Дніпровський державний технічний університет
Кафедра «Програмне забезпечення систем»

КУРСОВА РОБОТА

з дисципліни _____*Об'єктно-орієнтоване програмування*_____

_____

на тему ____*Розробка програмного застосунку гри «Вгадай колір»*

Студента групи ПЗ-23-1ду
Спеціальності 121 – Інженерія програмного забезпечення
_____ **Олексій ЖУРБА**
Керівник старший викладач
_____ Анастасія ІСКАНДАРОВА-МАЛА
Національна шкала _____
Кількість балів: _____ Оцінка: ECTS _____

Члени комісії _____ _____
_____ _____
_____ _____

м. Кам'янське
2024

Дніпровський державний технічний університет

Факультет *Комп'ютерних технологій та енергетики*

Кафедра *Програмного забезпечення систем*

Спеціальність *121 Інженерія програмного забезпечення*

ЗАВДАННЯ

на курсову роботу

з дисципліни Об'єктно-орієнтоване програмування

здобувачу вищої освіти групи *ПЗ-23-1ду*

ПІП *Журбі Олексія Ростиславовичу*

Тема курсової роботи *Розробка програмного застосунку гри «Вгадай колір»*

Дата видачі завдання _____

Дата здавання _____

Завдання видав

керівник курсової роботи _____ Анастасія ІСКАНДАРОВА-МАЛА

Завдання прийняв до виконання

здобувач групи ПЗ-23-1ду _____ Олексій ЖУРБА

Міністерство освіти і науки України

Дніпровський державний технічний університет

Здобувач О. Р. Журба **групи** ПЗ-23-1ду

Тема роботи *Розробка програмного застосунку гри «Вгадай колір»*

Оцінка за роботу

Критерії оцінки	Максимальна оцінка	Оцінка, балів
ОЦІНКА ЕТАПУ ПІДГОТОВКИ РОБОТИ	**20 б.**	
Дотримання календарного плану	5 б.	
Відповідність роботи постановці задачі	5 б.	
Самостійність	5 б.	
Ініціативність	5 б.	
ОЦІНКА ЗА ЗВІТ І	**20 б.**	
Структура, об’єм, оформлення	5 б.	
Реферат	5 б.	
Вступ, постановка задачі, висновки, перелік посилань	5 б.	
Опис предметної області (задачі)	5 б.	
ОЦІНКА ЗА ЗВІТ ІІ	**40 б.**	
Логічна структура (класи, об’єкти)	10 б.	
Опис програмного продукту	10 б.	
Функціональна повнота	5 б.	
Інтерфейсна частина	5 б.	
Опис тестування	5 б.	
Керівництво користувача	5 б.	
ОЦІНКА ЗА ПРАКТИЧНУ РЕАЛІЗАЦІЮ	**10 б.**	
Степінь закінченості	5 б.	
Демонстрація	5 б.	
РІВЕНЬ СКЛАДНОСТІ (наявність спеціалізації)	**10 б.**	
РАЗОМ	**100 б.**	

Захист відбувся _____

Загальна оцінка _____

Керівник курсової роботи _____ Анастасія ІСКАНДАРОВА-МАЛА

РЕФЕРАТ

Курсова робота: 40 с., 12 рис., 6 джерел, 2 додатки.

Об’єкт дослідження: об’єктно-орієнтоване програмування, мова програмування С++

Мета роботи: здобуття теоретичних знань і практичних навичок у використанні сучасних систем візуального проектування програмних засобів для комп'ютерних систем управління; освоєння принципів і методів сучасних технологій програмування, здобуття професійних та практичних навичок у наукових дослідженнях та підготовка відповідного наукового звіту; закріплення знань, отриманих під час вивчення курсу «Об’єктно-орієнтоване програмування», через створення програмного додатку, що є програмною реалізацією навчальної гри «Вгадай колір»; Оволодіння методикою експериментальних досліджень, аналізом результатів та їх порівнянням з літературними даними, а також узагальненням і представленням отриманих результатів.

Методи дослідження: вивчення властивостей об’єктно-орієнтованого програмування, використання математичного інструментарію логіки, застосування алгоритмів та структур даних, аналіз вимог до програмного забезпечення.

В теоретичній частині проведено аналіз предметної області гри «Вгадай колір» і досліджено можливості застосування об’єктно-орієнтованого підходу у розробці відповідного програмного забезпечення.

Друга частина курсової роботи містить опис створеного у середовищі Qt за допомогою мови програмування С++ програмного додатка для навчальної гри "Вгадай колір".

КЛАСИ, ОБ’ЄКТИ, ПОЛІМОРФІЗМ, ІНКАПСІЛЯЦІЯ, СПАДКУВАННЯ, Qt, ВГАДАЙ КОЛІР

ЗМІСТ

ВСТУП

Дана робота спрямована на створення ігрового навчального програмного додатку «Вгадай колір» для одного гравця за допомогою крос-платформенного інструментарію розробки програмного забезпечення «Qt», використовуючи мову програмування C++. Проєкт реалізовано за допомогою об’єктно-орієнтованої парадигми програмування, суть якої полягає в реалізації програми як набору "об’єктів", які мають взаємозв’язки між собою.

Об’єктно-орієнтованого програмування це – одна із парадигм програмування, в її основі лежать принципи інкапсуляції, успадкування та поліморфізму. У контексті даної парадигми, управління процесом реалізації програми здійснюється через передачу повідомлень об’єктам. Таким чином, об’єкти визначаються спільно з трьома типами повідомлень, на які вони реагують під час виконання програми.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп’ютеру, ООП-програми можна вважати сукупністю об’єктів.

Розробка об’єктно-орієнтованих програм включає наступні кроки:

- Визначення основних об’єктів (класів), необхідних для розв’язку задачі.
- Визначення закритих даних (даних стану) для цих об’єктів.
- Визначення вторинних об’єктів та їх закритих даних.
- Визначення ієрархічної системи класів, яка представляє обрані об’єкти.
- Визначення ключових повідомлень, які обробляють об’єкти кожного класу.
- Розробка послідовності виразів для вирішення поставленої задачі.
- Розробка методів, які обробляють кожне повідомлення.
- очищення проєкту, тобто видалення всіх допоміжних проміжних матеріалів, що використовувалися під час проєктування.
- кодування, налагодження, компонування і тестування.

Об’єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-х роках, одночасно з посиленням дискусій про кризу програмного забезпечення. Через ускладнення апаратного та програмного забезпечення було дуже важко зберегти якість програм. Об’єкто-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми.

У сучасному світі ООП дуже часто використовують при розробці добре організованих програм.

Основною метою даної парадигми програмування є можливість кожного об’єкту отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об’єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

Поява ООП була зумовлена спостереженням, що комп’ютерні програми представляють собою опис дій, які виконуються над різними об’єктами. Ці об’єкти можуть бути різного типу, такими як графічні об’єкти або записи в базах даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області гри «Вгадай колір»

Кольори трапляються нам усюди, від природи до наших міст, людина почала використовувати їх ще з кам’яного віку, коли люди зображали події зі свого життя на стінах печери при цьому використовуючи природні барвники добути із рослин, але колірне різноманіття будо дуже скромним, пізніше кольорів стало більше та їх стали використовувати для мистецтва та релігії, але вони в своїй основі теж використовували рослинні барвники.

На сьогоднішній день барви використовують компоненти із хімічних з’єднань і застосовуються не тільки для мистецтва і релігії але й для маркетингу, галузевого машинобудування та ін.

Для представлення кольору у ЄВМ використовують кодування кольору, тобто представляють його за допомогою спеціального алгоритму як набір значень, одною із таких є модель RGB, основою якої є три кольори, Red, Green, Blue — червоний, зелений, синій, представленні відповідними числовими значеннями від 0 до 255. Ці значення відповідають за яскравість свого кольору, якщо встановлено 0 – чорний, якщо 255 – колір повністю виражений.

При змішуванні цих кольорів можна отримати будь який колір, тому саме вона стала однією з найрозповсюдженіших колірних моделей. Так наприклад, якщо змішати червоний та синій ми отримаємо фіолетовий колір, тобто R= 255, G = 0, B = 255, або при R= 255, G = 102, B = 0 ми отримаємо помаранчевий.

RGB часто використовують при створенні сайтів та програмних продуктів, тому розробнику ПЗ необхідно в достатній формі володіти знанням RGB, тим як і який колір описується для більш швидкого і якісного виконання поставленої задачі.

Перед початком створення сайту веб-дизайнер за допомогою відповідних інструментів створює макет сайту, наприклад Figma, де з геометричних примітивів та наборів властивостей створюють «прототип» майбутнього сайту.

Після узгодження із замовником, «прототип» надається веб-програмісту, який як раз і відповідає за реалізацію сайту.

Для створення сайтів часто задають кольори у даній моделі в каскадній таблиці стилів (CSS – Cascading Style Sheet), наприклад background-color: rgb(255, 102, 0), даний стиль задає фоновий колір батьківському HTML-елементу, в даному випадку помаранчевий. Тобто Можна задати будь-який колір будь-якому HTML-елементу, наприклад елементи інтерфейсу, «шапка» сайту, «тіло» сайту, «підвал» сайту, бокове меню, та багато іншого.

Окрім веб-програмування також цю модель використовують для створення програмних застосунків, також визначаючи кольори інтерфейсу, фону.

Як можна побачити на прикладі Веб-дизайну, Веб-програмування та розробки програмних застосунків дана колірна модель часто використовується і її знання необхідні для достатньої компетенції розробника.

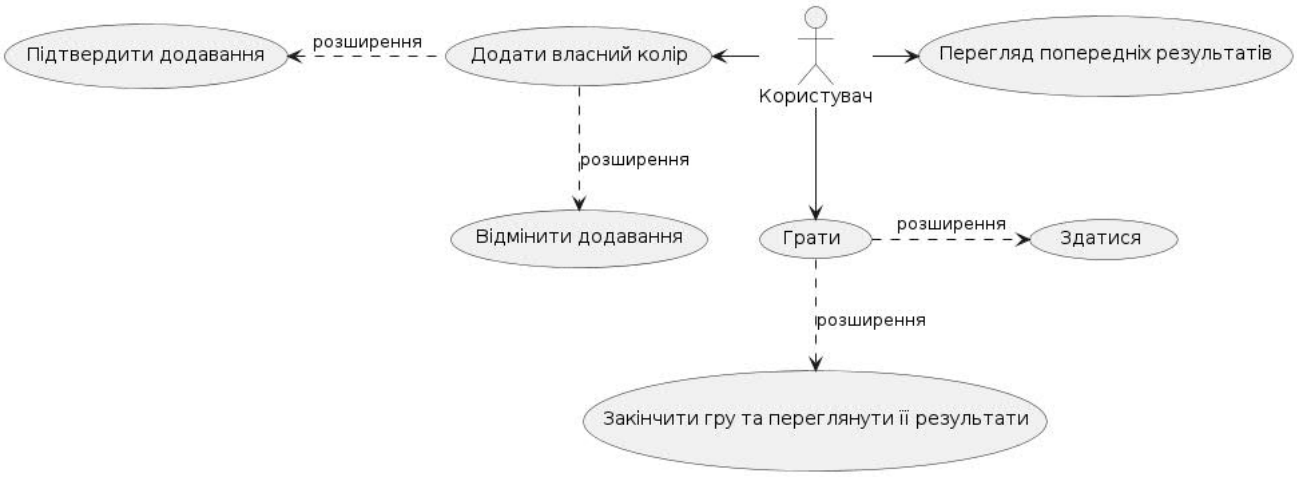


Рисунок 1.1 – Діаграма прецедентів

Програмний додаток гри «Вгадай колір» може бути встановлений на персональні комп’ютери під управлінням ОС Windows. В даній грі передбачена можливість додавання власного кольору до списку існуючих кольорів, перегляд результатів попередніх ігрових сесій. Результати після проходження гри записуються у відповідний json-файл. Якщо користувач бажає передчасно закінчити гру, тобто здатися, він може це зробити натисканням на кнопку з

написом «я пас». На рис. 1.1 представлено діаграму прецедентів для навчальної гри «вгадай колір», на якій зображанні актор «користувач» та його варіанти використання програмного додатку.

1.2 Програмний застосунок «Вгадай колір» для ОС Windows

Зараз для кожного приватного бізнесу, компанії, корпорацій потрібен свій сайт через який він матиме зв’язок з клієнтами, надавати послуги або інформацію. Враховуючи це можна зробити висновок – у сучасному світі Веб-сайти є невідмінною складовою бізнесу, тому важливо задовольнити даний попит.

Веб-розробники мають володіти усіма необхідними компетенціями щоб задовольнити потреби замовників, однією із таких компетенцій це розуміння колірних просторів, таких як CMYK, HSB, HSL, RGB, останній дуже часто використовується на практиці.

Існує багато ігор які пов’язані з кольорами, від дитячих, до ігор що підвищують компетентність веб-дизайнерів та веб-розробників, але жодна з них не надає налаштовувати набір кольорів, вони як правило генерують три випадкових значення від 0 до 255, натомість в грі «Вгадай колір» передбачено функціонал додавання власного кольору, це може бути корисним для того, коли потрібно запам’ятати певну палітру кольорів, бо існують різні за стилем та спрямованістю сайти які вимагають використання певної палітри.

Навчальна гра «Вгадай колір» є інструментом для вивчення кодів кольорів, що задані у RGB-просторі. Гра відображає квадрат з кольором який потрібно вгадати користувачу вказавши його у три відповідних поля для вводу, як тільки користувач натискає на кнопку «Перевір мене!» гра перевіряє наближеність кольору що вів користувач до еталонного на основі цього вираховуються оцінка та виводиться у окремому вікні разом із двома квадратами еталонного та користувацького кольору.

Вікно гри має наступний вигляд, наведений на рис. 1.2.

Рисунок 1.2 – Вікно гри «Вгадай колір»

Гра реалізована за допомогою мови програмування C++, де використовується базовий функціонал об’єктно-орієнтованого програмування в основі реалізації якого лежать поняття об’єкти та класи.

Поняття класу є ключовим розвитком модульного підходу в програмуванні. Клас поєднує структури даних та функції для їх обробки. Використання класу відбувається лише через його інтерфейс, деталі реалізації приховані, тобто інкапсульовані. Ідея класів відображає структуру об’єктів реального світу, де кожен об’єкт має набір характеристик (властивостей) та поведінку.

Клас являє собою користувацький тип даних. У класі визначаються властивості предмета або процесу у вигляді полів даних, а також функції для роботи з цими даними. Створений тип даних має ті самі властивості, що й стандартні типи.

В ООП основою керування програмою є передача повідомлень об’єктам. Об’єкти визначаються разом з повідомленнями, на які вони повинні реагувати під час виконання програми. Це відрізняється від процедурного програмування, де структури даних передаються в процедури як параметри. Об’єктно-орієнтована програма складається з об’єктів – фрагментів коду, що обробляють дані та взаємодіють через певні інтерфейси.

ООП є новим підходом до створення програм, що виник у міру ускладнення програмного забезпечення. Нині ООП домінує в галузі прикладного програмування (Java, C#, C++, JavaScript тощо), хоча в системному програмуванні ще переважає процедурна парадигма. Проте вплив ООП відчутний і на рівні взаємодії системного та прикладного програмного забезпечення операційних систем.

Об’єктно-орієнтоване програмування (ООП) - це парадигма, в якій програма розглядається як множина взаємодіючих між собою об’єктів. ООП використовує технології попередніх парадигм, такі як успадкування, модульність, поліморфізм та інкапсуляцію.

На відміну від традиційного погляду, коли програма вважалася набором підпрограм або інструкцій, в ООП програма - це сукупність об’єктів. Згідно з цією парадигмою, кожен об’єкт може отримувати повідомлення, обробляти дані та надсилати повідомлення іншим об'єктам.

ООП дозволяє програмісту моделювати об'єкти певної предметної області шляхом програмування їх змісту та поведінки в межах класу. Конструкція "клас" забезпечує механізм інкапсуляції для реалізації абстрактних типів даних і приховування реалізації від користувача.

Ключові поняття ООП:

Об’єкти – основні сутності в ООП. Об’єкт являє собою екземпляр класу і має стан (властивості/атрибути) та поведінку (методи). Властивості визначають характеристики об'єкта, а методи - дії, які він може виконувати.

Класи – це користувацькі типи даних, які визначають структуру і поведінку об'єктів. Клас діє як шаблон або прототип, за яким створюються екземпляри

об'єктів. Він містить оголошення властивостей і методів, що визначають загальні характеристики та функціональність об'єктів цього типу.

Методи – це функції, оголошені всередині класу, які реалізують поведінку об'єктів відповідного типу. Вони можуть отримувати доступ і маніпулювати даними, збереженими у властивостях об’єкта. Методи дозволяють інкапсулювати логіку разом з даними, що належать об’єкту.

Інкапсуляція – це об’єднання даних та методів їх обробки, а також приховування внутрішньої реалізації. Користувач взаємодіє лише з інтерфейсом класу, а конкретна реалізація залишається невидимою. Це підвищує рівень абстракції програми, оскільки деталі реалізації приховані. Завдяки інкапсуляції можна змінити реалізацію класу без впливу на решту програми, якщо зовнішній інтерфейс залишився незмінним. Таким чином забезпечується інкапсуляція даних та логіки в межах класу.

Наслідування - це механізм, за допомогою якого один об’єкт (клас) здобуває властивості іншого об'єкта (класу). Це дозволяє створювати нові класи, базуючись на існуючих. Новий клас, який успадковує властивості від іншого класу, називається похідним, а клас, від якого успадковується - батьківським або базовим. Похідний клас успадковує всі доступні методи та властивості базового класу і може додавати свої нові властивості та методи або перевизначати (перекривати) успадковані методи власними реалізаціями.

Існує також поняття множинного успадкування, за якого клас може успадковувати властивості відразу від кількох батьківських класів.

Успадкування лежить в основі створення ієрархії класів та забезпечує можливість повторного використання існуючого коду. Це є однією з ключових концепцій об’єктно-орієнтованого програмування.

Абстракція – це принцип, який дозволяє приховати внутрішні деталі реалізації об'єкта та надати користувачеві лише необхідний інтерфейс для взаємодії з ним. Абстракція забезпечує спрощення складних систем, оскільки дозволяє ігнорувати несуттєві деталі та концентруватися на найважливіших

аспектах. Вона тісно пов'язана з інкапсуляцією, адже приховування реалізації є її ключовим аспектом.

Поліморфізм – це функціональна можливість, яка дозволяє використовувати одне ім’я для іменування схожих за змістом дій в різних класах ієрархії та обирати необхідні дії під час виконання програми. Тип даних з віртуальними функціями називається поліморфним типом. Для практичної реалізації поліморфізму в мові програмування тип об’єктів повинен бути поліморфним. Якщо працювати безпосередньо з об’єктами, їх тип відомий компілятору наперед, і поліморфізм не потрібен.

Щоб реалізувати поліморфізм, компілятор повинен зберігати додаткову інформацію про тип об’єкта в кожному об'єкті поліморфної ієрархії класів. Ця інформація необхідна для визначення коректної версії віртуальної функції, яку слід викликати.

Поліморфізм дозволяє обробляти об’єкти різних класів однаковим чином, використовуючи загальний інтерфейс. Це підвищує гнучкість та повторне використання коду, оскільки код, написаний для базового класу, може працювати з об'єктами похідних класів без модифікацій. Ці ключові поняття разом забезпечують модульність, повторне використання коду, розширюваність та гнучкість програмного забезпечення, що розробляється з використанням парадигми об’єктно-орієнтованого програмування.

2 ОПИС ПРОГРАМИ

2.1 Постановка задачі

Розробити навчальну гру «Вгадай колір» для вивчення колірного простору RGB. При її розробці використовуючи парадигму ООП реалізувати програму, яка у ігровій формі розвиває розуміння RGB-простору. Необхідно виконати реалізацію мовою C++ у вигляді об’єктів, що взаємодіють, кожен з них є екземпляром свого класу, в свою чергу класи є членами своєї ієрархії наслідування.

Також програма відповідно повинна включати в себе використання інкапсуляції та поліморфізму

2.2 Діаграма класів

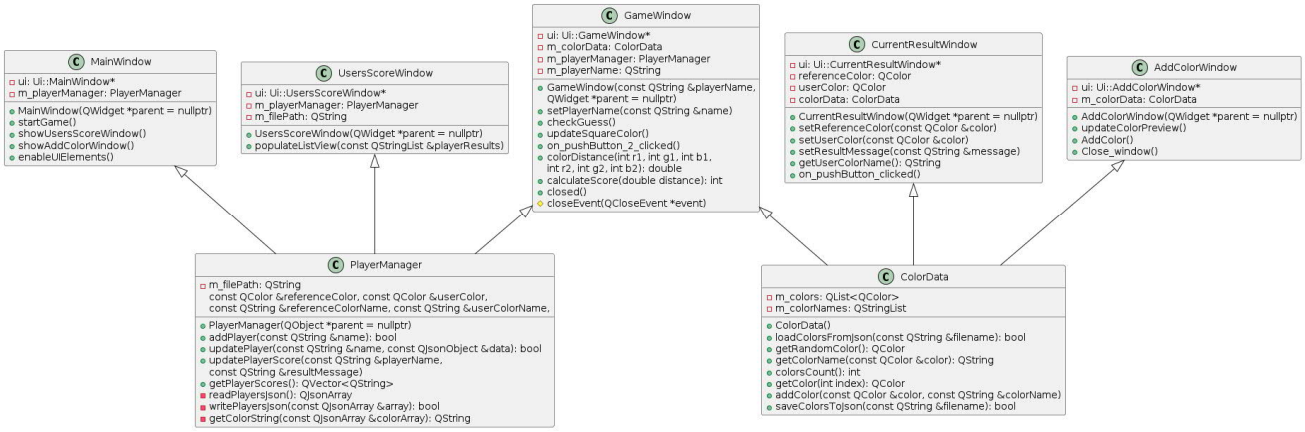


Рисунок 2.1 – Діаграма класів

Виходячи із постановки завдання і предметної області, можна виділити наступний набір можливих класів, які наведені на рис. 2.1

2.3 Опис класів та об’єктів

Опис методів та полів класу MainWindow, який є нащадком PlayerManager:

- ui: Вказівник на об’єкт класу Ui::MainWindow, який представляє користувацький інтерфейс головного вікна.
- m_playerManager: Об’єкт класу PlayerManager, який керує даними про гравців та їхні результати.
- MainWindow(QWidget *parent = nullptr): Конструктор класу. Ініціалізує об'єкт MainWindow, встановлює користувацький інтерфейс, налаштовує зв'язки сигналів і слотів та завантажує зображення.
- startGame(): Слот, який викликається при натисканні кнопки «Почати гру». Перевіряє введене ім’я гравця, створює новий об'єкт GameWindow і відкриває його.
- showUsersScoreWindow(): Слот, який викликається при виборі пункту меню «Показати таблицю результатів». Відкриває вікно UsersScoreWindow.
- showAddColorWindow(): Слот, який викликається при виборі пункту меню «Додати власний колір». Відкриває вікно AddColorWindow.
- enableUIElements(): Слот, який викликається при закритті вікна GameWindow. Активує елементи інтерфейсу головного вікна.
- ~MainWindow(): Деструктор класу. Звільняє пам’ять, зайняту об’єктом ui.

Опис методів та полів класу GameWindow, який є нащадком PlayerManager та ColorData:

- ui: Вказівник на об’єкт класу Ui::GameWindow, який представляє користувацький інтерфейс вікна гри.
- m_colorData: Об’єкт класу ColorData, який містить дані про кольори для гри.
- m_playerManager: Об’єкт класу PlayerManager, який керує даними про гравців та їхні результати.
- m_playerName: Рядок, який містить ім’я поточного гравця.
- setPlayerName(const QString &name): Встановлює ім’я гравця.

- GameWindow(const QString &playerName, QWidget *parent = nullptr): Конструктор класу. Ініціалізує об’єкт GameWindow, встановлює користувацький інтерфейс, завантажує дані про кольори та налаштовує зв’язки сигналів і слотів.
- on_pushButton_2_clicked(): Слот, який викликається при натисканні кнопки «Я пас.». Закриває вікно гри.
- updateSquareColor(): Оновлює колір квадрата у вікні гри на випадковий колір з набору даних.
- closeEvent(QCloseEvent *event): Переважений метод, який викликається при закритті вікна гри. Випромінює сигнал closed().
- checkGuess(): Перевіряє, наскільки введений користувачем колір близький до кольору у вікні гри. Відображає відповідне повідомлення про результат відкриваючи вікно CurrentResultWindow та оновлює рахунок гравця у players_score.json.
- colorDistance(int r1, int g1, int b1, int r2, int g2, int b2): Обчислює відстань між двома кольорами в просторі RGB за допомогою формули Евклідової відстані.
- ~GameWindow(): Деструктор класу. Звільняє пам’ять, зайняту об’єктом ui.

Опис методів та полів класу CurrentResultWindow, який є нащадком ColorData:

- ui: Вказівник на об’єкт класу Ui::CurrentResultWindow, який представляє користувацький інтерфейс вікна результатів.
- referenceColor: Об’єкт класу QColor, який зберігає еталонний колір, який потрібно було вгадати.
- userColor: Об’єкт класу QColor, який зберігає колір, вгаданий користувачем.
- colorData: Об’єкт класу ColorData, який містить дані про назви кольорів.
- CurrentResultWindow(QWidget *parent = nullptr): Конструктор класу. Ініціалізує об’єкт CurrentResultWindow та завантажує дані про кольори.

- ~CurrentResultWindow(): Деструктор класу. Звільняє пам'ять, зайняту об’єктом ui.
- setReferenceColor(const QColor &color): Встановлює еталонний колір, який потрібно було вгадати, та оновлює відповідні елементи інтерфейсу.
- setUserColor(const QColor &color): Встановлює колір, вгаданий користувачем, та оновлює відповідні елементи інтерфейсу.
- setResultMessage(const QString &message): Встановлює текстове повідомлення про результат гри.
- getUserColorName(): Повертає назву кольору, вгаданого користувачем.
- on_pushButton_clicked(): Слот, який викликається при натисканні кнопки «OK». Закриває вікно результатів.

Опис методів та полів класу UsersScoreWindow, який є нащадком PlayerManager:

- ui: Вказівник на об’єкт класу Ui::UsersScoreWindow, який представляє користувацький інтерфейс вікна таблиці результатів.
- m_playerManager: Об’єкт класу PlayerManager, який керує даними про гравців та їхні результати.
- m_filePath: Рядок, який містить шлях до файлу з даними про результати гравців.
- UsersScoreWindow(QWidget *parent = nullptr): Конструктор класу. Ініціалізує об'єкт UsersScoreWindow, встановлює користувацький інтерфейс, налаштовує стилі елементів інтерфейсу та заповнює таблицю результатів даними з файлу.
- ~UsersScoreWindow(): Деструктор класу. Звільняє пам’ять, зайняту об’єктом ui.

Опис методів та полів класу AddColorWindow, який є нащадком ColorData:

- ui: Вказівник на об’єкт Ui::AddColorWindow, який представляє користувацький інтерфейс вікна для додавання кольору.

- m_colorData: Об’єкт класу ColorData, який містить дані про кольори для гри.
- AddColorWindow(QWidget *parent = nullptr): Конструктор класу. Ініціалізує об’єкт AddColorWindow, встановлює користувацький інтерфейс, завантажує дані про кольори та налаштовує зв’язки сигналів і слотів.
- ~AddColorWindow(): Деструктор класу. Звільняє пам’ять, зайняту об’єктом ui.
- Close_window(): Слот, який викликається при натисканні кнопки «Закрити». Закриває вікно для додавання кольору.
- updateColorPreview(): Слот, який викликається при зміні значення одного з полів введення для компонентів кольору (червоний, зелений, синій). Оновлює колір у вікні перегляду.
- AddColor(): Слот, який викликається при натисканні кнопки «Додати». Перевіряє введені дані на коректність, запитує підтвердження у користувача та додає новий колір до списку кольорів, якщо всі перевірки пройдені успішно. Зберігає оновлений список кольорів у файл.

Опис методів та полів класу ColorData, нащадками якого є GameWindow, CurrentResultWindow, AddColorWindow:

- m_colors: Список (QList) об’єктів QColor, який містить кольори.
- m_colorNames: Список (QStringList) рядків, який містить назви кольорів.
- ColorData(): Конструктор класу. Ініціалізує порожній список кольорів і назв.
- loadColorsFromJson(const QString &filename): Завантажує список кольорів і їх назв з JSON-файлу. Повертає true, якщо завантаження пройшло успішно, і false у разі помилки.
- getRandomColor() const: Повертає випадковий колір зі списку m_colors.
- getColorName(const QColor &color) const: Повертає назву кольору за його значенням кольору. Якщо колір не знайдено в списку, повертає рядок «Невизначений колір».
- colorsCount() const: Повертає кількість кольорів у списку m_colors.

- getColor(int index) const: Повертає колір за його індексом у списку m_colors. Якщо індекс недійсний, повертає порожній об’єкт QColor.
- addColor(const QColor &color, const QString &colorName): Додає новий колір і його назву до списків m_colors і m_colorNames відповідно.
- saveColorsToJson(const QString &filename) const: Зберігає поточний список кольорів і їх назв у JSON-файл. Повертає true, якщо збереження пройшло успішно, і false у разі помилки.

Опис методів та полів класу PlayerManager, нащадками якого є MainWindow, GameWindow, UsersScoreWindow:

- m_filePath: Рядок, який містить шлях до JSON-файлу, де зберігаються дані про гравців.
- PlayerManager(QObject *parent = nullptr): Конструктор класу. Ініціалізує об’єкт PlayerManager та встановлює шлях до файлу даних гравців.
- addPlayer(const QString &name): Додає нового гравця до списку гравців у файлі. Повертає true, якщо додавання пройшло успішно, і false, якщо гравець з таким іменем вже існує.
- updatePlayerScore(const QString &playerName, const QColor &referenceColor, const QColor &userColor, const QString &referenceColorName, const QString &userColorName, const QString &resultMessage): Оновлює результат гравця з вказаним іменем, додаючи інформацію про еталонний колір, вгаданий колір, назви кольорів та повідомлення про результат.
- getPlayerScores(): Повертає вектор рядків, кожен з яких містить інформацію про гравця (ім'я, еталонний колір, назву еталонного кольору, вгаданий колір, назву вгаданого кольору та повідомлення про результат).
- getColorString(const QJsonArray &colorArray): Допоміжний метод, який перетворює масив JSON з компонентами кольору (червоний, зелений, синій) у рядок у форматі «червоний зелений синій».

2.4 Застосування наслідування, інкапсуляції та поліморфізму

Інкапсуляція – це один із принципів об’єктно-орієнтованого програмування, який полягає в об’єднанні даних та методів, які працюють з ними, в класі. Основна ідея полягає в тому, щоб приховати деталі реалізації від користувача і надати зовні лише визначений інтерфейс взаємодії з об’єктом. Це полегшує розробку програмного забезпечення, оскільки компоненти можуть бути розроблені, тестовані та доповнені незалежно один від одного. Правильно інкапсульовані компоненти є більш зрозумілими та легше налагоджуються, що спрощує підтримку програми.

У мові програмування C++, інкапсуляція реалізована за допомогою класів, пакетів та модифікаторів доступу. Класи дозволяють об'єднувати дані та методи, що працюють з ними, в одному місці. Пакети групують класи за певним критерієм, спрощуючи організацію програми. Модифікатори доступу визначають рівень доступу до класів, їхніх полів та методів, що дозволяє контролювати, як дані та функціональність класу використовуються в інших частинах програми.

Синтаксисом мови програмування C++ передбачено три модифікатори доступу:

- public: Надає повний доступ до поля або методу класу з будь-якого іншого пакету або класу.
- protected: Забезпечує доступ до поля або методу тільки для класів у тому ж пакеті і для нащадків класу, який містить це поле або метод.
- private: Обмежує доступ до поля або методу лише всередині самого класу, де вони оголошені.

Наприклад у навчальній грі «Вгадай колір» інкапсульованими є поля ColorData m_colorData, PlayerManager m_playerManager, та QString m_playerName; класу GameWindow:

```
public:
   explicit GameWindow(const QString &playerName, QWidget *parent = nullptr);
   void setPlayerName(const QString &name);
   ~GameWindow(); private:
private:
   Ui::GameWindow *ui;
```

```
ColorData m_colorData;
PlayerManager m_playerManager;
QString m_playerName;
```

Наступним чином реалізовано метод класу PlayerManager для встановлення ім’я користувача:

```
void GameWindow::setPlayerName(const QString &name) {
    m_playerName = name;
}
```

Наслідування є одним з фундаментальних принципів об’єктно-орієнтованого програмування, бо дозволяє створювати структуровану ієрархію об’єктів. Цей принцип дозволяє використовувати батьківський клас, який визначає характеристики та поведінку для його нащадків. Пізніше цей клас може бути успадкований іншими класами, тобто його нащадками, які в свою чергу можуть додавати свої унікальні характеристики або змінювати поведінку батьківського класу.

Таким прикладом є клас GameWindow, що успадковує клас QMainWindow, додавши або змінюючи певні властивості та методи базового класу.

```
class GameWindow : public QMainWindow
```

У купі з наслідуванням необхідно розгадати й поняття поліморфізму, бо другий витикає з властивостей першого, коли в нас є один інтерфейс, але декілька різних реалізацій, прикладом цього може слугувати метод closeEvent що є віртуальним методом, який перевизначається з базового класу QMainWindow. Він викликається завжди, коли вікно закривається, наприклад, коли користувач натискає кнопку «Я пас.» у вікні гри.

```
void GameWindow::closeEvent(QCloseEvent *event)
{
    emit closed();
    QMainWindow::closeEvent(event);
}
```

2.5 Оцінювання гравця

Оцінювання гравця реалізовано за допомогою двох методів checkGuess() та colorDistance() класу GameWindow, перша приймає колір введений користувачем, потім викликається функція colorDistance() в яку передається еталонний колір та який що ввів користувач.

```
void GameWindow::checkGuess() {
    int userRed = ui->spinBox->value();
    int userGreen = ui->spinBox_2->value();
    int userBlue = ui->spinBox_3->value();

    QColor selectedColor = ui->frame->palette().color(QPalette::Window);

    double distance = colorDistance(selectedColor.red(), selectedColor.green(),
    selectedColor.blue(), userRed, userGreen, userBlue);

    QString resultMessage;
    if (distance <= 1) {
       resultMessage = "Відмінно!";
    } else if (distance <= 20 && distance > 1) {
       resultMessage = "Дуже близько!";
    } else if (distance <= 60 && distance > 20) {
       resultMessage = "Близько!";
    } else if (distance <= 100 && distance > 60) {
       resultMessage = "50/50";
    } else {
       resultMessage = "Спробуй ще раз.";
    }

    QColor referenceColor = ui->frame->palette().color(QPalette::Window);
    QColor userColor(userRed, userGreen, userBlue);

    QString referenceColorName = m_colorData.getColorName(selectedColor);
    QString userColorName = m_colorData.getColorName(userColor);

    CurrentResultWindow *resultWindow = new CurrentResultWindow;
    resultWindow->setReferenceColor(selectedColor);
    resultWindow->setUserColor(userColor);
    resultWindow->setResultMessage(resultMessage);
    resultWindow->show();

    m_playerManager.updatePlayerScore(m_playerName, referenceColor, userColor,
    referenceColorName, userColorName, resultMessage);
    this->close();
}
```

За допомогою формули Евклідової відстані знаходиться відстань кольору користувача від еталонного, та повертається числове значення відстані на основі якого визначається оцінка і потім результати виводяться у новому вікні поточних результатів (Рис. 2.2), а результати гри записуються у відповідний файл.

```
double GameWindow::colorDistance(int r1, int g1, int b1, int r2, int g2, int b2)
{
    return sqrt(pow(r2 - r1, 2) + pow(g2 - g1, 2) + pow(b2 - b1, 2));
}
```

Рисунок 2.2 – Вікно поточних результатів

2.6 Збереження результатів гри

Після завершення гри її результати зберігаються у відповідний файл з розширенням .json за допомогою методу updatePlayerScore() класу PlayerManager, якому передаються аргументи через метод checkGuess() класу GameWindow.

```
void GameWindow::checkGuess() {
…
    m_playerManager.updatePlayerScore(m_playerName, referenceColor, userColor,
    referenceColorName, userColorName, resultMessage);
…
}
```

Після того як у метод updatePlayerScore були передані аргументи він оновлює відведені під результати поля об’єктів гравців у players_score.json.

```
void PlayerManager::updatePlayerScore(const QString &playerName, const QColor
&referenceColor, const QColor &userColor, const QString &referenceColorName, const
QString &userColorName, const QString &resultMessage) {
```

```
…

for (QJsonValueRef playerValue : playersArray) {
    QJsonObject playerObject = playerValue.toObject();
    QString playerNameInJson = playerObject["name"].toString().trimmed();

    if (playerNameInJson.compare(playerName, Qt::CaseInsensitive) == 0) {
        playerFound = true;

        playerObject["score"] = resultMessage;

        QJsonArray userColorArray;
        userColorArray.append(userColor.red());
        userColorArray.append(userColor.green());
        userColorArray.append(userColor.blue());
        playerObject["user_color"] = userColorArray;

        QJsonArray referenceColorArray;
        referenceColorArray.append(referenceColor.red());
        referenceColorArray.append(referenceColor.green());
        referenceColorArray.append(referenceColor.blue());
        playerObject["reference_color"] = referenceColorArray;
        playerObject["user_color_name"] = userColorName;
        playerObject["reference_color_name"] = referenceColorName;
        playerValue = playerObject;

        break;
    }
}

…
}
```

2.7 Перегляд попередніх результатів гри

Збережені результати попередніх ігрових сесій користувач може переглянути у вікні «Попередні результати» вибравши відповідний пункт у меню головного вікна (рис. 2.3).

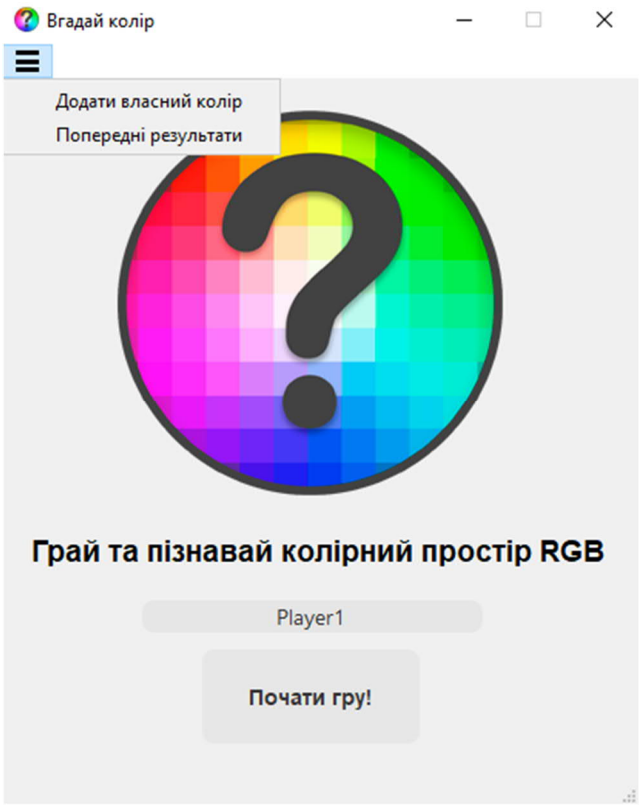


Рисунок 2.3 – Головне вікно гри

Результати виводяться в текстове поле у вигляді відформатованого тексту, для цього необхідно зчитати дані гравців з players_score.json за допомогою двох методів getPlayerScores() та getColorString(), перша виконує зчитування даних, а друга викликається першою для перетворення чисельних значень RGB у рядок.

```
QVector<QString> PlayerManager::getPlayerScores() {
    …

    QJsonArray playersArray = root["players"].toArray();

    for (const QJsonValue &playerValue : playersArray) {
        QJsonObject playerObject = playerValue.toObject();
        QString playerName = playerObject["name"].toString();
        QString referenceColor = getColorString(playerObject["reference_color"].toArray());
        QString referenceColorName = playerObject["reference_color_name"].toString();
        QString userColor = getColorString(playerObject["user_color"].toArray());
        QString userColorName = playerObject["user_color_name"].toString();
        QString score = playerObject["score"].toString();

        QString playerInfo = QString("%1, %2, %3, %4, %5, %6, %7").arg(playerName,
        referenceColor, referenceColorName, userColor, userColorName, score);
        playerScores.append(playerInfo);
    }

    return playerScores;
```

```
}

QString PlayerManager::getColorString(const QJsonArray &colorArray) {
    if (colorArray.size() != 3)
        return QString();

    int red = colorArray[0].toInt();
    int green = colorArray[1].toInt();
    int blue = colorArray[2].toInt();

    return QString("%1 %2 %3").arg(red).arg(green).arg(blue);
}
```

Для виведення та форматування результатів (рис. 2.4) у конструкторі класу UsersScoreWindow використовується наступний цикл for:

```
for (const QString &score : playerScores) {
        QStringList scoreParts = score.split(",");
        QString playerName = scoreParts[0].trimmed();
        QString referenceColor = scoreParts[1].trimmed();
        QString referenceColorName = scoreParts[2].trimmed();
        QString userColor = scoreParts[3].trimmed();
        QString userColorName = scoreParts[4].trimmed();
        QString resultMessage = scoreParts[5].trimmed();


        QString formattedScore = QString("\n%1:\n Еталонний колір: %2 (%3);\n Колір
        гравця: %4 (%5);\n Результат гри: %6")
        .arg(playerName)
        .arg(referenceColor)
        .arg(referenceColorName)
        .arg(userColor)
        .arg(userColorName)
        .arg(resultMessage);

        ui->plainTextEdit->appendPlainText(formattedScore);
    }
```

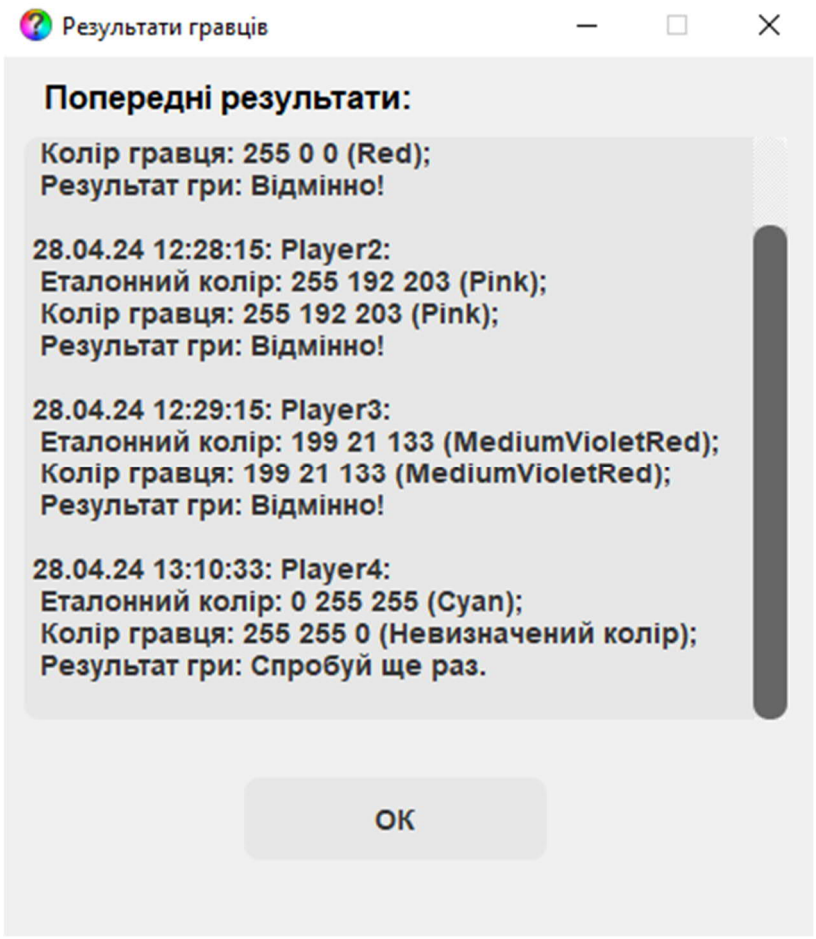


Рисунок 2.4 – Вікно попередніх результатів

2.8 Додавання власного кольору

Для додавання користувацького кольору необхідно у головному вікні програми (рис. 2.3) обрати пункт «Додати власний колір», після чого відкриється нове вікно (рис. 2.5) схоже на вікно процесу гри (рис. 1.2), але користувачеві буде запропоновано ввести власний колір (RGB та його назву латинськими літерами), якщо всі значення введені коректно програма попросить підтвердження дії, після чого буде додано колір до списку існуючих.

Рисунок 2.5 – Вікно додавання кольору

Тоді при наступних ігрових сесіях він може випасти гравцю у якості еталонного кольору (рис. 2.6).

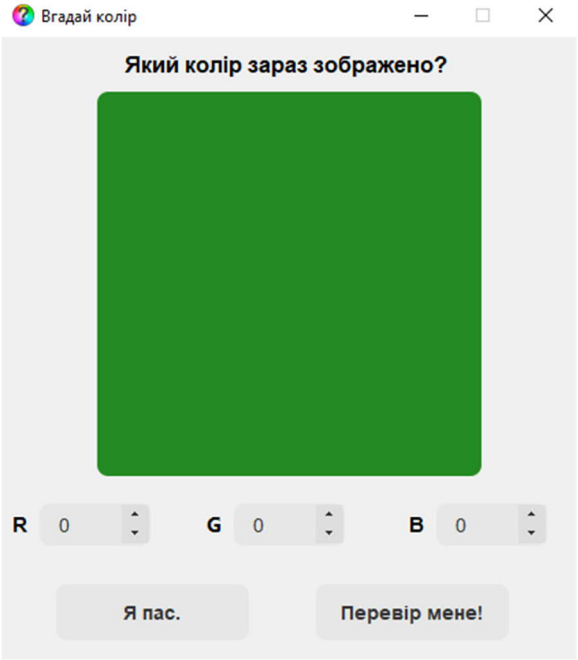


Рисунок 2.6 – Вікно гри з новим еталонним кольором

2.9 Процес гри

Як тільки користувач запустить програму перед ним з’явиться головне вікно, з логотипом гри меню та кнопкою з написом «Почати гру!», після натискання якої починається сам ігровий процес, користувач повинен вказати значення складових кольору RGB у відповідних полях які повинні відповідати значенням еталонного кольору яким зображено квадрат на полями вводу (рис 2.7).

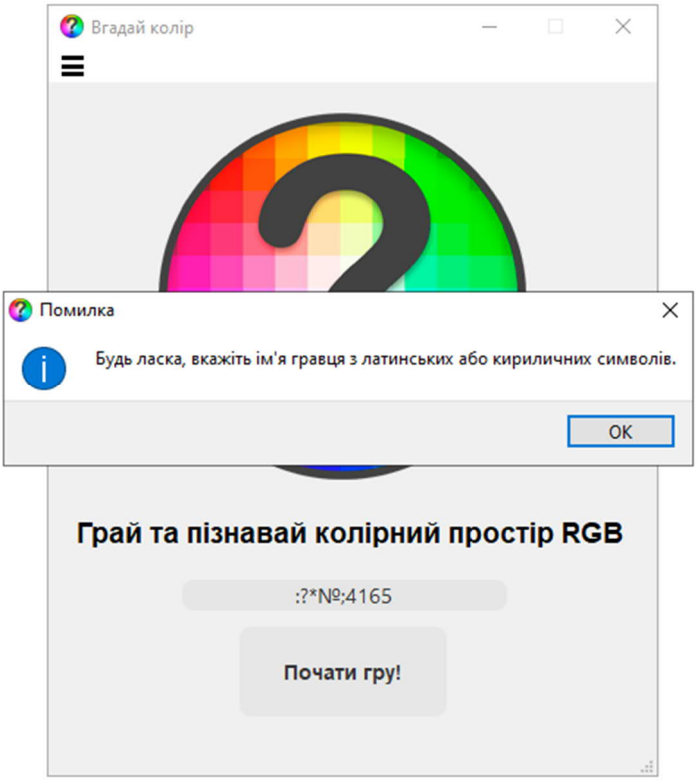
Рисунок 2.7 – Вікно гри

Існує також можливість вводити значення прокручуванням колесо миші, що наведена курсором на відповідне поле, далі користувач може здатися натиснувши на кнопку з написом «Я пас.» або перевірити себе натиснувши на «Перевір мене», у останньому випадку гри виведе результати даної гри у окремому вікні (рис. 2.2).

2.10 Тестування та відлагодження програмного застосунку гри

Шляхом введенні випадкових рядкових значень у поле вказання ім’я користувача головного вікна проявило коректну поведінку при введені недопустимих імен користувачів (рис 2.8).

Рисунок 2.8 – Головне вікно гри при введені некоректного ім’я гравця



Аналогічним шляхом здійснювалось тестування перевірки коректності вводу назви кольору (рис. 2.9). Також при введені кольору, що вже існує гра реагує наступним чином(рис. 2.10).

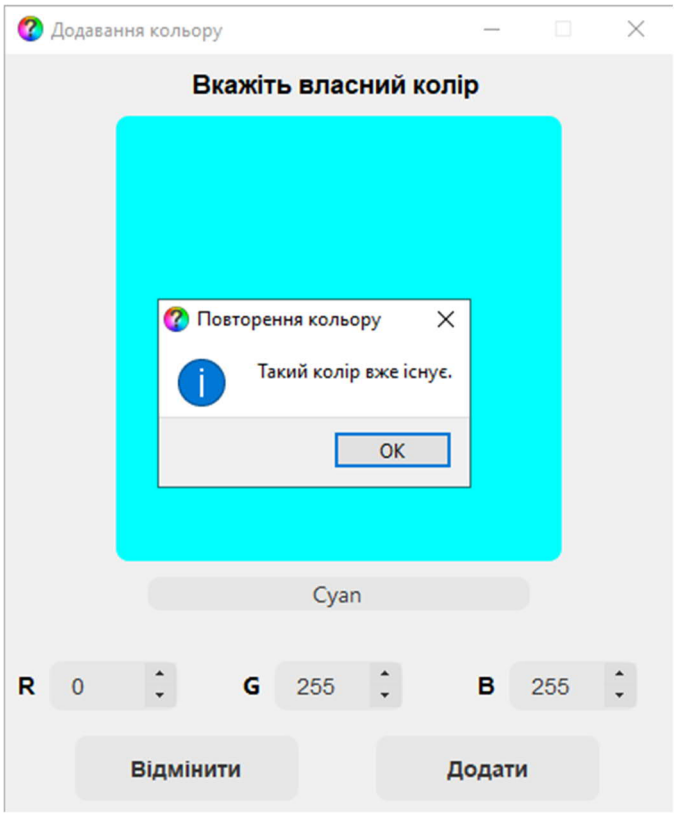


Рисунок 2.9 – Вікно додавання кольору при введені некоректної назви кольору

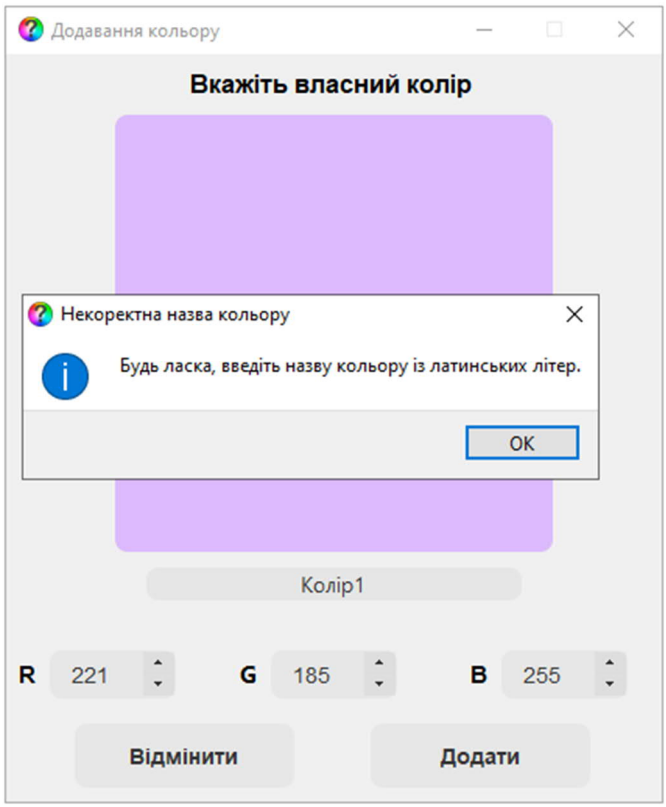


Рисунок 2.10 – Вікно додавання кольору при введені вже існуючого кольору

Як можна побачити із рис. 2.8, 2.9 та 2.10 програма обробляє відповідним чином всі випадки коли користувач вказує некоректні дані.

2.11 Керівництво користувача

Програма надає можливість у ігровій формі вивчити кольори у колірному просторі RGB, їх назву і відповідні їм значення, що сприяє підвищенню компетенції як веб-розробника, дизайнера та розробника програмного забезпечення.

Для початку гри треба вказати ім’я користувача яке містить латинські або кириличні літери і не більше довжиною за 20 символів, потім натиснути на кнопку у головному вікні програми з написом «Почати гру!», після чого відкриється вікно гри де треба буде вказати колір зображеного квадрату у три відповідних поля значення в RGB, від 0 до 255 для кожного.

Щоб закінчити гру та побачити власні результати цієї гри необхідно натиснути на кнопку з написом «Перевір мене!». Після натискання будуть виведені результати поточної гри у вигляді двох квадратів колір кожного з яких буде відповідати еталонному, який було необхідно вгадати, та вказаний колір гравцем, під цими квадратами відобразиться відповідні ним значення у RGB та їхня назва у круглих дужках.

Для звершення гри у вікні поточних результатів потрібно натиснути на кнопку з написом «Завершити», після чого користувача буде повернуто на головне вікно програми.

Для додавання власного кольору до списку існуючих необхідно обрати у меню «☰» пункт «Додати власний колір», після чого буде відкрито вікно подібне до вікна гри, вводячи значення у поля RGB колір квадрату буде змінюватись у відповідності введеним значенням, після їх введення необхідно вказати назву кольору латинськими літерами не більше довжиною за 15 символів, потім натиснути на кнопку із написом «Додати», після натискання необхідно підтвердити додаванням натиснувши «Так» у модальному вікні, після чого буде додано колір до списку кольорів, а користувача буде повернуто на головне вікно програми.

Щоб переглянути попередні результати у головному вікну у меню «☰» обираємо пункт «Попередні результати», тоді відкриється вікно де у текстовому полі буде зображено результати попередніх ігор, щоб закрити вікно необхідно натиснути на кнопку з написом «ОК» тоді користувача буде повернуто на головне вікно програми.

ВИСНОВКИ

У ході виконання курсової роботи було закріплено знання із загальних та професійно спрямованих компетенцій, також на набуто навички та вміння робити із літературними джерелами, результатами власних досліджень, сформовано навички представлення отриманих результатів з даної теми дослідження та закріплено вміння реалізації програмних продуктів згідно принципів об’єктно-орієнтованого програмування на основі C++ та інтегрованого середовища розробки Qt та її бібліотек.

На основі отриманих знань можна зробити висновок, що ООП є дуже важливою складовою сучасної інженерії програмного забезпечення яка надає розробниками реалізовувати добре структуровані програмні продукти та полегшує їх подальшу підтримку, але слід пам’ятати що дану парадигму необхідно використовувати у випадках де для вирішення поставленої задачі буде доцільним.

При виконанні даної курсової роботи були пройдені основні етами розробки програми: була поставлена задача, розроблено алгоритм, складено програму, встановлено її на ПК, налагоджено та протестовано її функціональність, перевірено коректність роботи у різних випадках.

Описано в повному обсязі робота програми, її алгоритм та надано керівництво користувача. Аналіз проблемної області виявив подібну з даної тематики. Розроблено інтуїтивно зрозумілий та простий користувацький інтерфейс для широкого загалу.

У пояснювальній записці задокументовано всі етапи виконання поставленої задачі та описана робота програмного застосунку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1 Eckel, Bruce. Thinking in C++, Volume 1: Introduction to Standard C++. Prentice Hall, 2003

2 Lipp'man, Stanley, Lajoie, Josée, and Moo, Barbara. C++ Primer (5th Edition). AddisonWesley, 2012.

3 Dewhurst, Stephen, and Stark, James L. C++ Gotchas: Avoiding Common Problems in Coding and Design. Addison-Wesley, 2007.

4 Stro'ustrup, Bjarne. The C++ Programming Language (4th Edition). Addison-Wesley, 2013.

5 Color Space and Its Divisions: Color Order from Antiquity to the Present Hardcover, 2003

6 Методичні вказівки до виконання лабораторних робіт з дисципліни «Об'єктно-орієнтоване програмування» для студентів спеціальності 121 «Інженерія програмного забезпечення» / К.В. Яшина, К.М. Ялова, Н.М. Лимар // Кам’янське: ДДТУ, 2019

Додаток А

Лістинг класів

MainWindow

```
#include <QMainWindow>
#include <QMessageBox>
#include <QDateTime>
#include <QRegularExpression>
#include <QFile>
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "gamewindow.h"
#include "addcolorwindow.h"
#include "usersscorewindow.h"
#include "playermanager.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    connect(ui->mainPushButton, &QPushButton::clicked, this, &MainWindow::startGame);
    connect(ui->action, &QAction::triggered, this, &MainWindow::showAddColorWindow);
    connect(ui->action_2, &QAction::triggered, this, &MainWindow::showUsersScoreWindow);

    QPixmap pixmap("../Guess_color_game/images/Guess the color.png");

    ui->label->setPixmap(pixmap);
    ui->mainPushButton->setCursor(Qt::PointingHandCursor);
    ui->mainLineEdit->setCursor(Qt::IBeamCursor);
    ui->menu->setCursor(Qt::PointingHandCursor);
    ui->mainMenubar->setCursor(Qt::PointingHandCursor);
}
…
```

GameWindow

```
#include "gamewindow.h"
#include "ui_gamewindow.h"
#include "currentresultwindow.h"
#include "playermanager.h"
#include <QMessageBox>

void GameWindow::setPlayerName(const QString &name) {
    m_playerName = name;
}

GameWindow::GameWindow(const QString &playerName, QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::GameWindow)
    , m_playerName(playerName)
{
    ui->setupUi(this);
    if (!m_colorData.loadColorsFromJson("../Guess_color_game/colors/colors.json")) {
        QMessageBox::critical(this, "Помилка", "Не вдалося завантажити список кольорів.");
        return;
    }

    setPlayerName(m_playerName);
    connect(ui->pushButton, &QPushButton::clicked, this, &GameWindow::checkGuess);
```

```
    disconnect(ui->pushButton_2, &QPushButton::clicked, nullptr, nullptr);
    connect(ui->pushButton_2, &QPushButton::clicked, this, &GameWindow::on_pushButton_2_clicked);
    ui->pushButton->setCursor(Qt::PointingHandCursor);
    ui->pushButton_2->setCursor(Qt::PointingHandCursor);
    updateSquareColor();
}
…
```

Додаток Б

Лістинг інтересної частини головного вікна

```
void MainWindow::startGame()
{
    QString playerName = ui->mainLineEdit->text();

    QRegularExpression regex("[a-zA-Za-яA-Я]+");


    if (playerName.isEmpty() || !playerName.contains(regex)) {
        QMessageBox::information(this, "Помилка", "Будь ласка, вкажіть ім'я гравця з латинських або
            кириличних символів.");
        return;
    }


    if (playerName.length() > 20) {
        QMessageBox::information(this, "Помилка", "Ім'я гравця не може перевищувати 20 символів.");
        return;
    }

    playerName = QDateTime::currentDateTime().toString("dd.MM.yy hh:mm:ss") + ": " + playerName;

    m_playerManager.addPlayer(playerName);

    this->setEnabled(false);
    GameWindow *gameWindow = new GameWindow(playerName, this);

    connect(gameWindow, &GameWindow::closed, this, &MainWindow::enableUIElements);

    gameWindow->show();
}
```

Метод обчислення оцінки за гру

```
void GameWindow::checkGuess() {
        int userRed = ui->spinBox->value();
        int userGreen = ui->spinBox_2->value();
        int userBlue = ui->spinBox_3->value();

        QColor selectedColor = ui->frame->palette().color(QPalette::Window);

        double distance = colorDistance(selectedColor.red(), selectedColor.green(), selectedColor.blue(), userRed,
        userGreen, userBlue);

        QString resultMessage;
        if (distance <= 1) {
            resultMessage = "Відмінно!";
        } else if (distance <= 20 && distance > 1) {
            resultMessage = "Дуже близько!";
        } else if (distance <= 60 && distance > 20) {
            resultMessage = "Близько!";
        } else if (distance <= 100 && distance > 60) {
            resultMessage = "50/50";
        } else {
            resultMessage = "Спробуй ще раз.";
        }

        QColor referenceColor = ui->frame->palette().color(QPalette::Window);
```

```
    QColor userColor(userRed, userGreen, userBlue);

    QString referenceColorName = m_colorData.getColorName(selectedColor);
    QString userColorName = m_colorData.getColorName(userColor);

    CurrentResultWindow *resultWindow = new CurrentResultWindow;
    resultWindow->setReferenceColor(selectedColor);
    resultWindow->setUserColor(userColor);
    resultWindow->setResultMessage(resultMessage);
    resultWindow->show();

    m_playerManager.updatePlayerScore(m_playerName, referenceColor, userColor, referenceColorName,
    userColorName, resultMessage);
    this->close();
}
```