Міністерство освіти і науки України

Черкаський національний університет імені Богдана Хмельницького

Факультет обчислювальної техніки, інтелектуальних та управляючих систем Кафедра інформаційних технологій

КУРСОВА РОБОТА

з дисципліни «Програмуваннята алгоритмічні мови»

на тему: «Генератор кросвордів »

Студента 1го курсу КІ-18 групи

спеціальності

Системний аналіз (124)

(код Назва)

Вакулінський О. Ю.

(прізвище та ініціали)

Керівник: Гребенович Ю. Є.

(прізвище та ініціали)

Оцінка за шкалою:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(національною, кількість балів, ECTS)

Члени комісії: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Черкаси – 2019

**Зміст**

[**Вступ** 3](#_Toc8962279)

[**Розділ 1. Огляд інформаційних джерел** 5](#_Toc8962280)

[1.1. Поняття кросворду. Різновиди кросвордів 5](#_Toc8962281)

[1.2. Існуючі генератори кросвордів 5](#_Toc8962282)

[1.3. Метод реалізації 7](#_Toc8962283)

[1.4. Прийняття рішень 7](#_Toc8962284)

[**Розділ 2. Розробка структури програми та алгоритмів її роботи** 8](#_Toc8962285)

[2.1. Загальний алгоритм роботи програми 8](#_Toc8962286)

[2.2. Загальна структура програми 8](#_Toc8962287)

[2.3. Алгоритм перебору з поверненням 11](#_Toc8962288)

[**Розділ 3. Реалізація програмного проекту** 13](#_Toc8962289)

[3.1. Основний метод 13](#_Toc8962290)

[3.2. Методи 16](#_Toc8962291)

[3.3. Висновки щодо розділу 21](#_Toc8962292)

[**Висновки** 22](#_Toc8962293)

[**Список використаних джерел** 23](#_Toc8962294)

[**Додатки** 24](#_Toc8962295)

[Додаток А. Зображення початкового та кінцевого станів програми 24](#_Toc8962296)

[Додаток Б. Код основного алгоритму програми 25](#_Toc8962297)

[Додаток В. Блок-схема основного алгоритму 31](#_Toc8962298)

# **Вступ**

Мета курсової роботи: розробка програми генератора кросвордів.

Задачі курсової роботи:

* вивчення алгоритму перебору з поверненням
* розробити власні функції для створення, збереження та читання словника та шаблону кросворда
* розробка функції для вставки слова до відповідного рядка сітки;
* побудувати блок-схему програмного алгоритму перебору з поверненням;
* реалізувати спроектований програмний продукт;
* зробити висновки щодо виконаної роботи та оформити пояснювальну записку.

Актуальність: Кросворди можна використовувати в навчальному процесі для запам’ятовування термінів і визначень, самостійної роботи учнів, як своєрідний метод (прийом) тематичного або підсумкового контролю знань учнів в ігровій форми. Великий інтерес в учнів професійно-технічних навчальних закладів (ПТНЗ) викликає складання кросвордів та робота з ними за тематикою навчальних дисциплін, що вивчаються. Завдяки тому, що це досить поширена і улюблена багатьма форма дозвілля, то ігрова навчальна діяльність звільняє учнів від скутості. Даний вид роботи допомагає учням не тільки в цікавій формі перевірити і закріпити свої знання з навчальної дисципліни, але й розвиває світогляд учнів, підвищує його мотивацію, а також прививає інтерес до вивчення предмету, оволодіння професією. Практика показує, що завдання розгадати кросворд сприймається учнями позитивно і з готовністю до дії. Традиційно, складання кросвордів здійснюється «в ручну» засобами прикладних програм MS Office Word, MS Office Excel, Paint, що займає багато часу. Сучасні засоби  комп’ютерного створення кросвордів дозволяють суттєво полегшити цей процес. Ці програмні продукти прості у використанні, швидко і автоматично створюють кросворди різних типів і конфігурацій.

# **Розділ 1. Огляд інформаційних джерел**

* 1. Поняття кросворду. Різновиди кросвордів

Кросворд - гра-задача, яка полягає у вписуванні літер у перехресні рядки клітинок накресленої фігури так, щоб за горизонталлю та вертикаллю вийшли загадані слова[3].

Існує простіша головоломка під назвою кріс-крос. Кожен кріс-крос складається з списку слів, для зручності розбитих на групи у відповідності з довжиною слів, і схеми в яку потрібно вписати слова. Схема підпорядковується тим же правилам, що і в класичному кросворді – в місцях перетину слова мають спільну букву, але номери відсутні, так як слова відомі завчасно, потрібно їх лише вписати в схему кріс-кроса. Зазвичай, кріс-крос має менше перетинів ніж звичайний кросворд, а також завжди має хоча б один правильний варіант заповнення[1].

Саме кріс-крос взятий мною, як приклад для реалізації даної програми, адже в моїй програмі основний алгоритм – це генерація слів. Тобто немає обов’язкової прив’язки до питань, а є вже готовий список слів.

* 1. Існуючі генератори кросвордів
* КроссВорд 1.4 – програма створює класичні кросворди, сітка генерується випадковим чином, заповнюється словами, можна записати у файл, надрукувати, і розгадувати на папері. Також можна вручну змінити форму, програма її перенаповнить. У словнику програми в процесі її роботи відбувається ротація, тому ймовірність отримання схожих кросвордів мінімальна. Є зручна система підказок, звукових ефектів та інше.
* Decalion 1.2 - програма призначена для складання кросворду з наступним його оформленням у власному стилі, вставляти зображення, змінювати товщину, форму і колір клітинок, розмір шрифтів, друкувати або експортувати в графічний файл. Забезпечується швидкий пошук необхідних слів, а також підтримується можливість створення власних словників. У програмі реалізовані підтримка швидкої зміни форми кросворду, миттєвий пошук, підказки програми, визначення у складі словників, автоматична нумерація слів, система контролю за повтореннями, неіснуючими та хибними словами – це дозволяє збільшити швидкість складання і поліпшити якість роботи.
* Crossword Set 1.01 – програма складається з трьох програм: CrossGrid (створення класичних кросвордів); CrossDict (створення словника для складання класичних кросвордів); CrossJapa (створення японських кросвордів). За допомогою цих програм можна легко і швидко створювати класичні і японські кросворди будь-якої складності. Кросворди можуть бути збережені в текстовому файлі або в файлі HTML-сторінки (за шаблонами). Отримані кросворди можна відразу вирішувати (в інтернет-браузері) або роздрукувати.
* Eclipse Crossword, программа автоматично створює кросворди за списком слів з ключами, з однакових слів списку вона складає декілька варіантів кросвордів з різними перетинами слів, що дозволяє використовувати для створення кросвордів у декількох варіантах, або обрати той що більше подобається і роздрукувати готовий кросворд. Дозволяє доповнювати списки слів, об’єднувати їх, тим самим оперативно на любому етапі вивчення теми складати кросворди від простих до складних, підсумкових з теми або з всього курсу та інші[2].

* 1. Метод реалізації

Додаток Windows Forms для Visual C#. Windows Forms - інтерфейс програмування додатків, що відповідає за графічний інтерфейс користувача, є частиною Microsoft.NET Framework і спрощує доступ до елементів інтерфейсу Microsoft Windows. Всередині .NET Framework, Windows Forms реалізується в межах простору імен System.Windows.Forms.

Бібліотека Windows Forms - це платформа для розробки графічних додатків на базі .NET Framework. Дана бібліотека містить набір простих у використанні і розширюваних класів, що дозволяють створювати додатки з розвинутим інтерфейсом.

Студія передбачає використання стандартних об’єктів (кнопка, надпис, текстове поле, картинка тощо) з інтуїтивно зрозумілими параметрами (висота, ширина і т.п.). Більш того, в студії процес зміни деяких з параметрів, додавання нового об’єкта в форму навіть не потребує написання коду - все це є візуалізованим. Користувач просто перетягує необхідний об’єкт з панелі на форму.

Виконуваний код - класи, що реалізують API для Windows Forms, не залежать від мови розробки. Тобто програміст однаково може використовувати Windows Forms як при написанні програмного забезпечення на C#, так і на інших мовах програмування.

* 1. Прийняття рішень

Так як моя програма потребує лише в використанні стандартних об’єктів, я обрав Windows Forms Також візуальна зміна параметрів полегшує написання коду.

В програмі користувач може створити або відкрити існуючий шаблон(файл типу .csc), відкрити список слів (файл типу .txt). Після запуску генерації ми отримаємо заповнений кросворд[4].

# **Розділ 2. Розробка структури програми та алгоритмів її роботи**

## 2.1. Загальний алгоритм роботи програми

1. Запуск програми
2. Відкрити шаблон( створити новий шаблон, якщо не існує)
3. Відкрити текстовий документ
4. Присвоєння словам значення по горизонталі та по вертикалі
5. Запуск генерації кросворда
6. Знаходження варіантів перетину кросворда(кількість варіантів може обрати користувач)
7. Заповнення шаблону

## 2.2. Загальна структура програми

Класи і структури, що представляють модель генератора об’єднані в простір імен Model. Інші елементи знаходяться або в кореневому просторі імені проекту, або в службовому

**Клас Grid. Шаблон (Сітка) кросворду**

Цей клас представляє ні що інше, як шаблон кросворду: прямокутний масив логічних значень {комірка символу (true), пусте "чорне" місце (false)}. До задач класу входить зовсім небагато: зміна розміру шаблону, редагування комірок (вставка комірки для символу або порожнього місця) і формування списку шаблонів слів, як по горизонталі, так і по вертикалі.

**Клас Word. шаблон слова**

Це шаблон слова, який є основною структурною одиницею кросворду. Шаблон характеризується координатами на сітці кросворду, орієнтацією (по горизонталі або по вертикалі), довжиною формованого слова і символами, які разом або утворюють значення слова - поточне значення шаблону слова, або є символами, які перебувають в місцях перетину з іншими шаблонами слів. Незаповнені місця в шаблоні слова представлені пробілами, а для зберігання його орієнтації використовується перерахування EWordDirection.

**Клас Words. Колекція шаблонів слів**

Колекція шаблонів слів формується екземпляром класу Grid для конкретного кросворду. Клас Words розбиває всю колекцію слів на два списки: слова по горизонталі і слова по вертикалі. Також, в ході побудови колекції (клас Grid, функція CreateWords), відбувається формування списку всіх перетинів шаблонів слів (клас Words, метод Prepare), що складається з екземплярів класу WordIntersection. Клас WordIntersection є допоміжним структурним елементом і використовується класом Words для обробки перетинів в операціях присвоєння значення і скасування значення шаблоном слова. Отже, крім доступу до елементів колекції, клас Words надає наступну функціональність:

* Перевірка, чи можна поточному шаблоном слова з урахуванням всіх його перетинів привласнити конкретне значення - функція CanAssign.
* Присвоєння значення шаблону слова з обробкою всіх пов'язаних з ним перетинів - метод AssignValue.
* Скасування значення шаблону слова з відновленням символів на перетинах - метод ClearValue.
* Створення копії екземпляра Words, як одного з варіантів заповненого шаблону - функція Copy.

**Клас Content. Генератор заповнених варіантів кросворду**

Це основний клас всієї моделі, в задачу якого входить формування словника для генерації заповнених варіантів кросворду і, власне, сама генерація цих варіантів.

Метод Append додає нові слова в словник, виділяючи їх з текстового рядка за допомогою регулярного виразу. Кожен елемент словника представлений примірником допоміжного класу ContentItem. Примірники ContentItem, крім значення слова, зберігають ознаку, використано дане слово при заповненні кросворду чи ні, щоб уникнути повторень в ході повного перебору варіантів. Весь словник розбитий на розділи, кожен з яких містить слова однієї довжини. Зроблено це для того щоб при переборі варіантів значень для конкретного шаблону слова перевіряти тільки значення потрібної довжини.

Метод Generate приймає в якості аргументу екземпляр класу Grid і починає для нього повний перебір варіантів заповнення шаблону кросворду. Також, є перевизначена версія методу Generate, яка приймає в якості аргументу ще й максимальну кількість необхідних варіантів заповнення, сформувавши яке, повний перебір завершується. Метод StopGeneration зупиняє повний перебір, якщо метод Generate був запущений в паралельному потоці. Алгоритм генерації заповнених варіантів попередньо сортує шаблони слів по горизонталі - по зростанню їх довжини, а слова по вертикалі - по спадаючій. Сортування відбувається з використанням допоміжного сервісу порівняння двох шаблонів слів - класу WordComparer.

**Компонент TemplateControl**

Цей елемент керування надає можливості користувачу за допомогою графічного інтерфейсу редагувати шаблон кросворду, а саме: "малювати" шаблони слів, натискаючи мишею на відповідних осередках шаблону. Головним "продуктом" його роботи є екземпляр класу Grid, доступ до якого можна отримати за допомогою однойменного властивості. До додаткових обов'язків компонента TemplateControl відносяться збереження шаблону кросворду в текстовий файл і читання його звідти, а також, відображення одного з варіантів заповнення кросворду - метод DrawWords, аргументом якого, як не складно здогадатися, є екземпляр класу Words.

Окремо варто відзначити, що з метою уникнути всіляких моргань елемента управління сіткою кросворду в процесі малювання слів, компонент TemplateControl використовує не просто екземпляри Panel, а перевизначені їхньою версією - екземпляри DoubleBufferedPanel. У конструкторі класу DoubleBufferedPanel захищенній (protected) властивості DoubleBuffered присвоєно значення true.

**Компонент ContentControl**

Як нескладно здогадатися, це елемент управління словником, використовуваним для генерації заповнених варіантів кросворду. Крім читання текстових файлом для наповнення словника, компонент ContentControl також синхронізує свою роботу з роботою екземплярами TemplateControl, посилання на який передається йому в процесі ініціалізації. Під синхронізацією розуміють запуск генерації кросвордів в синхронному або асинхронному режимі і відображення засобами TemplateControl обраного варіанту кросворду. Зрозуміло, що більшу частину роботи ContentControl виконує не сам, а делегує її екземплярами Grid і Content - елементам моделі генератора кросвордів.

**Форма ProceedingForm**

Тут все просто. Форма ProceedingForm має відображати процес повного перебору варіантів заповнення для заданого шаблону кросворду і, в разі необхідності, зупиняти його роботу. Форма створюється і відкривається компонентом ContentControl в момент асинхронного запуску процесу генерації - в тілі методу AsyncGenerate. Форма автоматично закривається після завершення або зупинки повного перебору - в тілі обробника події AsyncGenerateCompleted.

**Форма MainForm**

Форма MainForm - головна форма програми, в завдання якої входить створення екземплярів TemplateControl і ContentControl, їх ініціалізація, а також, управління ними в процесі обробки подій головного меню і "тулбару".

## 2.3. Алгоритм перебору з поверненням

Алгоритм перебору з поверненням реалізують за допомогою рекурсії: на кожному кроці рекурсії здійснюють пошук частинного рішення для одного елемента, і, знайшовши його, рекурсивно запускають той же пошук для наступного елемента. Якщо для поточного елемента рішення знайти не вдається, то здійснюється перехід до попереднього елемента (повернення з рекурсії на один крок), де знайдене рішення скасовується і починається пошук іншого варіанту.

Як тільки знайдені рішення для всіх елементів - знайдено всі рішення. Далі, алгоритм зберігає варіант рішення і продовжує шукати інші. В ході аналізу рекурсивного алгоритму перебору дуже важливо вміти давати оцінку, чи містить поточна гілка рекурсії рішення, або її можна відсікти. Наприклад, чи варто намагатися розставляти інші слова в кросворді, коли ви натрапили на нерозв'язне перетин: в словнику немає слів потрібної довжини для підстановки в шаблон з дотримань умов перетину. Аналіз і відсікання свідомо "безрезультатних" гілок рекурсії називається методом гілок і меж. Оптимізація алгоритмів повного перебору може проводитися з використанням евристичного підходу до вирішення завдання. Цей підхід пов'язаний з набором припущень, правильність яких довести або складно або формально неможливо, але дотримання їх дозволяє знаходити рішення досить швидко. Нагадаю, що мета предпринимаемой оптимізації - якомога швидше знайти хоча б одне рішення, а не всі варіанти заповнення кросворду. Таким чином, оптимізація рекурсивного алгоритму повного перебору, спрямована на мінімізацію кількості елементарних операцій здійснюється за рахунок застосування двох підходів:

* Пошук і відсікання гілок рекурсії, які свідомо не містять рішень - метод гілок і меж.
* Вибір, в першу чергу, тих гілок рекурсії, які містять найбільшу кількість рішень - евристичний підхід.

# **Розділ 3. Реалізація програмного проекту**

* 1. Основний метод

У двох словах представлений алгоритм робить наступне: спочатку розставляє слова в одному напрямку - перша лінія (GenerateFirstLine), а потім намагається розставити слова в іншому напрямку з урахуванням перетинів - друга лінія (GenerateSecondLine). Зрозуміло, що в ході заповнення першої лінії не потрібно враховувати ніякі перетину, а досить просто підбирати слова відповідної шаблоном довжини.

На кожному кроці здійснюється пошук відповідного варіанту для конкретного шаблону слова і, якщо пошук вдалий, здійснюється перехід до наступного шаблону. Рішення для послідовності кроків в рамках однієї лінії слабо залежні, оскільки пов'язані тільки перевіркою, чи не використано конкретне слово зі словника при заповненні шаблона на попередніх кроках (заповнення декількох шаблонів слів одним і тим же значенням заборонено!). Але кроки другої лінії дуже і дуже залежать від знайдених рішень першої лінії .

//Крок 1. Розстановка слів в першому напрямку

private void GenerateFirstLine(Words \_Words, Word[] \_FirstLineWords, Word[] \_SecondLineWords)

{

int \_BackToIndex = -1;

GenerateFirstLine(\_Words, \_FirstLineWords, \_SecondLineWords, 0, ref \_BackToIndex);

}

private void GenerateFirstLine(Words \_Words, Word[] \_FirstLineWords, Word[] \_SecondLineWords, int \_Index, ref int \_BackToIndex)

{

if (!m\_ContinueGeneration)

return;

Word \_Word = \_FirstLineWords[\_Index];

\_Word.Index = \_Index;

List<ContentItem> \_Contents = this.GetContent(\_Word.Length);

if (\_Contents == null)

return;

if (\_Index == 0)

{

m\_ProgressValue = 0;

m\_ProgressMaxValue = \_Contents.Count;

}

foreach (ContentItem \_item in \_Contents)

{

m\_OperationsCount++;

m\_CurrentWord = \_Word; m\_CurrentItem = \_item;

if (\_Index == 0)

m\_ProgressValue++;

if (!\_item.Used)

{

\_Words.AssignValue(\_Word, \_item.Value, true);

\_item.Used = true;

\_BackToIndex = -1;

if (\_Index == \_FirstLineWords.Length - 1)

GenerateSecondLine(\_Words, \_SecondLineWords, 0, 0, ref \_BackToIndex);

else

GenerateFirstLine(\_Words, \_FirstLineWords, \_SecondLineWords, \_Index + 1, ref \_BackToIndex);

\_Words.ClearValue(\_Word, false);

\_item.Used = false;

if (\_BackToIndex != -1 && \_BackToIndex < \_Index)

return;

else

\_BackToIndex = -1;

}

}

}

Друга оптимізація - евристичний підхід. Робимо припущення, що якщо шаблони слів, що заповнюються на першій лінії впорядкувати за зростанням, а шаблони слів другої лінії - по спадаючій довжини, то це допоможе швидше знайти перший готовий варіант заповненого кросворду. Мотивація наступна:

* Найбільша кількість повернень на попередні кроки рекурсії буде відбуватися на межі першої і другої лінії.
* Чим більше довжина поточного шаблону слова, тим серйозніше впливає його значення на хід пошуку за все залишився рішення.

//Починаємо генерацію з переважаючого напрямку.

if (\_Words.HorizontalWords.Count < \_Words.VerticalWords.Count)

{

//Сортування слів по горизонталі у спадаючому порядку

\_Words.HorizontalWords.Sort(new WordComparer(-1));

// Сортування слів по вертикаллі у зростаючому порядку

\_Words.VerticalWords.Sort(new WordComparer(1));

//Запуск генерації

GenerateFirstLine(\_Words, \_Words.VerticalWords.ToArray(), \_Words.HorizontalWords.ToArray());

}

else

{

// Сортування слів по горизонталі у зростаючому порядку

\_Words.HorizontalWords.Sort(new WordComparer(1));

// Сортування слів по вертикаллі у спадаючому порядку

));

//Запуск генерації

GenerateFirstLine(\_Words, \_Words.HorizontalWords.ToArray(), \_Words.VerticalWords.ToArray());

}

Таким чином, найбільш довгі шаблони слів сконцентровані біля кордону двох ліній рекурсивного алгоритму. Перебір слів при спробі заповнити довгі шаблони буде приводити до швидкої зміни областей пошуку рішень, а значить, збільшить ймовірність наштовхнутися хоча б на одне з них на самому початку повного перебору варіантів.

Третя оптимізація - метод гілок і меж. Якщо пошук значення для шаблону слова в рамках другої лінії не приносить результату, на попередньому кроці заповнювався шаблон тієї ж лінії (тієї ж орієнтації), то немає ніякого сенсу повертатися на попередній крок рекурсії, якщо довжина попереднього шаблону більше (пам'ятаємо, що довжини шаблонів відсортовані по спаданню). Сенсу в цьому немає, оскільки в даному випадку всі попередні рішення другої черги незалежні з рішенням для поточного шаблону слова. Має сенс повернутися до того шаблону першої черги, з яким є перетин. Якщо пересічний кілька, то вибираємо найближчий крок рекурсії - шаблон з найбільшою ординатою.

//Крок 2. Розстановка слів у другому напрямі з урахуванням перетинів

private void GenerateSecondLine(Words \_Words, Word[] \_SecondLineWords, int \_Index, int \_PrevWordLength, ref int \_BackToFirstLineIndex)

{

if (!m\_ContinueGeneration)

return;

if (\_SecondLineWords.Length == 0)

{

if (m\_MaxVariantsCount == 0 || m\_Variants.Count < m\_MaxVariantsCount)

m\_Variants.Add(\_Words.Copy());

if (m\_Variants.Count == m\_MaxVariantsCount)

StopGeneration();

return;

}

Word \_Word = \_SecondLineWords[\_Index];

List<ContentItem> \_Contents = this.GetContent(\_Word.Length);

if (\_Contents == null)

return;

if (\_Index == 0)

{

m\_CurrentDepth = 0;

m\_MaxDepth = \_SecondLineWords.Length;

}

m\_CurrentDepth = \_Index + 1;

int \_MaxFirstNonEqualIndex = -1;

foreach (ContentItem \_item in \_Contents)

{

m\_OperationsCount++;

m\_CurrentWord = \_Word; m\_CurrentItem = \_item;

if (!\_item.Used)

{

int \_FirstNonEqualIndex = -1;

if (\_Words.CanAssign(\_Word, \_item, ref \_FirstNonEqualIndex))

{

\_Words.AssignValue(\_Word, \_item.Value, false);

\_item.Used = true;

\_BackToFirstLineIndex = -1;

if (\_Index == \_SecondLineWords.Length - 1)

{

if (m\_MaxVariantsCount == 0 || m\_Variants.Count < m\_MaxVariantsCount)

m\_Variants.Add(\_Words.Copy());

if (m\_Variants.Count == m\_MaxVariantsCount)

StopGeneration();

}

else

GenerateSecondLine(\_Words, \_SecondLineWords, \_Index + 1, \_Word.Length, ref \_BackToFirstLineIndex);

\_Words.ClearValue(\_Word, true);

\_item.Used = false;

if (\_BackToFirstLineIndex != -1)

return;

}

else

{

if (\_MaxFirstNonEqualIndex == -1 || \_MaxFirstNonEqualIndex < \_FirstNonEqualIndex)

\_MaxFirstNonEqualIndex = \_FirstNonEqualIndex;

}

}

}

if (\_MaxFirstNonEqualIndex != -1 && \_PrevWordLength != \_Word.Length)

\_BackToFirstLineIndex = \_Words.GetCrossedIndex(\_Word, \_MaxFirstNonEqualIndex);

* 1. Методи

Метод формування слів по текстовому рядку **Append** - оптимізація використовуваних структур даних: розбити весь словник на розділи, кожен з яких містить слова тільки однієї довжини, і організувати швидкий доступ до розділу по заданій довжині. Зроблено це з використанням словника на базі хеш-таблиці, значеннями якого є списки слів - розділи, а ключем - довжина слів у розділі. Оскільки перебір слів першої лінії виконується швидше (не враховуються перетину), то першою лінією є той напрямок, кількість шаблонів слів якого більше.

public void Append(string \_Text)

{

Regex \_regex = new Regex("(?i)[а-я]{2,}"); //Мінімальна довжина слова – 2 символа

MatchCollection \_textMatches = \_regex.Matches(\_Text);

foreach (Match \_wordMatch in \_textMatches)

{

string \_word = \_wordMatch.Value.ToUpper();

List<ContentItem> \_wordsl = null;

if (m\_Content.ContainsKey(\_word.Length))

\_wordsl = m\_Content[\_word.Length];

else

{

\_wordsl = new List<ContentItem>();

m\_Content.Add(\_word.Length, \_wordsl);

}

bool \_Exists = false;

IEnumerator<ContentItem> \_items = \_wordsl.GetEnumerator();

\_items.Reset();

while (!\_Exists && \_items.MoveNext())

{

ContentItem \_item = \_items.Current;

if (\_item.Value == \_word)

\_Exists = true;

}

if (!\_Exists)

\_wordsl.Add(new ContentItem(\_word));

}

}

**Copy()** Копіювання колекції слів; Використовується при додаванні поточного варіанту в колекцію результатів генерації кросворду

public Words Copy()

{

Words \_Copy = new Words();

foreach (Word \_Word in this)

\_Copy.Add(\_Word.Copy());

return \_Copy;}

**CreateWords()** Формує список слів кросворда по даним сітки кросворда, задає значення по горизонталі чи по вертикалі

public Words CreateWords()

{

Words \_words = new Words();

//Формування слів по горизонталі

for (int j = 0; j < this.Height; j++)

{

int x = 0; //Початок поточного слова

int l = 0; //Довжина поточного слова

for (int i = 0; i < this.Width; i++)

if (m\_Grid[i, j])

{

if (l == 0)

x = i;

l++;

}

else

{

if (l > 1)

\_words.Add(new Word(x, j, EWordDirection.wdHorizontal, l));

l = 0;

}

if (l > 1)

\_words.Add(new Word(x, j, EWordDirection.wdHorizontal, l));

}

//Формування слів по вертикалі

for (int i = 0; i < Width; i++)

{

int y = 0; //Початок поточного слова

int l = 0; //Довжина поточного слова

for (int j = 0; j < Height; j++)

if (m\_Grid[i, j])

{

if (l == 0)

y = j;

l++;

}

else

{

if (l > 1)

\_words.Add(new Word(i, y, EWordDirection.wdVertical, l));

l = 0;

}

if (l > 1)

\_words.Add(new Word(i, y, EWordDirection.wdVertical, l));

}

**GetContent()** завдяки цьому методу ми отримуємо розділи словника зі значеннями довжини

public List<ContentItem> GetContent(int \_length)

{

if (!m\_Content.ContainsKey(\_length))

return null;

return m\_Content[\_length];

}

**AssignValue** Присвоєння слову значення з можливим оновленням слів що перетинаються: в місцях перетину для цих слів заповнюються відповідні значенням \_value символи. Якщо слово що перетинаються вже заповнене (IsDefine == true), то нічого оновлювати не потрібно.

public void AssignValue(Word \_Word, string \_value, bool \_UpdateCrosses)

{

\_Word.Value = \_value;

\_Word.IsDefined = true;

if (!\_UpdateCrosses)

return;

int x = \_Word.X;

int y = \_Word.Y;

int xp = 0;

int yp = 0;

if (\_Word.Direction == EWordDirection.wdHorizontal)

{

xp = 1;

yp = 0;

}

else

{

xp = 0;

yp = 1;

}

for (int i = 0; i < \_Word.Length; i++)

{

string \_Key = WordIntersection.GetKey(x, y);

if (m\_Intersections.ContainsKey(\_Key))

{

WordIntersection \_Intersection = m\_Intersections[\_Key];

Word \_CrossedWord = \_Intersection.CrossedBy(\_Word);

if (!\_CrossedWord.IsDefined)

if (\_CrossedWord.Direction == EWordDirection.wdHorizontal)

\_CrossedWord[x - \_CrossedWord.X] = \_Word[i];

else

\_CrossedWord[y - \_CrossedWord.Y] = \_Word[i];

}

x += xp;

y += yp;

}

}

**ClearValue()** Очищення значення слова зі збереженням символів в місцях перетину. Значення символів визначаються за відповідними перетину.

public void ClearValue(Word \_Word, bool \_UpdateCrosses)

{

\_Word.Null();

\_Word.IsDefined = false;

if (!\_UpdateCrosses)

return;

int x = \_Word.X;

int y = \_Word.Y;

int xp = 0;

int yp = 0;

if (\_Word.Direction == EWordDirection.wdHorizontal)

{

xp = 1;

yp = 0;

}

else

{

xp = 0;

yp = 1;

}

for (int i = 0; i < \_Word.Length; i++)

{

string \_Key = WordIntersection.GetKey(x, y);

if (m\_Intersections.ContainsKey(\_Key))

{

WordIntersection \_Intersection = m\_Intersections[\_Key];

Word \_CrossedWord = \_Intersection.CrossedBy(\_Word);

if (\_CrossedWord.IsDefined)

if (\_CrossedWord.Direction == EWordDirection.wdHorizontal)

\_Word[i] = \_CrossedWord[x - \_CrossedWord.X];

else

\_Word[i] = \_CrossedWord[y - \_CrossedWord.Y];

}

x += xp;

y += yp;

}

}

**GetCrossedIndex()** Отримання індексу перетину слова по номеру символу

internal int GetCrossedIndex(Word \_Word, int \_Index)

{

int x = \_Word.X;

int y = \_Word.Y;

int xp = 0;

int yp = 0;

if (\_Word.Direction == EWordDirection.wdHorizontal)

{

xp = 1;

yp = 0;

}

else

{

xp = 0;

yp = 1;

}

x = x + xp \* \_Index;

y = y + yp \* \_Index;

string \_Key = WordIntersection.GetKey(x, y);

if (m\_Intersections.ContainsKey(\_Key))

return m\_Intersections[\_Key].CrossedBy(\_Word).Index;

return -1;

}

* 1. Висновки щодо розділу

В даному розділі описано реалізацію алгоритму перебору з поверненням, алгоритму формування словника та алгоритм перевірки перетину слів. Тестування показало, що програма написана правильно, проблеми в процесі виконання роботи відсутні. Програма виконує поставлену задачу.

# **Висновки**

Програмний продукт був реалізований на мові C# завдяки інтерфейсу програмування додатків (API) Windows Forms . Створена грама здатна розв’язати будь яку, задану користувачем головоломку кріс-крос. При кожному запуску в користувача є можливість створити новий шаблон та зберегти його, або ж відкрити вже існуючий. Алгоритм перебору з поверненням дозволяє доволі швидко знайти розв’язок задачі, а такі оптимізації як : швидкий доступ до словника, евристичний підхід та метод гілок і меж прискорюють роботу програми.

При подальшій розробці планується створення функції збільшення поля, задля збільшення варіативності.

# **Список використаних джерел**

Армбрастер (Armbruster F.). Computer Crosswords, Troubadour Press, San Francisco, CA, 1974.

Л. Майборода м. Київ «Використання комп’ютерних навчальних кросвордів у підготовці майбутніх кваліфікованих робітників.» [Електронний документ]. Режим доступу: http://umo.edu.ua/images/content/nashi\_vydanya/metod\_upr\_osvit/v\_8/17.pdf Перевірено 13.05.2019.

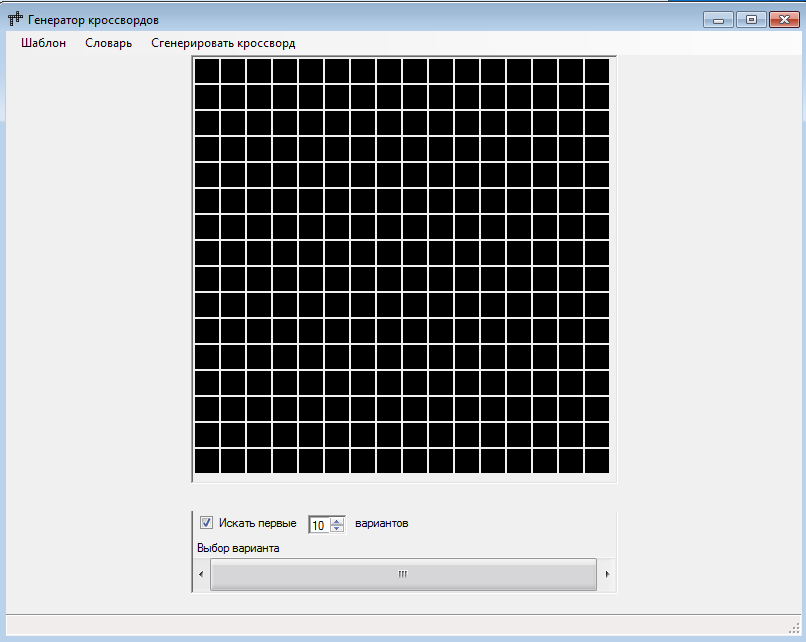
Кросворди [Електронний документ]. Режим доступу: https://uk.wikipedia.org/wiki/Кросворд. Перевірено 13.05.2019.

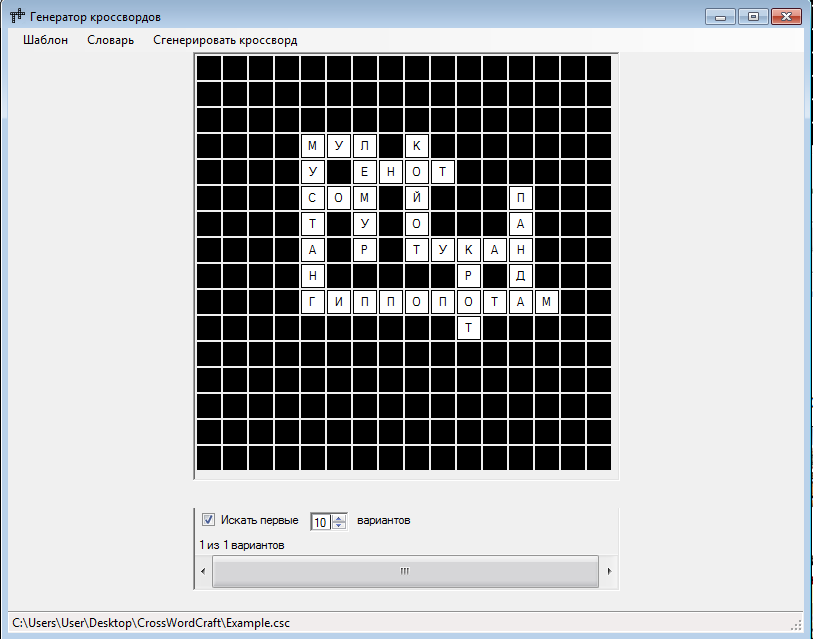
Створення першої програми Windows Forms на C# на прикладі гри «хрестики-нулики» . [Електронний документ]. Режим доступу: <http://isearch.kiev.ua/uk/news/programs/o-appl/1593-create-your-first-windows-forms-application-in-c-for-example-the-game-qtic-tac-toeq>. Перевірено 13.05.2019.

Criss-cross. [Електронний документ]. Режим доступу: http://programming-lang.com/ru/comp\_programming/uezerell/0/j8.html Перевірено 13.05.2019.

# **Додатки**

Додаток А. Зображення початкового та кінцевого станів програми

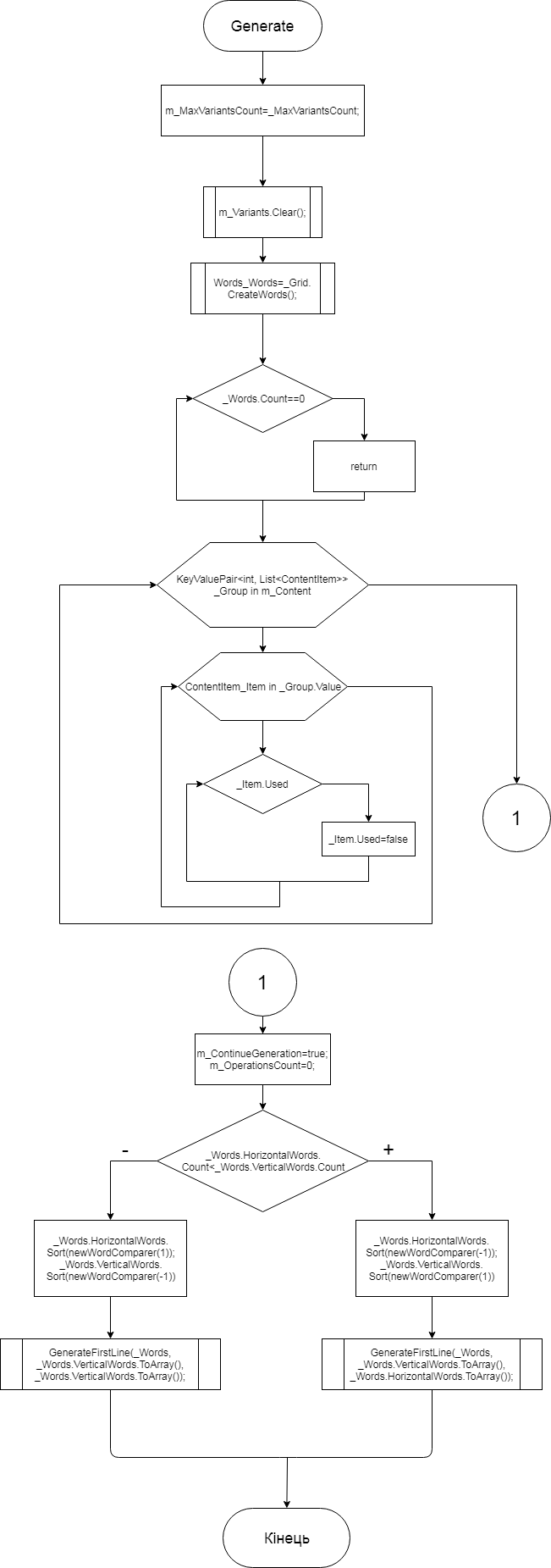


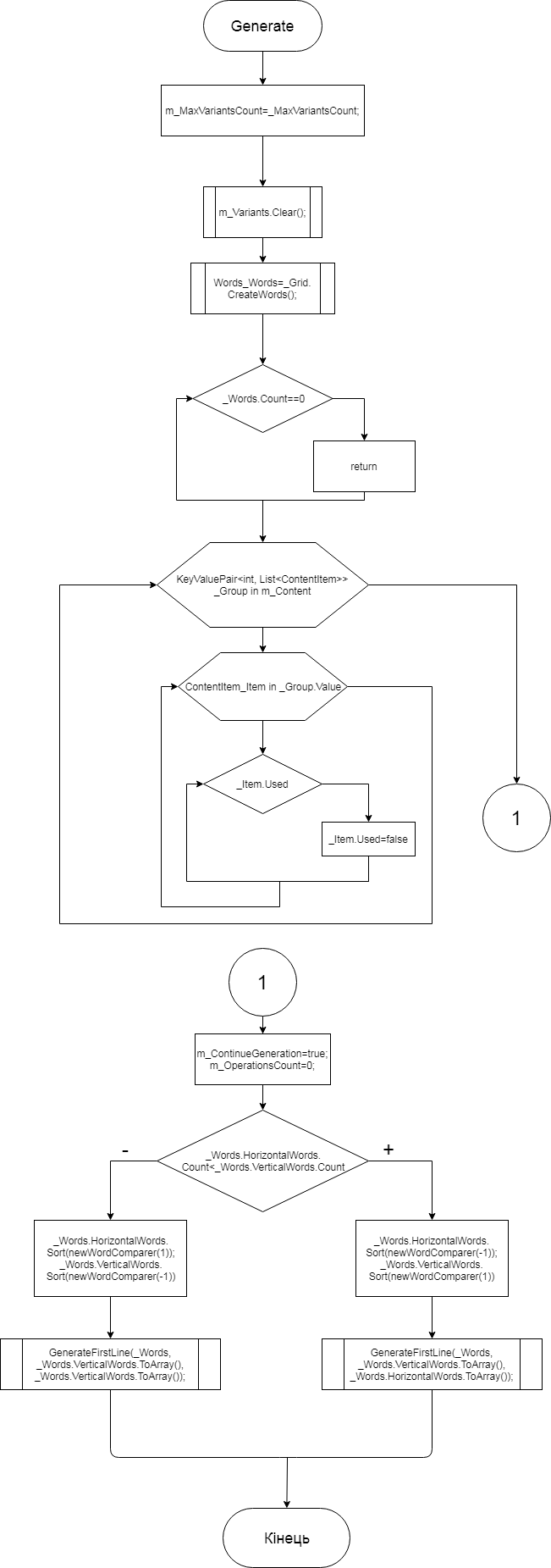


Додаток Б. Код основного алгоритму програми

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Text.RegularExpressions; |
|  |  |
|  | namespace CrossWordCraft |
|  | { |
|  | namespace Model |
|  | { |
|  | //Класс "Значение слова". |
|  | public class ContentItem |
|  | { |
|  | //Признак, что значение уже использовано в текущем варианте заполнения кроссворда |
|  | public bool Used { get; set; } |
|  | //Конструкторы |
|  | private ContentItem() { Used = false; } |
|  | public ContentItem(string value) : this() { Value = value; } |
|  |  |
|  | private string m\_Value = string.Empty; |
|  | //Значение |
|  | public string Value |
|  | { |
|  | get {return m\_Value; } |
|  | set |
|  | { |
|  | m\_Value = value; |
|  |  |
|  | if (m\_CharArray != null) |
|  | m\_CharArray = null; |
|  | } |
|  | } |
|  | private char[] m\_CharArray = null; |
|  | //Массив символов |
|  | public char[] Chars |
|  | { |
|  | get |
|  | { |
|  | if (m\_CharArray == null) |
|  | m\_CharArray = Value.ToCharArray(); |
|  |  |
|  | return m\_CharArray; |
|  | } |
|  | } |
|  | //Символ |
|  | public char this[int index] { get { return this.Chars[index]; }} |
|  | } |
|  |  |
|  | //Класс "Исходные данные для генерации кроссворда" |
|  | public class Content |
|  | { |
|  | //Словарь с разделами. Каждый раздел содержит слова одной длины |
|  | Dictionary<int, List<ContentItem>> m\_Content = null; |
|  | public Dictionary<int, List<ContentItem>> Groups { get { return m\_Content; } } |
|  |  |
|  | //Коллекция вариантов заполнения кроссворда |
|  | List<Words> m\_Variants = null; |
|  | public List<Words> Variants { get { return m\_Variants; } } |
|  |  |
|  | //Параметры управления и свойства статуса завершенности генерации |
|  | private int m\_MaxVariantsCount = 0; |
|  | private bool m\_ContinueGeneration = false; |
|  | private long m\_OperationsCount = 0; |
|  | private int m\_ProgressValue = 0; |
|  | private int m\_ProgressMaxValue = 0; |
|  | private int m\_CurrentDepth = 0; |
|  | private int m\_MaxDepth = 0; |
|  | private Word m\_CurrentWord; |
|  | private ContentItem m\_CurrentItem; |
|  |  |
|  |  |
|  | public long OperationsCount { get { return m\_OperationsCount; } } |
|  | public int ProgressValue { get { return m\_ProgressValue; } } |
|  | public int ProgressMaxValue { get { return m\_ProgressMaxValue; } } |
|  | public int CurrentDepth { get { return m\_CurrentDepth; } } |
|  | public int MaxDepth { get { return m\_MaxDepth; } } |
|  | public string Operation |
|  | { |
|  | get |
|  | { |
|  | if (m\_CurrentWord == null || m\_CurrentItem == null) |
|  | return string.Empty; |
|  |  |
|  | return string.Format("{0} ? {1}", m\_CurrentWord.Value, m\_CurrentItem.Value); |
|  | } |
|  | } |
|  |  |
|  | //Конструктор |
|  | public Content() { m\_Content = new Dictionary<int, List<ContentItem>>(); m\_Variants = new List<Words>(); } |
|  |  |
|  | //Очистка |
|  | public void Clear() |
|  | { |
|  | m\_Content.Clear(); |
|  | m\_Variants.Clear(); |
|  | } |
|  |  |
|  | //Отримуємо розділ словника розділу зі значенням певної довжиниы |
|  | public List<ContentItem> GetContent(int \_length) |
|  | { |
|  | if (!m\_Content.ContainsKey(\_length)) |
|  | return null; |
|  |  |
|  | return m\_Content[\_length]; |
|  | } |
|  |  |
|  | //Формування словника по текстовому рядку |
|  | public void Append(string \_Text) |
|  | { |
|  | Regex \_regex = new Regex("(?i)[а-я]{2,}"); //Мінімальна довжина слова – 2символа |
|  | MatchCollection \_textMatches = \_regex.Matches(\_Text); |
|  |  |
|  | foreach (Match \_wordMatch in \_textMatches) |
|  | { |
|  | string \_word = \_wordMatch.Value.ToUpper(); |
|  | List<ContentItem> \_wordsl = null; |
|  |  |
|  | if (m\_Content.ContainsKey(\_word.Length)) |
|  | \_wordsl = m\_Content[\_word.Length]; |
|  | else |
|  | { |
|  | \_wordsl = new List<ContentItem>(); |
|  | m\_Content.Add(\_word.Length, \_wordsl); |
|  | } |
|  |  |
|  | bool \_Exists = false; |
|  | IEnumerator<ContentItem> \_items = \_wordsl.GetEnumerator(); |
|  | \_items.Reset(); |
|  | while (!\_Exists && \_items.MoveNext()) |
|  | { |
|  | ContentItem \_item = \_items.Current; |
|  | if (\_item.Value == \_word) |
|  | \_Exists = true; |
|  | } |
|  |  |
|  | if (!\_Exists) |
|  | \_wordsl.Add(new ContentItem(\_word)); |
|  | } |
|  | } |
|  |  |
|  | //Зупинка генерації |
|  | public void StopGeneration() |
|  | { |
|  | m\_ContinueGeneration = false; |
|  | } |
|  |  |
|  | //Запуск генерації по сітці |
|  | public void Generate(Grid \_Grid) |
|  | { |
|  | this.Generate(\_Grid, 0); |
|  | } |
|  |  |
|  | public void Generate(Grid \_Grid, int \_MaxVariantsCount) |
|  | { |
|  | //Максимальне число варіантів; 0 - кількість варіантів не обмежене |
|  | m\_MaxVariantsCount = \_MaxVariantsCount; |
|  |  |
|  | //Очистка коллекції варіантів |
|  | m\_Variants.Clear(); |
|  |  |
|  | //Формування слів по сітці |
|  | Words \_Words = \_Grid.CreateWords(); |
|  | if (\_Words.Count == 0) |
|  | return; |
|  |  |
|  | //Установка для слів статуса 'не використано' |
|  | foreach (KeyValuePair<int, List<ContentItem>> \_Group in m\_Content) |
|  | foreach (ContentItem \_Item in \_Group.Value) |
|  | if (\_Item.Used) |
|  | \_Item.Used = false; |
|  |  |
|  | //Запуск процеса генерації: |
|  | m\_ContinueGeneration = true; |
|  | m\_OperationsCount = 0; |
|  |  |
|  | //Починаємо генерацію з переважаючого напрямку. |
|  | if (\_Words.HorizontalWords.Count < \_Words.VerticalWords.Count) |
|  | { |
|  | // Сортування слів по горизонталі у спадаючому порядку |
|  | \_Words.HorizontalWords.Sort(new WordComparer(-1)); |
|  | // Сортування слів по вертикаллі у зростаючому порядку |
|  | \_Words.VerticalWords.Sort(new WordComparer(1)); |
|  | //Запуск генерації |
|  | GenerateFirstLine(\_Words, \_Words.VerticalWords.ToArray(), \_Words.HorizontalWords.ToArray()); |
|  | } |
|  | else |
|  | { |
|  | // Сортування слів по горизонталі у зростаючому порядку |
|  | \_Words.HorizontalWords.Sort(new WordComparer(1)); |
|  | // Сортування слів по вертикаллі у спадаючому порядку |
|  | \_Words.VerticalWords.Sort(new WordComparer(-1)); |
|  | //Запуск генерації |
|  | GenerateFirstLine(\_Words, \_Words.HorizontalWords.ToArray(), \_Words.VerticalWords.ToArray()); |
|  | } |
|  | } |
|  |  |
|  | // Крок 1. Розстановка слів в першому напрямку |
|  | private void GenerateFirstLine(Words \_Words, Word[] \_FirstLineWords, Word[] \_SecondLineWords) |
|  | { |
|  | int \_BackToIndex = -1; |
|  | GenerateFirstLine(\_Words, \_FirstLineWords, \_SecondLineWords, 0, ref \_BackToIndex); |
|  | } |
|  |  |
|  | private void GenerateFirstLine(Words \_Words, Word[] \_FirstLineWords, Word[] \_SecondLineWords, int \_Index, ref int \_BackToIndex) |
|  | { |
|  | if (!m\_ContinueGeneration) |
|  | return; |
|  |  |
|  | Word \_Word = \_FirstLineWords[\_Index]; |
|  | \_Word.Index = \_Index; |
|  |  |
|  | List<ContentItem> \_Contents = this.GetContent(\_Word.Length); |
|  |  |
|  | if (\_Contents == null) |
|  | return; |
|  |  |
|  | if (\_Index == 0) |
|  | { |
|  | m\_ProgressValue = 0; |
|  | m\_ProgressMaxValue = \_Contents.Count; |
|  | } |
|  |  |
|  | foreach (ContentItem \_item in \_Contents) |
|  | { |
|  | m\_OperationsCount++; |
|  | m\_CurrentWord = \_Word; m\_CurrentItem = \_item; |
|  |  |
|  | if (\_Index == 0) |
|  | m\_ProgressValue++; |
|  |  |
|  | if (!\_item.Used) |
|  | { |
|  | \_Words.AssignValue(\_Word, \_item.Value, true); |
|  | \_item.Used = true; |
|  |  |
|  | \_BackToIndex = -1; |
|  |  |
|  | if (\_Index == \_FirstLineWords.Length - 1) |
|  | GenerateSecondLine(\_Words, \_SecondLineWords, 0, 0, ref \_BackToIndex); |
|  | else |
|  | GenerateFirstLine(\_Words, \_FirstLineWords, \_SecondLineWords, \_Index + 1, ref \_BackToIndex); |
|  |  |
|  | \_Words.ClearValue(\_Word, false); |
|  | \_item.Used = false; |
|  |  |
|  | if (\_BackToIndex != -1 && \_BackToIndex < \_Index) |
|  | return; |
|  | else |
|  | \_BackToIndex = -1; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | //Крок 2. Розстановка слів у другому напрямі з урахуванням перетинів |
|  | private void GenerateSecondLine(Words \_Words, Word[] \_SecondLineWords, int \_Index, int \_PrevWordLength, ref int \_BackToFirstLineIndex) |
|  | { |
|  | if (!m\_ContinueGeneration) |
|  | return; |
|  |  |
|  | if (\_SecondLineWords.Length == 0) |
|  | { |
|  | if (m\_MaxVariantsCount == 0 || m\_Variants.Count < m\_MaxVariantsCount) |
|  | m\_Variants.Add(\_Words.Copy()); |
|  |  |
|  | if (m\_Variants.Count == m\_MaxVariantsCount) |
|  | StopGeneration(); |
|  |  |
|  | return; |
|  | } |
|  |  |
|  | Word \_Word = \_SecondLineWords[\_Index]; |
|  | List<ContentItem> \_Contents = this.GetContent(\_Word.Length); |
|  |  |
|  | if (\_Contents == null) |
|  | return; |
|  |  |
|  | if (\_Index == 0) |
|  | { |
|  | m\_CurrentDepth = 0; |
|  | m\_MaxDepth = \_SecondLineWords.Length; |
|  | } |
|  |  |
|  | m\_CurrentDepth = \_Index + 1; |
|  |  |
|  | int \_MaxFirstNonEqualIndex = -1; |
|  | foreach (ContentItem \_item in \_Contents) |
|  | { |
|  | m\_OperationsCount++; |
|  | m\_CurrentWord = \_Word; m\_CurrentItem = \_item; |
|  |  |
|  | if (!\_item.Used) |
|  | { |
|  | int \_FirstNonEqualIndex = -1; |
|  | if (\_Words.CanAssign(\_Word, \_item, ref \_FirstNonEqualIndex)) |
|  | { |
|  | \_Words.AssignValue(\_Word, \_item.Value, false); |
|  | \_item.Used = true; |
|  |  |
|  | \_BackToFirstLineIndex = -1; |
|  |  |
|  | if (\_Index == \_SecondLineWords.Length - 1) |
|  | { |
|  | if (m\_MaxVariantsCount == 0 || m\_Variants.Count < m\_MaxVariantsCount) |
|  | m\_Variants.Add(\_Words.Copy()); |
|  |  |
|  | if (m\_Variants.Count == m\_MaxVariantsCount) |
|  | StopGeneration(); |
|  | } |
|  | else |
|  | GenerateSecondLine(\_Words, \_SecondLineWords, \_Index + 1, \_Word.Length, ref \_BackToFirstLineIndex); |
|  |  |
|  | \_Words.ClearValue(\_Word, true); |
|  | \_item.Used = false; |
|  |  |
|  | if (\_BackToFirstLineIndex != -1) |
|  | return; |
|  | } |
|  | else |
|  | { |
|  | if (\_MaxFirstNonEqualIndex == -1 || \_MaxFirstNonEqualIndex < \_FirstNonEqualIndex) |
|  | \_MaxFirstNonEqualIndex = \_FirstNonEqualIndex; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | if (\_MaxFirstNonEqualIndex != -1 && \_PrevWordLength != \_Word.Length) |
|  | \_BackToFirstLineIndex = \_Words.GetCrossedIndex(\_Word, \_MaxFirstNonEqualIndex); |
|  | } |
|  | } |
|  | } |
|  | } |

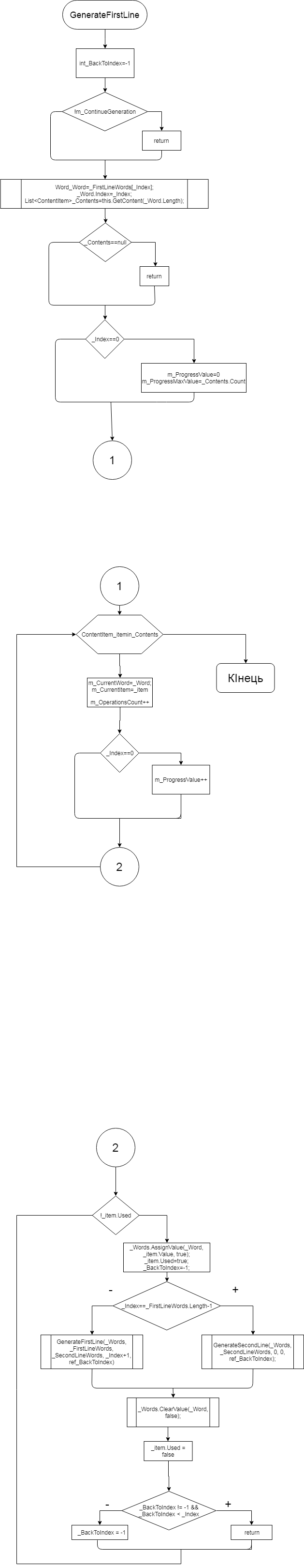
Додаток В. Блок-схема основного алгоритму

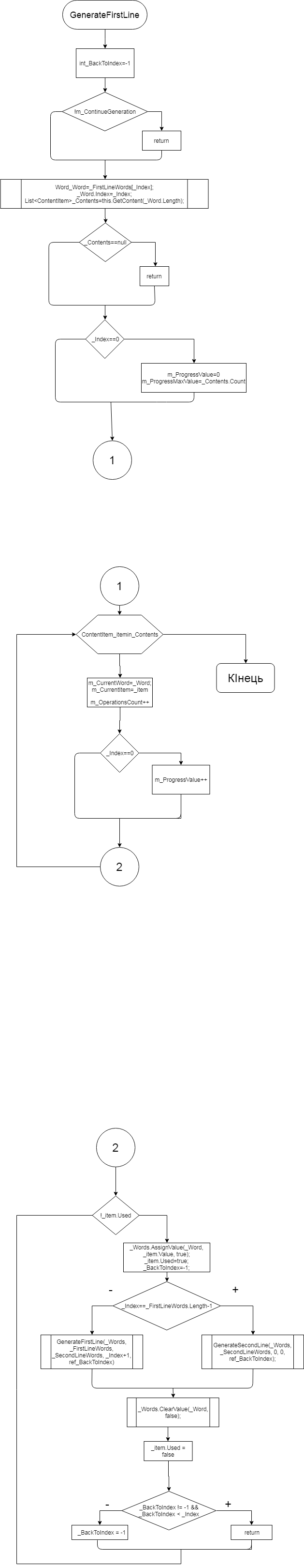
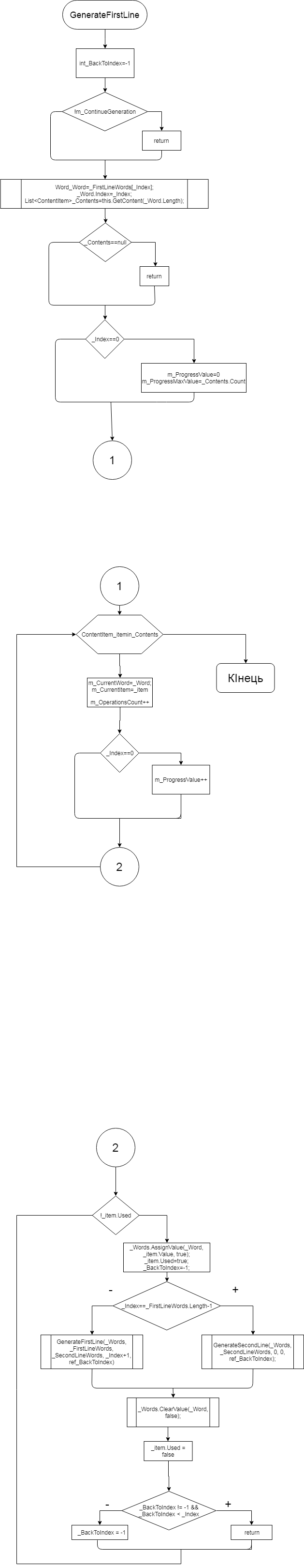




Допоміжні Методи

1)





2)

