МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»
КРИВОРІЗЬКИЙ ЕКОНОМІЧНИЙ ІНСТИТУТ

Факультет: Обліково-економічний
Кафедра: Інформатики та прикладного програмного забезпечення

К В А Л І Ф І К А Ц І Й Н А Р О Б О Т А

за напрямом підготовки «**Програмна інженерія**»

Короновський Михайло Віталійович
(прізвище, ім'я, по батькові)

«Розробка програмного забезпечення контролю знань студентів»
(тема кваліфікаційної роботи)

Науковий керівник

д. т. н., професор
(посада, учений ступінь, учене звання)

Зеленський О. С.
(прізвище, ім'я, по батькові)

Кваліфікаційна робота допущена
до захисту рішенням кафедри
інформатики та прикладного
програмного забезпечення
Протокол №___від «___» _____20__р.

Керівник програми

_____ Зеленський О.С., д. т. н., професор
(підпис) (прізвище, ініціали, учений ступінь, учене звання)

Кривий Ріг 2016р.

ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»
КРИВОРІЗЬКИЙ ЕКОНОМІЧНИЙ ІНСТИТУТ

Факультет: Обліково-економічний

Кафедра: Інформатики та прикладного програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри інформатики та ППЗ

_____ Зеленський О.С.
(підпис) (прізвище, ім'я, по-батькові)

"_____"_____20__р.

З А В Д А Н Н Я

на виконання кваліфікаційної роботи
за напрямом підготовки «**Програмна інженерія**»

студенту Короновському Михайлу Віталійовичу
(прізвище, ініціали)

Тема: «Розробка програмного забезпечення контролю знань студентів»

Затверджена наказом по інституту від "_____"_____20____р. №_____

Кваліфікаційна робота виконується на матеріалах навчального процесу.

Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити): виконання постановки задачі, розробка алгоритма розв'язання задачі, розробка програмного забезпечення.

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Відмітка керівника про фактичне виконання

Термін здачі роботи _____

Дата видачі завдання _____

Завдання підготував науковий керівник _____

(підпис) (посада, прізвище)

Завдання прийняв до виконання _____

(підпис) (посада, прізвище)

Р Е Ф Е Р А Т

Короновський Михайло Віталійович

Розробка програмного забезпечення для контролю знань студентів

Кваліфікаційна робота на здобуття освітньо – кваліфікаційного рівня бакалавра за напрямом підготовки «Програмна інженерія». Криворізький економічний інститут ДВНЗ «Криворізький національний університет». Кривий Ріг, 2016.

Обсяг роботи: 63 стор., 4 табл., 16 рис., 21 джерело, 3 додатки.

Предмет дослідження: обробка сканованих бланків з відповідями.

Об'єкт дослідження: алгоритми комп'ютерного розпізнавання образів.

Мета роботи: розробити програмного забезпечення для атоматичного розпознавання бланків на мові C#.

Методи дослідження: комп'ютерний зір, розпізнавання образів, графічні перетворення, адаптивне порогування.

Програмне забезпечення: мова програмування C# та бібліотека Microsoft .NET Framework.

Результати та їх новизна: розроблене програмне забезпечення для автоматичного розпізнавання бланків планується застосовуавти в якості модулю для іншого комплексу контролю знань, який розробляється паралельно.

Рекомендації щодо використання результатів роботи: рекомендується до використовування в навчальному процесі для оперативного контролю знань студентів.

Галузь застосування: навчальний процес.

Значущість роботи та висновки: розроблене програмне може застосовуватись вчителем (викладачем) в навчальному процесі.

Ключові слова: КОМП'ЮТЕРНИЙ ЗІР, РОЗПІЗНАВАННЯ ОБРАЗІВ, ОБРОБКА ГРАФІЧНИХ ДАНИХ, АДАПТИВНЕ ПОРОГУВАННЯ.

ЗМІСТ

ВСТУП

Метою цієї роботи є розробка програмного забезпечення для розпізнавання бланків, яке в майнутньому буде включено до складу іншого програмного комплексу для контролю знань студентів, яке розробляється паралельно з цим проектом студентом четвертого курсу вищого навчального закладу КНУ КЕІ Єрьоміним Іваном Володимировичем для його застосування в якості основного інструменту для оцінки знань під час навчального процесу.

Контроль якості засвоєння навчального матеріалу є невід'ємною частиною процесу навчання, і в той же час це найбільш складний вид взаємодії викладача і студента. В наш час, найширше розповсюдження знаходять методи контролю знань шляхом тестування.

Тестування є однією з форм контролю знань, вмінь та навичок студентів в процесі вивчення ними окремої теми або навчальної дисципліни. Зокрема, впровадження модульно-рейтингової системи в навчальний процес вищих закладів освіти вимагає застосування тестового контролю для оцінки знань учнів, що забезпечує високу технологічність проведення контролю та об’єктивність його результатів.

Використання нових інформаційних технологій при вивченні різних дисциплін, надає значні можливості для розширення і поглиблення теоретичної бази знань, практичної значущості і застосовності результатам навчання. З іншого боку, впровадження комп’ютера у навчальний процес вимагає наявності відповідного практичного забезпечення, розробки нових методичних систем навчання, комп’ютерного контролю знань, аналізу і коригування результатів діяльності.

Для підвищення ефективності організації тестового контролю його доцільно проводити з використанням комп’ютерних тестових програм, що дозволяє автоматизувати процес проведення контролю та обробку результатів тестування. Комп’ютер – на цей час один із оптимальніших

засобів для проведення тестування як форми контролю та діагностики знань студентів. Використовуючи при цьому декілька різновидів тестів, можлива повноцінна, незалежна та адекватна оцінка не лише знань студентів, але й їх вміння.

Комп'ютерне тестування успішності дає можливість реалізувати основні дидактичні принципи контролю навчання:

- принцип індивідуального характеру перевірки й оцінки знань;
- принцип системності перевірки й оцінки знань;
- принцип тематичності;
- принцип диференційованої оцінки успішності навчання;
- принцип однаковості вимог викладачів до студентів;
- принцип об'єктивності.

На сьогодняшній день можлива комп’ютерна обробка сканованних паперових бланків з відповідями студентів. Ця техніка надає декілька переваг в порівнянні з класичним автоматизованним тестуванням, де студент під час контролю є оператором ПК. Контроль можна проводити в класичній формі, надав студенту питання у друкованному виді та бланк для відповідей, який згодом треба сканувати та передати зображення в спеціалізоване програмне забезпечення, яке виконає перетворення даних та перевірку відповідей. Для розробки даної підсистемі треба реалізувати певні алгоритми для роботи з зображеннями.

Об’єктом дослідження є обробка сканованих бланків з відповідями.

Предметом дослідження є алгоритми комп’ютерного розпізнавання образів.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

1.1 Характеристика задачі

Для того, щоб правильно сформулювати характеристику задачі, необхідно зібрати матеріал з предметної області та побудувати програмне забезпечення на фого основі, яке б відповідало меті кваліфікаційної роботи. Характеристика задачі полягає у описанні призначення задачі, підстави для її рішення, предмету задачі, а також визначення перших вимог до розроблюваної програми, що стосуються її інтерфейсу, вхідної та вихідної інформації, звітування тощо.

Вивчивши сутність та деякі аспекти досліджуваної теми можна приступити до постановки мети та визначення етапів роботи.

При розробці даного програмного забезпечення, було поставлено наступну задачу – спроектувати програмне забезпечення розпізнання відсканованих бланків з відповідями.

Розроблювана програма на мові C# буде вирішувати наступні функціональні задачі:

- прграмне забезпечення має буті спроєктоване таким чином, щоб у майбутньому стати частиную більш великого програмного комплексу контролю знань;
- завантаження зображення;
- підготовка зображення для подальшого аналізу, графічна обробка алгоритмами для зміни контрасту;
- аналіз;
- детальний вивід результатів роботи алгоритмів розпізнавання;
- надання користувачеві допоміжних інструментів для внесення правок та аналізу роботи програми;

- побудова гістограми інтенсивностей клітинок в двох режимах: простий та детальний.
- завантаження зображення;

Дане програмне забезпечення було розроблене за допомогою мови програмування C# та бібліотеки Microsoft .NET Framework, тому що материнська програма, якак розроблюється паралельни з цим проектом використовує такий самий стек технологій. Програма є додатком на основі бібліотеки WinForms, яка входить до складу стеку .NET Framework, тому можлива подальша інтеграція цього програмного забезпечення з сервісами, які базуються на бібліотеках ADO .NET та використовують клієнт-серверну архітектуру.

Програмне забезпечення має справлятися з задачами обробленя сканованих бланків з відповідями та вилучення з них інформації про клітинки для відповідей, а саме – які клітинки є помічиними як відповідь, а які залишились пустими.

Алгоритми розпізнавання та методи розв’язання даної задачи мають бути незалежними від почерку відповідаючего та не вносити ніяких припущень в навчальний процес і бути толерантним до індивідуальних здібностей кожного сканованого зображення, адаптуватися до змінних і предоставити користувачу інформацію про протекання процесів аналізу, дати змогу вплинути на вихідні зані, якщо похибка породить невірний результат. Треба мінімізувати все похибки розпізнавання.

Дана реалізація прогрмного модулю є автономною, всі результати роботи виводяться на екран.

Таблиця 1.1

Системні вимоги програми

	Мінімальні системні вимоги	Рекомендовані системні вимоги
Процесор	Intel, 1 ГГц ті вище	Intel Core i3 та потужніше

Продовження таблиці 1.1

	Мінімальні системні вимоги	Рекомендовані системні вимоги
Відеокарта	256 Мб графічної, DirectX 9 сумісна	512 Мб відеопам'яті, DirectX 11 сумісна
Оперативна пам'ять	512 МБ	1 ГБ
Операційна система	Microsoft Windows XP	Microsoft Windows 7
Підтримуване розширення	1024x768	1280x1024 та вище

Дані системні вимоги обумовлені, перш за все, офіційними системними вимогами для стеку Microsoft .NET Framework 4, який, на сьогоднішній день, є актуальною версією ті має офіційну підтримку з боку Microsoft.

Користувач повинен мати змогу візуально оцінити результат работи аналізу вхідного зображення, для чого йому необхідно надати допоміжні інструменти та елменти управління. Для аналізу роботи алгоритмів розпізнавання зображення в програмі є два інструменти – цифрове представлення бланку з усіма клітинками та гітограма інтенсивностей клітинок інтенсивностей, а також деякі допоміжні.

1.2 Вхідна та вихідна інформація

Вхідна інформація – дані, що надходять на вхід задачі та використовуються для її вирішення. В даному випадку, вхідними даними для програмного запезпечення є графічний файл у форматі BMP, JPEG або PNG. Більшість ПЗ сканерів зберігають зображення в дані формати.

Дане програмне забезпечення потребує застосовування сканів спеціально зформованих бланків. Ці бланки містять у собі спеціяльни допоміжні маркери.

Графічний файл, який завантажується до програмного забезпечення, має стуктуру матриці певних розмірів. Оброблюванні зображення є двовимірними, їх матриця має елементи, структура даних яких зображена на таблиці 1.2.

Таблиця 1.2

Структура даних елементу графічної матриці

№ п/п	Найменування	Поле	Тип поля
1	Червоний канал	R	Числовий
2	Зелений Канал	G	Числовий
3	Блакитний канал	B	Числовий
4	Альфа-канал	A	Числовий

Усі поля у структурі типу RGBA, яка зображена на таблиці 1.2 мають розмір одного байту та граничні значення від 0 до 255 включно. Така структура формує 32 – бітний колір, а матриця з таких елементів – двовимірне зображення.

Вхідні дані – зображення, отриманні за допомогою сканера, який обробляє бланки за шаблоном, який показан на рисунку 1.1.

Даний шаблон бланку витримує мінімальни вимоги навчального процесу в даному навчальному закладі, має достатню кількисть питань та номер, за допомогою якого можливо ідентифікувати студента, який цій бланк заповнював.

Візуально, від класичного бланку, запропонований формат пропонує добавити два нових єлементів – це спеціальни маркери для полегшення подальшої розробки алгоритма розпізнавання. Останні елементи не були змінені та залишилися недоторканими.

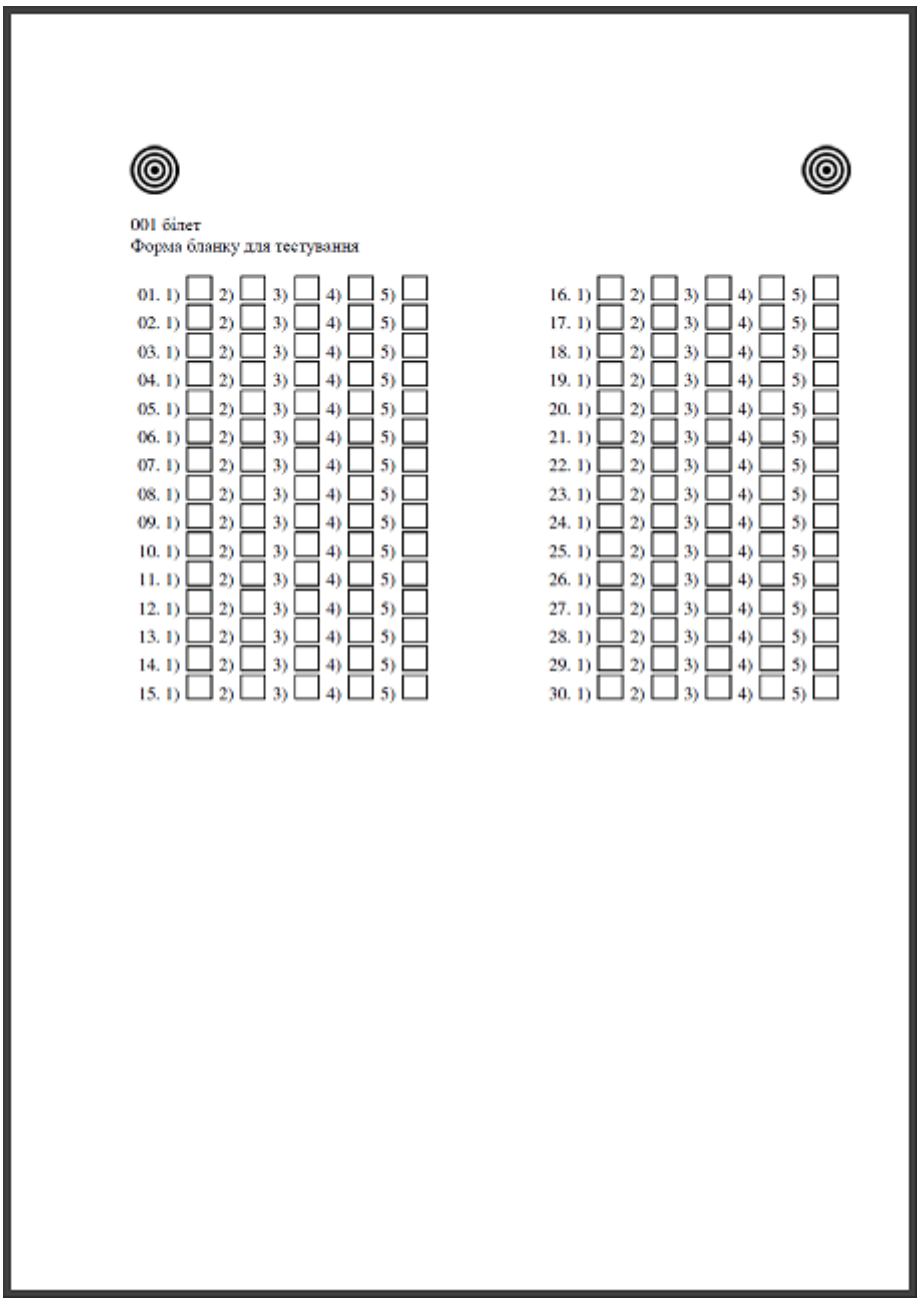

Рис. 1.1. Пустий шаблон вхідного скана для першого білету

Для виконання алгоритмів розпізнавання необхідно перевести вхідні дані – зображення в бінарний вид та позбавитися від діталізації м’яких віттінків і градієнтів, таким чином одие елемент вхідної матриці повинен бути характеризован як однобітне число. Існує багато методів для трансформації кольорових зображень в чорно-білі і для даної задачі розпізнавання бланків достатньо одного з найпростішого. Для удобства реалізації, значення, які містять інформацію про окремий елемент матриці,

будуть зберігатися в однобайтній структурі, тому що сучасні центральні процесори не мають змогу прямого доступу до окремого біту в оперативній пам’яті, найменшою одиницею даних є один байт. Для переводу чотирьохбайтного кольору в бінарне уявлення був використан алгоритм адаптивного порогування, який обчислює графічну інформацію тільки з червоногу каналу, інші канали ігноруються. Таким чином, протекає процес відготовки вхідних даних після їх завантаження для подальшої їх обробки основними алгоритмами даного програмного забезпечення.

Вхідні зображення мають певні вимоги, як бланки. В данії реалізації програмного забезпечення, підтримується один єдиний шаблон. Окрім того, що бланк має два маркери, він налічує 150 клітинок для відподіей на питання, по 5 клітинок на відповідь, усього 30 питань. Перші 15 питань розтошовані у лівій колонці, а останнь 15, з 16 по 30 – в правій колонці. Кожний графічний елемент розташован статично, усі параметри їх характеристик, таких як відстані та розміри, враховуються в програмному запезпеченні під час розпізнавання, тому програма комплектується готовими спеціально згенерованний друкований файлами в форматі Adobe PDF, які друкуються на усіх правильно налаштованих системах для друку однаковим чином, з одними пропорціями. Друкований файл має 100 сторінок для 100 білетів, по одній сторінці на білет. Перед початком тестування, користувач повинен надрукувати необхідні білети, наприклад від 1 до 30. Ця здібність не має нічого спільного з принципами алгоритмів розпізнання, які були реалізовані в данії програми та є универсальними, тому в майбутньому можливо реалізувати генератор бланків з індивідуальними опціями, не змінюючи базові принципи даного підходу.

Для збереження вхідних даних у пам’яті, а також для їх обробки, треба використовувати класи, яки є складовою Microsoft .NET Framework, зокрема System.Drawing.Bitmap. Слід помітити, що даний клас передбачає нетрадиційний порядок каналів на один елемент матриці. Його порядок зворотний (BGR), якщо розсматрювати бітовий поток.

Для успішного завантаження вхідних даних в середовищі користувача требо поставитиє певні вимоги:

- в момент завантаження вхідного файлу, він не має бути заблокованим, або відкритим для користування будь-яким іншим програмним забезпеченням;
- в момент завантаження вхідного файлу, ніяке інше программе забезпечення не повинне налагоджувати до нього будь-який доступ, окрім зчитування.

Якщо буде порушена одна із вимог, програмне забезпечення повинне проінформувати користувача про помилку доступу до вхідного файлу, не завершить свою роботу та повернеться в стан очікування дії користувача. Посилання про подібну помилку зображена на рисунку1.2.

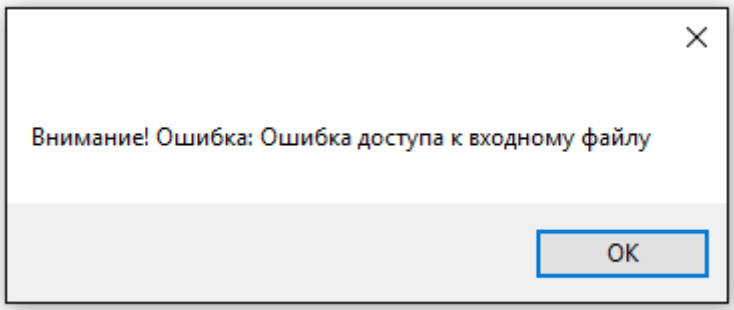

Рис. 1.2. Посилання про помилку доступу до вхідного файла

Після того, як процес завантаження було виконано, а графічні дані були зображені на головному модулі програми, доступ до вхідних даних більш не налагоджуваються та вони повинні бути свободними для других програм в операційній системі.

Користувач завантажує вхідне зображення у програму для подальшої обробки. Вибір файлу зображено на рисунку 1.3. Діалогове вікно вибору вхідного файлу має бути реалізовано за допомогою класа OpenFileDialog в .NET Framework. Такі діалоги мають вигляд, який є стандартним для

операційної системи, де може виконатися віртуальна машина .NET CLR. На даний момент, це сечасні операційні системи сімейства Microsoft Windows.

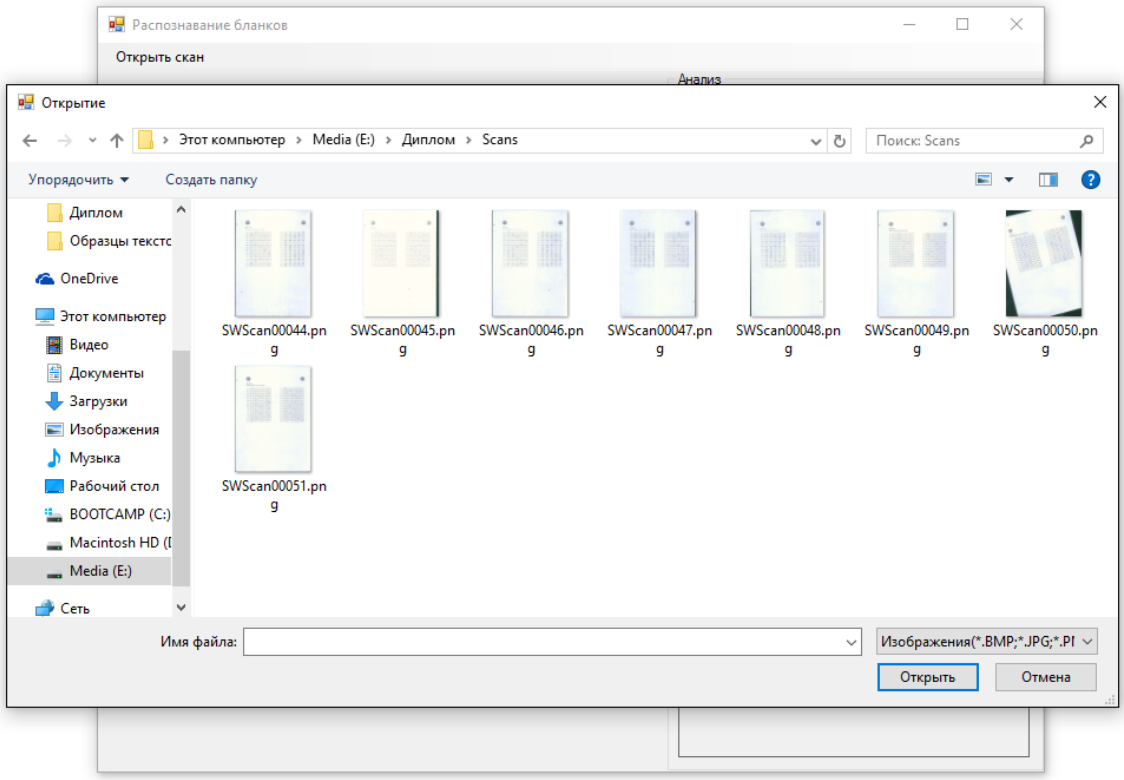

Рис. 1.3. Діалогове вікно завантаження вхідного зображення.

Операції, які виконуються над вхідними даними в цьому програмному забезпеченні треба виконувати паралельно, для підвищення швидкісті роботи програми. Це було досягнено за допомогою створення робочих потоків, які виконують операції над непересічними областями даних. Такий паралеллізм забезпечує значне прискорення виконання основної роботи програми, особливо, якщо центральний процесор персонального комп'ютеру користувача має два, або більше обчислювальних ядер. Один сеанс перевірки бланків може налічувати декілька десятків зображень високої плотності, такі вхідні дані майже завжди, за декількома виключеннями, можна обробляти паралельно в тій чи іншій мірі паралелелізації. Для реалізації паралелізму використати методи, яки описані в простірі System.Parallel.

Вихідна інформація представляє собою дані, які відображують результат аналізу графічних даних, отриманих зі сканеру – відмітки для кожної клітинки. В даній реалізації програмного забезпечення, вихідна інформація зберігається в елементах користувача на головної формі. В майбутньому можливо ззбереження виїідних даних в централізовану базу даних, яка містить усю інформацію, яка стосується контролю знань. Таким чином, можно досягти великої деталізації кожного акту тестування та зберігати інформацію о протеканні цього процесу протягом тривалого терміну.

Користувач повинен мати змогу змінювати стан клітинок, якщо похибка аналізу скану даст невірний результат. Для цього, програмне забезпечення містить спеціальні елементи користувача, які реалізують дані вимоги, яки стосуються контролю та перевірки безпосередньо вихідних даних. Окрім цього, треба надати користувачеві наочну інформацію про аналіз у вигляді гістограми інтенсивностей, що є графічним образом вихідних даних у вигляді двовимірного графіку. Даний елемент застосовує окреме вікно для підвищеної деталізації вихідної інформації.

Програма розпізнавання бланків розроблена як составна частина великого комплексу для контроля знань студентів, тому вихідні дані треба підготувати з оглядкою на подальше застосовування їх у материнській програмі, чому сприяє використовування спільного технологічного стеку, а особливо – мови програмування.

В таблиці 1.3 представлена структура даних для зберігання інформації про одну клітинку на аркуші.

Таблиця 1.3

Структура даних Cell

№ п/п	Найменування	Поле	Тип поля
1	Індекс клітинки	Index	Числовий
2	Інтенсивність	Value	Числовий
3	Правка користувача	Force	Числовий

Дані, породжені від цієї структури, є вихідними та формуються у процесі роботи алгоритмів розпізнавання. Усі гістограми та інші елементи управління користувача роботають з даними, породженими від цієї структури.

Індекс клітинки, її номер на аркуші – це одне число, від 0 до повної кількості клітинок для відповідей не включно. Нумерація клітинок ведеться з лівої верхньої зліва направо. Спочатку, на аркуші розташовано клітинки для відповіді на перве питання, вони мають індекси від 0 і до вількості відповідей на це питання (в даному випадку 0, 1, 2, 3 та 4). Далі, нумерація продовжується зі збереженням поточного номеру вже для наступного питання (5, 6, 7, 8, 9 і 10 для другого питання) і т. д.

Інтенсивність клітинки – це сума «чорних» (для удобства, вважається, що в бінарній системі 0 – це білий, 1 – це чорний пиксель) пікселів в площі, яка обмежується нутрощами даної клітинка. Це поле грає рішучу роль в формуванні вихідних даних і є осносним джерелом інформації, стосовно результату аналізів даної клітинки.

Поле, яке зберігає рішення про виправлення користувачем (правка) результату обробки клітинки може мати три стани:

- значення 0, якщо виправлення відсутнє, або якщо користувач спочатку зробив виправлення, а потім скасував його;

- значення -1, якщо користувач встановив, що дана клітинка – порожня;
- значення 1, якщо користувач встановив, що дана клітинка була помічена.

Це поле надає основну інформацію, яка стосується виправлень користувачем результатів роботи програми, якщо вони були необхідні. Воно використовується для елементу користувача, яке передбачає подібні виправлення.

В майбутньому, додаткова інформація може бути передана в материнський комалекся контролю знань, яка буде зберігатися в БД. Наприклад, це номер білету, який теж є вхідними даними. Для того, зоб здобути ци дані без технічних похибок, можна оголосити вимогу, яка стосується назви файлу зображення бланка. Користувач, який сканує бланки, має задавати кожному файлу ім’я, яке буде містити номер сканованого білету, який цій файл зберігає. Ці дані необхідні для комплексу контролю знань для того, щоби співвіднести всі відповіді на запитання з конкретною персоную, яка складала цей тест. Кожному студенту присвоюється номер, який відвовідає номеру білету, який поступає до даної програми для розпознавання бланків, як вхідний файл.

Програмне забезпечення має бути розроблено в середовищі Microsoft Visual Studio з використанням технології Microsoft .NET Framework на мові програмування C#. Програма буде працювати на всіх сучасних версіях операційної системи Microsoft Windows, також користувачу знадобиться встановлений пакет Microsoft .NET Framework.

РОЗДІЛ 2
РОЗРОБКА АЛГОРИТМУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1 Алгоритм реалізації задачі

Алгоритм аналізу даних очікує матрицю, в клітинках якої, будить міститися графічний стан пікселя. Процес порогування класифікує кожен піксель зображення в одну з двох категорій: «чорний» та «білий». Усі процеси розпізнання в даному програмному забезпеченні використовують бінарну чорно-білу модель без віттінків сірого, тому що така модель даних для аналізу є достатньою для роботи, але типічний графічний файл, який завантажує користувач - це кольорове зображення з 255^3 (по байту на канал) варіаціями. Для того, щоб підготувати такі даня для подальшої обробки, треба применити фільтр порогування та позбавитися від двох каналів і використовувати, наприклад, тільки червоний.

Порогування - це найпростіший метод сегментації зображень. Цей метод може бути використуваний для створення бінарних зображень з чорно-білих. Найпростіший метод заміни в порогової обробки зображення: встановити чорний піксель, якщо його інтенсивність менше деякого фіксованого значення або білий пікселя, якщо інтенсивність зображення більше, ніж ця константа. У наведеному прикладі зображення (рисунок 2.1), це призводить до того, що темне дерево стає повністю чорним, а білий сніг стає повним білим.

Рис. 2.1. Приклад фіксованого порогування

Зліва, на рисунку 2.1., зображено початкове зображення, а справа – обработане за допомогою алгоритма фіксованого порогування.

Фіксоване порогування має недолік – різні частини одного зображення майже николи не освітлюються однаково, це породжує ложні чорні пікселі та вносить велику похибку до подальших розрахунків. Це створює проблему вибору порогу. В деяких випадках, будь-яке значення порогу може завжди дати незадовільні результати, тому цій алгоритм в чистому виду не рекомендується використовувати для задач цього проекту.

Щоб повністю автоматизувати процес порогування та зміншити похибку, необхідний алгоритм для автоматичного вибору порогового значення. Мехмед Сезгін та Бюлент Санкур у своєму дослідженні в 2004 році класифікували ці методи порогових значень в наступні шість груп на основі інформації якою маніпулює алгоритм:

- аналіз гістограми;
- кластеризація на основі методів, де зразки рівнів сірого зосереджені в двох частинах в якості фону і переднього плану (об'єкта), або попеpемінно моделюються у вигляді суміші двох гауссіан;
- алгоритми, зосновані на ентропії, які використовують ентропію областей переднього плану і фону, крос-ентропією між вихідним і бінарного зображення і т.д.;
- алгоритми, зосновані на атрибутах об’єктів на основі методів пошуку міру подібності між сірим рівнем і бінарного зображень, таких як нечітка форма подібності, краю збігу і т.д.;
- просторові методи які використовують розподіл високого порядку ймовірностей і / або кореляції між пікселями;
- локальні методи, що адаптатують порогове значення на кожен піксель до місцевих характеристик зображення. У цих методах, інший поріг вибирається для кожного пікселя в зображенні.

В даній реалізації був використан алгоритм, який базується на локальніх адаптивних методах порогування. Ці методи враховують прострові варіації освітленності зображення.

Для того, щоб враховувати зміни освітленості, загальним рішенняя є адаптивне пороговування. Основна здібність цього алгоритму полягає в обчисленні іншого порогового значення для кожного пікселя в зображенні. Цей метод забезпечує більшу стійкість на зміну освітленості.

Інтегральна матриця є інструментом, який може бути використаний щоразу, коли є функція від пікселів до дійсних чисел P (x, y) (наприклад, інтенсивності пікселів), і треба обчислити суму цієї функції над прямокутною областю зображення. Без інтегральної матриці, сума може бути обчислена в лінійному часу на прямокутнику шляхом обчислення значення функції, окремо для кожного пікселю. Проте, якщо нам потрібно обчислити суму за кількома пересічними прямокутними вікнами, ми можемо використовувати інтегральну матрицю і досягти постійної кількості операцій в прямокутнику для підготовки даних (побудова інтегральної матриці).

Для обчислення інтегральнї матриці зображення, ми зберігаємо в кожному її вузлі I (x, y) суму всіх P (x, y) членів зліва і вище пікселя (x, y). Це досягається за лінійний час, використовуючи наступне рівняння (рисунок 2.2) для кожного пікселя.

$$I(x,y) = f(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$

Рис 2.2. Рівняння для поелементного формування інтегральної матриці

Рисунок 2.4 (зліва і в центрі) ілюструє обчислення інтегральної матриці. Після того, як здобуто інтегральну матрицю, сума функції для будь-якого прямокутника з лівим верхнім кутом (x1, y1) і правим нижнім кутом (x2, y2) може бути обчислена за постійний час, використовуючи наступне рівняння, яке зображене на рисунку 2.3.

$$\sum_{x=x_1}^{x_2}\sum_{y=y_1}^{y_2} f(x,y) = I(x_2,y_2) - I(x_2,y_1-1) - I(x_1-1,y_2) + I(x_1-1,y_1-1)$$

Рис 2.3. Рівняння пошуку суми в прямокутному вікні

На рисунку 2.4 (праворуч) показно, що для обчислення суми P (x, y) над прямокутником D з використанням рівняння на рисунку 2.3 еквівалентно обчисленню сум над прямокутниками (A + B + C + D) - (A + B) - (A + C) + A.

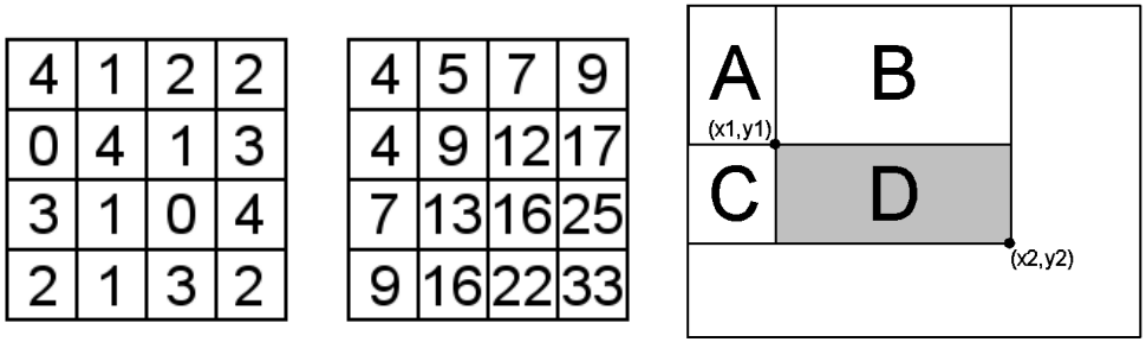

Рис. 2.4. Приклад використання інтегральної матриці

Зліва на рисунку 2.4 зображена вхідна матриця зображення, по-центру - інтегральна матриця, а справа - приклад використання інтегральної матриці, щоб обчислити суму в прямокутнику D.

Даний адаптивний метод порогування є простим розширенням методу Велнера, основною ідеєю якого є порівняння кожного пікселю із середнім показником оточуючих пікселів. Автори цієї ідеї – Дерек Бредлі з університету Карлтон (Канада) та Герхард Рос з Канадської Національної ради з науково-дослідної роботи.

Наближена змінна середня останніх s пікселів обчислюється під час проходження зображення. Якщо значення поточного пікселя на t відсотків нижче, ніж в середньому, тоді він встановлюється як чорний, в іншому випадку він встановлюється як білий колір. Цей метод працює таким чином, що порівнення пікселя із середнім значенням сусідніх пікселів збереже жорсткий контраст ліній і ігнорує м'які зміни градієнта. Перевага цього

методу полягає в тому, що потрібен тільки один прохід через зображення. Велнер використовує 1 / 8 ширини зображення для значення s і 15 для значення t. Проте, цей метод має проблему, яка полягає в залежності від порядку зчитування пікселів. Крім того, ковзне середнє не є хорошим поданням оточуючих пікселів на кожному кроці, оскільки зразки околиці не рівномірно розподілені по всіх напрямках. Використовуючи інтегральну матрицю (і приносячи в жертву ще одну ітерацію через зображення), можна розробити рішення, яке не страждає від цих проблем. Замість обчислення ковзного середнього останніх s пікселів, обчислюється середнє значення s * s вікна пікселів, зосереджених навколо кожного пікселя. Це краще середнє значення для порівняння, так як воно розглядає сусідні пікселі з усіх боків. Обчислення середнього виконується за лінейний час з використанням інтегральної матриці. Треба розрахувати інтегральну матрицю в першому проході через вхідне зображення. У другому проході треба обчислити середнє s * s вікна за допомогою інтегрального зображення для кожного пікселя за постійний час, а потім виконти порівняння. Якщо значення поточного пікселя на t відсотків менше, ніж цей середній, тоді він встановлюється як чорний, в іншому випадку він встановлюється як білий. На рисунку 2.5 зображено: зліва – початкове вхідне зображення, по-центру – результат обробки за допомогою фиксованого порогування, справа – результат обробки за допомогою алгоритму адаптивного порогування.

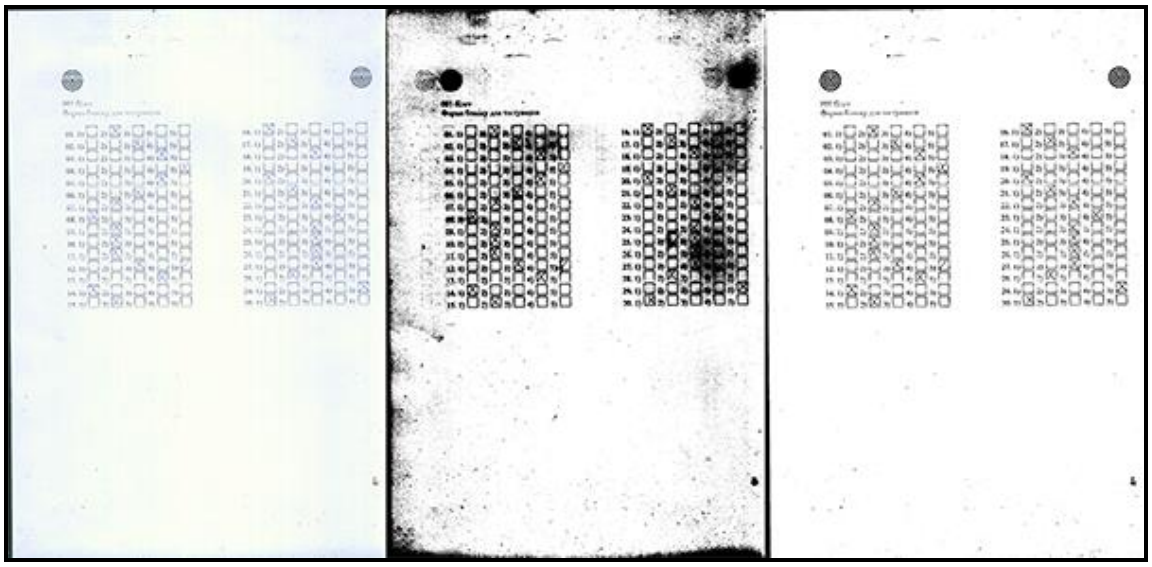

Рис. 2.5. Порівняння алгоритмів порогування

На рисунку 2.6 зображен пвседо-код даного алгоритму.

procedure $AdaptiveThreshold(in, out, w, h)$

1: **for** $i = 0$ to w **do**
2: $sum \leftarrow 0$
3: **for** $j = 0$ to h **do**
4: $sum \leftarrow sum + in[i, j]$
5: **if** $i = 0$ **then**
6: $intImg[i, j] \leftarrow sum$
7: **else**
8: $intImg[i, j] \leftarrow intImg[i-1, j] + sum$
9: **end if**
10: **end for**
11: **end for**
12: **for** $i = 0$ to w **do**
13: **for** $j = 0$ to h **do**
14: $x1 \leftarrow i - s/2$ {border checking is not shown}
15: $x2 \leftarrow i + s/2$
16: $y1 \leftarrow j - s/2$
17: $y2 \leftarrow j + s/2$
18: $count \leftarrow (x2 - x1) \times (y2 - y1)$
19: $sum \leftarrow intImg[x2, y2] - intImg[x2, y1-1] - intImg[x1-1, y2] + intImg[x1-1, y1-1]$
20: **if** $(in[i, j] \times count) \leq (sum \times (100 - t)/100)$ **then**
21: $out[i, j] \leftarrow 0$
22: **else**
23: $out[i, j] \leftarrow 255$
24: **end if**
25: **end for**
26: **end for**

Рис. 2.6. Псевдо-код алгоритму адаптивного порогування

Отже, обробка зображення за допомогою методів адаптивного порогування дозволяє отримати чітке зображення у бінарному вігляді, відділити корисну інформацію від фона та позбавитися від впливу освітлення.

Після підготовки графічних даних для подальшого їх аналізу, необхідно локалізувати області клітинок. Ця задача є комплексною, тому що всі вхідні зображення мають різні параметри та фактичне положення ціх областей. Існує три змінних, яки сприяють на локалізацію областей у двовимірнії площині: це точка відліку, кут поворота координатної системи та масштаб. Ці змінни треба обчислювати автоматично та індивідуально для кожного вхідного скана.

Для того, щоб визначити всі необхідні характеристики двовимірної системи координат для клітинок, необхідни фізични орієнтири на скані – маркери. При друці бланку на аркуш, маркери розташовуються в одній системі координат з клітинками. Знаючи приблизне місце їх знаходження, можна знайти його фактичні координати у пікселях на скані. Таких маркеров треба бути два, тому що система двовимірна. Двох маркерів достатньо, щоб знайти усі її характеристики. Відстань між ними завжди співвідноситься з усіма відстанями на аркуші, вона дає нам характеристику масштабу - співвідносення відстаней між двома пікселями та реальними фізичними даними аркуша. Зная ці дані, мі можемо обчислити реальну відстань між двома пікселями на скані у фізичних єдиницях. Кут між ціми двома маркерами – це кут, на який був повернут лист при скануванні, на нього треба повернути координатну вісь, початкова точка якої буде вважатися координатами одного з двох маркерів, щоб вирівняти скан.

Для того, щоб точність найденого кута повороту була найвища, треба розтошовувати маркери на максимальнії відстані. Для удобства реалізації алгоритму маркери були розтошовані паралельно осі абсцис. Якщо бланк був надрукован якісно, без деформацій, а розташування маркерів було виявленно коректно – всі області клітинок (та будь яка точка в їх системі координат)

може бути локалізована досконально, але на практиці завжди присутня похибка, тому резонно зміншити шукану площу клітинки, щоб вона вписувалася в фактичну клітинку та перекривала її границі, що є важливим для подальшого аналізу.

На рисунку 2.7 зображен маркер, який використан при побудові бланку.

Рис. 2.7 Маркер

Даний маркер має круглу форму, тому що при будь-якому повороту його зображення, такі форми ніяк не зміняться, тому його буде легше шукати, тому що зникає різниця між повернутим маркером та неповернутим.

Пошук маркерів можна здійснити за лінейний час за допомогою інтегральної матриці таким самим чином, як було здійснено обчислювання середнього прямокутного вікна в процесі підготовки даних зображення. Треба знайти дві найінтенсивніши квадратні області на зображенні, тому ці маркери мають бути достатньою плотними, відносно другого графічного наповнення бланку. Область пошуку – квадратна та буде мати такий розмір, щоб її можна було вписати в гранічну окружність маркеру, тому що в цій області сума інтенсивностей – найвища. Довжина сторони цього квадрату свідомо не відома, тому можна прийняти її залежною від размерів зображення, наприклад від його ширини. Ця залежність породжує ще одну похибку, тому що від розмірів цієй області буде залежити і результат пошуку. Якщо квадрат пошуку буде більший, ніж квадрат, який можна вписати в гранічну окружність маркеру тоді найдений центр маркеру почне залежити від артефактів і дефектів сканування, які можуть розтошовуватися поряд з маркером, а менший розмір породить похибку, яка буде лажежити від якості друку в області цього маркера. Залежність розмерів маркеру віз

розмерів скану визначаються після формування бланку. Щоб мінімізувати ці похибки, треба друкувати та сканувати бланки максимально доступним якісним образом.

В даному випадку, на бланку розтошовано два маркери: лівий і правий. Лівий треба шукати як найбільш інтенсивну область у лівій половині зображення, а правий – у правій, також не має сенсу виконувати пошук в нижній половині аркушу.

Після того, як координати маркерів були знайдені, необхідно обчислити інші характеристики координатної системи, в якій вони розтошовані. Ціна ділення обчислюється за допомогою відстані між маркерами. Кут повороту цієї системи – це кут між маркерами. Лівий маркер – це точка відліку, в даному випадку – ліва верхня. Працючи з системою, відкладемо координатну вісь з центром в найденому центрі лівого маркера. Вісь абсцис направлена праворуч, а вісь ординат – вниз, так само як у структурі даних System.Drawing.Bitmap. В даному випадку, удобно працювати з координатами, які є відносними до відстані між маркерами, тому що ця відстань є репрезентацією реальної фізичної велічини на скані. Для того, щоб здобути доступ до пікселя у новій системі координат, треба повернути задані відносні координати на кут між маркерами, помножити їх на відстань між маркерами та прибавити коодінати лівого маркера. Таким чином, можно здобути фактичні координати шуканої точки на зображенні. Формула локалізації точки в системі координат маркерів зображена на рисунку 2.8. В даному випадку, маркери знаходяться в одній системі координат з усім вмістом бланку.

$$\begin{cases} mx = x * markerDistance, \\ my = y * markerDistance, \\ x = mx * cos(a) - my * sin(a) + leftMarkerX, \\ y = mx * sin(a) + my * cos(a) + leftMarkerY; \end{cases}$$

Рис 2.8. Формула локалізації точки в системи координат маркерів

Позначення на формулі, що зображена на рисунку 2.8: mx, my – відносні координати поточної точки, які можна здобути шляхом зіставлення фактичних координат з відстанню між маркерамі на зображенні для калібровки, на якому треба винайти всі відносні відстані; x, y – координати шуканої точки; leftMarkerX, leftMarkerY – координати центра лівого маркеру; a – кут між маркерами; markerDistance – відстань між маркерами.

Процес фінального збору та аналізу даних – це фільтрація здобутих даних і трансформація їх у кінцеві. На даному етапі локалізовані області шуканих клітинок на зображенні. Далі, необхідно, для кожної клітинки підрахувати суму чорних пікселів. Це можна зробити, використовуючи алгоритм локалізування однієї точки та підрахувати суму всіх чорних точок, що находяться в клітинки, найдену суму зберігти та повторювати цей процес з усіми клітинками. Таким чином, будуть здобуті інтенсивністі всіх клітинок, у цьому полягає суть збору даних. Чим більше інтенсивність у клітинки, тим більше в ній чорних пікселів. Пусті клітинки будуть мати найменьшу інтенсивність, а заповненні – найбільшу. Задача аналізу даних, полягає в тому, щоб відділити заповнені клітинки від пустих - впорядкувати дані інтенсивностей клітинок за зменшенням та розділити його на дві частини в тому місці, де різниця між сусідними значеннями буде найбільшою, як було показано в гістограмі інтенсивностей. Ліва частина буде містити заповненні клітинки, а права – пусті.

2.2 Проектування програмного забезпечення

Даний проект містить автономну реалізацію інструменту для розпізнавання бланків для демонстрації роботи алгоритмів графічної обробки та аналізу даних. В майбутньому ця програма стане модулем великого комплексу, який буде вирішувати задачі, яки стосуються виключно розпізнавання бланків.

Для забезпечення невисокого рівню складності інтеграції даного модулю, для його реалізації було обрано спільний стек технологій – Microsoft .NET Framework та мова програмування C#.

В будь-якому випадку, програма повинна мати графічний інтерфейс користувача та виконуватися безпосередньо на доступному ньому апаратному забезпеченні. Необхідно обрати технологію, за допомогою которою буде реалізовано графічний інтерфейс користувача.

Microsoft .NET Framework містить дві технології представлення графічного інтерфейсу з елементами управління – це Winfows Presentation Foundation та WinForms.

Windows Presentation Foundation (WPF) - система для побудови клієнтських додатків Windows з візуально привабливими можливостями взаємодії з користувачем, графічна (презентаційна) підсистема у складі .NET Framework (починаючи з версії 3.0), яка використовує мову розмітки XAML.

В основі WPF лежить векторна система візуалізації, яка не залежить від дозволу пристрою виводу і створена з урахуванням можливостей сучасного графічного обладнання. WPF надає кошти для створення візуального інтерфейсу, включаючи мову XAML (Extensible Application Markup Language), елементи управління, прив'язку даних, макети, двомірну і тривимірну графіку, анімацію, стилі, шаблони, документи, текст, мультимедіа і оформлення.

Графічної технологією, що лежить в основі WPF, є DirectX, на відміну від Windows Forms, де використовується GDI / GDI + [4]. Продуктивність WPF вище, ніж у GDI + за рахунок використання апаратного прискорення графіки через DirectX.

Windows Forms - інтерфейс програмування додатків (API), що відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Даний інтерфейс спрощує доступ до елементів інтерфейсу

Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді.

Додаток Windows Forms є подієво-орієнтований додаток, підтримуване Microsoft .NET Framework. На відміну від пакетних програм, велика частина часу витрачається на очікування від користувача будь-яких дій, як, наприклад, введення тексту в текстове поле або натиснення мишою по кнопці.

Усередині .NET Framework, Windows Forms реалізується в рамках простору імен System.Windows.Forms.

Для реалізації даного програмного забезпечення, було обрано бібліотеку WinForms, як інструмент для побудови графічного чнтерфейсу користувача, тому що дане API складається з класичних принципів будування віконного інтерфейсу в операційній системі Microsoft Windows.

Для того, щоб розробити віконний додаток на технології WinForms, необхідно реалізувати клас, породжений від стандартного класу Form, щоб включити в проект подієву систему WinForms.

Запуск подієвої системи забезпучує функція Run стандартного класу Application, яка очікує форму користувача як віхдний параметр.

Такий підхід можна також визначити як спосіб побудови комп'ютерної програми, при якому в коді (як правило, в головний функції програми) явно виділяється головний цикл програми, тіло якого складається з двох частин: вибірки події і обробки події.

Подійно-орієнтоване програмування (англ. event-driven programming; надалі ПОП) - парадигма програмування, в якій виконання програми визначається подіями - діями користувача (клавіатура, миша), повідомленнями інших програм і потоків, подіями операційної системи (наприклад, надходженням мережевого пакета).

Як правило, в реальних завданнях виявляється неприпустимим тривале виконання обробника події, оскільки при цьому програма не може реагувати

на інші події. У зв'язку з цим при написанні подійно-орієнтованих програм часто застосовують автоматне програмування.

Подійно-орієнтоване програмування, як правило, застосовується в трьох випадках:

- при побудові користувацьких інтерфейсів (в тому числі графічних);
- при створенні серверних застосунків у разі, якщо з тих чи інших причин небажано породження обслуговуючих процесів;
- при програмуванні ігор, в яких здійснюється управління значною кількістю об'єктів.

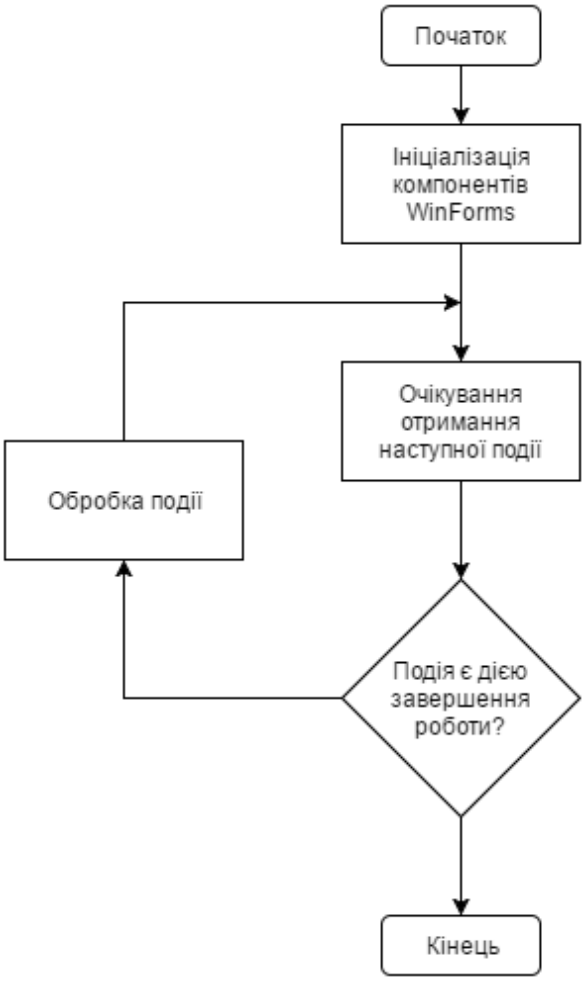

Рис. 2.9. Алгоритм стандартної подієвої моделі

На рисунку 2.9 зображено алгоритм обробки подій для додатку, зосновaного на базі класу Form. Подіє генерує операційна система, це може бути події зміни стану вікна, дій користувака, завершення роботи і т. д.

Операційна система розподілює всі події до окремих додатків. Адресація подій забезпечує їх відправку до тих додатків, до котрих вони були створені, це означає, що додаток не буде отримувати жодних подій, які його не стосуються, тому програма повинна обробити кожну отриману подію.

Дані механізми вже реалізовані в стандартній бібліотеці, тому бібліотека WinForms полегшує процес розробки та подальшого інтегрування даного програмного забезпечення.

Бібліотека WinForms містить багатий склад стандартних елементів користувача, які повністю покривають нужди даного проекту.

Програмний інтерфейс користувача повинен складатися з головного вікна, на якому буде реалізовано більшіст функцій програми, та з допоміжного діалоговогу вікна, на я кому буде розтошовано гістограми інтенсивностей. Також, програмний інтерфейс буде містити допоміжний елемент, який даст змогу користувачу візуально оцінити вихідні дані програми та внести коректировки за бажанням.

Цифрове передставлення бланку наочно показує результат работи алгоритмів аналізу паперового бланку. Він містить у собі таку ж кількість клітинок, які можуть бути заповнени кольором, або ні. Заповнені кольором клітинки позначають, що дана клітинка була розпізнана як закреслена. Також, за допомогою натиснення мишою всередині клітинки, користувач може редагувати вихідні дани програми. Для удобства користувача, в цифровому передставленні бланку, так саме як і у справжньому бланку, є два стовці та нумерація всіх питань. Для програмної реалізації цього елементу користувача було використано стандартний клас PictureBox. Елемент застосовує клас System.Drawing.Bitmap для формування графичной інформації. Графічні дані можуть буди динамічно завантажені до даного елементу незалежно від його фактичного розміру на формі та будуть масштабовані таким чином, щоб покрити максимальну площу елементу.

Елемент цифрового передставлення бланку для оцінки виконаної роботи алгоритмів аналізу вхідного зображення показано на рисунку 2.10.

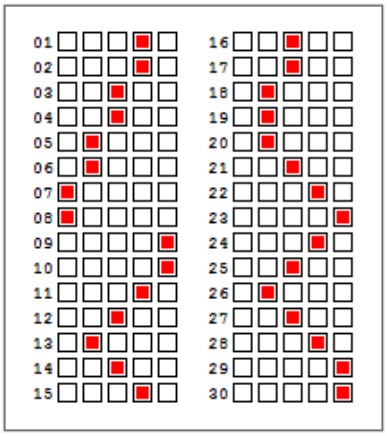

Рис. 2.10. Цифрове передставлення бланку

Гістограма інтенсивностей надає корисну інформацю користувачеві щодо протекання роботи алгоритмів аналізу. На ньому зображені суми інтеснивностей для усіх клітинок. На основі ціх інтенсивностей, алгоритм визначає, є ця клітинка закресленою, чи ні. Цей графік є двовимірним, він має дві осі (v – інтенсивність, та i – індекс). Інтенсивність клітинки, в даному випадку – це сума чорних пікселів у локалізованої області цієї клітинки. Графік генерується динамічно, тому може бути вписаний у будь-яке вікно. Елемент застосовує клас PictureBox для виводу графичной інформації та клас System.Drawing.Bitmap для маніпулювання нею. Інструмент, який містить у собі графік інтенсивностей клітинок, зображе на рисунку 2.11.

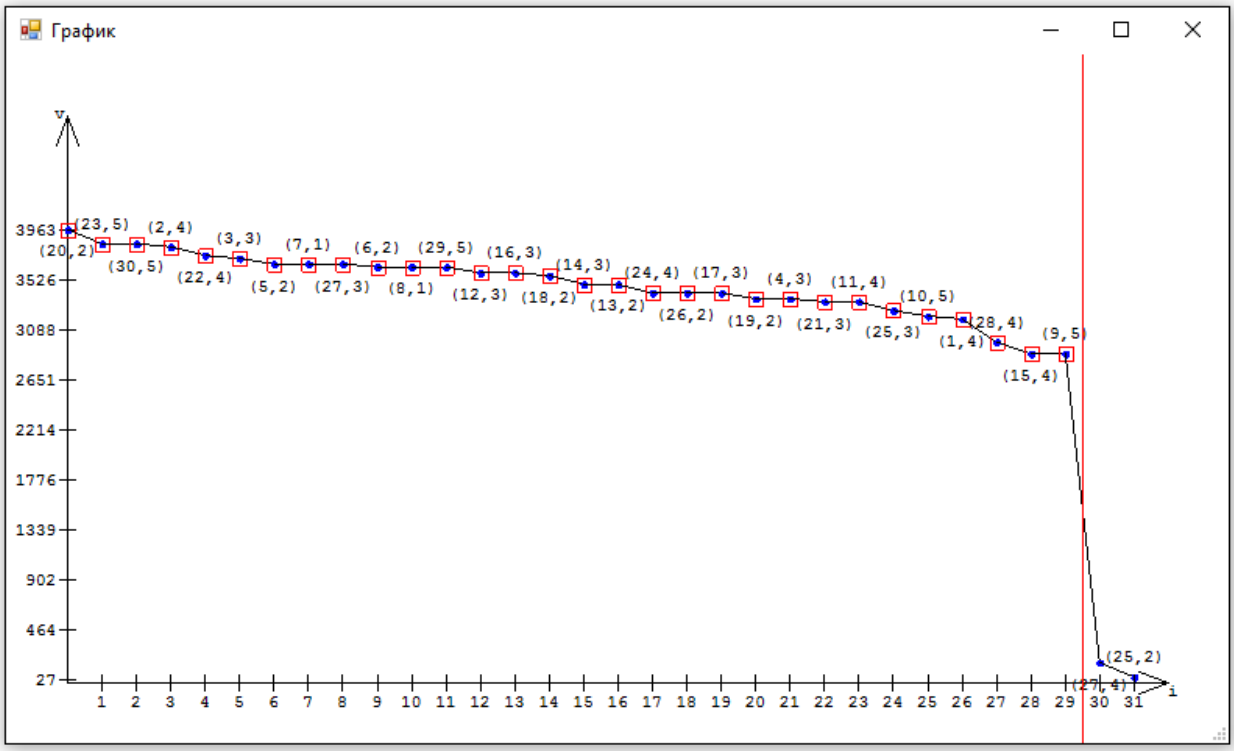

Рис. 2.11. Інструмент з гістограмою інтенсивностей клітинок

Гістограма – це спосыб представлення таблычних даних. Точки на графіку – це клітинки, які мають свій номер, який складається з двох частин: з номеру питання та номеру відповіді. На графіку присутня вертикальна лінія, яка розділяє заповнені клітинки з пустими. Усі клітинки на даному графіку впорядковані за спаданням їх інтенсивностей. Красна линія проводиться між двох точок, різниця між інтенсивностями яких є найбільшою. Таким самим чином працює алгоритм збору та аналізу даних, цей інструмент це наочно демонструє.

Вікно з гістограмою інтенсивностей э діалоговим та буде відкриватися при натисненні мишою на зменшеному графіку, який буде розтошований на основному вікни. Графічні дани гістограми повинні генеруватися таким чином, щоб вона заповнила весь корисний простір вікна та відновлювалась при змінень його розмірів.

WinForms надає можливість реалізувати всі вимоги цього проекту та є зручним у використанні, також він має широку підтримку, тому що

підтримуються кожною сучасною версію .NET Framework, яку можна знайти та працює ефективно.

Програмний інтерфейс також містить декілька елементів, які надають користувачеві ще більше інформації, щодо аналізу: список з усіма заповненими клітинками, координати маркерів. Найдени маркери відмічаються перехрестями для того, щою користувач мог бути впевненим, що всі області клітинок будуть локалізовані, також для цього усі ці області помічаються кольором, програма має елемент користувача для просмотру завантаженого зображення бланку. Не завжди можливо точно локалізувати області клітинок через похибки, яки виникають на тому чи іншому єтапі розпізнавання, тому користувач повинен мати інструменти не тільки для аналізу результатів роботи алгоритмів розпізнавання, а також інструмент, який надає йому можливість самостійно редагувати кінцеві дані, якщо вони на його погляд, є невірними. Результат дуже залежить від якості апаратного забезпечення сканеру, від його налаштувань, від відміток, які зробив студент на аркуші та від угла поворота цього аркушу в сканері. Існує дуже багато факторів, які треба врахувати та навіть, якщо теоретично всі нюанси були враховані, це не дає гарантії в коректності вихідних даних. Від них залежить оцінка знань студента, а вона завжди має бути справедливою.

Графічний інтерфейс головного вікна користувача зображено на рисунку 2.12. Основну площу вікна займає вхідне зображення. Для того, щоб завантажити зображення до програми, користувач може натиснути на кнопку «Відкрити скан», яка розтошована в головному меню та викликає діалогове вікно відкриття файлу.

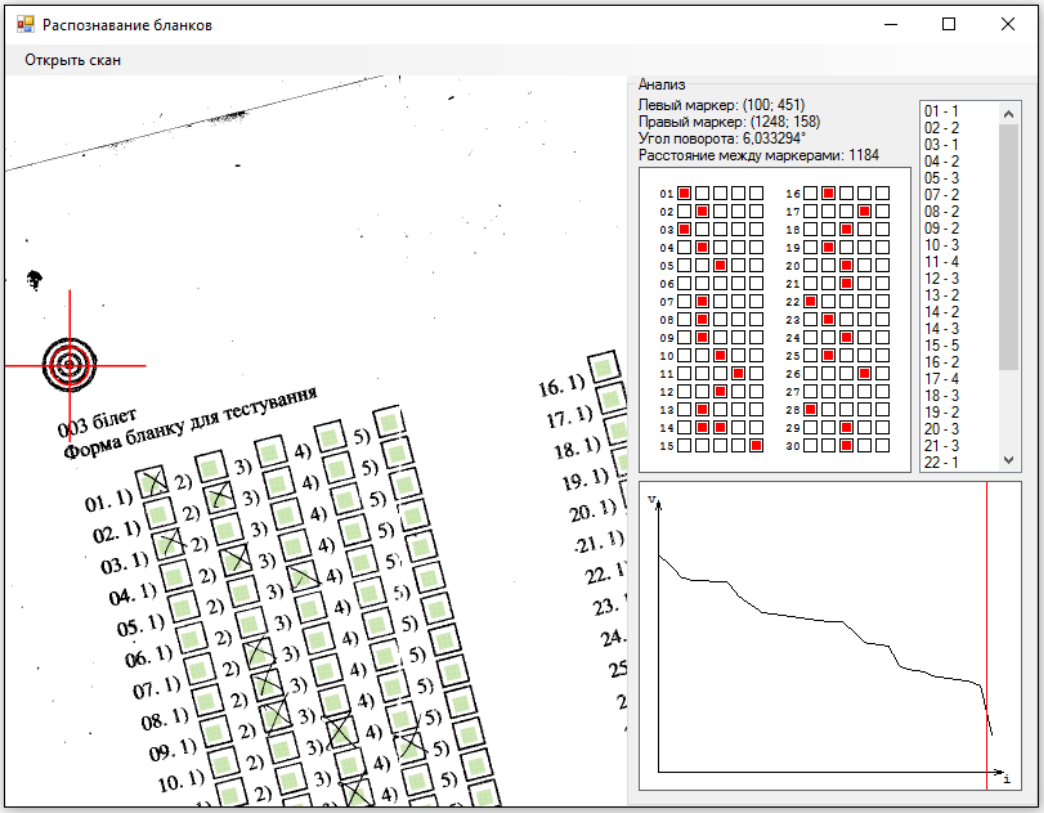

Рис. 2.12. Графічний інтерсейс програми

На вхідному зображенні було вирішено зобразити деякі допоміжни фігури, такі як – червоні перехрестя в містах автоматично знайдених центрів маркерів та зелені квадратні області, що візуально позначують автоматично локалізовані області клітинок в результаті роботи алгоритму програми для того, щоб можна було зробити висновки, можуть лі бути дані взагалі оброблені коректно. Розташування ціх областей в більшій мірі належить від точності знайдених центрів обох маркерів ті в большій мірі впливають на корректність вихідних даних.

Користувачеві предоставлено точні дані, щодо розташування маркерів не тільки в графічному вигляді, але і в числовому. Після завантаження вхідних даних і після того, як вони будуть оброблені, буде виведена ця інформація за допомогою елементів клристувача Label, розтошованих в правої частині главного вікна.

Архітектура класу Form влаштована таким чином, що будь-яку форму можно легко зробити частиною іншого проекту та об’єднювати їх в системи інтерфейсів, налагоджувати взаємодію. Це дуже важливий фактор, враховуюч те, що дане програмне забезпечення розроблене як модуль для подальшої інтеграції в материнську програму.

Microsoft .NET — програмна технологія, запропонована фірмою Microsoft як платформа для створення як звичайних програм, так і веб-застосунків. Багато в чому є продовженням ідей та принципів, покладених в технологію Java. Одною з ідей .NET є сумісність служб, написаних різними мовами. Хоча ця можливість рекламується Microsoft як перевага .NET, платформа Java має таку саму можливість.

Кожна бібліотека (збірка) в .NET має свідчення про свою версію, що дозволяє усунути можливі конфлікти між різними версіями збірок.

.NET — крос-платформова технологія, в цей час існує реалізація для платформи Microsoft Windows, FreeBSD (від Microsoft) і варіант технології для ОС Linux в проекті Mono (в рамках угоди між Microsoft з Novell), DotGNU.

Захист авторських прав відноситься до створення середовищ виконання (CLR — Common Language Runtime) для програм .NET. Компілятори для .NET випускаються багатьма фірмами для різних мов вільно.

.NET поділяється на дві основні частини — середовище виконання (по суті віртуальна машина) та інструментарій розробки.

Середовища розробки .NET-програм: Visual Studio .NET (C++, C#, J#), SharpDevelop, Borland Developer Studio (Delphi, C#) тощо. Середовище Eclipse має додаток для розробки .NET-програм. Застосовні програми також можна розроблювати в текстовому редакторі та використовувати консольний компілятор.

Як і технологія Java, середовище розробки .NET створює байт-код, призначений для виконання віртуальною машиною. Вхідна мова цієї машини в .NET називається CIL (Common Intermediate Language), також відома як

MSIL (Microsoft Intermediate Language), або просто IL. Застосування байт-кода дозволяє отримати крос-платформовість на рівні скомпільованого проекту (в термінах .NET: збірка), а не на рівні сирцевого тексту, як, наприклад, в C. Перед запуском збірки в середовищі виконання (CLR) байт-код перетворюється вбудованим в середовище JIT-компілятором (just in time, компіляція на льоту) в машинні коди цільового процесора.

Слід зазначити, що один з перших JIT-компіляторів для Java був також розроблений фірмою Microsoft (тепер в Java використовується досконаліша багаторівнева компіляція — Sun HotSpot). Сучасна технологія динамічної компіляції дозволяє досягнути аналогічного рівня швидкодії з традиційними «статичними» компіляторами (наприклад, C++) і питання швидкодії часто залежить від якості того чи іншого компілятора.

Байт-код — машинно-незалежний код низького рівня, що генерується транслятором і виконуваний інтерпретатором. Більшість інструкцій байт-кода еквівалентні одній або кільком командам асемблера. Трансляція в байт-код займає проміжне положення між компіляцією в машинний код і інтерпретацією.

Машинний код, маши́нна мова в інформатиці — набір команд (інструкцій), які виконуються безпосередньо центральним процесором комп'ютера без транслятора. Кожен тип центрального процесора має власний машинний код. Оскільки машинний код складається повністю з двійкових чисел (бітів), більшість програмістів пишуть програми на мовах програмування високого рівня. Програми, написані такими мовами, повинні транслюватися в машинний код, що здійснює компілятор або інтерпретатор програм, до того, як комп'ютер починає їх виконувати. В той час як прості процесори виконують інструкції одна за одною, суперскалярні процесори здатні виконувати декілька інструкцій одночасно.

Кожен процесор або сімейство процесорів має свій власний набір команд (інструкцій) машинного коду. Кожна машинна інструкція виконує певну дію, такими є операції з даними (наприклад, додавання чи копіювання

машинного слова в регістрі або в пам'яті) або перехід до іншої частини коду (зміна порядку виконання; при цьому перехід може бути безумовним або умовним, залежним від результатів попередніх інструкцій). Будь-яка виконувана програма складається з послідовності таких атомарних машинних операцій.

Машинний код можна розглядати як примітивну мову програмування або як найнижчий рівень представлення скомпільованих або ассембльованих комп'ютерних програм. Хоча цілком можливо створювати програми прямо в машинному коді, зараз цього практично ніхто не робить. Якщо потрібно написати оптимізовану програму під певний мікропроцесор то використовують переважно різні види мови Асемблера. Також, якщо під рукою немає початкового коду програми на високій мові програмування, а необхідно внести певні зміни в код, зламати код і т.п. використовують дизасемблери, програми, що перетворюють машинний код у асемблерний, який більш зрозумілий кваліфікованим спеціалістам. За допомогою дизасемблери в деяких випадках можна змінювати машинний код таким чином що непотрібна повна перекомпіляція програм. Написання коду на машинному коді і наближених до нього асемблерних мовах доволі трудомістка задача, потребує хорошого знання будови мікропроцесора та апаратних засобів. Тому переважна більшість програм пишеться мовами більш високого рівня і транслюється в машинний код компіляторами. Існують також спеціальні декомпілятори, які дозволяють перетворити машинний код в код на мові високого рівня.

Програми на інтерпретованих мовах (таких як Бейсик або Python) не транслюються в машинний код; замість цього вони або виконуються безпосередньо інтерпретатором мови, або транслюються у псевдокод (байт-код), який згодом виконується інтерпретатори. Самі інтерпретатори - це програми представлені в машинному коді. Загалом виконання псевдокоду не значно повільніше за виконання машинного коду і значно швидше за виконанням інтерпретатором звичайного коду зрозумілого людині. Програми

на Java зазвичай передаються на цільову машину у вигляді байт-кода, який перед виконання транслюється в машинний код «на льоту» — за допомогою JIT-компіляції.

Байт-код називається так, тому що довжина кожного коду операції — один байт, але довжина коду команди різна. Кожна інструкція є однобайтовим кодом операції від 0 до 255, за яким слідують такі параметри, як регістри або адреси пам'яті. Це в типовому випадку, але специфікація байт-кода значно розрізняється в мові.

Програма на байт-коді зазвичай виконується інтерпретатором байт-кода (зазвичай він називається віртуальною машиною, оскільки подібний до комп'ютера). Перевага — в портованості, тобто один і той байт-код може виконуватися на різних платформах і архітектурі. Ту ж саму перевагу дають мови, що інтерпретуються. Проте, оскільки байт-код зазвичай менш абстрактний, компактніший і більш «комп'ютерний» ніж початковий код, ефективність байт-кода зазвичай вища, ніж чиста інтерпретація початкового коду, призначеного для правки людиною. З цієї причини багато сучасних інтерпетованих мов насправді транслюють в байт-код і запускають інтерпретатор байт-кода. До таких мов відносяться Perl, PHP і Python. Програми на Java зазвичай передаються на цільову машину у вигляді байт-кода, який перед виконання транслюється в машинний код «на льоту» — за допомогою JIT-компіляції. У стандарті відкритих завантажувачів Open Firmware фірми Sun Microsystems байт-код представляє оператори мови Forth.

В той же час можливе створення процесорів, для яких даний байт-код є безпосередньо машинним кодом (такі процесори існують, наприклад, для Java і Forth).

Just-in-time compilation (JIT) (також відома як dynamic translation) — компіляція «на льоту» — це технологія збільшення продуктивності програмних систем, що використовують байт-код, шляхом трансляції байт-коду в машинний код безпосередньо під час роботи програми. У такий спосіб

досягається висока швидкість виконання за рахунок збільшення споживання пам'яті (для зберігання результатів компіляції) і витрат часу на компіляцію. JIT комбінує переваги інтерпретації та статичної компіляції.

Віртуальна машина — модель обчислювальної машини, створеної шляхом віртуалізації обчислювальних ресурсів: процесора, оперативної пам'яті, пристроїв зберігання та вводу і виводу інформації.

Віртуальна машина, на відміну від програми емуляції конкретного пристрою, забезпечує повну емуляцію фізичної машини чи середовища виконання (для програми).

Раніше віртуальну машину визначали як «ефективну ізольовану копію реальної машини». Проте сучасні віртуальні машини можуть не мати прямого апаратного аналогу. Наприклад, в залежності від способу моделювання набору інструкцій віртуального центрального процесора, віртуальна машина може моделювати реальну або абстрактну обчислювальні машини. При моделюванні реальної обчислювальної машини набір інструкцій процесора віртуальної машини збігається з набором інструкцій обраного для моделювання центрального процесора.

Віртуальні машини поділяються на 2 головні категорії, в залежності від їх використання та відповідності до реальної апаратури:

- системні (апаратні) віртуальні машини, що забезпечують повноцінну емуляцію всієї апаратної платформи і відповідно підтримують виконання операційної системи;
- прикладні віртуальні машини, які розробленні для виконання лише застосунків (прикладних програм), наприклад, Віртуальна машина, що входить в .NET Framework.

Системні віртуальні машини дозволяють розподіл апаратних ресурсів фізичної машини між різними копіями віртуальних машин, на кожній з яких може бути встановлена своя операційна система. Пласт програмного забезпечення, що виконує віртуалізацію, називається гіпервізором. Гіпервізори поділяються на 2 типи: ті, що можуть виконуватися на «голій»

апаратурі, та ті, що виконуються в певній операційній системі. Основні переваги системних ВМ:

- різні операційні системи можуть співіснувати на одному комп'ютері, і при цьому знаходитися в строгій ізоляції одна від одної
- ВМ можуть забезпечувати розширений набір машинних інструкцій, адже при моделюванні абстрактної обчислювальної машини набір інструкцій процесора віртуальної машини може бути довільним.
- широкі можливості контролю за програмами
- легкість модифікацій та відновлення

Основний недолік: віртуальна машина не така ефективна як реальна, тому що доступ до апаратури в ній відбувається опосередковано. Різні ВМ, на кожній з яких може бути встановлена своя власна ОС (які також називається гостьовими ОС), часто використовуються для серверного об'єднання: різні сервіси (що повинні виконуватися на окремих машинах, щоб запобігти взаємовтручанню) запускаються в різних ВМ, проте на одній фізичній машині, що дозволяє економити апаратні ресурси.

Прикладні віртуальні машини виконують звичайні програми всередині ОС. Вони зазвичай створюються коли програма запускається та знищуються після її завершення. Їхня ціль — забезпечити платформно-незалежне програмне середовище, яке дозволяє абстрагуватися від конкретної апаратури та операційної системи, на якій виконується програма.

Прикладна ВМ забезпечує високорівневу абстракцію (наприклад, інтерпретатори високорівневих мов програмування — Lisp, Java, Python, Perl), в той час як системні ВМ зазвичай обмежуються низькорівневою абстракцією (машинним набором кодів). Сучасні прикладні ВМ, що реалізуються за допомогою інтерпретаторів, для підвищення швидкості виконання використовують компіляцію «на льоту» (англ. JIT — just-in-time).

Розробка Microsoft технології .NET Framework почалась у 1999 році. Офіційно про розробку нової технології було оголошено 13 січня 2000 року.

В цей день керівництвом компанії була озвучена нова стратегія, яка отримала назву Next Generation Windows Services (скор. NGWS, укр. Нове покоління служб Windows). Нова стратегія повинна була об'єднати у єдине вже існуючі і майбутні розробки Microsoft для надання можливості користувачам працювати з Всесвітньою павутиною з допомогою безпровідних пристроїв, що мають доступ в Інтернет, як зі стаціонарних комп'ютерів.

РОЗДІЛ 3
ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Основний програмний комплекс, частиною якого повинна стати дана програма, використовує СУБД для доступу до БД і зберігання всіх даних. Система контролю знань має багато таблиць, але розроблена програма для розпізнавання бланків повинна взаємодіїти тільки з однією з них – з таблицею, яка містить результати проведення тестування. Опис структури даних цієї таблиці наведен у таблиці 3.1.

Таблиця 3.1

№ п/п	Назва поля	Ім’я поля	Тип даних	Властивості
1	Id	id	Числовий	Первинний ключ
2	Назва питання	title_test	Текстовий	
3	Оцінка	mark	Числовий	
4	Кількість відповідей	answers_count	Числовий	
5	Кількість запитань	qustions_count	Числовий	
6	Кількість правильних відповідей	correct	Числовий	
7	Кількість направильних відповідей	incorrect	Числовий	
8	Час початку тестування	beginTime	Дата/Час	
9	Час закінчення тестування	endTIme	Дата/Час	
10	Id студента	id_student	Числовий	Зовнішний ключ
11	Id группи	id_group	Числовий	Зовнішний ключ
12	Дата додання	add_time	Дата/Час	

В системі контролю знань також зберігається інформація про всі запитання, які може отримати студент при виконанні роботи. Перед проведенням тестування, програма перемішує всі питання случайним образім, орієнтуючись тільки на їх вагу.

Вага питання – це кількість балів, яку отримає студент, якщо даст правильну відповідь на це питання. Питання составляються таким чином, щоб одержити питання різної складністі для більш дітальної оцінки. Крім того, в бланку майже завжди є декілька питань з однаковою вагою, таким чином можна планувати оціночний матеріал достатньо універсально. При кожної генерації, якщо в системі зареєстровано достатня кількість питань, буде створено бланк, який не буде містити у собі багато спільних з іншими білетами патянь, це досягається завзяки великої вариативності отриманих даних під час перемішування, настільки, наскільки достатня їх кількість. Дані методі дозволяють ускладнити списування та навпаки, дати більше пристіру студенту завдяки різних ваг питань, які можуть значно перевищувати деякі попередні.

Після того, як питання були перемішані з оглядом на їх вагу для кожногу бланку, студент отримує завдання – відповісти на всі ці питання на бланку. Модуль, який відповідає за розпізнавання, перетворює скановані бланки в інформацію, яку можливо зберігти у заданих структурах БД, це основна задача даного модулю після того, як він буде інтегрований в комплекс контролю знань.

Клієнт-серверна СУБД розташовується на сервері разом з БД і здійснює доступ до БД безпосередньо, в монопольному режимі. Всі клієнтські запити на обробку даних обробляються клієнт-серверної СУБД централізовано. Недолік клієнт-серверних СУБД полягає в підвищених вимогах до сервера.

При роботі з реляційними базами даних використовують мову структурних запитів SQL (Structured Query Language), яка поєднує всі три функції (визначення даних, модифікація даних та формування вибірок). Мова

SQL стандартизована ANSI та ISO: починаючи з 1986 року, регулярно виходять поновлені стандарти. Слід зауважити, що кожна сучасна СКБД (MySQL, PostgreSQL, Microsoft SQL Server та інші) підтримує свою власну видозміну SQL, так що SQL-запит для однієї СКБД може не працювати в середовищі іншої. Але головні принципи формування SQL-запитів та їх структура однакові та відповідають стандартам ANSI/ISO. При необхідності виконання якоїсь операції над даними клієнт формує лінгвістичну конструкцію мовою SQL, яку називають SQL-запитом, і надсилає її до СКБД. СКБД опрацьовує запит, і результат його виконання (наприклад, вибірку даних) повертає клієнту. Мова, якою оперує СКБД, також може містити засоби для:

- конфігурування СКБД;
- модифікації, форматування даних та розрахунків;
- формування обмежень даних.

Взаємодія з БД – задача комплексу контролю знань, а не даного модулю. Дана програма буде взаємодіїти з материнським програмним забезпеченням за допомогою взаємодії об’єктів на основних принципах ООП. Так як обидві програми реалізовані на мові програмування C#, це створює сприятливі умови для інтеграції, ця особливість – результат проектування даного програмного забезпечення.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗАДАЧІ

4.1 Опис головного модулю програми

Головний модуль програми породжен від класу Form бібліотеки WinForms та містить у собі інші елементи управління. Усі основні алгоритми реалізуються в цьому модулі.

Першим чином, необхідно розробити інтерфейс модулю, завдяки інструментам графічного редактору інтерфейсів, що входить у склад середовища розробки Microsoft Visual Studio, яке повністю підтримує мову програмування C# та набір бібліотек .NET Framework.

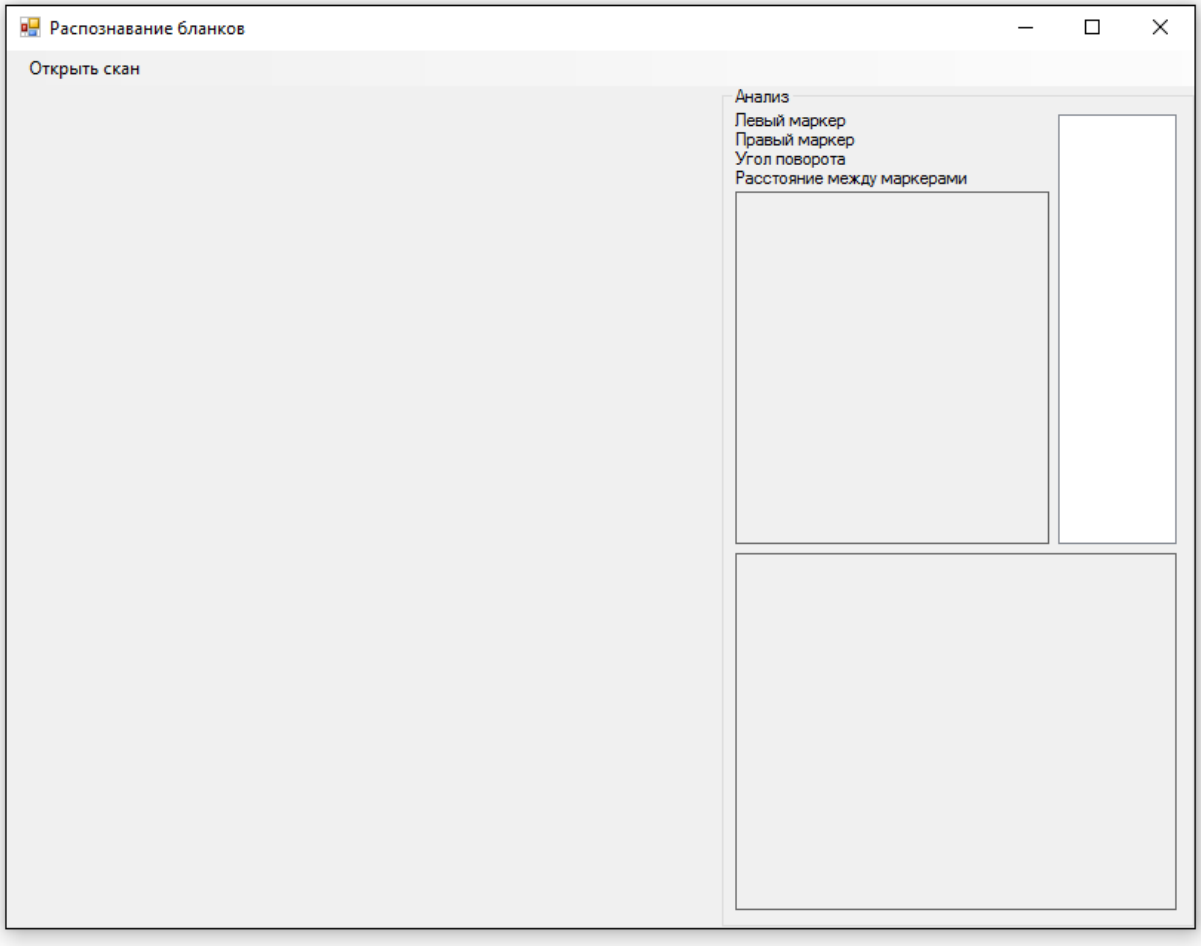

Рис. 4.1 Графічний інтерфейс головного модулю

На рисунку 4.1 зображено розроблений графічний інтерфейс головного модулю програми на базі класу Form.

Головний модуль містить у собі елементи управління, такі як:

- MenuStrip – головне меню програми, яке маже містити інші різнорівневі під-меню, зараз містить пункт меню для завантаження вхідних даних;
- PictureBox – зображення вхідних даних, використовується для візуального представленя завантаженного бланку для аналізу; також, в лівої частини форми, в контейнері GroupBox розтошовано два елемента управління PictureBox з цифровим представленням бланку (результат аналізу) та гістограмою інтенсивностей клітинок;
- GroupBox – елемент для групування інших елементів, використовується як контейнер для інструментів, що надають інформацію щодо аналізу;
- Label – ці елементи використовуються для виведення текстової інформації, наприклад виведення координат маркерів та відстані між ними, кутом між ними;
- ListBox – список, який використовуються для виводу номерів відмічених клітинок після того, як бланк був аналізован.

Для цифрового представлення бланку – елементу, що розроблен на базі класу PictureBox, для контролю вихідних даниї, треба реалізувати додаткову логіку при натисненні по ньому за допомогою подійної системи, яку надає WinForms.

При натисненні мишою по елементу графічного представлення бланку, знаючи координати миші в цей момент, обчислюється індекс клітинки, на котру було здійснено натиснення. Далі, змінюється стан цієї клітинки. Як і було описано раньше, структура даних Cell, яка зберігає дані про клітинку, має спеціальне поле Force, яке використовується для зберігань змін стану клітинки користувачем, якщо він потім не скасував ці зміни. Таким чином, якщо користувач натисне на заповнену клітинку, вона стане порожньою, а

поле Force встановиться значенням -1. Якщо користувач натисне на порожню клітинку, вона заповниться, а поле Force встановиться значенням 1. Якщо користувач натисне на клітинку, поле Force якої відрізняється від 0, воно буде встановлене в 0, а сама клітинка повернеться в свій нормальний стан згідно результатів роботи алгоритмів розпізнавання.

Гістограма також реалізована за допомогою класу PictureBox. На головному модулі міститься спрощена гістограма інтенсивностей. При натисненні мишою по ній відкривається діалогове вікно, яке містить детальну масштабовану гістограму.

Детальна гістограма має ціни ділення та додаткову індикацію точок на графіку, пронумеровані точки з номерами кожної відповіді.

4.2 Опис створених функцій

Під час реалізації програмного забезпечення, виникла необхідність розробити функції для того, щоб мінімізувати розміри поточного коду. Усі функції були реалізовані всередині головного модулю на базі класу Form та мають можливість взаємодіяти в рамках їх батьківського обїекту.

У сучасних мовах програмування події та обробники подій є центральною ланкою реалізації графічного інтерфейсу користувача. Розглянемо, наприклад, взаємодію програми з подіями від миші. Натискання правої клавіші миші викликає системне переривання, що запускає певну процедуру всередині операційної системи. У цій процедурі відбувається пошук вікна, що знаходиться під курсором миші. Якщо вікно знайдено, то дана подія надсилається в чергу обробки повідомлень цього вікна. Далі, в залежності від типу вікна, можуть генеруватися додаткові події. Наприклад, якщо вікно є кнопкою (у Windows всі графічні елементи є вікнами), то додатково генерується подія натискання на кнопку. Відмінність останньої події в тому, що вона більш абстрактна, а саме, не містить координат

курсору, а говорить просто про те, що було вироблено натискання на цю кнопку.

Далі приведені функції, та їх описа які були створені в даному програмному забезпеченні:

- IntegralImageAndBradleyLocalThresholding – реалізація алгоритма адаптивного порогування, отримує графічні дані в об’єкті класу Bitmap на вхід;
- DrawSight – рисує червоне перехрестя на заданому полотні; використовується після винаходження координат маркерів, коли користувач завантажує вхідні дані;
- DrawGraph – рисує гістограму на заданому полотні, є змога задати режим підвищеної деталізації;
- DrawBlank – рисує бланк з заповненними або порожніми клітинками на основі вихідної інформації;
- DrawToolGraph – функція оновленя графычноъ інформації в діалоговому вікні;
- pictureBoxChecked_Click – функція обробки натиснення мишою на елементі цифрового представлення бланку, оброблює зміни користувача;
- formListBox – формирує елемент списку з вихідними даними;
- pictureBoxGraph_Click – викливається при натисненні мишою на спрощеному варіанті гістограми на головному вікні, відкриває діалогове вікно з детальною гістограмою;
- Recognize – основний алгоритм розпізнавання, формує вихідні дані;
- openScanToolStripMenuItem_Click – дія, яка визивається при натиснені на кнопку «Відкрити бланк», яка розтошована в голоному меню програми. При натиснені на неї, відкривається діалогове вікно відкриття файлу, потім файл завантажується до структури динах Bitmap, оброблюється за допомогою функції

IntegralImageAndBradleyLocalThresholding та викликається процес аналізу за допомогою функції Recognize.

Реалізація програмного забезпеченню з використанная інтерфейсу користувача на базі WinForms має здібності як подійно-орієнтованої, так і об’єктно-орієнтованої парадигми.

Об'єктно-орієнтоване програмування (ООП) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають три основні концепції: інкапсуляція, успадкування та поліморфізм. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дозволив писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм.

Об'єктно-орієнтоване програмування - це метод програмування, заснований на поданні програми у вигляді сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування. Програмісти спочатку пишуть клас, а на його основі при виконанні програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється ієрархія класів.

Об'єктно-орієнтований підхід полягає в наступному наборі основних принципів:

- все є об'єктами;
- всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, при якій один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення — це запит на виконання

дії, доповнений набором аргументів, які можуть знадобитися при виконанні дії;

- кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів;
- кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів;
- у класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії;
- класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається ієрархією успадкування. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Таким чином, програма являє собою набір об'єктів, що мають стан та поведінку. Об'єкти взаємодіють використовуючи повідомлення. Будується ієрархія об'єктів: програма в цілому — це об'єкт, для виконання своїх функцій вона звертається до об'єктів що містяться у ньому, які у свою чергу виконують запит шляхом звернення до інших об'єктів програми. Звісно, щоб уникнути безкінечної рекурсії у зверненнях, на якомусь етапі об'єкт трансформує запит у повідомлення до стандартних системних об'єктів, що даються мовою та середовищем програмування. Стійкість та керованість системи забезпечуються за рахунок чіткого розподілення відповідальності об'єктів (за кожну дію відповідає певний об'єкт), однозначного означення інтерфейсів міжоб'єктної взаємодії та повної ізольованості внутрішньої структури об'єкта від зовнішнього середовища (інкапсуляції).

ООП виникло в результаті розвитку ідеології процедурного програмування, де дані і підпрограми (процедури, функції) їх обробки формально не пов'язані. Для подальшого розвитку об'єктно-орієнтованого програмування велике значення мають поняття події (так зване подієво-

орієнтоване програмування) і компоненти (компонентне програмування, КОП).

Формування КОП від ООП відбулося, так само як формування модульного від процедурного програмування: процедури сформувалися в модулі - незалежні частини коду до рівня збірки програми, так об'єкти сформувалися в компоненти - незалежні частини коду до рівня виконання програми. Взаємодія об'єктів відбувається за допомогою повідомлень. Результатом подальшого розвитку ООП, мабуть, буде агентно-орієнтоване програмування, де агенти - незалежні частини коду на рівні виконання. Взаємодія агентів відбувається за допомогою зміни середовища, в якій вони знаходяться.

Мовні конструкції, конструктивно не пов'язані безпосередньо з об'єктами, але необхідні їм для їх безпечної (виняткові ситуації, перевірки) та ефективної роботи, інкапсулюються від них в аспекти (в аспектно - орієнтованому програмуванні). Суб'єктно-орієнтоване програмування розширює поняття об'єктів шляхом забезпечення більш уніфікованого і незалежної взаємодії об'єктів. Може бути перехідною стадією між ООП та агентним програмуванням в частині самостійної їх взаємодії.

До визначень об'єктно-орієнтованого програмування належить:

- клас, що визначає абстрактні характеристики деякої сутності, включаючи характеристики самої сутності (її атрибути або властивості) та дії, які вона здатна виконувати (її поведінки, методи або можливості). Як правило, клас має бути зрозумілим для не-програмістів, що знаються на предметній області, що, у свою чергу, значить, що клас повинен мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються його членами;
- об'єкт - окремий екземпляр класу (створюється після запуску програми і ініціалізації полів класу);

- метод - можливості об'єкта. В межах програми, використання методу має впливати лише на один об'єкт;
- обмін повідомленнями - передача даних від одного процесу іншому, або надсилання викликів методів.»;
- успадкування (наслідування): клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватись цілі дерева успадкування. Підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть вводити свої власні. Успадкування може бути одиничне (один безпосередній батьківський клас) та множинне (кілька батьківських класів). Це залежить від вибору програміста, який реалізовує клас та мови програмування. Так, наприклад, в Java дозволене лише одинарне успадкування, а в C++ - і те і інш;
- приховування інформації (інкапсуляція): приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення. Інкапсуляція досягається шляхом вказування, які класи можуть звертатися до членів об'єкта. Як наслідок, кожен об'єкт представляє кожному іншому класу певний інтерфейс — члени, доступні іншим класам. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись. Це дозволить полегшити внесення змін, тобто, без потреби змінювати і користувачів інтерфейсу. Часто, члени класу позначаються як публічні (англ. public), захищені (англ. protected) та приватні (англ. private), визначаючи, чи доступні вони всім класам, підкласам, або лише до класу в якому їх визначено. Деякі мови програмування йдуть ще далі: Java використовує ключове слово private для обмеження доступу, що буде дозволений лише з методів того самого класу, protected — лише з методів того самого класу і його нащадків та з класів із того ж самого пакету, C# та VB.NET

відкривають деякі члени лише для класів із тієї ж збірки шляхом використання ключового слова internal (C#) або Friend (VB.NET), а Eiffel дозволяє вказувати які класи мають доступ до будь-яких членів.

- абстрагування - спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми;
- поліморфізм, що означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення. На практиці - це реалізовується шляхом реалізації ряду підпрограм (функцій, процедур, методи тощо) з однаковими іменами, але з різними параметрами. В залежності від того, що передається і вибирається відповідна підпрограма.

В об'єктно-орієнтованому програмуванні, клас — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, згідно з термінологією ООП, глобальні змінні класу (члени-змінні) називаються полями даних (також властивостями або атрибутами), а члени-функції називають методами класу. Створений та ініціалізований екземпляр класу називають об'єктом класу. На основі одного класу можна створити безліч об'єктів, що відрізнятимуться один від одного своїм станом (значеннями полів).

На основі класів можна створювати підкласи, які успадковують властивості та поведінку батьківських класів. Таким чином можна створити цілу ієрархію класів. Різні мови дещо по різному реалізовують механізм успадкування. Існує множинне та одинарне успадкування. Множинне — це, коли підклас створюється на основі кількох безпосередніх батьків (як то в мові програмування C++). Одинарне успадкування — це коли клас може мати одного безпосереднього батька (мова програмування Java). Надкласи

можуть мати свої надкласи, підкласи можуть також бути надкласами для певних класів.

Через методи реалізується поведінка об'єктів. Практично вся робота з об'єктами відбувається через методи. Вони можуть змінювати стан об'єкта або ж просто надавати доступ до даних, які були інкапсульовані в об'єкті. Існує кілька видів методів, які мають деякі відмінності в різних мовах програмування. До методів та полів даних можна надавати різні права доступу, від яких залежатиме доступ до них з різних частин програмного коду. Права доступу та вид методів задаються модифікаторами при описі методів. Метод, який проводить створення та початкову ініціалізацію екземпляра класу називають конструктором класу. Метод, який проводить знищення об'єкта, називають деструктором класу.

У програмуванні існує поняття програмного інтерфейсу, що означає перелік можливих обчислень, які може виконати та чи інша частина програми, включаючи опис того, які аргументи і в якому порядку потрібно передавати на вхід алгоритмам з цього переліку, а також що і в якому вигляді вони будуть повертати. Абстрактний тип даних інтерфейс придуманий для формалізованого опису такого переліку. Самі алгоритми, тобто дійсний програмний код, який буде виконувати всі ці обчислення, інтерфейсом не задається, програмується окремо та називається реалізацією інтерфейсу.

Програмні інтерфейси, а також класи, можуть розширюватися шляхом спадкування, яке є одним з важливих засобів повторного використання готового коду в ООП. Успадкований клас або інтерфейс буде містити в собі все, що зазначено для всіх його батьківських класів (в залежності від мови програмування та платформи, їх може бути від нуля до нескінченності). Наприклад, можна створити свій варіант текстового рядка шляхом успадкування класу «мій рядок тексту» від вже існуючого класу «рядок тексту», при цьому передбачається, що програмісту не доведеться заново переписувати алгоритми пошуку та інше, оскільки вони автоматично будуть успадковані від готового класу, і будь-який екземпляр класу «мій рядок

тексту» може бути переданий не лише в готові методи батьківського класу «рядок тексту» для проведення потрібних обчислень, а й взагалі в будь-який алгоритм, здатний працювати з об'єктами типу «рядок тексту», оскільки екземпляри обох класів сумісні по програмним інтерфейсам.

Клас дозволяє задати не лише програмний інтерфейс до самого себе і до своїх екземплярів, але і в явному вигляді написати код, відповідальний за обчислення. Якщо при створенні свого нового типу даних успадковувати інтерфейс, то ми отримаємо можливість передавати примірник свого типу даних в будь-який алгоритм, який вміє працювати з цим інтерфейсом. Однак нам доведеться самим написати реалізацію інтерфейсу, тобто ті алгоритми, якими буде користуватися цікавий нам алгоритм для проведення обчислень з використанням нашого екземпляра. Водночас, при наслідуванні класу, автоматично успадковується готовий код під інтерфейс (це не завжди так, батьківський клас може вимагати реалізації якихось алгоритмів в дочірньому класі в обов'язковому порядку). В такій можливості успадковувати готовий код і проявляється те, що в об'єктно-орієнтованої програми тип даних клас визначає одночасно як інтерфейс, так і реалізацію для всіх своїх екземплярів.

Правила, що визначають можливість або неможливість безпосередньо змінювати будь-які змінні, називаються правилами завдання областей доступу. Слова «приватний» та «публічний» в цьому випадку є так званими «модифікаторами доступу». Вони називаються «модифікаторами» тому, що в деяких мовах вони використовуються для зміни раніше встановлених прав при спадкуванні класу. Спільно класи та модифікатори доступу задають область доступу, тобто у кожної ділянки коду, залежно від того, до якого класу вона належить, буде своя область доступу щодо тих чи інших елементів (членів) свого класу та інших класів, включаючи змінні, методи, функції, константи, тощо. Існує основне правило: ніщо в одному класі не може бачити приватних елементів іншого класу. Щодо інших, більш складних правил, у різних мовах існують інші модифікатори доступу та правила їх взаємодії з класами.

Майже кожному члену класа можна встановити модифікатор доступу (за винятком статичних конструкторів та деяких інших речей). У більшості об'єктно-орієнтованих мов програмування підтримуються такі модифікатори доступу:

- private (закритий, внутрішній член класу) — звернення до члену допускаються лише з методів того класу, у якому цей член визначений. Будь-які спадкоємці класу вже не зможуть отримати доступ до цього члену. Спадкування за типом private забороняє доступ з дочірнього класу до всіх членів батьківського класу, включаючи навіть public-члени (C++);
- protected (захищений, внутрішній член ієрархії класів) — звернення до члена допускаються з методів того класу, у якому цей член визначений, а також з будь-яких методів його класів-спадкоємців. Спадкування за типом protected робить все public-члени батьківського класу protected-членами класу-спадкоємця (C++);
- public (відкритий член класу) — звернення до члена допускаються з будь-якого коду. Спадкування за типом public не міняє модифікаторів батьківського класу (C++).

Проблема підтримки правильного стану змінних актуальна і для самого першого моменту виставлення початкових значень. Для цього в класах передбачені спеціальні методи/функції, звані конструкторами. Жоден об'єкт (екземпляр класу) не може бути створений інакше, як шляхом виклику на виконання коду конструктора, який поверне викликає стороні створений та правильно заповнений примірник класу. У багатьох мовах програмування тип даних «структура», як і клас, може містити змінні та методи, але екземпляри структур, залишаючись просто розміченими ділянками оперативної пам'яті, можуть створюватися в обхід конструкторам, що заборонено для примірників класів (за винятком спеціальних виняткових методів обходу всіх подібних правил ООП, передбачених в деяких мовах та

платформах). У цьому виявляється відмінність класів від інших типів даних — виклик конструктора обов'язковий.

У сучасних об'єктно-орієнтованих мовах програмування (в тому числі в php, Java, C++, Oberon, Python, Ruby, Smalltalk, Object Pascal) створення класу зводиться до написання деякої структури, що містить набір полів та методів (серед останніх особливу роль грають конструктори, деструктори, фіналізатори). Практично клас може розумітися як якийсь шаблон, за яким створюються об'єкти — екземпляри цього класу. Усі примірники одного класу створені за одним шаблоном, тому мають один і той же набір полів та методів.

В об'єктно-орієнтованому програмуванні (ООП), об'єкт є окремою одиницею сховища даних під час роботи програм, що використовується як базовий елемент побудови програм. Ці об'єкти можуть взаємодіяти один з одним, на противагу традиційним поглядам, відповідно до яких програма розглядається як набір підпрограм, або просто перелік інструкцій комп'ютера. Кожний об'єкт здатний отримувати повідомлення, обробляти дані та надсилати повідомлення іншим об'єктам. Кожний об'єкт може розглядатись як незалежний малий автомат або актор, з визначеним призначенням або відповідальністю.

Поліморфізм — концепція в програмуванні та теорії типів, відповідно до якої використовується спільний інтерфейс для обробки різних спеціалізованих типів.

Поліморфізм — один з трьох найважливіших механізмів об'єктно-орієнтованого програмування. Поліморфізм дозволяє писати більш абстрактні програми і підвищити коефіцієнт повторного використання коду.

Спільні властивості об'єктів об'єднуються в систему, яку можуть називати по різному: інтерфейс, клас. Спільність має зовнішнє і внутрішнє вираження. Зовнішня спільність проявляється як однаковий набір методів з однаковими іменами і сигнатурами (типами аргументів і результатів).

Внутрішня спільність є однакова функціональність методів. Її можна описати інтуїтивно виразити у вигляді строгих законів, правил, яким повинні підкорятись методи. На противагу поліморфізму, концепція мономорфізму вимагає однозначного зіставлення.

Таким чином, для реалізації програмного забезпеченню даного проекту була застосована суміш об’єктно-орієнтованої ті подіє-орієнтованої парадигм, що є типовим прийомом у розробці інтерактивних графічних інтерфейсів користувача. Фінальна імплементація відповідає вимогам ящі стосуються інтерфейсів, що ставлять процеси аналізу та розпізнавання, а самі алгоритми були імплементовані найбільш ефективними методами, які доступні в даному випадку.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено функціонуюче програмне забезпечення для автоматизування контролю знань студентів, основною функцією якого є розпізнання бланків з відповідями.

Розроблена система розпізнання бланків створена спеціально для подальшої інтеграції в існуючий комплекс контролю знань, яка паралельно розробляється студентом четвертого курсу вищого навчального закладу КНУ КЕІ Єрьоміним Іваном Володимировичем. Програмне забезпечення має гнучки налаштування та невеликий обсяг коду, що надає їй можливість до швидкої адаптації до змін стандартів.

У ході вирішення поставлених на початку роботи задач отримані знання в сфері комп’ютерного зору та розпізнавання образів. Були сформовані специільні бланки, придатні для подальшого аналізу.

Усі алгоритми аналізу даних біли складені з початку і до кінця без застосування жодних сторонніх бібліотек. Завдяки цьому, був отриман багатий опит в проектуванні та реалізації подібних систем.

Інтерфейс користувача розроблен таким, щоб надати зручні функції контроля вихідних даних. Користувач був забезпечений інформативними інструментами для аналізу здобутих даних. Використані технології дозволяють запуск програмного запезпечення на будь-якому сучасному персоняльному компь’ютері.

Під час реалізації програмного забезпечення, був здобут опит у роботі з технологіями Microsoft .NET Framework та мовою C#. Цей стек технологій, на сьогодняшній день, є одним з найбільш затребуваних і є актуальним. Мова програмування C# є дуже гнучкою та підійде майже до будь-якої прикладної задачі ті являються потужним інструментом для розробки сучасного програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adaptive Thresholding Using the Integral Image: Derek Bradley (Carleton University, Canada; derek@derekbradley.ca), Gerhard Roth (National Research Council of Canada) Gerhard.Roth@nrc-cnrc.gc.ca
2. C# 5.0 и платформа .NET 4.5 [Електронний ресурс]. – Режим доступу: http://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php
3. Абрамян М. Visual C# на примерах. - СПб.: БХВ-Петербург, 2008. – 496с.
4. Гуннерсон Э. Введение в C#. Библиотека программиста. – СПб.:Питер, 2001. – 304 с.
5. Культин. Н.Б. Microsoft Visual C# в задачах и примерах. - СПб. : БХВ-Петербург, 2009. – 634 с.
6. Левитин, Ананий В. Алгоритмы: введение в разработку и анализ. - М.: «Вильямс», 2006. – 576 с.
7. Нейгел К. Visual C#. Учебное пособие. – М.:Диалектика, 2010. – 710 с.
8. Нейгл, Кристиан, Ивьен, Билл, Глинн, Джей, Скиннер, Морган, Уотсон, Карли. C# 2005 и платформа .NET 3.0 для профессионалов. - М.: «Вильямс», 2008. – 528 с.
9. Павловская Т.А. Программирование на языке высокого уровня: учебник для вузов. - СПб.: Питер, 2013. 432 с.
10. Стэкер Метью А., Стэйн Стивен Дж., Нортроп Тони. Разработка клиентских Windows-приложений на платформе Microsoft .NET Framework. - СПб.: Питер, 2008. – 624 с.
11. Фролов А.В. Язык C#: Самоучитель. – Диалог, 2003. – 560 с.
12. Компьютерное зрение. Современный подход - Дэвид А. Форсайт, Джин Понс 2004 ISBN: 5-8459-0542-7
13. Компьютерное зрение - Шапиро Л., Стокман Дж. 2009 – 763 с.

14. Распознавание образов. Введение в методы статистического обучения, 2011 г., 256 с., ISBN 978-5-354-01337-1
15. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. Учебник, 400 с., 2015 г. ISBN 978-5-97060-273-7, 978-1-107-09639-4
16. An Introduction to Statistical Learning with Applications in R Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani – Springer, ISBN 978-1-4614-7137-0
17. «Адаптивное управление сложными системами на основе теории распознавания образов» В. С. Симанков, Е. В. Луценко, Краснодар: Техн. ун-т Кубан. гос. технол. ун-та, 1999. — 318 с.
18. Trevor Hastie, Robert Tibshirani, Jerome Friedman - Springer Series in Statistics, The Elements of Statistical Learning, Data Mining, Inference, and Prediction – Springer 2008
19. Визильтер Ю.В., Желтов С.Ю., Бондаренко А.В. и др. Обработка и анализ изображений в задачах машинного зрения, Физматкнига 2010 - 672 с. - ISBN: 978-5-89155-201-2
20. Цифровая обработка изображений в среде MATLAB, Гонсалес Р., Вудс Р., Эддинс С., Издательство: Техносфера, 2006
21. Цифровая обработка изображений, Яне Бернд, 584 с., 2007, ISBN 978-5-94836-122-2, 3-540-24035-7

ДОДАТКИ

ДОДАТОК А
Файл, що містить точку входу Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Rec {
   static class Program {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main() {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new MainWindow());
      }
   }
}
```

ДОДАТОК Б
Дизайн-файл GraphWindow.Designer.cs

```
namespace Rec {
   partial class GraphWindow {
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.IContainer components = null;

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
      protected override void Dispose(bool disposing) {
         if (disposing && (components != null)) {
            components.Dispose();
         }
         base.Dispose(disposing);
      }

      #region Windows Form Designer generated code

      /// <summary>

      /// Required method for Designer support - do not modify
```

Продовження додатку Б

```
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent() {
            this.SuspendLayout();
            //
            // GraphWindow
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(784, 441);
            this.DoubleBuffered = true;
            this.Name = "GraphWindow";
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "График";
            this.Load += new System.EventHandler(this.GraphWindow_Load);
            this.Resize += new System.EventHandler(this.GraphWindow_Resize);
            this.ResumeLayout(false);

        }

        #endregion
    }
}
```

Файл імплементації GraphWindow.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Rec {
    public partial class GraphWindow : Form {
        private MainWindow mainWindow;

        public GraphWindow(MainWindow mainWindow) {
            InitializeComponent();
            this.mainWindow = mainWindow;
        }

        private void GraphWindow_Load(object sender, EventArgs e) {
            mainWindow.DrawToolGraph();
        }
```

Продовження додатку Б

```
        private void GraphWindow_Resize(object sender, EventArgs e) {
            mainWindow.DrawToolGraph();
        }
    }
}
```

ДОДАТОК В

Дизайн-файл MainWindow.Designer.cs

```
namespace Rec {
    partial class MainWindow {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing) {
            if (disposing && (components != null)) {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent() {
            this.menuStrip = new System.Windows.Forms.MenuStrip();
            this.openScanToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.openFileDialog = new System.Windows.Forms.OpenFileDialog();
            this.saveFileDialog = new System.Windows.Forms.SaveFileDialog();
            this.labelLeftMarker = new System.Windows.Forms.Label();
            this.labelRightMarker = new System.Windows.Forms.Label();
            this.labelAngle = new System.Windows.Forms.Label();
            this.labelMarkersDistance = new System.Windows.Forms.Label();
            this.groupBoxAnalysis = new System.Windows.Forms.GroupBox();
            this.pictureBoxGraph = new System.Windows.Forms.PictureBox();
            this.pictureBoxChecked = new System.Windows.Forms.PictureBox();
            this.listBoxChecked = new System.Windows.Forms.ListBox();
```

Продовження додатку В

```
            this.pictureBoxBlank = new System.Windows.Forms.PictureBox();
            this.menuStrip.SuspendLayout();
            this.groupBoxAnalysis.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBoxGraph)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBoxChecked)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBoxBlank)).BeginInit();
            this.SuspendLayout();
            //
            // menuStrip
            this.menuStrip.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.openScanToolStripMenuItem});
            this.menuStrip.Location = new System.Drawing.Point(0, 0);
            this.menuStrip.Name = "menuStrip";
            this.menuStrip.Size = new System.Drawing.Size(802, 24);
            this.menuStrip.TabIndex = 1;
            this.menuStrip.Text = "menuStrip";
            //
            // openScanToolStripMenuItem
            //
            this.openScanToolStripMenuItem.Name = "openScanToolStripMenuItem";
            this.openScanToolStripMenuItem.Size = new System.Drawing.Size(94, 20);
            this.openScanToolStripMenuItem.Text = "Открыть скан";
            this.openScanToolStripMenuItem.Click += new
System.EventHandler(this.openScanToolStripMenuItem_Click);
            //
            // openFileDialog
            //
            this.openFileDialog.Filter =
"Изображения(*.BMP;*.JPG;*.PNG)|*.BMP;*.JPG;*.PNG|Все файлы (*.*)|*.*";
            this.openFileDialog.Title = "Открытие";
            //
            // saveFileDialog
            //
            this.saveFileDialog.Filter = "Документ RTF(*.rtf)|*.rtf";
            this.saveFileDialog.Title = "Сохранение";
            //
            // labelLeftMarker
            //
            this.labelLeftMarker.AutoSize = true;
            this.labelLeftMarker.Location = new System.Drawing.Point(6, 16);
            this.labelLeftMarker.Name = "labelLeftMarker";
            this.labelLeftMarker.Size = new System.Drawing.Size(82, 13);
            this.labelLeftMarker.TabIndex = 3;
            this.labelLeftMarker.Text = "Левый маркер";
            //
            // labelRightMarker
            //
            this.labelRightMarker.AutoSize = true;
            this.labelRightMarker.Location = new System.Drawing.Point(6, 29);
```

Продовження додатку В

```
this.labelRightMarker.Name = "labelRightMarker";
this.labelRightMarker.Size = new System.Drawing.Size(88, 13);
this.labelRightMarker.TabIndex = 4;
this.labelRightMarker.Text = "Правый маркер";
//
// labelAngle
//
this.labelAngle.AutoSize = true;
this.labelAngle.Location = new System.Drawing.Point(6, 42);
this.labelAngle.Name = "labelAngle";
this.labelAngle.Size = new System.Drawing.Size(82, 13);
this.labelAngle.TabIndex = 5;
this.labelAngle.Text = "Угол поворота";
//
// labelMarkersDistance
//
this.labelMarkersDistance.AutoSize = true;
this.labelMarkersDistance.Location = new System.Drawing.Point(6, 55);
this.labelMarkersDistance.Name = "labelMarkersDistance";
this.labelMarkersDistance.Size = new System.Drawing.Size(164, 13);
this.labelMarkersDistance.TabIndex = 6;
this.labelMarkersDistance.Text = "Расстояние между маркерами";
//
// groupBoxAnalysis
//
this.groupBoxAnalysis.Controls.Add(this.pictureBoxGraph);
this.groupBoxAnalysis.Controls.Add(this.pictureBoxChecked);
this.groupBoxAnalysis.Controls.Add(this.listBoxChecked);
this.groupBoxAnalysis.Controls.Add(this.labelLeftMarker);
this.groupBoxAnalysis.Controls.Add(this.labelRightMarker);
this.groupBoxAnalysis.Controls.Add(this.labelMarkersDistance);
this.groupBoxAnalysis.Controls.Add(this.labelAngle);
this.groupBoxAnalysis.Dock = System.Windows.Forms.DockStyle.Right;
this.groupBoxAnalysis.Location = new System.Drawing.Point(483, 24);
this.groupBoxAnalysis.Name = "groupBoxAnalysis";
this.groupBoxAnalysis.Size = new System.Drawing.Size(319, 568);
this.groupBoxAnalysis.TabIndex = 8;
this.groupBoxAnalysis.TabStop = false;
this.groupBoxAnalysis.Text = "Анализ";
//
// pictureBoxGraph
//
this.pictureBoxGraph.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
this.pictureBoxGraph.Location = new System.Drawing.Point(9, 315);
this.pictureBoxGraph.Name = "pictureBoxGraph";
this.pictureBoxGraph.Size = new System.Drawing.Size(298, 241);
this.pictureBoxGraph.TabIndex = 10;
this.pictureBoxGraph.TabStop = false;
this.pictureBoxGraph.Click += new System.EventHandler(this.pictureBoxGraph_Click);
```

Продовження додатку В

```
            //
            // pictureBoxChecked
            //
            this.pictureBoxChecked.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle;
            this.pictureBoxChecked.Location = new System.Drawing.Point(9, 71);
            this.pictureBoxChecked.Name = "pictureBoxChecked";
            this.pictureBoxChecked.Size = new System.Drawing.Size(212, 238);
            this.pictureBoxChecked.TabIndex = 9;
            this.pictureBoxChecked.TabStop = false;
            this.pictureBoxChecked.Click += new
System.EventHandler(this.pictureBoxChecked_Click);
            //
            // listBoxChecked
            //
            this.listBoxChecked.FormattingEnabled = true;
            this.listBoxChecked.Location = new System.Drawing.Point(227, 19);
            this.listBoxChecked.Name = "listBoxChecked";
            this.listBoxChecked.Size = new System.Drawing.Size(80, 290);
            this.listBoxChecked.TabIndex = 8;
            //
            // pictureBoxBlank
            //
            this.pictureBoxBlank.Dock = System.Windows.Forms.DockStyle.Fill;
            this.pictureBoxBlank.Location = new System.Drawing.Point(0, 24);
            this.pictureBoxBlank.Name = "pictureBoxBlank";
            this.pictureBoxBlank.Size = new System.Drawing.Size(802, 568);
            this.pictureBoxBlank.TabIndex = 2;
            this.pictureBoxBlank.TabStop = false;
            //
            // MainWindow
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(802, 592);
            this.Controls.Add(this.groupBoxAnalysis);
            this.Controls.Add(this.pictureBoxBlank);
            this.Controls.Add(this.menuStrip);
            this.MainMenuStrip = this.menuStrip;
            this.Name = "MainWindow";
            this.Text = "Распознавание бланков";
            this.Load += new System.EventHandler(this.MainWindow_Load);
            this.menuStrip.ResumeLayout(false);
            this.menuStrip.PerformLayout();
            this.groupBoxAnalysis.ResumeLayout(false);
            this.groupBoxAnalysis.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBoxGraph)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBoxChecked)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBoxBlank)).EndInit();
            this.ResumeLayout(false);
```

Продовження додатку В

```
            this.PerformLayout();
        }

        #endregion
        private System.Windows.Forms.MenuStrip menuStrip;
        private System.Windows.Forms.ToolStripMenuItem openScanToolStripMenuItem;
        private System.Windows.Forms.OpenFileDialog openFileDialog;
        private System.Windows.Forms.SaveFileDialog saveFileDialog;
        private System.Windows.Forms.PictureBox pictureBoxBlank;
        private System.Windows.Forms.Label labelLeftMarker;
        private System.Windows.Forms.Label labelRightMarker;
        private System.Windows.Forms.Label labelAngle;
        private System.Windows.Forms.Label labelMarkersDistance;
        private System.Windows.Forms.GroupBox groupBoxAnalysis;
        private System.Windows.Forms.ListBox listBoxChecked;
        private System.Windows.Forms.PictureBox pictureBoxChecked;
        private System.Windows.Forms.PictureBox pictureBoxGraph;
    }
}
```

Файл імплементації MainWindow.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Collections;

namespace Rec {
    public partial class MainWindow : Form {
        private CellIndexComparer cic; // Компаратор для сотировки клеток по номеру (вопрос
+ ответ)
        private Bitmap bitmapBlank, bitmapBlankOriginal;
        private GraphWindow graphWindow;
        public Cell[] cells; // Массив с информацией о клетках, формируется после анализа
        public int maxDiffI, m_size; // maxDiffI - количество отмеченных клеток (по порядку от
0 до maxDiffI не включительно)
                                     // в массиве Cells после анализа Recognize()
        int[,] intImg; // Интегральная матрица (используется для алгоритма адаптивного
порога и для поиска маркеров)

        public class Cell : IComparable<Cell> {
```

Продовження додатку В

```
        public Cell() {
            Index = 0; // Индекс клетки (в данном случае от 0 до 150, по пять клеток на один вопрос)
            Value = 0; // Сумма интенсивностей для каждого пикселя внутри клетки
            Force = 0; // Для правки результата. Если Force = 0, клетка считается неотредактированной (без изменений)
                       // и ее состояние определяется по ее положению в массиве Cells (если меньше maxDiffI - то отмечена),
                       // если Force = 1 - клетка отмечена, если Force = -1, то клетка пуста
        }

        public Cell(int i) {
            Index = i;
            Value = 0;
            Force = 0;
        }

        public int Index { get; set; }
        public int Value { get; set; }
        public int Force { get; set; }

        override public string ToString() {
            string str = (Index / 5 + 1).ToString();
            if (str.Length == 1)
                str = '0' + str;
            return str + " - " + (Index % 5 + 1).ToString();
        }
        public int CompareTo(Cell other) {
             return Value.CompareTo(other.Value);
        }
    }

    public class CellIndexComparer : IComparer<Cell> {
        public int Compare(Cell a, Cell b) {
            return a.Index.CompareTo(b.Index);
        }
    }

    public MainWindow() {
        InitializeComponent();
        maxDiffI = -1;
    }

    private void MainWindow_Load(object sender, EventArgs e) {
        cic = new CellIndexComparer();
        graphWindow = new GraphWindow(this);
    }

    private void ShowErrorMsgBox(string text) {
```

Продовження додатку В

```
        MessageBox.Show("Внимание! Ошибка: " + text);
    }

    private void openScanToolStripMenuItem_Click(object sender, EventArgs e) {
        openFileDialog.ShowDialog();
        if (openFileDialog.FileName.Length > 0) {
            try {
                bitmapBlank = new Bitmap(openFileDialog.FileName);
                bitmapBlankOriginal = new Bitmap(bitmapBlank);
                float zoomFactor = 0.5f;
                Size newSize = new Size((int)(bitmapBlank.Width * zoomFactor),
(int)(bitmapBlank.Height * zoomFactor)); // Масштаб изображения для PictureBox
                IntegralImageAndBradleyLocalThresholding(bitmapBlank); // Подготовка
                Recognize(bitmapBlank); // Распознавание
                pictureBoxBlank.Image = new Bitmap(bitmapBlank, newSize);
            }
            catch (Exception ex) {
                ShowErrorMsgBox(ex.Message);
            }
        }
    }
    // Формирование интегральной матрицы и приведение изображения к виду после
адаптивного порога
    private void IntegralImageAndBradleyLocalThresholding(Bitmap bitmap) {
        try {
            unsafe
            {
                BitmapData bitmapData = bitmap.LockBits(new Rectangle(0, 0, bitmap.Width,
bitmap.Height), ImageLockMode.ReadWrite, bitmap.PixelFormat);
                int s = bitmapData.Width / 8;
                float t = 0.85f;
                int bytesPerPixel = Image.GetPixelFormatSize(bitmap.PixelFormat) / 8;
                int widthInBytes = bitmapData.Width * bytesPerPixel;
                byte* ptrFirstPixel = (byte*)bitmapData.Scan0; // BGR
                intImg = new int[bitmapData.Width,bitmapData.Height]; // Integral image, summed
area table
                int sum;
                for (int x = 0; x < bitmapData.Width; x++) {
                    int xBytes = x * bytesPerPixel;
                    sum = 0;
                    for (int y = 0; y < bitmapData.Height; y++) {
                        byte* currentLine = ptrFirstPixel + y * bitmapData.Stride;
                        sum += currentLine[xBytes + 2];
                        if (x == 0)
                            intImg[x, y] = sum;
                        else
                            intImg[x, y] = intImg[x - 1, y] + sum;
                    }
```

Продовження додатку В

```
            }
            int hs = s / 2;
            Parallel.For(0, bitmapData.Width, x => {
               int xBytes = x * bytesPerPixel;
               int x1, x1m, y1, y1m, x2, y2, count;
               for (int y = 0; y < bitmapData.Height; y++) {
                  byte* currentLine = ptrFirstPixel + (y * bitmapData.Stride);
                  x1 = Math.Max(0, x - hs);
                  y1 = Math.Max(0, y - hs);
                  x2 = Math.Min(bitmapData.Width - 1, x + hs);
                  y2 = Math.Min(bitmapData.Height - 1, y + hs);
                  if (x < s) {
                     x1 = 1; x2 = s;
                  }
                  else if (x > bitmapData.Width - s - 1) {
                     x1 = bitmapData.Width - s;
                     x2 = bitmapData.Width - 1;
                  }
                  if (y < s) {
                     y1 = 1;
                     y2 = s;
                  }
                  else if (y > bitmapData.Height - s - 1) {
                     y1 = bitmapData.Height - s;
                     y2 = bitmapData.Height - 1;
                  }
                  count = (x2 - x1) * (y2 - y1);
                  x1m = Math.Max(0, x1 - 1);
                  y1m = Math.Max(0, y1 - 1);
                  sum = intImg[x2, y2] - intImg[x1m, y2] - intImg[x2, y1m] + intImg[x1m,
y1m];
                  if (currentLine[xBytes + 2] * count < sum * t) {
                     currentLine[xBytes] = 0;
                     currentLine[xBytes + 1] = 0;
                     currentLine[xBytes + 2] = 0;
                  }
                  else {
                     currentLine[xBytes] = 255;
                     currentLine[xBytes + 1] = 255;
                     currentLine[xBytes + 2] = 255;
                  }
               }
            });
            bitmap.UnlockBits(bitmapData);
         }
      }
      catch (InvalidOperationException ex) {

         ShowErrorMsgBox(ex.Message);
```

Продовження додатку В

```
        }
    }


    // Рисование красного перекрестия
    private void DrawSight(Graphics g, Pen pen, int x, int y, float md) {
        g.DrawLine(pen, x - md * 0.1f, y, x + md * 0.1f, y);
        g.DrawLine(pen, x, y - md * 0.1f, x, y + md * 0.1f);
        g.DrawEllipse(pen, x - md * 0.025f, y - md * 0.025f, md * 0.05f, md * 0.05f);
    }

    // Рисование графика интенсивностей (advanced = true для большей информативности)
    private void DrawGraph(Bitmap bmp, Cell[] cells, int maxDiffI, bool advanced = false) {
        int nonZeroCells = 0;
        for (int i = 0; i < cells.Length; i++) {
            if (cells[i].Value != 0)
                nonZeroCells++;
        }
        int graphOffset = (int)(bmp.Width * 0.05f);
        int graphZeroX = graphOffset, graphZeroY = bmp.Height - graphOffset;
        int graphEndX = bmp.Width - graphOffset, graphEndY = graphOffset;
        string str;
        using (Graphics g = Graphics.FromImage(bmp)) {
            g.FillRectangle(Brushes.White, new Rectangle(0, 0, bmp.Width, bmp.Height));
            g.DrawLine(Pens.Black, graphZeroX, graphZeroY, graphZeroX, graphEndY);
            g.DrawLine(Pens.Black, graphZeroX, graphZeroY, graphEndX, graphZeroY);
            g.DrawLine(Pens.Black, graphZeroX, graphEndY, graphZeroX - graphOffset / 5, graphEndY + graphOffset / 2);
            g.DrawLine(Pens.Black, graphZeroX, graphEndY, graphZeroX + graphOffset / 5, graphEndY + graphOffset / 2);
            g.DrawLine(Pens.Black, graphEndX, graphZeroY, graphEndX - graphOffset / 2, graphZeroY - graphOffset / 5);
            g.DrawLine(Pens.Black, graphEndX, graphZeroY, graphEndX - graphOffset / 2, graphZeroY + graphOffset / 5);
            Font graphFont = new Font(new FontFamily("Courier New"), 8, FontStyle.Regular);
            g.DrawString("i", graphFont, Brushes.Black, graphEndX - 2, graphZeroY - 2);
            g.DrawString("v", graphFont, Brushes.Black, graphZeroX - 10, graphEndY - 9);
            float max = cells[0].Value * 1.25f;
            if (advanced) {
                int v_step = 32;
                for (int i = graphZeroY - (int)((graphZeroY - graphEndY) / 1.25f); i < graphZeroY; i += v_step) {
                    g.DrawLine(Pens.Black, graphZeroX - graphOffset / 7.5f, i, graphZeroX + graphOffset / 7.5f, i);
                    str = ((int)(((graphZeroY - i) / ((graphZeroY - graphEndY) / 1.25f)) * m_size * m_size)).ToString();
                    SizeF measure1 = g.MeasureString(str + ' ', graphFont);
                    g.DrawString(str, graphFont, Brushes.Black, graphZeroX - graphOffset / 7.5f - measure1.Width, i - measure1.Height / 2);                }
```

Продовження додатку В

```
                g.DrawRectangle(Pens.Red, graphZeroX - 4, graphZeroY - (graphZeroY -
graphEndY) * (cells[0].Value / max) - 4, 9, 9);
                g.FillEllipse(Brushes.Blue, graphZeroX - 2, graphZeroY - (graphZeroY -
graphEndY) * (cells[0].Value / max) - 2, 5, 5);
                str = '(' + (cells[0].Index / 5 + 1).ToString() + ',' + (cells[0].Index % 5 +
1).ToString() + ')';
                SizeF measure = g.MeasureString(str, graphFont);
                g.DrawString(str, graphFont, Brushes.Black, graphZeroX - measure.Width / 2,
graphZeroY - (graphZeroY - graphEndY) * (cells[0].Value / max) + measure.Height * 0.5f);
            }
            for (int i = 1; i < nonZeroCells; i++) {
                g.DrawLine(Pens.Black, graphZeroX + (graphEndX - graphZeroX) * ((i - 1) /
(float)nonZeroCells), graphZeroY - (graphZeroY - graphEndY) * (cells[i - 1].Value / max),
                    graphZeroX + (graphEndX - graphZeroX) * (i / (float)nonZeroCells),
graphZeroY - (graphZeroY - graphEndY) * (cells[i].Value / max));
                if (advanced) {
                    if (i < maxDiffI)
                        g.DrawRectangle(Pens.Red, graphZeroX + (graphEndX - graphZeroX) * (i /
(float)nonZeroCells) - 4, graphZeroY - (graphZeroY - graphEndY) * (cells[i].Value / max) - 4,
9, 9);
                    g.FillEllipse(Brushes.Blue, graphZeroX + (graphEndX - graphZeroX) * (i /
(float)nonZeroCells) - 2, graphZeroY - (graphZeroY - graphEndY) * (cells[i].Value / max) - 2,
5, 5);
                    str = '(' + (cells[i].Index / 5 + 1).ToString() + ',' + (cells[i].Index % 5 +
1).ToString() + ')';
                    SizeF measure = g.MeasureString(str, graphFont);
                    if (i % 2 == 0)
                        g.DrawString(str, graphFont, Brushes.Black, graphZeroX + (graphEndX -
graphZeroX) * (i / (float)nonZeroCells) - measure.Width / 2, graphZeroY - (graphZeroY -
graphEndY) * (cells[i].Value / max) + measure.Height * 0.5f);
                    else
                        g.DrawString(str, graphFont, Brushes.Black, graphZeroX + (graphEndX -
graphZeroX) * (i / (float)nonZeroCells) - measure.Width / 2, graphZeroY - (graphZeroY -
graphEndY) * (cells[i].Value / max) - measure.Height * 1.5f);
                    g.DrawLine(Pens.Black, graphZeroX + (graphEndX - graphZeroX) * (i /
(float)nonZeroCells), graphZeroY - graphOffset / 7.5f, graphZeroX + (graphEndX -
graphZeroX) * (i / (float)nonZeroCells), graphZeroY + graphOffset / 7.5f);
                    str = i.ToString();
                    g.DrawString(str, graphFont, Brushes.Black, graphZeroX + (graphEndX -
graphZeroX) * (i / (float)nonZeroCells) - g.MeasureString(str, graphFont).Width / 2,
graphZeroY + graphOffset / 7.5f);
                }
            }
            g.DrawLine(Pens.Red, graphZeroX + (graphEndX - graphZeroX) * ((maxDiffI - 0.5f)
/ nonZeroCells), 0, graphZeroX + (graphEndX - graphZeroX) * ((maxDiffI - 0.5f) /
nonZeroCells), bmp.Height - 1);
        }
    }
```

Продовження додатку В

```
// Рисование бланка с учетом состояния всех клеток и правок пользователя после анализа
private void DrawBlank(Bitmap bmp, Cell[] cells, int maxDiffI) {
    using (Graphics g = Graphics.FromImage(bmp)) {

        g.FillRectangle(Brushes.White, new Rectangle(0, 0, bmp.Width, bmp.Height));
        int boxSize = bmp.Height / 16;
        int space = 2;
        int midX = bmp.Width / 2 - ((10 + space -1) * boxSize - 1) / 2;
        int midY = bmp.Height / 2 - (15 * boxSize - 1) / 2;
        string str;
        for (int i = 0; i < 150; i++) {
            int c = i % 5;
            int r = (i % 75) / 5;
            if (i >= 75)
                c += 5 + space;
            g.DrawRectangle(Pens.Black, midX + c * boxSize, midY + r * boxSize, boxSize - 4, boxSize - 4);
            if (i % 5 == 0) {
                str = (i / 5 + 1).ToString();
                if (str.Length == 1)
                    str = '0' + str;
                g.DrawString(str, new Font(new FontFamily("Courier New"), boxSize / 2, FontStyle.Regular), Brushes.Black, midX + c * boxSize - boxSize * 1.1f, midY + r * boxSize);
            }
        }
        for (int i = 0; i < 150; i++) {
            if ((i < maxDiffI && cells[i].Force == 0) || cells[i].Force == 1) {
                int c = cells[i].Index % 5;
                int r = (cells[i].Index % 75) / 5;
                if (cells[i].Index >= 75)
                    c += 5 + space;
                g.FillRectangle(Brushes.Red, midX + c * boxSize + 2, midY + r * boxSize + 2, boxSize - 7, boxSize - 7);
            }
        }
    }
}

public void DrawToolGraph() {
    if (maxDiffI >= 0) {
        Bitmap graphBmp = new Bitmap(graphWindow.ClientSize.Width, graphWindow.ClientSize.Height);
        DrawGraph(graphBmp, cells, maxDiffI, true);
        graphWindow.BackgroundImage = graphBmp;
    }
}
```

Продовження додатку В

```
    private void pictureBoxChecked_Click(object sender, EventArgs e) {
        if (maxDiffI > 0) {
            MouseEventArgs me = (MouseEventArgs)e;
            Point coordinates = me.Location;
            Bitmap checkedBmp = new Bitmap(pictureBoxChecked.Width - 2,
pictureBoxChecked.Height - 2);
            int boxSize = checkedBmp.Height / 16;
            int space = 2;
            int midX = checkedBmp.Width / 2 - ((10 + space - 1) * boxSize - 1) / 2;
            int midY = checkedBmp.Height / 2 - (15 * boxSize - 1) / 2;
            bool found = false;
            for (int i = 0; i < 150; i++) {
                int c = i % 5;
                int r = (i % 75) / 5;
                if (i >= 75)
                    c += 5 + space;
                if (coordinates.X > midX + c * boxSize && coordinates.X < midX + c * boxSize +
boxSize - 4 &&
                    coordinates.Y > midY + r * boxSize && coordinates.Y < midY + r * boxSize +
boxSize - 4) {
                    int j = 0;
                    for (j = 0; j < cells.Length; j++)
                        if (cells[j].Index == i)
                            break;
                    if (j < maxDiffI) {
                        if (cells[j].Force == 0)
                            cells[j].Force = -1;
                        else
                            cells[j].Force = 0;
                    }
                    else {
                        if (cells[j].Force == 0)
                            cells[j].Force = 1;
                        else
                            cells[j].Force = 0;
                    }
                    found = true;
                    break;
                }
            }
            if (found) {
                DrawBlank(checkedBmp, cells, maxDiffI);
                pictureBoxChecked.Image = checkedBmp;
                formListBox();
            }
        }
    }
```

Продовження додатку В

```
        // Формирование списка с отмеченными клектами после анализа
        public void formListBox() {
            listBoxChecked.Items.Clear();
            List<Cell> lbCells = new List<Cell>();
            for (int i = 0; i < cells.Length; i++) {
                if ((i < maxDiffI && cells[i].Force == 0) || cells[i].Force == 1)
                    lbCells.Add(cells[i]);
            }
            lbCells.Sort(cic);
            // В данный момент в lbCells хранятся только отмеченные клетки с учетом правок пользователя (конечный результат для данного бланка)
            foreach (Cell cell in lbCells)
                listBoxChecked.Items.Add(cell.ToString());
            lbCells.Clear();
        }

        public void pictureBoxGraph_Click(object sender, EventArgs e) {
            if (maxDiffI >= 0) {
                graphWindow.ShowDialog(this);
            }
        }
        // Распознавание
        // ПАРАМЕТРЫ:
        // threshold - порог черного и белого пикселя (если больше либо равен threshold значит белый, иначе черный)
        // leftMarkerStartX - координата X начала квадратной зоны левого маркера
        // rightMarkerStartX - координана X начала квадратной зоны правого
        // markerY - координата Y для начала зон обоих маркеров
        // markerSize - размер каждой из зон (сторона квадрата)
        // markerCutoff - максимально допустимый результат, при привышении которого итерация поиска будет пропущена (для борьбы с дефектами сканирования)
        private void Recognize(Bitmap bitmap) {
            try {
                // http://csharpexamples.com/fast-image-processing-c/
                unsafe
                {
                    BitmapData bitmapData = bitmap.LockBits(new Rectangle(0, 0, bitmap.Width, bitmap.Height), ImageLockMode.ReadWrite, bitmap.PixelFormat);
                    int bytesPerPixel = Image.GetPixelFormatSize(bitmap.PixelFormat) / 8;
                    int widthInBytes = bitmapData.Width * bytesPerPixel;
                    byte* ptrFirstPixel = (byte*)bitmapData.Scan0; // BGR
                    //int leftMarkerMaxSumHor = 0, leftMarkerMaxSumVert = 0, rightMarkerMaxSumHor = 0, rightMarkerMaxSumVert = 0;
                    int leftMarkerX = -1, leftMarkerY = -1, rightMarkerX = -1, rightMarkerY = -1; // Искомые координаты маркеров
                    // Поиск маркеров
                    m_size = (int)(bitmapData.Width * 0.0385f);
                    int m_min = 255 * m_size * m_size, m_cur;
                    int m_sy = 1;
```

Продовження додатку В

```
            int m_fy = (int)(bitmapData.Height * 0.5f);
            for (int x = m_size + 1 + (int)(bitmapData.Width * 0.03f); x < bitmapData.Width /
2; x++)
               for (int y = m_sy + m_size; y < m_fy; y++) {
                  m_cur = intImg[x, y] - intImg[x - m_size - 1, y] - intImg[x, y - m_size - 1] +
intImg[x - m_size - 1, y - m_size - 1];
                  if (m_cur < m_min) {
                     m_min = m_cur;
                     leftMarkerX = x - m_size / 2;
                     leftMarkerY = y - m_size / 2;
                  }
               }
            m_min = 255 * m_size * m_size;
            for (int x = bitmapData.Width / 2 + m_size + 1; x < (int)(bitmapData.Width *
0.97f); x++)
               for (int y = m_sy + m_size; y < m_fy; y++) {
                  m_cur = intImg[x, y] - intImg[x - m_size - 1, y] - intImg[x, y - m_size - 1] +
intImg[x - m_size - 1, y - m_size - 1];
                  if (m_cur < m_min) {
                     m_min = m_cur;
                     rightMarkerX = x - m_size / 2; rightMarkerY = y - m_size / 2;
                  }
               }
            labelLeftMarker.Text = "Левый маркер: (" + leftMarkerX.ToString() + "; " +
leftMarkerY.ToString() + ")";
            labelRightMarker.Text = "Правый маркер: (" + rightMarkerX.ToString() + "; " +
rightMarkerY.ToString() + ")";
            // Анализ клеток
            cells = new Cell[150];
            for (int i = 0; i < 150; i++)
               cells[i] = new Cell(i);
            float angle = (float)(Math.Atan2(leftMarkerY - rightMarkerY, leftMarkerX -
rightMarkerX) + Math.PI);
            labelAngle.Text = "Угол поворота: " + angle.ToString() + '°';
            float markersDistance = (float)Math.Sqrt((leftMarkerX - rightMarkerX) *
(leftMarkerX - rightMarkerX) + (leftMarkerY - rightMarkerY) * (leftMarkerY - rightMarkerY));
            labelMarkersDistance.Text = "Расстояние между маркерами: " +
((int)markersDistance).ToString();
            float sina = (float)Math.Sin(angle);
            float cosa = (float)Math.Cos(angle);
            // x * cosa - y * sina;
            // x * sina + y * cosa;
            int firstBoxOffsetX = (int)(0.064543f * markersDistance);
            int firstBoxOffsetY = (int)(0.176471f * markersDistance);
            float boxOffsetX = 0.080882f * markersDistance;
            float boxOffsetY = 0.0423f * markersDistance;
            //int boxInnerSideHalf = (int)(0.013072f * markersDistance);
            int boxInnerSideHalf = (int)(0.01f * markersDistance);
            int columnOffsetX = (int)(0.677288 * markersDistance);
```

Продовження додатку В

```
Parallel.For(0, 150, i => {
   int bx, by, tx, ty, cx, cy, x, y, xBytes;
   byte* currentLine;
   bx = i % 5;
   by = i / 5;
   if (by < 15)
      cx = (int)(firstBoxOffsetX + boxOffsetX * bx);
   else {
      cx = (int)(columnOffsetX + boxOffsetX * bx); by -= 15;
   }
   cy = (int)(firstBoxOffsetY + boxOffsetY * by);
   for (int ix = -boxInnerSideHalf; ix <= boxInnerSideHalf; ix++)
      for (int iy = -boxInnerSideHalf; iy <= boxInnerSideHalf; iy++) {
         tx = cx + ix;
         ty = cy + iy;
         x = (int)(tx * cosa - ty * sina + leftMarkerX);
         y = (int)(tx * sina + ty * cosa + leftMarkerY);
         xBytes = x * bytesPerPixel;
         if (x > 0 && x < bitmapData.Width && y > 0 && y < bitmapData.Height) {
            currentLine = ptrFirstPixel + (y * bitmapData.Stride);
            currentLine[xBytes + 0] = (byte)Math.Max(0, currentLine[xBytes + 0] -
75);
            currentLine[xBytes + 1] = (byte)Math.Max(0, currentLine[xBytes + 1] -
25);
            currentLine[xBytes + 2] = (byte)Math.Max(0, currentLine[xBytes + 2] -
50);
            if (currentLine[xBytes] == 0)
               cells[i].Value++;
         }
      }
});
bitmap.UnlockBits(bitmapData);
using (Graphics g = Graphics.FromImage(bitmap)) {
   Pen redBold = new Pen(Brushes.Red, 3.0f);
   DrawSight(g, redBold, leftMarkerX, leftMarkerY, markersDistance);
   DrawSight(g, redBold, rightMarkerX, rightMarkerY, markersDistance);
}
Array.Sort(cells);
Array.Reverse(cells);
// Поиск наибольшей разницы
int maxDiff = 0;
for (int i = 1; i < cells.Length; i++) {
      if (cells[i - 1].Value - cells[i].Value > maxDiff) {
         maxDiff = cells[i - 1].Value - cells[i].Value;
         maxDiffI = i;
      }
}
// Сохранение результата
Array.Sort(cells, 0, maxDiffI, cic);
```

Продовження додатку В

```
                // Все отмеченные вопросы по порядку в cells (от 0 до maxDiffI), начиная с
maxDiffI - пустые клетки
                // cells[i].Index / 5 + 1 - номер вопроса
                // cells[i].Index % 5 + 1 - номер ответа
                formListBox();
                Array.Sort(cells);
                Array.Reverse(cells);
                // Формирование изображения цифрового представления бланка
                Bitmap graphBmp = new Bitmap(pictureBoxGraph.Width - 2,
pictureBoxGraph.Height - 2);
                DrawGraph(graphBmp, cells, maxDiffI);
                pictureBoxGraph.Image = graphBmp;
                // Формирование изображения цифрового представления бланка
                Bitmap checkedBmp = new Bitmap(pictureBoxChecked.Width - 2,
pictureBoxChecked.Height - 2);
                DrawBlank(checkedBmp, cells, maxDiffI);
                pictureBoxChecked.Image = checkedBmp;
            }
        }
        catch (InvalidOperationException ex) {
            ShowErrorMsgBox(ex.Message);
        }
    }
  }
}
```