. 4.3 – Характеристики режимів

Режим	Яскравість	Контраст	R	G	B	Призначення
За замовчування	100	100	100	100	100	Стандартний нейтральний режим
Шутер	105	105	105	100	100	Для комфортної гри у відеоігри
Декорації	90	120	95	105	100	Для перегляду природи, пейзажів
Книга	90	80	100	100	80	Для читання великих текстових матеріалів
Кіно	90	110	100	100	100	Для перегляду відеоматеріалів
Захист очей	90	90	100	100	60	Захист очей, м’яке тепле світло
Ніч	90	70	100	100	90	Комфортна взаємодія з дисплеєм вночі

Приклад роботи режимів представлено на рисунках 4.6 – 4.11 на зображеннях видно зміну налаштувань основних параметрів дисплею.

Режим захисту очей призначений для зменшення випромінювання шкідливого блакитного світла. Він працює шляхом регулювання значень гами, насичення синього кольору та рівня яскравості, що сприяє зниженню навантаження на очі та підвищенню комфорту під час тривалої роботи з екраном (див. рисунок 4.6).

Рисунок 4.6 – Режим «Захист очей»

Нічний режим екрана призначений для зменшення яскравості та контрастності зображення, що забезпечує комфортніший перегляд у темряві та сприяє зниженню втоми очей під час роботи в умовах недостатнього освітлення (див. рисунок 4.7).

Рисунок 4.7 – Нічний режим

«Шутер» – режим з посиленою контрастністю та червоним каналом, оптимізований для динамічних ігор та сцен з великою кількістю деталей. Підвищує видимість об’єктів у темних ділянках зображення (див. рисунок 4.8).

Рисунок 4.8 – Режим «Шутер»

Режим «Книга» призначений для зменшення яскравості, контрасту й синього каналу, створений для комфортного читання з екрана. Зменшує навантаження на очі при довготривалому перегляді тексту (див. рисунок 4.9).

Рисунок 4.9 – Режим «Книга»

«Декорації» – режим з підвищеним контрастом і зеленим кольором, призначений для перегляду пейзажів, зображень та відео. Відтворює яскраве, насичене середовище з акцентом на візуальну глибину (див. рисунок 4.10).

Рисунок 4.10 – Режим «Декорації»

Режим «Кіно» призначений для підвищення контрасту та яскравості із нейтральним кольоровим балансом, призначений для перегляду відеоконтенту. Забезпечує глибокі тіні та чітку деталізацію зображення (див. рисунок 4.11).

Рисунок 4.11 – Режим «Кіно»

Параметри кожного режиму налаштовані для виконання конкретної задачі користувача, якщо у користувача є унікальна задача для якої не створено режиму, користувач може створити свій власний режим, змінивши параметри під власні потреби.

Зміна параметрів вручну відбувається через відповідні trackBar, пов’язані з відповідними обробниками:

– trackBar_brightness_Scroll, зміна параметру яскравості;
– trackBar_contrast_Scroll, зміна параметру контрасту;
– trackBar_color_Scroll, зміна параметрів колірних каналів RGB.

Після зміни параметрів оновлюється таблиця гами і одразу застосовується до дисплею. Це дозволяє побачити результат миттєво.

Приклад виставлених користувацьких параметрів (див. рисунок 4.12):

– яскравість 80%
– контраст 120%
– R 80%, G 100%, B 80%.

Рисунок 4.12 – Застосовані користувацькі налаштування

Також реалізовано профілювання користувацьких налаштувань. Для цього реалізовано клас DisplayProfile.

Користувач обирає потрібні параметри у відповідних trackBar користувацьких налаштувань, після цього натискає кнопку інтерфейсу «зберегти профіль», вводить назву створеного профілю та зберігає. Система обмежена створенням до 5 користувацьких профілів, задля коректної та швидкої роботи програми (див. рисунок 4.13).

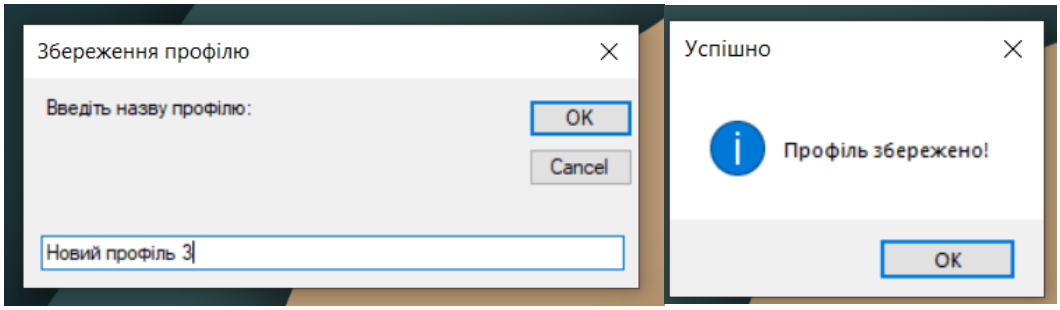

Рисунок 4.13 – Створення профілю з користувацьким налаштуванням

При збереженні профіля він автоматично записується у файлі «profiles.dat», який створюється у папці програмного застосунку [21] (див. рисунок 4.14).

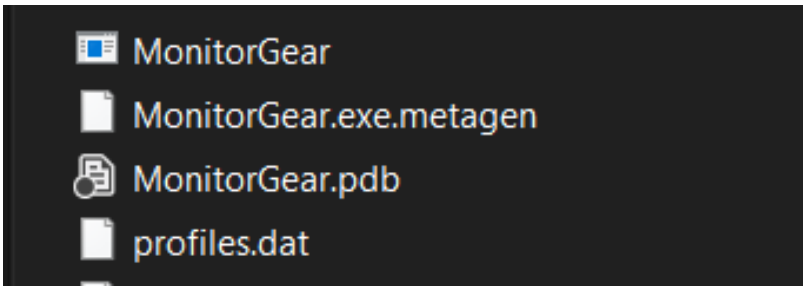

Рисунок 4.14 – Створений файл «profiles.dat» після створення профілей

Увімкнути профіль можливо через інтерфейс вибравши його зі списку у comboBoxProfiles та натиснувши кнопку «застосувати профіль», а також видалити натиснувши кнопку «видалити профіль», профіль автоматично видалиться зі списку та з файлу (див. рисунок 4.15).

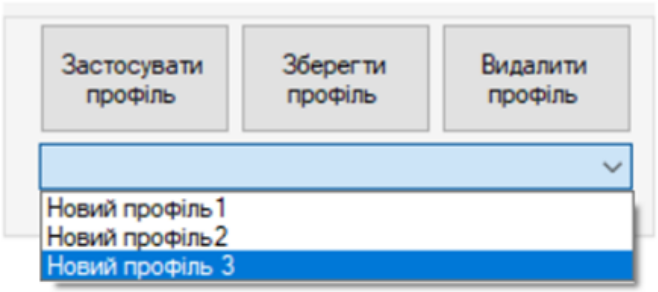

Рисунок 4.15 – Список збережених профілей

Таким чином, користувач має змогу не лише використовувати заздалегідь підготовлені режими роботи екрана, але й створювати, зберігати та перемикатися між власними індивідуальними, гнучко налаштованими профілями, що робить застосунок комфортним, персоналізованим інструментом адаптації відображення. Збереження профілів у файл дозволяє зберігати налаштування між сесіями.

4.3 Адаптивні режими роботи екрану

Інтелектуальне адаптивне керування параметрами дисплея є ключовою особливістю розробленого програмного застосунку. Застосунок реалізує два незалежних механізми адаптації параметрів екрану до умов використання:

– адаптивний режим за часом доби (режим 24/7);
– адаптація до активного вікна (контекстно-залежний режим).

Кожен з режимів працює на основі таймерів і автоматично змінює параметри gammaRamp згідно з виявлених умов.

Механізм адаптації за часом доби представляє у собі автоматичне регулювання параметрів дисплея, визначених для конкретного проміжку дня. Режим поєднує у собі як збільшення яскравості та контрасту у середині робочого дня, для максимальної уваги та концентрації, так і зниження яскравості, контрасту та синього колірного каналу для комфортної роботи у вечірню та ранкову пору. Діаграму роботи режиму представлено на рисунку 4.16. На ній зображено значення таких параметрів як яскравість та контраст, у відповідний час доби, також показано з якими значеннями та у який проміжок вмикається фільтр синього.

Значення параметрів режиму 24/7 встановлено відповідно до аналізу активності користувача, та спрямований оптимізувати робочий день перед екраном, адже до 10:00 та після 18:00 програма встановлює значення яскравості та контрасту нижче 100 відсотків задля комфортного переходу, концентрації очей, також з 22:00 до 8:00 зображення на екрані теплішає завдяки фільтру синього кольору, такий підхід запобігає погіршенню зору у ранковий та вечірній період, проблем зі сном та головного болю. Під час активної фази користування з 12:00 по 16:00 параметри яскравості та контрасту збільшуються для максимальної концентрації, уваги та чіткості над завданнями поставленими у роботі з монітором.

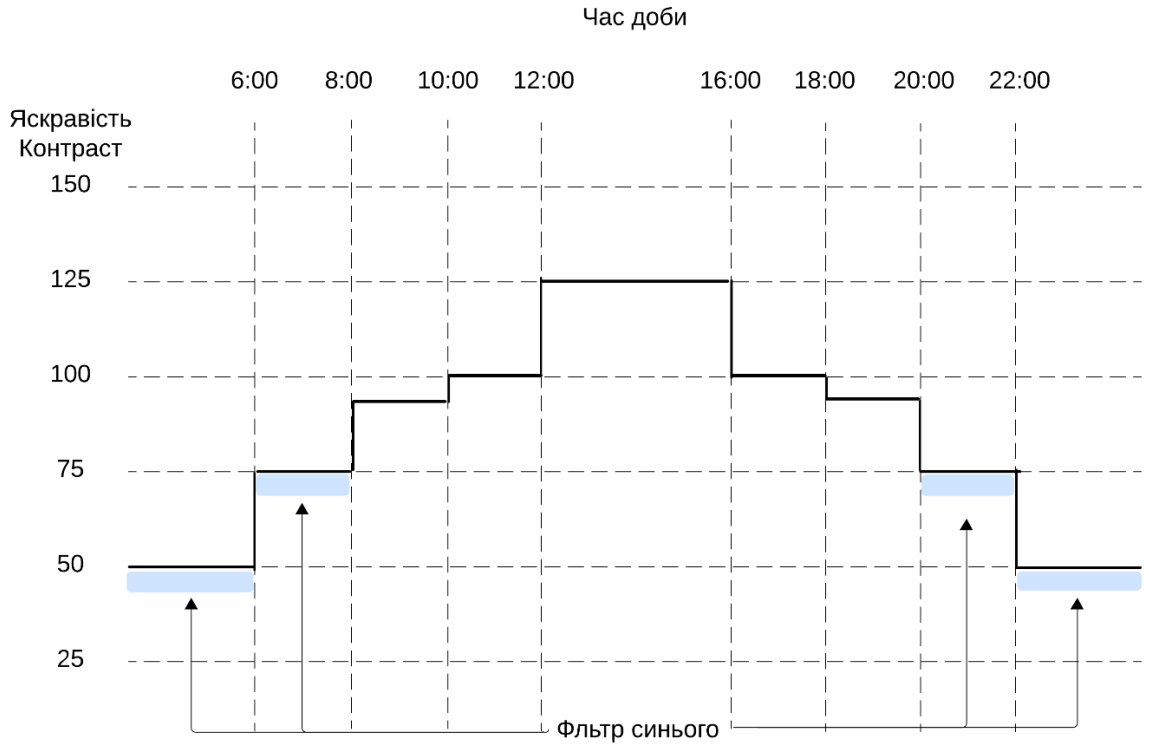

Рисунок 4.16 – Діаграма роботи режиму 24/7

Режим вмикається та вимикається відповідними кнопками в інтерфейсі програмного застосунку:

– button_enableAuto;

– button_disableAuto.

Після ввімкнення функція викликається щохвилини, щоб перевірити час та змінити відповідно налаштування

Інтелектуальний адаптивний режим зміни параметрів відповідно до відкритого вікна – призначений для автоматизації зміни режимів екрану. Користувач може призначити кожному активному вікну відповідний користувацький профіль створений ним раніше. При ввімкненні нового вікна програма визначає що це вікно нове та до нього не застосовано жоден профіль.

Після виявлення нового активного вікна програма надсилає сповіщення із рекомендацією призначити вікну режим. Приклад виявлення нового вікна «Google Chrome» (див. рисунок 4.17).

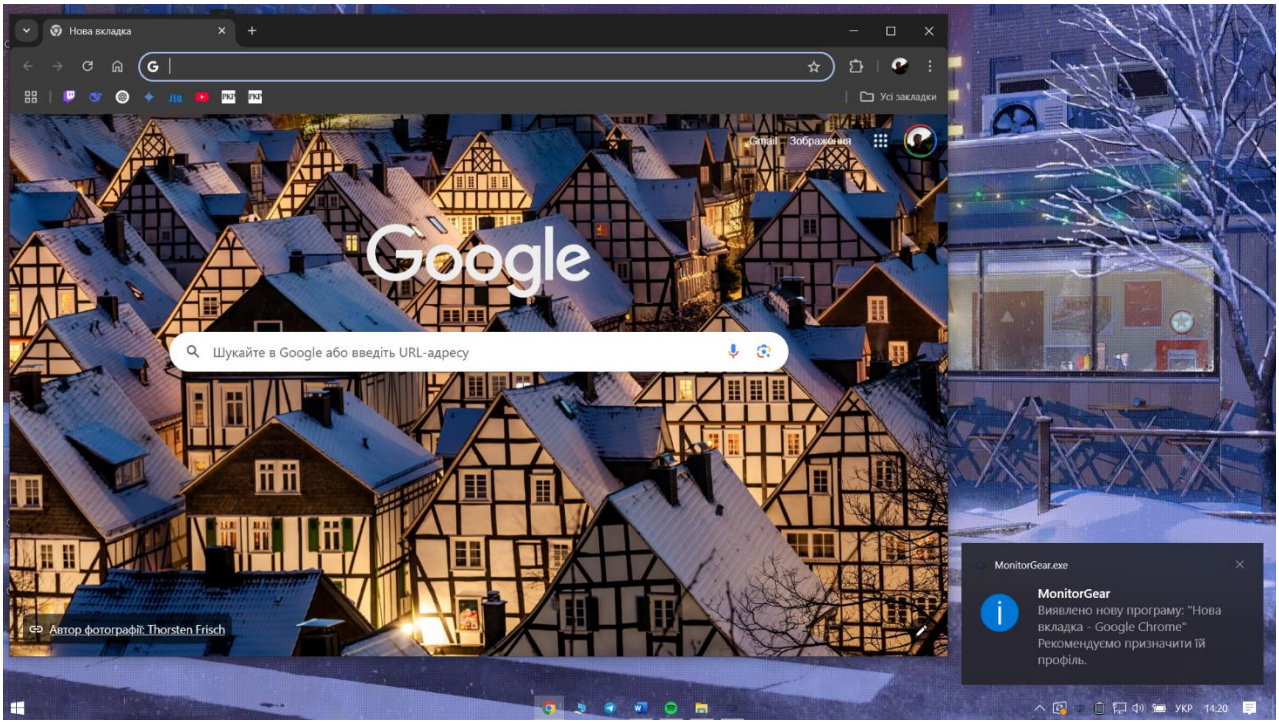

Рисунок 4.17 – Приклад сповіщення про нову програму

Для призначення профіля активній програмі чи вікну потрібно вибрати зі списку comboBoxProfiles потрібний профіль, вибрати з listBoxOpenWindows потрібне вікно чи програму та натиснути кнопку «Призначити профіль». Усі програм, вікна та призначені до них профілі відображаються у listBoxProfiles, вибравши необхідну прив’язку її можна видалити натиснувши відповідну кнопку. Створенні прив’язані профілі до програм зберігаються у файл «window_profiles.dat» (див. рисунок 4.18).

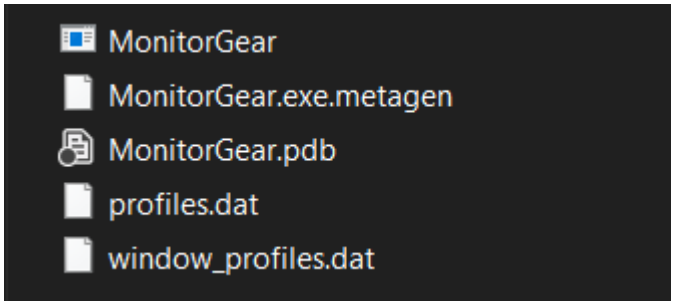

Рисунок 4.18 – Створений файл «window_profiles.dat»

Реалізація адаптивних, інтелектуальних режимів роботи екрану забезпечує регулювання параметрів зображення без участі користувача. Автоматична адаптація до часу доби покращує ергономіку, а перемикання профілів залежно від активного вікна робить застосунок контекстно чутливим і зручним для щоденного використання. Такі рішення демонструють високий рівень інтеграції з середовищем ОС та орієнтованість на реальні потреби користувача.

4.4 Робота програмного застосунку у фоновому режимі

З метою зручності та безперервної фонової роботи у програмному застосунку реалізовано підтримку системного трею. Це дозволяє приховати головне вікно з екрану, але зберегти доступ до основних функцій через контекстне меню. Таким чином, користувач може швидко викликати необхідний режим або завершити роботу програми, не відкриваючи її повністю.

При закриті головного вікна, з’являється сповіщення про згортання програми у трей а не повне закриття (див. рисунок 4.19).

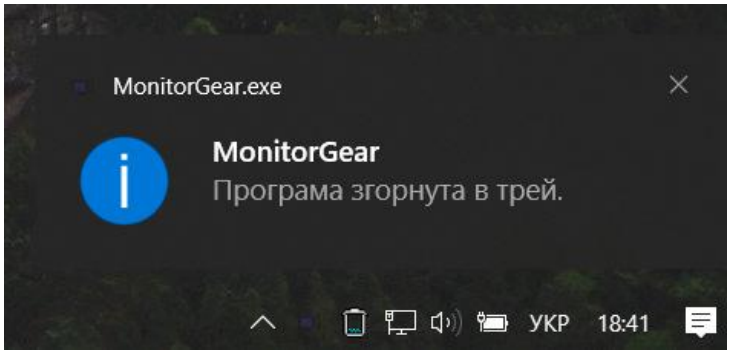

Рисунок 4.19 – Сповіщення про згортання у трей

Через трей можливо відкрити головне вікно програмного застосунку та закрити повністю програму, переглянути раніше створені користувацькі профілі та ввімкнути один з них, побачити чи активовано режим 24/7 (див. рисунок 4.20).

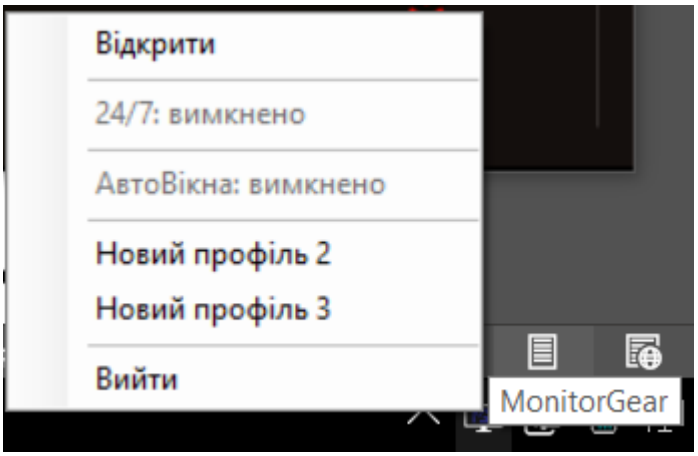

Рисунок 4.20 – Функціонал меню у треї

Інтеграція з системним треєм дозволяє програмі працювати у фоновому режимі, зберігаючи доступ до основних функцій та забезпечуючи непомітну, але ефективну присутність у системі. Такий підхід підвищує зручність використання, особливо для користувачів, які часто перемикаються між режимами або працюють з програмою тривалий час.

4.5 Система сповіщень рекомендацій про перерву

Однією з функціональних переваг розробленого програмного застосунку є вбудована система нагадувань, що інформує користувача про тривале використання певного режиму екрану без перерви. Це спрямовано на підтримку гігієни зору, зменшення втоми очей та підвищення ергономічності взаємодії з комп’ютером.

Систему сповіщень інтегровано у клас таймер який зв’язаний з інструментами інтерфейсу та сповіщеннями.

При ввімкнені користувачем одного з режимів чи профілів, вмикається таймер, який засікає безперервний час користування відповідним режимом чи профілем. При зміні режиму або натисканні кнопки «За замовчування», таймер

зупиняється, це дозволяє перезапускати облік часу при зміні активного профілю, не накопичуючи помилкових значень.

При проходженні певного часу заданого системою, користувач отримує попередження про необхідність перерви задля запобіганню проблем із здоров'ям:

– у головному вікні програми, де зображено час безперервного користування режимом та повідомлення попередження;
– на екрані через сповіщення (див рисунок 4.21).

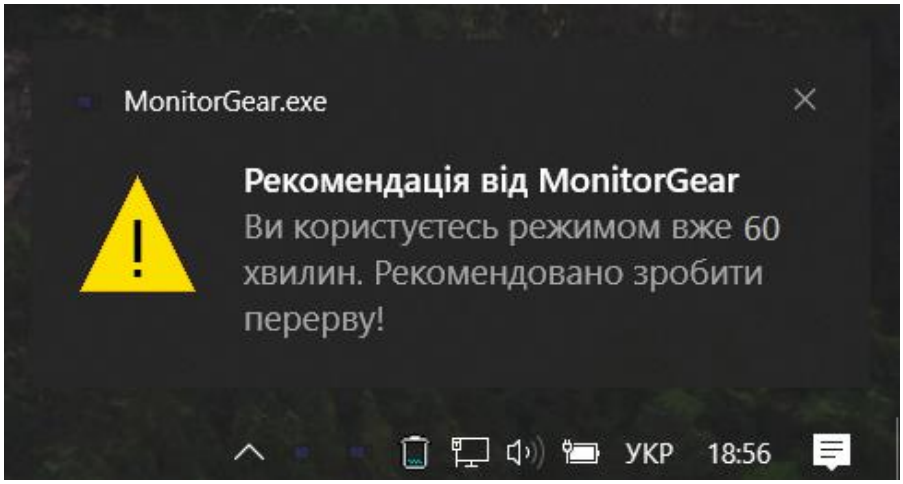

Рисунок 4.21 – Сповіщення про перерву

Сповіщення надходять через годину користування режимом чи профілем, дві години та чотири.

Реалізована система сповіщень є важливою частиною інтелектуального застосунку. Вона надає динамічний зворотний зв'язок щодо тривалості роботи в конкретному режимі, сприяє покращенню умов праці за комп'ютером і відповідає сучасним підходам до турботи про здоров'я користувача.

4.6 Розробка інтерфейсу користувача

Інтерфейс користувача (UI) є одним із ключових елементів будь-якого програмного застосунку, оскільки саме він визначає спосіб взаємодії користувачів із продуктом. Якісний та продуманий інтерфейс значною мірою впливає на загальне враження від використання програми, рівень задоволеності користувачів та ефективність виконання завдань, для яких створено застосунок. Важливість створення зручного, інтуїтивно зрозумілого та естетично

привабливого інтерфейсу неможливо переоцінити, адже саме UI виступає посередником між функціональністю системи та її кінцевими користувачами. Належна увага до розробки інтерфейсу сприяє підвищенню продуктивності роботи користувачів, зменшенню кількості помилок та формуванню позитивного користувацького досвіду [22].

Ключові елементи, які роблять інтерфейс користувача надзвичайно важливим є привабливий зовнішній вигляд, естетично оформлений інтерфейс відіграє важливу роль у створенні першого позитивного враження про програмний продукт. Гармонійне поєднання кольорів, продумане розташування елементів та візуальна привабливість інтерфейсу здатні зацікавити користувачів та стимулювати їх до активнішого використання програми.

Зручність використання, інтерфейс повинен бути інтуїтивно зрозумілим, простим у навігації та не вимагати від користувача зайвих зусиль для виконання основних дій. Чим менше часу користувач витрачає на освоєння інтерфейсу і виконання ключових функцій, тим ефективніше він досягає своїх цілей при роботі з програмним застосунком.

Інтерфейс повинен виконувати усі необхідні функції програмного застосунку. Усі поля, кнопки, меню і т.д повинні працювати належним чином [23].

З огляду на дані вимоги, інтерфейс програмного застосунку відповідає усім сучасним вимогам з урахуванням принципів ергономіки, інтуїтивної навігації та естетичної візуалізації, орієнтовану на темну колірну схему. Застосунок складається з 5 основних вкладок, кожна з яких має унікальне функціональне наповнення та іконку для швидкої ідентифікації.

У першій вкладці реалізовано блок керування режимами роботи екрану, для кожного режиму передбачено коротких опис та ідентифікатори зміни основних параметрів. Активний режим підсвічується зеленою лінією. Також виводиться таймер активності режиму та повідомлення про необхідність перерви у використані, що підвищує рівень турботи про зорове здоров’я користувача (див. рисунок 4.22).

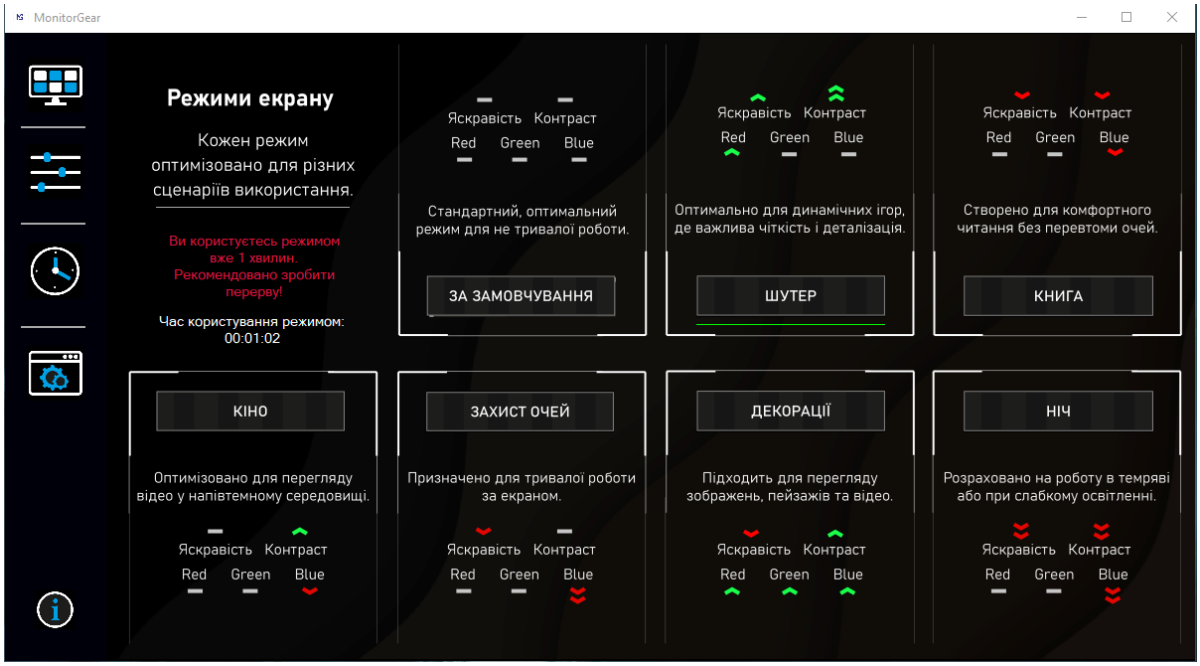

Рисунок 4.22 – Вкладка режимів роботи екрану

У другій вкладці розташовано модуль керування користувацьким налаштуванням параметрами дисплея. Кожен параметр змінюється за допомогою трекбарів із позначками відсотків. Ліворуч розташовано модуль створення 5 профілів із користувацьким налаштуванням (див. рисунок 4.23).

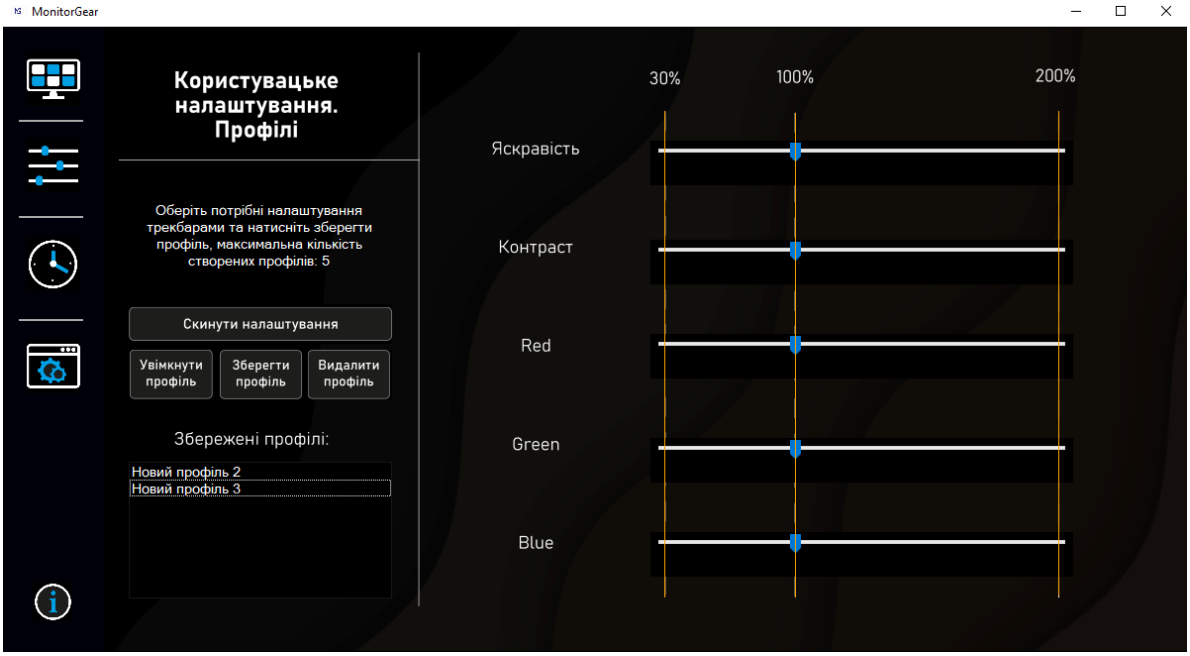

Рисунок 4.23 – Вкладка користувацького налаштування

У третій вкладі інтерфейс режиму 24/7 – автоматичної адаптації параметрів дисплею відповідно до часу доби. Користувачу демонструється графік зміни яскравості, контрасту та фільтру синього. Логіка налаштувань відповідає циркадному ритму людини [24](див. рисунок 4.24).

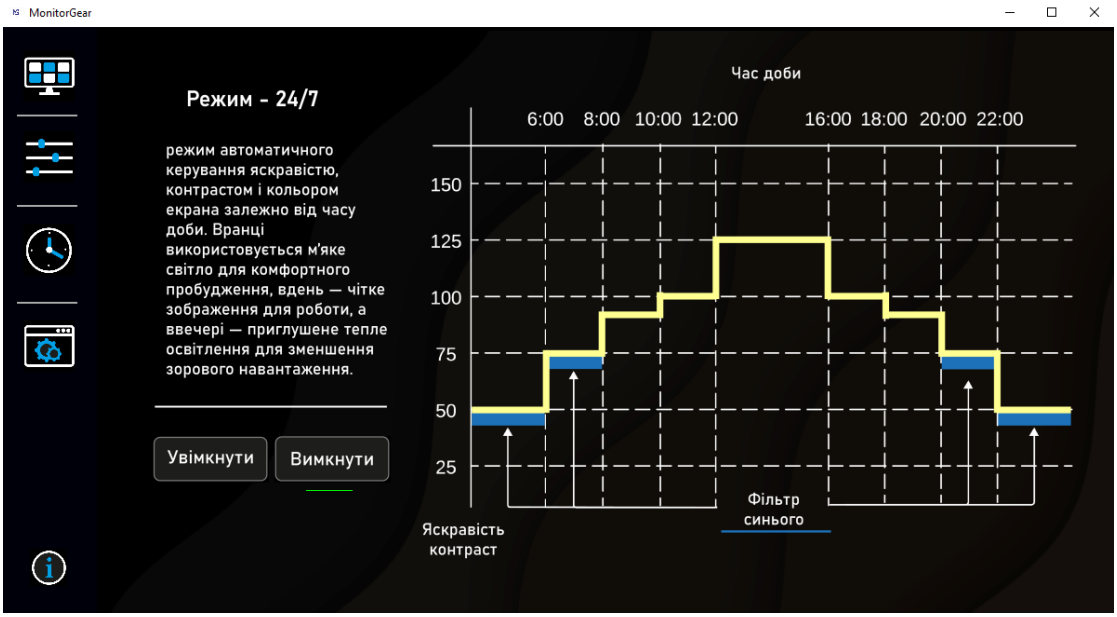

Рисунок 4.24 – Інтерфейс режиму 24/7

Четверта вкладка відповідає режиму автозміни параметрів відповідно до активного вікна. У списках користувачу показується доступні профілі, активні вікна та створені прив1язки профілів до вікон (див. рисунок 4.25).

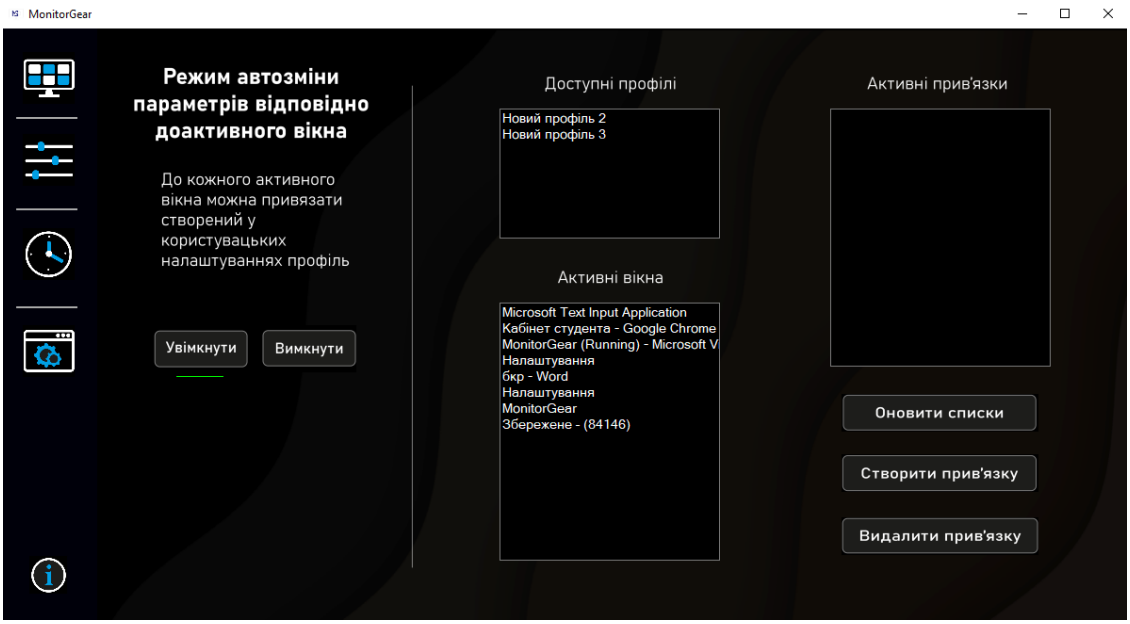

Рисунок 4.25 – Вкладка режиму автозміни до активних вікон

Також реалізовано вікно «Про застосунок» у якому представлено іконку програмного застосунку, назву, версію, розробника, короткий опис функціоналу (див. рисунок 4.26)

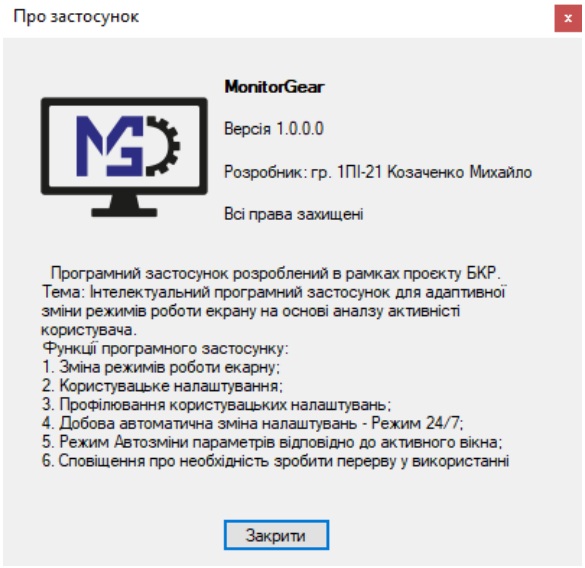

Рисунок 4.26 – Вікно «Про застосунок»

Використання темного інтерфейсу з неоновими акцентами забезпечує мінімальне навантаження на зір і відповідає сучасним стандартам UI-дизайну. Навігаційна панель зліва дозволяє швидко перемикатися між вкладками, кожна кнопка та графічний елемент підібрані з урахуванням зручності взаємодії та візуальної читабельності.

4.7 Тестування програмного застосунку

Тестування програмного застосунку є важливим етапом розробки, що дозволяє оцінити функціональність та перевірити відповідність очікуваним сценаріям використання.

Мета тестування: перевірити коректність роботи кожного режиму та модуля, визначити час реакції системи на зміну параметрів, оцінити зручність взаємодії з інтерфейсом, перевірити роботу адаптивних інтелектуальних режимів, переконатися у стабільності при тривалому використанні [25].

При функціональному тестуванні було протестовано усі ключові функції, а саме активацію 7 режимів роботи екрану, оновлення gammaRamp. Підтверджено

правильну зміну яскравості, контрасту, RGB у всіх режимах роботи екрану, оновлення gammaRamp виконується без помилок, навіть при частому перемиканні, усі режимні кнопки у першій вкладці функціонують коректно, індикатори параметрів відповідають очікуванням. (див. рисунок 4.27).

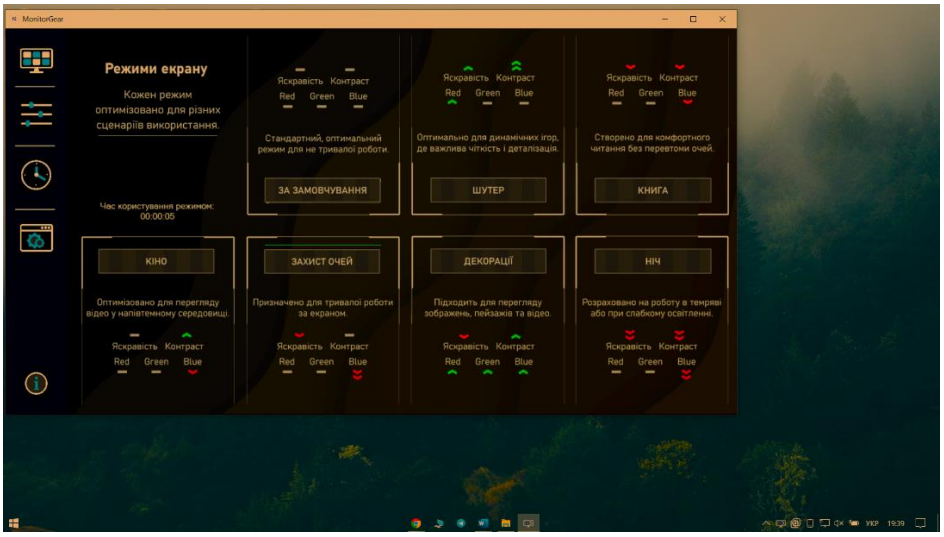

Рисунок 4.27 – Коректна робота режиму «Захист очей»

Проведено тестування користувацького налаштування, робота трекбарів та їх взаємодія з системними параметрами відповідає очікуваним результатам. Функціонал створення, видалення, увімкнення профілей працює без помилок. Збереження профілів та обмеження до 5 підтверджено (див. рисунок 4.28).

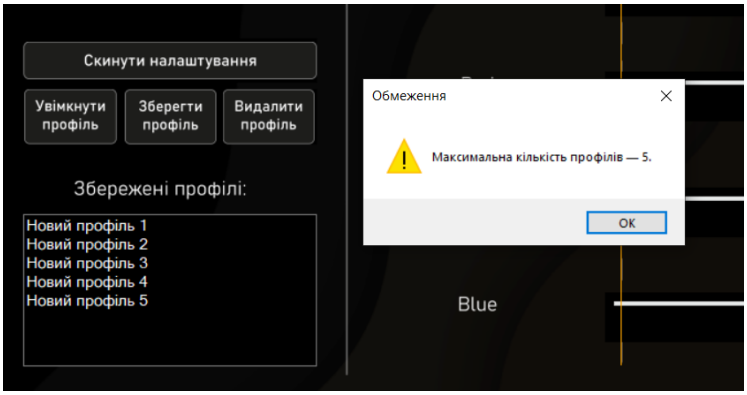

Рисунок 4.28 – Обмеження створення >5 профілів

Підтверджено не можливість створення користувацького профілю без назви (див. рисунок 4.29)

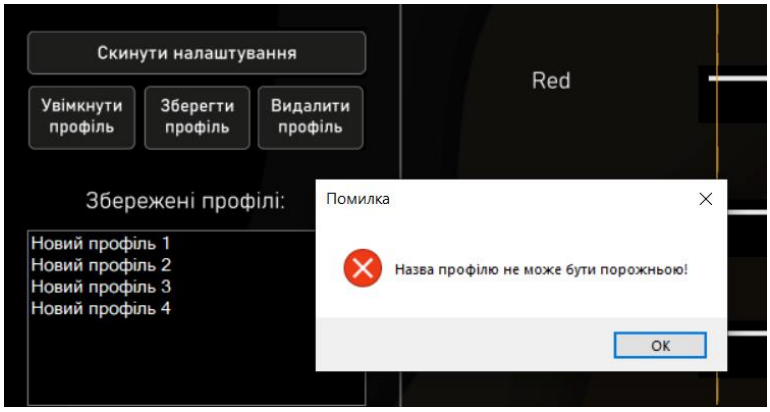

Рисунок 4.29 – Помилка при створенні профілю без назви

Проведено симуляцію добового циклу режиму 24/7, в якому: програма коректно змінює параметри відповідно до часу доби, значення яскравості, контрасту та фільтру синього відповідають таблиці описаній у розділі 3.4. Час оновлення параметрів – 5 секунд, спрацювання без затримок (див. рисунок 4.30).

Рисунок 4.30 – Спрацювання режиму 24/7 о 20:00

Проведено тестування модуля визначення активного вікна та автоматичного застосування відповідного профілю, працює стабільно. Усі перевірені програми (браузер, текстовий редактор, ігрові вікна) були розпізнані коректно, профіль застосовувався миттєво без додаткові взаємодії користувача (див. рисунок 4.30).

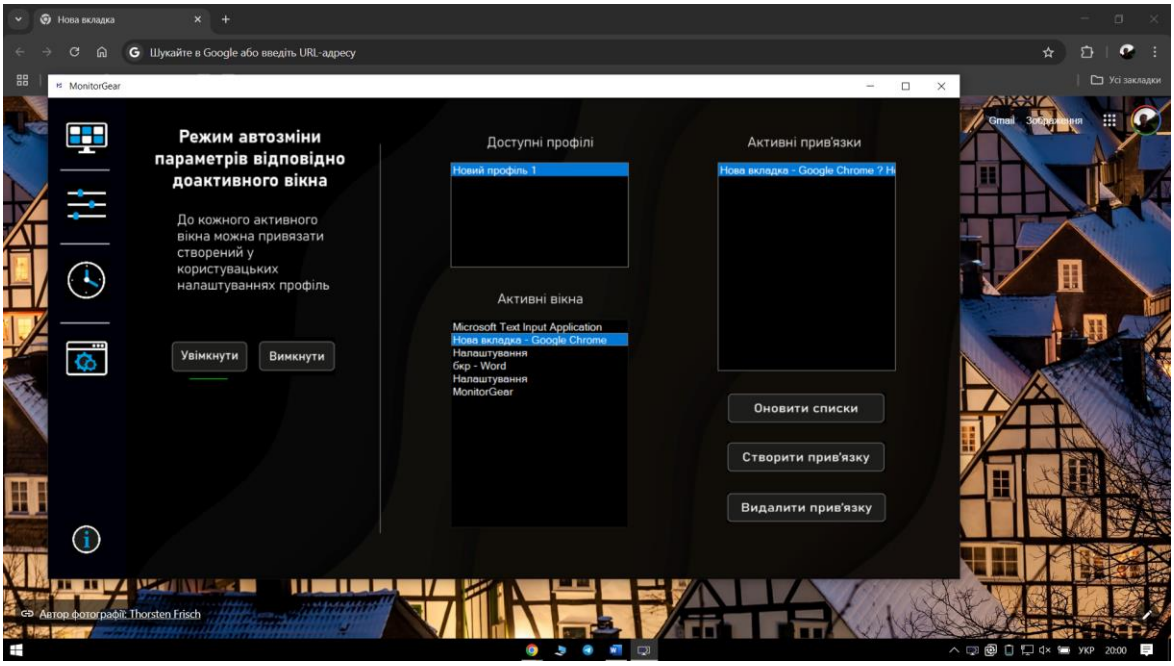

Рисунок 4.30 – Визначення активних вікон та автоматична зміна

Систему сповіщень, рекомендацій про перерву та усі сповіщення від програми відображаються коректно (див. рисунки 4.31 – 4.32).

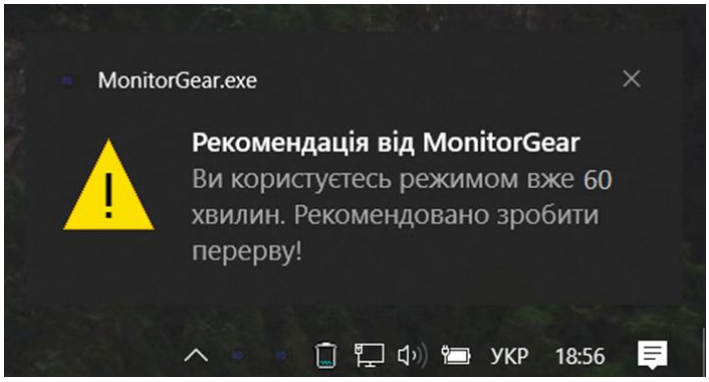

Рисунок 4.31 – Сповіщення про необхідність перерви

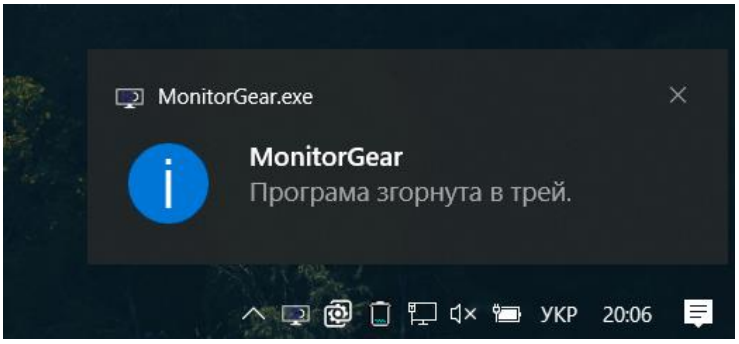

Рисунок 4.31 – Сповіщення про роботу програми у треї

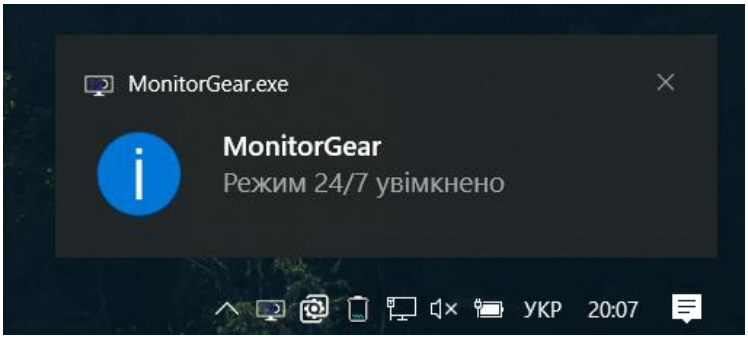

Рисунок 4.32 – Сповіщення про активацію режиму 24/7

При роботі програми у фоновому режимі з увімкненим режимом 24/7, режимом автозміни параметрів відповідно до активного вікна та таймері активності програма майже не навантажує систему, використовуючи 206,5 мб пам’яті пристрою (див. рисунок 4.33)

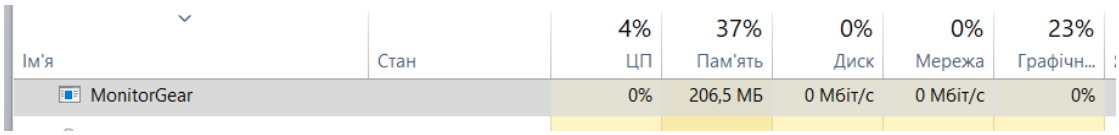

Рисунок 4.33 – Навантаження системи при фоновій роботі режимів

При тестуванні програмного застосунку на кількох пристроях (2 ноутбуки та 1 ПК) програма демонструвала хороші показники (див. таблиця 4.4).

Таблиця 4.4 – Навантаження на різних пристроях

Пристрій	Ноутбук 1	Ноутбук 2	ПК
1	2	3	4
Конфігурація	Intel i5, 8 Гб RAM, SSD, Win 10	AMD Ryzen 5, 16 Гб RAM, SSD, Win 11	Intel i7, 16 Гб RAM, HDD, Win 10
Час безперервної роботи	8 год. 30 хв.	6 год. 30 хв.	10 год. 30 хв.
Середнє споживання RAM	~190-200 Мб	~200-208 Мб	~190-198 Мб

Продовження таблиці 4.4 – Навантаження на різних пристроях

1	2	3	4
Середнє завантаження GPU	0.2 – 0.6%	0.4 – 1 %	0.1 – 0.3 %
Середній час реакції системи	~300 мс.	~ 305 мс.	~ 294 мс.
Проблеми/збої	-	-	-

На усіх пристроях програмний застосунок працював стабільно при тривалому безперервному використані, середнє споживання пам’яті не перевищувало 208 Мб., час реакції системи залишався стабільним у межах 294 – 305 мс., збої, витоки пам’яті чи значне зростання навантаження не зафіксовано.

4.8 Технічні вимоги та інструкція експлуатації

Розроблений програмний застосунок має невисокі системні вимоги і може ефективно працювати на більшості сучасних комп’ютерів. Мінімальні та рекомендовані характеристика наведено ц таблиці 4.5.

Таблиця 4.5 – Технічні вимоги до пристроїв

	Мінімальні системні вимоги	Рекомендовані системні вимоги
1	2	3
ОС	Windows 10 (64-bit)	Windows 10 / 11 (64-bit)
Процесор	Intel Core i3 / Ryzen 3	Intel Core i5 / Ryzen 5 або вище
Оперативна пам’ять	4 Гб.	8 Гб.
Відеокарта	Інтегрована / дискретна з підтримкою gammaRamp API	Будь-яка з підтримкою WinAPI GammaRamp
Вільне місце на диску	50 Мб.	50 Мб.

Продовження таблиці 4.5 – Технічні вимоги до пристроїв

1	2	3
Роздільна здатність екрану	1280x720	Full HD (1920×1080) або вище
Програмний застосунок не потребує підключення до інтернету для роботи		

Після запуску програми перед користувачем буде відкрита вкладка вибору режимів роботи екрану «Шутер», «Декорації», «Книга», «Кіно», «Захист очей», «Ніч» та «За замовчуванням», кожен режим вмикається відповідною кнопкою. Увімкнений режим підсвічується. При запуску програми активовано режим «За замовчування», у лівій частині програми знаходиться меню з вибором вкладок програми.

При переході у вкладку користувацьке налаштування користувачу надається доступ до користувацького налаштування, а саме зміни параметрів яскравості, контрасту та колірних каналів RGB відповідними трекбарами. Зміни параметрів трекбарів одразу відображається на екрані. Також користувач може створити власні профілі з відповідними налаштуваннями використовуючи відповідні кнопки функціоналу у вкладці.

При відкриті вкладки 24/7 користувачеві доступне увімкнення режиму 24/7 та перегляд графіку зміни параметрів даного режиму.

При переході у вкладку автоматичної зміни параметрів відповідно до активного вікна, відкривається можливість створення прив’язок профілів до активних вікон відповідними кнопками функціоналу.

При закриті програми вона продовжує роботу у фоновому режимі та користувачу доступне меню у треї, через яке користувач може відкрити та повністю закрити програму, переглянути и активні режими 24/7 та режим автозміни відповідно до активного вікна, та увімкнути створений профіль із користувацьким налаштуванням.

Програма не потребує підключення до інтернету, зберігає усі файли необхідні для роботи локально у папці програми, та сумісна з усіма сучасними версіями Windows.

4.9 Висновки

У даному розділі було розглянуто реалізацію функціональних компонентів інтелектуального програмного застосунку для адаптивного керування режимами роботи екрана. Запропоноване рішення забезпечує гнучке регулювання основних параметрів зображення – яскравості, контрастності та колірного балансу RGB – у реальному часі шляхом використання таблиці гами (gammaRamp), що дозволяє уникнути втручання в апаратну частину системи. Розроблено режими під конкретні задачі, розроблено функцію ручної зміни користувацьких налаштувань та створення користувацьких профілів на основі обраних параметрів. Реалізовано два інтелектуальні, адаптивні режими для автоматичної зміни параметрів екрану відповідно до часу доби та відкритої програми. Програма інтегрована із системним треєм та працює у фоновому режимі. Розроблено систем сповіщень, рекомендацій, про необхідність перерви після довготривалого користування обраним режимом. Реалізовано інтерфейс користувача та проведено тестування програмного застосунку.

ВИСНОВКИ

В межах бакалаврської кваліфікаційної роботи, було змодельовано та розроблено інтелектуальний програмний застосунок для адаптивного керування режимами роботи екрану на основі аналізу активності користувача засобами та можливостями мови програмування C++ у середовищі розробки Microsoft Visual Studio. Реалізація бакалаврської кваліфікаційної роботи відповідає усім методичним вказівкам.

На етапі аналізу існуючих рішень виявлено обмеження популярних програмних застосунків у сфері регулювання параметрів екрану, що обґрунтовує актуальність та доцільність розробки нового застосунку з розширеним функціоналом. Проведено аналіз методів розв’язання задачі та поставлено задачі проєкту.

Теоретична частина роботи охопила особливості зміни параметрів дисплеїв у ОС Windows з використанням API. Методи зміни параметрів зображення: яскравості, контрасту, колірних каналів RGB, а також можливостей їх інтеграції з графічним інтерфейсом користувача.

Моделювання та експериментальне дослідження дало змогу перевірити ефективність реалізованих режимів роботи екрану. Змодельовано інтегральний показник зорового навантаження та обґрунтовано необхідність системи рекомендацій щодо перерв. Сформульовано добову модель автоматичного інтелектуального регулювання параметрів екрану режимом «24/7».

У останньому розділі представлено реалізацію ключових модулів, зокрема: 7 режимів роботи екрану, гнучке користувацьке налаштування, створення користувацьких профілів, роботу у фоновому режимі, систему сповіщень рекомендацій, адаптивні інтелектуальні режими «24/7» та «контекстний режим», інтерфейс користувача, тестування та інструкцію користувача.

Поставлені в роботі мета та завдання повністю виконані, бакалаврська кваліфікаційна робота оформлена відповідно до методичних вказівок [26].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Козаченко М. Р., Чехместрук Р. Ю. Інтелектуальний програмний застосунок для адаптивного керування режимами роботи екрану на основі аналізу активності користувача. Матеріали: LIV Всеукраїнської науково-технічної конференції підрозділів ВНТУ, Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/23297 (Дата звернення: 25.03.2025).

2. Козаченко М. Р., Ліщинська Л. Б. Аналіз вимог до інтелектуального програмного застосунку для адаптивного керування параметрами екрана. Матеріали: IV Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи». Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/24805 (дата звернення: 28.03.2025).

3. Рейтинг операційних систем: веб-сайт. URL: https://dou.ua/forums/topic/44292/ (дата звернення: 19.02.25).

4. Audrey Cougnard-Gregoire, Benedicte M J Merle, Tariq Aslam, Johanna M Seddon. Blue Light Exposure: Ocular Hazards and Prevention-A Narrative Review. 2022. 20p.

5. Alan Colman1, Jun Han1, Asif Irshad. BehavDT: A Behavioral Decision Tree Learning to Build User-Centric Context-Aware Predictive Model. 2019. 8p. URL: https://www.researchgate.net/publication/337635880_BehavDT_A_Behavioral_Decision_Tree_Learning_to_Build_User-Centric_Context-Aware_Predictive_Model

6. Програмний застосунок f.lux: веб-сайт. URL: https://justgetflux.com/ (дата звернення: 06.04.2025).

7. Програмний застосунок ARMPURY CRATE: веб-сайт. URL: https://www.asus.com/ua-ua/support/faq/1037674/ (дата звернення: 07.04.25).

8. Програмний застосунок eye saver: веб-сайт. URL: https://www.eye-saver.net/ (дата звернення: 21.03.25).

9. Abdallah Saad Pe. The ergonomics (human factors) and display systems Engineering theory and practice Handbook. 2021. 1166p.

10. Jeremy Likness, John Garland. Programming the Windows Runtime by Example: A Comprehensive Guide to WinRT with Examples in C# and XAML. 2014. 816p.

11. GammaRamp Structure (Microsoft.DirectX.Direct3D): веб-сайт. URl: https://learn.microsoft.com/ua-ua/previous-versions/windows/desktop/bb322490 (дата звернення: 15.04.2025)

12. О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. Комп'ютерна графіка. Вінниця : ВНТУ, 2023. 147 с.

13. C#: веб-сайт. URL: https://beetroot.academy/blog/shcho-take-c-chi-pidhodit-meni-cya-mova-programuvannya-chomu-vona-kruta (дата звернення: 20.04.25).

14. Python: веб-сайт. URL: https://www.python.org/ (дата звернення: 21.04.25).

15. Java: веб-сайт. URL: https://www.oracle.com/ua/java/technologies /downloads/ (дата звернення: 22.04.25).

16. C++: веб-сайт. URL : https://learn.microsoft.com/ukua/cpp/?view= msvc-170 (дата звернення: 23.04.25).

17. Microsoft Visual Studio: веб-сайт. URL : https://learn.microsoft.com/uk-ua/lifecycle/products/visual-studio-2022 (дата звернення: 24.04.25).

18. CLion: веб-сайт. URL: https://allsoft.ua/p779411939-jetbrains-clion-202212.html?srsltid=AfmBOop402Vuavdu23YxKOH_lG6L2DPzAqYjNOAWChM9H9jw6BZ9Oj (дата звернення: 25.04.25).

19. Code::Bloks: веб-сайт. URL: https://www.codeblocks.org/ (дата звернення: 26.04.25).

20. Eclipse: веб-сайт. URL: https://eclipseide.org/ (дата звернення: 27.04.25).

21. А. М. Пєтух, О. В. Романюк, О. Н. Романюк. Бази даних. Мова запитів, управління транзакціями, розподілена обробка даних. Вінниця, ВНТУ 2016. 97с.

22. Jaime Levy. UX Strategy: Product Strategy Techniques for Devising Innovative Digital Solutions. 2021. 302p.

23. Ben Shneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs. Designing the User Interface: Strategies for Effective Human-Computer Interaction, Global Edition. 2018. 624p.

24. Циркадні ритми людини: веб-сайт. URL: https://emm.ua/article /tsirkadni-ritmi (дата звернення: 07.05.25).

25. Matthew Heusser, Michael Larsen. Software Testing Strategies A Testing Guide for the 2020s. 2023. 378p.

26. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк. Г. О. Черноволик. - Вінниця : ВНТУ. 2022. 52с.

ДОДАТКИ

Додаток А – Технічне завдання

75

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

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"24" березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача»
за спеціальністю
121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ Чехместрук Р.Ю.
" 23 " березня 2025 р.
Виконав:
студент гр.1ПП-21Б Козаченко М.Р.
" 23 " березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача».

Галузь застосування - системи комп’ютерної графіки.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025р. ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є покращення та полегшення взаємодії користувачів із екраном за рахунок зміни режимів роботи екрану.

Призначення роботи – розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача

4. Вихідні дані для проведення НДР

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

1. Jeremy Likness, John Garland. Programming the Windows Runtime by Example: A Comprehensive Guide to WinRT with Examples in C# and XAML. 2014. 816p.

2. Jaime Levy. UX Strategy: Product Strategy Techniques for Devising Innovative Digital Solutions. 2021. 302p.

3. О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. Комп’ютерна графіка. Вінниця : ВНТУ, 2023. 147 с.

4. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк. Г. О. Черноволик. - Вінниця : ВНТУ. 2022. - 52 с.

5. Технічні вимоги

Режими роботи екрану, зміна параметрів екрану (яскравості, контрасту та колірних каналів RGB), створення профілів із заданими параметрами,

автоматичний адаптивний режим зміни параметрів відповідно до часу доби, автоматичний адаптивний режим зміни параметрів екрану відповідно до ввімкненої програми, система обчислення безперервного користування увімкненим режимом та сповіщенням про необхідність перерви, фонова робота програми та меню у системному треї.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні. Графічна та текстова документація повинна відповідати діючим стандартам України.

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

- пояснювальна записка до БКР;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п.	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз існуючих рішень та проблематики	25.03.2025 – 05.04.2025
2	Теоретична частина	06.04.2025 – 20.04.2025
3	Моделювання та експериментальне дослідження	21.04.2025 – 01.05.2025
4	Розробка програмного застосунку	02.05.2025 – 19.05.2025
5	Розробка інтерфейсу та тестування	20.05.2025 – 30.05.2025

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Дозволяється коригування бакалаврської кваліфікаційної роботи.

Додаток Б – Протокол перевірки БКР на плагіат

80

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка інтелектуального програмного застосунку для адаптивного керування режимами роботи екрану на основі аналізу активності користувача»

Тип роботи: БКР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Чехместрук Р. Ю.

Оригінальність	94,5 %
Схожість	5,5 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Strikeplagiarism

Автор роботи Козаченко М. Р.

Керівник роботи Чехместрук Р. Ю.

Додаток В – Лістинг програми

frmMain.h

```
#pragma once
#include "frmAbout.h"
#include "Windows.h"
#include "Timer.h"
#include "AutoScreenAdjuster.h"
#include <cmath>
#include <math.h>
#include <iostream>
#include <windows.h>
#include <vcclr.h>
#using <Microsoft.VisualBasic.dll>

namespace MonitorGear {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace Microsoft::VisualBasic;
	using namespace System::Collections::Generic;
	using namespace System::Runtime::InteropServices;
	using namespace System::Diagnostics;

	public ref class frmMain : public System::Windows::Forms::Form

	{
	public:
		frmMain(void)
		{
			InitializeComponent();

			ShowPage(panelModes);

			UpdateTrayStatus(false);
			InitializeTrayIcon();

			InitializeAutoAdjustTimer();

			this->FormClosing += gcnew FormClosingEventHandler(this,
&frmMain::frmMain_FormClosing);

			profiles = gcnew System::Collections::Generic::List<DisplayProfile^>();
			this->Load += gcnew System::EventHandler(this, &frmMain::Form_Load);

			frmabout = gcnew frmAbout();

			timer = gcnew Timer(label_timer, label_message, trayIcon);
			System::Collections::Generic::List<int>^ customNotifyTimes = gcnew
System::Collections::Generic::List<int>();
			customNotifyTimes->Add(1);
```

```
                customNotifyTimes->Add(2);
                customNotifyTimes->Add(180);
                timer->SetNotifyIntervals(customNotifyTimes);

                windowToProfileMap = gcnew
System::Collections::Generic::Dictionary<String^, String^>();
                notifiedWindows = gcnew System::Collections::Generic::List<String^>();
                LoadWindowProfilesFromFile();

                UpdateProfileListDisplay();

                trayIcon->BalloonTipClicked += gcnew System::EventHandler(this,
&frmMain::TrayIcon_BalloonTipClicked);

                label_v1_1->Visible = true;
                label_v1_2->Visible = false;
                label_v1_3->Visible = false;
                label_v1_4->Visible = false;
                label_v1_5->Visible = false;
                label_v1_6->Visible = false;
                label_v1_7->Visible = false;

                label_v3_1->Visible = false;
                label_v3_2->Visible = true;

                label_v4_1->Visible = false;
                label_v4_2->Visible = true;

        }

    protected:
        ~frmMain()
        {
            if (components)
            {
                delete components;
            }
        }

        Timer^ timer;

        ToolStripMenuItem^ trayStatusMenuItem;
        ToolStripMenuItem^ trayWindowProfileStatusMenuItem;

        System::Windows::Forms::Timer^ autoAdjustTimer;
        bool autoAdjustEnabled = false;

        System::Windows::Forms::Timer^ windowProfileTimer;
    bool windowProfileEnabled = false;

    System::Collections::Generic::Dictionary<String^, String^>^ windowToProfileMap;
        System::Collections::Generic::List<String^>^ notifiedWindows;

private: System::Windows::Forms::TrackBar^ trackBar_contrast;
```

```
private: System::Windows::Forms::TrackBar^ trackBar_brightness;
private: System::Windows::Forms::TrackBar^ trackBar_blue;
private: System::Windows::Forms::TrackBar^ trackBar_green;
private: System::Windows::Forms::TrackBar^ trackBar_red;
private: System::Windows::Forms::NotifyIcon^ trayIcon;
private: System::Windows::Forms::ContextMenuStrip^ trayMenu;
private: System::Windows::Forms::ListBox^ listBoxProfiles;
private: System::Windows::Forms::ListBox^ listBoxOpenWindows;
private: System::Windows::Forms::Panel^ panelMenu;
private: System::Windows::Forms::PictureBox^ pictureBoxModes;
private: System::Windows::Forms::PictureBox^ pictureBoxfrmAbout;
private: System::Windows::Forms::PictureBox^ pictureBoxAutoMode;
private: System::Windows::Forms::PictureBox^ pictureBoxAlwaysOn;
private: System::Windows::Forms::PictureBox^ pictureBoxProfiles;
private: System::Windows::Forms::Panel^ panelContent;
private: System::ComponentModel::BackgroundWorker^ backgroundWorker1;
private: System::Windows::Forms::Panel^ panel24_7;
private: System::Windows::Forms::Panel^ panelProfiles;
private: System::Windows::Forms::Panel^ panelAutoMode;
private: System::Windows::Forms::PictureBox^ pictureBox3;
private: System::Windows::Forms::PictureBox^ pictureBox2;
private: System::Windows::Forms::PictureBox^ pictureBox1;
private: System::Windows::Forms::PictureBox^ pictureBox5;
private: System::Windows::Forms::PictureBox^ pictureBox4;
private: System::Windows::Forms::Panel^ panelModes;
private: System::Windows::Forms::Label^ label_timer;
private: System::Windows::Forms::Label^ label_message;
private: System::Windows::Forms::PictureBox^ pictureBox9;
private: System::Windows::Forms::PictureBox^ pictureBox8;
private: System::Windows::Forms::PictureBox^ pictureBox7;
private: System::Windows::Forms::PictureBox^ pictureBox6;
private: System::Windows::Forms::PictureBox^ pictureBox12;
private: System::Windows::Forms::PictureBox^ pictureBox11;
private: System::Windows::Forms::PictureBox^ pictureBox10;
private: System::Windows::Forms::PictureBox^ pictureBox13;
private: System::Windows::Forms::PictureBox^ pictureBox14;
private: System::Windows::Forms::PictureBox^ pictureBox17;
private: System::Windows::Forms::PictureBox^ pictureBox16;
private: System::Windows::Forms::PictureBox^ pictureBox15;
private: System::Windows::Forms::PictureBox^ pictureBox18;
private: System::Windows::Forms::ListBox^ listBoxSaveProfile;
private: System::Windows::Forms::Label^ label1;
private: System::Windows::Forms::PictureBox^ pictureBox21;
private: System::Windows::Forms::PictureBox^ pictureBox20;
private: System::Windows::Forms::PictureBox^ pictureBox19;
private: System::Windows::Forms::Label^ label_v3_2;
private: System::Windows::Forms::Label^ label_v3_1;
private: System::Windows::Forms::ListBox^ listBoxProfiles3;
private: System::Windows::Forms::PictureBox^ pictureBox23;
private: System::Windows::Forms::PictureBox^ pictureBox22;
private: System::Windows::Forms::Label^ label_v4_2;
private: System::Windows::Forms::Label^ label_v4_1;
private: System::Windows::Forms::PictureBox^ pictureBox24;
private: System::Windows::Forms::PictureBox^ pictureBox26;
```

```
private: System::Windows::Forms::PictureBox^ pictureBox25;
private: System::Windows::Forms::Label^ label_v1_4;
private: System::Windows::Forms::Label^ label_v1_3;
private: System::Windows::Forms::Label^ label_v1_2;
private: System::Windows::Forms::Label^ label_v1_7;
private: System::Windows::Forms::Label^ label_v1_6;
private: System::Windows::Forms::Label^ label_v1_1;
private: System::Windows::Forms::Label^ label_v1_5;

        protected:


        private:
                System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

                void InitializeComponent(void)
                {
                        System::ComponentModel::ComponentResourceManager^ resources =
(gcnew System::ComponentModel::ComponentResourceManager(frmMain::typeid));
        …
                }
        void ShowPage(Panel^ target) {
                for each (Control ^ ctrl in panelContent->Controls) {
                        Panel^ p = dynamic_cast<Panel^>(ctrl);
                        if (p != nullptr)
                                p->Visible = false;
                }
                target->Visible = true;
                target->BringToFront();
        }

        private: System::Void btnModes_Click(System::Object^ sender, System::EventArgs^ e) {
                ShowPage(panelModes);
        }

private: System::Void btnProfiles_Click(System::Object^ sender, System::EventArgs^ e) {
           ShowPage(panelProfiles);
}

private: System::Void btn24_7_Click(System::Object^ sender, System::EventArgs^ e) {
           ShowPage(panel24_7);
}

private: System::Void btnAutoMode_Click(System::Object^ sender, System::EventArgs^ e) {
           ShowPage(panelAutoMode);
}

          void InitializeTrayIcon() {
                trayMenu = gcnew System::Windows::Forms::ContextMenuStrip();

                trayMenu->Items->Add("Відкрити", nullptr, gcnew System::EventHandler(this,
&frmMain::TrayOpen_Click));
```

```
            trayMenu->Items->Add(gcnew ToolStripSeparator());

            trayStatusMenuItem = gcnew ToolStripMenuItem("24/7: вимкнено");
            trayStatusMenuItem->Enabled = false;
            trayMenu->Items->Add(trayStatusMenuItem);

            trayMenu->Items->Add(gcnew ToolStripSeparator());

            trayWindowProfileStatusMenuItem = gcnew ToolStripMenuItem("АвтоВікна:
вимкнено");
            trayWindowProfileStatusMenuItem->Enabled = false;
            trayMenu->Items->Add(trayWindowProfileStatusMenuItem);

            trayMenu->Items->Add(gcnew ToolStripSeparator());

            trayMenu->Items->Add("Вийти", nullptr, gcnew System::EventHandler(this,
&frmMain::TrayExit_Click));

            trayIcon = gcnew System::Windows::Forms::NotifyIcon();
            trayIcon->Icon = this->Icon;
            trayIcon->ContextMenuStrip = trayMenu;
            trayIcon->Text = "MonitorGear";
            trayIcon->Visible = true;

            trayIcon->DoubleClick += gcnew System::EventHandler(this,
&frmMain::TrayOpen_Click);
        }

void UpdateTrayStatus(bool isAutoModeActive) {
    if (trayStatusMenuItem != nullptr) {
        trayStatusMenuItem->Text = isAutoModeActive ? "24/7: увімкнено" : "24/7:
вимкнено";
    }
}

void UpdateWindowProfileTrayStatus(bool isEnabled) {
    if (trayWindowProfileStatusMenuItem != nullptr) {
        trayWindowProfileStatusMenuItem->Text = isEnabled ? "АвтоВікна: увімкнено" :
"АвтоВікна: вимкнено";
    }
}

void UpdateTrayProfilesInTrayMenu() {
    int indexStart = 6;

    while (trayMenu->Items->Count > indexStart + 1) {
        trayMenu->Items->RemoveAt(indexStart);
    }

    if (profiles->Count > 0) {
        for each (DisplayProfile ^ profile in profiles) {
            ToolStripMenuItem^ item = gcnew ToolStripMenuItem(profile->Name);
            item->Click += gcnew System::EventHandler(this,
&frmMain::TrayProfile_Click);
```

```
                trayMenu->Items->Insert(trayMenu->Items->Count - 1, item);
            }

            trayMenu->Items->Insert(trayMenu->Items->Count - 1, gcnew
ToolStripSeparator());
        }
}

private: System::Void TrayProfile_Click(System::Object^ sender, System::EventArgs^ e) {
        ToolStripMenuItem^ item = dynamic_cast<ToolStripMenuItem^>(sender);
        if (item == nullptr) return;

        System::String^ selectedName = item->Text;

        for each (DisplayProfile ^ p in profiles) {
            if (p->Name == selectedName) {
                current_brightness = p->Brightness;
                current_contrast = p->Contrast;
                current_red = p->Red;
                current_green = p->Green;
                current_blue = p->Blue;
                UpdateGammaRamp();
                break;
            }
        }
}

private: System::Void frmMain_FormClosing(System::Object^ sender,
System::Windows::Forms::FormClosingEventArgs^ e) {
        if (e->CloseReason == CloseReason::UserClosing) {
            e->Cancel = true;
            this->Hide();
            trayIcon->ShowBalloonTip(1000, "MonitorGear", "Програма згорнута в трей.",
ToolTipIcon::Info);
        }
}

private: System::Void TrayOpen_Click(System::Object^ sender, System::EventArgs^ e) {
        this->Show();
        this->WindowState = FormWindowState::Normal;
        this->Activate();
}

private: System::Void TrayExit_Click(System::Object^ sender, System::EventArgs^ e) {
   trayIcon->Visible = false;
   Application::Exit();
}

System::Void frmMain::TrayIcon_BalloonTipClicked(System::Object^ sender,
System::EventArgs^ e) {
        this->Show();
        this->WindowState = FormWindowState::Normal;
        this->BringToFront();
        this->Activate();
```

```
}

ref class DisplayProfile
{
public:
	System::String^ Name;
	int Brightness;
	int Contrast;
	int Red;
	int Green;
	int Blue;

	DisplayProfile(System::String^ name, int brightness, int contrast, int red, int green, int
blue)
	{
		Name = name;
		Brightness = brightness;
		Contrast = contrast;
		Red = red;
		Green = green;
		Blue = blue;
	}
};

System::Collections::Generic::List<DisplayProfile^>^ profiles = gcnew
System::Collections::Generic::List<DisplayProfile^>();

private: System::Void btnSaveProfile_Click(System::Object^ sender, System::EventArgs^ e) {
	if (profiles->Count >= 5) {
		MessageBox::Show("Максимальна кількість профілів — 5.", "Обмеження",
MessageBoxButtons::OK, MessageBoxIcon::Warning);
		return;
	}

	System::String^ name = Microsoft::VisualBasic::Interaction::InputBox(
		"Введіть назву профілю:",
		"Збереження профілю",
		"Новий профіль",
		-1, -1
	);

	if (String::IsNullOrWhiteSpace(name)) {
		MessageBox::Show("Назва профілю не може бути порожньою!", "Помилка",
MessageBoxButtons::OK, MessageBoxIcon::Error);
		return;
	}

	DisplayProfile^ profile = gcnew DisplayProfile(
		name,
		trackBar_brightness->Value,
		trackBar_contrast->Value,
		trackBar_red->Value,
		trackBar_green->Value,
		trackBar_blue->Value
```

```
        );

        profiles->Add(profile);
        SaveProfilesToFile();
        listBoxSaveProfile->Items->Add(profile->Name);
        UpdateTrayProfilesInTrayMenu();
        MessageBox::Show("Профіль збережено!", "Успішно", MessageBoxButtons::OK,
MessageBoxIcon::Information);
}

private: System::Void btnApplyProfile_Click(System::Object^ sender, System::EventArgs^ e) {
        if (listBoxSaveProfile->SelectedIndex == -1) return;

        System::String^ selectedName = listBoxSaveProfile->SelectedItem->ToString();
        DisplayProfile^ selectedProfile = nullptr;

        for each (DisplayProfile ^ p in profiles) {
                if (p->Name == selectedName) {
                        selectedProfile = p;
                        break;
                }
        }

        if (selectedProfile == nullptr) return;

        trackBar_brightness->Value = selectedProfile->Brightness;
        trackBar_contrast->Value = selectedProfile->Contrast;
        trackBar_red->Value = selectedProfile->Red;
        trackBar_green->Value = selectedProfile->Green;
        trackBar_blue->Value = selectedProfile->Blue;

        current_brightness = selectedProfile->Brightness;
        current_contrast = selectedProfile->Contrast;
        current_red = selectedProfile->Red;
        current_green = selectedProfile->Green;
        current_blue = selectedProfile->Blue;

        UpdateGammaRamp();
        timer->Start();
}

private: System::Void btnDeleteProfile_Click(System::Object^ sender, System::EventArgs^ e) {
                        if (listBoxSaveProfile->SelectedIndex == -1) return;

                        System::String^ selectedName = listBoxSaveProfile->SelectedItem-
>ToString();

                        for (int i = 0; i < profiles->Count; ++i) {
                                if (profiles[i]->Name == selectedName) {
                                        profiles->RemoveAt(i);
                                        SaveProfilesToFile();

                                        break;
                                }
```

```
                }

                listBoxSaveProfile->Items->Remove(selectedName);
                UpdateTrayProfilesInTrayMenu();
                MessageBox::Show("Профіль видалено.", "Інформація",
MessageBoxButtons::OK, MessageBoxIcon::Information);
}

void SaveProfilesToFile() {
                try {
                        System::IO::StreamWriter^ writer = gcnew
System::IO::StreamWriter("profiles.dat");
                        for each (DisplayProfile ^ profile in profiles) {
                                writer-
>WriteLine(String::Format("{0}|{1}|{2}|{3}|{4}|{5}",
                                        profile->Name,
                                        profile->Brightness,
                                        profile->Contrast,
                                        profile->Red,
                                        profile->Green,
                                        profile->Blue));
                        }
                        writer->Close();
                }
                catch (Exception^ ex) {
                        MessageBox::Show("Помилка при збереженні профілів: " +
ex->Message);
                }
}

void LoadProfilesFromFile() {
                if (!System::IO::File::Exists("profiles.dat")) return;

                array<String^>^ lines =
System::IO::File::ReadAllLines("profiles.dat");

                for each (String ^ line in lines) {
                        array<String^>^ parts = line->Split('|');
                        if (parts->Length == 6) {
                                DisplayProfile^ profile = gcnew DisplayProfile(
                                        parts[0],
                                        Int32::Parse(parts[1]),
                                        Int32::Parse(parts[2]),
                                        Int32::Parse(parts[3]),
                                        Int32::Parse(parts[4]),
                                        Int32::Parse(parts[5])
                                );
                                profiles->Add(profile);
                                listBoxSaveProfile->Items->Add(profile->Name);
                                UpdateTrayProfilesInTrayMenu();
                        }
                }
}
```

```
private: System::Void Form_Load(System::Object^ sender, System::EventArgs^ e) {
                                LoadProfilesFromFile();
                                UpdateTrayProfilesInTrayMenu();

}

void InitializeAutoAdjustTimer() {
        autoAdjustTimer = gcnew System::Windows::Forms::Timer();
        autoAdjustTimer->Interval = 60000; // 60 секунд
        autoAdjustTimer->Tick += gcnew System::EventHandler(this,
&frmMain::AutoAdjustTick);
}

void AutoAdjustTick(System::Object^ sender, System::EventArgs^ e) {
        if (!autoAdjustEnabled) return;

        int hour = DateTime::Now.Hour;
        ApplySettingsForTime(hour);
}

void ApplySettingsForTime(int hour) {
        if (hour >= 22 || hour < 6) {
                                current_brightness = 50;
                                current_contrast = 50;
                                current_blue = 50;
        }
        else if (hour >= 6 && hour < 8) {
                                current_brightness = 75;
                                current_contrast = 75;
                                current_blue = 75;
        }
        else if (hour >= 8 && hour < 10) {
                                current_brightness = 100;
                                current_contrast = 100;
        }
        else if (hour >= 12 && hour < 16) {
                                current_brightness = 125;
                                current_contrast = 125;
        }
        else if (hour >= 16 && hour < 18) {
                                current_brightness = 100;
                                current_contrast = 100;
        }
        else if (hour >= 18 && hour < 20) {
                                current_brightness = 90;
                                current_contrast = 90;
        }
        else if (hour >= 20 && hour < 22) {
                                current_brightness = 70;
                                current_contrast = 70;
                                current_blue = 90;
        }

        UpdateGammaRamp();
```

```
}

private: System::Void btnEnableAutoAdjust_Click(System::Object^ sender, System::EventArgs^ e)
{
	autoAdjustEnabled = true;
	autoAdjustTimer->Start();
	UpdateTrayStatus(true);
	ApplySettingsForTime(DateTime::Now.Hour);

	trayIcon->ShowBalloonTip(1000, "MonitorGear", "Режим 24/7 увімкнено",
ToolTipIcon::Info);

	label_v3_1->Visible = true;
	label_v3_2->Visible = false;
}


private: System::Void btnDisableAutoAdjust_Click(System::Object^ sender, System::EventArgs^
e) {
	if (autoAdjustTimer != nullptr) {
		autoAdjustTimer->Stop();
	}
	autoAdjustEnabled = false;
	UpdateTrayStatus(false);

	current_brightness = 100;
	current_contrast = 100;
	current_saturation = 100;
	current_red = 100;
	current_green = 100;
	current_blue = 100;

	UpdateGammaRamp();

	trayIcon->ShowBalloonTip(1000, "MonitorGear", "Режим 24/7 вимкнено",
ToolTipIcon::Info);

	label_v3_1->Visible = false;
	label_v3_2->Visible = true;
}

		void LoadWindowProfilesFromFile() {
			windowToProfileMap = gcnew
System::Collections::Generic::Dictionary<String^, String^>();

			String^ filePath =
System::IO::Path::Combine(Application::StartupPath, "window_profiles.dat");
			if (!System::IO::File::Exists(filePath)) return;

			array<String^>^ lines =
System::IO::File::ReadAllLines(filePath);

			for each (String ^ line in lines) {
				array<String^>^ parts = line->Split('|');
```

```
                                        if (parts->Length == 2) {
                                            windowToProfileMap[parts[0]]
= parts[1];
                                        }
                                    }
                                }

                                void SaveWindowProfilesToFile() {
                                    String^ filePath =
System::IO::Path::Combine(Application::StartupPath, "window_profiles.dat");
                                    System::IO::StreamWriter^ writer = gcnew
System::IO::StreamWriter(filePath);

                                    for each (auto pair in windowToProfileMap) {
                                        writer->WriteLine(pair.Key + "|" +
pair.Value);
                                    }

                                    writer->Close();
                                }

                                void ApplyProfileByName(String^ profileName) {
                                    for each (DisplayProfile ^ p in profiles) {
                                        if (p->Name == profileName) {
                                            current_brightness = p-
>Brightness;
                                            current_contrast = p->Contrast;
                                            current_red = p->Red;
                                            current_green = p->Green;
                                            current_blue = p->Blue;
                                            UpdateGammaRamp();
                                            break;
                                        }
                                    }
                                }

private: System::Void AssignProfileToWindow(String^ windowName, String^ profileName) {
                                windowToProfileMap[windowName] = profileName;
                                SaveWindowProfilesToFile();
                                UpdateProfileListDisplay();

}

private: System::Void OnWindowCheck(System::Object^ sender, System::EventArgs^ e) {
                                if (!windowProfileEnabled) return;

                                HWND hWnd = GetForegroundWindow();
                                if (!hWnd) return;

                                wchar_t windowTitle[256];
                                GetWindowText(hWnd, windowTitle, sizeof(windowTitle) /
sizeof(wchar_t));
                                String^ activeTitle = gcnew String(windowTitle);
```

```
                for each (auto pair in windowToProfileMap) {
                    if (activeTitle->Contains(pair.Key)) {
                        ApplyProfileByName(pair.Value);
                        return;
                    }
                }

                for each (String ^ notified in notifiedWindows) {
                    if (activeTitle->Contains(notified)) return;
                }

                trayIcon->ShowBalloonTip(3000, "MonitorGear",
                    "Виявлено нову програму: \"" + activeTitle +
"\"\nРекомендуємо призначити їй профіль.",
                    ToolTipIcon::Info);

                notifiedWindows->Add(activeTitle);
}

private: System::Void btnEnableWindowProfile_Click(System::Object^ sender,
System::EventArgs^ e) {
                if (windowProfileTimer == nullptr) {
                    windowProfileTimer = gcnew
System::Windows::Forms::Timer();
                    windowProfileTimer->Interval = 2000;
                    windowProfileTimer->Tick += gcnew
System::EventHandler(this, &frmMain::OnWindowCheck);
                }
                windowProfileEnabled = true;
                windowProfileTimer->Start();
                trayIcon->ShowBalloonTip(1000, "MonitorGear",
"Автозастосування профілів за вікном увімкнено", ToolTipIcon::Info);

                UpdateWindowProfileTrayStatus(true);


                label_v4_1->Visible = true;
                label_v4_2->Visible = false;

}

private: System::Void btnDisableWindowProfile_Click(System::Object^ sender,
System::EventArgs^ e) {
                if (windowProfileTimer != nullptr) windowProfileTimer->Stop();
                windowProfileEnabled = false;
                trayIcon->ShowBalloonTip(1000, "MonitorGear",
"Автозастосування профілів за вікном вимкнено", ToolTipIcon::Info);

                UpdateWindowProfileTrayStatus(false);

                label_v4_1->Visible = false;
                label_v4_2->Visible = true;
}
```

```
private: System::Void btnDeleteProfileAssignment_Click(System::Object^ sender,
System::EventArgs^ e) {
                        if (listBoxProfiles->SelectedItem == nullptr) return;

                        String^ selectedEntry = listBoxProfiles->SelectedItem->ToString();
                        int separatorIndex = selectedEntry->IndexOf(" → ");
                        if (separatorIndex == -1) return;

                        String^ windowTitle = selectedEntry->Substring(0, separatorIndex);
                        if (windowToProfileMap->ContainsKey(windowTitle)) {
                                windowToProfileMap->Remove(windowTitle);
                                SaveWindowProfilesToFile();
                                UpdateProfileListDisplay();
                        }
}

private: void UpdateProfileListDisplay() {
                        listBoxProfiles->Items->Clear();
                        for each (auto pair in windowToProfileMap) {
                                listBoxProfiles->Items->Add(pair.Key + " → " + pair.Value);
                        }
}

                        List<String^>^ GetVisibleWindowTitlesViaProcesses() {
                                List<String^>^ windowTitles = gcnew List<String^>();
                                array<Process^>^ processes = Process::GetProcesses();

                                for each (Process ^ proc in processes) {
                                        try {
                                                if (!String::IsNullOrEmpty(proc-
>MainWindowTitle)) {
                                                        windowTitles->Add(proc-
>MainWindowTitle);
                                                }
                                        }
                                        catch (...) {

                                        }
                                }
                                return windowTitles;
                        }

                        void RefreshWindowListAndProfiles() {
                                listBoxOpenWindows->Items->Clear();
                                listBoxProfiles3->Items->Clear();

                                auto titles = GetVisibleWindowTitlesViaProcesses();
                                for each (String ^ title in titles) {
                                        listBoxOpenWindows->Items->Add(title);
                                }

                                for each (DisplayProfile ^ profile in profiles) {
                                        listBoxProfiles3->Items->Add(profile->Name);
                                }
```

```
                }

private: System::Void btnRefreshWindows_Click(System::Object^ sender, System::EventArgs^ e)
{
                RefreshWindowListAndProfiles();
}

private: System::Void btnAssignProfileToWindow_Click(System::Object^ sender,
System::EventArgs^ e) {
                if (listBoxOpenWindows->SelectedItem == nullptr || listBoxProfiles3-
>SelectedItem == nullptr) {
                        MessageBox::Show("Оберіть вікно і профіль.");
                        return;
                }

                String^ windowTitle = listBoxOpenWindows->SelectedItem-
>ToString();
                String^ profileName = listBoxProfiles3->SelectedItem->ToString();

                windowToProfileMap[windowTitle] = profileName;
                SaveWindowProfilesToFile();
                UpdateProfileListDisplay();
}

int current_brightness = 100;
int current_contrast = 100;
int current_red = 100;
int current_green = 100;
int current_blue = 100;
int current_saturation = 100;

void UpdateGammaRamp() {
        HDC hDC = GetDC(NULL);
        WORD gammaRamp[3][256];

        float b = current_brightness / 100.0f;
        float c = current_contrast / 100.0f;
        float s = current_saturation / 100.0f;

        float rFactor = current_red / 100.0f;
        float gFactor = current_green / 100.0f;
        float bFactor = current_blue / 100.0f;

        for (int i = 0; i < 256; ++i) {
                float value = i / 255.0f;

                value = ((value - 0.5f) * c + 0.5f) * b;
                if (value < 0.0f) value = 0.0f;
                if (value > 1.0f) value = 1.0f;

                WORD baseValue = static_cast<WORD>(value * 65535.0f);

                WORD gray = baseValue;
```

```
                gammaRamp[0][i] = static_cast<WORD>(min(65535, gray + s * (baseValue *
rFactor - gray)));
                gammaRamp[1][i] = static_cast<WORD>(min(65535, gray + s * (baseValue *
gFactor - gray)));
                gammaRamp[2][i] = static_cast<WORD>(min(65535, gray + s * (baseValue *
bFactor - gray)));
        }

        SetDeviceGammaRamp(hDC, gammaRamp);
        ReleaseDC(NULL, hDC);
}


void UpdateTrackBarsFromState() {
        trackBar_brightness->Value = current_brightness;
        trackBar_contrast->Value = current_contrast;
        trackBar_red->Value = current_red;
        trackBar_green->Value = current_green;
        trackBar_blue->Value = current_blue;
}

void SetDisplayProfile(int brightness, int contrast, int red, int green, int blue) {
        current_brightness = brightness;
        current_contrast = contrast;
        current_red = red;
        current_green = green;
        current_blue = blue;

        UpdateGammaRamp();
        UpdateTrackBarsFromState();
}

private: System::Void trackBar_brightness_Scroll(System::Object^ sender, System::EventArgs^ e)
{
   current_brightness = trackBar_brightness->Value;
   UpdateGammaRamp();
}

private: System::Void trackBar_contrast_Scroll(System::Object^ sender, System::EventArgs^ e) {
   current_contrast = trackBar_contrast->Value;
   UpdateGammaRamp();
}

private: System::Void trackBar_color_Scroll(System::Object^ sender, System::EventArgs^ e) {
   current_red = trackBar_red->Value;
   current_green = trackBar_green->Value;
   current_blue = trackBar_blue->Value;
   UpdateGammaRamp();
}

void SetMonitorGammaRamp(HDC hDC, int redPercent, int greenPercent, int bluePercent) {
  WORD gammaRamp[3][256];
  for (int i = 0; i < 256; ++i) {
   gammaRamp[0][i] = static_cast<WORD>(min(65535, i * 256 * redPercent / 100.0));
```

```
   gammaRamp[1][i] = static_cast<WORD>(min(65535, i * 256 * greenPercent / 100.0));
   gammaRamp[2][i] = static_cast<WORD>(min(65535, i * 256 * bluePercent / 100.0));
  }
  SetDeviceGammaRamp(hDC, gammaRamp);
 }

void ApplyGammaToAllMonitors(int redPercent, int greenPercent, int bluePercent) {
  DISPLAY_DEVICE dd;
  DEVMODE devMode;
  ZeroMemory(&dd, sizeof(dd));
  dd.cb = sizeof(dd);
  int deviceIndex = 0;

  while (EnumDisplayDevices(NULL, deviceIndex, &dd, 0)) {
   ZeroMemory(&devMode, sizeof(devMode));
   devMode.dmSize = sizeof(devMode);

   if (EnumDisplaySettings(dd.DeviceName, ENUM_CURRENT_SETTINGS, &devMode)) {
            HDC hDC = CreateDC(L"DISPLAY", dd.DeviceName, NULL, NULL);
            std::cout << "Applying gamma to monitor: " << dd.DeviceName << std::endl;
            SetMonitorGammaRamp(hDC, redPercent, greenPercent, bluePercent);
            DeleteDC(hDC);
   }
   deviceIndex++;
  }
 }

private: frmAbout^ frmabout;

private: System::Void tmsiAbout_Click(System::Object^ sender, System::EventArgs^ e) {
        frmabout->ShowDialog();
}

        // ---------- За замовчуванням ----------
                void Default() {
                        SetDisplayProfile(100, 100, 100, 100, 100);
                        timer->Stop();
                        label_message->Text = "";
                        label_timer->Text = "";

                        label_v1_1->Visible = true;
                        label_v1_2->Visible = false;
                        label_v1_3->Visible = false;
                        label_v1_4->Visible = false;
                        label_v1_5->Visible = false;
                        label_v1_6->Visible = false;
                        label_v1_7->Visible = false;
                }

private: System::Void button_default(System::Object^ sender, System::EventArgs^ e) {
        Default();
}

            // ---------- Шутер ----------
```

```
        void Shooter() {
            SetDisplayProfile(105, 105, 105, 100, 100);

            label_v1_1->Visible = false;
            label_v1_2->Visible = true;
            label_v1_3->Visible = false;
            label_v1_4->Visible = false;
            label_v1_5->Visible = false;
            label_v1_6->Visible = false;
            label_v1_7->Visible = false;
        }

private: System::Void button_shooter(System::Object^ sender, System::EventArgs^ e) {
    Shooter();
    label_message->Text = "";
    timer->Start();
}

        // ---------- Декорації ----------
        void Scenery() {
            SetDisplayProfile(90, 120, 95, 105, 100);

            label_v1_1->Visible = false;
            label_v1_2->Visible = false;
            label_v1_3->Visible = false;
            label_v1_4->Visible = false;
            label_v1_5->Visible = false;
            label_v1_6->Visible = true;
            label_v1_7->Visible = false;
        }

private: System::Void button_scenery(System::Object^ sender, System::EventArgs^ e) {
    Scenery();
    label_message->Text = "";
    timer->Start();
}

        // ---------- Книга ----------
        void Book() {
            SetDisplayProfile(90, 80, 100, 100, 80);

            label_v1_1->Visible = false;
            label_v1_2->Visible = false;
            label_v1_3->Visible = true;
            label_v1_4->Visible = false;
            label_v1_5->Visible = false;
            label_v1_6->Visible = false;
            label_v1_7->Visible = false;
        }

private: System::Void button_book(System::Object^ sender, System::EventArgs^ e) {
    Book();
    label_message->Text = "";
    timer->Start();
```

```
}

        // ---------- Кіно ----------
        void Cinema() {
                SetDisplayProfile(90, 110, 100, 100, 100);

                label_v1_1->Visible = false;
                label_v1_2->Visible = false;
                label_v1_3->Visible = false;
                label_v1_4->Visible = true;
                label_v1_5->Visible = false;
                label_v1_6->Visible = false;
                label_v1_7->Visible = false;
        }

private: System::Void button_cinema(System::Object^ sender, System::EventArgs^ e) {
        Cinema();
        label_message->Text = "";
        timer->Start();
}

        // ---------- EyeCare ----------
        void EyeCare() {
                SetDisplayProfile(90, 90, 100, 100, 60);

                label_v1_1->Visible = false;
                label_v1_2->Visible = false;
                label_v1_3->Visible = false;
                label_v1_4->Visible = false;
                label_v1_5->Visible = true;
                label_v1_6->Visible = false;
                label_v1_7->Visible = false;
        }

private: System::Void button_eye(System::Object^ sender, System::EventArgs^ e) {
        EyeCare();
        label_message->Text = "";
        timer->Start();
}

        // ---------- Ніч ----------
        void Night() {
                SetDisplayProfile(70, 70, 100, 100, 90);

                label_v1_1->Visible = false;
                label_v1_2->Visible = false;
                label_v1_3->Visible = false;
                label_v1_4->Visible = false;
                label_v1_5->Visible = false;
                label_v1_6->Visible = false;
                label_v1_7->Visible = true;
        }

private: System::Void button_night(System::Object^ sender, System::EventArgs^ e) {
```

```
            Night();
            label_message->Text = "";
            timer->Start();
}

};
}
```

Timer.h

```
#pragma once
#include <Windows.h>
#include <vcclr.h>

namespace MonitorGear {

    using namespace System;
    using namespace System::Windows::Forms;
    using namespace System::Collections::Generic;

    public ref class Timer {
    public:
        Timer(Label^ label, Label^ helloLabel, NotifyIcon^ trayIcon) {
            timerLabel = label;
            this->helloLabel = helloLabel;
            this->trayIcon = trayIcon;

            timer = gcnew System::Windows::Forms::Timer();
            timer->Interval = 1000;
            timer->Tick += gcnew EventHandler(this, &Timer::OnTimerTick);

            notifyIntervals = gcnew List<int>();
            nextNotifyIndex = 0;
        }

        void Start() {
            startTime = DateTime::Now;
            nextNotifyIndex = 0;
            timer->Start();
        }

        void Stop() {
            timer->Stop();
        }

        void SetNotifyIntervals(List<int>^ intervals) {
            notifyIntervals = intervals;
            notifyIntervals->Sort();
            nextNotifyIndex = 0;
        }

    private:
        Label^ timerLabel;
        Label^ helloLabel;
        NotifyIcon^ trayIcon;
```

```
        System::Windows::Forms::Timer^ timer;

        DateTime startTime;
        List<int>^ notifyIntervals;
        int nextNotifyIndex;

        void OnTimerTick(Object^ sender, EventArgs^ e) {
            TimeSpan elapsed = DateTime::Now - startTime;

            timerLabel->Text = String::Format("Час користування режимом:
{0:D2}:{1:D2}:{2:D2}",
                elapsed.Hours, elapsed.Minutes, elapsed.Seconds);

            // Перевірка сповіщення
            if (nextNotifyIndex < notifyIntervals->Count &&
                elapsed.TotalMinutes >= notifyIntervals[nextNotifyIndex]) {

                String^ message = String::Format("Ви користуєтесь режимом вже {0} хвилин.
Рекомендовано зробити перерву!",
                    notifyIntervals[nextNotifyIndex]);

                helloLabel->Text = message;

                if (trayIcon != nullptr) {
                    trayIcon->ShowBalloonTip(
                        5000,
                        "Рекомендація від MonitorGear",
                        message,
                        ToolTipIcon::Warning
                    );
                }

                nextNotifyIndex++;
            }
        }
    };
}
```

frmMain.h

```
#include "frmMain.h"

using namespace MonitorGear;
using namespace System;
using namespace System::Windows::Forms;

[STAThreadAttribute]
void main() {
        Application::EnableVisualStyles();
        Application::SetCompatibleTextRenderingDefault(false);
        frmMain frmmain;
        Application::Run(% frmmain);
}
```

Додаток Г – Ілюстративна частина

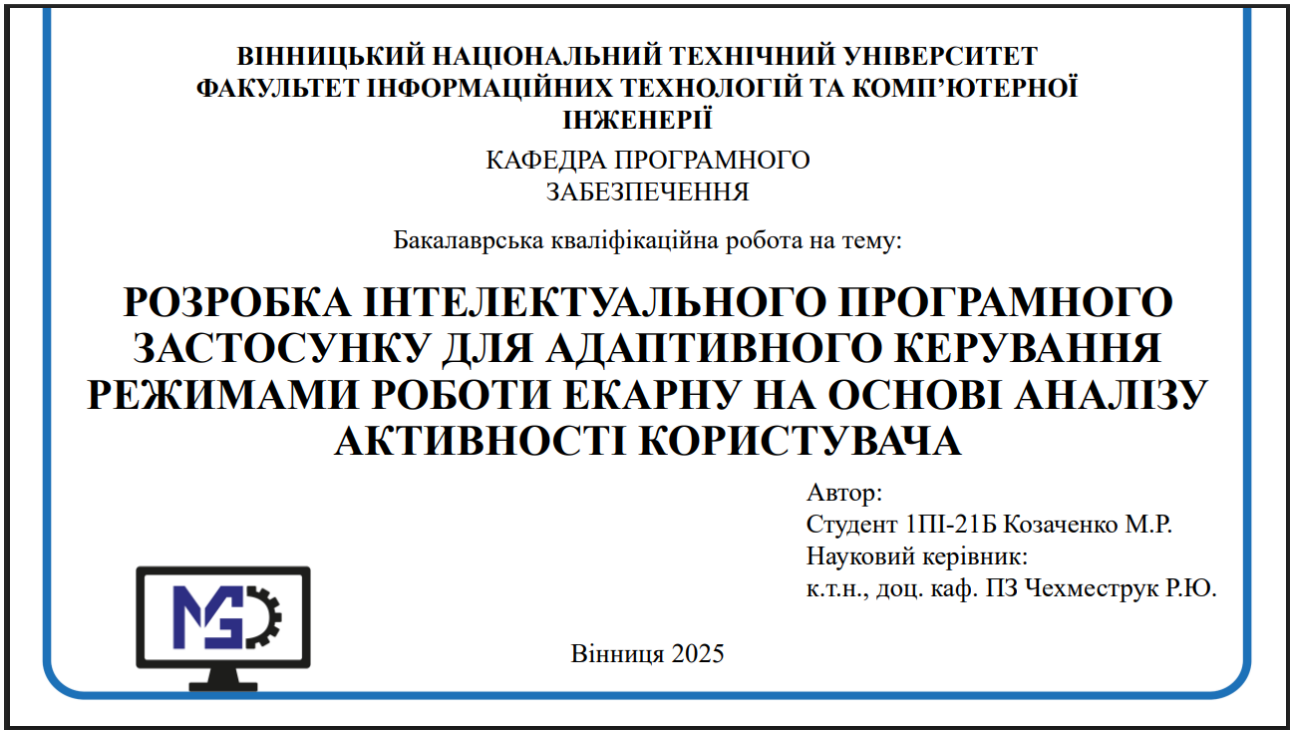

Рисунок Г.1 – Титульний слайд

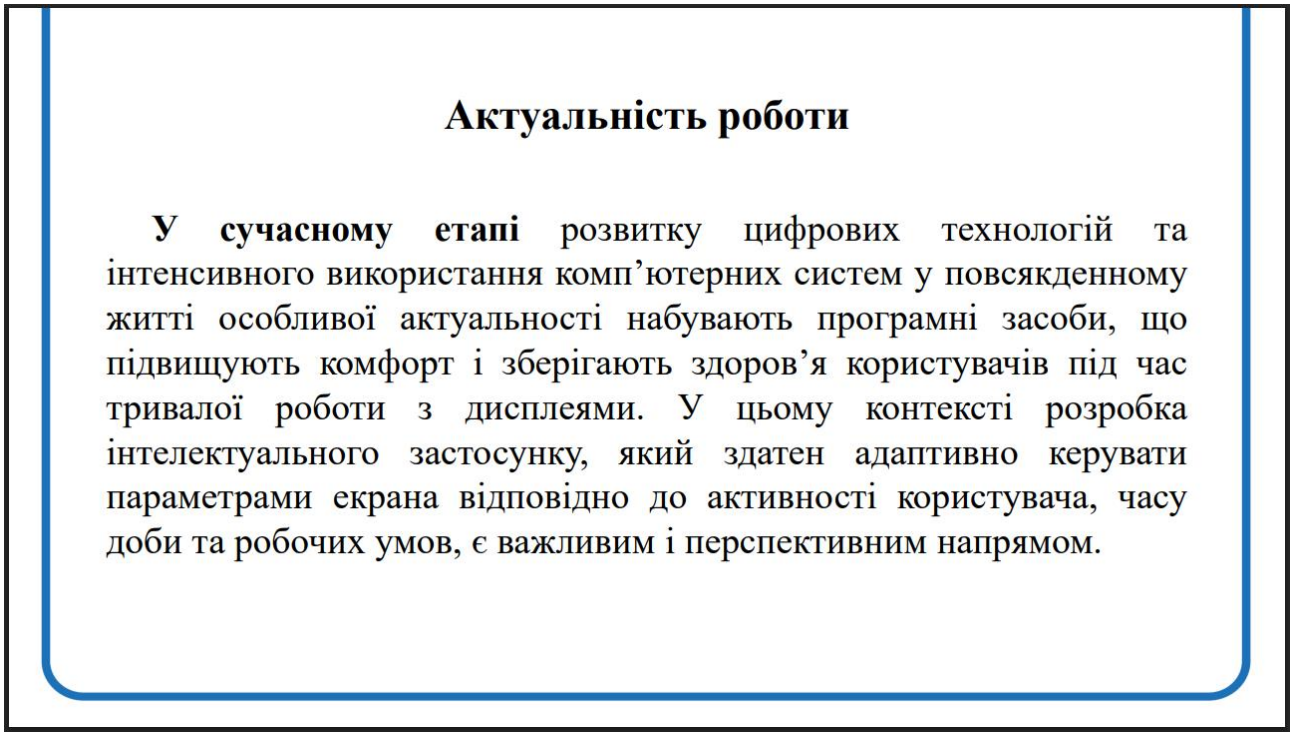

Рисунок Г.2 – Актуальність роботи

Мета, предмет та об’єкт дослідження

Мета роботи – покращення та полегшення взаємодії користувачів із екраном за рахунок зміни режимів роботи екрану.

Об’єкт дослідження – процес зміни налаштування екрану для адаптивної зміни режимів роботи екрану.

Предмет дослідження – модулі та засоби, що забезпечать зміни основних налаштувань екрану під різні індивідуальні потреби.

Рисунок Г.3 – Мета, предмет та об’єкт дослідження

Аналіз аналогів

Критерії	f.lux	Armoury Crate	Eye Saver	Власна розробка
Зміна режимів роботи екрану відповідно до потреб	-	+	-	+
Користувацька зміна параметрів (яскравість, контраст, RGB)	-	-	+	+
Регулювання температури кольору	+	-	+	+
Створення користувацьких профілів	+	-	+	+
Автоматична зміна за часом доби	+	-	-	+
Адаптивна автоматична зміна режимів до програм	-	-	-	+
Керування у фоновому режимі через меню у треї	+	+	+	+
Нагадування про необхідність перерви	-	-	-	+
Загальна оцінка	50%	25%	50%	100%

Рисунок Г.4 – Аналіз аналогів

Постановка задачі проєкту

На основі проведеного аналізу існуючих рішень, методів та інструментів адаптивного керування режимами роботи екрану, сформульовано наступні завдання для реалізації проєкту:

- гнучке налаштування основних параметрів екрану (яскравості, контрасту та колірних каналів RGB);
- створення та використання користувацьких профілів;
- адаптацію налаштувань екрану відповідно до часу доби;
- автоматичне адаптивне перемикання режимів відповідно до увімкнених програм;
- систему рекомендацій про необхідність перерви;
- робота застосунку у фоновому режимі та меню в треї.

Рисунок Г.5 – Постановка задачі проєкту

Новизна

- **Вперше запропоновано** метод інтелектуального, адаптивного автоматичного керування параметрами екрану (яскравість, контраст, кольорові канали RGB) з урахуванням часу доби, активності користувача та контексту активних програм, що дозволяє підвищити продуктивність роботи та комфорт користувача протягом дня.
- **Розроблено новий підхід** до побудови динамічних профілів екрану, який забезпечує можливість створення, збереження та автоматичного застосування до п’яти індивідуальних режимів через інтеграцію із системним треєм або на основі контекстного аналізу активної програми, що підвищує ергономіку користувацької взаємодії.

Рисунок Г.6 – Новизна

Новизна

• **Дістала подальшого розвитку** система рекомендацій щодо організації перерв у роботі з екраном, яка базується на тривалості використання поточного режиму та реалізована через систему попереджувальних повідомлень, що сприяє збереженню зорового здоров'я користувача.

Рисунок Г.7 – Новизна

Практична цінність

Полягає в здатності програмного застосунку підвищувати комфорт, продуктивність і зорову безпеку користувача під час тривалої роботи за комп'ютером. Завдяки можливості адаптивного регулювання параметрів екрана залежно від часу доби, активного вікна або профілю користувача, застосунок забезпечує оптимальні візуальні умови для роботи, навчання, перегляду контенту або відпочинку. Крім того, система рекомендацій щодо перерв сприяє профілактиці зорового перевантаження, сухості очей та цифрової втоми, що особливо актуально в умовах постійної роботи з дисплеями.

Рисунок Г.8 – Практична цінність

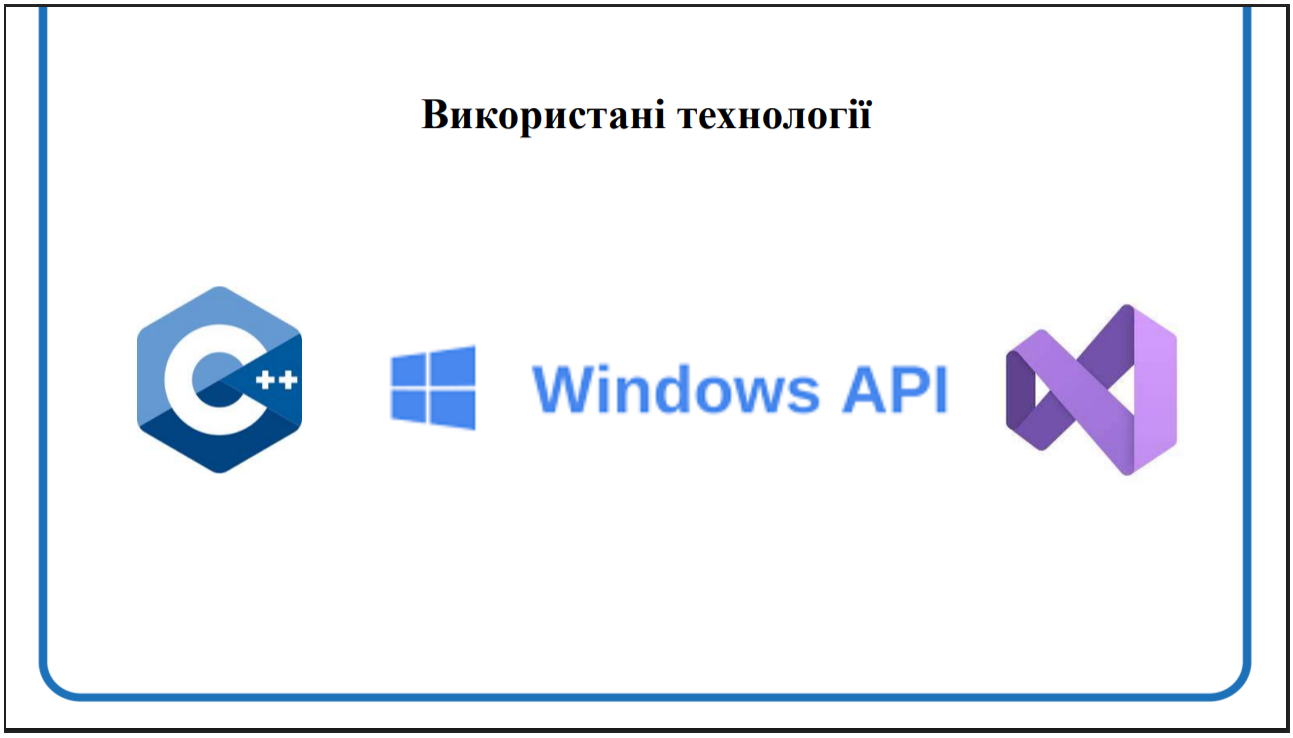

Рисунок Г.9 – Використані технології

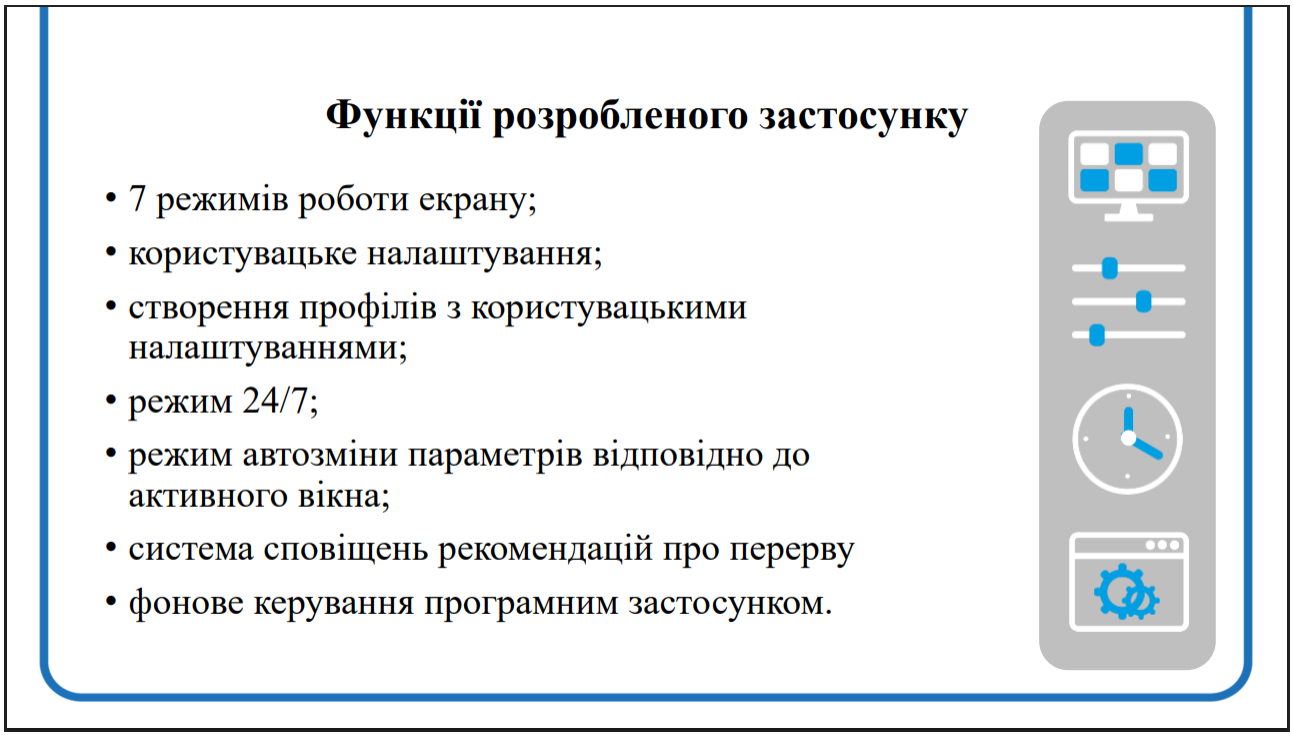

Рисунок Г.10 = Функції розробленого застосунку

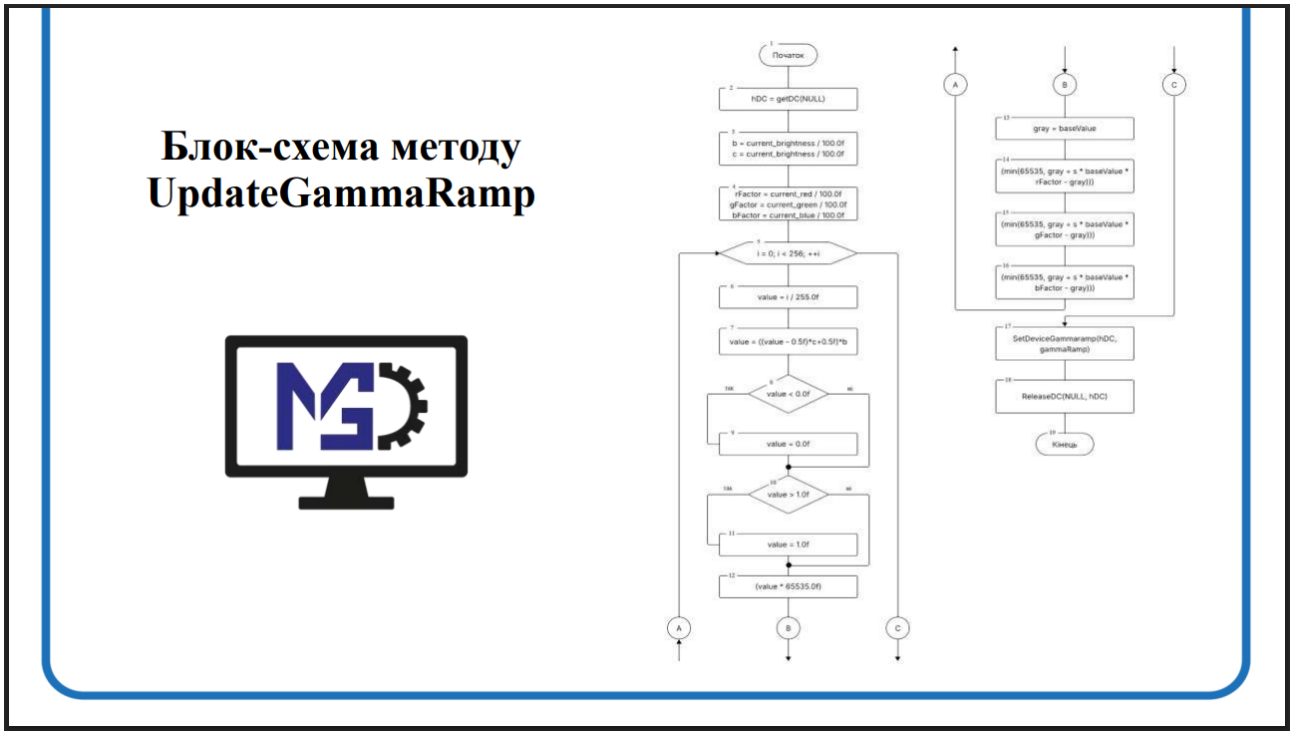

Рисунок Г.11 – Блок-схема методу UpdateGammaRamp

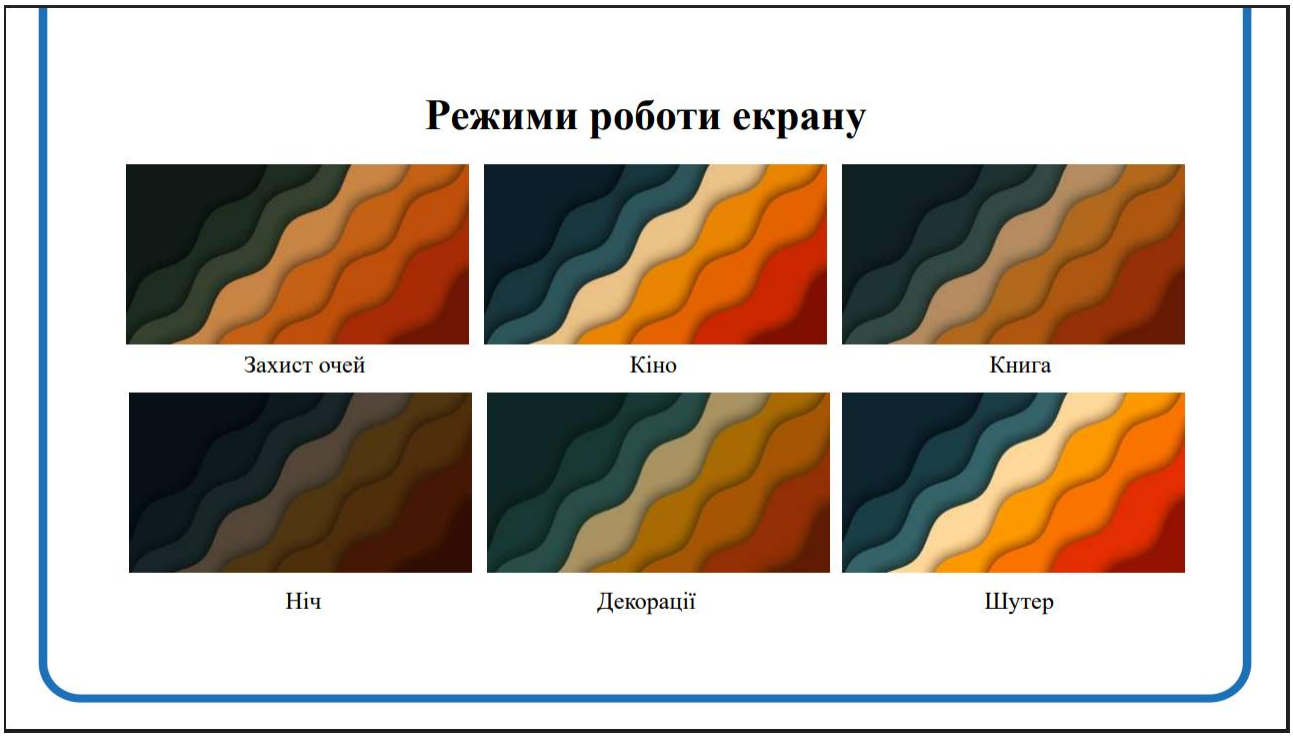

Рисунок Г.12 – Режими роботи екрану

Рисунок Г.13 – Діаграма роботи режиму 24/7

Рисунок Г.14 – Апробація результатів

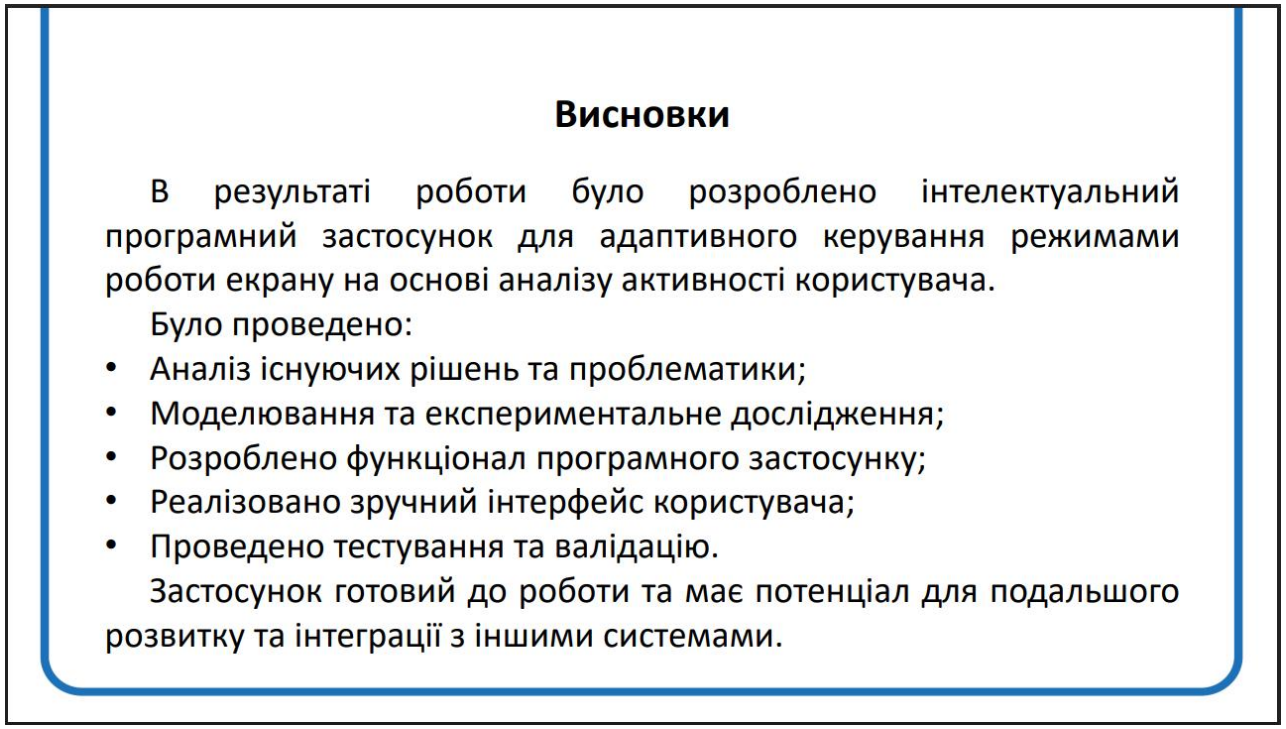

Рисунок Г.15 – Висновки