УДК 004.4 DOI https://doi.org/10.35546/kntu2078-4481.2025.2.2.25

Р. І. КОРДЮК
аспірант кафедри комп'ютерних технологій
у видавничо-поліграфічних процесах
Національний університет «Львівська політехніка»
ORCID: 0009-0008-6082-901X

МОДЕЛЬ ОПРАЦЮВАННЯ КОЛАЖІВ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Проведено аналіз існуючих онлайн-технологій (Photoshop Online, Canva, Pixlr Editor) та виявлено основні недоліки їхніх моделей, головним з яких є відсутність автоматичного зв'язку між контейнерами зображень. Це призводить до значних витрат часу на ручні налаштування, а також до можливих помилок у формуванні фінальної композиції колажу. Недоліки проявляються, зокрема, у порушенні композиційної цілісності при зміні розмірів одного чи кількох елементів колажу.

У роботі також виконано аналіз таких відомих алгоритмічних моделей: автоматичного вибору та компонування важливих ділянок зображень (AutoCollage), групування зображень на основі семантичних зв'язків (Correlation-Preserving Photo Collage), створення естетичних композицій через навчання з підкріпленням (Aesthetic Photo Collage with Deep Reinforcement Learning) та диференційованого компонування колажів на основі ймовірнісних дерев (SoftCollage).

Основою проведеного дослідження стала розробка та детальний опис алгоритмів реалізації функції handleMoving, яка застосовується для автоматичної адаптації елементів колажу при їх інтерактивному переміщенні та зміні розмірів користувачами. Розроблені алгоритми для трьох основних структурних компонентів: деревоподібної ієрархічної структури колажу (CollageTree), горизонтального розміщення елементів (CollageRow) та вертикального розміщення елементів (CollageColumn). Детально описано логіку роботи цих алгоритмів, наведено блок-схеми, що ілюструють алгоритмічні процеси переміщення, перерахунку розмірів та автоматичної адаптації елементів залежно від змін виконуваних користувачами. Проведено детальний аналіз розроблених алгоритмів, який показав високу ефективність навіть за умов масштабування колажів, які містять значну кількістю зображень. Функція handleMoving, яка розроблена в межах даного дослідження, може слугувати доповненням або розширенням згаданих алгоритмічних моделей, забезпечуючи автоматичне підтримання композиційної цілісності колажів під час їх інтерактивного редагування.

Ключові слова: *інформаційна система колажів, алгоритмічна модель, деревовидна структура, handleMoving, collageTree, collageRow, collageColumn.*

R. I. KORDIUK
Postgraduate Student at the Department of Computer Technologies
in Publishing and Printing Processes
Lviv Polytechnic National University
ORCID: 0009-0008-6082-901X

OPERATIONAL MODEL OF A COLLAGE INFORMATION SYSTEM

An analysis of existing online technologies (Photoshop Online, Canva, Pixlr Editor) identified significant shortcomings in their models, primarily the absence of automatic linkage between image containers. This results in substantial time spent on manual adjustments and potential errors in the final collage compositions. These issues particularly disrupt compositional integrity when resizing one or several collage elements.

The paper also analyzes known algorithmic models such as AutoCollage (automatic selection and arrangement of important image areas), Correlation-Preserving Photo Collage (grouping images based on semantic relationships), Aesthetic Photo Collage with Deep Reinforcement Learning (creating aesthetic compositions through reinforcement learning), and SoftCollage (differentiable collage composition based on probabilistic trees).

The fundamental contribution of this research is the development and detailed description of algorithms implementing the handleMoving function, designed for the automatic adaptation of collage elements during interactive repositioning and resizing by users. Algorithms were developed for three primary structural components: CollageTree (hierarchical tree structure of the collage), CollageRow (horizontal arrangement of elements), and CollageColumn (vertical arrangement of elements). The logic behind these algorithms is thoroughly described, accompanied by flowcharts illustrating the algorithmic processes of movement, size recalculations, and automatic adaptation of elements based on user actions. Detailed analysis of the developed algorithms demonstrated high efficiency, even when scaling collages containing a significant number of images. The handleMoving function, developed in this study, can complement or extend the aforementioned algorithmic models, ensuring automatic maintenance of compositional integrity during interactive collage editing.

Key words: *collage information system, algorithmic model, tree-like structure, handleMoving, collageTree, collageRow, collageColumn.*

Вступ

Відомо багато програмних та онлайн-засобів, що спрощують роботу із зображеннями та створення композицій на кшталт фотоальбомів чи колажів. Однак низка завдань залишається невирішеною. Одне з них – відсутність узгоджених, структурованих зв'язків між елементами графічних композицій (колажів). У популярних онлайн-сервісах для створення колажів, таких як Canva, Pixlr Editor, Online Photoshop, зміна одного фрагмента колажу не призводить до автоматичної зміни інших. Кожен контейнер зображення діє незалежно, тому після редагування розмірів одного елементу користувачу доводиться вручну коригувати сусідні елементи, щоб усунути накладання чи незбалансованість композиції. Це значно ускладнює редагування та знижує ефективність роботи.

У галузі автоматичного створення колажів постійно з'являються нові підходи, що мають на меті підвищити якість та швидкість генерації композицій. Автоматичне компонування фото-колажів розглядається як оптимізаційна задача: визначається функція якості композиції (наприклад, сумарна видима важливість областей фото), яку алгоритм намагається максимізувати. Такий підхід потребує значних обчислювальних ресурсів, і зі збільшенням кількості зображень генерація оптимального колажу стає складною задачею. Для підвищення ефективності розроблені різні алгоритми: від методів на основі розбиття полотна до оптимізації функцій композиції. Сучасні дослідження приділяють увагу використанню штучного інтелекту. Застосування ефективних алгоритмів оптимізації дозволило прискорити створення колажів у десятки і навіть сотні разів. З'явилися й принципово нові підходи: зокрема, методи глибокого навчання та підсиленого навчання для генерації колажів, які навчаються розташовувати зображення з високою естетичною якістю. Так, запропоновано моделювати процес створення колажу як послідовність дій агента у середовищі з винагородою за естетичний результат (Deep RL). Інші сучасні роботи фокусуються на побудові диференційованих моделей колажу: наприклад, метод SoftCollage формулює генерацію колажу як диференційований процес побудови деревовидної структури, що оптимізується градієнтними методами. Таким чином, актуальними є дослідження, спрямовані на підвищення гнучкості та автоматизації створення колажів, аби ефективно обробляти зростаючі обсяги візуальних даних.

Проблема узгодженого та ефективного редагування вже створених графічних композицій (колажів) залишається відкритою. Існуючі системи не забезпечують динамічної підтримки цілісності композиції при внесенні локальних змін. Це стимулює розробку нових моделей, які б дозволяли автоматично адаптувати пов'язані елементи колажу при редагуванні, спрощували процес внесення змін та підвищували ефективність роботи користувачів.

Постановка проблеми

Існуючі системи онлайн-редакторів колажів не забезпечують автоматичного узгодження положення і розмірів усіх елементів при перестановках. Зокрема, відсутність зв'язаності між елементами колажу ускладнює їх трансформацію: після переміщення одного зображення користувач змушений вручну налаштовувати сусідні елементи, щоб зберегти цілісність композиції. Це призводить до збільшення часу редагування та ризику порушення пропорцій. Обмеженням відомих моделей є також їх низька масштабованість: коли колаж містить десятки зображень, перебудова компоновки стає вкрай неефективною. Таким чином, проблема полягає в розробці моделей алгоритмів, які автоматично обробляють переміщення елементів у колажі будь-якого розміру, мінімізуючи кількість необхідних операцій і гарантують збереження структури колажу.

Аналіз останніх досліджень і публікацій

Для розв'язання проблеми варто розглянути наукові дослідження в галузі автоматичного створення колажів та практичні реалізації в існуючих системах.

Останні роки характеризуються активним розвитком алгоритмічних методів автоматизованого створення колажів, обумовлених широким застосуванням цього виду візуального контенту в соціальних мережах, рекламі та інших медіа. Дослідження в цій галузі можна умовно поділити на кілька ключових напрямків, а саме на: методи автоматичного компонування, алгоритми, що враховують семантичну кореляцію між зображеннями, а також інтеграції алгоритмів глибокого навчання.

Однією з перших популярних систем автоматизованого компонування стала AutoCollage, розроблена дослідниками з Microsoft Research. Система AutoCollage використовує алгоритм графового розрізу для визначення регіонів інтересу (ROI), які потім безшовно об'єднуються у фінальний колаж завдяки згладжуванню меж. Попри свою ефективність, даний алгоритм має недоліки, такі як збереження неінформативного фону та відсутність повноцінного аналізу семантичних зв'язків між зображеннями [9, с. 847].

З метою подолання цих обмежень розроблена система Correlation-Preserving Photo Collage. Автори використали глибокі нейронні мережі для визначення кореляцій між зображеннями, забезпечуючи семантично осмислене компонування. Це значно покращило сприйняття колажів завдяки близькому розташуванню пов'язаних зображень та компактному використанню простору полотна [6, с. 1956].

У роботі автори розробили модель компонування колажів із застосуванням глибокого навчання з підкріпленням (Deep Reinforcement Learning). У цій моделі агент послідовно розміщує зображення, отримуючи підкріплення за естетичні характеристики колажу, такі як компактність, пропорції та відсутність перекриттів. Це дозволило

значно поліпшити естетичну якість та ефективність створення колажів, забезпечивши більш природне і оптимальне компонування [11].

Перспективною стала диференційована модель SoftCollage [10], яка використовує ймовірнісне деревоподібне представлення для оптимізації компонування за допомогою градієнтних методів. Вона дозволила не тільки уникнути спотворень пропорцій зображень, але й забезпечити високу швидкодію завдяки диференційованій структурі дерева. За результатами експериментів модель продемонструвала високу ефективність у створенні компактних та семантично узгоджених колажів.

У роботі автори представили модель Variational Transformer Network (VTN), яка поєднує варіаційні автокодери (VAE) та трансформери для генерації макетів. Модель використовує алгоритми самоуваги для виявлення високорівневих взаємозв'язків між елементами макету, що дозволяє навчатися правилам дизайну, таким як відступи та вирівнювання, без явного нагляду. Ця модель забезпечує високу схожість з навчальними даними та різноманітність згенерованих макетів, досягаючи передових результатів у різних типах макетів, включаючи документи та інтерфейси користувача [12].

У дослідженні автори описали розроблену систему Iris, орієнтовану на користувача, яка забезпечує інтерактивне середовище для генерації графічних макетів з урахуванням множинних обмежень. Дизайнер може вказати певні обмеження, такі як вирівнювання елементів або залишення певних областей порожніми, після чого генеративна модель системи пропонує відповідні макети. Ця система підкреслює важливість контролю та інтерактивності в практичних умовах, дозволяючи дизайнерам швидко отримувати якісні пропозиції, які відповідають їхнім вимогам [13, с. 968].

У статті досліджується застосування штучного інтелекту для дизайну макетів постерів у візуальній комунікації. Автори розробили модель, що поєднує навчання та генерацію: «учень» використовує мережу просторового трансформера для класифікації елементів макету та створення початкового шаблону, а «генератор» вдосконалює цей шаблон, застосовуючи принципи дизайну, такі як золоте січення. Ця модель демонструє, як поєднання класичних правил дизайну з машинним навчанням може покращити якість макетів постерів [14, с. 2–6].

AutoPoster – це система, яка автоматично генерує рекламні постери, використовуючи вхідні дані, такі як зображення продукту та його назва. Модель функціонування системи проходить через кілька етапів: попереднє опрацювання зображення, генерація макету, створення тексту та налаштування стилю. Враховування контексту забезпечує гармонійне поєднання макету та тексту з зображенням продукту. Ця модель демонструє практичне застосування автоматизованого дизайну в рекламній індустрії, дозволяючи швидко створювати візуально привабливі матеріали [21].

Flowy – система, яка використовує мультимодальний штучний інтелект для аналізу великих колекцій існуючих інтерфейсів користувача та автоматично анотує дизайн-патерни в них. Вона ідентифікує загальні патерни інтерфейсу (наприклад, навігаційні панелі, карткові макети, форми реєстрації) та надає дизайнерам огляд цих патернів, їхніх цілей, переваг і недоліків, а також приклади застосування. Це допомагає дизайнерам приймати обґрунтовані рішення під час розробки багатосторінкових інтерфейсів, забезпечуючи узгодженість та покращуючи користувацький досвід [16].

У статті досліджується проблема автоматичного розташування декількох фотографій на заданому полотні з високою естетичною якістю. Існуючі методи в основному базуються на оптимізації ручних характеристик, які не можуть адекватно відобразити високо рівневі людські естетичні відчуття. Автори пропонують нову модель для автоматичної генерації колажів із заданим співвідношенням сторін, вперше впроваджуючи техніку глибокого навчання з підкріпленням у цю сферу. Щоб навчити агента покращувати як загальний макет, так і локальні деталі, функція винагороди спеціально розроблена для колажу, враховуючи суб'єктивні та об'єктивні фактори. Щоб подолати нестачу навчальних даних, вони попередньо навчають свою глибоку естетичну мережу на великому наборі даних з естетики зображень (CPC) для загального вилучення естетичних характеристик та пропонують модуль злиття уваги для представлення структурних характеристик колажу. Результати показують, що їхня модель перевершує інші моделі в оцінці естетичної якості, що підтверджується як об'єктивними метриками, так і суб'єктивними оцінками користувачів [10, с. 3720–3725].

Чао Й, Ма Й, Джоу Й. Та інші автори представили роботу «Composition-Aware Graphic Layout GAN for Visual-Textual Presentation Designs» на конференції IJCAI. Вони дослідили проблему генерації графічних макетів для візуально-текстових презентацій, зокрема рекламних постерів. Автори відзначили, що композиція зображення, яка включає як глобальну семантику, так і просторову інформацію, суттєво впливає на результати макету. Для вирішення цієї проблеми вони запропонували глибоку генеративну модель під назвою Composition-Aware Graphic Layout GAN (CGL-GAN), яка синтезує макети на основі глобального та просторового вмісту вхідних зображень. Щоб подолати розбіжність між навчальними та тестовими даними, автори розробили новий модуль вирівнювання доменів (Domain Alignment Module, DAM), який звужує цей розрив. Для навчання вони створили великий набір даних, що складається з 60 548 рекламних постерів з анотованою інформацією про макет. Запропонована модель продемонструвала здатність синтезувати високоякісні графічні макети відповідно до композицій зображень [15].

Джоу Й., Джинг К., Лі З. Та інші автори представили роботу «Element-Conditioned GAN for Graphic Layout Generation» на платформі SSRN. Вони відзначили, що існуючі генеративно-змагальні мережі (GAN) для генерації макетів не враховують ситуації, коли кількість та типи вхідних елементів дизайну є заданими та визначеними. Для вирішення цієї проблеми вони запропонували елементно-умовну генеративно-змагальну мережу (EcGAN) для генерації графічних макетів, обумовлених заданими елементами дизайну (кількість та типи елементів). Вони представили кожен елемент у вигляді обмежувальної рамки та запропонували три компоненти: маску елементів, функцію втрат для умов елементів та двоетапні дискримінатори, щоб вирішити проблему моделювання обмежувальних рамок для генерації макетів з урахуванням елементів. Експерименти показали, що EcGAN перевершує існуючі моделі як кількісно, так і якісно. Автори також продемонстрували два застосування EcGAN для практичних сценаріїв дизайну. Ці роботи підкреслюють постійний прогрес у застосуванні генеративних моделей, зокрема GAN, для автоматизації процесу дизайну графічних макетів, враховуючи як візуальні, так і текстові елементи, а також специфічні умови та обмеження, що робить їх більш адаптивними та корисними для практичних застосувань у графічному дизайні [21].

Сучасні онлайн-сервіси для редагування зображень, такі як Photoshop Online, Canva та Pixlr Editor, надають базові засоби створення колажів.

Система Photoshop Online (Adobe Photoshop, n.d.) базується на моделі у вигляді фіксованої сітки: користувач задає кількість рядків і колонок на полотні, після чого утворюється рівномірна решітка для розміщення зображень. Кожен доданий на полотно елемент міститься в прямокутному контейнері, що займає одну або кілька комірок сітки (рис. 1, *a*). Колаж формується розташуванням зображень у цих контейнерах (рис. 1, *b*). Основним недоліком такої моделі є відсутність зв'язку між контейнерами для яких зміна розмірів одного не впливає на сусідні.

a b

Рис. 1. Полотно із сіткою та контейнерами *a*, Колаж зображення *b*

Система Canva (Canva, 2025) реалізує схожу модель шаблонів в якій користувач обирає макет, що складається з декількох контейнерів для фото. Елементи шаблону також незалежні, і зміна, наприклад, висоти одного контейнера не приводить до автоматичного підлаштування інших. На рис. 2 показано приклад застосування системи Canva. Після збільшення висоти нижнього контейнера він наклався на три верхніх (рис. 2, *b*), оскільки між ними відсутній зв'язок, користувач мусить вручну зменшити висоту верхніх контейнерів (рис. 2, *c*).

a b c

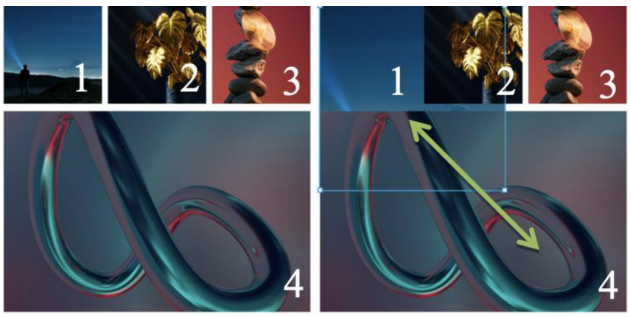
Рис. 2. Колаж в початковому вигляді *a*, накладання фрагментів зображень *b*, ілюстрація відсутності зв'язків між елементами колажа *c*

У системі Pixlr Editor (Online Photo Editor | Pixlr Editor, 2019) користувачу пропонуються готові моделі шаблонів різних компоновок або можливість самому додавати контейнери зображень. Цей редактор теж не забезпечує узгодженого масштабування: після зміни розміру одного контейнера інші залишаються фіксованими, що може спричинити накладання контейнерів. На рис. 3 показано приклад у Pixlr Editor: після збільшення висоти верхнього контейнера він наклався на верхній

та нижній контейнери (рис. 3, *b*), оскільки між ними відсутній зв'язок, користувач мусить вручну зменшити висоту контейнерів.

Загалом, усі три згадані системи (Canva, Pixlr, Photoshop Online) не мають алгоритмів автоматичного перерахунку розмірів та позицій сусідніх елементів при редагуванні (відповідні правки виконує користувач вручну) [3; 1]. Як наслідок, для внесення навіть відносно простої зміни композиції (наприклад, збільшення одного зображення) потрібно послідовно відредагувати декілька інших зображень, щоб відновити збалансований вигляд колажу. Це значно знижує продуктивність і зручність роботи з такими системами.

Наукові дослідження пропонують різні моделі автоматичного створення колажів, що можуть надати корисні ідеї для створення новітніх систем. Більшість моделей розглядають формування колажу як задачу оптимізації. Як зазначено вище, формується функція якості композиції, яка враховує певні критерії (мінімізація перекриття зображень, максимізація видимих важливих областей, естетичне розташування тощо). Далі моделями алгоритмів визначається таке розміщення зображень на полотні, яке оптимізує цю функцію. Проте пряма оптимізація для великої кількості елементів – NP-складна задача з величезним простором параметрів, тому практичні реалізації застосовують різні спрощення. Ранні роботи пропонували евристичні та генетичні моделі алгоритмів для підбору розміщення зображень у колажі. Інша група досліджень спрямована на покращення візуальної привабливості колажів – вводяться метрики естетики композиції і алгоритми оптимізують їх із залученням користувача. Наприклад, модель, описана у роботі авторів Джанг М., Лі М., та інших, використовує алгоритми глибокого навчання [11]. Система поступово розташовує зображення на канві, отримуючи винагороду за гармонійність композиції. Ця модель імітує процес ручного складання колажу, навчаючи агента вибирати наступний крок (обертання, масштаб, позицію чергового зображення) для покращення загальної якості. Отриманий таким чином колаж має збалансовану композицію з мінімумом порожнього простору.



a b

Рис. 3. Колаж в початковому вигляді *a*, ілюстрація зміни розмірів елемента колажа з № 1, та відсутності зв'язку між елементами з № 2, 3, 4 *b*

Окремо варто відзначити моделі з використанням деревовидних структур. Ідея полягає в рекурсивному розбитті області колажу на секції: наприклад, алгоритм SoftCollage конструює дерево, де кожен внутрішній вузол відповідає горизонтальному або вертикальному розбиттю області на підобласті, а листові вузли містять окремі зображення [10]. Таким чином формується ієрархічна структура «колаж-дерево». На відміну від моделей простих шаблонів, дерево дозволяє гнучкіше задавати макет. В роботі SoftCollage модель структури оптимізується нейронною мережею, яка навчається генерувати розбиття полотна, зберігаючи співвідношення сторін зображень і групуючи схожі за змістом зображення. Результати показали, що врахування структури колажу на рівні моделі дає змогу автоматично отримувати високоякісні композиції без сильних спотворень, з тематичним групуванням зображень.

Незважаючи на інтенсивний розвиток моделей на основі штучного інтелекту для автоматичного компонування графічних елементів [18; 20; 13], велика кількість практичних задач потребує детермінованих, контрольованих і передбачуваних алгоритмів редагування. Це особливо актуально у професійних і навчальних середовищах, де важлива точна ручна композиція з гарантованою підтримкою композиційної узгодженості.

У той час як сучасні генеративні моделі (наприклад, SoftCollage, EcGAN, AutoPoster) забезпечують високу якість автоматично згенерованих макетів, вони не надають засобів для інтерактивного редагування структури колажу після генерації, і найчастіше орієнтовані на одноразовий генеративний процес. Редагування вже створеної композиції, особливо з великою кількістю зображень, залишається поза їх можливостями.

Натомість деревовидна модель колажу, надає унікальні можливості автоматичного узгодження елементів композиції в процесі їхнього редагування користувачем. Завдяки гнучкій ієрархічній структурі (*CollageTree* → *CollageRow* → *CollageColumn* → *ImageContainer*) та алгоритмам типу *handleMoving*, будь-яка локальна зміна автоматично відображається на всіх пов'язаних елементах, зберігаючи композиційну цілісність і скорочуючи кількість необхідних дій у кілька разів.

Ця модель дозволяє подолати обмеження ручного редагування, яке наявне в таких інформаційних системах як Canva, Pixlr, Photoshop Online, де відсутній зв'язок між контейнерами, що вимагає багатоетапної ручної корекції кожного сусіднього елемента [1; 3; 8].

Формулювання цілі статті. Метою роботи є розробка та аналіз деревовидної моделі опрацювання колажів зображень. Зокрема, ставиться завдання створити алгоритм для базових компонентів моделі (*handleMoving*) і алгоритми вузла колажу (дерева), рядка колажу та стовпця колажу, які забезпечать автоматичне перевпорядкування елементів при їх переміщенні користувачем. Ціль полягає в тому, щоб мінімізувати кількість операцій при кожній перестановці і зберегти пропорції зображень та відступи між ними без ручного втручання. Досягнення цієї мети включає побудову блок-схем для візуалізації логіки алгоритмів та перевірку ефективності на граничних випадках. Очікується, що реалізація таких алгоритмів підвищить інтерактивність і продуктивність інформаційної системи опрацювання колажів, забезпечивши адаптивне реагування на дії користувача.

Виклад основного матеріалу

Архітектура деревовидної моделі колажу. Розроблена модель колажу організована у вигляді дерева (ієрархічної структури), що містить чотири рівні. Перший рівень – це кореневий елемент, який відповідає всьому колажу (полотну), як цілісному графічному об'єкту. Кореневий елемент має фіксовані загальні розміри колажу (наприклад, ширину і висоту), колір фону та інші глобальні параметри. На цьому рівні можна задавати відступи від країв полотна, а також, за потреби, рівномірне заокруглення кутів (коефіцієнт заокругленості) – початково він нульовий, але може динамічно змінюватися, якщо інтерфейс дозволяє закруглювати краї колажу. Другий рівень – це рядки колажу. Кореневий елемент містить рядків, розташованих вертикально один під одним, що формують вертикальну структуру колажу. Між кожною парою сусідніх рядків розташований горизонтальний повзунок рядка – інтерактивний роздільник, пересування якого змінює пропорції висот двох суміжних рядків. Таким чином, якщо – кількість рядків, на другому рівні буде рядків і горизонтальних повзунків. Третій рівень – вміст рядків. Кожен рядок може містити або контейнери зображень, розташованих горизонтально, або один чи більше під-рівнів у вигляді колонок. Якщо рядок містить безпосередньо контейнери зображень, то між ними розташовуються вертикальні повзунки секцій, що дозволяють змінювати їхні ширини. Нехай в рядку розміщено елементів (контейнерів зображень чи колонок) – тоді між ними буде вертикальних повзунків. Четвертий рівень – це випадок, коли на третьому рівні є колонки. Колонка являє собою додатковий рівень вкладеності всередині рядка. Вона містить контейнери зображень, розташовані вертикально, між якими вже присутні горизонтальні повзунки (аналогічні до повзунків між рядками). Тобто колонка фактично схожа на «рядок всередині рядка». Якщо в колонці зображень, між ними буде горизонтальних повзунків. Таким чином, колаж може містити і змішані структури: частина рядків складається безпосередньо з зображень, а частина – з колонок, які вміщують зображення. Загальний четвертий рівень – це контейнери зображень, які є листовими вузлами дерева (вони не містять інших елементів, а лише графічний вміст – власне зображення).

На рис. 4 показано приклад трирівневого колажу. кореневий елемент містить два рядки (перший і другий рівень), причому перший рядок має три контейнери зображень (№ 1, № 2, № 3) на третьому рівні, а другий рядок – один контейнер (№ 4). Між контейнерами 1, 2, 3 знаходяться вертикальні повзунки (розділювачі), а між першим і другим рядком – горизонтальний повзунок.

Рис. 4. Трьохрівневий колаж

Введені взаємозв'язки між усіма цими елементами утворюють складну і високо функціональну структуру. Кожна складова колажу «знає» про своїх сусідів і «батьківські» елементи та може впливати на них. Кожен елемент дерева в цій моделі є самостійним об'єктом, проте вони не ізольовані – між ними встановлені відносини «батько-нащадок» і «сусід-сусід». Завдяки цьому усі частини колажу утворюють єдину динамічну систему. Основний принцип функціонування моделі: внесені зміни в одному елементі автоматично та пропорційно відображаються на всіх інших пов'язаних з ним елементах. Це означає, що якщо користувач змінює розміри чи положення якогось контейнера, модель переобчислює розміри сусідніх контейнерів, а за потреби і батьківських елементів (рядків або колонок), щоб зберегти узгодженість структури. Така деревовидна структура уможливлює відповідну

поведінку через рекурсивні зв'язки: кожен вузол «знає» про сусідні вузли на тому ж рівні та про батьківський вузол і може повідомити їм про зміну. Описана модель є суттю алгоритмів пропорційного перерозподілу (функцій *handleMoving*). У межах моделі реалізовано спеціальні функції, що відповідають за опрацювання дій користувача при перетягуванні повзунків – умовно названі *handleMoving*.

Ключовим елементом функціонування описаної системи є алгоритм динамічного перерахунку розмірів при взаємодії користувача з роздільниками. В інтерактивному режимі, коли користувач пересуває повзунок, система повинна відповідно змінити розміри сусідніх областей колажу і, за необхідності, інших пов'язаних елементів. Цей процес реалізовано через функцію *handleMoving*, яка визначена для кожного типу контейнера (дерево, рядок, колонка) з урахуванням специфіки їх вкладеності. Нижче розглянуто алгоритми роботи *handleMoving* для різних рівнів структури.

Алгоритм handleMoving для CollageTree (кореневий рівень). Функція *CollageTree.handleMoving*() викликається при переміщенні горизонтального роздільника між рядками колажу. Нехай користувач пересунув даний роздільник на відстань Δy (вертикальне зміщення). Ця величина обчислюється як різниця між новою та старою ординатою повзунка:

$$\Delta y = y - y', \tag{1}$$

де y' – початкова ордината повзунка, y – теперішня ордината повзунка.

Перед початком перерахунку зберігаються початкові координати роздільника (*oldY*) та визначаються сусідні елементи: *previousEl* – рядок, що знаходиться безпосередньо над повзунком, і *nextEl* – рядок під повзунком. Алгоритм виконує такі кроки:

1. Перевірка граничних випадків. Якщо повзунок знаходиться над першим рядком (тобто немає попереднього елемента) або під останнім рядком (немає наступного елемента), то зміщення може бути застосоване лише в одному напрямку. У такому разі алгоритм виконує оновлення групи наявних рядків зверху або знизу, відповідно, розподіляючи зміщення між ними. Фактично, якщо повзунок у верхній межі, всі рядки нижче повзунка зсуваються вгору/вниз на Δy; якщо у нижній – всі рядки вище повзунка масштабуються.

2. Перевірка обмежень розміру. Перед застосуванням зміщення перевіряється, чи не стане висота будь-якого із сусідніх рядків або їхніх вкладених елементів надто малою (меншою за деякий мінімум, наприклад 50 px). Для цього аналізуються дочірні елементи *previousEl* та *nextEl*: якщо, скажімо, у попереднього рядка є вкладена колонка, всі її внутрішні зображення перевіряються на мінімальну висоту після потенційного зменшення. Якщо хоча б один елемент порушує обмеження (наприклад, всі зображення в колонці стали б менше 50 px заввишки), подальший рух повзунка блокується – функція *handleMoving* достроково завершується без змін, встановлюючи повзунок назад на попередню позицію. Таким чином гарантується збереження мінімально допустимих розмірів всіх частин колажу.

3. Оновлення розмірів сусідніх рядків. Якщо обмеження не порушені, виконується пропорційна зміна висоти двох сусідніх рядків, між якими рухається повзунок. Верхній рядок (*previousEl*) збільшується або зменшується на Δy, нижній (*nextEl*) – відповідно зменшується або збільшується на Δy (тобто один росте на ту величину, на яку інший втрачає). Формально, якщо H_{prev} та H_{next} – початкові висоти цих рядків, то після руху повзунка:

$$H'_{prev} = H_{prev} + \Delta y, \quad H'_{next} = H_{next} - \Delta y. \tag{2}$$

Позиції цих рядків по вертикалі також зсуваються: верхній зміщується на вниз (щоб залишатися центровано вирівняним у нових межах), а нижній – на $\Delta/2$ вгору.

4. Оновлення вкладених елементів. Рядки можуть містити вкладені колонки. При зміні висоти рядка необхідно скоригувати висоти внутрішніх контейнерів у колонках, щоб зберегти пропорції. Алгоритм проходить по кожному дочірньому елементу рядків *previousEl* та *nextEl*. Якщо якийсь елемент є колонкою (*CollageColumn*), то її вміст (набір зображень) перерозподіляється: здійснюється пропорційне збільшення або зменшення висот всіх вкладених зображень у колонці на величину Δy, поділену порівну між ними. Спеціально опрацьовуються випадки, коли висота окремих зображень у колонці досягла мінімуму – тоді така колонка більше не масштабується (це перевірено на кроці 2). У реалізації це відповідає виклику допоміжних методів *handleColumnBlockMove* та *handleColumnReverseBlockMove*, які розподіляють між дочірніми елементами колонки, що належать до верхнього та нижнього рядків, відповідно.

5. Відображення змін. Після оновлення числових параметрів висоти та позицій здійснюється рендеринг змін на полотні – викликається перерахунок координат всіх елементів колажу та перемальовування канви. Поточне значення позиції повзунка зберігається як його *oldY* для подальших взаємодій.

Алгоритм *handleMoving* на рівні *CollageTree* забезпечує узгоджене вертикальне масштабування компонувань колажу. Важливо, що зміна відбувається локально (між двома рядками), але завдяки рекурсивним зв'язкам ця зміна може вплинути і на глибші рівні (вкладені колонки та зображення), які були скориговані на кроці 4. Таким чином досягається *глобальна узгодженість*: колаж як єдина структура реагує на дію користувача без порушення цілісності макету.

Створений алгоритм *handleMoving* реалізований рекурсивно. Коли повзунок рухається, то система виконує такі дії:

- Визначає, які дві секції (або рядки) безпосередньо задіяні.
- Змінює їхні розміри відповідно до переміщення користувача, не виходячи за межі мінімумів/максимумів.
- Перевіряє граничні умови – якщо сусідня секція досягла мінімуму, рекурсивно викликає аналогічну операцію для наступної межі (повзунка) в напрямку руху.
- Оновлює розміри усіх пов'язаних елементів (наприклад, перераховує процентні співвідношення для дочірніх контейнерів у зміненому рядку або колонці).

Таким чином досягається автоматична адаптація композиції під час редагування.

Алгоритм *handleMoving* для *CollageRow* (горизонтальний рівень). Функція *CollageRow.handleMoving* відповідає за опрацювання переміщення вертикального повзунка всередині рядка, тобто зміну пропорцій сусідніх елементів по ширині. Припустимо, що в рядку є дві сусідні області – ліворуч (*previousEl*) і праворуч (*nextEl*) від повзунка. Нехай повзунок переміщено на величину Δx горизонтально (вправо – додатнє Δx, вліво – від'ємне). Значення Δx обчислюється аналогічно:

$$\Delta x = x - x', \tag{3}$$

де x' – початкова абсциса повзунка, x – теперішня абсциса повзунка. Алгоритм дій наступний:

1. Обмеження на краї рядка. Якщо повзунок пересувається з крайнього лівого положення (немає попереднього елемента) або крайнього правого (немає наступного елемента), то зміни ширини застосовуються до групи елементів з одного боку. Це крайні випадки, подібні до випадку з рядками: якщо повзунок біля лівого краю, всі елементи праворуч стискаються або розширюються на Δx; якщо біля правого – всі елементи ліворуч масштабуються на Δx. У реалізації використовується допоміжна функція *updateGroup* (Ψ) з відповідними параметрами.

2. Перевірка мінімальної ширини. Перед зміною розмірів перевіряється, чи залишиться достатня ширина у сусідніх елементів. Кожен контейнер (або колонка) має мінімально допустиму ширину (аналогічно ~50 px). Якщо пересування повзунка занадто сильно звужує один з елементів, рух блокується. У коді перевірка відбувається неявно: колонки не дозволять зменшити свої вкладені елементи менш ніж на мінімум (подібно до перевірки висоти в *CollageTree*), а для базових зображень мінімум задається явно.

3. Зміна розмірів сусідніх областей. Якщо обидва сусідні елементи існують і рух допустимий, їх ширини оновлюються пропорційно. Ліва область (*previousEl*) розширюється на Δx, права (*nextEl*) звужується на ту ж величину (при перетягуванні вправо) або навпаки при русі вліво. Якщо якийсь із сусідніх елементів сам є колонкою (вертикальним контейнером), то потрібно коригувати ширину всіх вкладених у нього зображень. У реалізації: для *CollageColumn* ліва колонка отримує збільшення ширини на Δx і всі її внутрішні зображення теж розширюються на (зсування вліво/вправо на $\Delta x/2$ для центрованого вирівнювання). Для правої колонки – відповідно звуження на Δx для неї і її вмісту. Якщо елементи не є колонками (тобто безпосередньо зображення), то їхні ширини змінюються прямо, без додаткового розподілу. Після цих операцій рядок оновлює свої координати.

4. Синхронізація та відображення. *CollageRow* після зміни викликає метод оновлення своїх дочірніх елементів, щоб перерахувати їхні нові положення. Потім оновлюються координати самого рядка (з врахуванням нових габаритів). Зміни негайно відображаються на екрані шляхом перемальовування полотна (виклик *requestRenderAll*). Позиція повзунка зберігається як $x' = x$ для подальших дій.

Алгоритм *handleMoving* для рядка забезпечує узгоджене масштабування елементів по горизонталі. Таким чином, при зміні ширини одного сегмента рядка інший сегмент автоматично компенсує цю зміну, зберігаючи загальну ширину рядка постійною. Якщо сегментами є колонки, то їх внутрішня структура теж перебудовується аналогічно до вертикального випадку: зміна ширини колонки викликає зміну ширини всіх вкладених зображень пропорційно, щоб не виникло спотворень.

Алгоритм *handleMoving* для *CollageColumn* (вертикальний рівень). Функція *CollageColumn.handleMoving* обробляє переміщення горизонтального повзунка всередині колонки, тобто зміну пропорцій суміжних зображень по висоті. Цей випадок дуже подібний до алгоритму для *CollageTree*, але обмежений межами однієї колонки. Нехай у колонці є два сусідні зображення (або вкладені елементи) над і під повзунком. При зміщенні повзунка на Δy вниз або вгору перевіряється мінімальна висота цих зображень, щоб жодне з них не стало меншим за встановлений мінімум. Далі виконується оновлення висот: верхній елемент отримує додаткову висоту Δy, нижній – втрачає Δy (або навпаки, якщо повзунок рухається в інший бік). Їхні вертикальні позиції зсуваються на $\Delta y/2$ для збереження контактного розташування. Колонка після цього викликає метод оновлення всіх вкладених зображень *updateImageCell*, щоб, наприклад, кожен контейнер зображення всередині знав про свою нову висоту і міг змінити масштаб зображення, якщо потрібно. Після перерахунку параметрів виконується перемальовування полотна (*Canvas*) для відображення результату, а поточна позиція повзунка фіксується як старе значення (*oldY*).

У випадку, коли повзунок у колонці доходить до верхньої або нижньої межі (тобто один з сусідніх елементів відсутній), алгоритм аналогічно оновлює групу елементів зі сторони, де присутні кілька зображень. Наприклад,

якщо повзунок під найвищим зображенням колонки, то при його русі вниз всі нижченаведені зображення колонки зсуваються пропорційно, звільняючи місце. Загалом, *handleMoving* для колонки повторює логіку *handleMoving* для всього колажу, але на локальному рівні: він не впливає на інші частини колажу за межами даної колонки.

Зазначені алгоритми гарантують цілісність колажу під час інтерактивного редагування. Завдяки ієрархічним зв'язкам між елементами, зміна на будь-якому рівні автоматично узгоджується з пов'язаними елементами на інших рівнях. Функція *handleMoving* виступає універсальним алгоритмом для адаптивного масштабування: її реалізація в компонентах ***CollageTree***, ***CollageRow*** та ***CollageColumn*** дозволяє ефективно обробляти зміни розмірів на різних рівнях, забезпечуючи узгодженість між ними та гнучкість інтерфейсу користувача.

Опишемо блок-схему алгоритму *handleMoving* для *CollageTree* (рис. 5–6). На першому кроці визначаємо параметри алгоритму, де y_s – теперішня ордината координат повзунка, y'_s – попередня ордината координат

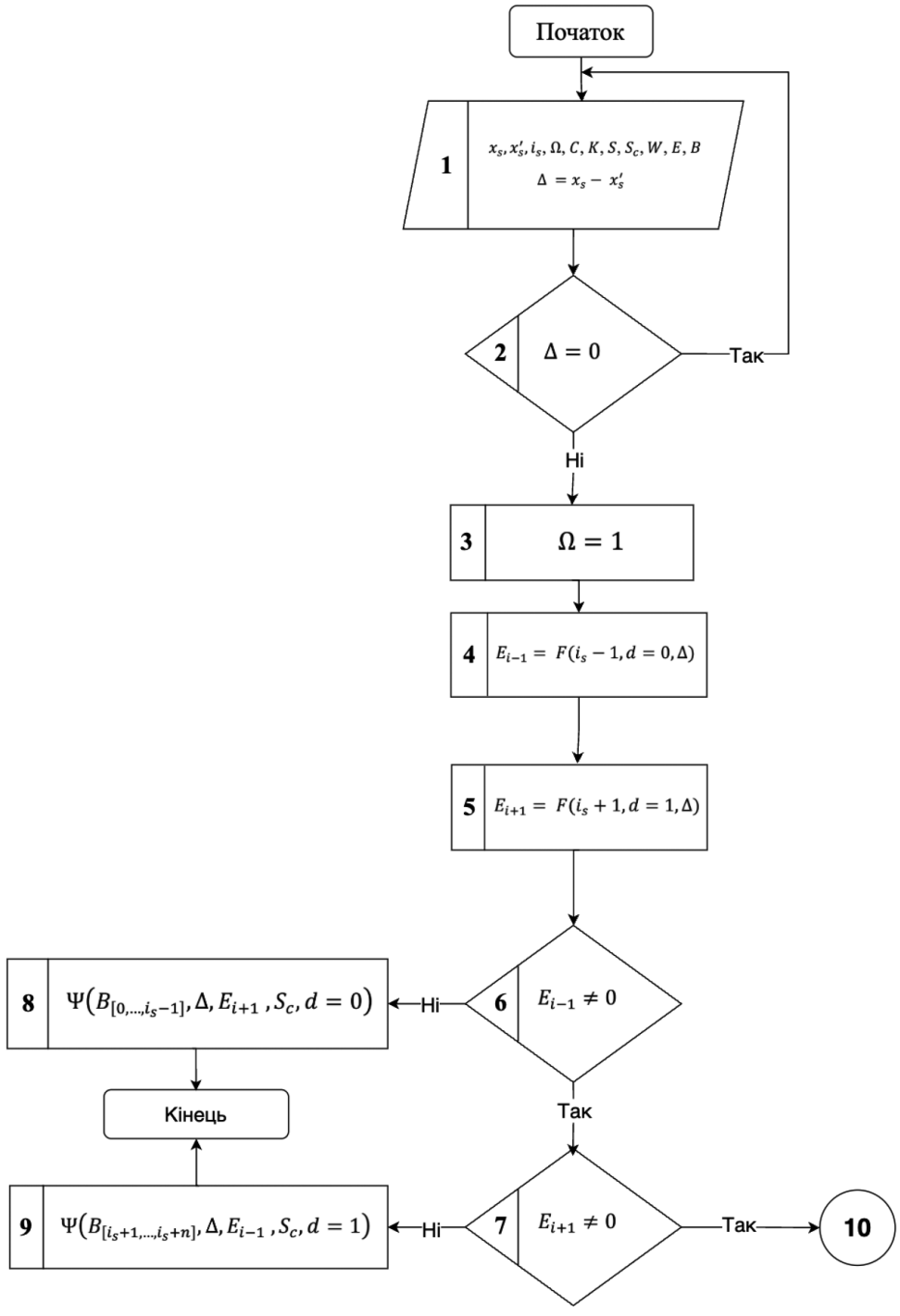


Рис. 5. Блок-схема алгоритму handleMoving для CollageTree

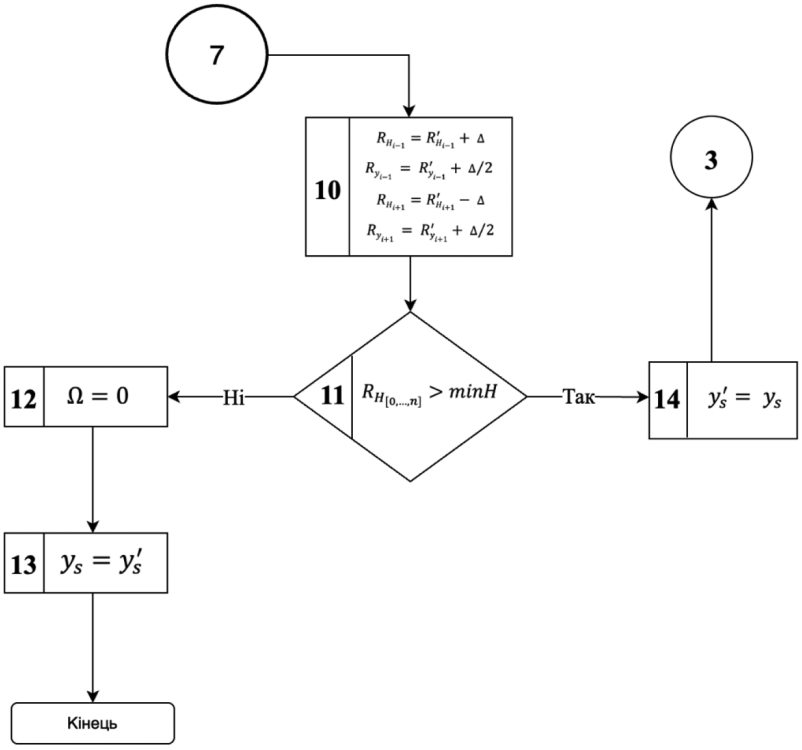


Рис. 6. Блок-схема алгоритму handleMoving для CollageTree (продовження Рис. 5)

повзунка, i_s – індекс рухомого повзунка, Ω – позначення дозволу руху далі (1,0), *C* – *CollageColumn* (колонка), *K* – *ImageContainer* (контейнер зображення), *S* – *CollageSeparator* (повзунок), S_c – рухомий повзунок, *d* – напрям руху *S* (1,0), *r* – елемент рядка, r_s – повзунок рядка, r_C– колонка рядка, r_K – контейнер зображення рядка, $R \in r_s$ чи r_C чи r_K, *H* – висота R_H (висота рядка), $E = R \cup S$. На другому кроці перевіряємо чи змінив рухомий повзунок своє розташування, якщо ні, то повертаємось на початок, якщо так то переходимо на третій крок. На третьому кроці присвоюємо Ω = 1, і дозволяємо рух далі. На четвертому кроці шукаємо *R* перед S_c, який має висоту більшу за min*H*, а на п'ятому кроці шукаємо *R* після S_c, який має висоту більшу за min*H*. На шостому кроці перевіряємо, чи знайшли на четвертому кроці рядок *R* перед S_c з *H* більшою за min*H*. Якщо ні, то переходимо на восьмий крок і виконуємо функцію Ψ. ,ка оновлює ординату координат усіх переданих *E*, для яких ця трансформація можлива, а також збільшує висоту і оновлює ординату координат усіх переданих елементів рядка *R*, який розташований після S_c, та закінчуємо виконання алгоритму *handleMoving*. Якщо на четвертому кроці знайшли *R* перед S_c з *H* більшою за min*H*, то переходимо на сьомий крок і перевіряємо, чи знайшли рядок *R* після S_c з *H* більшою за min*H*, якщо ні, то переходимо на дев'ятий крок і виконуємо функцію Ψ, яка оновлює ординату координат усіх переданих *E*, для яких ця трансформація можлива, а також збільшує висоту і оновлює ординату координат усіх переданих елементів рядка *R*, який розташований перед S_c, та закінчуємо виконання алгоритму *handleMoving*. Якщо на п'ятому кроці знайшли *R* після S_c з *H* більшою за min*H*, то переходимо на десятий крок, на якому оновлюємо R_H та R_y, *R* до та після S_c. Далі переходимо на одинадцятий крок на якому перевіряємо чи містить *CollageTree* ще *R*, у якого $R_H >$ min*H*. Якщо не містить, то переходимо на крок дванадцятий, на якому присвоюємо Ω = 0, і забороняємо рух S_c далі. На тринадцятому кроці присвоюємо y_s значення попередньої ординати координат y'_s, щоб позиція S_c залишалась незмінною і закінчуємо виконання функції. Якщо на одинадцятому кроці ми знаходимо *R*, у якого $R_H >$ min*H*, то переходимо на чотирнадцятий крок. На якому присвоюємо y'_s значення теперішньої ординати координат y_s, і переходимо на третій крок для продовження роботи алгоритму.

Далі опишемо блок-схему алгоритму *handleMoving* для *CollageRow* (рис. 7–8). На другому кроці визначаємо параметри алгоритму, де x_s – теперішня абсциса координат повзунка, x'_s – попередня абсциса координат повзунка, i_s – індекс рухомого повзунка, Ω – позначення дозволу руху далі (1,0), *C* – *CollageColumn* (колонка), *K* – *ImageContainer* (контейнер зображення), *S* – *CollageSeparator* (повзунок), S_c – рухомий повзунок, *d* – напрям руху *S* (1,0), $E = C \cup K$, $B = C \cup K \cup S$, *W* – ширина E_w. На другому кроці перевіряємо чи змінив рухомий повзунок своє розташування, якщо ні, то повертаємось на початок, якщо так то переходимо на третій крок. На третьому кроці присвоюємо Ω = 1, і дозволяємо рух далі. На четвертому кроці шукаємо *E* перед S_c, який має ширину більшу за min*W*. На п'ятому кроці шукаємо *E* після S_c, який має ширину більшу за min*W*. На шостому кроці перевіряємо, чи знайшли на четвертому кроці *E* перед S_c з *W* більшою за min*W*. Якщо ні, то переходимо на восьмий крок

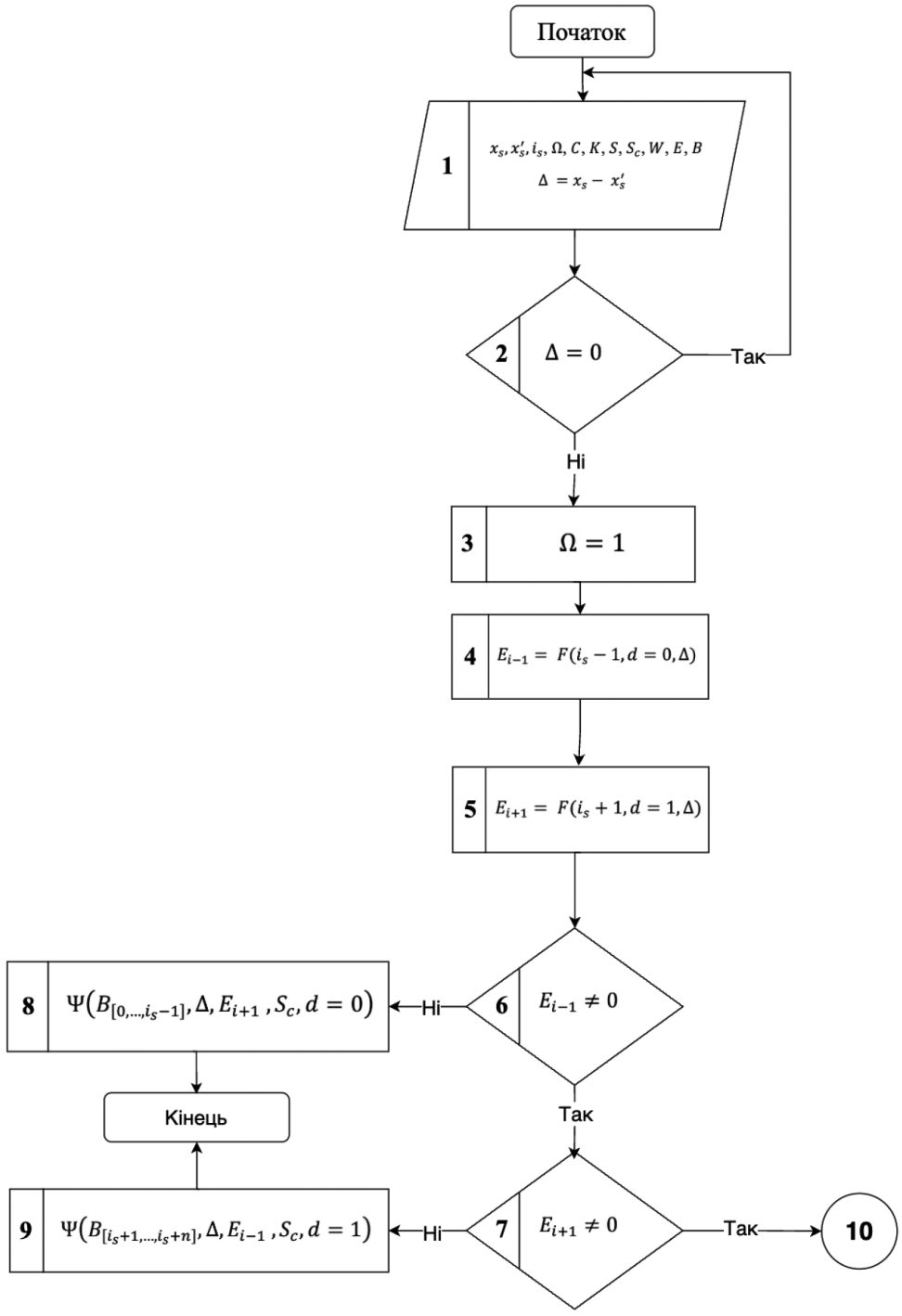


Рис. 7. Блок-схема алгоритму handleMoving для CollageRow

і виконуємо функцію Ψ. ,ка оновлює абсцису координат усіх переданих *B*, для яких ця трансформація можлива, а також збільшує ширину і оновлює абсцису координат усіх переданих елементів *E*, які розташовані після S_c. Якщо на четвертому кроці знайшли *E* перед S_c з *W* більшою за min *W*, то переходимо на сьомий крок. Перевіряємо, чи знайшли *E* після S_c з *W* більшою за min *W*. Якщо ні, то переходимо на дев'ятий крок і виконуємо функцію Ψ, яка оновлює ординати координат усіх переданих *B*, для яких ця трансформація можлива, а також збільшує ширину і оновлює абсцису координат усіх переданих елементів *E*, які розташовані перед S_c. Якщо на п'ятому кроці знайшли *E* після S_c з *W* більшою за min *W*, то переходимо на десятий крок. На ньому оновлюємо E_w та E_y, *E* до та після S_c. Далі переходимо на одинадцятий крок на якому перевіряємо чи містить *CollageRow* ще *E*, у якого $E_w >$ min *W*. Якщо не містить, то переходимо на крок дванадцятий, на якому присвоюємо Ω = 0 і забороняємо рух S_c далі. На тринадцятому кроці присвоюємо x_s значення попередньої абсциси координат x'_s, щоб позиція S_c

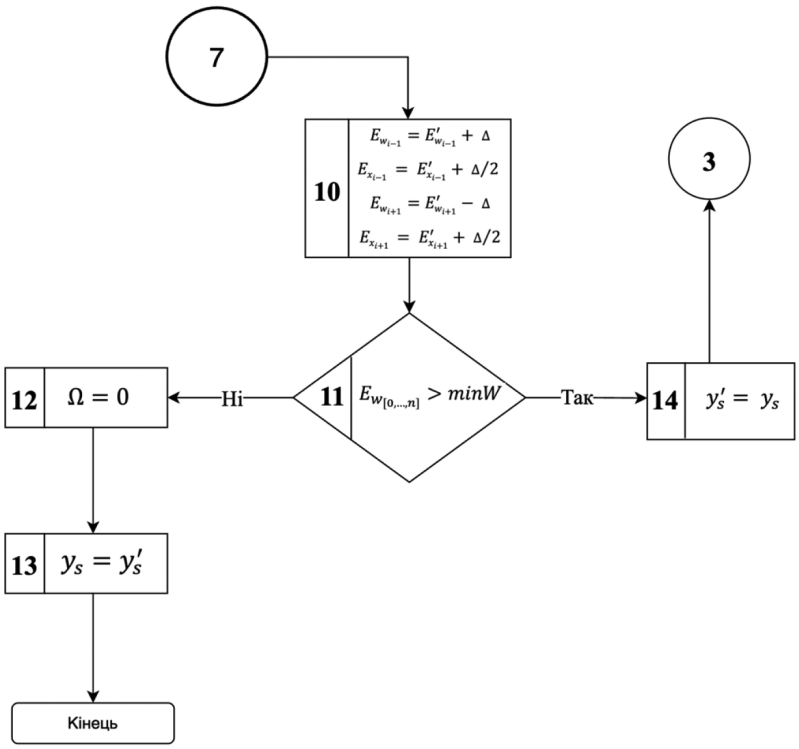


Рис. 8. Блок-схема алгоритму handleMoving для CollageRow (продовження Рис. 7)

залишалась незмінною і закінчуємо виконання функції. Якщо на одинадцятому кроці знаходимо E, у якого $E_w > \min W$, то переходимо на чотирнадцятий крок. На ньому присвоюємо x'_s значення теперішньої абсциси координат x_s, і переходимо на третій крок для продовження роботи алгоритму.

Опишемо блок-схему алгоритму *handleMoving* для *CollageColumn* (рис. 9–10). На першому кроці визначаємо параметри алгоритму, де y_s – теперішня ордината координат повзунка, y'_s – попередня ордината координат повзунка, i_s – індекс рухомого повзунка, Ω – позначення дозволу руху далі (1,0), K – *ImageContainer* (контейнер зображення), S – *CollageSeparator* (повзунок), S_c – рухомий повзунок, d – напрям руху S (1,0), H – висота K_H (висота контейнера зображень), $E = R \cup S$. На другому кроці перевіряємо чи змінив рухомий повзунок своє розташування. Якщо ні, то повертаємось на початок. Якщо так, то переходимо на третій крок. На третьому кроці присвоюємо $\Omega = 1$ і дозволяємо рух далі. На четвертому кроці шукаємо K перед S_c, який має висоту більшу за $\min H$, а на п'ятому кроці шукаємо K після S_c, який має висоту більшу за $\min H$. На шостому кроці перевіряємо, чи знайшли на четвертому кроці рядок R перед S_c з H більшою за $\min H$. Якщо ні, то переходимо на восьмий крок і виконуємо функцію Ψ. ,ка оновлює ординату координат усіх переданих E, для яких ця трансформація можлива, а також збільшує висоту і оновлює ординату координат K, який розташований після S_c, та закінчуємо виконання алгоритму *handleMoving*. Якщо на четвертому кроці знайшли K перед S_c з H більшою за $\min H$, то переходимо на сьомий крок і перевіряємо, чи знайшли K після S_c з H більшою за $\min H$. Якщо ні, то переходимо на дев'ятий крок і виконуємо функцію Ψ, яка оновлює ординату координат усіх переданих E, для яких ця трансформація можлива, а також збільшує висоту і оновлює ординату координат K, який розташований перед S_c, та закінчуємо виконання алгоритму *handleMoving*. Якщо на п'ятому кроці знайшли R після S_c з H більшою за $\min H$, то переходимо на десятий крок. На ньому оновлюємо K_H та K_y, K до та після S_c. Далі переходимо на одинадцятий крок на якому перевіряємо чи містить *CollageColumn* ще K, у якого $K_H > \min H$. Якщо не містить, то переходимо на крок дванадцятий, на якому присвоюємо $\Omega = 0$ і забороняємо рух S_c далі. На тринадцятому кроці присвоюємо y_s значення попередньої ординати координат y'_s, щоб позиція S_c залишалась незмінною і закінчуємо виконання алгоритму. Якщо на одинадцятому кроці знаходимо R, у якого $R_H > \min H$, то переходимо на чотирнадцятий крок, на якому присвоюємо y'_s значення теперішньої ординати координат y_s, і переходимо на третій крок для продовження роботи алгоритму.

Приклад структурованого колажу. Розглянемо більш складний випадок – чотирирівневий колаж, що містить як рядки, так і колонки. На рис. 11 зображено таку структуру: колаж складається з двох рядків на другому рівні. Перший рядок містить чотири контейнери зображень (№ 1, № 2, № 3, № 4) на третьому рівні; другий рядок містить два елементи – колонку з трьома зображеннями (№ 5, № 6, № 7) та окремий контейнер (№ 8). У першому рядку між контейнерами 1–4 розташовані три вертикальні повзунки; у другому рядку між колонкою і контейнером № 8 – один вертикальний повзунок; всередині колонки між зображеннями 5–7 – два горизонтальні

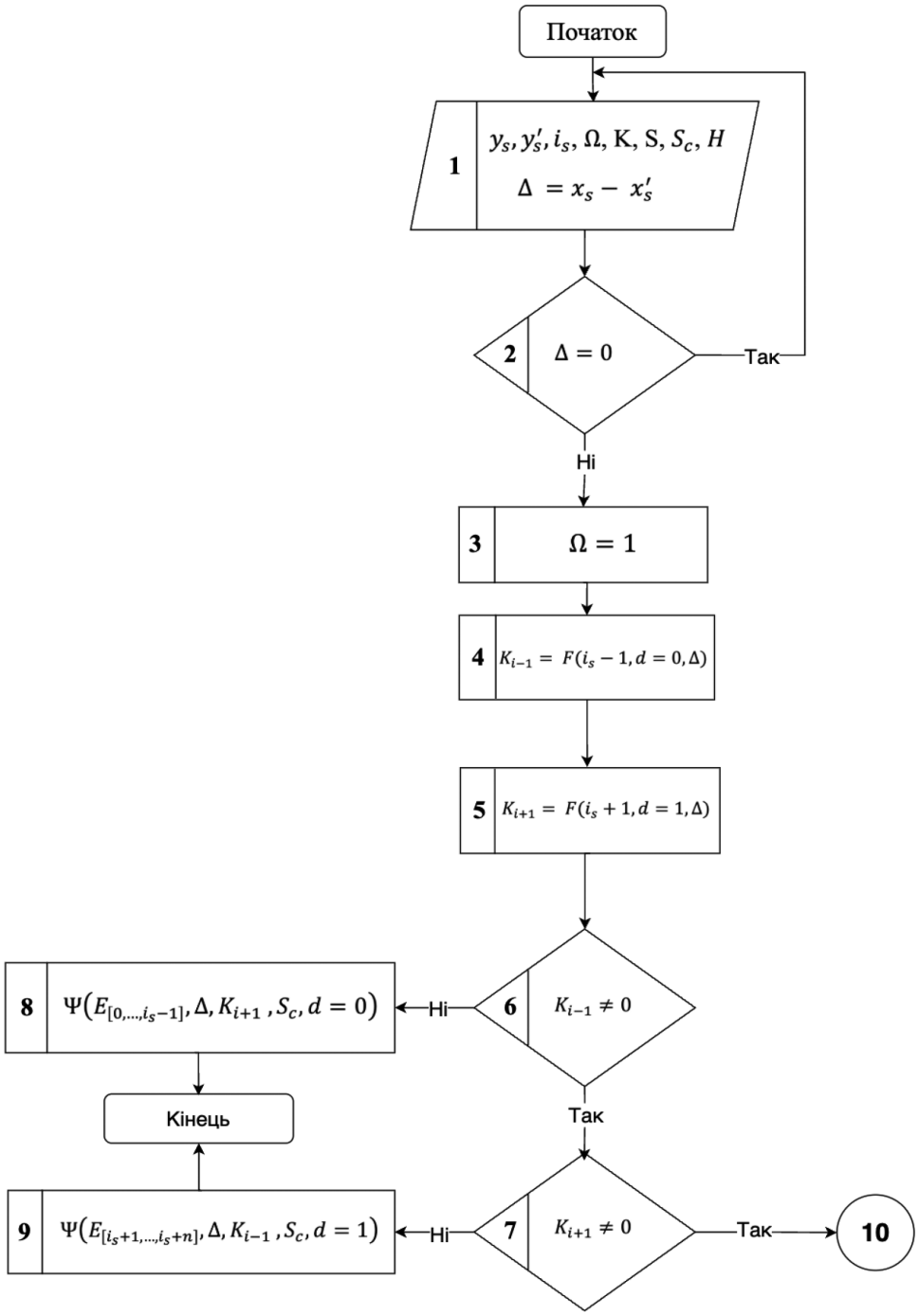


Рис. 9. Блок-схема алгоритму handleMoving для CollageColumn

повзунки; між самим першим і другим рядком – горизонтальний повзунок полотна. Таким чином, дерево колажу має: корінь → 2 рядки; перший рядок → 4 контейнери; другий рядок → 1 колонка + 1 контейнер; колонка (в другому рядку) → 3 контейнери. Всього 8 зображень. Колонка в складі другого рядка дозволяє згрупувати зображення № 5–7 вертикально. У такому колажі також діють всі описані алгоритми. Якщо, наприклад, пересунути вертикальний повзунок між контейнерами № 3 та № 4 у першому рядку вліво, контейнер № 4 почне збільшуватися, а № 3 – зменшуватися. Коли № 3 досягне мінімальної ширини, подальший рух спричинить зменшення контейнера № 2 (другого зліва), рис. 11, *b*. В результаті кілька елементів змінять розміри одночасно, перерозподіливши вивільнений простір. Аналогічно, при збільшенні висоти контейнера № 5 (верхнього в колонці) за рахунок зменшення № 6 і № 7, якщо останні досягнуть мінімуму – рух повзунка зупиниться рис. 11, *c*. Таким чином, деревовидна модель забезпечує узгодженість на всіх рівнях: навіть складні багаторівневі структури автоматично підтримують логічні пропорції між всіма частинами композиції.

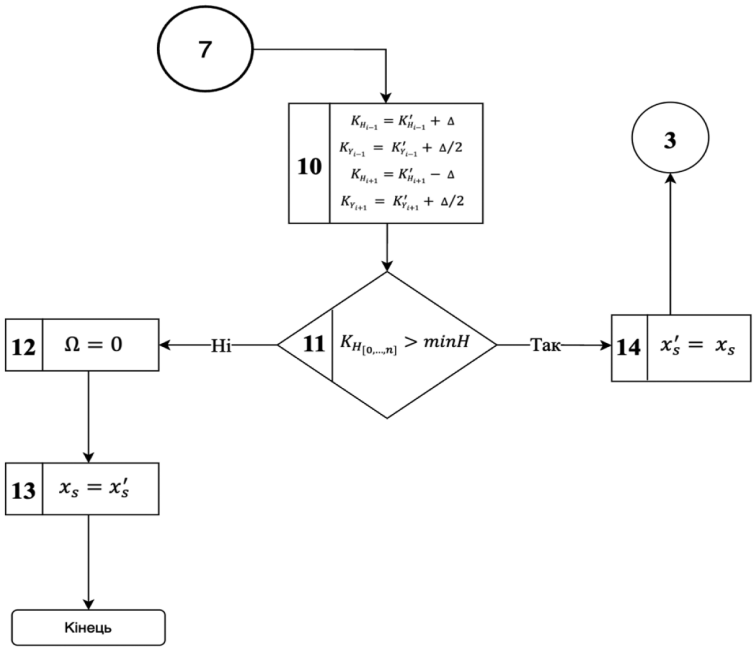


Рис. 10. Блок-схема алгоритму handleMoving для CollageColumn (продовження рис. 9)

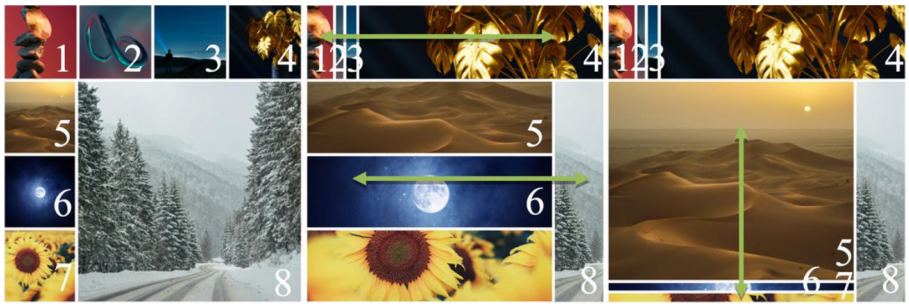


Рис. 11. Ілюстрація чотирьохрівневого колажу в початковому вигляді *a*, руху повзунка між елементами рядка *b* та горизонтального повзунка між елементами колонки *c*

Технологічна реалізація. Для впровадження розробленої моделі було обрано веб-фреймворк Angular [2] для побудови інтерфейсу користувача та бібліотеку Fabric.js [5], для роботи з HTML5 Canvas [4]. Canvas-технологія забезпечує відмалювання зображень і контейнерів, а Fabric.js спрощує маніпуляції з об'єктами на полотні (масштабування, перетягування тощо). Використання Angular дозволило побудувати компонентну архітектуру: кожен елемент колажу (рядок, колонка, контейнер) став окремим компонентом із власною логікою.

Зв'язки між компонентами реалізовані через алгоритми властностей та подій Angular: наприклад, подія «зміна ширини секції» відправляється батьківському компоненту рядка, який у відповідь змінює властивості сусідніх секцій. Така модель полегшує підтримку рекурсивних залежностей. Крім того, використано технології WebGL [7] для апаратного прискорення рендерингу (через Fabric.js), що дозволяє плавно перетягувати повзунки навіть у великих колажах. Діаграму класів інформаційної системи наведено на рис. 12.

Після реалізації, модель інтегрована у веб-додаток та протестована на різних сценаріях редагування. Зокрема, перевірено: додавання/видалення секцій, зміна розмірів контейнерів у різних частинах колажу, екстремальні співвідношення (коли один елемент займає майже весь простір, а інші мінімальні), тощо. В усіх випадках система підтримувала узгодженість – жодних накладень чи виходів за межі полотна не відбувалося, всі пов'язані елементи коректно реагували на зміни.

Порівняльний аналіз з традиційними моделями. Для кількісної оцінки ефективності було змодельовано типову операцію редагування: збільшення одного з контейнерів у колажі та приведення решти елементів до узгодженого вигляду. В умовах Canva або Pixlr Editor ця операція вимагала, як уже зазначалося, до 10 послідовних ручних дій. У нашій деревовидній моделі той самий результат досягається двома діями: пересуванням відповідного вертикального повзунка, а потім (якщо зміна висоти теж необхідна) горизонтального повзунка. Наприклад,

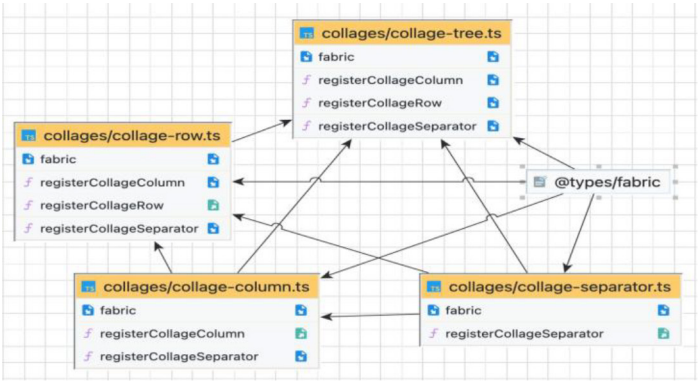


Рис. 12. Діаграма класів інформаційної системи

для колажу з Pixlr Editor потрібно було окремо змінити ширини кількох контейнерів і їх розташування, щоб усунути накладання. Натомість у деревовидній моделі достатньо перетягнути один повзунок ширини і один повзунок висоти – система сама перерахує всі пов'язані параметри. У результаті отримуємо, що виконання 2-ох дій у розробленій моделі, потребує виконання 10-ти дій, у моделях відомих систем, тим самим зменшено кількість дій у 5 разів. Це суттєво економить час і зусилля користувача. Крім того, автоматизація усуває можливі помилки, пов'язані з ручним налаштуванням (наприклад, неточне вирівнювання). Результати тестування підтвердили, що деревовидна модель колажів забезпечує високу інтерактивність та адаптивність. Користувач може в реальному часі бачити, як зміни одного елементу впливають на всю композицію: перетягування повзунка відразу перерозподіляє простір між зображеннями, зберігаючи загальний баланс. Це робить процес редагування інтуїтивно зрозумілішим і швидшим, оскільки відпадає потреба вручну контролювати кожен фрагмент – система сама підтримує логічні зв'язки і пропорції. Звісно, розроблені моделі алгоритмів не вирішують всіх можливих задач автоматичної композиції (наприклад, вона не підказує, які саме зображення обрати або як переставити їх місцями для кращої композиції – ці аспекти належать до сфери інтелектуальних алгоритмів компонування). Проте вони ефективно вирішують поставлену проблему узгодженого редагування вже існуючого макету. У поєднанні з алгоритмами автоматичної генерації початкового розташування (де можуть використовуватися згадані ШІ-моделі), деревовидна модель може стати потужною основою для сучасних інформаційних систем.

Висновки

У роботі створено та досліджено модель опрацювання колажів, яка забезпечує:

Автоматичну адаптацію елементів колажу. Зміни в одній частині композиції автоматично і пропорційно впливають на всі пов'язані елементи, зберігаючи загальну цілісність. Це усуває проблему розсинхронізації елементів, характерну для традиційних моделей колажів без зв'язків.

Підвищення швидкості та зручності редагування. Кількість дій користувача зменшується у кілька разів (приблизно в 5 разів для розглянутого сценарію) завдяки тому, що система сама виконує рутинні коригування сусідніх елементів. Редагування складних колажів стає менш трудомістким і більш інтуїтивним.

Масштабованість на складні композиції. Модель підтримує багаторівневі структури (рядки всередині рядків у вигляді колонок), що дає змогу створювати колажі довільної складності та розміру. В кожному випадку зберігається працездатність алгоритмів перерозподілу розмірів.

Програмна реалізація і апробація моделей алгоритмів колажів підтвердила їх ефективність та правильність функціонування. Створені алгоритми пропорційного перерозподілу коректно опрацьовують граничні випадки (мінімальні розміри), забезпечуючи відсутність накладань та виходів за межі полотна. Порівняння з традиційними моделями (реалізованими в інформаційних системах Canva, Pixlr) продемонструвало очевидні переваги у швидкості редагування та підтримці цілісності композиції. Напрямки подальших досліджень можуть включати інтеграцію елементів штучного інтелекту для автоматичної генерації початкового макету колажу на основі змістового аналізу зображень (наприклад, групування тематично схожих зображень, як у методі кореляційного колажу. Це дозволило б поєднати переваги автоматичного компонування з гнучкістю створеної моделі редагування. Також перспективним є розширення моделі на випадки нестандартних форм області колажу (не прямокутна канва) та адаптація під мобільні пристрої з торкальним інтерфейсом, де користувач міг би жестами змінювати структуру колажу. Розроблена деревовидна модель може бути ефективно впроваджена в сучасні онлайн-платформи редагування зображень, підвищуючи їх продуктивність, зручність користування та загальну функціональність у задачах створення складних графічних колажів. Розроблена модель може бути використана в інформаційних системах орієнтованих на UI/UX дизайнерів, фотографів, рекламних агенцій тощо.

Список використаної літератури

1. Adobe Photoshop. (n.d.). Photoshop.adobe.com. URL: https://photoshop.adobe.com/discover
2. Angular. (2025). Angular.dev. URL: https://angular.dev/api
3. Canva. (2025). Canva. URL: https://www.canva.com/
4. Canvas API. (2019, December 2). MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
5. JSDoc: Home. (n.d.). Fabricjs.com. URL: http://fabricjs.com/docs/
6. Liu, L., Zhang, H., Jing, G., Guo, Y., Chen, Z., & Wang, W. (2017). Correlation-Preserving Photo Collage. *IEEE Transactions on Visualization and Computer Graphics*, vol. 24(6), pp. 1956–1968. DOI: 10.1109/TVCG.2017.2703853
7. Mozilla. (2019, March 4). WebGL: 2D and 3D graphics for the web. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
8. Online Photo Editor | Pixlr Editor. (2019). Pixlr.com. URL: https://pixlr.com/editor/
9. Rother, C., Bordeaux, L., Hamadi, Y., & Blake, A. (2006). AutoCollage. *ACM Transactions on Graphics*, vol. 25(3), pp. 847–852. DOI: 10.1145/1141911.1141965
10. Yu, J., Chen, L., Zhang, M., & Li, M. (2022). SoftCollage: A Differentiable Probabilistic Tree Generator for Image Collage. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (CVPR), pp. 3719–3728. DOI: 10.1109/CVPR52688.2022.00371
11. Zhang, M., Li, M., Chen, L., & Yu, J. (2021). Aesthetic Photo Collage with Deep Reinforcement Learning. *ArXiv (Cornell University)*. DOI: 10.48550/arXiv.2110.09775
12. Fernández, D. P., Postels, J., & Tombari, F. (2021). Variational Transformer Networks for Layout Generation. *Conference: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/cvpr46437.2021.01343
13. Chen, L., Jing, Q., Tsang, Y., & Zhou, T. (2024). Iris: a multi-constraint graphic layout generation system. *Frontiers of Information Technology & Electronic Engineering*, vol. 25(7), pp. 968–987. DOI: 10.1631/fitee.2300312
14. Huo, H., & Wang, F. (2022). A Study of Artificial Intelligence-Based Poster Layout Design in Visual Communication. *Scientific Programming*, vol. 2022, pp. 1–9. https://doi.org/10.1155/2022/1191073
15. Cao, Y., Ma, Y., Zhou, M., Liu, C., Xie, H., Ge, T., & Jiang, Y. (2022). Geometry Aligned Variational Transformer for Image-conditioned Layout Generation. *Proceedings of the 30th ACM International Conference on Multimedia*, vol. 1, pp. 1561–1571. DOI: 10.1145/3503161.3548332
16. Lu, Y., Tong, Z., Zhao, Q., Oh, Y., Wang, B., & Li, T. J.-J. (2024). Flowy: Supporting UX Design Decisions Through AI-Driven Pattern Annotation in Multi-Screen User Flows. ArXiv (Cornell University). DOI: 10.48550/arxiv.2406.16177
17. Shabani, M. A., Wang, Z., Liu, D., Zhao, N., Yang, J., & Furukawa, Y. (2024). Visual Layout Composer: Image-Vector Dual Diffusion Model for Design Layout Generation. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9222–9231. DOI: 10.1109/cvpr52733.2024.00881
18. Shi, Y., Shang, M., & Qi, Z. (2023). Intelligent layout generation based on deep generative models: A comprehensive survey. *Information Fusion*, vol. 100, pp. 101940–101940. DOI: 10.1016/j.inffus.2023.101940
19. Hadi AlZayer, Lin, H., & Bala, K. (2021). AutoPhoto: Aesthetic Photo Capture using Reinforcement Learning. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 944–951. DOI: 10.1109/iros51168.2021.9636788
20. Zhou, M., Xu, C., Ma, Y., Ge, T., Jiang, Y., & Xu, W. (2022). Composition-aware Graphic Layout GAN for Visual-Textual Presentation Designs. *Proceedings of the 31 International Joint Conference on Artificial Intelligence*, pp. 4995–5001. DOI: 10.24963/ijcai.2022/692
21. Zhou, Y., Jing, Q., Li, Z., Shi, L., Chen, L., & Sun, L. (2023). Element-Conditioned Gan for Graphic Layout Generation. *Neurocomputing*, vol. 591, pp. 127730. DOI: https://doi.org/10.1016/j.neucom.2024.127730