

Міністерство освіти і науки України
Національний університет "Львівська Політехніка"
Кафедра ЕОМ



Пояснювальна записка
до курсового проекту
з дисципліни "Системне програмне забезпечення"
на тему: "Утиліта діагностики монітора комп'ютера з ОС Windows"

Виконав ст. гп. КІ-34:

Романів В. А.

Перевірив:

Мархівка В. С.

Львів-2023

Анотація

Даний курсовий проєкт присвячений створенню власної утиліти для тестування моніторів комп'ютера під назвою "Monitor Diagnostics", яка є прототипом програми AIDA64, конкретно вкладки "Tools/Monitor Diagnostics". Ця утиліта розроблена для операційної системи Windows з відкритим вихідним кодом і дозволяє перевіряти правильність відображення кольорів, чіткість зображення, читабельність тексту та фокусування монітора. Крім того, вона надає можливість регулювання контрастності, яскравості та розширення монітора комп'ютера.

У першому розділі проводиться аналіз методів тестування моніторів комп'ютера, а також огляд існуючих розробок з наголосом на їхні проблеми та обмеження.

Другий розділ присвячений розробці архітектури програми. Розглядається розбиття утиліти на модулі, використання принципів Model-View Programming (MVP), об'єктно-орієнтованого підходу (ООП) та механізму сигналів/слотів. Для реалізації програми використовується мова програмування C++ з використанням фреймворку Qt.

У третьому розділі детально описано реалізацію функціональності утиліти для тестування монітора комп'ютера. Описано вибір структури даних для досягнення поставленої мети, а також загальний алгоритм роботи та функціонування кожного модуля програми. Використовується механізм сигналів і слотів для оновлення даних в реальному часі, що дозволяє об'єктам взаємодіяти між собою та передавати сигнали в окремому потоці, що уникне блокування та затримок у виконанні основних операцій.

Четвертий розділ детально розглядає графічний інтерфейс утиліти. В результаті тестування програма забезпечує стабільну роботу і правильне тестування моніторів комп'ютера. Розроблена утиліта є навчальним проєктом і може використовуватись для базового тестування дисплеїв комп'ютерів.

Зміст

Анотація.....	2
Технічне завдання.....	5
Вступ.....	9
1. Аналітичний огляд.....	10
1.1. Огляд методів тестування монітору комп'ютера.....	10
1.1.1. Вимірювання параметрів монітора.....	10
1.1.2. Аналіз драйверів та налаштувань.....	10
1.1.3. Тестування функціональності.....	11
1.2. Існуючі розробки у сфері моніторингу мережі комп'ютера та їхні проблеми.....	11
1.2.1. AIDA64.....	11
1.2.2. DisplayCAL.....	12
1.3. Функціональні можливості вкладки "Tools/Monitor Diagnostics" у програмі "AIDA64"	14
1.4. Обґрунтування необхідності вдосконалення та модернізації утиліти тестування моніторів компютера.....	15
2. Аналіз завдання та вибір підходів до проектування утиліти.....	18
2.1. Аналіз функцій та вибір структури програмного рішення.....	18
2.1.1. Уточнення функцій.....	18
2.1.2. Вибір структури утиліти на рівні модулів.....	21
2.2. Вибір архітектурного рішення.....	22
2.2.1. Архітектура: модель – подання (MVC).....	22
2.2.2. Об'єктно – орієнтований підхід.....	24
2.2.3. Механізм сигналів і слотів.....	25
2.3. Вибір засобів для реалізації.....	26
2.3.1. Вибір мови програмування.....	26
2.3.2. Вибір середовища.....	26
2.3.3. Вибір готових бібліотек.....	26
3. Програмна реалізація утиліти.....	28
3.1. Проектування та вибір структур даних.....	28
3.2. Розробка алгоритму роботи утиліти.....	29
3.3 Розробка діаграми класів та її опис.....	32
3.3.1. Опис класів та їх взаємозв'язків з іншими класами.....	32
3.4. Розробка модулів утиліти.....	34
3.4.1. Модуль тестів.....	34
3.4.1.1. Опис функцій класу CalibrationFocusTest.....	35
3.4.1.2. Опис функцій класу ColorTest.....	39
3.4.1.3. Опис функцій класу GridTest.....	40
3.4.1.4. Опис функцій класу Read Test.....	41
3.4.2. Модуль локалізації.....	42
3.4.2.1. Реалізація модуля локалізації.....	43
3.4.3. Модуль системної треї.....	44
3.4.3.1. Опис реалізації модуля системної треї.....	45
3.4.4. Модуль графічного інтерфейсу.....	46
3.4.4.1. Реалізація та опис функцій модуля графічного інтерфейсу.....	47
3.5. Створення та опис програмного проекту.....	52
4. Тестування програмного забезпечення та інструкції користувачеві.....	56
4.1. Тестування та опис інтерфейсу програми.....	56
4.2. Інструкції користувачеві.....	63

Висновок.....	67
Список використаної літератури.....	69
Додатки.....	70
Додаток А. Утиліта Monitor testing. Схема алгоритму.	
Додаток Б. Утиліта Monitor testing. Діаграма класів.	
Додаток В. Утиліта Monitor testing. Лістинг коду	

Технічне завдання

Тема: утиліта діагностики монітора комп'ютера з ОС Windows (прототип програма «AIDA64», вкладка “Tools/Monitor Diagnostics”)

Функції, що підлягають реалізації:

1. **Calibration test** – тест який дозволяє відкалибрувати кольори та розширення на моніторі комп'ютера. Даний тест включає в себе такі тести:
 - 1.1. **LCD-Calibration** – це процес налаштування параметрів дисплея для досягнення оптимальної якості зображення. Під час калібрування LCD використовуються спеціальні інструменти та програмне забезпечення для налаштування таких параметрів, як яскравість, контрастність, кольорова гама, тон, насиченість та інші. Калібрування LCD дозволяє досягнути точного та стабільного відображення зображень, що особливо важливо для професійних застосувань, таких як медична діагностика, графічний дизайн та фотографія.
 - 1.2. **Brightness/Contrast** – даний тест позволяє налаштовувати яскравість та контраст монітора
 - 1.3. **Scope** – це тест, який використовується для перевірки якості відображення монітора. Цей тест відображає на екрані різні геометричні фігури, кольори і рівні яскравості з метою перевірки точності відтворення кольорів, контрастності, роздільної здатності, кута огляду та інших параметрів. Цей тест допомагає виявляти проблеми з відображенням на моніторі, такі як затемнення, блимання, артефакти, розмивання та інші. Використання scope тесту монітора може допомогти поліпшити якість відображення, знизити втомлюваність очей та забезпечити більш комфортну роботу з комп'ютером.

- 1.4. **Gamma** – це тест, який позволяє налаштувати гаму кольорів
- 1.5. **Convergence** - це спосіб перевірити якість відображення на екрані монітора шляхом перевірки точності відображення кольорів на різних частинах екрану. В процесі цього тесту на екрані з'являється сітка, складена з різно кольорових ліній, і користувач повинен переконатися, що кольори на краях ліній збігаються з кольорами по центру. Якщо кольори на краях ліній відрізняються від кольорів по центру, це може свідчити про проблеми з конвергенцією монітора, які можуть бути вирішені за допомогою налаштувань монітора.
- 1.6. **Screen size** – це тест який перевіряє розширення монітора
- 1.7. **Focus tests** – це тести, які використовуються для визначення якості фокусування зображення на моніторі. Вони можуть включати в себе різноманітні тестові зображення, такі як текст, графіки та зображення, що допомагають визначити чіткість та рівномірність зображення на різних ділянках екрану. Ці тести допомагають виявити проблеми з фокусуванням, такі як розмиття, розмивання та нерівномірність зображення. Даний тест складається з таких тестів:
- 1.8. **White Pattern** – даний тест висвітлює білі квадратики на чорному фоні
- 1.9. **Black Pattern** – даний тест висвітлює чорні квадратики на білому фоні
- 1.10. **Sharpness** - є тестом, який дозволяє визначити рівень чіткості і деталізації зображення.
- 1.11. **Dots** - це тест, що використовується для перевірки якості відображення на моніторі. Зазвичай це маленькі точки різного кольору або форми, що відображаються на екрані монітора. Цей тест дозволяє виявити дефекти монітора, такі як dead pixel (мертві

пікселі), stuck pixel (застрялий піксель), розмитість і нечіткість зображення.

1.12. **Vertical lines** – це тест, який допомагає визначити якість відтворення вертикальних ліній на моніторі.

1.13. **Horizontal lines** - це тест, який допомагає визначити якість відтворення горизонтальних ліній на моніторі.

2. **Grid tests** – є одним із способів перевірки геометрії монітора. Під час проведення цього тесту на екрані монітора відображається сітка з горизонтальними та вертикальними лініями, які повинні бути рівномірно розташовані та відстань між ними повинна бути однаковою. Цей тест допомагає виявити будь-які відхилення у відображені геометрії монітора, такі як зміщення ліній, нахил, нерівномірність розміщення ліній тощо. Даний тест може виводити лінії на білому або чорному фоні, а самі лінії можуть бути білого, чорного, червоного, зеленого, синього, фіолетового або жовтого кольору.
3. **Color tests** - це спеціальні зображення, які використовуються для перевірки правильності відображення кольорів на моніторі. Ці тести можуть допомогти виявити проблеми з кольоровою точністю монітора, такі як відхилення від правильної кольорової гами, нахил кольорів, недостатня насиченість кольорів та інші аномалії. Даний тест включає в себе такі тести кольорами, як червоний, зелений, синій, оранжевий, білий, чорний та сірий. Також даний тест може виводити відтінки кольорів, а саме червоний градієнт, зелений градієнт, синій градієнт, оранжевий градієнт та сірий градієнт.
4. **Reading test** – це спеціальний тест, який використовується для перевірки читабельності тексту на моніторі. Ці тести перевіряють різні аспекти читабельності, такі як розмір шрифту, яскравість фону, контрастність тексту, якість рендерингу шрифту та інші. Під час даного тесту на екран виводяться одне і те саме речення різними кольорами в різній кількості.

Платформа: ОС Windows

Мова програмування: C++

Середовище розробки: фреймворк Qt

Бібліотеки:

Qt libraries – внутрішні бібліотеки фреймворку.

Загальні вимоги:

1. Передбачити використання піктограм для швидкого запуску утиліти.
2. Реалізувати portable-версію та інталлятор розробленої утиліти.
3. Реалізувати help-систему для розробленої утиліти.
4. Обов'язкова наявність графічної частини

Вступ

Програма для тестування моніторів є важливим інструментом у сучасному інформаційному середовищі, знаходить широке застосування в галузі інформаційних технологій та виробництва комп'ютерних пристрій. Вона дозволяє користувачам перевіряти працевздатність моніторів, виявляти проблеми та вдосконалювати їх ефективність з метою поліпшення якості зображення.

Тестування моніторів є ключовим завданням для забезпечення високоякісного відображення графічного контенту, виявлення дефектів та налаштування параметрів. На сьогоднішній день існує кілька програм для тестування моніторів, включаючи програму AIDA64 з вкладкою "Monitor Diagnosing", яка надає функціонал для проведення різноманітних тестів та аналізу параметрів моніторів.

Однак, більшість наявних програм мають застарілий інтерфейс та обмежені можливості для звичайних користувачів, оскільки вони є комерційними проектами з обмеженим доступом.

Головною метою даної роботи є розробка прототипу програми з меншим функціоналом, спрямованого на тестування моніторів, зокрема вкладки "Monitor Diagnosing" програми AIDA64. Також, цей проект буде доступний на моїй GitHub сторінці, що дозволить користувачам переглядати та вносити вдосконалення до програми. Такий підхід дозволить зосередитися на конкретній галузі та покращити функціонал тестування моніторів, забезпечити зрозуміліше відображення даних та поліпшити інтерфейс користувача. До того ж, розробка прототипу з меншим функціоналом допоможе набути практичних навичок у проектуванні програмних систем та розширити розуміння процесу розробки програмного забезпечення для тестування моніторів.

1. Аналітичний огляд

У цьому розділі роботи проведено аналітичний огляд питання, пов'язаного з утилітою діагностики монітора комп'ютера з операційною системою Windows. Виконано короткий опис основних аспектів та важливих питань, що стосуються діагностики моніторів, зокрема їхніх параметрів та налаштувань. Оглянуті різні аспекти, які можуть впливати на роботу монітора та які варто враховувати при розробці утиліти діагностики монітора комп'ютера.

1.1. Огляд методів тестування монітору комп'ютера

Монітор є одним з ключових пристрій комп'ютерної системи, і правильна його робота має велике значення для якісного візуального досвіду користувача. Існують різні методи та інструменти, які використовуються для діагностики монітора, які дозволяють перевірити його на різноманітні аспекти, такі як роздільна здатність, колірна точність, контрастність, яскравість та інші параметри.

1.1.1. Вимірювання параметрів монітора

Вимірювання параметрів монітора: даний метод використовує спеціальне обладнання для вимірювання різних параметрів монітора, таких як роздільна здатність, яскравість, контрастність, колірна точність тощо. Вимірювальні прилади, які використовуються, можуть бути візуальними калібраторами або спеціальними програмними засобами.

1.1.2. Аналіз драйверів та налаштувань

Аналіз драйверів та налаштувань: даний метод оцінює наявні драйвери монітора та налаштування графічної підсистеми ОС Windows. Він дозволяє виявити проблеми зі сумісністю драйверів, неправильні налаштування або відсутність необхідного програмного забезпечення для повноцінної роботи монітора.

1.1.3. Тестування функціональності

Тестування функціональності: даний метод передбачає запуск різноманітних тестів для перевірки функціональності монітора. Такі тести можуть включати перевірку реакції пікселів, перевірку наявності заливки екрану, тестування режимів показу зображень тощо.

1.2. Існуючі розробки у сфері моніторингу мережі комп'ютера та їхні проблеми

1.2.1. AIDA64

AIDA64: AIDA64 є потужною та популярною утилітою для діагностики та тестування комп'ютерних компонентів, включаючи монітори. Ця програма дозволяє користувачам отримати детальну інформацію про характеристики та працевдатність їхнього монітора.[1]

В AIDA64 є спеціальна вкладка "Monitor Diagnostics" (Діагностика монітора), яка надає широкі можливості для тестування та аналізу різних параметрів монітора. Серед основних параметрів, які можна перевірити, варто виділити:

1. Роздільна здатність: AIDA64 дозволяє визначити поточну роздільну здатність монітора, яка визначає кількість пікселів у вертикальному та горизонтальному напрямках. Користувач може перевірити, чи відповідає роздільна здатність монітора заданим специфікаціям та налаштуванням.

2. Частота оновлення: Програма дозволяє перевірити поточну частоту оновлення монітора. Цей параметр визначає, скільки разів на секунду змінюються зображення на екрані. Він важливий для плавного відображення рухливих зображень і відео.

3. Яскравість та контрастність: AIDA64 надає можливість налаштування та тестування яскравості та контрастності монітора. Користувач може перевірити, які значення цих параметрів використовуються в даний момент і налаштувати їх за необхідністю.

4. Колірна точність: Програма дозволяє провести тестування колірної точності монітора. Користувач може перевірити відтворення різних кольорів, градацію кольору та наявність колірних артефактів.

AIDA64 також надає можливість збереження результатів тестування та створення звітів, що дозволяє користувачам зберігати інформацію про характеристики та стан свого монітора для подальшого аналізу чи порівняння.

Загалом, AIDA64 є потужним інструментом для діагностики та тестування моніторів, який надає розширені можливості для аналізу параметрів монітора та перевірки його працездатності.

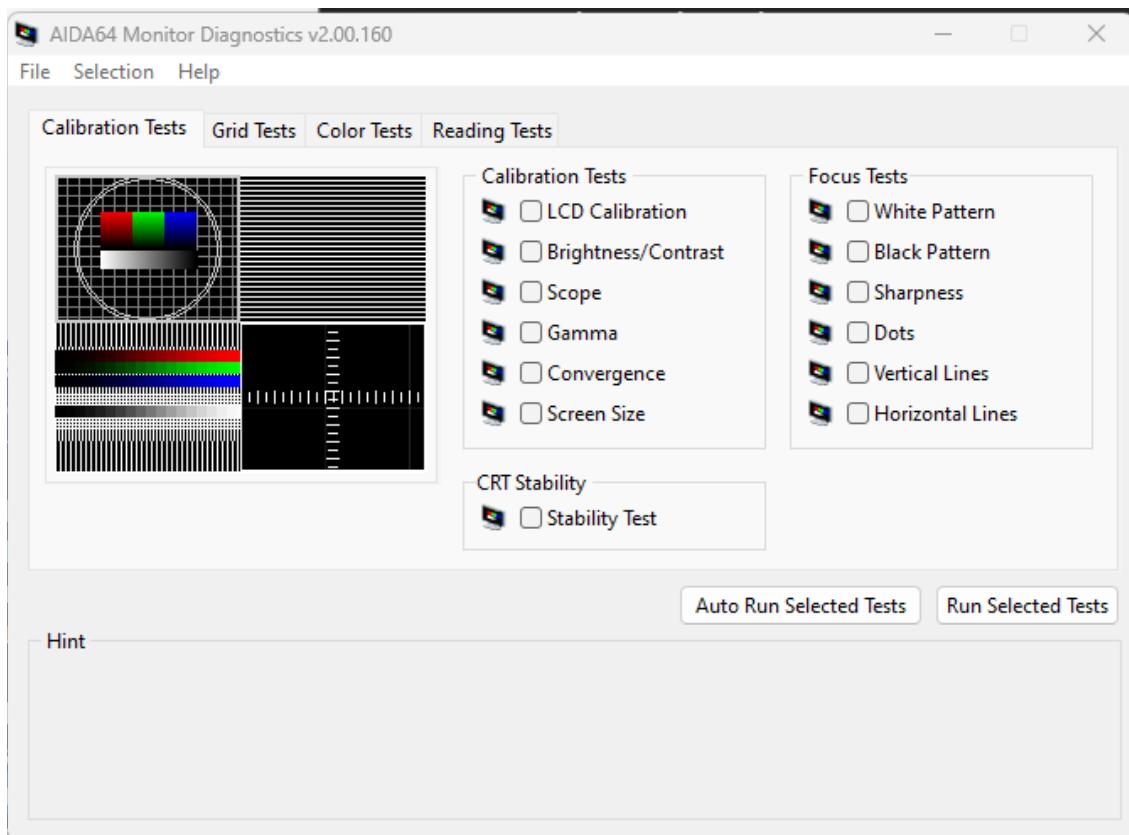


Рис. 1. Вигляд програми AIDA64

1.2.2. DisplayCAL

DisplayCAL є безкоштовною і відкритою програмою, спеціалізованою на калібруванні та тестуванні моніторів. Вона надає розширений набір інструментів для аналізу та налаштування різних параметрів монітора, що

дозволяє користувачам отримати максимально точне та відтворюване кольори візуальне відображення.[2]

Основні можливості DisplayCAL включають:

1. Калібрування монітора: DisplayCAL дозволяє користувачам калібрувати свій монітор, тобто налаштовувати його параметри таким чином, щоб досягти точного відтворення кольорів. Це важливо для фотографів, дизайнерів та інших професіоналів, які залежать від високої колірної точності.

2. Аналіз роздільної здатності: DisplayCAL дозволяє перевірити роздільну здатність монітора, тобто кількість пікселів, яку він може відображати. Користувачі можуть переконатися, що монітор працює згідно з вказаними специфікаціями та отримати точне розуміння його потенційних можливостей.

3. Колірна точність і градація кольору: DisplayCAL дозволяє проводити тестування колірної точності монітора та перевірку його градації кольору. Користувачі можуть визначити, наскільки точно монітор відтворює різні кольори і визначити потенційні відхилення.

4. Тестування функціональності: DisplayCAL також надає можливість тестувати функціональність монітора. Це включає перевірку реакції пікселів на зміни кольорів або переходи, а також перевірку наявності заливки екрану. Ці тести допомагають виявити проблеми з відображенням, такі як залишкові ефекти або мерехтіння.

DisplayCAL є потужним інструментом для вимірювання та налаштування параметрів монітора. Вона надає користувачам зручний інтерфейс, можливість збереження результатів та створення звітів, що дозволяє виконувати докладний аналіз та порівняння характеристик моніторів. Ця програма особливо корисна для професіоналів, які працюють з високоякісним візуальним контентом і потребують точного відтворення кольорів.

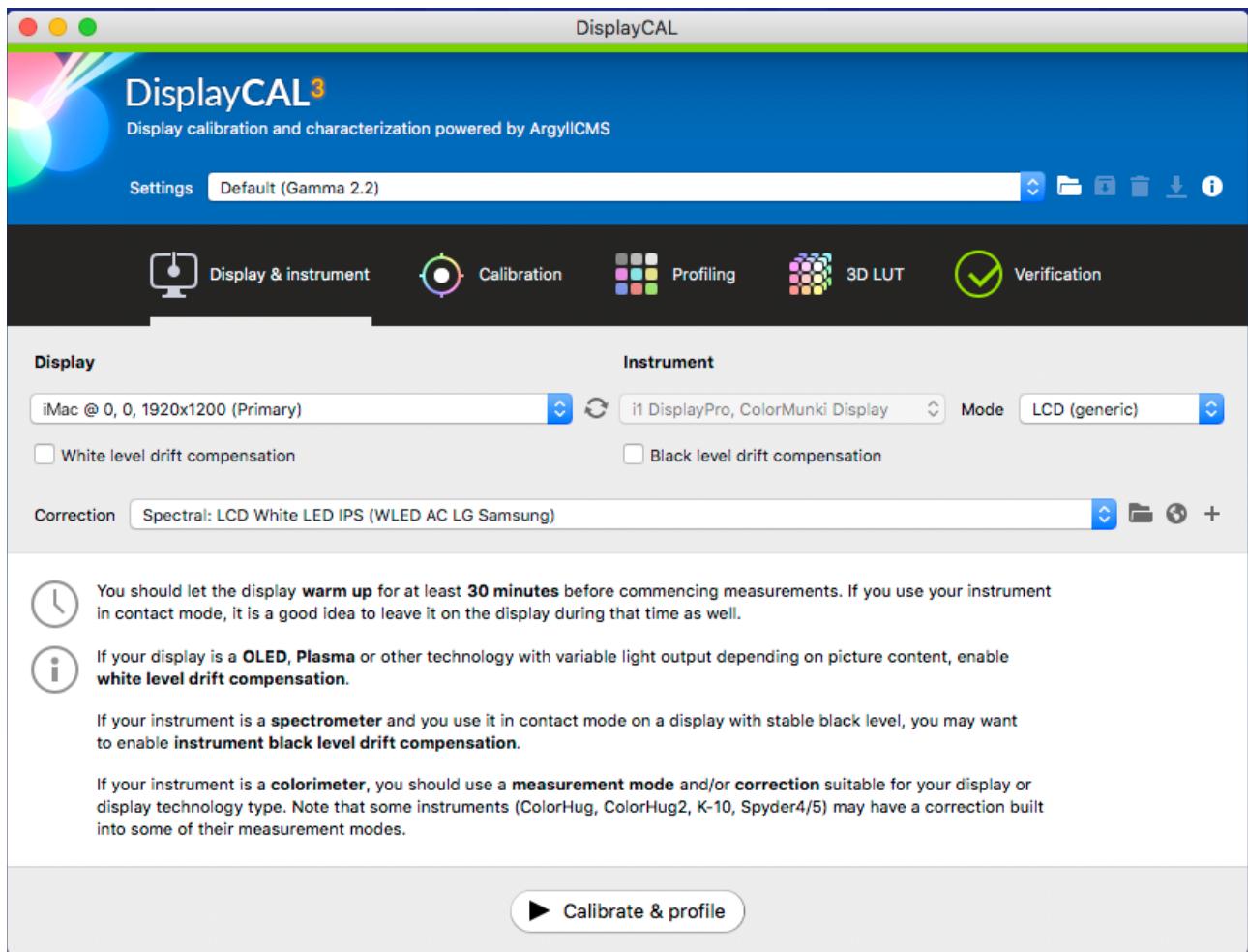


Рис. 2. Вигляд програми DisplayCAL

Ці програми надають зручний спосіб тестування різних аспектів монітора та аналізу їхньої працездатності. Використання таких програм може допомогти виявити проблеми та покращити якість зображення, що відображається на екрані монітора.

1.3. Функціональні можливості вкладки "Tools/Monitor Diagnostics" у програмі "AIDA64"

Вкладка "Tools/Monitor Diagnostics" у програмі "AIDA64" дозволяє різні види тестування моніторів комп’ютерів, а саме:

- **Calibration tests** - ці тести дозволяють перевірити точність кольорів та градацію кольору на моніторі. Вони використовують спеціальні кольорові шаблони для перевірки кольорової точності та відтворення кольорів у різних зонах екрану.

- **Focus tests** - ці тести призначені для перевірки чіткості та роздільної здатності монітора. Вони використовують зображення з різними деталізованими елементами для оцінки, наскільки добре монітор відтворює дрібні деталі та текст.
- **Grid tests** - тести сітки допомагають перевірити рівномірність та розподіл пікселів на екрані монітора. Вони використовують різноманітні сітки та шаблони для перевірки, чи немає видимих артефактів, таких як нерівномірність освітлення, мерехтіння або дефекти пікселів.
- **Solid fills tests** - ці тести дозволяють перевірити рівномірність відтворення кольору на всьому екрані монітора. Вони використовують однорідні кольорові шаблони, щоб переконатися, що монітор рівномірно та точно відтворює кольори без видимих плям або змін у насиченості.
- **Gradient fills tests** - ці тести оцінюють здатність монітора відтворювати плавні переходи кольорів у градієнтних зображеннях. Вони демонструють різні градієнти з різними переходами кольорів, допомагаючи перевірити, чи немає видимих смуг або переривань у градієнтному переході.
- **Color palettes tests** - ці тести дозволяють перевірити репрезентацію кольорових палітр на моніторі. Вони використовують спеціально підібрани палітри для перевірки точності відтворення кольорів та відображення різних відтінків.
- **Reading tests** - ці тести перевіряють чіткість та легкість читання тексту на моніторі. Вони використовують різні шрифти та розміри тексту для оцінки, наскільки добре монітор відображає текстову інформацію з різною деталізацією та контрастом.

1.4. Обґрунтування необхідності вдосконалення та модернізації утиліти тестування моніторів комп'ютера

Необхідність вдосконалення та модернізації утиліти тестування моніторів комп'ютера "AIDA64" з вкладкою "Tools/Monitor Diagnostics" обумовлена кількома причинами:

1. Розширення функціоналу: з врахуванням швидкого розвитку технологій моніторів і збільшення їхньої складності, необхідно додати нові корисні можливості для їхнього тестування та діагностики. Це включає розширення аналізу роздільної здатності, колірної точності, частоти оновлення, яскравості, контрастності та інших параметрів монітора.

2. Покращення інтерфейсу користувача: забезпечення зручного та інтуїтивно зрозумілого інтерфейсу є важливим аспектом, оскільки користувачі повинні легко навігувати та отримувати доступ до необхідної інформації про свої монітори. Модернізація інтерфейсу може включати поліпшення взаємодії, оптимізацію розташування елементів, додавання графічних елементів та забезпечення зручності використання.

3. Розширення підтримки моніторів: з огляду на різноманітність моніторів на ринку, утиліта "AIDA64" повинна підтримувати якомога більше моделей і брендів моніторів. Важливо регулярно оновлювати базу даних і забезпечувати сумісність з новими моделями, щоб користувачі мали доступ до повного спектру функціональності тестування.

4. Підтримка нових технологій: з розвитком технологій моніторів, таких як HDR (High Dynamic Range), FreeSync, G-Sync тощо, необхідно вдосконалювати утиліту "AIDA64" для підтримки цих нових функцій та можливостей. Це дозволить користувачам отримувати актуальну та повну інформацію про свої монітори.

5. Забезпечення якості та надійності: важливо регулярно оновлювати та вдосконалювати утиліту "AIDA64" з вкладкою "Tools/Monitor Diagnostics" для забезпечення високої якості та надійності результатів тестування моніторів. Такі оновлення можуть включати виправлення помилок, оптимізацію алгоритмів тестування та покращення загальної стабільності програми.

Одним словом, вдосконалення та модернізація утиліти "AIDA64" з вкладкою "Tools/Monitor Diagnostics" необхідні для забезпечення користувачам доступу до потужних та сучасних інструментів тестування та діагностики

моніторів, а також покращення загального досвіду користувачів у галузі моніторингу комп'ютерних компонентів.

2. Аналіз завдання та вибір підходів до проектування утиліти

2.1. Аналіз функцій та вибір структури програмного рішення

2.1.1. Уточнення функцій

1. **Calibration test** – тест який дозволяє відкалибрувати кольори та розширення на моніторі комп'ютера. Даний тест включає в себе такі тести:
 - 1.1. **LCD-Calibration** – це процес налаштування параметрів дисплея для досягнення оптимальної якості зображення. Під час калібрування LCD використовуються спеціальні інструменти та програмне забезпечення для налаштування таких параметрів, як яскравість, контрастність, кольорова гама, тон, насиченість та інші. Калібрування LCD дозволяє досягнути точного та стабільного відображення зображень, що особливо важливо для професійних застосувань, таких як медична діагностика, графічний дизайн та фотографія.
 - 1.2. **Brightness/Contrast** – даний тест позволяє налаштовувати яскравість та контраст монітора
 - 1.3. **Scope** – це тест, який використовується для перевірки якості відображення монітора. Цей тест відображає на екрані різні геометричні фігури, кольори і рівні яскравості з метою перевірки точності відтворення кольорів, контрастності, роздільної здатності, кута огляду та інших параметрів. Цей тест допомагає виявляти проблеми з відображенням на моніторі, такі як затемнення, блимання, артефакти, розмивання та інші. Використання scope тесту монітора може допомогти поліпшити якість відображення, знизити втомлюваність очей та забезпечити більш комфортну роботу з комп'ютером.
 - 1.4. **Gamma** – це тест, який позволяє налаштовувати гаму кольорів

- 1.5. **Convergence** - це спосіб перевірити якість відображення на екрані монітора шляхом перевірки точності відображення кольорів на різних частинах екрану. В процесі цього тесту на екрані з'являється сітка, складена з різно кольорових ліній, і користувач повинен переконатися, що кольори на краях ліній збігаються з кольорами по центру. Якщо кольори на краях ліній відрізняються від кольорів по центру, це може свідчити про проблеми з конвергенцією монітора, які можуть бути вирішенні за допомогою налаштувань монітора.
- 1.6. **Screen size** – це тест який перевіряє розширення монітора
- 1.7. **Focus tests** – це тести, які використовуються для визначення якості фокусування зображення на моніторі. Вони можуть включати в себе різноманітні тестові зображення, такі як текст, графіки та зображення, що допомагають визначити чіткість та рівномірність зображення на різних ділянках екрану. Ці тести допомагають виявити проблеми з фокусуванням, такі як розмиття, розмивання та нерівномірність зображення. Даний тест складається з таких тестів:
- 1.8. **White Pattern** – даний тест висвітлює білі квадратики на чорному фоні
- 1.9. **Black Pattern** – даний тест висвітлює чорні квадратики на білому фоні
- 1.10. **Sharpness** - є тестом, який дозволяє визначити рівень чіткості і деталізації зображення.
- 1.11. **Dots** - це тест, що використовується для перевірки якості відображення на моніторі. Зазвичай це маленькі точки різного кольору або форми, що відображаються на екрані монітора. Цей тест дозволяє виявити дефекти монітора, такі як dead pixel (мертві пікселі), stuck pixel (застрялий піксель), розмитість і нечіткість зображення.

1.12. **Vertical lines** – це тест, який допомагає визначити якість відтворення вертикальних ліній на моніторі.

1.13. **Horizontal lines** - це тест, який допомагає визначити якість відтворення горизонтальних ліній на моніторі.

2. **Grid tests** – є одним із способів перевірки геометрії монітора. Під час проведення цього тесту на екрані монітора відображається сітка з горизонтальними та вертикальними лініями, які повинні бути рівномірно розташовані та відстань між ними повинна бути однаковою. Цей тест допомагає виявити будь-які відхилення у відображені геометрії монітора, такі як зміщення ліній, нахил, нерівномірність розміщення ліній тощо. Даний тест може виводити лінії на білому або чорному фоні, а самі лінії можуть бути білого, чорного, червоного, зеленого, синього, фіолетового або жовтого кольору.
3. **Color tests** - це спеціальні зображення, які використовуються для перевірки правильності відображення кольорів на моніторі. Ці тести можуть допомогти виявити проблеми з кольоровою точністю монітора, такі як відхилення від правильної кольорової гами, нахил кольорів, недостатня насиченість кольорів та інші аномалії. Даний тест включає в себе такі тести кольорами, як червоний, зелений, синій, оранжевий, білий, чорний та сірий. Також даний тест може виводити відтінки кольорів, а саме червоний градієнт, зелений градієнт, синій градієнт, оранжевий градієнт та сірий градієнт.
4. **Reading test** – це спеціальний тест, який використовується для перевірки читабельності тексту на моніторі. Ці тести перевіряють різні аспекти читабельності, такі як розмір шрифту, яскравість фону, контрастність тексту, якість рендерингу шрифту та інші. Під час даного тесту на екран виводяться одне і те саме речення різними кольорами в різній кількості.
 - Help system – допомога з використанням додатку
 - About – про програму

- Використання піктограми - для швидкого запуску утиліти

2.1.2. Вибір структури утиліти на рівні модулів

В розробці утиліти, вибір її структури на рівні модулів є критичним кроком, оскільки це визначає організацію функціональних компонентів та їх взаємодію між собою. Правильно підібрана структура модулів дозволяє досягти високої модульності, зрозуміlostі коду, спрощує розширення та підтримку системи.

Під час вибору структури модулів для моого курсового проекту, я розглянув різні аспекти утиліти і врахував їх функціональну залежність та взаємодію. З метою забезпечення чіткого розподілу відповідальностей та зменшення зв'язків між модулями, я обрав наступну структуру на рівні модулів, яка зображена на рис. 3.

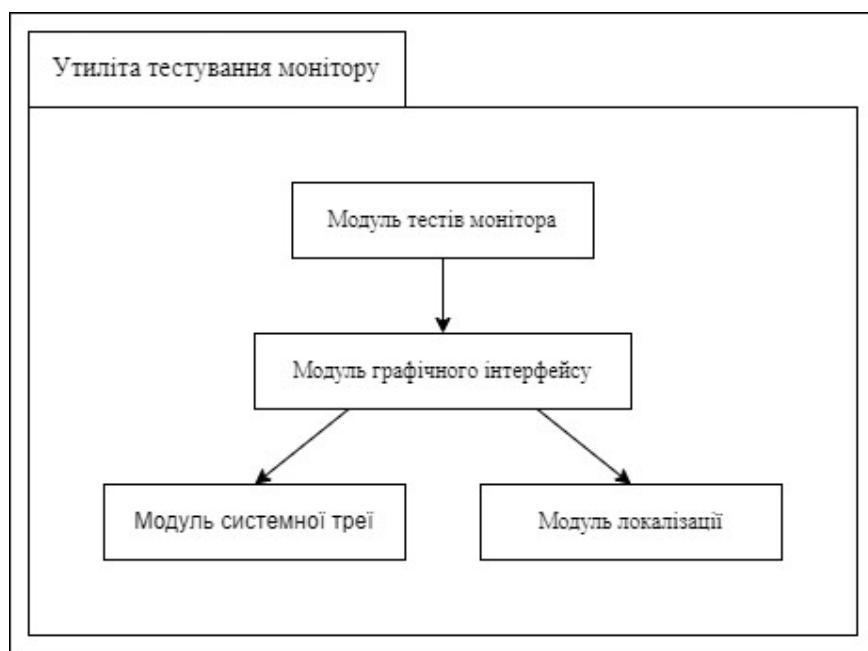


Рис. 3. Структура програмної реалізації утиліти тестування монітору

Як ми можемо побачити з рис. 3 утиліта складається з чотирьох модулів, а саме:

- **Модуль тестування монітора** виконує різні види тестів для оцінки працездатності монітора комп'ютера, зокрема калібрування, фокусування,

перевірку читабельності та відображення тексту, тестування кольорів, відтінків кольорів та сіткового тесту.

- **Модуль графічного інтерфейсу** відповідає за створення графічного інтерфейсу користувача. Він містить різні компоненти але найважливіші з них це кнопки для запуску тестів, поля виведення зображення на екран комп'ютера та інші елементи, які дозволяють користувачу взаємодіяти з утилітою.
- **Модуль системної треї** забезпечує інтеграцію утиліти з системою тресю (system tray). Він відповідає за функціональність, таку як показ/приховання утиліти, відображення сповіщень або контекстного меню в системній треї.
- **Модуль локалізації** відповідає за адаптацію програми до англійської та української мов. Цей модуль забезпечує можливість використання програми в різних мовних середовищах.

2.2. Вибір архітектурного рішення

2.2.1. Архітектура: модель – подання (MVC)

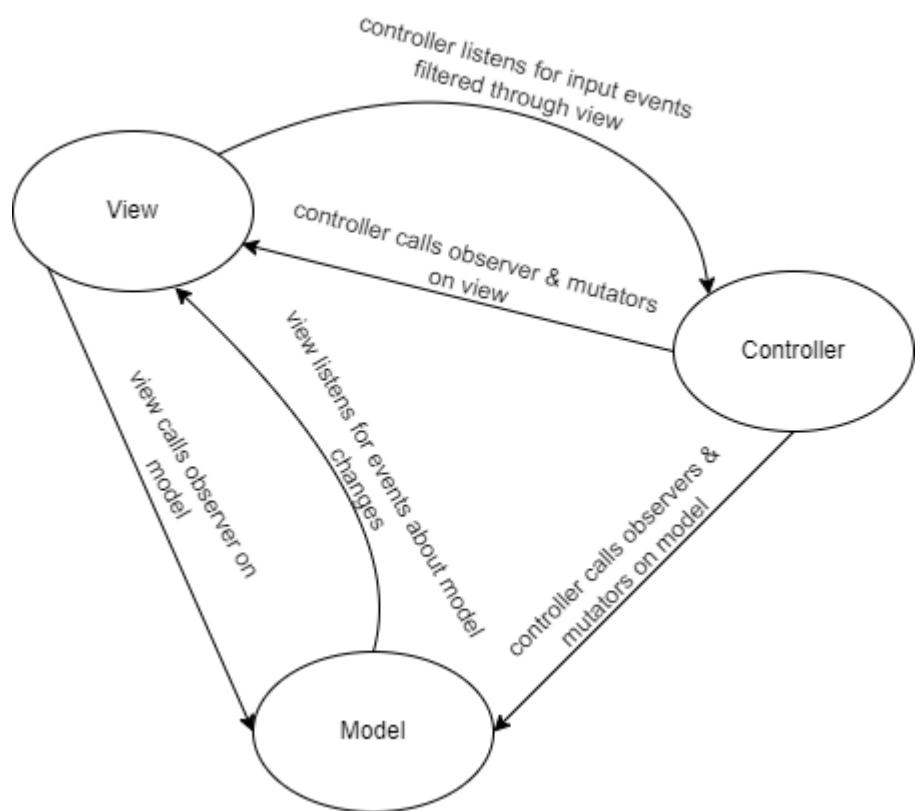
Для відображення даних у своїй програмі я буду користуватися моделлю під назвою "Model-View Programming". Модель-подання-контролер (MVC) є популярним підходом до організації програмного забезпечення, особливо в розробці додатків з користувацьким інтерфейсом. MVC розділяє компоненти системи на три основні частини: модель, подання та контролер.[3]

1. Модель (Model): Модель відповідає за управління даними та бізнес-логікою програми. Вона зберігає дані, виконує операції з ними, обробляє запити та забезпечує правильність даних. Модель представляє собою абстракцію реальних об'єктів або процесів, з якими працює програма.

2. Подання (View): Подання відповідає за візуалізацію даних та взаємодіє з користувачем. Воно відображає інформацію з моделі у зручній для сприйняття формі, такій як веб-сторінки, графічний інтерфейс тощо. Подання також обробляє користувацькі події і передає їх контролеру для подальшої обробки.

3. Контролер (Controller): Контролер взаємодіє з користувачем та виконує керуючі функції. Він обробляє користувальські запити, взаємодіє з моделлю та відправляє відповідні дані до подання. Контролер також відповідає за координацію роботи моделі та подання.

Головна ідея архітектури MVC полягає в тому, що кожен компонент має чітко визначені обов'язки. Це сприяє збереженню модульності, зрозумілості коду, полегшує розширення та підтримку системи. Крім того, MVC розділяє логіку додатку на рівні, що дозволяє розробникам працювати над окремими компонентами незалежно один від одного.



Rис. 4. Узагальнена діаграма “Model-View Programming”

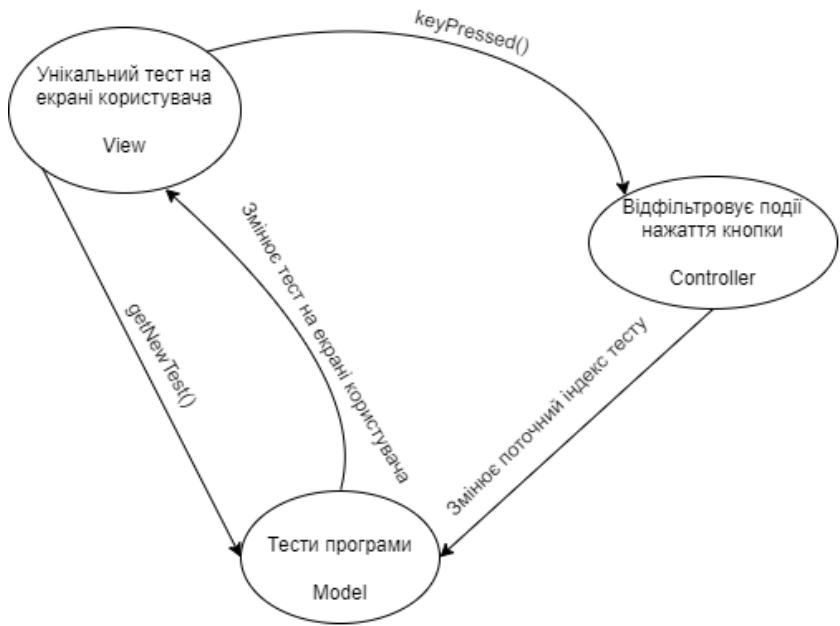


Рис. 5. Діаграма “Model-View Programming” для виконання тестів утиліною

2.2.2. Об'єктно – орієнтований підхід

Для розробки даної утиліти я буду притримуватися парадигми об'єктно-орієнтованого програмування (ООП[4]), яка має декілька переваг для розробки утиліти для тестування монітору комп'ютера:

- **Модульність і перевикористання коду:** ООП дозволяє організувати функціональність утиліти у вигляді незалежних об'єктів, які можуть бути легко модифіковані та перевикористані в інших проектах. Наприклад, ви можете створити окремий об'єкт для кожного типу тесту монітора, такого як калібрування чи перевірка кольорів, і легко комбінувати їх у своїй утиліті.
- **Ієрархія та спадкування:** ООП дозволяє створювати ієрархію класів, що спрощує організацію та категоризацію різних типів тестів. Наприклад, ви можете мати базовий клас "Тест монітора", від якого успадковуються класи "Калібрування", "Перевірка кольорів" і т.д. Це спрощує розширення та додавання нових типів тестів у майбутньому.[5]
- **Засоби абстракції:** ООП дозволяє абстрагувати складність системи шляхом розбиття на окремі об'єкти з визначеними властивостями та

методами. Це полегшує розуміння та управління кодом, а також сприяє зменшенню залежностей між різними компонентами.

- **Зручність управління станом:** ООП дозволяє зберігати стан об'єктів, що відповідають різним аспектам тестів монітора. Наприклад, ви можете мати об'єкт, який представляє стан калібрування монітора зі збереженими значеннями налаштувань. Це дозволяє зручно керувати та змінювати стан тестів в процесі їх виконання.
- **Читабельність та підтримка:** ООП підвищує читабельність коду і спрощує підтримку системи. Код, організований згідно з принципами ООП, стає більш структурованим, зрозумілим та підтримуваним, що сприяє подальшій розробці та розширенню функціональності утиліти.

Використання ООП у розробці утиліти для тестування монітора комп'ютера допоможе зробити код більш організованим, модульним та легко розширюваним, що в свою чергу покращить зручність використання та підтримки утиліти.

2.2.3. Механізм сигналів і слотів

Механізм сигналів та слотів є важливим компонентом Qt фреймворку і дозволяє реалізувати ефективну комунікацію та взаємодію між об'єктами у розроблені даної програмі[6]. Основна ідея полягає в тому, що об'єкт може випускати сигнали, які інші об'єкти можуть приймати та обробляти у своїх слотах. У своїй програмі я можу використати даний механізм таким чином:

Зміна тесту при натисканні на стрілочки клавіатури. Фреймворк Qt надає сигнал keyPressed з класу, який відповідає за обробку введення клавіатури. При натисканні стрілочок клавіатури, цей сигнал випускається з відповідними даними, такими як напрямок стрілки. Інші об'єкти, наприклад, об'єкт, який відповідає за відображення поточного тесту, можуть мати слот, який реагує на цей сигнал і змінюює поточний тест залежно від отриманих даних.

Виведення попереднього перегляду тесту в клієнтському меню при наведенні на назву тесту. Фреймворк Qt надає сигнал hovered з об'єкту, який

представляє називу тесту в клієнтському меню. Коли користувач наводить курсор на називу тесту, цей сигнал випускається з відповідними даними, такими як індекс тесту. Інші об'єкти, наприклад, об'єкт, який відповідає за зміну картинки в меню, можуть мати слот, який реагує на цей сигнал і змінює картинку відповідно до отриманих даних.

Таким чином, використання механізму сигналів та слотів дозволяє забезпечити гнучку та розширену взаємодію між різними компонентами програми, динамічно змінювати поведінку та взаємодію об'єктів, використовуючи сигнали та слоти, без необхідності використання прямих звернень та залежностей. Крім того, цей підхід полегшує розробку, підтримку та тестування коду.

2.3. Вибір засобів для реалізації

2.3.1. Вибір мови програмування

Для написання програми було обрано мову програмування C++ з використанням фреймворку Qt для створення графічного інтерфейсу

2.3.2. Вибір середовища

Середовищем в якому відбудуватиметься написання коду є Qt Creator під керуванням ОС Windows. Це інтегроване середовище розробки (IDE), призначене для створення програм на основі фреймворку Qt. Воно надає розробникам зручний інтерфейс, інструменти та функціональні можливості для створення графічних додатків[9].

2.3.3. Вибір готових бібліотек

У даній програмі буде використано різні бібліотеки фреймворку Qt основні з них це:

1. QPainter: QPainter є класом, який надає функціональність для малювання та відображення графічних об'єктів на вікні або на різних пристроях виводу. Він дозволяє рисувати лінії, криві, прямокутники, еліпси, текст та інші

графічні елементи на графічних пристроях. QPainter може бути використаний для малювання на QWidget, QPaintDevice, QPixmap, QImage та інших класах.

2. QEvent: QEvent є класом, який представляє події в Qt. Події можуть бути різних типів, таких як натискання клавіші, відпускання клавіші, натискання кнопки миші, переміщення миші, зміна розміру вікна та інші. QEvent надає спосіб перехоплення та обробки цих подій у вашій програмі. Ви можете створювати власні обробники подій, успадковуючи від класу QObject і переоприділяючи метод event() для обробки конкретних типів подій.

3. QPixmap: QPixmap є класом, який використовується для роботи з растровими зображеннями в Qt. Він дозволяє завантажувати, зберігати та маніпулювати растровими зображеннями. QPixmap може бути використаний для відображення зображень на вікні або для створення елементів інтерфейсу, які включають зображення. Він також підтримує різні операції зображення, такі як масштабування, обрізання, поворот та інші.

3. Програмна реалізація утиліти

3.1. Проектування та вибір структур даних

Створимо необхідні структури даних для полегшення роботи нашої програми.

1. Перелік, який зберігає поточний тип локалізації (було використано вбудований в C++17 тип `enum class`, який є покращеною і більш захищеною версією типу `enum`)

```
enum class LanguageMode {  
    kUA, // дане поле відповідає за українську мову  
    kEN // дане поле відповідає за англійську мову  
};
```

2. `std::vector<uint8_t> testCodeVector` - структура даних, яка зберігає ідентифікатор тесту монітора.

Ідентифікатори тестів:

Focus And Calibration tests — 1

Grid tests with Black background — 2

Grid tests with White background — 3

Solid color tests — 4

Gradient color tests — 5

Color palette tests — 6

*Color palette tests 180** — 7

Read tests — 8

3. `std::vector<QColor> testColorVector` - структура даних, яка зберігає кольори для Solid color tests та Gradient color tests.

4. `std::vector<uint8_t> testFocusVector` - структура даних, яка зберігає ідентифікатори для Focus та Calibration тестів.

Ідентифікатори FOCUS tests:

WhitePattern — 7

BlackPattern — 8

Sharpness - 9

Dots - 1

VerticalLines – 2

HorizontalLines – 3

Ідентифікатори **Calibration tests**:

LCDCalibration – 4

Brightness – 5

Scope – 11

Gamma – 6

Convergence – 12

ScreenSize – 10

3.2. Розробка алгоритму роботи утиліти

У даному розділі проведено розробку алгоритму роботи утиліти на рівні користувачького використання. Під час розробки алгоритму були враховані основні функції та можливості, які надає утиліта тестування монітора комп'ютера. Було визначено послідовність дій користувача, необхідних для запуску тестів монітора чи перегляду інформації про програму.

Для реалізації алгоритму роботи утиліти (*рис. 6.*) використано об'єктно – орієнтований підхід і розроблено діаграму класів, яка подана в додатку Б та на *рис. 7.*

Алгоритм роботи утиліти включає в себе такі кроки, як завантаження основних модулів для коректного функціонування, переключення між вікнами утиліти, приховування в системну трею, зміна локалізації, зміна налаштувань утиліти, запуск тестів та отримання інформації про програму.

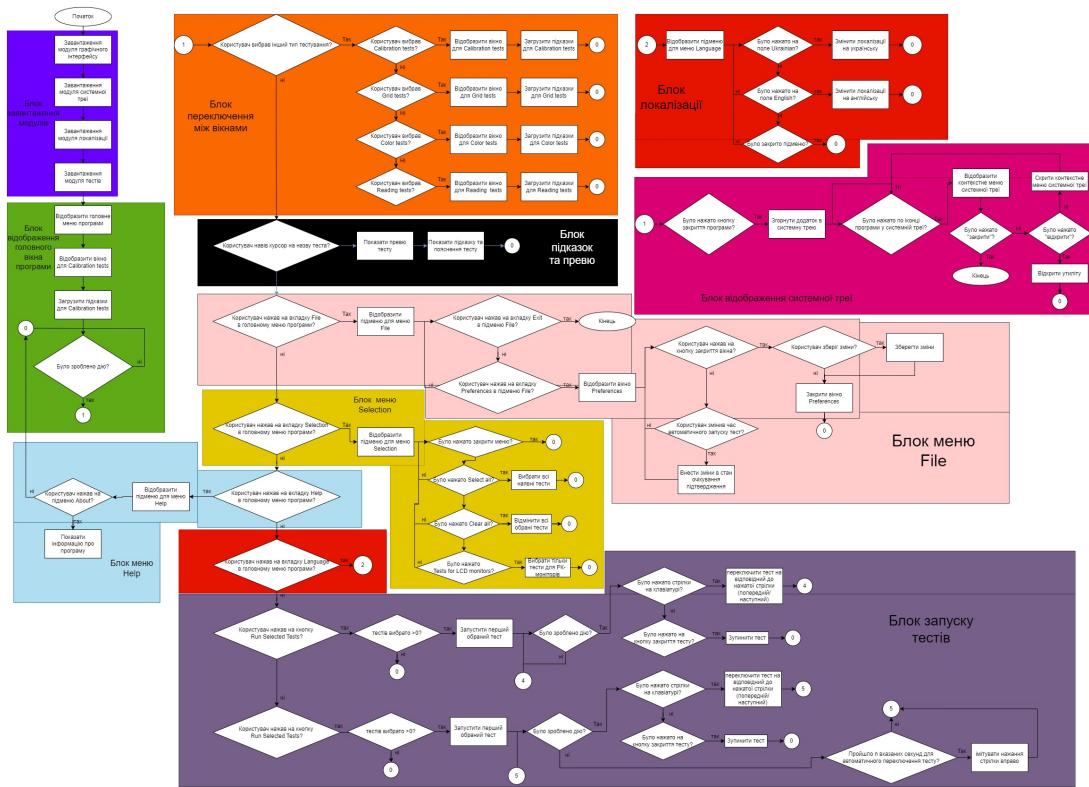


Рис. 6. Схема алгоритму роботи програми “Monitor testing”

Опис схеми алгоритму роботи програми:

- В блоці **завантаження модулів** виконується завантаження модулів із яких складається програма, а саме: графічного модуля, модуля тестів, модуля локалізації та модуля системної треї.
- В блоці **відображення головного вікна програми** виконується відображення головного вікна програми, а саме: відображення вікна для певного тесту та завантаження підказок для певного тесту.
- В блоці **меню Help** виконується відображення меню Help, та при натисканні на About відображається вікно з інформації про програму.
- В блоці **меню Selection** виконується або вибір всіх тестів, або очищення всіх тестів або вибір тестів для РК-монітору.
- В блоці **меню File** виконується відображення меню File, та при натисканні на *Exit* буде закрито програму, а при натисканні на *Preferences* відкриється меню з вмістом налаштувань програми.
- В блоці **відображення системної треї** відбувається приховання утиліти в системну трею при натисканні кнопки “**Закрити**”. При натисканні на

системну трею відбувається відкриття контекстного меню для взаємодії з програмою. Якщо в контекстному меню натиснуто “**Закрити**” то утиліта завершує свою роботу повністю. Якщо ж нажати “**Відкрити**” то відкриється вікно утиліти.

- **В блоці локалізації виконується** переключення мови інтерфейсу програми.
- **В блоці запуску тестів** виконується запуск тестів.
- **В блоці підказок та превью** виконується відображення підказок та перед огляду тестів.
- **В блоці переключання між вікнами** відображено алгоритм роботи програми при переключенні користувача між вікнами програми.

3.3 Розробка діаграми класів та її опис

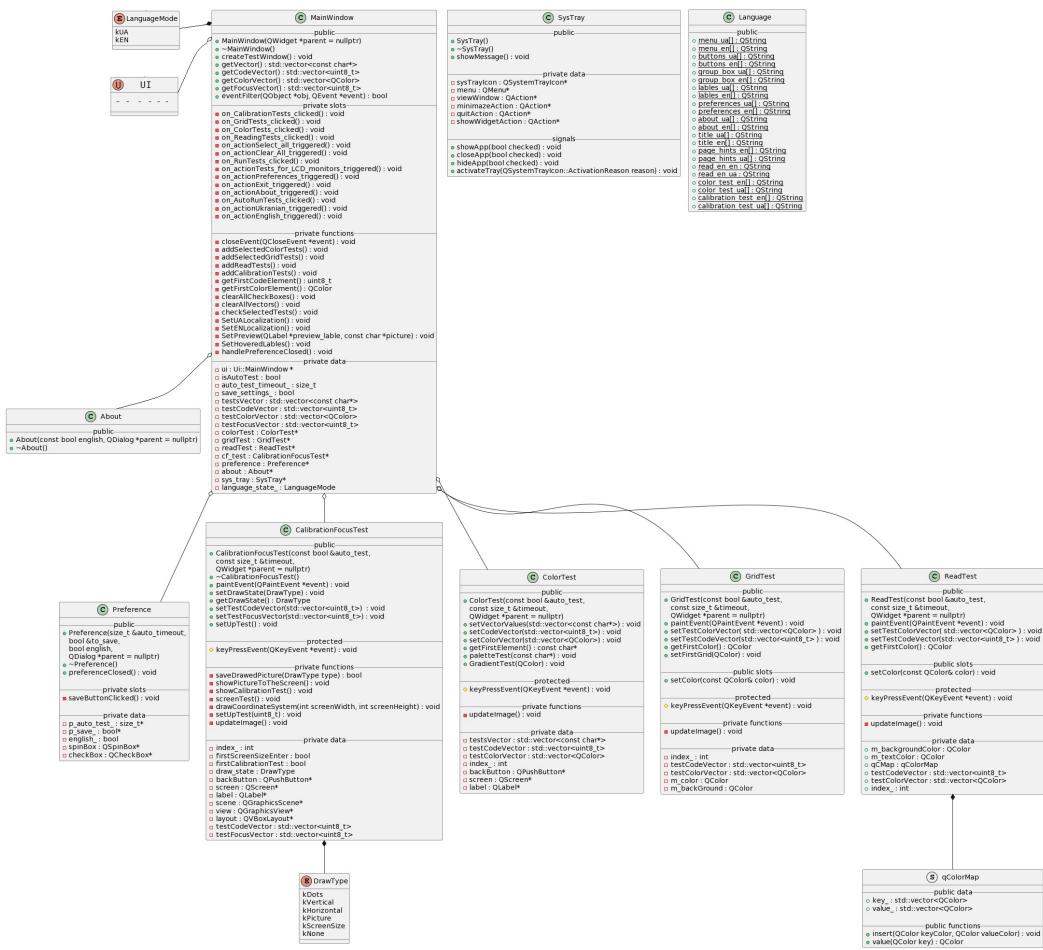


Рис. 7. Діаграма класів

На рис. 7 зображено діаграму класів розробленої утиліти, на якій можна побачити взаємозв'язки між класами та різними структурами даних.

Дана діаграма складається з дев'яти класів, двох переліків та однієї структури.

3.3.1. Опис класів та їх взаємозв'язків з іншими класами

1. Клас *MainWindow* - відповідає за обробку дій користувача.

MainWindow аргумент:

1. About – клас, який реалізовує вкладку “*Про програму*”.
 2. Preferences – клас, який реалізовує вкладку “*Уподобання\Налаштування*”.

3. *CalibrationFocusTest* — клас, який реалізовує логіку калібрувальних та фокусувальний тестів.
4. *ColorTest* – клас, який реалізовує логіку тестів на перевірку кольорової гами монітора.
5. *GridTest* – клас, який реалізовує логіку сіткових тестів.
6. *ReadTest* – клас, який реалізовує логіку тестів на перевірку читабельності та відображення тексту на моніторі.
7. *SysTray* – клас, який реалізує логіку згортання програми в системну трею.
8. *UI* – вбудований та неявно реалізований базовий клас, який надає можливості використання та створення графічного інтерфейсу програми.

MainWindow компонує:

1. *LanguageMode* – перелік, який відповідає за поточний стан локалізації.

Клас *CalibrationFocusTest* компонує перелік *DrawType*, який відповідає за тип тесту калібрування.

Клас *ReadTest* компонує структуру *qColorMap*, яка відповідає за призначення кольору тексту або фону вікна під час тестів для перевірки читабельності монітора комп’ютера.

Клас *Language*, який відповідає за локалізацію нічого **не** агрегує та **не** компонує.

3.4. Розробка модулів утиліти

3.4.1. Модуль тестів



Рис. 8. Діаграма класів модуля тестів

Модуль тестування складається з **четирех** класів:

- **CalibrationFocusTest** — клас, який реалізовує логіку калібрувальних та фокусувальний тестів.
- **ColorTest** — клас, який реалізовує логіку тестів на перевірку кольорової гами монітора.
- **GridTest** — клас, який реалізовує логіку сіткових тестів.
- **ReadTest** — клас, який реалізовує логіку тестів на перевірку читабельності та відображення тексту на моніторі.

3.4.1.1. Опис функцій класу CalibrationFocusTest

У класі **CalibrationFocusTest** виконуються калібрувальні та фокусувальні тести монітора.

Функція `void CalibrationFocusTest::setUpTest()` - обновляє тип тесту для його подальшого відображення на екрані користувача, беручи потрібний тип тесту з першої комірки приватного поля `testFocusVector`, дане поле зберігає всі коди калібрувальних та фокусувальних тестів, які були описані в розділі. 3.1. на сторінках 28-29.

Функція `void CalibrationFocusTest::setUpTest(uint8_t code)` — також, як і функція `CalibrationFocusTest::setUpTest()` обновляє тип тесту для його подальшого відображення на екрані, але беручи потрібний тип тесту за індексом, який передається в параметрах даної функції, а саме `uint8_t code`, даний параметр має тип `uint8_t`, який був використаний для економії пам'яті так як у нас всього **12** типів тестів, а тип `uint8_t` займає всього 8біт, які можуть зберегти будь-яке число в діапазоні **[0;255]**.

```
void CalibrationFocusTest::setUpTest(){
    if (testCaseVector[0] == 1) {
        switch (testFocusVector[0]) {
            case 4:
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
```

```

        case 11:
        case 12:
            draw_state = DrawType::kPicture;
            break;
        case 1:
            draw_state = DrawType::kDots;
            break;
        case 2:
            draw_state = DrawType::kVertical;
            break;
        case 3:
            draw_state = DrawType::kHorizontal;
            break;
        case 10:
            draw_state = DrawType::kScreenSize;
            break;
    }
}
}

```

Функції які присвоюють полям класу значення з свої параметрів:

```

void CalibrationFocusTest::setDrawState(DrawType state) {
    this->draw_state = state;
}
void CalibrationFocusTest::setTestCodeVector(std::vector<uint8_t> code_vec) {
    this->testCodeVector = code_vec;
}
void CalibrationFocusTest::setTestFocusVector(std::vector<uint8_t> focus_vec) {
    this->testFocusVector = focus_vec;
}

```

Функція `void CalibrationFocusTest::showCalibrationTest()` виводить картинку тесту на екран. Для цього використовується вбудовані класи Qt фреймворку `QLabel`[7] та `QPixmap`[8].

Функція `void CalibrationFocusTest::drawCoordinateSystem(int screenWidth, int screenHeight)` реалізує Screen Size тест. Спочатку ми створюємо дві локальні змінні та ініціалізуємо їх даними з параметрів функції:

```

int coordWidth = screenWidth; // ширина екрану монітора комп'ютера
int coordHeight = screenHeight; // висота екрану монітора комп'ютера
після чого ми визначаємо координати центра монітора:

```

```

int centerX = screenWidth / 2;
int centerY = screenHeight / 2;

```

далі ми малюємо сітку на весь екран з квадратами 50x50 пікселів:

```

for (int x = 0; x <= coordWidth; x += 50) {
    scene->addLine(x, 0, x, coordHeight, gridPen);
}
for (int y = 0; y <= coordHeight; y += 50) {
    scene->addLine(0, y, coordWidth, y, gridPen);
}

```

після чого ми вимальовуємо на екрані шкалу з позначенням розширень монітора:

```
// по x
    QPen xPen(AxisColor);
    xPen.setWidth(2);
    scene->addLine(centerX - coordWidth / 2, centerY, centerX + coordWidth / 2,
centerY, xPen);
    QGraphicsTextItem *textItemX = new QGraphicsTextItem("(" +
QString::number(screenWidth) + ")");
    textItemX->setDefaultTextColor(Qt::white);
    textItemX->setFont(TextFont);
    textItemX->setPos(centerX + coordWidth / 2 - 40, centerY - 40);
    scene->addItem(textItemX);
// по y
    QPen yPen(AxisColor);
    yPen.setWidth(2);
    scene->addLine(centerX, centerY - coordHeight / 2, centerX, centerY + coordHeight
/ 2, yPen);
    QGraphicsTextItem *textItemY = new
QGraphicsTextItem("(" +QString::number(screenHeight) + ")");
    textItemY->setDefaultTextColor(Qt::white);
    textItemY->setFont(TextFont);
    textItemY->setPos(centerX - 40, centerY + coordHeight / 2 - 20);
    scene->addItem(textItemY);
```

після чого наносимо текст та позначення на екран комп'ютера:

```
// Calculate the number of points to mark on the x-axis
int numPointsX = coordWidth / 50;
int startX = centerX - coordWidth / 2;
// Draw the points on the x-axis
for (int i = 0; i <= numPointsX; ++i) {
    int x = startX + i * 50;
    scene->addLine(x, centerY - 2, x, centerY + 2, xPen);
    QGraphicsTextItem *textItemX = new QGraphicsTextItem(QString::number(i * 50));
    textItemX->setDefaultTextColor(Qt::white);
    textItemX->setFont(TextFont);
    if (i == 0) {
        textItemX->setPos(x, centerY + 10);
    } else if (i % 2 == 0) {
        textItemX->setPos(x - 10, centerY + 10);
    } else {
        textItemX->setPos(x - 10, centerY - 20);
    }
    scene->addItem(textItemX);
}
// Calculate the number of points to mark on the y-axis
int numPointsY = coordHeight / 50;
int startY = centerY - coordHeight / 2;
// Draw the points on the y-axis
for (int i = 0; i <= numPointsY; ++i) {
    int y = startY + i * 50;
    scene->addLine(centerX - 2, y, centerX + 2, y, yPen);
    QGraphicsTextItem *textItemY = new QGraphicsTextItem(QString::number(i * 50));
    textItemY->setDefaultTextColor(Qt::white);
    textItemY->setFont(TextFont);
    if (i == 0) {
        textItemY->setPos(centerX - 40, y);
```

```

        } else if (i % 2 == 0) {
            textItemY->setPos(centerX - 40, y - 10);
        } else {
            textItemY->setPos(centerX + 10, y - 10);
        }
        scene->addItem(textItemY);
    }
}

```

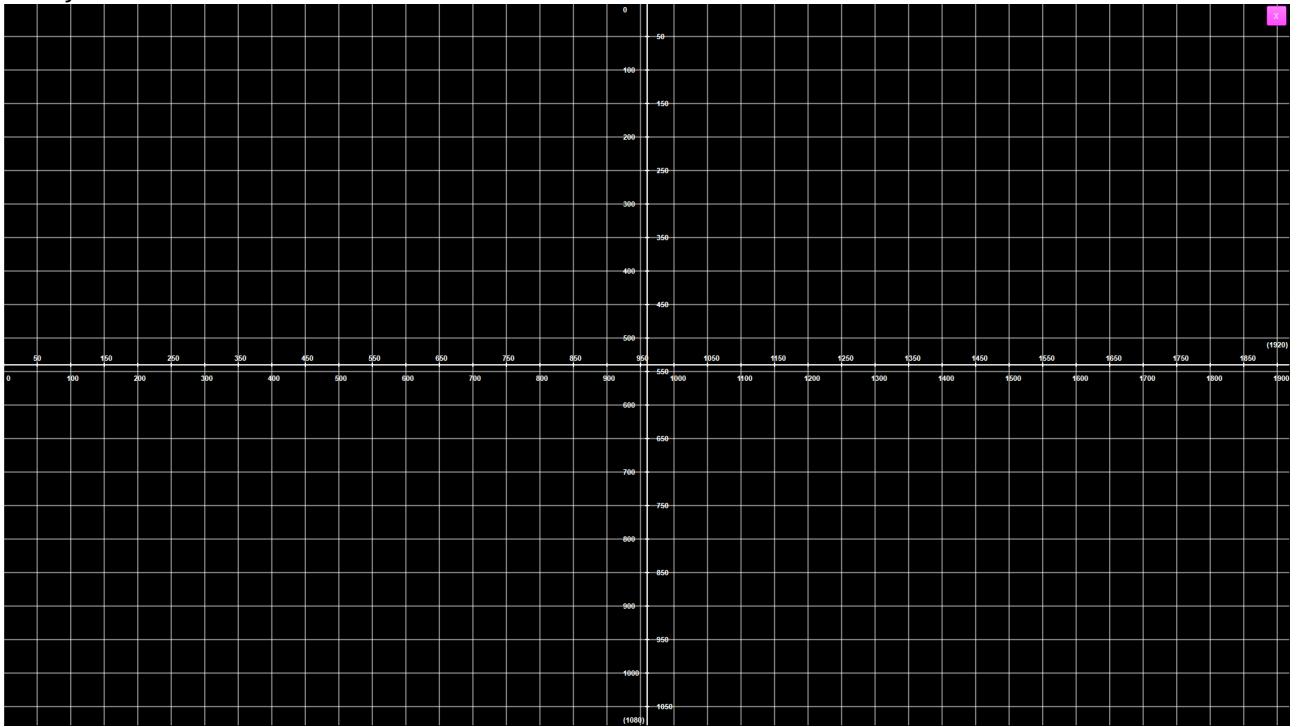


Рис. 9. Результат виконання Screen Size тесту

Функції `bool CalibrationFocusTest::saveDrawedPicture(DrawType type)` та `void CalibrationFocusTest::showPictureToTheScreen()` виконують 3 різних тести в залежності від поточного типу тесту. Спочатку функція `bool CalibrationFocusTest::saveDrawedPicture(DrawType type)` вимальовує та зберігає картинку відповідно до тесту(Dot, Vertical lines та Horizontal lines) після чого функція `void CalibrationFocusTest::showPictureToTheScreen()` виводить даний тест на екран.

Також наявна функція, яка при натисканні на стрілки клавіатури міняє тест на наступний\попередній. Дано функція має назву `void CalibrationFocusTest::keyPressEvent(QKeyEvent* event)`, використовується вбудований в фреймворк Qt механізм `QKeyEvent`, який при натисканні любої клавіші на клавіатурі генерує відповідний сигнал після чого моя функція `keyPressEvent` хендлить даний сигнал та обробляє його.

3.4.1.2. Опис функцій класу ColorTest

Функції сеттери, які присвоюють полям класу значення з своїх параметрів:

```
void ColorTest::setVectorValues(std::vector<const char *> v) {  
    this->testsVector = v;  
}  
void ColorTest::setCodeVector(std::vector<uint8_t> v){  
    this->testCodeVector = v;  
}  
void ColorTest::setColorVector(std::vector<QColor> v){  
    this->testColorVector = v;  
}
```

Функція `void ColorTest::paletteTest(const char *path)` — виконує palette тест, в параметри передається шлях, який відповідає картинці Palette або Palette180. Алгоритм роботи даної функції:

спочатку створюємо об'єкт класу `QLabel`:

```
label = new QLabel(this);
```

після чого запрошуємо в операційної системи розміри поточного розширення екрану:

```
QRect screenGeometry = QApplication::primaryScreen()->geometry();  
QSize screenSize = screenGeometry.size();
```

далі генеруємо картинку та виводимо її на екран:

```
QPixmap pixmap(path);  
label->setPixmap(pixmap.scaled(screenSize));  
label->setGeometry(QRect(QPoint(0, 0), screenSize));  
label->showFullScreen();  
backButton->raise();
```

Функція `void ColorTest::GradientTest(QColor color)` — виконує, тест з відтінками кольорів. Данна функція отримує початковий колір в параметри функції та виводить на екран градієнт даного кольору в лівому верхньому куті найтемніший відтінок в правому нижньому заданий колір і між цими кутами знаходяться всі відтінки цього кольору.

Solid тест проводиться за допомогою функції фреймворку Qt `setStyleSheet(QColor)` — дана функція міняє фон захопленого об'єкта в колір, який передається в параметри даної функції.

Також наявна функція, яка при натисканні на стрілки клавіатури міняє тест на наступний\попередній. Данна функція має називу `void`

`ColorTest::keyPressEvent(QKeyEvent *event)`, використовується вбудований в фреймворк Qt механізм `QKeyEvent`, який при натисканні любої клавіші на клавіатурі генерує відповідний сигнал після чого моя функція `keyPressEvent` хендлить даний сигнал та обробляє його.

3.4.1.3. Опис функцій класу GridTest

Головна функція `void GridTest::paintEvent(QPaintEvent *event)` — міняє колір сітки та фону вибраного тесту. Алгоритм роботи даної функції:

спочатку ми отримуємо поточне розширення екрану монітора та встановлюємо колір фону та колір сітки який в свою чергу беремо з приватного поля класу `QColor m_color`, яке відповідає за поточний колір сітки:

```
QScreen *screen = QGuiApplication::primaryScreen();
painter.setPen(QPen(m_color, 1, Qt::SolidLine));
```

після чого обчислюємо розмір клітинок:

```
const int cell_size = screen->geometry().width() / 20;
const int cell_size2 = screen->geometry().height() / 20;
```

потім визначаємо кількість клітинок, які влізуться на екран монітора і відстань між ними:

```
const int rows = 20;
const int cols = 20;
const int grid_width = cell_size * cols;
const int grid_height = cell_size2 * rows;
const int x_offset = (screen->geometry().width() - grid_width) / 2;
const int y_offset = (screen->geometry().height() - grid_height) / 2;
```

на завершення вимальовуємо сітку:

```
for (int i = 0; i <= cols; i++) {
    const int x = x_offset + i * cell_size;
    painter.drawLine(x, y_offset, x, y_offset + grid_height);
}
for (int i = 0; i <= rows; i++) {
    const int y = y_offset + i * cell_size2;
    painter.drawLine(x_offset, y, x_offset + grid_width, y);
}
```

Функція `void GridTest::setColor(const QColor &color)` міняє колір сітки відповідно до наступного кольору тесту.

Також наявна функція, яка при натисканні на стрілки клавіатури міняє тест на наступний\попередній. Даної функції має називу `void GridTest::keyPressEvent(QKeyEvent*event)`, використовується вбудований в фреймворк Qt механізм `QKeyEvent`, який при натисканні любої клавіші на

клавіатурі генерує відповідний сигнал після чого моя функція `keyPressEvent` хендлить даний сигнал та обробляє його.

3.4.1.4. Опис функцій класу Read Test

В класі `ReadTest` так само, як і у класі `GridTest` наявна функція `void ReadTest::setColor(const QColor &color)` - міняє колір тексту відповідно до наступного кольору тесту. Також міняючи задній фон програми, який береться з структури `qCMap`, яка реалізує структуру даних під назвою бінарне дерево ключем якого є колір тексту, а значенням колір фону.

Заповнення `qCMap`:

```
qCMap.insert(Qt::white, Qt::black);
qCMap.insert(Qt::black, Qt::white);
qCMap.insert(Qt::red, Qt::blue);
qCMap.insert(Qt::blue, Qt::green);
qCMap.insert(Qt::magenta, Qt::black);
qCMap.insert(Qt::yellow, Qt::blue);
```

Реалізація функції `void ReadTest::setColor(const QColor &color)`:

```
void ReadTest::setColor(const QColor &color){
    m_textColor = color;
    m_backgroundColor = qCMap.value(color);
    update();
}
```

Функція `void ReadTest::paintEvent(QPaintEvent *event)` є головною функцією даного класу. Данна функція вимальовує текст та задній фон на моніторі комп'ютера. Алгоритм роботи даної функції:

спочатку заповняємо задній фон вікна та встановлюємо колір і стиль тексту:

```
painter.fillRect(rect(), m_backgroundColor);
painter.setPen(m_textColor);
QFont font("Arial", 10);
painter.setFont(font);
```

після чого ми отримуємо поточне розширення екрану комп'ютера та підготовлюємо текст, який буде відображенний на екрані, визначаючи його довжину:

```
QScreen *screen = QGuiApplication::primaryScreen();
QString can_you_read_text = "Can you read text?";
long long text_length = (can_you_read_text.length() + 15) * 4;
```

отримавши поточне розширення екрану визначаємо кількість колонок і рядків з даним текстом, яке може поміститися на екрані комп'ютера:

```
int rowCount = screen->geometry().height() / 15;
int colCount = screen->geometry().width() / text_length;
```

```

int cellWidth = screen->geometry().width() / colCount;
int cellHeight = screen->geometry().height() / rowCount;

```

після завершення всіх цих дій, вимальовуємо на екрані комп'ютера текст:

```

for (int i = 0; i < rowCount; i++) {
    for (int j = 0; j < colCount; j++) {
        QRect rect(j * cellWidth, i * cellHeight + 15, cellWidth,
cellHeight);
        painter.drawText(rect, Qt::AlignCenter, "Can you read text?");
    }
}

```

Також, як і у всіх інших класах наявна функція, яка при натисканні на стрілки клавіатури міняє тест на наступний\попередній. Данна функція має назву `void GridTest::keyPressEvent(QKeyEvent*event)`, використовується вбудований в фреймворк Qt механізм `QKeyEvent`, який при натисканні любої клавіші на клавіатурі генерує відповідний сигнал після чого моя функція `keyPressEvent` хендлить даний сигнал та обробляє його.

3.4.2. Модуль локалізації



Rис.10. Діаграма класів модуля локалізації

Модуль локалізації в даній утиліті складається з одного класу, який зберігає усю текстову інформацію про програму.

3.4.2.1. Реалізація модуля локалізації

Для реалізації модуля локалізація було створено масиви стрічок для українського та англійського тексту, які при зміні локалізації користувачем просто міняються з англійської\української на українську\англійську. Даний принцип було вибрано через те, що його дуже просто реалізувати та він є дуже швидким з точки зору виконання. Мінус даного принципу полягає в тому, що нам потрібно більше пам'яті, щоб зберігати ці всі текстові дані тому було прийнято рішення зробити ці зміні статичними і розмістити їх тільки в .h файлі.

Таким чином під час запуску програми буде виконано тільки одне виділення пам'яті під всі тексти і дана пам'ять не буде видалятися\ перевиділятися до поки не завершиться виконання програми.

- Інформація про меню програми збережено в змінних:

static QString menu_ua[] - українська локалізація

static QString menu_en[] - англійська локалізація

- Інформація про кнопки програми збережено в змінних:

static QString buttons_ua[] - українська локалізація

static QString buttons_en[] - англійська локалізація

- Інформація про типи тестів програми збережено в змінних:

static QString group_box_ua[] - українська локалізація

static QString group_box_en[] - англійська локалізація

- Інформація про назви тестів програми збережено в змінних:

static QString lables_ua[] - українська локалізація

static QString lables_en[] - англійська локалізація

- Інформація про вкладку уподобання\налаштування програми збережено в змінних:

static QString preferences_ua[] - українська локалізація

static QString preferences_en[] - англійська локалізація

- Інформація про вкладку “Про програму” програми збережено в змінних:

static QString about_ua[] - українська локалізація

```
static QString about_en[] - англійська локалізація
```

- Інформацію про назву програми збережено в змінних:

```
static QString title_ua[] - українська локалізація
```

```
static QString title_en[] - англійська локалізація
```

- Інформацію з підказками програми збережено в змінних:

Українська локалізація:

```
static QString page_hints_ua[]
```

```
static QString read_ua
```

```
static QString color_test_ua[]
```

```
static QString calibration_test_ua[]
```

Англійська локалізація:

```
static QString page_hints_en[]
```

```
static QString read_en
```

```
static QString color_test_en[]
```

```
static QString calibration_test_en[]
```

3.4.3. Модуль системної треї

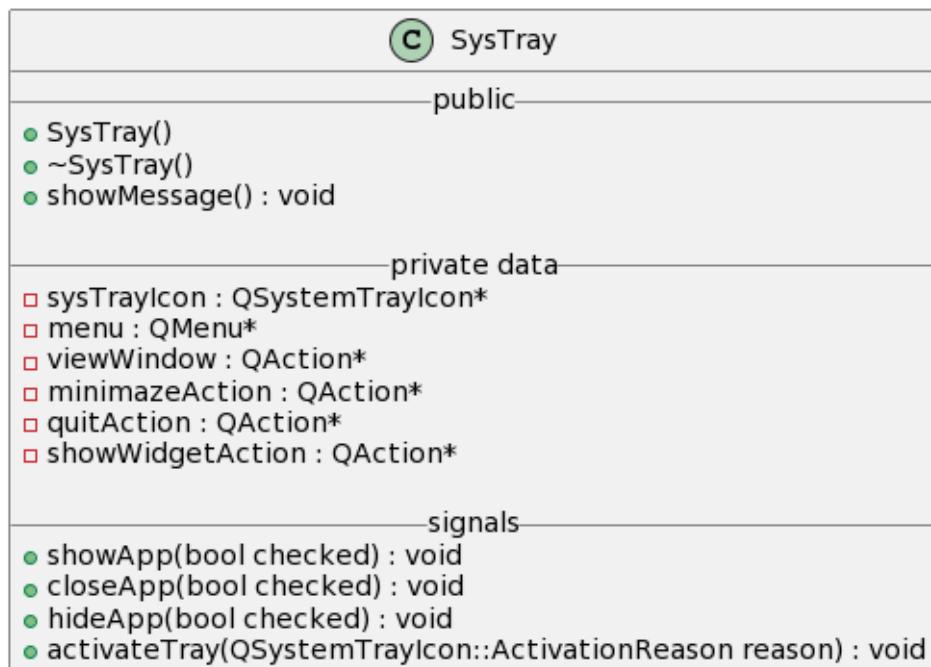


Рис.11. Діаграма класів модуля системної треї

Модуль системної треї складається із одного класу SysTray(рис. 11.), який і слугує для створення піктограми швидкого доступу.

Було створено необхідні поля:

- `QSystemTrayIcon *sysTrayIcon` - об'єкт системної треї, який представляє іконку та функціональність треї.
- `QMenu *menu` - контекстне меню, яке відображається при натисканні правою кнопкою миші по іконці програми у системній треї.
- `QAction *viewWindow` - дія, що відкриває головне вікно програми при її виборі у контекстному меню.
- `QAction *minimazeAction` - дія, яка мінімізує вікно програми до системної треї при виборі з контекстного меню.
- `QAction *quitAction` - дія, яка закриває програму при виборі з контекстного меню.

3.4.3.1. Опис реалізації модуля системної треї

Модуль системної треї створюється в конструкторі класу `SysTray` наступним чином:

1. Ініціалізуємо приватне поле класу `sysTrayIcon` та вказуємо у ньому шлях до іконки головної програми та назву, яка буде відображатися при наведені курсором миші на піктограму утиліти у системній треї:

```
sysTrayIcon = new QSystemTrayIcon(this);
sysTrayIcon->setToolTip(tr("Display testing"));
sysTrayIcon->setIcon(QIcon(":/color/palette/pictures/MonitorTesting.ico"));
після чого підключаємо сигнал, який буде емітитись при натисканні на іконку
утиліти в системній треї:
```

```
connect(sysTrayIcon, &QSystemTrayIcon::activated, this,
[this](QSystemTrayIcon::ActivationReason reason) { emit this->activateTray(reason); });
дальше налаштовуємо текст для системної треї:
```

```
menu = new QMenu(this);
viewWindow = new QAction(tr("Open"), this);
minimazeAction = new QAction(tr("Minimaze"), this);
quitAction = new QAction(tr("Quit"), this);
QFont font("Segoe UI", 12);
```

після виконання всіх попередніх кроків додаємо дії, які зможе використовувати наша утиліта в системній треї та прив'язуємо до них відповідні сигнали операційної системи:

```
menu->addAction(viewWindow);
menu->addAction(minimazeAction);
```

```

menu->addAction(quitAction);
menu->addSeparator();
menu->addAction(showWidgetAction);
menu->setStyleSheet(styleTray);
menu->setFont(font);
sysTrayIcon->setContextMenu(menu);
sysTrayIcon->show();
connect(viewWindow, &QAction::triggered, [this](bool checked = false) {
    emit this->showApp(checked);
});
connect(minimizeAction, &QAction::triggered, [this](bool checked = false) {
    emit this->hideApp(checked);
});
connect(quitAction, &QAction::triggered, [this](bool checked = false) {
    emit this->closeApp(checked);
});

```

3.4.4. Модуль графічного інтерфейсу



Рис.12. Діаграма класів модуля графічного інтерфейсу

Модуль графічного інтерфейсу складається із таких класів:

- **MainWindow** — головний (керуючий) клас графічного інтерфейсу, який відповідає за відображення усієї інформації, яка є доступна користувачеві.

- About — клас, який відповідає за графічне представлення інформації про програму.
- Preference — клас, який відповідає за графічне представлення налаштування програми.
- LanguageMode — перелік, який відповідає за поточний стан локалізації.

3.4.4.1. Реалізація та опис функцій модуля графічного інтерфейсу

Для створення графічного інтерфейсу були створені такі поля:

- `Ui::MainWindow *ui` — поле, яке відповідає за все налаштування графічного інтерфейсу.
- `Preference *preference` — вказівник на об'єкт класу Preference, який відповідає за графічне представлення налаштування програми.
- `About *about` — вказівник на об'єкт класу About, який відповідає за графічне представлення інформації про програму.

Для реалізації логіки тестування монітора комп'ютера були створені наступні поля:

- `bool isAutoTest` — відповідає за те, чи користувач нажав на кнопку авто тестування.
- `size_t auto_test_timeout_` — відповідає за перемикання тестів при автоматичному тестуванні.
- `bool save_settings_` — відповідає за збереження налаштувань утиліти.
- `std::vector<const char*> testsVector` — контейнер, який зберігає стиль вікон відповідних тестів.
- `std::vector<uint8_t> testCodeVector` — контейнер, який зберігає ідентифікатори тестів, які описані у розділі 3.1 на сторінці 28.
- `std::vector<QColor> testColorVector` — контейнер, який зберігає кольори для тестів.
- `std::vector<uint8_t> testFocusVector` — контейнер, який зберігає ідентифікатори калібрувальних та фокусувальних тестів, які були описані у розділі 3.1 на сторінках 28-29.

- `ColorTest *colorTest` — вказівник на об'єкт класу `ColorTest`.
- `GridTest *gridTest` — вказівник на об'єкт класу `GridTest`.
- `ReadTest *readTest` — вказівник на об'єкт класу `ReadTest`.
- `CalibrationFocusTest *cf_test` — вказівник на об'єкт класу `CalibrationFocusTest`.
- `LanguageMode language_state_` — змінна, яка відповідає за поточний стан локалізації програми.

Також було створено наступний функціонал:

Налаштування графічного інтерфейсу відбувається в конструкторі класу `MainWindow` наступним чином:

1. Ініціалізуємо всі стандартні приховані поля графічного інтерфейсу класу `MainWindow` за допомогою, зчитування їх з файлу `mainwindow.ui` (стандартний файл з графічним інтерфейсом):

```
QMainWindow(parent);
ui(new Ui::MainWindow);
ui->setupUi(this);
```

після чого регулюємо вікно програми під розширення монітора користувача:

```
int width_window_size_ = ui->line->width() + 20;
int height_ = QGuiApplication::primaryScreen()->geometry().height();
double delta_height_ = window_height_size_.find(height_) !=
window_height_size_.end() ? window_height_size_[height_] : 1;
this->setFixedSize(QSize(width_window_size_, height_ / delta_height_));
```

завершаємо роботу конструктора, підключенням відповідних сигналів системної треї:

```
connect(sys_tray, &SysTray::showApp, this, &MainWindow::show);
connect(sys_tray, &SysTray::hideApp, this, &MainWindow::close);
connect(sys_tray, &SysTray::closeApp, this, &QCoreApplication::exit);
```

Функція `void MainWindow::SetUALocalization()` - встановлює українську локалізацію, якщо клієнт натиснув на меню “`Language/Ukrainian`” і поточний стан локалізації програми був рівний `LanguageMode::kEN`. Даний механізм реалізовується сигналом `triggered` при натисканні меню “`Language/Ukrainian`” і

викликом функції `void MainWindow::on_actionUkrainian_triggered()`, яка хендлить даний сигнал.

```
void MainWindow::on_actionUkrainian_triggered(){
    if (language_state_ == LanguageMode::kUA)
        return;
    SetUALocalization();
    language_state_ = LanguageMode::kUA;
}
```

Функція `void MainWindow::SetENLocalization()` - встановлює англійську локалізацію, якщо клієнт натиснув на меню “Language/English” і поточний стан локалізації програми був рівний `LanguageMode::kUA`. Даний механізм реалізовується сигналом `triggered` при натисканні меню “Language/ English” і викликом функції `void MainWindow::on_actionEnglish_triggered()`, яка хендлить даний сигнал.

Наявна функція `void MainWindow::SetPreview(QLabel *preview_label, const char *picture)` - дана функція реалізує попередній огляд тесту. Коли користувач наводиться на тест то програма емітить кастомний сигнал `hoverred`, який хендлить функція `bool MainWindow::eventFilter(QObject *obj, QEvent *event)`.

Функція `bool MainWindow::eventFilter(QObject *obj, QEvent *event)` - велика за обсягом функція, яка відфільтровує на який графічний об'єкт було наведено курсор мишко та встановлює підказку та попередній огляд даного тесту.

Функції `void MainWindow::addSelectedColorTests()`, `void MainWindow::addSelectedGridTests()`, `void MainWindow::addReadTests()` та `void MainWindow::addCalibrationTests()` перевіряють чи відповідний тест було вибрано, якщо тест вибрано вони добавляють його у відповідні контейнери:

- `std::vector<const char*> testsVector`
- `std::vector<uint8_t> testCodeVector`
- `std::vector<QColor> testColorVector`
- `std::vector<uint8_t> testFocusVector`

Функція `void MainWindow::createTestWindow()` - реалізовує вивід тестів на екран користувача. Спочатку перевіряється, яка сторінка з тестами вибрана. Після чого створюється відповідний об'єкт класу, який буде проводити відповідний тест. В нього передаються ідентифікатори тестів та допоміжні дані такі, як кольори тестів, кольори фонів, ідентифікатори калібрувальних та фокусувальних тестів. Після чого дана функція передає управління програми відповідному об'єкту тесту та виводить його на весь екран користувача. Після чого вже реалізується робота певного тесту.

```
void MainWindow::createTestWindow(){
    if (ui->stackedWidget->currentIndex() == 0) {
        cf_test = new CalibrationFocusTest(isAutoTest, auto_test_timeout_);
        cf_test->setTestCodeVector(getCodeVector());
        cf_test->setTestFocusVector(getFocusVector());
        cf_test->setUpTest();
        cf_test->showFullScreen();
    } else if (ui->stackedWidget->currentIndex() == 1) {
        gridTest = new GridTest(isAutoTest, auto_test_timeout_);
        gridTest->setTestCodeVector(getCodeVector());
        gridTest->setTestColorVector(getColorVector());
        gridTest->setFirstGrid(gridTest->getFirstColor());
        gridTest->showFullScreen();
    } else if (ui->stackedWidget->currentIndex() == 2) {
        colorTest = new ColorTest(isAutoTest, auto_test_timeout_);
        colorTest->setVectorValues(getVector());
        colorTest->setCodeVector(getCodeVector());
        colorTest->setColorVector(getColorVector());
        colorTest->showFullScreen();
    } else if (ui->stackedWidget->currentIndex() == 3) {
        readTest = new ReadTest(isAutoTest, auto_test_timeout_);
        readTest->setTestCodeVector(getCodeVector());
        readTest->setTestColorVector(getColorVector());
        readTest->setColor(readTest->getFirstColor());
        readTest->showFullScreen();
    }
}
```

Функція `void MainWindow::on_actionPreferences_triggered()` - виконується, коли користувач натисне на меню “File/Preferences”. Спочатку створюється об'єкт класу `Preferences`, який в своєму конструкторі створює графічний інтерфейс вікна налаштувань. В конструктор передається змінна `auto_test_timeout_`, яка відповідає за час перемикання тестів при авто тестуванні, змінна `save_settings_`, яка відповідає за збереження налаштувань програми та тип локалізації.

```
void MainWindow::on_actionPreferences_triggered(){
```

```

        if (preference == nullptr) {
            preference = new Preference(auto_test_timeout_, save_settings_,
                                         language_state_ == LanguageMode::kEN ? true : false);
            connect(preference, &Preference::preferenceClosed, this,
                    &MainWindow::handlePreferenceClosed);
        }
        preference->show();
        preference->exec();
        if (preference != nullptr) {
            delete preference;
            preference = nullptr;
        }
    }
}

```

Функція `void MainWindow::on_actionAbout_triggered()` - виконується, коли користувач натисне на меню “Help/About”. Спочатку створюється об’єкт класу `About`, який в своєму конструкторі створює графічний інтерфейс вікна “Про програму”. В конструктор даного класу передається поточний стан локалізації.

```

void MainWindow::on_actionAbout_triggered(){
    if (about == nullptr) {
        about = new About(language_state_ == LanguageMode::kEN ? true : false);
    }
    about->show();
    about->exec();
    if (about != nullptr) {
        delete about;
        about = nullptr;
    }
}

```

Функція `void MainWindow::checkSelectedTests()` - дана функція викликає функції `addCalibrationTests()`, `addSelectedGridTests()`, `addSelectedColorTests()` та `addReadTests()`. В залежності на якій ми сторінці з тестами знаходимося після відроблення однієї з цих функцій викликається функція `createTestWindow()`, якщо хоч один тест було вибрано.

```

void MainWindow::checkSelectedTests(){
    if (ui->stackedWidget->currentIndex() == 0)
        addCalibrationTests();
    if (ui->stackedWidget->currentIndex() == 1)
        addSelectedGridTests();
    if (ui->stackedWidget->currentIndex() == 2)
        addSelectedColorTests();
    if (ui->stackedWidget->currentIndex() == 3)
        addReadTests();
    if (!testCodeVector.empty())
        createTestWindow();
}

```

Також наявні функції void on_RunTests_clicked() та void on_AutoRunTests_clicked(); дані функції викликають у собі функцію checkSelectedTests(), але функція on_AutoRunTests_clicked() спершу ставить стан змінної isAutoTest в true, що означає що програма повинна запустити тести в автоматичному режимі(тести будуть перемикатися самі через n(auto_test_timeout_) секунд).

3.5. Створення та опис програмного проекту

Розроблена утиліта під назвою Monitor Testing є структурованим програмним рішенням, де реалізація графічного інтерфейсу, частини, пов'язані з тестуванням, системною треєю та локалізацією відокремлені, як окремі модулі(описані в розділі 2.1.2), та групуються відповідно до директорій та класів.

Загалом проект містить:

- 9 файлів заголовків (розширення .h)
- 7 файлів з реалізацією програми (файли з розширенням .cpp)
- 1 ресурсний файл (.qrc), який містить перелік ресурсів програм, як файли зображень та значків.
- 1 файл (.ui) для конфігурації інтерфейсу програми

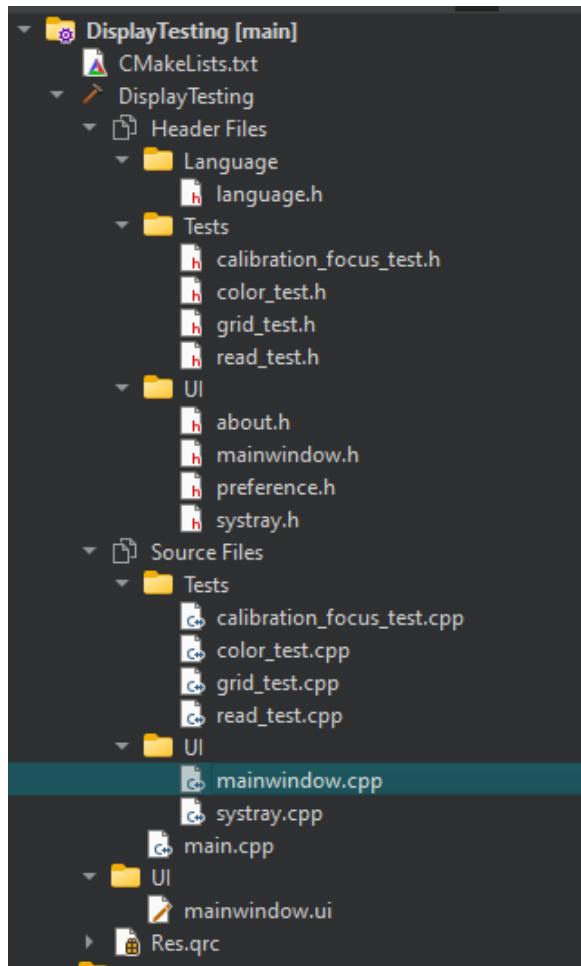


Рис.13.Структура програми в QtCreator

Проєкт містить такі директорії:

1. Language - містить language.h, файлу заголовку в якому зберігаються локалізовані тексти програми

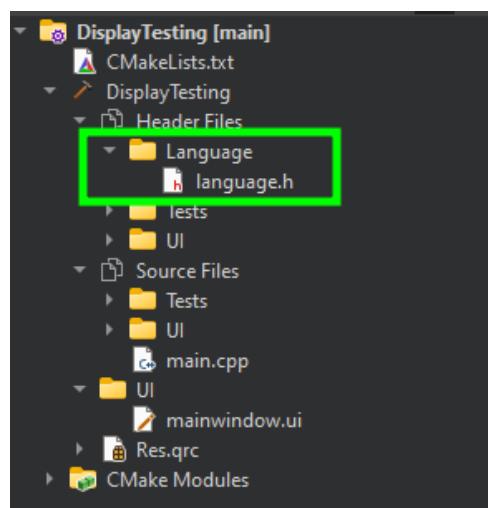
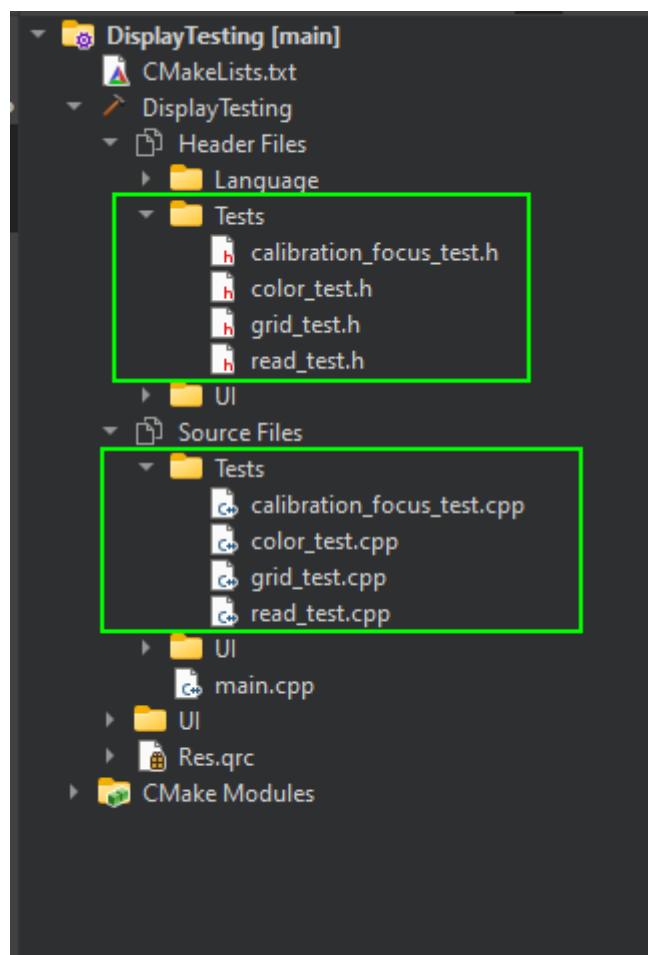


Рис. 14. Директорія Language

2. Tests - містить:

- calibration_focus_test(.h/.cpp) - в даних файлах реалізовано клас, який виконує тести для калібування та фокусування монітору комп'ютера.
- color_test(.h/.cpp) - в даних файлах реалізовано клас, який виконує тести для перевірки кольорової палітри монітора комп'ютера.
- grid_test(.h/.cpp) - в даних файлах реалізовано клас, який виконує сіткові тести
- read_test(.h/.cpp) - в даних файлах реалізовано клас, який виконує перевірку монітора на відображення та читабільність тексту.



Rис. 15. Директорія Tests

3. UI - містить:

- about.h - файл в якому реалізовано меню “Help/About”, яке містить інформацію про програму

- preference.h - файл в якому реалізовано меню “File/Preference”, яке містить інформацію про налаштування програми
- systray(.h/.cpp) - файли в яких реалізовано функціонал системної треї.
- mainwindow(.h/.cpp) - файли в яких реалізовано функціонал графічного інтерфейсу програми.

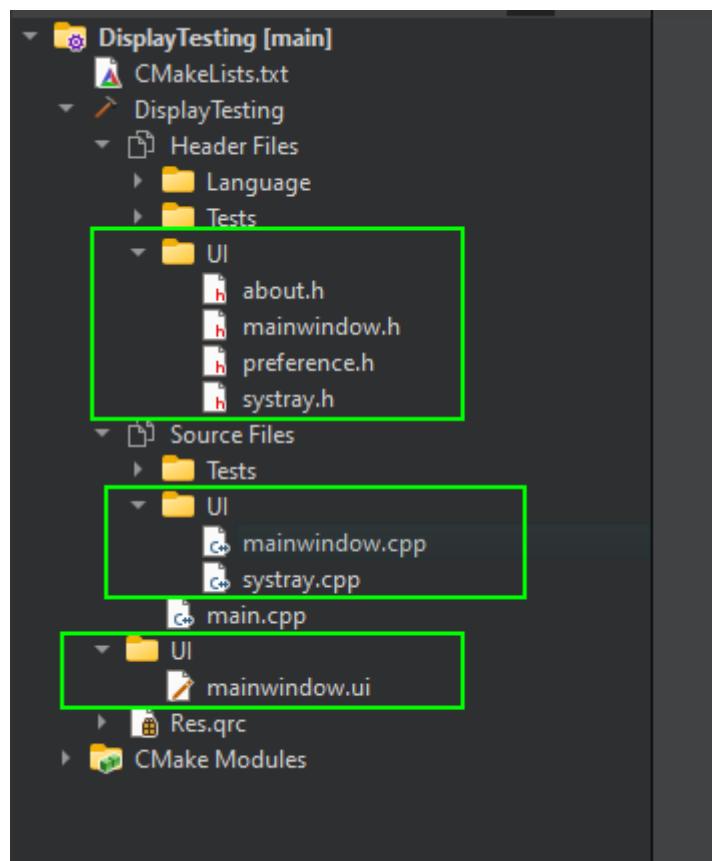


Рис. 16. Директорія UI

4. pictures - папка в якій зберігаються картинки та іконки програми, вони добавлені в Res.qrc файл
5. main.cpp - файл в якому реалізована функція main, яка створює в собі обєкт класу MainWindow та виконує роботу програми.

4. Тестування програмного забезпечення та інструкції користувачеві

4.1. Тестування та опис інтерфейсу програми

Для зручного користування додатком розроблено графічний інтерфейс за допомогою фреймворку Qt та мовою програмування C++.

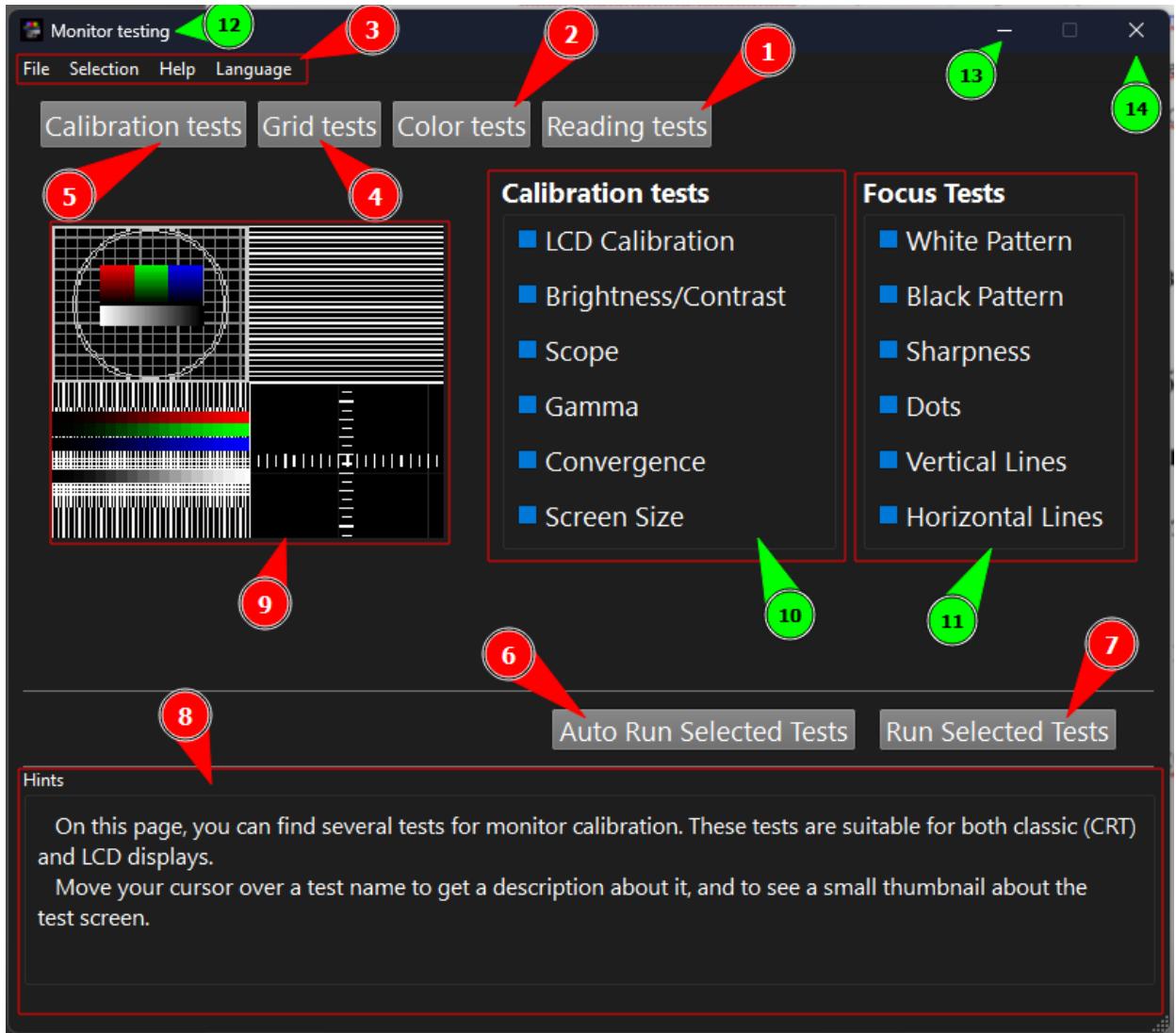


Рис.17. Інтерфейс головного вікна розробленої програми

Розглянемо кожен елемент головного вікна інтерфейсу (на рис.17. позначені цифри відповідають нумерації, яка подана нижче):

1. Кнопка “Reading tests” — переходить на вікно з вибором тестів на перевірку читабельності тексту та його відображення на екрані(рис. 18).

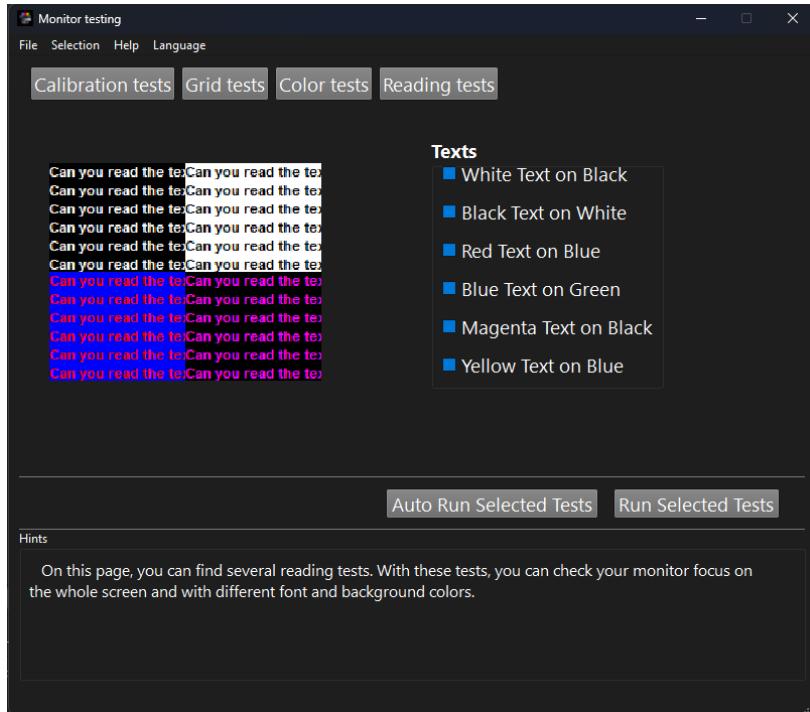


Рис. 18. Вигляд вікна Reading tests

2. Кнопка “Color tests” — переходить на вікно з вибором тестів для відрегулювання правильної передачі кольорової гами(рис. 19).

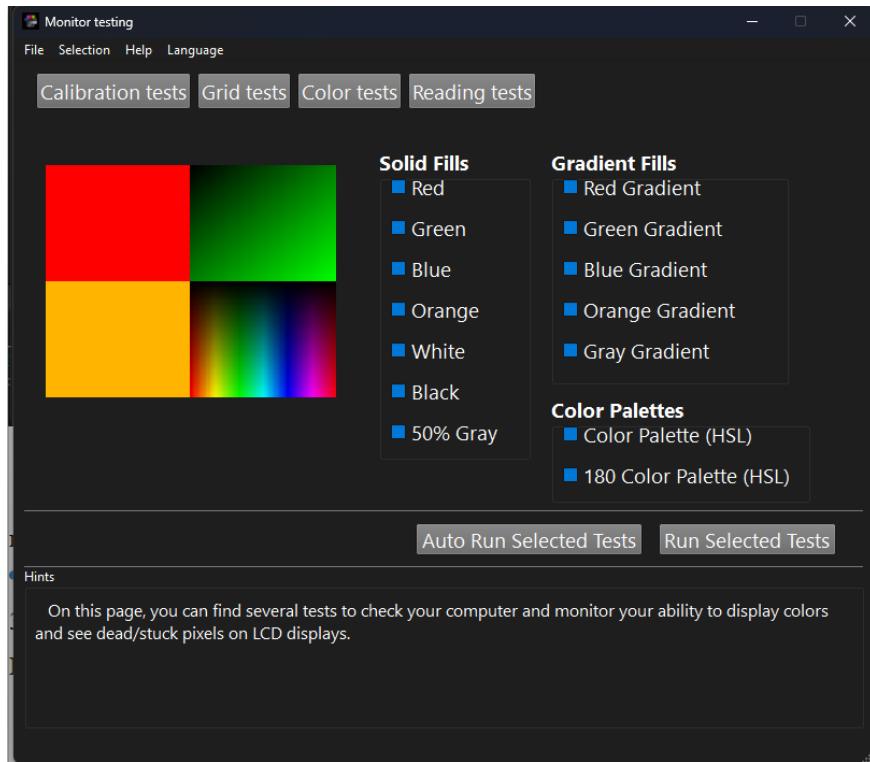


Рис. 19. Вигляд вікна Color tests

3. Головне меню програми в якому знаходяться підменю File, Selection, Help та Language.

3.1. Підменю File містить такі поля, як Exit та Preferences(рис. 20):

- Exit – вийти з програми
- Preferences – відкрити уподобання програми.

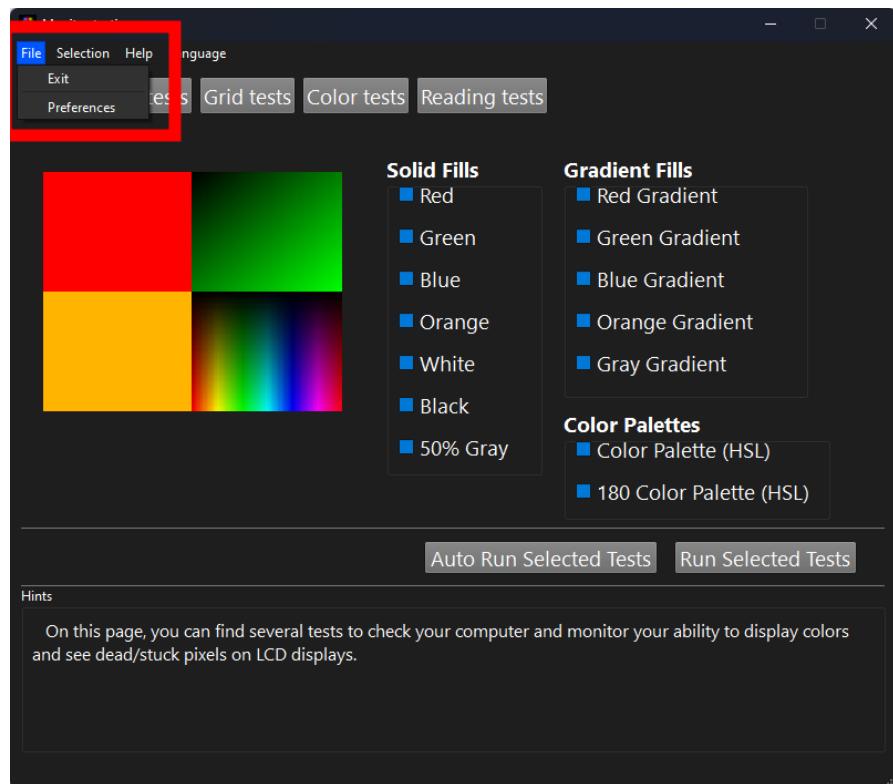


Рис. 20. Вигляд меню File

3.2. Підменю Selection містить такі поля, як Select all, Clear all та Tests for LCD monitors(рис. 21):

- Select all – вибрати всі тести
- Clear all – очистити вибір всіх тестів
- Tests for LCD monitors – вибрати тільки ті тести, які підходять для Рідкокристалічних моніторів

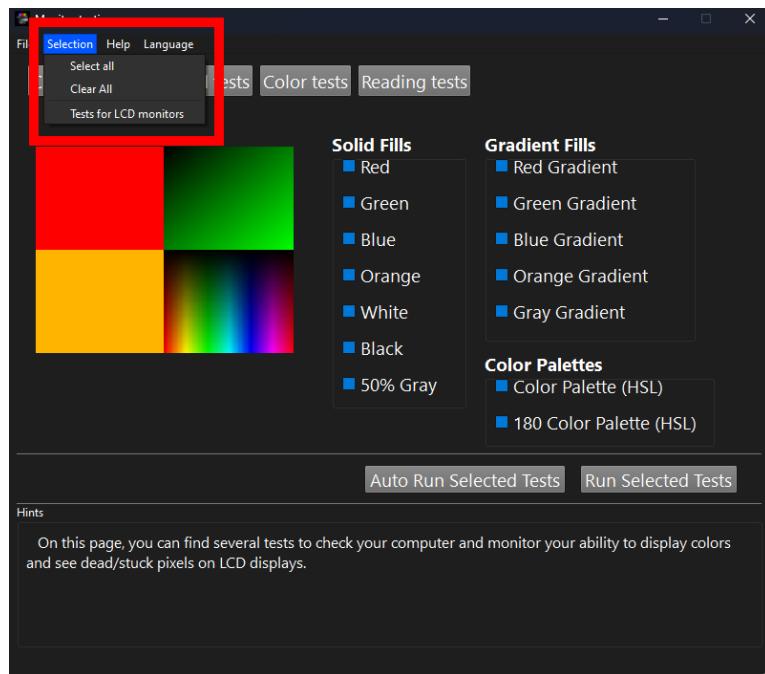


Рис. 21. Вигляд меню Selection

3.3. Підменю Help містить такі поля, як About — інформація про програму(рис. 22).

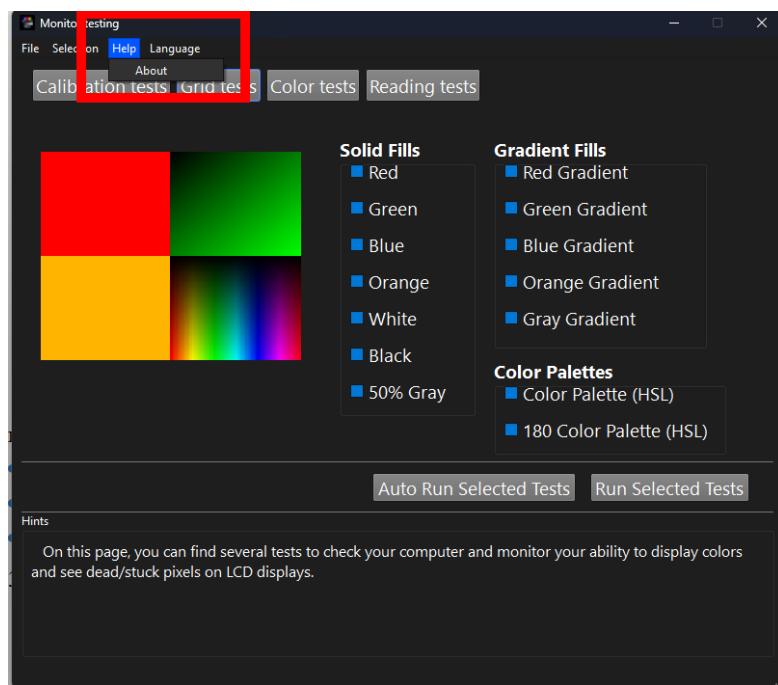


Рис. 22. Вигляд меню Help

3.4. Підменю Language містить такі поля, як Ukrainian та English(рис. 23).

- Ukrainian – відповідає за українську локалізацію

- English – відповідає за англійську локалізацію

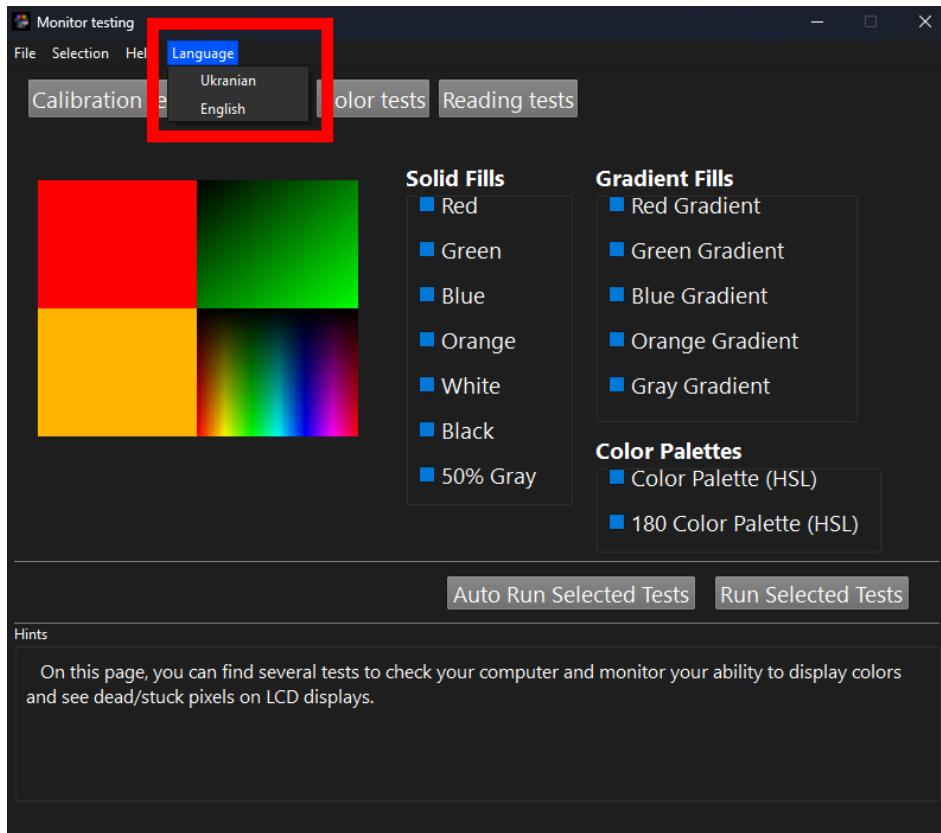


Рис. 23. Вигляд меню Language

4. Кнопка “Grid tests” — переходить на вікно з вибором сіткових тестів тести, які допоможуть відрегулювати розширення та чіткість екрану монітора(рис. 24).

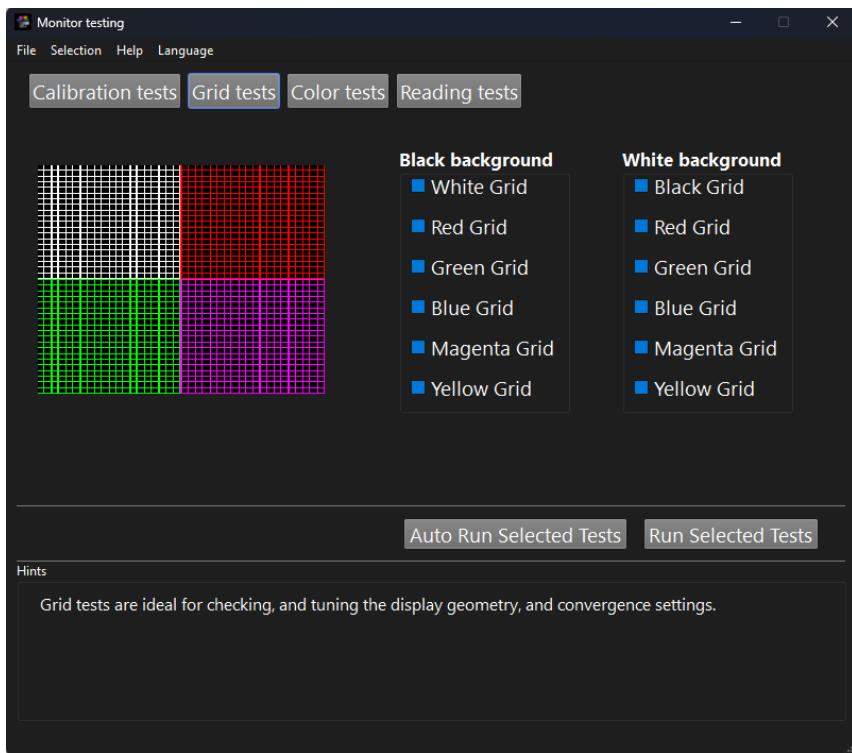


Рис. 24. Вигляд вікна Grid tests

5. Кнопка “Calibaration tests” — переходить на вікно з вибором тестів для калібрування чіткості зображення монітора(рис. 17).
6. Кнопка “Auto Run Selected Tests” — запускає обрані тести в автоматичному режимі.(автоматичний режим означає те, що тести будуть перемикатися самі через певний проміжок часу який вибере користувач у підменю Preferences по дефолту затримка одного тесту рівна 4-м секундам).
7. Кнопка “Run Selected Tests” — запускає обрані тести в простому режимі для перемикання тестів потрібно використовувати кнопки з зображенням стрілок на клавіатурі вашого пристрою.
8. Поле “Hints” — дане поле надає підказки користувачу що потрібно робити або що робить даний тест воно міняється відповідно до того на що вказує курсор користувача.
9. Поле “Preview image” — дане поле відображає попередній огляд тесту який хоче вибрати користувач Дане поле міняється відповідно до того на що вказує курсор користувача.

10. Поле для вибору Calibaration tests. Тести можна вибрати натискаючи на них.

11. Поле для вибору Focus tests. Тести можна вибрати натискаючи на них.

12. Назва програми.

13. Приховати додаток до таск – бару

14. Закрити програму, але після цього вона переходить в системну терю, і щоб повністю закрити програму потрібно закрити її з контекстного меню системної треї.

Розглянемо згортання додатку в системну трею:

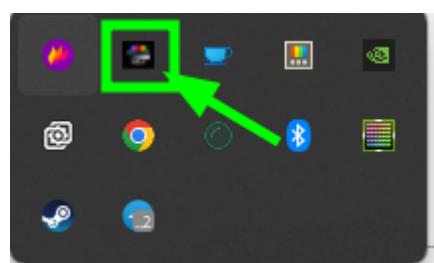


Рис.25. Іконка утиліти в системній треї

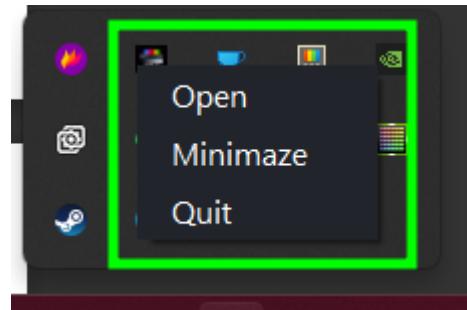


Рис.26. Контекстне меню при натисканні правої кнопки комп'ютерної миші на іконку в системній треї

4.2. Інструкції користувачеві

Для інсталяції даної утиліти користувачеві потрібно перейти на GitHub сторінку з проєктом – <https://github.com/bu7ch3RiP/KursovaZSP>, після чого потрібно виконати дані пункти:

1. Зайти на вкладку “Releases”

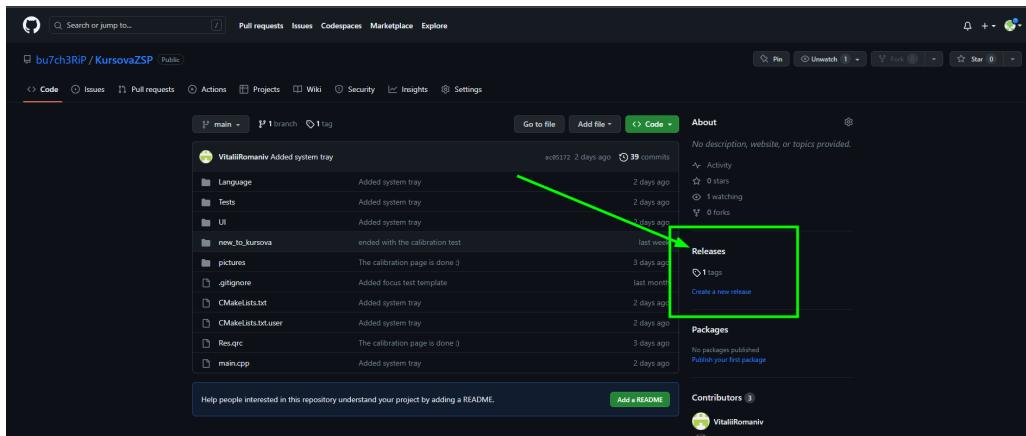


Рис. 27. Github сторінка проекту

2. В відкритому вікні на вкладці Assets можна завантажити інсталятор додатку та портабл версію

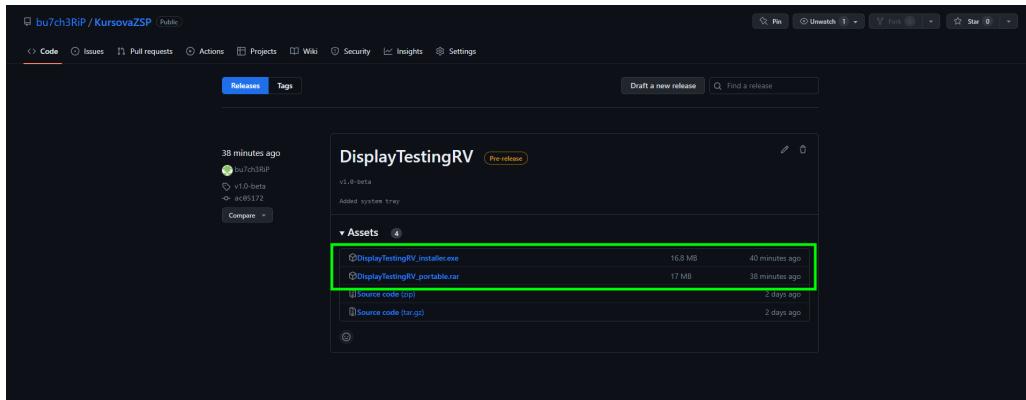


Рис.28. Завантаження проекту

3. Якщо завантажили інсталятор то потрібно слідувати крокам, які надасть інсталятор.

4. Якщо завантажили портабл версію потрібно розархівувати її та зайти в папку PortableVersion та в ній відкрити DisplayTesting.exe

generic	97 984	40 747	Папка файлів	07.06.2023 22:54
iconengines	69 824	27 954	Папка файлів	07.06.2023 22:54
imageformats	611 584	216 181	Папка файлів	07.06.2023 22:54
networkinformation	88 256	45 898	Папка файлів	07.06.2023 22:54
platforms	1 031 872	363 915	Папка файлів	07.06.2023 22:54
styles	162 496	68 847	Папка файлів	07.06.2023 22:54
tls	679 488	231 437	Папка файлів	07.06.2023 22:54
translations	0	0	Папка файлів	07.06.2023 22:54
D3Dcompiler_47.dll	4 173 928	1 464 660	Розширення застосунку	11.03.2014 12:54 36D3FED2
DisplayTesting.exe	1 280 543	573 551	Застосунок	07.06.2023 22:53 C8AEE9F2
libgcc_s_seh-1.dll	107 187	42 924	Розширення застосунку	27.04.2023 9:27 AC189315
libstdc++-6.dll	2 024 354	514 062	Розширення застосунку	27.04.2023 9:27 9E3B3BD8
libwinpthread-1.dll	60 356	24 203	Розширення застосунку	09.04.2023 19:51 90E82859
opengl32sw.dll	20 633 208	5 588 972	Розширення застосунку	04.06.2020 10:50 92F954CD
Qt6Core.dll	6 510 776	2 271 708	Розширення застосунку	25.03.2023 8:48 6B14208D
Qt6Gui.dll	9 728 176	3 439 770	Розширення застосунку	25.03.2023 8:48 969BCE42
Qt6Network.dll	1 660 592	565 213	Розширення застосунку	25.03.2023 8:48 8CCD25CA
Qt6Svg.dll	359 600	126 552	Розширення застосунку	25.03.2023 21:35 11E3D661
Qt6Widgets.dll	6 640 304	2 251 928	Розширення застосунку	25.03.2023 8:48 DE410539

Рис.29 Вміст папки PortableVersion

5. Якщо завантажили Source code.zip потрібно розархіювати даний архів та зайти в папку KursovaZSP-1.0-beta, після чого потрібно відкрити CMakeLists.txt(рис. 30) за допомогою QtCreator(рис. 31). Далі у нас відкриється QtCreator і нам потрібно нажати на кнопку Run яка знаходиться в лівому нижньому меню

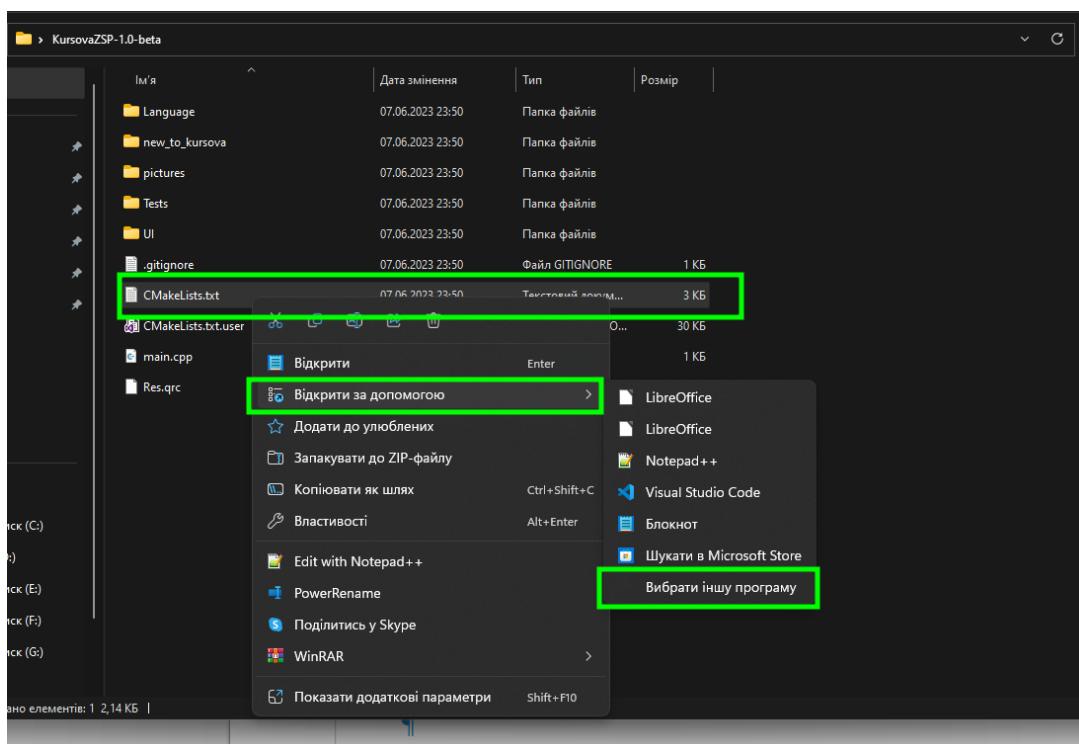


Рис. 30. Місце знаходження CMakeLists.txt

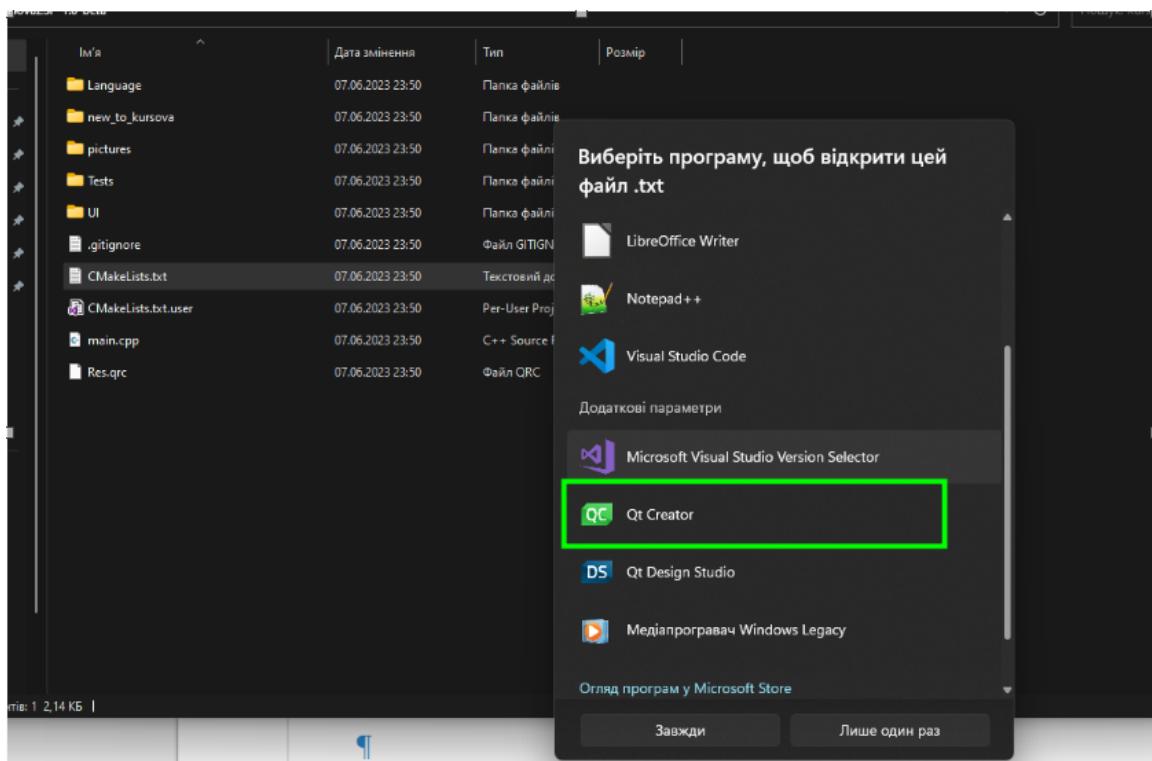


Рис. 31. Відкриття CMakeLists.txt за допомогою Qt Creator

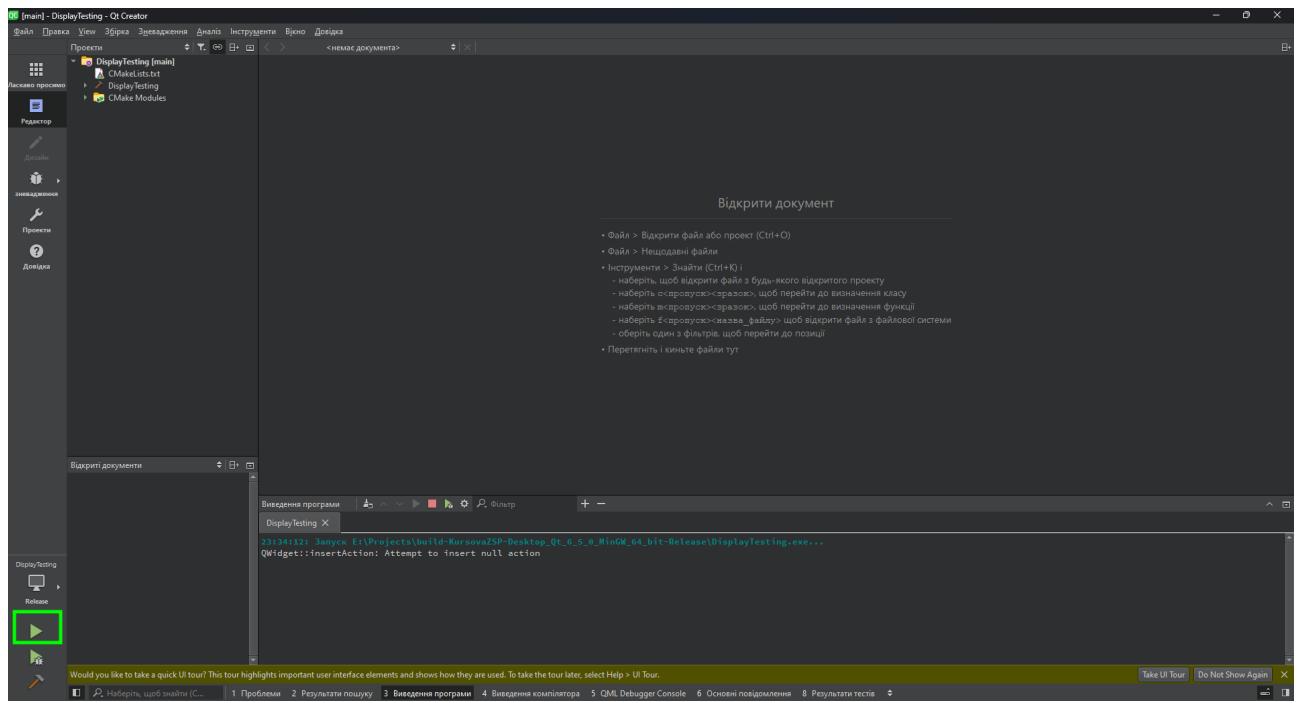


Рис. 32. Запуск програми

Висновок

Під час виконання даного курсового проєкту було розроблено програму для тестування моніторів комп'ютера та закріплено знання про тестування моніторів комп'ютерів. Програма, реалізована з використанням мови програмування C++ та фреймворку Qt.

В результаті чого було виконано таку роботу:

1. Проведено аналіз і огляд існуючих розробок і методів у галузі тестування моніторів комп'ютера. Описано різні підходи, програмні засоби та технології, що використовуються для тестування комп'ютерів.

2. Проведено проєктування програми та вибір структур даних для її правильної роботи. Проєктувалася програма за принципом Model-View Programming. Було використано об'єктно-орієнтовану парадигму програмування та вбудований функціонал фреймворку Qt, а саме сигнали та слоти.

3. Розроблено прототип програми AIDA64, тобто її вкладку “Tools/Monitor diagnosing” із меншим функціоналом.

4. Під час тестування була підтверджена працездатність системи, яка ефективно виконувала свої функції і забезпечувала точне тестування монітора комп'ютера.

Загалом, розробка цієї утиліти тестування моніторів комп'ютера виявилася успішною, демонструючи важливі аспекти Model-View Programming, об'єктно-орієнтованого підходу та ефективного використання механізму сигналів і слотів. Результати роботи відповідають поставленим цілям та відображають успішну реалізацію програми.

Проте дану програму потрібно ще оптимізувати по використання ресурсів процесора, добавити більшу функціональність та реалізувати збереження результатів тестування монітора комп'ютера для користувачів.

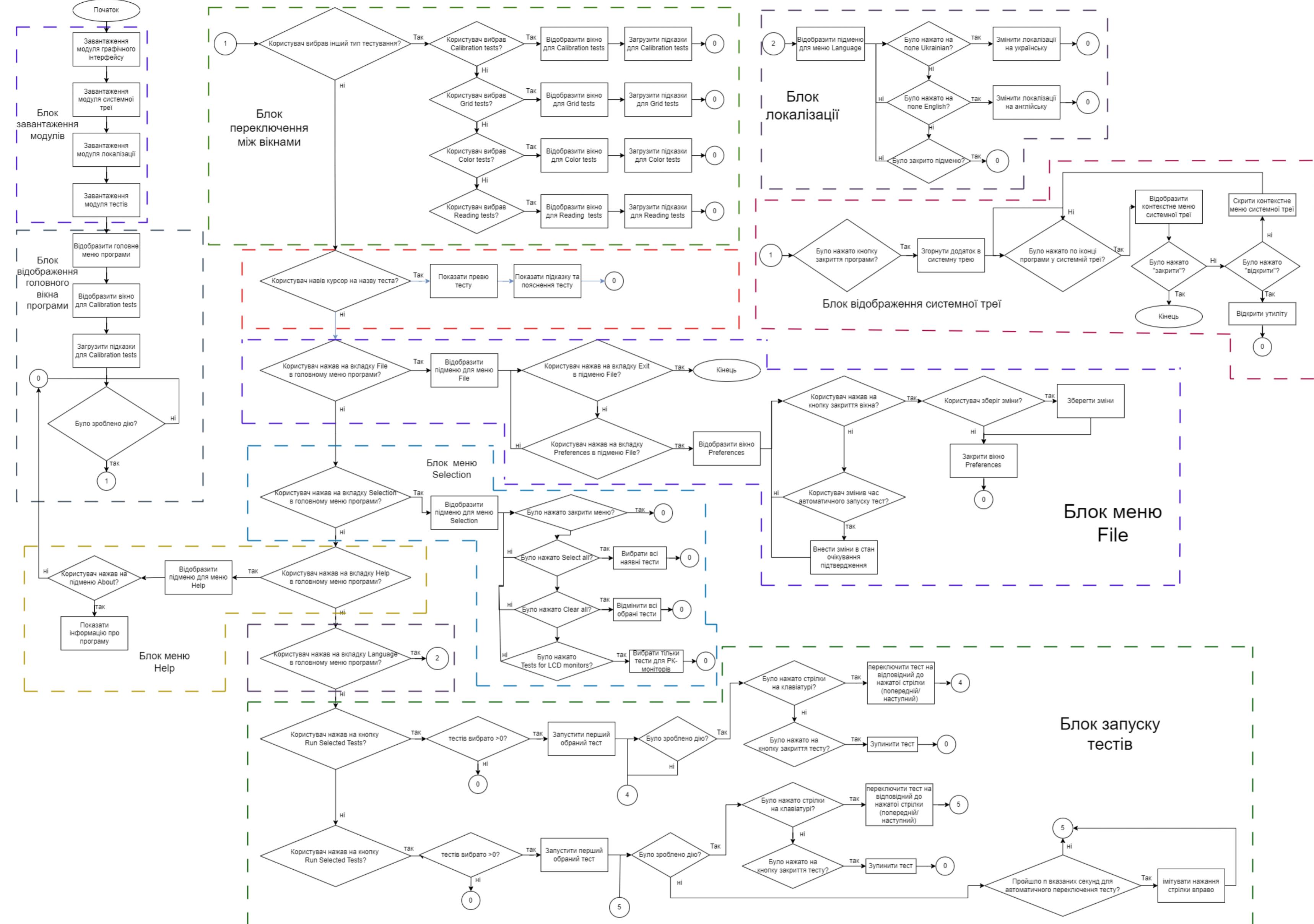
Отже, розробка утиліти тестування монітора комп'ютера дозволила набути практичних навичок у проектуванні програмних систем та поглибити розуміння процесу розробки програмного забезпечення.

Список використаної літератури

1. AIDA64 - AIDA64 Extreme. AIDA64 - Тому що навіть IT-фахівцям потрібна підтримка. URL: <https://www.aida64.com/>
2. DisplayCAL - DisplayCAL Display Calibration and Characterization powered by ArgyllCMS. URL: <https://displaycal.net/>
3. Model–view–controller - Wikipedia
URL: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
4. OOP – Wikipedia
URL: https://en.wikipedia.org/wiki/Object-oriented_programming
5. OOP - "Object-Oriented Programming in C++" by Robert Lafore, 2002 year, 800 East 96th St., Indianapolis, Indiana 46240 USA
URL: <https://docs.google.com/file/d/0B21HoBq6u9TsUHhqS3JIUmFuamc/view?resourcekey=0-MYlet9RIjEukd6CvLEHUbw>
6. Signals & Slots | Qt Core 6.5.0. Qt Documentation | Home.
URL: <https://doc.qt.io/qt-6/signalsandslots.html>
7. QLabel class.
URL: <https://doc.qt.io/qt-5/qlabel.html>
8. QPixmap class.
URL: <https://doc.qt.io/qt-6/qpixmap.html>
9. Qt Creator.
URL: <https://www.qt.io/product/development-tools>

Додатки

Додаток А. Утиліта Monitor testing. Схема алгоритму



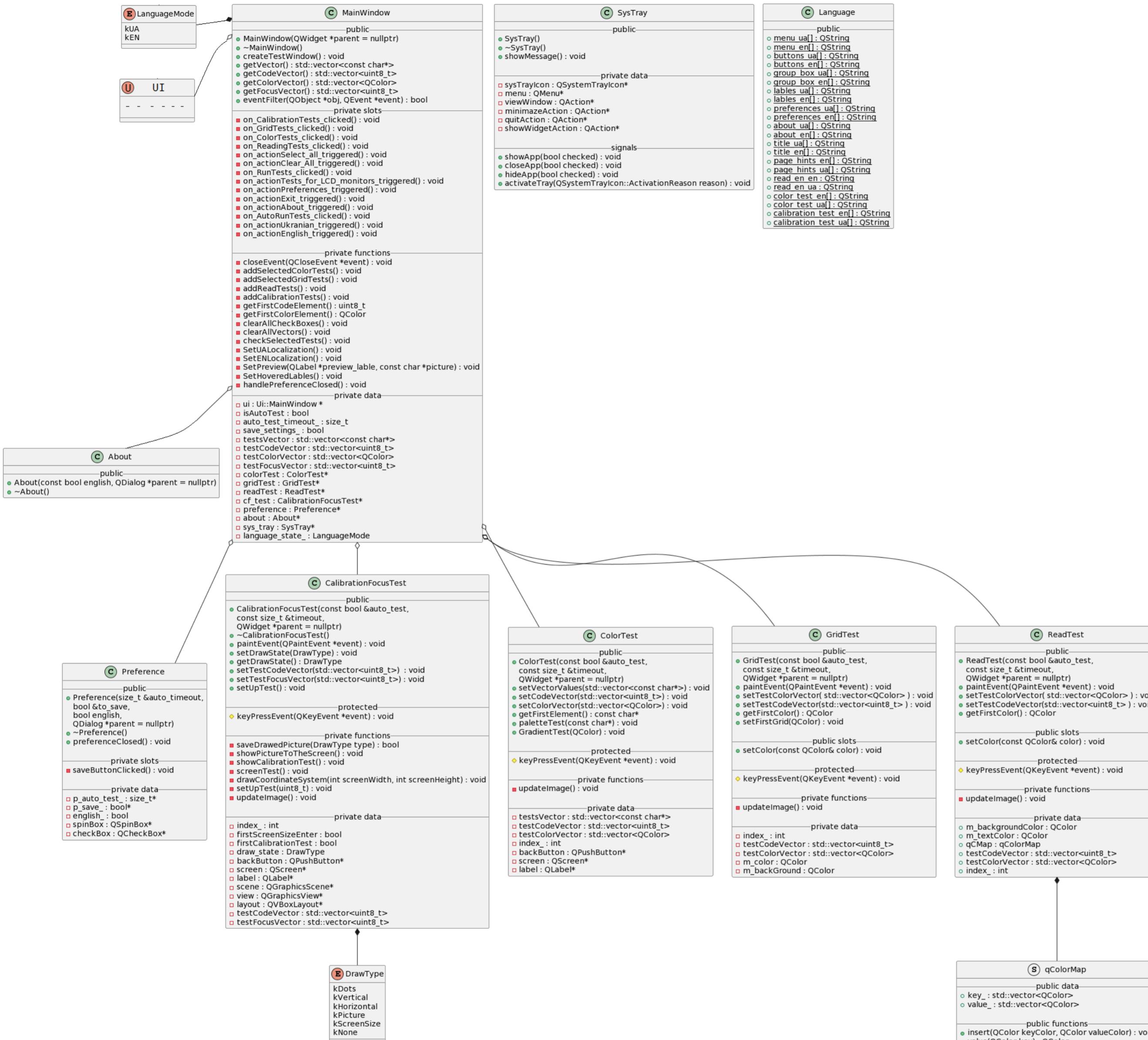
Міністерство освіти і науки України				
Зм.	Арк.	№ докум.	Підпис	Дат.
Виконав		Романів В. А.		
Керівник		Мархівка В. С.		
Консульт.				
Консульт.				
Зав. каф.		Мельник А. О.		
Реценз.				

КУРСОВИЙ ПРОЄКТ

Утиліта тестування моніторів комп'ютера з OC Windows

Утиліта Monitor testing Схема алгоритму	Літера	Маса	Масштаб
	у		
	Аркуш	Аркуш 1	
НУ «ЛП», ІКТА, каф. ЕОМ, гр КІ-34			

Додаток Б. Утиліта Monitor testing. Діаграма класів.



Міністерство освіти і науки України	КУРСОВИЙ ПРОЕКТ		
Утиліта тестування моніторів комп’ютера з OC Windows			
Зм. Арк.	№ докум.	Підпис	Дата
Виконав.	Романів В. А.		
Керівник	Мархівка В. С.		
Консульт.			
Консулт.			
Зав. каф.	Мельник А. О.		
Реценз.			
Аркуш			
Аркушів 1			
НУ «ЛП», ІКТА, каф. ЕОМ, гр КІ-34			

Додаток В. Утиліта Monitor testing. Лістинг коду

main.cpp

```
#include <QApplication>
#include <QLabel>
#include "UI/mainwindow.h"

#include <QFile>
#include <QString>
#include <QStyleFactory>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    a.setWindowIcon(QIcon(":/color/palette/pictures/MonitorTesting.ico"));
    QApplication::setStyle(QStyleFactory::create("Fusion"));
    MainWindow w;
    w.show();
    return a.exec();
}
```

```
mainwindow.h
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

```
#include <QColor>
#include <QEvent>
#include <QLabel>
#include <QMainWindow>
#include <QMouseEvent>
#include "../Tests/calibration_focus_test.h"
#include "../Tests/color_test.h"
#include "../Tests/grid_test.h"
#include "../Tests/read_test.h"
#include "systray.h"
#include <vector>
```

```
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
```

```
class Preference;
class About;
```

```
enum class LanguageMode { kUA, kEN };
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
```

```
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

```
    void createTestWindow();
    std::vector<const char*> getVector();
    std::vector<uint8_t> getCodeVector();
    std::vector<QColor> getColorVector();
    std::vector<uint8_t> getFocusVector();
```

```
    bool eventFilter(QObject *obj, QEvent *event);
```

```
protected:
```

```
private slots:
```

```
    void on_CalibrationTests_clicked();
    void on_GridTests_clicked();
    void on_ColorTests_clicked();
    void on_ReadingTests_clicked();
    void on_actionSelect_all_triggered();
```

```

void on_actionClear_All_triggered();
void on_RunTests_clicked();
void on_actionTests_for_LCD_monitors_triggered();
void on_actionPreferences_triggered();
void on_actionExit_triggered();
void handlePreferenceClosed();
void on_actionAbout_triggered();
void on_AutoRunTests_clicked();

void on_actionUkrainian_triggered();
void on_actionEnglish_triggered();

private:
    Ui::MainWindow *ui;
    bool isAutoTest{};
    size_t auto_test_timeout{};
    bool save_settings{};
    std::vector<const char*> testsVector{};
    std::vector<uint8_t> testCodeVector{};
    std::vector<QColor> testColorVector{};
    std::vector<uint8_t> testFocusVector{};
    ColorTest *colorTest = nullptr;
    GridTest *gridTest = nullptr;
    ReadTest *readTest = nullptr;
    CalibrationFocusTest *cf_test = nullptr;
    Preference *preference = nullptr;
    About *about = nullptr;
    SysTray *sys_tray = nullptr;
    LanguageMode language_state;

private:
    void addSelectedColorTests();
    void addSelectedGridTests();
    void addReadTests();
    void addCalibrationTests();
    uint8_t getFirstCodeElement();
    QColor getFirstColorElement();
    void clearAllCheckBoxes();
    void clearAllVectors();
    void checkSelectedTests();
    void SetUALocalization();
    void SetENLocalization();

    void SetPreview(QLabel *preview_lable, const char *picture);
    void SetHoveredLables();

private:
    void closeEvent(QCloseEvent *event) override;
};

#endif // MAINWINDOW_H

mainwindow.cpp

#include "mainwindow.h"
#include "../Language/language.h"
#include "./ui_mainwindow.h"
#include "about.h"
#include "preference.h"

#include <QAction>
#include <QCheckBox>
#include <QCoreApplication>
#include <QKeyEvent>
#include <QList>
#include <QMessageBox>
#include <QScreen>
#include <QWidget>
#include <unordered_map>

```

```

std::unordered_map<int, double> window_height_size_{ {1080, 1.6},
    {1050, 1.6},
    {1024, 1.4},
    {960, 1.3},
    {900, 1.4},
    {800, 1.2},
    {768, 1.3},
    {720, 1.15}};

MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
, ui(new Ui::MainWindow)
, auto_test_timeout_{4}
, save_settings_{true}
, isAutoTest{false}
, language_state_{LanguageMode::kEN}
{
    ui->setupUi(this);
    sys_tray = new SysTray();
    setWindowTitle("Monitor testing");
    int width_window_size_ = ui->line->width() + 20;
    int height_ = QGuiApplication::primaryScreen()->geometry().height();
    double delta_height_ = window_height_size_.find(height_) != window_height_size_.end()
        ? window_height_size_[height_]
        : 1;
    this->setFixedSize(QSize(width_window_size_, height_ / delta_height_));
    SetHoveredLables();
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0] : page_hints_ua[0]);
    SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/CalibrationTestPreview.png"));
    connect(sys_tray, &SysTray::showApp, this, &MainWindow::show);
    connect(sys_tray, &SysTray::hideApp, this, &MainWindow::close);
    connect(sys_tray, &SysTray::closeApp, this, &QCoreApplication::exit);
}

MainWindow::~MainWindow()
{
    delete ui;
    delete sys_tray;
}

void MainWindow::SetUALocalization()
{
    setWindowTitle(title_ua[0]);

    //bar menu
    ui->menuFile->setTitle(menu_ua[0]);
    ui->menuSelection->setTitle(menu_ua[1]);
    ui->menuHelp->setTitle(menu_ua[2]);
    ui->actionExit->setText(menu_ua[3]);
    ui->actionPreferences->setText(menu_ua[4]);
    ui->actionSelect_all->setText(menu_ua[5]);
    ui->actionClear_All->setText(menu_ua[6]);
    ui->actionTests_for_LCD_monitors->setText(menu_ua[7]);
    ui->actionAbout->setText(menu_ua[8]);
    ui->menuLanguage->setTitle(menu_ua[9]);
    ui->actionUkrainian->setText(menu_ua[10]);
    ui->actionEnglish->setText(menu_ua[11]);

    //buttons
    ui->RunTests->setText(buttons_ua[0]);
    ui->RunTests->setGeometry(QRect(535, 430, 240, 30));

    ui->AutoRunTests->setText(buttons_ua[1]);
    ui->AutoRunTests->setGeometry(QRect(250, 430, 280, 30));

    ui->CalibrationTests->setText(buttons_ua[2]);
    ui->CalibrationTests->setGeometry(QRect(21, 11, 185, 34));

    ui->GridTests->setText(buttons_ua[3]);
}

```

```

ui->GridTests->setGeometry(QRect(212, 11, 120, 34));
ui->ColorTests->setText(buttons_ua[4]);
ui->ColorTests->setGeometry(QRect(338, 11, 160, 34));

ui->ReadingTests->setText(buttons_ua[5]);
ui->ReadingTests->setGeometry(QRect(504, 11, 200, 34));

//group boxes
ui->groupBox->setTitle(group_box_ua[0]);
ui->groupBox_2->setTitle(group_box_ua[1]);
ui->groupBox_3->setTitle(group_box_ua[2]);
ui->groupBox_4->setTitle(group_box_ua[3]);
ui->groupBox_5->setTitle(group_box_ua[4]);
ui->groupBox_6->setTitle(group_box_ua[5]);
ui->groupBox_7->setTitle(group_box_ua[6]);
ui->groupBox_8->setTitle(group_box_ua[7]);

//labels
ui->LCDCalibration->setText(lables_ua[0]);
ui->Brightness->setText(lables_ua[1]);
ui->Scope->setText(lables_ua[2]);
ui->Gamma->setText(lables_ua[3]);
ui->Convergence->setText(lables_ua[4]);
ui->ScreenSize->setText(lables_ua[5]);
ui->WhitePattern->setText(lables_ua[6]);
ui->BlackPattern->setText(lables_ua[7]);
ui->Sharpness->setText(lables_ua[8]);
ui->Dots->setText(lables_ua[9]);
ui->VerticalLines->setText(lables_ua[10]);
ui->HorisontalLines->setText(lables_ua[11]);
ui->WhiteGrid->setText(lables_ua[12]);
ui->RedGrid->setText(lables_ua[13]);
ui->GreenGrid->setText(lables_ua[14]);
ui->BlueGrid->setText(lables_ua[15]);
ui->MagentaGrid->setText(lables_ua[16]);
ui->YellowGrid->setText(lables_ua[17]);
ui->BlackGrid->setText(lables_ua[18]);
ui->RedGridBlack->setText(lables_ua[19]);
ui->GreenGridBlack->setText(lables_ua[20]);
ui->BlueGridBlack->setText(lables_ua[21]);
ui->MagentaGridBlack->setText(lables_ua[22]);
ui->YellowGridBlack->setText(lables_ua[23]);
ui->Red->setText(lables_ua[24]);
ui->Green->setText(lables_ua[25]);
ui->Blue->setText(lables_ua[26]);
ui->Orange->setText(lables_ua[27]);
ui->White->setText(lables_ua[28]);
ui->Black->setText(lables_ua[29]);
ui->Gray50->setText(lables_ua[30]);
ui->RedGradient->setText(lables_ua[31]);
ui->GreenGradient->setText(lables_ua[32]);
ui->BlueGradient->setText(lables_ua[33]);
ui->OrangeGradient->setText(lables_ua[34]);
ui->GrayGradient->setText(lables_ua[35]);
ui->ColorPalette->setText(lables_ua[36]);
ui->ColorPalette180->setText(lables_ua[37]);
ui->WhiteText->setText(lables_ua[38]);
ui->BlackText->setText(lables_ua[39]);
ui->RedText->setText(lables_ua[40]);
ui->BlueText->setText(lables_ua[41]);
ui->MagentaText->setText(lables_ua[42]);
ui->YellowText->setText(lables_ua[43]);

ui->groupBox_8->setGeometry(409, 30, 290, 251);

QFont lable_font_("Segoe UI", 12);
ui->LCDCalibration->setFont(lable_font_);
ui->Brightness->setFont(lable_font_);
ui->Scope->setFont(lable_font_);

```

```

ui->Gamma->setFont(lable_font_);
ui->Convergence->setFont(lable_font_);
ui->ScreenSize->setFont(lable_font_);
ui->WhitePattern->setFont(lable_font_);
ui->BlackPattern->setFont(lable_font_);
ui->Sharpness->setFont(lable_font_);
ui->Dots->setFont(lable_font_);
ui->VerticalLines->setFont(lable_font_);
ui->HorisontalLines->setFont(lable_font_);
ui->WhiteGrid->setFont(lable_font_);
ui->RedGrid->setFont(lable_font_);
ui->GreenGrid->setFont(lable_font_);
ui->BlueGrid->setFont(lable_font_);
ui->MagentaGrid->setFont(lable_font_);
ui->YellowGrid->setFont(lable_font_);
ui->BlackGrid->setFont(lable_font_);
ui->RedGridBlack->setFont(lable_font_);
ui->GreenGridBlack->setFont(lable_font_);
ui->BlueGridBlack->setFont(lable_font_);
ui->MagentaGridBlack->setFont(lable_font_);
ui->YellowGridBlack->setFont(lable_font_);
ui->Red->setFont(lable_font_);
ui->Green->setFont(lable_font_);
ui->Blue->setFont(lable_font_);
ui->Orange->setFont(lable_font_);
ui->White->setFont(lable_font_);
ui->Black->setFont(lable_font_);
ui->Gray50->setFont(lable_font_);
ui->RedGradient->setFont(lable_font_);
ui->GreenGradient->setFont(lable_font_);
ui->BlueGradient->setFont(lable_font_);
ui->OrangeGradient->setFont(lable_font_);
ui->GrayGradient->setFont(lable_font_);
ui->ColorPalette->setFont(lable_font_);
ui->ColorPalette180->setFont(lable_font_);
ui->WhiteText->setFont(lable_font_);
ui->BlackText->setFont(lable_font_);
ui->RedText->setFont(lable_font_);
ui->BlueText->setFont(lable_font_);
ui->MagentaText->setFont(lable_font_);
ui->YellowText->setFont(lable_font_);

ui->LHints->setText(page_hints_ua[ui->stackedWidget->currentIndex()]);
}

void MainWindow::SetENLocalization()
{
    setWindowTitle(title_en[0]);
    //bar menu
    ui->menuFile->setTitle(menu_en[0]);
    ui->menuSelection->setTitle(menu_en[1]);
    ui->menuHelp->setTitle(menu_en[2]);
    ui->actionExit->setText(menu_en[3]);
    ui->actionPreferences->setText(menu_en[4]);
    ui->actionSelect_all->setText(menu_en[5]);
    ui->actionClear_All->setText(menu_en[6]);
    ui->actionTests_for_LCD_monitors->setText(menu_en[7]);
    ui->actionAbout->setText(menu_en[8]);
    ui->menuLanguage->setTitle(menu_en[9]);
    ui->actionUkranian->setText(menu_en[10]);
    ui->actionEnglish->setText(menu_en[11]);

    //buttons
    ui->RunTests->setText(buttons_en[0]);
    ui->RunTests->setGeometry(QRect(600, 430, 165, 30));

    ui->AutoRunTests->setText(buttons_en[1]);
    ui->AutoRunTests->setGeometry(QRect(374, 430, 211, 30));

    ui->CalibrationTests->setText(buttons_en[2]);
}

```

```

ui->CalibrationTests->setGeometry(QRect(21, 11, 144, 34));

ui->GridTests->setText(buttons_en[3]);
ui->GridTests->setGeometry(QRect(171, 11, 87, 34));

ui->ColorTests->setText(buttons_en[4]);
ui->ColorTests->setGeometry(QRect(264, 11, 97, 34));

ui->ReadingTests->setText(buttons_en[5]);
ui->ReadingTests->setGeometry(QRect(367, 11, 119, 34));

//group boxes
ui->groupBox->setTitle(group_box_en[0]);
ui->groupBox_2->setTitle(group_box_en[1]);
ui->groupBox_3->setTitle(group_box_en[2]);
ui->groupBox_4->setTitle(group_box_en[3]);
ui->groupBox_5->setTitle(group_box_en[4]);
ui->groupBox_6->setTitle(group_box_en[5]);
ui->groupBox_7->setTitle(group_box_en[6]);
ui->groupBox_8->setTitle(group_box_en[7]);

//labels
ui->LCDCalibration->setText(lables_en[0]);
ui->Brightness->setText(lables_en[1]);
ui->Scope->setText(lables_en[2]);
ui->Gamma->setText(lables_en[3]);
ui->Convergence->setText(lables_en[4]);
ui->ScreenSize->setText(lables_en[5]);
ui->WhitePattern->setText(lables_en[6]);
ui->BlackPattern->setText(lables_en[7]);
ui->Sharpness->setText(lables_en[8]);
ui->Dots->setText(lables_en[9]);
ui->VerticalLines->setText(lables_en[10]);
ui->HorisontalLines->setText(lables_en[11]);
ui->WhiteGrid->setText(lables_en[12]);
ui->RedGrid->setText(lables_en[13]);
ui->GreenGrid->setText(lables_en[14]);
ui->BlueGrid->setText(lables_en[15]);
ui->MagentaGrid->setText(lables_en[16]);
ui->YellowGrid->setText(lables_en[17]);
ui->BlackGrid->setText(lables_en[18]);
ui->RedGridBlack->setText(lables_en[19]);
ui->GreenGridBlack->setText(lables_en[20]);
ui->BlueGridBlack->setText(lables_en[21]);
ui->MagentaGridBlack->setText(lables_en[22]);
ui->YellowGridBlack->setText(lables_en[23]);
ui->Red->setText(lables_en[24]);
ui->Green->setText(lables_en[25]);
ui->Blue->setText(lables_en[26]);
ui->Orange->setText(lables_en[27]);
ui->White->setText(lables_en[28]);
ui->Black->setText(lables_en[29]);
ui->Gray50->setText(lables_en[30]);
ui->RedGradient->setText(lables_en[31]);
ui->GreenGradient->setText(lables_en[32]);
ui->BlueGradient->setText(lables_en[33]);
ui->OrangeGradient->setText(lables_en[34]);
ui->GrayGradient->setText(lables_en[35]);
ui->ColorPalette->setText(lables_en[36]);
ui->ColorPalette180->setText(lables_en[37]);
ui->WhiteText->setText(lables_en[38]);
ui->BlackText->setText(lables_en[39]);
ui->RedText->setText(lables_en[40]);
ui->BlueText->setText(lables_en[41]);
ui->MagentaText->setText(lables_en[42]);
ui->YellowText->setText(lables_en[43]);

ui->groupBox_8->setGeometry(409, 30, 231, 251);

QFont lable_font_("Segoe UI", 14);

```

```

ui->LCDCalibration->setFont(lable_font_);
ui->Brightness->setFont(lable_font_);
ui->Scope->setFont(lable_font_);
ui->Gamma->setFont(lable_font_);
ui->Convergence->setFont(lable_font_);
ui->ScreenSize->setFont(lable_font_);
ui->WhitePattern->setFont(lable_font_);
ui->BlackPattern->setFont(lable_font_);
ui->Sharpness->setFont(lable_font_);
ui->Dots->setFont(lable_font_);
ui->VerticalLines->setFont(lable_font_);
ui->HorisontalLines->setFont(lable_font_);
ui->WhiteGrid->setFont(lable_font_);
ui->RedGrid->setFont(lable_font_);
ui->GreenGrid->setFont(lable_font_);
ui->BlueGrid->setFont(lable_font_);
ui->MagentaGrid->setFont(lable_font_);
ui->YellowGrid->setFont(lable_font_);
ui->BlackGrid->setFont(lable_font_);
ui->RedGridBlack->setFont(lable_font_);
ui->GreenGridBlack->setFont(lable_font_);
ui->BlueGridBlack->setFont(lable_font_);
ui->MagentaGridBlack->setFont(lable_font_);
ui->YellowGridBlack->setFont(lable_font_);
ui->Red->setFont(lable_font_);
ui->Green->setFont(lable_font_);
ui->Blue->setFont(lable_font_);
ui->Orange->setFont(lable_font_);
ui->White->setFont(lable_font_);
ui->Black->setFont(lable_font_);
ui->Gray50->setFont(lable_font_);
ui->RedGradient->setFont(lable_font_);
ui->GreenGradient->setFont(lable_font_);
ui->BlueGradient->setFont(lable_font_);
ui->OrangeGradient->setFont(lable_font_);
ui->GrayGradient->setFont(lable_font_);
ui->ColorPalette->setFont(lable_font_);
ui->ColorPalette180->setFont(lable_font_);
ui->WhiteText->setFont(lable_font_);
ui->BlackText->setFont(lable_font_);
ui->RedText->setFont(lable_font_);
ui->BlueText->setFont(lable_font_);
ui->MagentaText->setFont(lable_font_);
ui->YellowText->setFont(lable_font_);

ui->LHints->setText(page_hints_en[ui->stackedWidget->currentIndex()]);
}

void MainWindow::SetPreview(QLabel *preview_lable, const char *picture)
{
    QPixmap pix(picture);
    preview_lable->setPixmap(
        pix.scaled(preview_lable->width(), preview_lable->height(), Qt::KeepAspectRatio));
}

void MainWindow::createTestWindow()
{
    if (ui->stackedWidget->currentIndex() == 0) {
        cf_test = new CalibrationFocusTest(isAutoTest, auto_test_timeout_);
        cf_test->setTestCodeVector(getCodeVector());
        cf_test->setTestFocusVector(getFocusVector());
        cf_test->setUpTest();
        cf_test->showFullScreen();
    } else if (ui->stackedWidget->currentIndex() == 1) {
        gridTest = new GridTest(isAutoTest, auto_test_timeout_);
        gridTest->setTestCodeVector(getCodeVector());
        gridTest->setTestColorVector(getColorVector());
        gridTest->setFirstGrid(gridTest->getFirstColor());
        gridTest->showFullScreen();
    } else if (ui->stackedWidget->currentIndex() == 2) {

```

```

colorTest = new ColorTest(isAutoTest, auto_test_timeout_);
colorTest->setVectorValues(getVector());
colorTest->setCodeVector(getCodeVector());
colorTest->setColorVector(getColorVector());
colorTest->showFullScreen();
} else if (ui->stackedWidget->currentIndex() == 3) {
    readTest = new ReadTest(isAutoTest, auto_test_timeout_);
    readTest->setTestCodeVector(getCodeVector());
    readTest->setTextColorVector(getColorVector());
    readTest->setColor(readTest->getFirstColor());
    readTest->showFullScreen();
}

switch (getFirstCodeElement()) {
case 2:
    break;
case 3:
    break;
case 4:
    colorTest->setStyleSheet(colorTest->getFirstElement());
    break;
case 5:
    colorTest->GradientTest(getFirstColorElement());
    break;
case 6:
    colorTest->paletteTest(":/color/palette/pictures/Palette.png");
    break;
case 7:
    colorTest->paletteTest(":/color/palette/pictures/Palette180.png");
    break;
case 8:
    break;
}
}

std::vector<uint8_t> MainWindow::getCodeVector()
{
    return testCodeVector;
}
std::vector<QColor> MainWindow::getColorVector(){return testColorVector;}

std::vector<uint8_t> MainWindow::getFocusVector()
{
    return testFocusVector;
}

std::vector<const char *> MainWindow::getVector()
{
    return testsVector;
}

void MainWindow::on_CalibrationTests_clicked()
{
    ui->stackedWidget->setcurrentIndex(0);
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0] : page_hints_ua[0]);
    SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/CalibrationTestPreview.png");
}

void MainWindow::on_GridTests_clicked()
{
    ui->stackedWidget->setcurrentIndex(1);
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[1] : page_hints_ua[1]);
    SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
}

void MainWindow::on_ColorTests_clicked()
{
    ui->stackedWidget->setcurrentIndex(2);
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2] : page_hints_ua[2]);
}

```

```

        SetPreview(ui->ColorPicture, "./color/palette/pictures/ColorTestPreview.png");
    }

void MainWindow::on_ReadingTests_clicked()
{
    ui->stackedWidget->setcurrentIndex(3);
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[3] : page_hints_ua[3]);
    SetPreview(ui->ReadPicture, "./color/palette/pictures/ReadPageTestPreview.png");
}

void MainWindow::on_actionSelect_all_triggered()
{
    for (int i = 0; i < ui->stackedWidget->count(); i++)
    {
        QList<QCheckBox*> checkBoxes = ui->stackedWidget->widget(i)->findChildren<QCheckBox*>();
        for (int j = 0; j < checkBoxes.count(); j++)
        {
            checkBoxes.at(j)->setChecked(true);
        }
    }
}

void MainWindow::clearAllCheckBoxes()
{
    for (int i = 0; i < ui->stackedWidget->count(); i++) {
        QList<QCheckBox *> checkBoxes = ui->stackedWidget->widget(i)->findChildren<QCheckBox *>();
        for (int j = 0; j < checkBoxes.count(); j++) {
            checkBoxes.at(j)->setChecked(false);
        }
    }
}

void MainWindow::on_actionClear_All_triggered()
{
    clearAllCheckBoxes();
}

void MainWindow::on_RunTests_clicked()
{
    isAutoTest = false;
    checkSelectedTests();
}

void MainWindow::on_AutoRunTests_clicked()
{
    isAutoTest = true;
    checkSelectedTests();
}

void MainWindow::addSelectedColorTests()
{
    clearAllVectors();

    //Solid Tests
    if(ui->Red->isChecked()){
        testsVector.push_back("background-color: #FF0000;");
        testCodeVector.push_back(4);
        testColorVector.push_back(Qt::white);
    }
    if(ui->Green->isChecked()){
        testsVector.push_back("background-color: #00FF00;");
        testCodeVector.push_back(4);
        testColorVector.push_back(Qt::white);
    }
    if(ui->Blue->isChecked()){
        testsVector.push_back("background-color: #0000FF;");
        testCodeVector.push_back(4);
        testColorVector.push_back(Qt::white);
    }
    if(ui->Orange->isChecked()){

```

```

    testsVector.push_back("background-color: #FFB400;");
    testCodeVector.push_back(4);
    testColorVector.push_back(Qt::white);
}
if(ui->White->isChecked()){
    testsVector.push_back("background-color: #FFFFFF;");
    testCodeVector.push_back(4);
    testColorVector.push_back(Qt::white);
}
if(ui->Black->isChecked()){
    testsVector.push_back("background-color: #000000;");
    testCodeVector.push_back(4);
    testColorVector.push_back(Qt::white);
}
if(ui->Gray50->isChecked()){
    testsVector.push_back("background-color: #7F7F7F;");
    testCodeVector.push_back(4);
    testColorVector.push_back(Qt::white);
}

//Gradient
if(ui->RedGradient->isChecked()){
    testsVector.push_back("background-color: #FF0000;");
    testCodeVector.push_back(5);
    testColorVector.push_back(QColor("#FF0000"));
}
if(ui->GreenGradient->isChecked()){
    testsVector.push_back("background-color: #FF0000;");
    testCodeVector.push_back(5);
    testColorVector.push_back(QColor("#00FF00"));
}
if(ui->BlueGradient->isChecked()){
    testsVector.push_back("background-color: #FF0000;");
    testCodeVector.push_back(5);
    testColorVector.push_back(QColor("#0000FF"));
}
if(ui->OrangeGradient->isChecked()){
    testsVector.push_back("background-color: #FF0000;");
    testCodeVector.push_back(5);
    testColorVector.push_back(QColor("#FFB400"));
}
if(ui->GrayGradient->isChecked()){
    testsVector.push_back("background-color: #FF0000;");
    testCodeVector.push_back(5);
    testColorVector.push_back(Qt::gray);
}

//ColorPallet
if(ui->ColorPalette->isChecked()){
    testsVector.push_back("background-color: #FF0000;");
    testColorVector.push_back(Qt::black);
    testCodeVector.push_back(6);
}
if(ui->ColorPalette180->isChecked()){
    testsVector.push_back("background-color: #FF0000;");
    testColorVector.push_back(Qt::black);
    testCodeVector.push_back(7);
}

void MainWindow::addSelectedGridTests()
{
    clearAllVectors();

    //White background
    if(ui->BlackGrid->isChecked())

```

```

{
    testCodeVector.push_back(3);
    testColorVector.push_back(Qt::black);
}

if(ui->RedGridBlack->isChecked())
{
    testCodeVector.push_back(3);
    testColorVector.push_back(Qt::red);
}

if(ui->GreenGridBlack->isChecked())
{
    testCodeVector.push_back(3);
    testColorVector.push_back(Qt::green);
}

if(ui->BlueGridBlack->isChecked())
{
    testCodeVector.push_back(3);
    testColorVector.push_back(Qt::blue);
}

if(ui->MagentaGridBlack->isChecked())
{
    testCodeVector.push_back(3);
    testColorVector.push_back(Qt::magenta);
}

if(ui->YellowGridBlack->isChecked())
{
    testCodeVector.push_back(3);
    testColorVector.push_back(Qt::yellow);
}

//Black background 2

if(ui->WhiteGrid->isChecked()){
    testCodeVector.push_back(2);
    testColorVector.push_back(Qt::white);
}

if(ui->RedGrid->isChecked()){
    testCodeVector.push_back(2);
    testColorVector.push_back(Qt::red);
}

if(ui->GreenGrid->isChecked()){
    testCodeVector.push_back(2);
    testColorVector.push_back(Qt::green);
}

if(ui->BlueGrid->isChecked()){
    testCodeVector.push_back(2);
    testColorVector.push_back(Qt::blue);
}

if(ui->MagentaGrid->isChecked()){
    testCodeVector.push_back(2);
    testColorVector.push_back(Qt::magenta);
}

if(ui->YellowGrid->isChecked()){
    testCodeVector.push_back(2);
    testColorVector.push_back(Qt::yellow);
}

}

void MainWindow::addReadTests()

```

```

{
    clearAllVectors();

    if(ui->WhiteText->isChecked()){
        testCodeVector.push_back(8);
        testColorVector.push_back(Qt::white);
    }

    if(ui->BlackText->isChecked()){
        testCodeVector.push_back(8);
        testColorVector.push_back(Qt::black);
    }

    if(ui->RedText->isChecked()){
        testCodeVector.push_back(8);
        testColorVector.push_back(Qt::red);
    }

    if(ui->BlueText->isChecked()){
        testCodeVector.push_back(8);
        testColorVector.push_back(Qt::blue);
    }

    if(ui->MagentaText->isChecked()){
        testCodeVector.push_back(8);
        testColorVector.push_back(Qt::magenta);
    }

    if(ui->YellowText->isChecked()){
        testCodeVector.push_back(8);
        testColorVector.push_back(Qt::yellow);
    }

}

void MainWindow::addCalibrationTests()
{
    clearAllVectors();

    //Calibration test
    if (ui->LCDCalibration->isChecked()) {
        testFocusVector.push_back(4);
        testCodeVector.push_back(1);
    }

    if (ui->Brightness->isChecked()) {
        testFocusVector.push_back(5);
        testCodeVector.push_back(1);
    }

    if (ui->Scope->isChecked()) {
        testFocusVector.push_back(11);
        testCodeVector.push_back(1);
    }

    if (ui->Gamma->isChecked()) {
        testFocusVector.push_back(6);
        testCodeVector.push_back(1);
    }

    if (ui->Convergence->isChecked()) {
        testFocusVector.push_back(12);
        testCodeVector.push_back(1);
    }

    if (ui->ScreenSize->isChecked()) {
        testFocusVector.push_back(10);
        testCodeVector.push_back(1);
    }

    //Focus test
}

```

```

if(ui->WhitePattern->isChecked()){
    testFocusVector.push_back(7);
    testCodeVector.push_back(1);
}
if(ui->BlackPattern->isChecked()){
    testFocusVector.push_back(8);
    testCodeVector.push_back(1);
}
if(ui->Sharpness->isChecked()){
    testFocusVector.push_back(9);
    testCodeVector.push_back(1);
}
if(ui->Dots->isChecked()){
    // Dots
    testFocusVector.push_back(1);
    testCodeVector.push_back(1);
}
if(ui->VerticalLines->isChecked()){
    // Vertical lines
    testFocusVector.push_back(2);
    testCodeVector.push_back(1);
}
if(ui->HorizontalLines->isChecked()){
    // Horizontal lines
    testFocusVector.push_back(3);
    testCodeVector.push_back(1);
}
}

uint8_t MainWindow::getFirstCodeElement()
{
    return testCodeVector[0];
}

QColor MainWindow::getFirstColorElement()
{
    return testColorVector[0];
}

void MainWindow::clearAllVectors()
{
    if (!testsVector.empty())
        testsVector.clear();
    if (!testCodeVector.empty())
        testCodeVector.clear();
    if (!testColorVector.empty())
        testColorVector.clear();
    if (!testFocusVector.empty())
        testFocusVector.clear();
}

void MainWindow::checkSelectedTests()
{
    if (ui->stackedWidget->currentIndex() == 0)
        addCalibrationTests();

    if (ui->stackedWidget->currentIndex() == 1)
        addSelectedGridTests();

    if (ui->stackedWidget->currentIndex() == 2)
        addSelectedColorTests();

    if (ui->stackedWidget->currentIndex() == 3)
        addReadTests();

    if (!testCodeVector.empty())
        createTestWindow();
}

void MainWindow::on_actionTests_for_LCD_monitors_triggered()

```

```

{
    clearAllCheckBoxes();
    ui->LCDCalibration->setChecked(true);
    ui->Brightness->setChecked(true);
    ui->Scope->setChecked(true);
    ui->Gamma->setChecked(true);
    ui->ScreenSize->setChecked(true);
    ui->Sharpness->setChecked(true);
    ui->Dots->setChecked(true);
    ui->MagentaGrid->setChecked(true);
    ui->MagentaGridBlack->setChecked(true);
    ui->YellowGridBlack->setChecked(true);
    ui->YellowGrid->setChecked(true);
    ui->Red->setChecked(true);
    ui->Green->setChecked(true);
    ui->Blue->setChecked(true);
    ui->Orange->setChecked(true);
    ui->White->setChecked(true);
    ui->Black->setChecked(true);
    ui->Gray50->setChecked(true);
    ui->RedGradient->setChecked(true);
    ui->GreenGradient->setChecked(true);
    ui->BlueGradient->setChecked(true);
    ui->OrangeGradient->setChecked(true);
    ui->GrayGradient->setChecked(true);
    ui->ColorPalette->setChecked(true);
    ui->ColorPalette180->setChecked(true);
    ui->WhiteText->setChecked(true);
    ui->BlackText->setChecked(true);
    ui->RedText->setChecked(true);
    ui->BlueText->setChecked(true);
    ui->MagentaText->setChecked(true);
    ui->YellowText->setChecked(true);
}

void MainWindow::on_actionPreferences_triggered()
{
    if (preference == nullptr) {
        preference = new Preference(auto_test_timeout_,
                                    save_settings_,
                                    language_state_ == LanguageMode::kEN ? true : false);
        connect(preference,
                &Preference::preferenceClosed,
                this,
                &MainWindow::handlePreferenceClosed);
    }

    preference->show();
    preference->exec();

    if (preference != nullptr) {
        delete preference;
        preference = nullptr;
    }
}

void MainWindow::SetHoveredLabels()
{
    //Calibration test
    ui->Label1Hider->installEventFilter(this);          // +
    ui->CalibrationHiderBrightness->installEventFilter(this); // +
    ui->CalibrationHiderScope->installEventFilter(this); // +
    ui->CalibrationHiderGamma->installEventFilter(this); // +
    ui->CalibrationHiderConvergence->installEventFilter(this); // +
    ui->CalibrationHiderScreenSize->installEventFilter(this); // +
    ui->CalibrationHiderWhitePattern->installEventFilter(this); // +
    ui->CalibrationHiderBlackPattern->installEventFilter(this); // +
    ui->CalibrationHiderSharpness->installEventFilter(this); // +
    ui->CalibrationHiderDotes->installEventFilter(this); // +
    ui->CalibrationHiderVertical->installEventFilter(this); // +
}

```

```

ui->CalibrationHiderHorizontal->installEventFilter(this); // +
//Read Test
ui->HideWhiteTextOnBlack->installEventFilter(this); // +
ui->HideBlackTextOnWhite->installEventFilter(this); // +
ui->HideBlueText->installEventFilter(this); // +
ui->HideMagentaText->installEventFilter(this); // +
ui->HideRedText->installEventFilter(this); // +
ui->HideYellowText->installEventFilter(this); // +

//Grid Test
ui->GridHideBlack->installEventFilter(this); // +
ui->GridHideBlueBlack->installEventFilter(this); // +
ui->GridHideBlueWhite_2->installEventFilter(this); // +
ui->GridHideGreenBlack->installEventFilter(this); // +
ui->GridHideGreenWhite->installEventFilter(this); // +
ui->GridHideMagentaBlack->installEventFilter(this); // +
ui->GridHideMagentaWhite->installEventFilter(this); // +
ui->GridHideRedBlack->installEventFilter(this); // +
ui->GridHideRedWhite->installEventFilter(this); // +
ui->GridHideWhite->installEventFilter(this); // +
ui->GridHideYellowWhite->installEventFilter(this); // +
ui->GridHideYellowBlack->installEventFilter(this); // +

//Color test
ui->ColorHideRed->installEventFilter(this); //+
ui->ColorHideGreen->installEventFilter(this); //+
ui->ColorHideBlue->installEventFilter(this); //+
ui->ColorHideOrange->installEventFilter(this); //+
ui->ColorHideWhite->installEventFilter(this); //+
ui->ColorHideBlack->installEventFilter(this); //+
ui->ColorHideGray->installEventFilter(this); //+
ui->ColorHideRedGradient->installEventFilter(this); //+
ui->ColorHideGreenGradient->installEventFilter(this); //+
ui->ColorHideBlueGradient->installEventFilter(this); //+
ui->ColorHideOrangeGradient->installEventFilter(this); //+
ui->ColorHideGreyGradient->installEventFilter(this); //+
ui->ColorHideColorPalette->installEventFilter(this); //+
ui->ColorHideColorPalette180->installEventFilter(this); //+
}

void MainWindow::closeEvent(QCloseEvent *event)
{
    if (this->isVisible()) {
        event->ignore();
        this->hide();
        sys_tray->showMessage();
    }
}

bool MainWindow::eventFilter(QObject *obj, QEvent *event)
{
    if (obj == (QObject *) ui->Lable1Hider) {
        if (event->type() == QEvent::Enter) {
            ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[0]
                : calibration_test_ua[0]);
            SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/LCD_Calibration.png");
        } else if (event->type() == QEvent::Leave) {
            ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                : page_hints_ua[0]);
            SetPreview(ui->CalibrationPicture,
                     (":/color/palette/pictures/CalibrationTestPreview.png");
        }
        return true;
    } else if (obj == (QObject *) ui->CalibrationHiderBrightness) {
        if (event->type() == QEvent::Enter) {
            ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[1]
                : calibration_test_ua[1]);
            SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/BrightnessAndContrast.png");
        } else if (event->type() == QEvent::Leave) {

```

```

ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                      : page_hints_ua[0]);
SetPreview(ui->CalibrationPicture,
         (":/color/palette/pictures/CalibrationTestPreview.png");
}
return true;
} else if (obj == (QObject *) ui->CalibrationHiderScope) {
    if (event->type() == QEvent::Enter) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[2]
                              : calibration_test_ua[2]);
        SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/Scope.png");
    } else if (event->type() == QEvent::Leave) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                              : page_hints_ua[0]);
        SetPreview(ui->CalibrationPicture,
                 (":/color/palette/pictures/CalibrationTestPreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->CalibrationHiderGamma) {
    if (event->type() == QEvent::Enter) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[3]
                              : calibration_test_ua[3]);
        SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/Gamma.png");
    } else if (event->type() == QEvent::Leave) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                              : page_hints_ua[0]);
        SetPreview(ui->CalibrationPicture,
                 (":/color/palette/pictures/CalibrationTestPreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->CalibrationHiderConvergence) {
    if (event->type() == QEvent::Enter) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[4]
                              : calibration_test_ua[4]);
        SetPreview(ui->CalibrationPicture,
                 (":/color/palette/pictures/CalibrationConveragePreviwe.png");
    } else if (event->type() == QEvent::Leave) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                              : page_hints_ua[0]);
        SetPreview(ui->CalibrationPicture,
                 (":/color/palette/pictures/CalibrationTestPreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->CalibrationHiderScreenSize) {
    if (event->type() == QEvent::Enter) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[5]
                              : calibration_test_ua[5]);
        SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/ScreenSizePreview.png");
    } else if (event->type() == QEvent::Leave) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                              : page_hints_ua[0]);
        SetPreview(ui->CalibrationPicture,
                 (":/color/palette/pictures/CalibrationTestPreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->CalibrationHiderWhitePattern) {
    if (event->type() == QEvent::Enter) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[6]
                              : calibration_test_ua[6]);
        SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/WhitePattern.png");
    } else if (event->type() == QEvent::Leave) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                              : page_hints_ua[0]);
        SetPreview(ui->CalibrationPicture,
                 (":/color/palette/pictures/CalibrationTestPreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->CalibrationHiderBlackPattern) {
    if (event->type() == QEvent::Enter) {
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[6]

```

```

        : calibration_test_ua[6]);
    SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/BlackPattern.png");
} else if (event->type() == QEvent::Leave) {
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                           : page_hints_ua[0]);
    SetPreview(ui->CalibrationPicture,
             (":/color/palette/pictures/CalibrationTestPreview.png");
}
return true;
} else if (obj == (QObject *) ui->CalibrationHiderSharpness) {
if (event->type() == QEvent::Enter) {
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[6]
                           : calibration_test_ua[6]);
    SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/Sharpness.png");
} else if (event->type() == QEvent::Leave) {
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                           : page_hints_ua[0]);
    SetPreview(ui->CalibrationPicture,
             (":/color/palette/pictures/CalibrationTestPreview.png");
}
return true;
} else if (obj == (QObject *) ui->CalibrationHiderDotes) {
if (event->type() == QEvent::Enter) {
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[6]
                           : calibration_test_ua[6]);
    SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/DotesPreview.png");
} else if (event->type() == QEvent::Leave) {
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                           : page_hints_ua[0]);
    SetPreview(ui->CalibrationPicture,
             (":/color/palette/pictures/CalibrationTestPreview.png");
}
return true;
} else if (obj == (QObject *) ui->CalibrationHiderVertical) {
if (event->type() == QEvent::Enter) {
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[6]
                           : calibration_test_ua[6]);
    SetPreview(ui->CalibrationPicture,(":/color/palette/pictures/VerticalLinesPreview.png");
} else if (event->type() == QEvent::Leave) {
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                           : page_hints_ua[0]);
    SetPreview(ui->CalibrationPicture,
             (":/color/palette/pictures/CalibrationTestPreview.png");
}
return true;
} else if (obj == (QObject *) ui->CalibrationHiderHorizontal) {
if (event->type() == QEvent::Enter) {
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? calibration_test_en[6]
                           : calibration_test_ua[6]);
    SetPreview(ui->CalibrationPicture,
             (":/color/palette/pictures/HorizontalLinesPreview.png");
} else if (event->type() == QEvent::Leave) {
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[0]
                           : page_hints_ua[0]);
    SetPreview(ui->CalibrationPicture,
             (":/color/palette/pictures/CalibrationTestPreview.png");
}
return true;
}

else if (obj == (QObject *) ui->HideWhiteTextOnBlack) {
if (event->type() == QEvent::Enter) {
    SetPreview(ui->ReadPicture,(":/color/palette/pictures/WhiteBlackReadPreview.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? read_en_en : read_en_ua);
} else if (event->type() == QEvent::Leave) {
    SetPreview(ui->ReadPicture,(":/color/palette/pictures/ReadPageTestPreview.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[3]
                           : page_hints_ua[3]);
}
return true;
}

```

```

} else if (obj == (QObject *) ui->HideBlackTextOnWhite) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/BlackWhiteReadPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? read_en_en : read_en_ua);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/ReadPageTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[3]
                            : page_hints_ua[3]);
    }
    return true;
} else if (obj == (QObject *) ui->HideBlueText) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/BlueReadPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? read_en_en : read_en_ua);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/ReadPageTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[3]
                            : page_hints_ua[3]);
    }
    return true;
} else if (obj == (QObject *) ui->HideMagentaText) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/MagentaReadPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? read_en_en : read_en_ua);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/ReadPageTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[3]
                            : page_hints_ua[3]);
    }
    return true;
} else if (obj == (QObject *) ui->HideRedText) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/RedReadPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? read_en_en : read_en_ua);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/ReadPageTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[3]
                            : page_hints_ua[3]);
    }
    return true;
} else if (obj == (QObject *) ui->HideYellowText) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/YellowReadPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? read_en_en : read_en_ua);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ReadPicture,(":/color/palette/pictures/ReadPageTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[3]
                            : page_hints_ua[3]);
    }
    return true;
} else if (obj == (QObject *) ui->GridHideBlack) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridBlack.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideBlueBlack) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridBlueBlack.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideBlueWhite_2) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridBlueWhite.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
}

```

```

    return true;
} else if (obj == (QObject *) ui->GridHideGreenBlack) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridGreenBlack.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideGreenWhite) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridGreenWhite.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideMagentaBlack) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridMagentaBlack.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideMagentaWhite) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridMagentaWhite.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideRedBlack) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridReadBlack.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideRedWhite) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridRedWhite.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideWhite) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridWhite.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideYellowWhite) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridYellowWhite.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->GridHideYellowBlack) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridYellowBlack.png");
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->GridPicture,(":/color/palette/pictures/GridPagePreview.png");
    }
    return true;
} else if (obj == (QObject *) ui->ColorHideRed) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestRed.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[0]
            : color_test_ua[0]);
    } else if (event->type() == QEvent::Leave) {

```

```

SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                      : page_hints_ua[2]);
}
return true;
} else if (obj == (QObject *) ui->ColorHideGreen) {
if (event->type() == QEvent::Enter) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestGreen.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[1]
                          : color_test_ua[1]);
} else if (event->type() == QEvent::Leave) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                          : page_hints_ua[2]);
}
return true;
} else if (obj == (QObject *) ui->ColorHideBlue) {
if (event->type() == QEvent::Enter) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestBlue.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[2]
                          : color_test_ua[2]);
} else if (event->type() == QEvent::Leave) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                          : page_hints_ua[2]);
}
return true;
} else if (obj == (QObject *) ui->ColorHideOrange) {
if (event->type() == QEvent::Enter) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestOrange.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[3]
                          : color_test_ua[3]);
} else if (event->type() == QEvent::Leave) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                          : page_hints_ua[2]);
}
return true;
} else if (obj == (QObject *) ui->ColorHideWhite) {
if (event->type() == QEvent::Enter) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestWhite.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[4]
                          : color_test_ua[4]);
} else if (event->type() == QEvent::Leave) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                          : page_hints_ua[2]);
}
return true;
} else if (obj == (QObject *) ui->ColorHideBlack) {
if (event->type() == QEvent::Enter) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestBlack.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[5]
                          : color_test_ua[5]);
} else if (event->type() == QEvent::Leave) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                          : page_hints_ua[2]);
}
return true;
} else if (obj == (QObject *) ui->ColorHideGray) {
if (event->type() == QEvent::Enter) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestGray.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[6]
                          : color_test_ua[6]);
} else if (event->type() == QEvent::Leave) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                          : page_hints_ua[2]);
}
}

```

```

    return true;
} else if (obj == (QObject *) ui->ColorHideRedGradient) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestRedGradient.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[7]
                            : color_test_ua[7]);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                            : page_hints_ua[2]));
    }
    return true;
} else if (obj == (QObject *) ui->ColorHideGreenGradient) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestGreenGradient.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[8]
                            : color_test_ua[8]);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                            : page_hints_ua[2]));
    }
    return true;
} else if (obj == (QObject *) ui->ColorHideBlueGradient) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestBlueGradient.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[9]
                            : color_test_ua[9]);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                            : page_hints_ua[2]));
    }
    return true;
} else if (obj == (QObject *) ui->ColorHideOrangeGradient) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestOrangeGradient.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[10]
                            : color_test_ua[10]);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                            : page_hints_ua[2]));
    }
    return true;
} else if (obj == (QObject *) ui->ColorHideGreyGradient) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestGrayGradient.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[11]
                            : color_test_ua[11]);
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                            : page_hints_ua[2]));
    }
    return true;
} else if (obj == (QObject *) ui->ColorHideColorPalette) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/Palette.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[12]
                            : color_test_ua[12]));
    } else if (event->type() == QEvent::Leave) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
        ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                            : page_hints_ua[2]));
    }
    return true;
} else if (obj == (QObject *) ui->ColorHideColorPalette180) {
    if (event->type() == QEvent::Enter) {
        SetPreview(ui->ColorPicture,(":/color/palette/pictures/Palette180.png");

```

```

ui->LHints->setText(language_state_ == LanguageMode::kEN ? color_test_en[13]
                      : color_test_ua[13]);
} else if (event->type() == QEvent::Leave) {
    SetPreview(ui->ColorPicture,(":/color/palette/pictures/ColorTestPreview.png");
    ui->LHints->setText(language_state_ == LanguageMode::kEN ? page_hints_en[2]
                          : page_hints_ua[2]);
}
return true;
}

else {
    // pass the event on to the parent class
    return QWidget::eventFilter(obj, event);
}
}

void MainWindow::on_actionExit_triggered()
{
    this->close();
}

void MainWindow::handlePreferenceClosed()
{
    //Here must be saving :)
}

void MainWindow::on_actionAbout_triggered()
{
    if (about == nullptr) {
        about = new About(language_state_ == LanguageMode::kEN ? true : false);
    }

    about->show();
    about->exec();

    if (about != nullptr) {
        delete about;
        about = nullptr;
    }
}

void MainWindow::on_actionUkrainian_triggered()
{
    if (language_state_ == LanguageMode::kUA)
        return;
    SetUALocalization();
    language_state_ = LanguageMode::kUA;
}

void MainWindow::on_actionEnglish_triggered()
{
    if (language_state_ == LanguageMode::kEN)
        return;
    SetENLocalization();
    language_state_ = LanguageMode::kEN;
}

```

systray.h

```

#ifndef SYSTRAY_H
#define SYSTRAY_H

#include <QMainWindow>
#include "QSystemTrayIcon"

class SysTray : public QMainWindow
{
    Q_OBJECT
public:
    SysTray();

```

```

~SysTray();
void showMessage();

private:
    QSystemTrayIcon *sysTrayIcon{};

    QMenu *menu{};
    QAction *viewWindow{};
    QAction *minimazeAction{};
    QAction *quitAction{};
    QAction *showWidgetAction{};

signals:
    void showApp(bool checked);
    void closeApp(bool checked);
    void hideApp(bool checked);
    void activateTray(QSystemTrayIcon::ActivationReason reason);
};

#endif // SYSTRAY_H

systray.cpp
#include "systray.h"
#include "qmenu.h"

const static QString styleTray
= "QMenu { background-color: rgb(33, 37, 43); color : white; }"
  "QMenu::item:selected { background-color: rgb(26, 58, 85); } "
  "QMenu::separator { background-color: white; height: 1px; margin: 5px 0px 5px 0px; }";

SysTray::SysTray()
{
    sysTrayIcon = new QSystemTrayIcon(this);
    sysTrayIcon->setToolTip(tr("Display testing"));
    sysTrayIcon->setIcon(QIcon(":/color/palette/pictures/MonitorTesting.ico"));

    /* Also connect clicking on the icon to the signal processor of this press */
    connect(sysTrayIcon,
            &QSystemTrayIcon::activated,
            this,
            [this](QSystemTrayIcon::ActivationReason reason) { emit this->activateTray(reason); });

    /*Create context menu for sys tray*/
    menu = new QMenu(this);
    viewWindow = new QAction(tr("Open"), this);
    minimazeAction = new QAction(tr("Minimize"), this);
    quitAction = new QAction(tr("Quit"), this);

    QFont font("Segoe UI", 12);

    menu->addAction(viewWindow);
    menu->addAction(minimazeAction);
    menu->addAction(quitAction);
    menu->addSeparator();
    menu->addAction(showWidgetAction);

    menu->setStyleSheet(styleTray);
    menu->setFont(font);

    sysTrayIcon->setContextMenu(menu);
    sysTrayIcon->show();

    // connect(sysTrayIcon, &QSystemTrayIcon::activated, )
    connect(viewWindow, &QAction::triggered, [this](bool checked = false) {
        emit this->showApp(checked);
    });
    connect(minimazeAction, &QAction::triggered, [this](bool checked = false) {
        emit this->hideApp(checked);
    });
    connect(quitAction, &QAction::triggered, [this](bool checked = false) {
        emit this->closeApp(checked);
    });
}

```

```

    });
}

SysTray::~SysTray()
{
    delete sysTrayIcon;
    delete menu;
    delete viewWindow;
    delete minimizeAction;
    delete quitAction;
    delete showWidgetAction;
}

void SysTray::showMessage()
{
    QSystemTrayIcon::MessageIcon i = QSystemTrayIcon::MessageIcon(QSystemTrayIcon::Information);
    sysTrayIcon->showMessage(
        "Display testing",
        (tr("The application is minimized to the tray. To maximize the application window "
            "click on the application icon in the tray")),
        i,
        2000);
}

calibration_focus_test.h
#ifndef CALIBRATION_FOCUS_TEST_H
#define CALIBRATION_FOCUS_TEST_H

#include <QGraphicsScene>
#include <QLabel>
#include <QPainter>
#include <QPushButton>
#include <QTimer>
#include <QVBoxLayout>
#include <QWidget>
#include <unordered_map>

constexpr int kDelta{3};
constexpr int kLineDelta{4};
constexpr int kLineWidth{3};
constexpr int kPixelsSize{2};

enum class DrawType { kDots, kVertical, kHorizontal, kPicture, kScreenSize, kNone };

class CalibrationFocusTest : public QWidget
{
    Q_OBJECT
public:
    explicit CalibrationFocusTest(const bool &auto_test,
                                  const size_t &timeout,
                                  QWidget *parent = nullptr);
    ~CalibrationFocusTest();
    void paintEvent(QPaintEvent *event);
    void setDrawState(DrawType);
    DrawType getDrawState();

    void setTestCodeVector(std::vector<uint8_t>);

    void setTestFocusVector(std::vector<uint8_t>);

    void setUpTest();

protected:
    void keyPressEvent(QKeyEvent *event) override;

signals:

private:
    int index_ = 0;
    bool firstScreenSizeEnter;
    bool firstCalibrationTest;
}

```

```

DrawType draw_state;
QPushButton *backButton;
QScreen *screen;
QLabel *label;
//QPainter painter;
QGraphicsScene *scene;
QGraphicsView *view;
QVBoxLayout *layout;

std::vector<uint8_t> testCodeVector{};
std::vector<uint8_t> testFocusVector{};

private:
    bool saveDrawedPicture(DrawType type);
    void showPictureToTheScreen();
    void showCalibrationTest();
    void screenTest();
    void drawCoordinateSystem(int screenWidth, int screenHeight);
    void setUpTest(uint8_t);
    void updateImage();
};

#endif // CALIBRATION_FOCUS_TEST_H
calibration_focus_test.cpp
#include "calibration_focus_test.h"

#include <QFont>
#include <QGraphicsTextItem>
#include <QGraphicsView>
#include <QGuiApplication>
#include <QKeyEvent>
#include <QLinearGradient>
#include <QPaintEvent>
#include <QPainter>
#include <QPen>
#include <QPixmap>
#include <QPushButton>
#include <QRect>
#include <QScreen>
#include <QVBoxLayout>
#include <QtWidgets>

const QColor AxisColor = Qt::white;
const QColor GridColor = Qt::white;
const QFont TextFont("Arial", 8, QFont::Bold);

std::unordered_map<uint8_t, const char *> calib_focus_test_map
= {{4,(":/color/palette/pictures/LCD_Calibration.png")},
   {5,(":/color/palette/pictures/BrightnessAndContrast.png")},
   {6,(":/color/palette/pictures/Gamma.png")},
   {7,(":/color/palette/pictures/WhitePattern.png")},
   {8,(":/color/palette/pictures/BlackPattern.png")},
   {9,(":/color/palette/pictures/Sharpness.png")},
   {11,(":/color/palette/pictures/Scope.png")},
   {12,(":/color/palette/pictures/Converage.png")}};

CalibrationFocusTest::CalibrationFocusTest(const bool &auto_test,
                                           const size_t &timeout,
                                           QWidget *parent)
: QWidget{parent}
//, painter{this}
{
    label = nullptr;
    backButton = new QPushButton("X", this);
    backButton->setStyleSheet("background-color: #ff00ff;");
    connect(backButton, &QPushButton::clicked, this, &QWidget::close);

    screen = QGuiApplication::primaryScreen();

    backButton->setGeometry(QRect(QPoint(screen->geometry().width() - 35, 5), QSize(30, 30)));
}

```

```

index_ = 0;
draw_state = DrawType::kNone;
scene = nullptr;
view = nullptr;
layout = nullptr;
firstScreenSizeEnter = true;
firstCalibrationTest = true;
setFocusPolicy(Qt::StrongFocus);

if (auto_test) {
    QTimer *timer = new QTimer(this);
    connect(timer, &QTimer::timeout, this, &CalibrationFocusTest::updateImage);
    timer->start(timeout * 1000);
}
}

CalibrationFocusTest::~CalibrationFocusTest()
{
    if (backButton != nullptr)
        delete backButton;
    if (screen != nullptr)
        delete screen;
    if (label != nullptr)
        delete label;
    if (scene != nullptr)
        delete scene;
    if (view != nullptr)
        delete view;
    if (layout != nullptr)
        delete layout;
}

void CalibrationFocusTest::paintEvent(QPaintEvent *event)
{
    if (firstCalibrationTest) {
        firstCalibrationTest = false;
        QPainter painter(this);
        switch (draw_state) {
            case DrawType::kNone:
                break;
            case DrawType::kHorizontal:
            case DrawType::kVertical:
            case DrawType::kDots:
                saveDrawedPicture(draw_state);
                showPictureToTheScreen();
                break;
            case DrawType::kPicture:
                showCalibrationTest();
                break;
            case DrawType::kScreenSize:
                if (firstScreenSizeEnter)
                    screenTest();
                firstScreenSizeEnter = false;
                break;
        }
    }
}

void CalibrationFocusTest::setDrawState(DrawType state)
{
    this->draw_state = state;
}

DrawType CalibrationFocusTest::getDrawState()
{
    return draw_state;
}

void CalibrationFocusTest::setTestCodeVector(std::vector<uint8_t> code_vec)

```

```

{
    this->testCodeVector = code_vec;
}

void CalibrationFocusTest::setTestFocusVector(std::vector<uint8_t> focus_vec)
{
    this->testFocusVector = focus_vec;
}

void CalibrationFocusTest::setUpTest()
{
    if (testCodeVector[0] == 1) {
        switch (testFocusVector[0]) {
            case 4:
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
            case 11:
            case 12:
                draw_state = DrawType::kPicture;
                break;
            case 1:
                draw_state = DrawType::kDots;
                break;
            case 2:
                draw_state = DrawType::kVertical;
                break;
            case 3:
                draw_state = DrawType::kHorizontal;
                break;
            case 10:
                draw_state = DrawType::kScreenSize;
                break;
        }
    }
}

void CalibrationFocusTest::showCalibrationTest()
{
    label = new QLabel(this);
    QRect screenGeometry = QGuiApplication::primaryScreen()->geometry();
    QSize screenSize = screenGeometry.size();
    QPixmap pixmap(calib_focus_test_map[testFocusVector[index_]]);
    label->setPixmap(pixmap.scaled(screenSize));
    label->setGeometry(QRect(QPoint(0, 0), screenSize));
    label->showFullScreen();
    backButton->raise();
}

void CalibrationFocusTest::drawCoordinateSystem(int screenWidth, int screenHeight)
{
    // Calculate the size of the coordinate system based on screen size
    int coordWidth = screenWidth;
    int coordHeight = screenHeight;

    // Calculate the center of the coordinate system
    int centerX = screenWidth / 2;
    int centerY = screenHeight / 2;

    // Draw the grid
    QPen gridPen(GridColor);
    for (int x = 0; x <= coordWidth; x += 50) {
        scene->addLine(x, 0, x, coordHeight, gridPen);
    }
    for (int y = 0; y <= coordHeight; y += 50) {
        scene->addLine(0, y, coordWidth, y, gridPen);
    }
}

```

```

// Draw the x-axis
QPen xPen(AxisColor);
xPen.setWidth(2);
scene->addLine(centerX - coordWidth / 2, centerY, centerX + coordWidth / 2, centerY, xPen);
QGraphicsTextItem *textItemX = new QGraphicsTextItem("(" + QString::number(screenWidth) + ")");
textItemX->setDefaultTextColor(Qt::white);
textItemX->setFont(TextFont);
textItemX->setPos(centerX + coordWidth / 2 - 40, centerY - 40);
scene->addItem(textItemX);

// Draw the y-axis
QPen yPen(AxisColor);
yPen.setWidth(2);
scene->addLine(centerX, centerY - coordHeight / 2, centerX, centerY + coordHeight / 2, yPen);
QGraphicsTextItem *textItemY = new QGraphicsTextItem("(" + QString::number(screenHeight) + ")");
textItemY->setDefaultTextColor(Qt::white);
textItemY->setFont(TextFont);
textItemY->setPos(centerX - 40, centerY + coordHeight / 2 - 20);
scene->addItem(textItemY);

// Calculate the number of points to mark on the x-axis
int numPointsX = coordWidth / 50;
int startX = centerX - coordWidth / 2;

// Draw the points on the x-axis
for (int i = 0; i <= numPointsX; ++i) {
    int x = startX + i * 50;
    scene->addLine(x, centerY - 2, x, centerY + 2, xPen);
    QGraphicsTextItem *textItemX = new QGraphicsTextItem(QString::number(i * 50));
    textItemX->setDefaultTextColor(Qt::white);
    textItemX->setFont(TextFont);
    if (i == 0) {
        textItemX->setPos(x, centerY + 10);
    } else if (i % 2 == 0) {
        textItemX->setPos(x - 10, centerY + 10);
    } else {
        textItemX->setPos(x - 10, centerY - 20);
    }
    scene->addItem(textItemX);
}

// Calculate the number of points to mark on the y-axis
int numPointsY = coordHeight / 50;
int startY = centerY - coordHeight / 2;

// Draw the points on the y-axis
for (int i = 0; i <= numPointsY; ++i) {
    int y = startY + i * 50;
    scene->addLine(centerX - 2, y, centerX + 2, y, yPen);
    QGraphicsTextItem *textItemY = new QGraphicsTextItem(QString::number(i * 50));
    textItemY->setDefaultTextColor(Qt::white);
    textItemY->setFont(TextFont);
    if (i == 0) {
        textItemY->setPos(centerX - 40, y);
    } else if (i % 2 == 0) {
        textItemY->setPos(centerX - 40, y - 10);
    } else {
        textItemY->setPos(centerX + 10, y - 10);
    }
    scene->addItem(textItemY);
}

void CalibrationFocusTest::setUpTest(uint8_t code)
{
    firstCalibrationTest = true;
    switch (code) {
    case 1:
        draw_state = DrawType::kDots;
        break;
}

```

```

case 2:
    draw_state = DrawType::kVertical;
    break;
case 3:
    draw_state = DrawType::kHorizontal;
    break;
case 10:
    firstScreenSizeEnter = true;
    draw_state = DrawType::kScreenSize;
    break;
case 4:
case 5:
case 6:
case 7:
case 8:
case 9:
case 11:
case 12:
    draw_state = DrawType::kPicture;
    break;
}
}

void CalibrationFocusTest::updateImage()
{
    // Create a QKeyEvent object for the right key press event
    QKeyEvent *event = new QKeyEvent(QKeyEvent::KeyPress, Qt::Key_Right, Qt::NoModifier);

    // Call the keyPressEvent function with the simulated event
    keyPressEvent(event);

    // Clean up the allocated event object
    delete event;
}

void CalibrationFocusTest::screenTest()
{
    layout = new QVBoxLayout(this);
    setLayout(layout);
    layout->setContentsMargins(0, 0, 0, 0); // Remove layout margins

    scene = new QGraphicsScene(this);
    scene->setBackgroundBrush(Qt::black);
    view = new QGraphicsView(scene, this);
    view->setRenderHint(QPainter::Antialiasing);
    view->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    view->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    view->setBackgroundBrush(Qt::black); // Set the background color to black
    layout->addWidget(view);

    setStyleSheet("QWidget { background-color: black; }");
    // Get the screen size
    QScreen *screen = QGuiApplication::primaryScreen();
    QRect screenRect = screen->geometry();
    int screenWidth = screenRect.width();
    int screenHeight = screenRect.height();

    // Set the scene rect to match the screen size
    scene->setSceneRect(0, 0, screenWidth, screenHeight);
    view->setGeometry(screenRect); // Set the view's geometry to fill the entire window

    // Draw the coordinate system
    drawCoordinateSystem(screenWidth, screenHeight);
    backButton->raise();
}

void CalibrationFocusTest::keyPressEvent(QKeyEvent *event)
{
    if (label != nullptr) {
        delete label;
}

```

```

        label = nullptr;
    }
    if (scene != nullptr) {
        delete scene;
        scene = nullptr;
    }

    if (view != nullptr) {
        delete view;
        view = nullptr;
    }

    if (layout != nullptr) {
        delete layout;
        layout = nullptr;
    }

    switch (event->key()) {
        case Qt::Key_Left:
            index_ = (index_ - 1 + testFocusVector.size()) % testFocusVector.size();
            setUpTest(testFocusVector[index_]);
            break;
        case Qt::Key_Right:
            index_ = (index_ + 1) % testFocusVector.size();
            setUpTest(testFocusVector[index_]);
            break;
        default:
            QWidget::keyPressEvent(event);
            break;
    }
}

bool CalibrationFocusTest::saveDrawedPicture(DrawType type)
{
    // Create a QPixmap object with the size of the widget
    QPixmap pixmap(screen->geometry().size());
    pixmap.fill(Qt::white); // Optional: Set the background to transparent

    QPainter painter(&pixmap); // Set the pixmap as the painter's device

    switch (type) {
        case DrawType::kDots:
            for (int y = 0; y < screen->geometry().height(); y += kDelta) {
                for (int x = 0; x < screen->geometry().width(); x += kDelta) {
                    painter.fillRect(x, y, kPixelsSize, kPixelsSize, Qt::black);
                    painter.fillRect(x + kPixelsSize,
                                    y + kPixelsSize,
                                    kPixelsSize,
                                    kPixelsSize,
                                    Qt::black);
                }
            }
            break;
        case DrawType::kVertical: {
            painter.setPen(QPen(Qt::black, kLineWidth));
            int x = 0;
            int spacing = kLineDelta;

            while (x < screen->geometry().width()) {
                painter.drawLine(x, 0, x, screen->geometry().height());
                x += spacing;
            }
        } break;
        case DrawType::kHorizontal: {
            painter.setPen(QPen(Qt::black, kLineWidth));
            int y = 0;
            int spacing = kLineDelta;

            while (y < screen->geometry().height()) {
                painter.drawLine(0, y, screen->geometry().width(), y);
                y += spacing;
            }
        } break;
    }
}

```

```

        y += spacing;
    }
} break;
}
painter.end(); // Release the painter
}

pixmap.save("PictureOutput.png"); // Save the pixmap as an image file
return true;
}

```

```

void CalibrationFocusTest::showPictureToTheScreen()
{
    label = new QLabel(this);
    QRect screenGeometry = QGuiApplication::primaryScreen()->geometry();
    QSize screenSize = screenGeometry.size();
    QPixmap pixmap("PictureOutput.png");
    label->setPixmap(pixmap.scaled(screenSize));
    label->setGeometry(QRect(QPoint(0, 0), screenSize));
    label->showFullScreen();
    backButton->raise();
}

```

```

color_test.h
#ifndef TESTWINDOW_H
#define TESTWINDOW_H

```

```

#include <QWidget>
#include <QKeyEvent>
#include <QColor>
#include <QLabel>
#include <QPushButton>
#include <QPixmap>

```

```

class ColorTest : public QWidget
{
    Q_OBJECT
public:
    explicit ColorTest(const bool &auto_test, const size_t &timeout, QWidget *parent = nullptr);

    void setVectorValues(std::vector<const char*>);

    void setCodeVector(std::vector<uint8_t>);

    void setColorVector(std::vector<QColor>);

    const char *getFirstElement();

    void paletteTest(const char*);

    void GradientTest(QColor);

protected:
    void keyPressEvent(QKeyEvent *event) override;

signals:

private:
    std::vector<const char*> testsVector{};
    std::vector<uint8_t> testCodeVector{};
    std::vector<QColor> testColorVector{};
    int index_ = 0;
    QPushButton *backButton;
    QScreen *screen;
    QLabel *label;

private:
    void updateImage();
};

#endif // TESTWINDOW_H

```

color_test.cpp

```

#include "color_test.h"
#include <QGuiApplication>
#include <QKeyEvent>
#include <QLinearGradient>
#include <QPaintEvent>
#include <QPainter>
#include <QPixmap>
#include <QPushButton>
#include <QRect>
#include <QScreen>
#include <QTimer>
#include <QVBoxLayout>
#include <QtWidgets>

ColorTest::ColorTest(const bool &auto_test, const size_t &timeout, QWidget *parent)
    : QWidget{parent}
{
    label = nullptr;
    backButton = new QPushButton("X", this);
    backButton->setStyleSheet("background-color: #ff00ff;");
    connect(backButton, &QPushButton::clicked, this, &QWidget::close);

    screen = QGuiApplication::primaryScreen();

    backButton->setGeometry(QRect(QPoint(screen->geometry().width()-35, 5), QSize(30, 30)));

    index_ = 0;
    setFocusPolicy(Qt::StrongFocus);

    if (auto_test) {
        QTimer *timer = new QTimer(this);
        connect(timer, &QTimer::timeout, this, &ColorTest::updateImage);
        timer->start(timeout * 1000);
    }
}

void ColorTest::updateImage()
{
    // Create a QKeyEvent object for the right key press event
    QKeyEvent *event = new QKeyEvent(QKeyEvent::KeyPress, Qt::Key_Right, Qt::NoModifier);

    // Call the keyPressEvent function with the simulated event
    keyPressEvent(event);

    // Clean up the allocated event object
    delete event;
}

void ColorTest::setVectorValues(std::vector<const char *> v)
{
    this->testsVector = v;
}

void ColorTest::setCodeVector(std::vector<uint8_t> v){
    this->testCodeVector = v;
}

void ColorTest::setColorVector(std::vector<QColor> v){
    this->testColorVector = v;
}

const char *ColorTest::getFirstElement()
{
    return testsVector[0];
}

void ColorTest::paletteTest(const char *path)
{
    label = new QLabel(this);
    QRect screenGeometry = QGuiApplication::primaryScreen()->geometry();

```

```

QSize screenSize = screenGeometry.size();
QPixmap pixmap(path);
label->setPixmap(pixmap.scaled(screenSize));
label->setGeometry(QRect(QPoint(0, 0), screenSize));
label->showFullScreen();
backButton->raise();
}

void ColorTest::GradientTest(QColor color)
{
    setStyleSheet("QWidget {
        background-color: palette(Window);
        color: palette(WindowText);
    }");
    QPalette defaultPalette = QApplication::palette();
    setPalette(defaultPalette);

    QLinearGradient gradient(0, 0, screen->geometry().width(), screen->geometry().height());
    gradient.setColorAt(0, QColor(0, 0, 0, 0));
    gradient.setColorAt(1, color);

    // Set the widget's background to the gradient
    setAutoFillBackground(false);
    QPalette pal = palette();
    pal.setBrush(QPalette::Window, QBrush(gradient));
    setPalette(pal);
}

void ColorTest::keyPressEvent(QKeyEvent *event)
{
    if(label != nullptr){
        delete label;
        label = nullptr;
    }
    switch (event->key()) {
    case Qt::Key_Left:
        index_ = (index_ - 1 + testCodeVector.size()) % testCodeVector.size();
        if(testCodeVector[index_] == 4){
            setStyleSheet(testsVector[index_]);
        }
        if(testCodeVector[index_] == 5){
            GradientTest(testColorVector[index_]);
        }
        if(testCodeVector[index_] == 6){
            paletteTest(":/color/palette/pictures/Palette.png");
        }
        if(testCodeVector[index_] == 7){
            paletteTest(":/color/palette/pictures/Palette180.png");
        }
        break;
    case Qt::Key_Right:
        index_ = (index_ + 1) % testCodeVector.size();
        if(testCodeVector[index_] == 4){
            this->setStyleSheet(testsVector[index_]);
        }
        if(testCodeVector[index_] == 5){
            GradientTest(testColorVector[index_]);
        }
        if(testCodeVector[index_] == 6){
            paletteTest(":/color/palette/pictures/Palette.png");
        }
        if(testCodeVector[index_] == 7){
            paletteTest(":/color/palette/pictures/Palette180.png");
        }
        break;
    default:
        QWidget::keyPressEvent(event);
        break;
    }
}

```

```

grid_test.h
#ifndef GRIDWIDGET_H
#define GRIDWIDGET_H

#include <QWidget>

class GridTest : public QWidget
{
    Q_OBJECT
public:
    explicit GridTest(const bool &auto_test, const size_t &timeout, QWidget *parent = nullptr);
    void paintEvent(QPaintEvent *event) override;

    void setTestColorVector( std::vector<QColor> );
    void setTestCodeVector(std::vector<uint8_t> );
    QColor getFirstColor();
    void setFirstGrid(QColor);

signals:

public slots:
    void setColor(const QColor& color);

protected:
    void keyPressEvent(QKeyEvent *event) override;

private:
    int index_;
    std::vector<uint8_t> testCodeVector{};
    std::vector<QColor> testColorVector{};
    QColor m_color = Qt::black;
    QColor m_backGround = Qt::white;

private:
    void updateImage();
};

#endif // GRIDWIDGET_H

```

```

grid_test.cpp

#include "grid_test.h"

#include <QGuiApplication>
#include <QKeyEvent>
#include <QLinearGradient>
#include <QPaintEvent>
#include <QPainter>
#include <QPixmap>
#include <QPushButton>
#include <QRect>
#include <QScreen>
#include <QTimer>
#include <QVBoxLayout>
#include <QtWidgets>

#include <unordered_map>

GridTest::GridTest(const bool &auto_test, const size_t &timeout, QWidget *parent)
    : QWidget{parent}
{
    // Створення кнопки "Повернутися"
    QPushButton *backButton = new QPushButton("X", this);
    backButton->setStyleSheet("background-color: #ff00ff;");
    connect(backButton, &QPushButton::clicked, this, &QWidget::close);

    // Розміщення кнопки на вікні
    QScreen *screen = QGuiApplication::primaryScreen();
    backButton->setGeometry(QRect(QPoint(screen->geometry().width()-35, 5), QSize(30, 30)));
}
```

```

index_ = 0;
setFocusPolicy( Qt::StrongFocus );

if (auto_test) {
    QTimer *timer = new QTimer(this);
    connect(timer, &QTimer::timeout, this, &GridTest::updateImage);
    timer->start(timeout * 1000);
}
}

void GridTest::updateImage()
{
    // Create a QKeyEvent object for the right key press event
    QKeyEvent *event = new QKeyEvent(QKeyEvent::KeyPress, Qt::Key_Right, Qt::NoModifier);

    // Call the keyPressEvent function with the simulated event
    keyPressEvent(event);

    // Clean up the allocated event object
    delete event;
}

void GridTest::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event);
    QPainter painter(this);
    QScreen *screen = QGuiApplication::primaryScreen();
    painter.setPen(QPen(m_color, 1, Qt::SolidLine));
    const int cell_size = screen->geometry().width() / 20;
    const int cell_size2 = screen->geometry().height() / 20;
    const int rows = 20;
    const int cols = 20;
    const int grid_width = cell_size * cols;
    const int grid_height = cell_size2 * rows;
    const int x_offset = (screen->geometry().width() - grid_width) / 2;
    const int y_offset = (screen->geometry().height() - grid_height) / 2;
    for (int i = 0; i <= cols; i++) {
        const int x = x_offset + i * cell_size;
        painter.drawLine(x, y_offset, x, y_offset + grid_height);
    }
    for (int i = 0; i <= rows; i++) {
        const int y = y_offset + i * cell_size2;
        painter.drawLine(x_offset, y, x_offset + grid_width, y);
    }
}

void GridTest::setTestColorVector(std::vector<QColor> color)
{
    testColorVector = color;
}

void GridTest::setTestCodeVector(std::vector<uint8_t> code)
{
    testCodeVector = code;
}

QColor GridTest::getFirstColor()
{
    return testColorVector[0];
}

void GridTest::setFirstGrid(QColor color)
{
    std::unordered_map<uint8_t, QColor> mColor = {{2, Qt::black}, {3, Qt::white}};
    QPalette palette = this->palette();
    palette.setColor(QPalette::Window, mColor[testCodeVector[0]]);
    this->setAutoFillBackground(true);
    this->setPalette(palette);
    setColor(testColorVector[0]);
}

```

```

}

void GridTest::setColor(const QColor &color)
{
    m_color = color;
    update();
}

void GridTest::keyPressEvent(QKeyEvent *event)
{
    std::unordered_map<uint8_t, QColor> mColor = {{2, Qt::black}, {3, Qt::white}};
    QPalette palette = this->palette();

    switch (event->key()) {
    case Qt::Key_Left:
        index_ = (index_ - 1 + testCodeVector.size()) % testCodeVector.size();
        palette.setColor(QPalette::Window, mColor[testCodeVector[index_]]);
        this->setAutoFillBackground(true);
        this->setPalette(palette);
        setColor(testColorVector[index_]);
        break;
    case Qt::Key_Right:
        index_ = (index_ + 1) % testCodeVector.size();
        palette.setColor(QPalette::Window, mColor[testCodeVector[index_]]);
        this->setAutoFillBackground(true);
        this->setPalette(palette);
        setColor(testColorVector[index_]);
        break;
    default:
        QWidget::keyPressEvent(event);
        break;
    }
}

```

```

read_test.h
#ifndef READTEST_H
#define READTEST_H

```

```

#include <QWidget>
#include <QColor>
#include <QMap>

struct qColorMap {
    std::vector<QColor> key_;
    std::vector<QColor> value_;

    void insert(QColor keyColor, QColor valueColor)
    {
        key_.push_back(keyColor);
        value_.push_back(valueColor);
    }

    QColor value(QColor key){
        int index = 0;
        for(int i = 0; i < key_.size(); i++){
            if(key_[i] == key){
                index = i;
                break;
            }
        }
        return value_[index];
    }
};


```

```

class ReadTest : public QWidget
{
    Q_OBJECT

```

```

public:
    explicit ReadTest(const bool &auto_test, const size_t &timeout, QWidget *parent = nullptr);
    void paintEvent(QPaintEvent *event) override;

    void setTestColorVector( std::vector<QColor> );
    void setTestCodeVector(std::vector<uint8_t>);

    QColor getFirstColor();
signals:
public slots:
    void setColor(const QColor& color);

protected:
    void keyPressEvent(QKeyEvent *event) override;

private:
    QColor m_backgroundColor{};
    QColor m_textColor{};

    qColorMap qCMap;

    std::vector<uint8_t> testCodeVector{};
    std::vector<QColor> testColorVector{};
    int index_{};

private:
    void updateImage();
};

#endif // READTEST_H

read_test.cpp

#include "read_test.h"
#include <QWidget>

#include <QGuiApplication>
#include <QKeyEvent>
#include <QLinearGradient>
#include <QPaintEvent>
#include <QPainter>
#include <QPixmap>
#include <QPushButton>
#include <QRect>
#include <QScreen>
#include <QTimer>
#include <QVBoxLayout>
#include <QtWidgets>

ReadTest::ReadTest(const bool &auto_test, const size_t &timeout, QWidget *parent)
    : QWidget{parent}
{
    qCMap.insert(Qt::white, Qt::black);
    qCMap.insert(Qt::black, Qt::white);
    qCMap.insert(Qt::red, Qt::blue);
    qCMap.insert(Qt::blue, Qt::green);
    qCMap.insert(Qt::magenta, Qt::black);
    qCMap.insert(Qt::yellow, Qt::blue);

    // Створення кнопки "Повернутися"
    QPushButton *backButton = new QPushButton("X", this);
    backButton->setStyleSheet("background-color: #ff00ff;");
    connect(backButton, &QPushButton::clicked, this, &QWidget::close);

    // Розміщення кнопки на вікні
    QScreen *screen = QGuiApplication::primaryScreen();
    backButton->setGeometry(QRect(QPoint(screen->geometry().width()-35, 5), QSize(30, 30)));
}

```

```

index_ = 0;
setFocusPolicy(Qt::StrongFocus);

if (auto_test) {
    QTimer *timer = new QTimer(this);
    connect(timer, &QTimer::timeout, this, &ReadTest::updateImage);
    timer->start(timeout * 1000);
}
}

void ReadTest::updateImage()
{
    // Create a QKeyEvent object for the right key press event
    QKeyEvent *event = new QKeyEvent(QKeyEvent::KeyPress, Qt::Key_Right, Qt::NoModifier);

    // Call the keyPressEvent function with the simulated event
    keyPressEvent(event);

    // Clean up the allocated event object
    delete event;
}

QColor ReadTest::getFirstColor()
{
    return testColorVector[0];
}

void ReadTest::setColor(const QColor &color)
{
    m_textColor = color;
    m_backgroundColor = qCMap.value(color);
    update();
}

void ReadTest::setTestColorVector(std::vector<QColor> color)
{
    testColorVector = color;
}

void ReadTest::setTestCodeVector(std::vector<uint8_t> code)
{
    testCodeVector = code;
}

void ReadTest::paintEvent(QPaintEvent *event){
    QPainter painter(this);
    painter.fillRect(rect(), m_backgroundColor);
    painter.setPen(m_textColor);
    QFont font("Arial", 10);
    painter.setFont(font);

    QScreen *screen = QGuiApplication::primaryScreen();
    QString can_you_read_text = "Can you read text?";
    long long text_length = (can_you_read_text.length() + 15) * 4;

    int rowCount = screen->geometry().height() / 15;
    int colCount = screen->geometry().width() / text_length;
    int cellWidth = screen->geometry().width() / colCount;
    int cellHeight = screen->geometry().height() / rowCount;

    for (int i = 0; i < rowCount; i++) {
        for (int j = 0; j < colCount; j++) {
            QRect rect(j * cellWidth, i * cellHeight + 15, cellWidth, cellHeight);
            painter.drawText(rect, Qt::AlignCenter, "Can you read text?");
        }
    }
}

void ReadTest::keyPressEvent(QKeyEvent *event){

```

```

switch (event->key()) {
case Qt::Key_Left:
    index_ = (index_ - 1 + testCodeVector.size()) % testCodeVector.size();
    setColor(testColorVector[index_]);
    break;
case Qt::Key_Right:
    index_ = (index_ + 1) % testCodeVector.size();
    setColor(testColorVector[index_]);
    break;
default:
    QWidget::keyPressEvent(event);
    break;
}
}

language.h
#ifndef LANGUAGE_H
#define LANGUAGE_H

#include <QString>

static QStringList menu_ua[]{"Файл",
                            "Вибір",
                            "Допомога",
                            "Вийти",
                            "Налаштування",
                            "Вибрати всі",
                            "Очистити всі",
                            "Тест для LCD моніторів",
                            "Про програму",
                            "Мова",
                            "Українська",
                            "Англійська"};
```

```

static QStringList menu_en[]{"File",
                           "Selection",
                           "Help",
                           "Exit",
                           "Preferences",
                           "Select all",
                           "Clear all",
                           "Test for LCD monitors",
                           "About",
                           "Language",
                           "Ukrainian",
                           "English"};
```

```

static QStringList buttons_ua[]{"Запустити вибрані тести",
                             "Авто-запуск вибраних тестів",
                             "Калібрувальні тести",
                             "Сіткові тести",
                             "Тести кольором",
                             "Тести читабельності"};
```

```

static QStringList buttons_en[]{"Run Selected Tests",
                             "Auto Run Selected Tests",
                             "Calibration tests",
                             "Grid tests",
                             "Color tests",
                             "Reading tests"};
```

```

static QStringList group_box_ua[]{"Калібрувальні тести",
                                 "Фокусувальний",
                                 "Чорний фон",
                                 "Білий фон",
                                 "Монотонні",
                                 "Градієнт",
                                 "Палітри кольорів",
                                 "Тексти"};
```

```

static QStringList group_box_en[]{"Calibration tests",
```

```
"Focus Tests",
"Black background",
"White background",
"Solid Fills",
"Gradient Fills",
"Color Palettes",
"Texts"};
```

```
static QStringList labels_ua[] {"Калібрування РК-дисплея",
"Яскравість/контрастність",
"Діапазон",
"Гамма",
"Конвергенція",
"Розмір екрану",
"Білий візерунок",
"Чорний візерунок",
"Різкість",
"Точки",
"Вертикальні лінії",
"Горизонтальні лінії",
"Біла сітка",
"Червона сітка",
"Зелена сітка",
"Синя сітка",
"Пурпурова сітка",
"Жовта сітка",
"Чорна сітка",
"Червона сітка",
"Зелена сітка",
"Синя сітка",
"Пурпурова сітка",
"Жовта сітка",
"Червоний",
"Зелений",
"Синій",
"Помаранчевий",
"Білий",
"Чорний",
"50% Сірий",
"Червоний градієнт",
"Зелений градієнт",
"Синій градієнт",
"Помаранчевий градієнт",
"Сірий градієнт",
"Палітра кольорів (HSL)",
"180 Палітра кольорів (HSL)",
"Білий текст на чорному фоні",
"Чорний текст на білому фоні",
"Червоний текст на синьому фоні",
"Синій Текст на зеленому фоні",
"Пурпурний Текст на чорному фоні",
"Жовтий Текст на синьому фоні"};
```

```
static QStringList labels_en[] {"LCD Calibration",
"Brightness/Contrast",
"Scope",
"Gamma",
"Convergence",
"Screen Size",
"White Pattern",
"Black Pattern",
"Sharpness",
"Dots",
"Vertical Lines",
"Horizontal Lines",
"White Grid",
"Red Grid",
"Green Grid",
"Blue Grid",
"Magenta Grid",
```

```

    "Yellow Grid",
    "Black Grid",
    "Red Grid",
    "Green Grid",
    "Blue Grid",
    "Magenta Grid",
    "Yellow Grid",
    "Red",
    "Green",
    "Blue",
    "Orange",
    "White",
    "Black",
    "50% Gray",
    "Red Gradient",
    "Green Gradient",
    "Blue Gradient",
    "Orange Gradient",
    "Gray Gradient",
    "Color Palette (HSL)",
    "180 Color Palette (HSL)",
    "White Text on Black",
    "Black Text on White",
    "Red Text on Blue",
    "Blue Text on Green",
    "Magenta Text on Black",
    "Yellow Text on Blue"};
}

static QStringList preferences_ua[]{"Загальні налаштування",
    "Зберегти стан вибору тесту при виході",
    "Автоматичний режим",
    "Секунди, протягом яких буде показано\показано тести на екрані",
    "Зберегти",
    "Налаштування збережено",
    "Налаштування успішно збережено!",
    "Гаразд"};
}

static QStringList preferences_en[]{"General settings",
    "Save test selection state on exit",
    "Automatic mode",
    "Seconds that each test screen will be shown",
    "Save",
    "Settings Saved",
    "The settings have been saved!",
    "Ok"};
}

static QString about_ua(){
    "Тестування монітора",
    "Інформація про продукт:",
    "Дана робота виконана в рамках курсового проекту\показано дисципліни: \"Системне "
    "програмне забезпечення\"",
    "Автор роботи:",
    "Романів Віталій Андрійович, студент групи КІ-35 Національного університету \"Львівська "
    "політехніка\" спеціальності \"Комп'ютерна інженерія\"",
    "Версія: 1.0 beta";
}

static QString about_en(){
    "Monitor Testing",
    "Product information:",
    "This work was performed within the scope of the course\nproject in the "
    "discipline: \"System Software\"",
    "Author of the work:",
    "Romaniv Vitalii Andriyovich, student of group KI-35 of the National University\n\"Lviv "
    "Polytechnic\" majoring in Computer Engineering",
    "Version: 1.0 beta";
}

static QStringList title_ua[]{"Тестування монітору", "Про програму", "Налаштування тестування монітору"};
static QStringList title_en[]{"Monitor testing", "About", "Display testing preferences"};

```

```
static QString page_hints_en[] {
    " On this page, you can find several tests for monitor calibration."
    "These tests are suitable for both classic (CRT)\nand LCD displays."
    "\n Move your cursor over a test name to get a description about it."
    "and to see a small thumbnail about the\ntest screen."
    " Grid tests are ideal for checking, and tuning the display geometry, and convergence"
    "settings."
    " On this page, you can find several tests to check your computer and monitor your ability"
    "to display colors\nand see dead/stuck pixels on LCD displays."
    " On this page, you can find several reading tests. With these tests, you can check your"
    "monitor focus on\nthe whole screen and with different font and background colors."};

static QString page_hints_ua[] {
    " На цій сторінці ви можете знайти кілька тестів для калібрування монітора."
    "Ці тести підходять як для\nпкласичних (ЕПТ) так і для рідкокристалічних дисплейів."
    "\n Наведіть курсор на називу тесту, щоб отримати його опис"
    "та побачити невелику мініатюру екрана тесту."
    " Тести сітки ідеально підходять для перевірки та налаштування геометрії дисплея і"
    "збіжності\nналаштувань."
    " На цій сторінці ви можете знайти декілька тестів для перевірки вашого комп'ютера та"
    "моніторингу\nпашої здатності відображати кольори та бачити мертві/биті пікселі на"
    "РК-дисплеях."
    " На цій сторінці ви можете знайти декілька тестів на читання. За допомогою цих тестів ви"
    "можете\nперевірити фокус монітора на всьому екрані та з різними кольорами шрифту і тла."};

static QString read_en_en {
    " With this test, you can check your monitor's focus on the whole screen. Text reading"
    "is especially important\nfor all kinds of desktop work like word processing and web surfing,"
    "where clear, sharp letters relax the eyes."};

static QString read_en_ua {
    " За допомогою цього тесту ви можете перевірити фокусування вашого монітора на всьому"
    "екрані.\n Читання тексту особливо важливе для всіх видів настільної роботи, таких як"
    "обробка текстів і веб-\nсерфінг, де чіткі, різкі літери розслаблюють очі."};

static QString color_test_en[] {
    " This solid red screen is ideal to check the color uniformity of your monitor. LCD users"
    "are also able to\ncheck the display area for dead (always dark) red sub-pixels. You should"
    "degauss your classic (CRT) display before\nthis test. Magnetic fields (e.g. speakers) near"
    "the monitor can produce color distortions.\n Small imperfections are common in LCD"
    "monitors, please refer to your user's manual for the manufacturer's dead\npixel policy.",

    " This solid green screen is ideal to check the color uniformity of your monitor. LCD users"
    "are also able to\ncheck the display area for dead (always dark) green sub-pixels. You should"
    "degauss your classic (CRT)\ndisplay before this test. Magnetic fields (e.g. speakers)"
    "near the monitor can produce color distortions.\n Small imperfections are common in LCD"
    "monitors, please refer to your user's manual for the\nmanufacturer's dead pixel policy.",

    " This solid blue screen is ideal to check the color uniformity of your monitor. LCD users"
    "are also able to\ncheck the display area for dead (always dark) blue sub-pixels. You should"
    "degauss your classic (CRT)\ndisplay before this test. Magnetic fields (e.g. speakers) near"
    "the monitor can produce color distortions.\n Small imperfections are common in LCD"
    "monitors, please refer to your user's manual for the\nmanufacturer's dead pixel policy.",

    " This solid orange screen is ideal to check the color uniformity of your monitor. You"
    "should degauss your\nclassic (CRT) display before this test. Magnetic fields (e.g. speakers)"
    "near the monitor can produce\ncolor distortions.",

    " Completely white screen for checking the backlight brightness and uniformity of the LCD"
    "monitor and is\nalso good to find dead (always dark) pixels.\n Small imperfections are"
    "common in LCD monitors, please refer to your user's manual for the\nmanufacturer's dead pixel"
    "policy.",

    " Completely black screen for checking the backlight brightness and uniformity of the LCD"
    "monitor and is\nalso good to find dead (always dark) pixels.\n Small imperfections are"
    "common in LCD monitors, please refer to your user's manual for the\nmanufacturer's dead pixel"
    "policy.",

    " You should degauss your classic (CRT) display before this test. Magnetic fields (e.g."
    "speakers) near\nthe monitor can produce color distortions.\n Small imperfections are common"
    "in the monitor's dead pixel policy."};
```

"in LCD monitors, please refer to your user's manual for the manufacturer's\ndead pixel "
"policy.",

" Gradient fill is an ideal test to check the monitor's ability to display a smooth color "
"transition from black to\nred. It is recommended to run this test in True color mode. Several "
"color management solutions can cause\nsome roughness in the color transition, this is a "
"normal "
"phenomenon.",

" Gradient fill is an ideal test to check the monitor's ability to display a smooth color "
"transition from black to\ngreen. It is recommended to run this test in True color mode. "
"Several color management solutions can cause\nsome roughness in the color transition, this is "
"a normal phenomenon.",

" Gradient fill is an ideal test to check the monitor's ability to display a smooth color "
"transition from black to\nblue. It is recommended to run this test in True color mode. "
"Several "
"color management solutions can cause\nsome roughness in the color transition, this is a "
"normal "
"phenomenon.",

" Gradient fill is an ideal test to check the monitor's ability to display a smooth color "
"transition from black to\norange. It is recommended to run this test in True color mode. "
"Several color management solutions can cause\nsome roughness in the color transition, this is "
"a normal phenomenon.",

" Gradient fill is an ideal test to check the monitor's ability to display a smooth color "
"transition from black to\ngray. It is recommended to run this test in True color mode. "
"Several "
"color management solutions can cause\nsome roughness in the color transition, this is a "
"normal "
"phenomenon.",

"Hue Saturation Lightness palette.",

"Hue Saturation Lightness palette roteted by 180*."};

static QString color_test_ua[] {

" Цей суцільно червоний екран ідеально підходить для перевірки однорідності кольорів вашого\n"
"монітора. Користувачі рідкокристалічних моніторів також можуть перевірити область "
"відображення на\nпнаявність мертвих (завжди темних) червоних субпікселів.\n Невеликі дефекти "
"характерні для РК-моніторів, будь ласка, зверніться до посібника користувача, щоб\nпідіznатися "
"про політику виробника щодо \"мертвих пікселів\".",

" Цей суцільний зелений екран ідеально підходить для перевірки однорідності кольорів вашого\n"
"монітора. Користувачі рідкокристалічних дисплейв також можуть перевірити область відображення "
"на\nпнаявність мертвих (завжди темних) зелених субпікселів.\n Невеликі дефекти характерні "
"для РК-моніторів, будь ласка, зверніться до посібника користувача, щоб\nпідіznатися про "
"політику виробника щодо \"мертвих пікселів\".",

" Цей суцільний синій екран ідеально підходить для перевірки однорідності кольорів вашого\n"
"монітора. Користувачі рідкокристалічних дисплейв також можуть перевірити область відображення "
"на\nпнаявність мертвих (завжди темних) синіх субпікселів.\n Невеликі дефекти характерні для "
"РК-моніторів, будь "
"ласка, зверніться до посібника користувача, щоб\nпідіznатися про політику виробника щодо "
"\\"мертвих пікселів\".",

" Цей суцільний помаранчевий екран ідеально підходить для перевірки однорідності кольорів "
"вашого\nмонітора.",

" Повністю білий екран для перевірки яскравості та рівномірності підсвічування РК-монітора, "
"а також\nдля пошуку \"мертвих\"(завжди темних) пікселів.\n Невеликі дефекти є звичайним "
"явищем для РК-моніторів, будь ласка, зверніться до посібника\nкористувача для ознайомлення з "
"політикою виробника щодо мертвих пікселів.",

" Повністю чорний екран для перевірки яскравості та рівномірності підсвічування РК-монітора, "
"а також\nдля пошуку \"мертвих\"(завжди темних) пікселів.\n Невеликі дефекти є звичайним "
"явищем для РК-моніторів, будь ласка, зверніться до посібника\nкористувача для ознайомлення з "
"політикою виробника щодо мертвих пікселів.",

" Перед проведенням цього тесту слід розмагнітити класичний (ЕПТ) монітор. Магнітні поля " "(наприклад, \пдинаміки) поблизу монітора можуть спричинити спотворення кольорів.\n Невеликі " "дефекти є звичайним явищем для РК-моніторів, будь ласка, зверніться до посібника\пкористувача " "для ознайомлення з політикою виробника щодо мертвих пікселів.",

" Градієнтна заливка - ідеальний тест для перевірки здатності монітора відображати плавний " "перехід\пвід чорного до червоного кольору. Рекомендується запускати цей тест у режимі " "\\"Справжній колір"\".\nДеякі рішення для керування кольором можуть спричинити деяку шорсткість " "у переході кольорів, це\пнормальне явище.",

" Градієнтна заливка - ідеальний тест для перевірки здатності монітора відображати плавний " "перехід\пвід чорного до зеленого кольору. Рекомендується запускати цей тест у режимі " "\\"Справжній колір"\".\nДеякі рішення для керування кольором можуть спричинити деяку шорсткість " "у переході кольорів, це\пнормальне явище.",

" Градієнтна заливка - ідеальний тест для перевірки здатності монітора відображати плавний " "перехід\пвід чорного до синього кольору. Рекомендується запускати цей тест у режимі " "\\"Справжній колір"\".\nДеякі рішення для керування кольором можуть спричинити деяку шорсткість " "у переході кольорів, це\пнормальне явище.",

" Градієнтна заливка - ідеальний тест для перевірки здатності монітора відображати плавний " "перехід\пвід чорного до оранжевого кольору. Рекомендується запускати цей тест у режимі " "\\"Справжній колір"\".\nДеякі рішення для керування кольором можуть спричинити деяку шорсткість " "у переході кольорів, це\пнормальне явище.",

" Градієнтна заливка - ідеальний тест для перевірки здатності монітора відображати плавний " "перехід\пвід чорного до сірого кольору. Рекомендується запускати цей тест у режимі " "\\"Справжній " "колір"\".\nДеякі рішення для керування кольором можуть спричинити деяку шорсткість у переході " "кольорів, це\пнормальне явище.",

"Палітра насичиних та легких відтінків.",

"Палітра насичиних та легких відтінків обернена на 180 градусів."};

```
static QString calibration_test_en[] {
    " Use this test screen to perform automatic screen calibration. LCD panels using digital "
    "(DVI) "
    "connector cables\ndo not require screen adjustments.",

    " This test screen helps you to select the right values for your display. First of all, set "
    "the "
    "brightness of your\monitor to the max, then decrease it until the color around the test "
    "patterns appears black, then adjust the\nccontrast until you are able to see all twenty shades "
    "of gray in both rows.",
```

" Use this test screen to perform basic screen geometry calibration, and to adjust advanced "
 "settings in the\ncase they are available in your monitor configuration menu.",

" Bring up your gamma correction controls. Adjust the overall gamma until the model and top "
 "sections of the\ngrey square on the right are the same brightness.",

" This calibration test screen is designed for classic (CRT) monitors to check their ability "
 "to "
 "position the\ncolor, green and blue electron beams accurately.",

" The test to calibrate the perfect screen resolution",

" Focus is not the only effect that reduces image sharpness. Take also a look at the "
 "convergence and\ndecrease the contrast if necessary. Use the focus test screen to search for "
 "differences between the\ncenter pattern and the corners.");

```
static QString calibration_test_ua[] {
    " Використовуйте цей тестовий екран для автоматичного калібрування екрана. РК-панелі,\nщо "
    "використовують кабелі з цифровим (DVI) роз'ємом, не потребують налаштування екрана.",
```

" Цей тестовий екран допоможе вам вибрати правильні значення для вашого дисплея. Перш\пза "
 "все, "
 "встановіть яскравість монітора на максимум, потім зменшуйте її до тих пір, поки\нколір "
 "навколо "

"тестових зразків не стане чорним, після чого відрегулюйте контрастність до тих пір,\nпоки не "\n"зможете побачити всі двадцять відтінків сірого в обох рядах.",

" Використовуйте цей тестовий екран для виконання базового калібрування геометрії екрана, а\n"також для налаштування додаткових параметрів, якщо вони доступні в меню конфігурації "\n"монітора.",

" Відкрийте елементи керування гамма-корекції. Налаштуйте загальну гаму так, щоб модель і\n"верхні секції сірого квадрата праворуч мали однакову яскравість.",

" Цей тестовий екран калібрування призначений для класичних (ЕПТ) моніторів, щоб перевірити\n"їхню здатність точно позиціонувати червоний, зелений і синій електронні промені.",

" Тест для калібрування ідеальної роздільної здатності екрана",

" Фокусування - не єдиний ефект, який знижує різкість зображення. Погляньте також на\n"конвергенцію і зменшіть контрастність, якщо це необхідно. Використовуйте екран перевірки\n"фокусування для пошуку відмінностей між центральною частиною зображення і кутами."};

```
#endif // LANGUAGE_H

about.h
#ifndef ABOUT_H
#define ABOUT_H

#include <QCheckBox>
#include <QDialog>
#include <QGroupBox>
#include <QLabel>
#include <QMessageBox>
#include <QPushButton>
#include <QSpinBox>
#include <QVBoxLayout>
#include <QWidget>

#include "../Language/language.h"

class About : public QDialog
{
    Q_OBJECT
public:
    explicit About(const bool english, QDialog *parent = nullptr)
    {
        QVBoxLayout *layout = new QVBoxLayout(this);
        QFont titleFont("Arial", 16, QFont::Bold);
        QFont productFont("Arial", 14, QFont::Bold);
        QFont infoFont("Arial", 14);
        setWindowTitle(english == true ? title_en[1] : title_ua[1]);
        QLabel *titleLabel = new QLabel(english == true ? about_en[0] : about_ua[0]);
        titleLabel->setAlignment(Qt::AlignCenter);
        titleLabel->setFont(titleFont);
        QLabel *productLabel = new QLabel(english == true ? about_en[1] : about_ua[1]);
        productLabel->setFont(productFont);
        QLabel *projectLabel = new QLabel(english == true ? about_en[2] : about_ua[2]);
        projectLabel->setFont(infoFont);
        QLabel *authorLabel = new QLabel(english == true ? about_en[3] : about_ua[3]);
        authorLabel->setFont(productFont);
        QLabel *nameLabel = new QLabel(english == true ? about_en[4] : about_ua[4]);
        nameLabel->setFont(infoFont);
        QLabel *versionLabel = new QLabel(english == true ? about_en[5] : about_ua[5]);
        versionLabel->setAlignment(Qt::AlignCenter);
        versionLabel->setFont(productFont);
        layout->addWidget(titleLabel);
        layout->addSpacing(10);
        layout->addWidget(productLabel);
        layout->addWidget(projectLabel);
        layout->addSpacing(20);
        layout->addWidget(authorLabel);
        layout->addWidget(nameLabel);
        layout->addWidget(new QLabel(""));
    }
};
```

```

layout->addWidget(versionLabel);
setLayout(layout);
setFixedSize(520, 300);
}
~About() {}
};

#endif // ABOUT_H

preference.h
#ifndef PREFERENCE_H
#define PREFERENCE_H

#include <QCheckBox>
#include <QDialog>
#include <QGroupBox>
#include <QLabel>
#include <QMessageBox>
#include <QPushButton>
#include <QSpinBox>
#include <QVBoxLayout>
#include <QWidget>

#include "../Language/language.h"

class Preference : public QDialog
{
    Q_OBJECT
public:
    explicit Preference(size_t &auto_timeout, bool &to_save, bool english, QDialog *parent = nullptr)
        : p_auto_test_{&auto_timeout}
        , p_save_{&to_save}
        , english_{english}
    {
        QVBoxLayout *layout = new QVBoxLayout(this);
        setWindowTitle(english == true ? title_en[2] : title_ua[2]);
        QGroupBox *automaticGroupBox = new QGroupBox(english == true ? preferences_en[2]
                                                       : preferences_ua[2]);
        spinBox = new QSpinBox;
        spinBox->setRange(4, 20);
        QLabel *label = new QLabel(english == true ? preferences_en[3] : preferences_ua[3]);
        QVBoxLayout *automaticLayout = new QVBoxLayout;
        automaticLayout->addWidget(label);
        automaticLayout->addWidget(spinBox);
        automaticGroupBox->setLayout(automaticLayout);
        spinBox->setValue(auto_timeout);

        QGroupBox *generalGroupBox = new QGroupBox(english == true ? preferences_en[0]
                                                       : preferences_ua[0]);
        checkBox = new QCheckBox(english == true ? preferences_en[1] : preferences_ua[1]);
        QVBoxLayout *generalLayout = new QVBoxLayout;
        checkBox->setChecked((to_save == true) ? true : false);
        generalLayout->addWidget(checkBox);
        generalGroupBox->setLayout(generalLayout);

        QPushButton *saveButton = new QPushButton(english == true ? preferences_en[4]
                                                       : preferences_ua[4]);
        connect(saveButton, &QPushButton::clicked, this, &Preference::saveButtonClicked);

        layout->addWidget(generalGroupBox);
        layout->addWidget(automaticGroupBox);
        layout->addWidget(saveButton);
    }

    setLayout(layout);
    setFixedSize(300, 230);
}

~Preference() { emit preferenceClosed(); }

signals:

```

```
void preferenceClosed();

private slots:
    void saveButtonClicked()
    {
        *p_auto_test_ = static_cast<size_t>(spinBox->value());
        *p_save_ = checkBox->isChecked();
        QMessageBox::information(this,
            english_ == true ? preferences_en[5] : preferences_ua[5],
            english_ == true ? preferences_en[6] : preferences_ua[6]);
        close();
    }

private:
    size_t *p_auto_test_;
    bool *p_save_;
    bool english_;
    QSpinBox *spinBox;
    QCheckBox *checkBox;
};

#endif // PREFERENCE_H
```