**ХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХ**

ХХХХХХХХХХ

(повна назва кафедри, циклової комісії)

**КУРСОВИЙ ПРОЕКТ**

з дисципліни «Операційні системи»

на тему: «**Розробка програми автоматизації визначення часу виявлення порушення технологічного процесу на виробництві (ГВСС)**»

Студента 3 курсу групи ХХХХ

напряму підготовки Комп'ютерні науки

спеціальності: ХХХХХХХХ

ХХХХХХХХ

(прізвище та ініціали)

Керівник ХХХХХХ

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала

Кількість балів: \_\_\_\_\_\_\_ Оцінка: ECTS

Члени комісії ХХХХХХ

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
**ХХХХХХХХХХ**  
(повне найменування вищого навчального закладу)

Інститут, факультет ХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХХ ХХ

Кафедра ХХХХХ

Ступінь вищої освіти (освітній ступінь) бакалавр

Спеціальність ХХХХХХХХХ

Курс 3 Група ХХХХ Семестр V

Завдання

на курсовий проект студентові

ХХХХХХХХХХ   
 (прізвище, ім’я, по батькові)

1. Тема проекту: розробка програми автоматичного визначення часу виявлення порушення на виробництві

2. Термін здачі студентами закінченого проекту: ХХХХХХХ

3. Вихідні дані: початкові дані: структура порушень і їх ознак, заданих у вигляді графа; вихідні дані: час виявлення порушення оператором.

4. Зміст розрахунково-пояснювальної записки:

Вступ

1 Вибір і обгрунтування проектних рішень

2 Розробка структури програмного забезпечення

3 Розробка програми

4 Керівництво програміста

5 Керівництво оператора \_

Висновки

Перелік посилань

Додаток А. Технічне завдання

Додаток Б. Текст програми

5. Дата видачі завдання: ХХХХХХХХ

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів курсового  проекту (роботи) | Термін виконання етапів  проекту (роботи) | Примітка |
| 1. | Отримання та аналіз індивідуального завдання. | 1 - 2 тиждень |  |
| 2. | Аналіз програмних засобів, що будуть використані в роботі. | 3 - 4 тиждень |  |
| 3. | Розробка логічної моделі системи. Заповнення системи даними | 4 - 5 тиждень |  |
| 4. | Вивчення можливостей програмної реалізації інтерфейсу користувача. | 5 - 6 тиждень |  |
| 5. | Проектування додатків системи (звіти, макроси, запити) | 6 - 8 тиждень |  |
| 6. | Оформлення відповідних пунктів пояснювальної записки | 9 тиждень |  |
| 7. | Захист курсової роботи. | 10 тиждень |  |

Студент ХХХХХХ

Керівник ХХХХХХХ

«\_\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 р.

реферат

Об’єкт проектування – програма для автоматичного визначення часу виявлення порушення на виробництві оператором (ГВСС).

Мета курсового проекту – створення програми, що дозволяє створити модель структури порушень на деякому виробництві у вигляді графу, а потім отримати час виявлення порушення, який розраховується автоматично виходячи з заданої структури.

Програмне забезпечення було розроблено з використанням мови програмування С#, програмної платформи .NET, фреймворку Windows Presentation Foundation (WPF) для розробки Desktop-застосунків.

ГВСС, СКРІНШОТ, МОДЕЛЬ, ГРАФ, МОДЕЛЮВАННЯ, ГРАФІЧНЕ МОДЕЛЮВАННЯ, СТВОРЕННЯ СТРУКТУРИ ГВСС, ДОДАВАННЯ ОЗНАК, ВИДАЛЕННЯ ОЗНАК, РЕДАГУВАННЯ ОЗНАК, ЗБЕРЕЖЕННЯ, ЗАВАНТАЖЕННЯ, DESKTOP, DESKTOP-ЗАСТОСУНОК, C#, WINDOWS.

**ЗМІСТ**

[Завдання 2](#_Toc89289422)

[Календарний план 3](#_Toc89289423)

[Реферат 4](#_Toc89289424)

[Перелік умовних позначень і скорочень 7](#_Toc89289425)

[Вступ 8](#_Toc89289426)

[1. Вибір і обгрунтування проектних рішень 9](#_Toc89289427)

[1.1 Аналiз технiчного завдання 9](#_Toc89289428)

[1.2 Огляд алгоритмів і методів 10](#_Toc89289429)

[1.2.1 Алгоритм пошуку в глибину 10](#_Toc89289430)

[1.2.2 Метод розрахунку часу виявлення порушення оператором 11](#_Toc89289431)

[2. Розробка структури програмного забезпечення 24](#_Toc89289432)

[2.1 Визначення компонентів системи 24](#_Toc89289433)

[2.2 Аналіз технічних і програмних засобів 28](#_Toc89289434)

[2.3 Функціональна структура програми 31](#_Toc89289435)

[3. Розробка програми 34](#_Toc89289436)

[3.1 Розробка Desktop-застосунку 34](#_Toc89289437)

[3.1.1 Визначення класів Desktop-застосунку 34](#_Toc89289438)

[3.1.2 Розробка графічного інтерфейсу користувача 39](#_Toc89289439)

[4. Керівництво програміста 50](#_Toc89289440)

[4.1 Призначення та умови застосування програми 50](#_Toc89289441)

[4.2 Характеристики програми 51](#_Toc89289442)

[4.3 Початкові та вихідні дані 51](#_Toc89289443)

[4.4 Модифікація програми 52](#_Toc89289444)

[5 Керівництво оператора 55](#_Toc89289445)

[5.1 Призначення та умови виконання програми 55](#_Toc89289446)

[5.2 Виконання програми 55](#_Toc89289447)

[5.3 Повідомлення оператору 67](#_Toc89289448)

[Висновки 68](#_Toc89289449)

[Перелік посилань 69](#_Toc89289450)

[Додаток А. Технічне завдання 71](#_Toc89289451)

[А.1 Підстава для розробки 71](#_Toc89289452)

[А.2 Призначення розробки 71](#_Toc89289453)

[А.3 Вимог до програми або програмного продукту 71](#_Toc89289454)

[А.3.1 Функціональні вимоги 71](#_Toc89289455)

[А.3.2 Вимоги до надійності 72](#_Toc89289456)

[А.3.3 Умови експлуатації 73](#_Toc89289457)

[А.3.3 Вимоги до складу та параметрів технічних засобів 73](#_Toc89289458)

[А.3.4 Вимоги до інформаційної та програмної сумісності 73](#_Toc89289459)

[А.3.5 Вимоги до програмної документації 74](#_Toc89289460)

[А.4 Стадії та етапи розробки 74](#_Toc89289461)

[А.5 Порядок приймання і тестування 74](#_Toc89289462)

[Додаток Б. Текст програми 75](#_Toc89289463)

[Б.1 Текст Desktop-застосунку 75](#_Toc89289464)

перелік умовних позначень і скорочень

ОС – операційна система

ПЗ – програмне забезпечення

МП – мова програмування

EDA – Event Driven Architecture

ТЗ – технічне завдання

ООП – об’єктно орієнтовне програмування

вступ

Під час виробництва, як і будь якого технічного процесу, виникають порушення в роботі обладнання і бувають випадки, коли такі порушення можуть спричинити дуже великі матеріальні, а у найгіршому випадку – людські втрати. Тому проектувальникам необхідно заздалегіть знати час виявлення порушення оператором, бо цей час цілком залежить від тренованості оператора, кількості і видів органів запиту параметрів, а також їх розміщення.

Ця величина показує як швидко людина, яка стежить за работою обладнання, зможе визначити причину проблеми у його роботі і почне прикладати зусилля для відновлення роботи системи у штатному режимі.

Подібне програмне забезпечення, найчастіше, надає можливості графічного редагування системи порушень і їх ознак, а також автоматичного визначення часу виявлення порушення.

Отже, створення такого програмного забезпечення є актуальним на данний момент.

1. вибір і обгрунтування проектних рішень

1.1 Аналiз технiчного завдання

Програма заснована на науковій праці А. І. Галактіонова «Представление информации оператору (исследования деятельности человека-оператора производственных процессов)» [1].

Відповідно до висунутих у технічному завдані вимог до програмного забезпечення, розроблюваний продукт має надавати користувачеві графічно створювати структуру ГВСС. Таке рішення було прийняте у зв’язку з тим, що складно взаємодіяти з програмою, де вся інформація представлена лише у текстовому вигляді. Люди краще сприймають інформацію у графічній формі – це визначення дійсно складно заперечити. Тому створення і редагування структури ГВСС повинні виконуватися у наглядній для користувачів формі. Тому при виборі мови програмування і фреймворку перевагу отримували такі технології, що дають можливість швидко і зручно створювати графічний інтерфейс і з’єднувати його з безпосереднім рушієм програми. Більш детально питання реалізації системи, що дозволяє графічно моделювати структуру ГВСС буде розглянуто на етапі проектування під час вибору технічних і програмних засобів реалізації (зокрема, мови програмування, бібліотек і фреймворків).

Хоча графічне відображення і складає найбільшу і найскладнішу частину розробки програми, але все ж таки вона повинна робити деякі обрахунки. Результат програми – приблизний час виявлення порушення на виробництві оператором. Ця величина розраховується відповідно до заданої у графічному вигляді структури ГВСС і за формулами, які визначив А. І. Галактіонов у своїй роботі.

1.2 Огляд алгоритмів і методів

1.2.1 Алгоритм пошуку в глибину

Так як структура ГВСС являє собою граф, тому без алгоритмів обходу графу (graph traversal algorithms) було просто неможливо реалізувати дану програму. Алгоритм обходу у глиб [2] – є дуже швидким і надійним алгоритмом. Він має дві реалізації: рекурсивна (рис 1.1) і за допомогою циклів (рис 1.2).

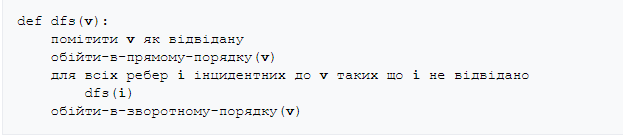


Рисунок 1.1 – Алгоритм обходу в глибину (версія з рекурсією)

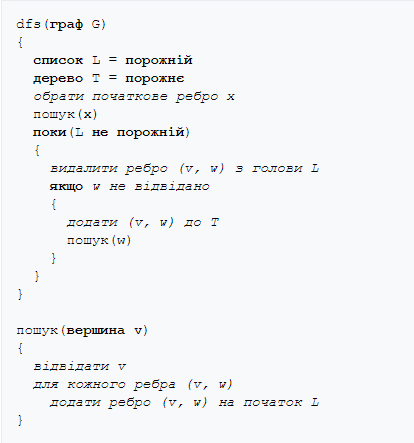


Рисунок 1.2 – Алгоритм обходу в глибину (реалізація за допомогою циклів)

У своїй програмі я використав версію з рекурсією. Без алгоритму обходу графу я не зміг би реалізувати більшість обчислень. Тому цей алгоритм є дуже важливим у моїй роботі і я його описав саме у цьому розділі.

1.2.2 Метод розрахунку часу виявлення порушення оператором

Методами розрахунку часу виявлення порушення є методи приведені в науковій праці А. І. Галактіонова [1]. Він за допомогою досліджень і експериментів визначив закони, яким підкоряється такий час.

Закони описані за допомогою математичного апарату і являють собою формули. Ці формули і були використані при розробці програми. Далі вони будуть розглянуті більш детально.

Згідно з науковою працею А. І. Галактіонова час виявлення порушення не є самостійною величиною, а складається з ряду інших розраховуваних значень, а саме:

1. Початковий час – величина, яка прямопропорційно залежить від інформаційної складності події (відповідно до теорії інформації представляє собою кількіть «інформації», яка міститься в певній події, що являє собою порушення і ланцюг його ознак) і від рівня тренованості оператора;
2. Вплив методу контролю параметрів (автоматичний чи запит параметрів). Визначається в секундах і являє собою час, який витратить оператор на визначення значення параметрів залежно від методу контролю цих параметрів;
3. Вплив кількості ірелевантної (неактуальної, непотрібної інформації). У реальних системах поряд з «корисною» інформацією, що відноситься до певної події, присутні інші сигнали, що не відносяться до неї (ірелевантні сигнали). Визначається в секундах
4. Вплив типу засобів представлення інформації. Ця величина пов’язана з приростом часу виявлення події за рахунок необхідності зчитування показів прибору і за рахунок знаходження приборів, що позначають відхилення від норми.
5. Вплив часу знаходження серед загальної кількості однотипних вимірювальних приборів потрібного прибору. Вимірюється в секундах

Далі будуть розглянуті формули визначення цих величин і розрахунки на прикладі ГВСС з 5 подіями (рис. 1.3)

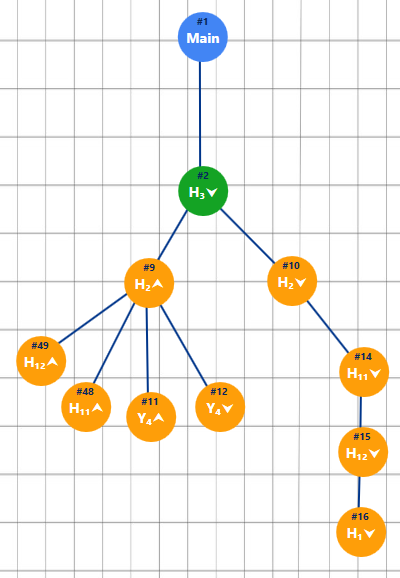


Рисунок 1.3 – Приклад ГВСС, на основі якої будуть проводитися розрахунки

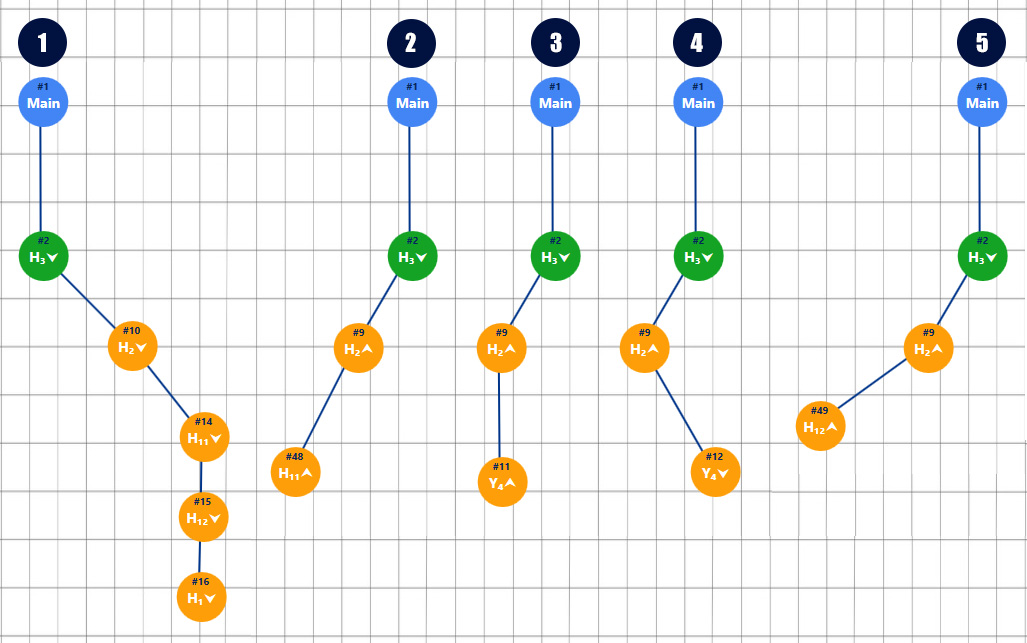
Всі 5 подій окремо зображені на рис. 1.4.

Рисунок 1.4 – Всі п’ять подій окремо для контрольної ГВСС

Буде розглядатися подія під номером 3 (рис. 1.4), що складається з 4 вершин: 1 -> 2 -> 9 -> 11. Налаштування поточної ГВСС наведені нижче (рис. 1.5)



Рисунок 1.5 – Приклад ГВСС, на основі якої будуть проводитися розрахунки

**Визначення початкового часу**

Визначення початкового часу вимірюється за формалою на рис 1.6



Рисунок 1.6 – Формула визначення початкового часу

Коефіцієнти *a* і *b* визначаються в залежності від рівня тренованості оператора:

* I рівень: відповідає випадку, коли оператор не визначив алгоритму, співпадаючого з структурною схемою ГВСС, але він здатен вирішувати задачі по виявленню складних події. На цьому рівні працюють оператори тих об’єктів, де ситуація постійно змінюється (у програмі даний рівень не визначений).
* II рівень: відповідає випадку, при якому у ході навчання оператор визначає для себе алгоритм, який співпадає зі структурной схемой подій, але він не знає ймовірностей параметрів, які контролюються. Цей випадк є характерним для тих об’єктів, де частоти виходу за норму параметрів не постійні у часі чи коли є велика кількість параметрів, можливих порушен, а також їх причин і оператор не здатен їх запам’ятати і впорядкувати.
* III рівень: це такий рівень тренованості оператора, при якому оператор визначає для себе алгоритм і познає закон розподілу ймовірностей основних параметрів, але він не знає ймовірностей виникнення допоміжних параметрів.
* IV рівень: рівень, при якому оператор визначив для себе алгоритм, співпадаючий зі структурною схемою, він знає ймовірності виникнення порушень і ймовірності появи частини подій (причин порушень) або ймовірностей виходу з норми частини допоміжних параметрів. На такому рівні працюють оператори транспорту, невеликих виробництв, або об’єктів з невеликою кількістю подій середньої складності.
* V рівень: це рівень роботи добре навчаних операторів, визначивших алгоритм вирішення задач, тих, що пізнали всі ймовірності виходу з норми основних і допоміжних параметрів і використовуючих у якості допоміжного інструменту «образами» деяких подій.
* VI рівень: відповідає найвищому рівню навчаності, коли оператор у більшості випадків використовує образи подій. Цей рівень є характерним для невеликих об’єктів з невеликим числом подій (у програмі даний рівень не визначений).

На рис 1.7 показані коефіцієнти для 2, 3, 4 і 5 рівнів тренованості операторів.

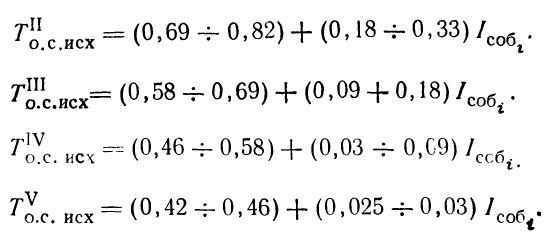


Рисунок 1.7 – Коефіцієнти для різних рівнів тренованості операторів

Параметер *I*, або іншими словами інформаційна складність події визначається по формулам на рис 1.8. Ці формули визначені у математичній теорії інформації [3].

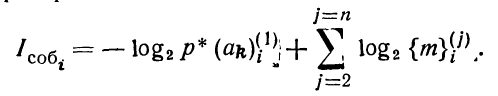


Рисунок 1.8 – Формула визначення інформаційної складності події

Як ми бачимо інформаційна складність залежить від кількості технологічних ліній об’єкту і кількості допоміжних ознак порушень, а також загальної кількості ознак на певному рівні піддерева ознак.

Розрахунок для контрольного прикладу наведено на рис 1.9

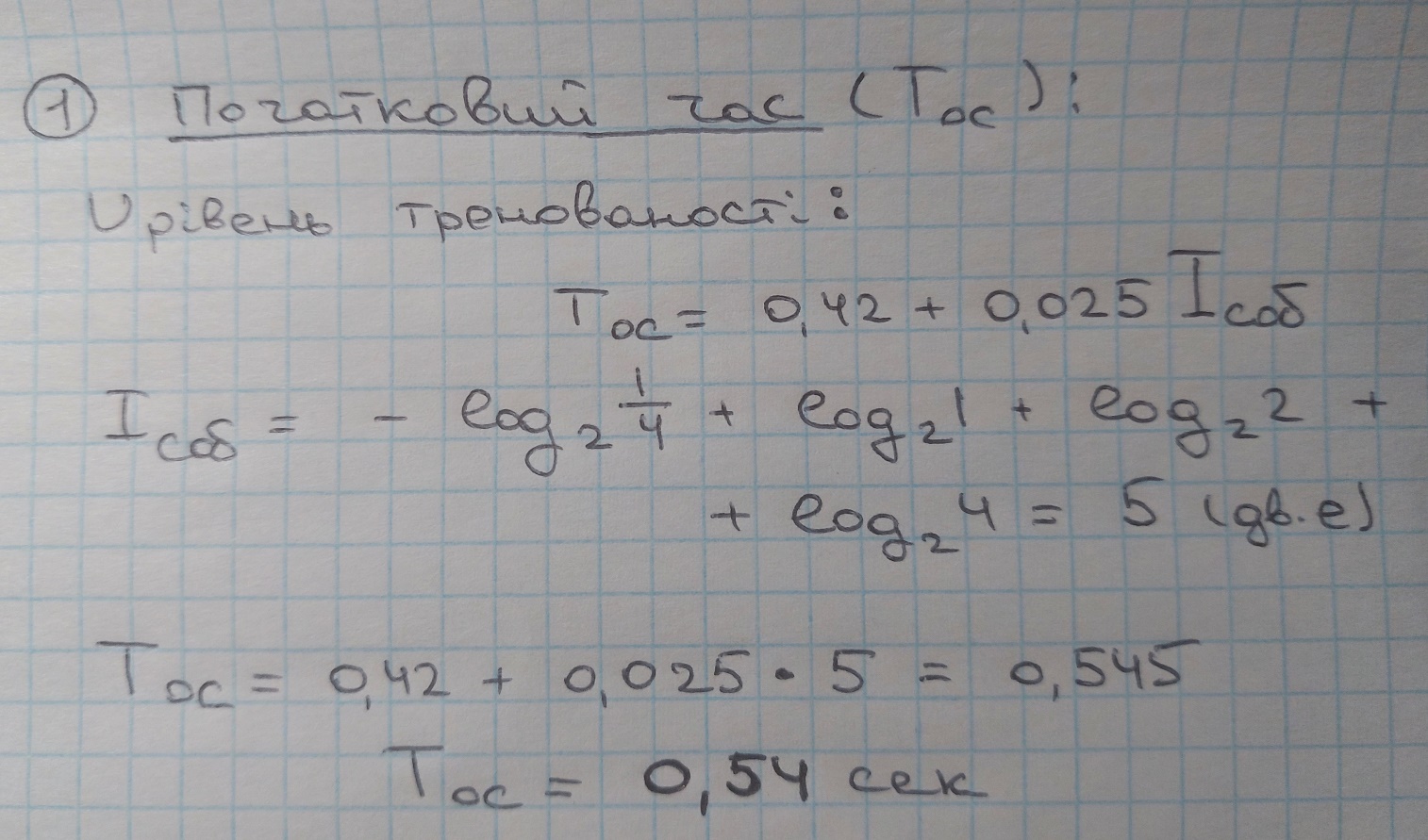


Рисунок 1.9 – Розрахунок початкового часу для контрольного прикладу

Коефіцієнти 0.42 і 0.025 приведені в розрахунках на рис. 1.9, тому що саме ці коефіцієнти визначені Галактіоновим [8] для розрахунку початкового часу для 5-го рівня тренованості оператора (рис. 1.7)

**Визначення впливу методу контролю параметрів**

Цей параметер визначається в секундах і являє собою час, який витратить оператор на визначення значення параметрів залежно від методу контролю цих параметрів. Ця величини не дорівнює 0, якщо в події присутні параметри, для отримання значень яких оператор повинен зробити запит. Обчислюється по формулі на рис. 1.10

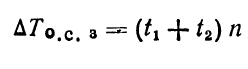
****

Рисунок 1.10 – Формула визначення впливу методу контролю параметрів (сек.)

Параметер позначає час одного запиту параметру, в залежності від способу розміщення органу запиту і його типу (таблична величина).

Параметер позначає чвс зчитування значення параметру з прибору (таблична величина). Для сигналізаторів цей .

Параметер *n* позначає кількість однакових приборів, для отримання значення яких необхідно зробити запит, у події.

Розрахунок для контрольного прикладу наведено на рис 1.11



Рисунок 1.11 – Розрахунок впливу методу контролю параметрів для контрольного прикладу

**Визначення впливу кількості ірелевантної інформації**

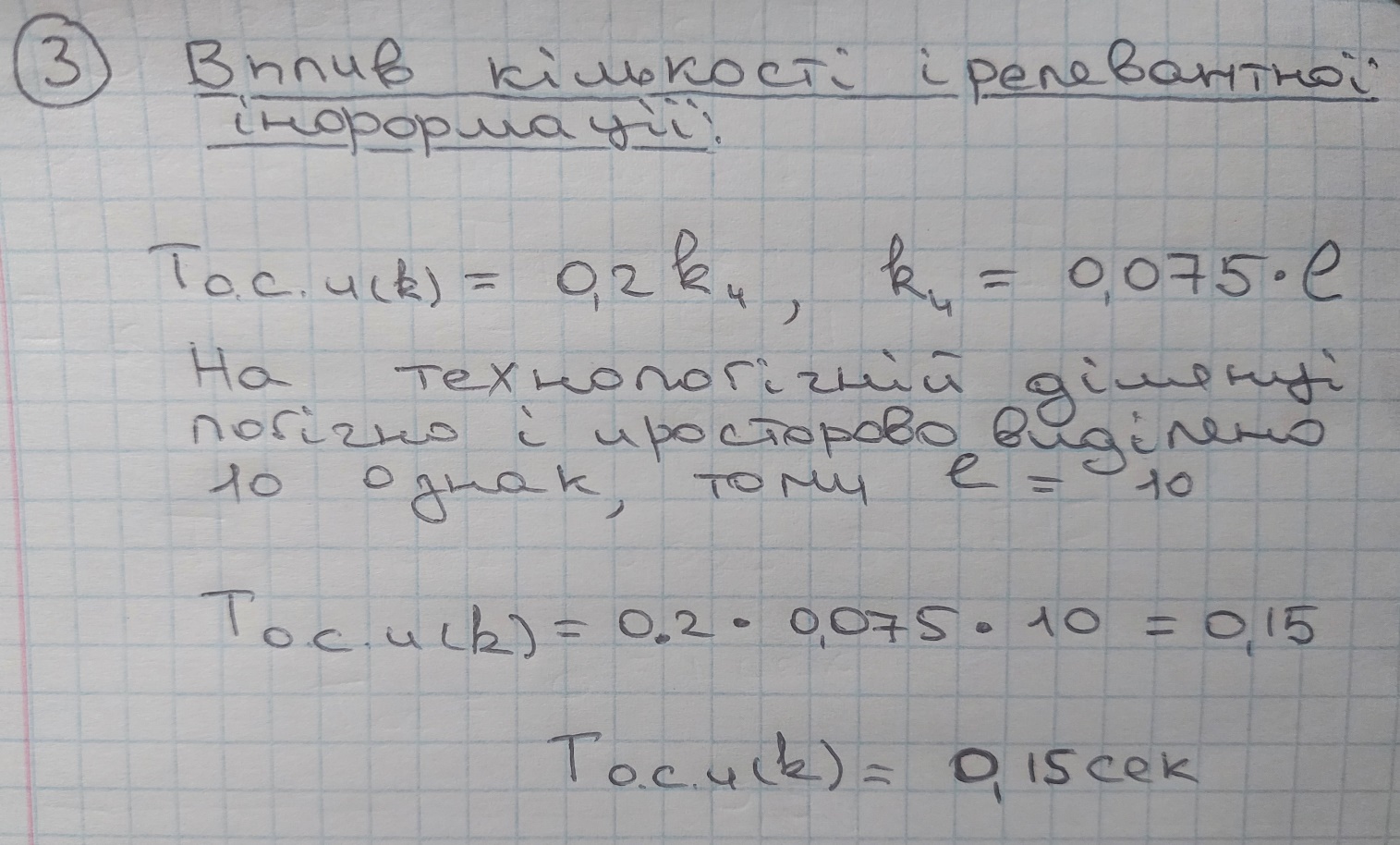
У реальних системах поряд з «корисною» інформацією, що відноситься до певної події, присутні інші сигнали, що не відносяться до неї (ірелевантні сигнали). Вплив кількості ірелевантних сигналів визначається по формулі на рис. 1.12. Визначається в секундах.



Рисунок 1.12 – Формула визначення впливу кількості ірелевантної інформації в події (сек.)

Параметер *l* позначає число ознак на даній ділянці об’єкту, просторово і логічно визначеному на контрольному щиті.

Розрахунок для контрольного прикладу наведено на рис 1.13

Рисунок 1.13 – Розрахунок впливу кількості ірелевантної інформації для контрольного прикладу

**Визначення впливу засобів представлення інформації**

Ця величина пов’язана з приростом часу виявлення події за рахунок необхідності зчитування показів прибору і за рахунок знаходження приборів, що позначають відхилення від норми. Визначення впливу засобів представлення інформації відбувається по формулі на рис. 1.14. Визначається в секундах.



Рисунок 1.14 – Формула визначення впливу засобів представлення інформації (сек.)

Параметер *t* позначає час зчитування значення з певного прибору (засобу представлення інформації). Всі значення цього параметру є табличними значеннями.

Параметер *n* позначає кількість приборів одного типу, що відображають ознаки даної події. Цей параметер підраховується по структурній схемі.

Розрахунок для контрольного прикладу наведено на рис 1.15

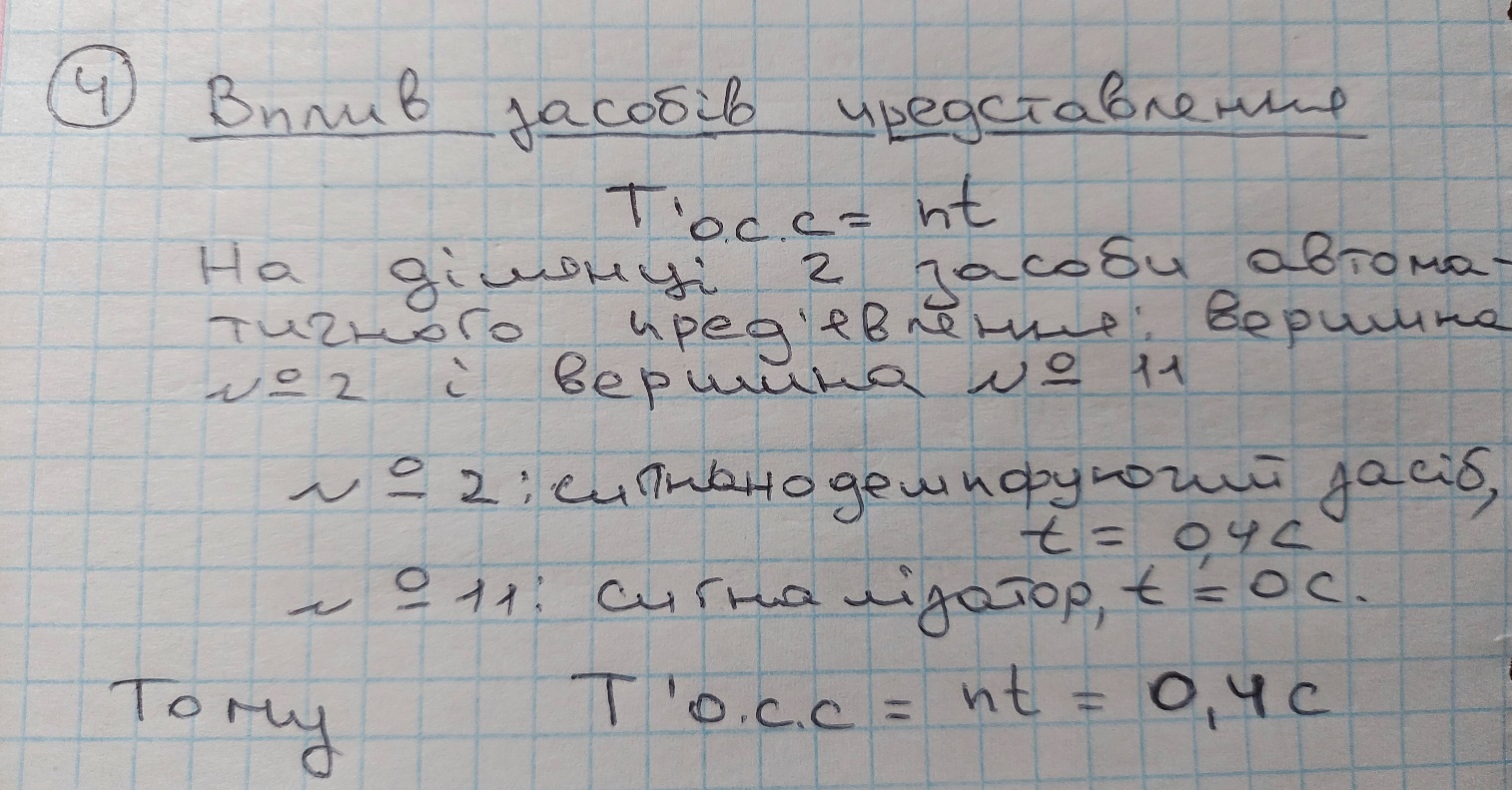


Рисунок 1.15 – Розрахунок впливу засобів представлення інформації для контрольного прикладу

**Визначення часу знаходження потрібних приборів**

У системі можуть існувати декілька однотипних пристроїв, які позначають різні параметри, що належать до різних подій. Вплив часу знаходження серед загальної кількості однотипних вимірювальних приборів потрібного прибору визначається відповідно до графіку на рисунку 1.16. Вимірюється в секундах

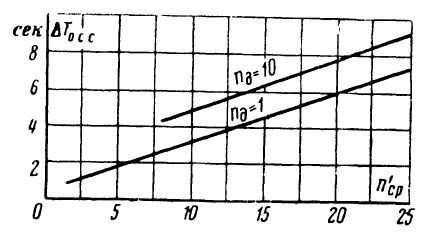


Рисунок 1.16 – Графік залежності на цифровому пристрої відхилень від норми числа пристроїв в умовній підмножині ознак

Параметер позначає кількість однакових вимірювальних приборів на мнемосхемі, які вказують відхилення від норми.

Параметер позначає кількість однотипних вимірювальних приборів у певній події.

Розрахунок для контрольного прикладу наведено на рис 1.17

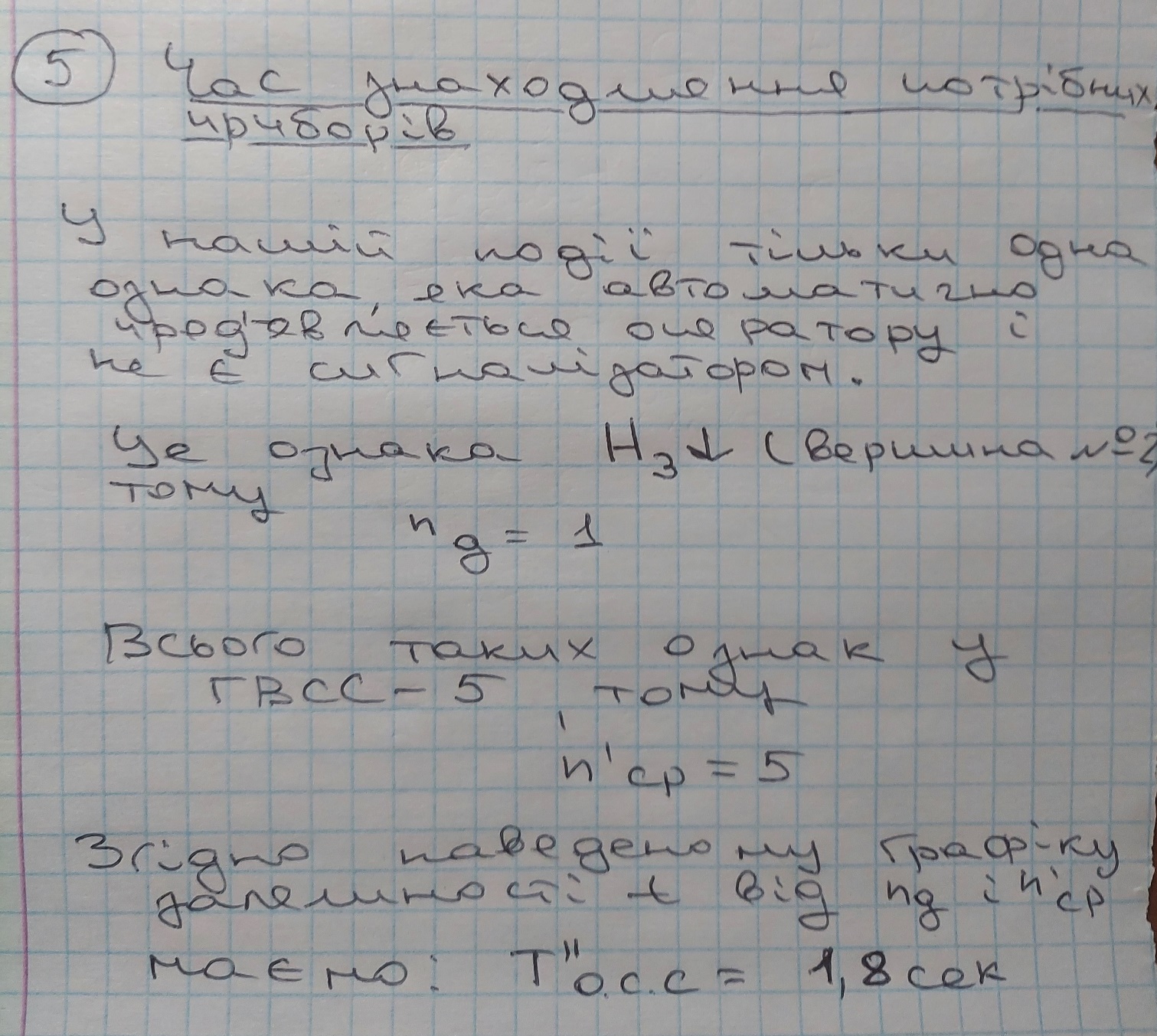


Рисунок 1.17 – Розрахунок часу знаходження потрібних приборів для контрольного прикладу

**Отримання загального часу виявлення порушення**

Час виявлення порушення – кінцевий результат програми, він є сумою всих суміжних параметрів, а саме: початковий час, час впливу методів контролю параметрів, час впливу кількості ірелевантної інформації, час впливу засобів представлення інформації, час, необхідний для знаходження потрібних приборів. Визначається по формулі на рис. 1.18.



Рисунок 1.18 – Кінцева формула визначення часу виявлення події

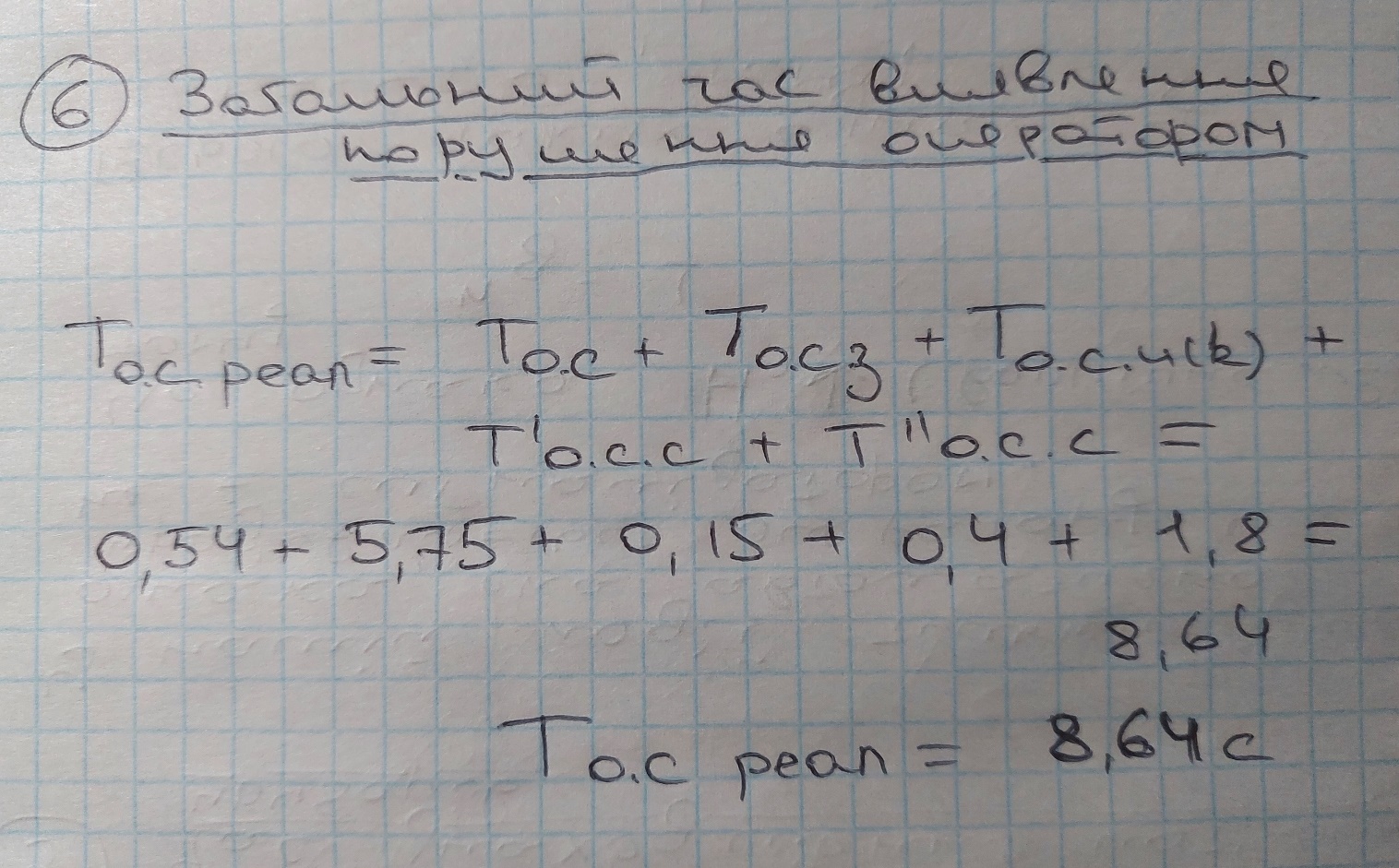
Розрахунок для контрольного прикладу наведено на рис 1.19. Програмний резульзат – рис. 1.20

Рисунок 1.19 – Визначення часу виявлення події для контрольного прикладу

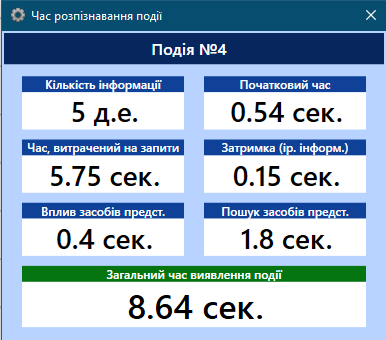


Рисунок 1.20 – Час виявлення події для контрольного прикладу (програмний розрахунок)

2. розробка структури програмного забезпечення

2.1 Визначення компонентів системи

Як вже було зазначено раніше, розроблюване програмне забезпечення являє собою Desktop застосунок, функціонал якого повністю задовольняє всі вимоги, зазначені у ТЗ. Проектуючи структуру ПЗ, можна розділити виділені модулі на ще менші частки – компоненти. Перелік проектованих модулів та компонентів наведено нижче:

* Рушій програми – об’єднує всі модулі в єдину систему.
* ГВСС рушій, представлений класом GVSS (більш детально його буде розглянуто пізніше). В його основі структура даних «граф», тобто він він складається з вершин і взаємозвязків між ними. Всі дії, які користувач може робити за допомогою цього класу – реалізовані у виді його методів.
* Програма захоплення зображення структури ГВСС – реалізує функціональність для захоплення зображення з робочої області програми і дає можливість користувачеві зберегти це зображення у форматі .png. Коли користувач вже повністю визначив структуру ГВСС у нього може виникнути необхідність збереження структури ГВСС у графічному вигляду для своїх особистих потреб (наприклад, щоб зробити звіт). Саме для цього даний модуль вбудований у програму. Зображення повністю відповідає структурі ГВСС визначеній користувачем у програмі.
* Програма збереження класів у бінарному форматі з можливістю подальшого відкриття збереженного раніше файлу у програмі
* Система графічного моделювання, яка дає можливість користувачеві створювати ГВСС у явному вигляді
* Модуль який дозволяє відкривати у програмі файл з результатами обчислень приблизного часу виявлення порушення оператором, отриманих за допомогою нейромережі, створенної на мові програмування MathLab

Нижче, на рисунку 2.1, наведено візуалізацію спроектованої системи, подану у вигляді діаграми компонентів.

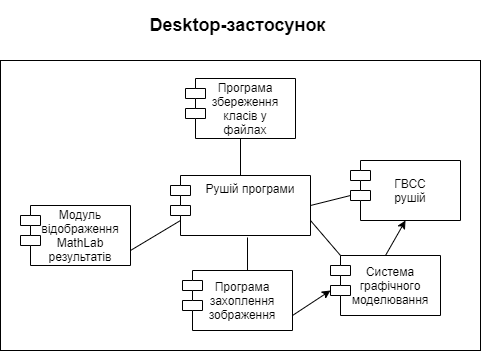


Рисунок 2.1 –Діаграма компонентів

Давайте більш детальніше розглянемо взаємозв’язки між компонентами системи. Рушій програми пов’язаний зі всіма модулями за допомогою асоціації. Це означає що він пов’язує всі виконувані модулі між собою. Як ми бачимо з діаграми компонентів «Програма захоплення зображення» залежить від «Системи графічного моделювання», тобто для отримання зображення ГВСС, вона повинна бути графічно визначена. З іншого боку «Система графічного моделювання» є самостійним модулем відповідно до «Програми захоплення зображення», але залежить від «ГВСС рушія», бо саме визначені у класі GVSS дані під час виконання програми проецуються на робочу область у графічному вигляді.

Як результат отримуємо програму, яка відповідає всім вимогам технічного завдання.

Для проектування архітектури ПЗ був обраний шаблон EDA (Event driven architecture) [4]. Він передбачає, що виконання програми визначається подіями — діями користувача (клавіатура, миша, сенсорний екран), повідомленнями інших програм і потоків.

Даний шаблон є дуже простим в реалізації: програмісту необхідно визначити обробники для необхідних подій. Зазвичай обробниками подій є методи і функції. Через таку нескладну і гнучку стуктуру шаблон EDA може бути втілений у життя за допомогою більшісті мов програмування, які підтримують ООП. Але він має достатньо вагомий недолік. Порівнюючи з такими відомими підходами до проектування як MVVM, MVP, або MVC – використання підходу EDA дуже ускладнює майбутнє доповнення програми, її тестування і підтримку. Але саме простота реалізації нівелює цей недолік і для відносно невеликих програм EDA є найкращим вибором.

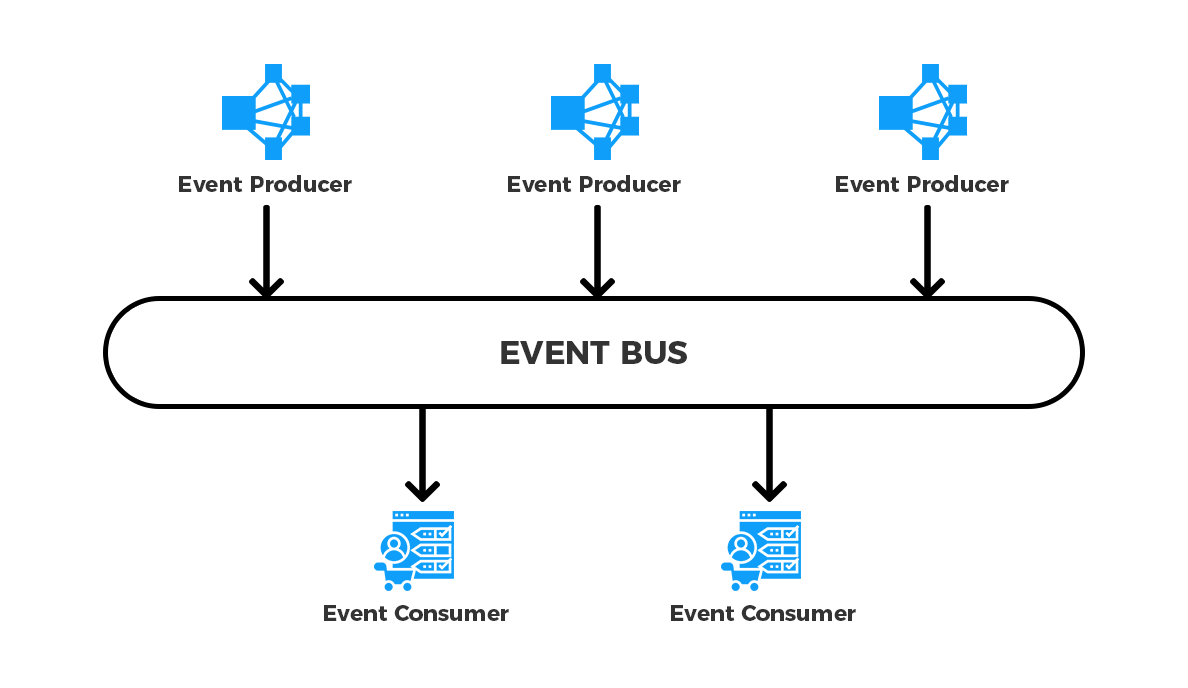
Графічне представлення EDA наведено на рисунку 2.2.

Рисунок 2.2 – EDA архітектура

Подійно-орієнтована система як правило складається з емітерів подій (Event Producer) (або агентів) і споживачів подій (Event Consumer) (або стоків). Стоки несуть відповідальність за здійснення реагування на появу події. Реакція не завжди може бути повністю забезпечена самим стоком. Наприклад, стік, може бути відповідальним лише за фільтрацію, трансформацію і відправку події до іншого компонента або він може забезпечити повністю самостійну реакцію на таку подію. Перша категорія стоків може бути заснована на традиційних компонентах, таких як проміжне програмне забезпечення, орієнтоване на обробку повідомлень в той час, як друга категорія стоків (самостійна реакція в режимі онлайн) може вимагати більш придатної платформи (фреймворку) для виконання транзакцій.

2.2 Аналіз технічних і програмних засобів

Оскільки розроблюваний програмний продукт є невеликого розміру та не потребує роботи одразу декількох спеціалістів або обчислювальних машин, основними апаратними засобами реалізації проекту було обрано наявний персональний комп’ютер у повній комплектації.

Переходячи до вибору операційної системи, необхідно звернутись до статистики. Згідно з дослідженнями NetMarketShare [5] (рис. 2.3) ОС Windows займає більш ніж 80 % ринку, а тому розроблюване програмне забезпечення першочергово повинно задовольняти потреби користувачів саме цієї операційної системи.

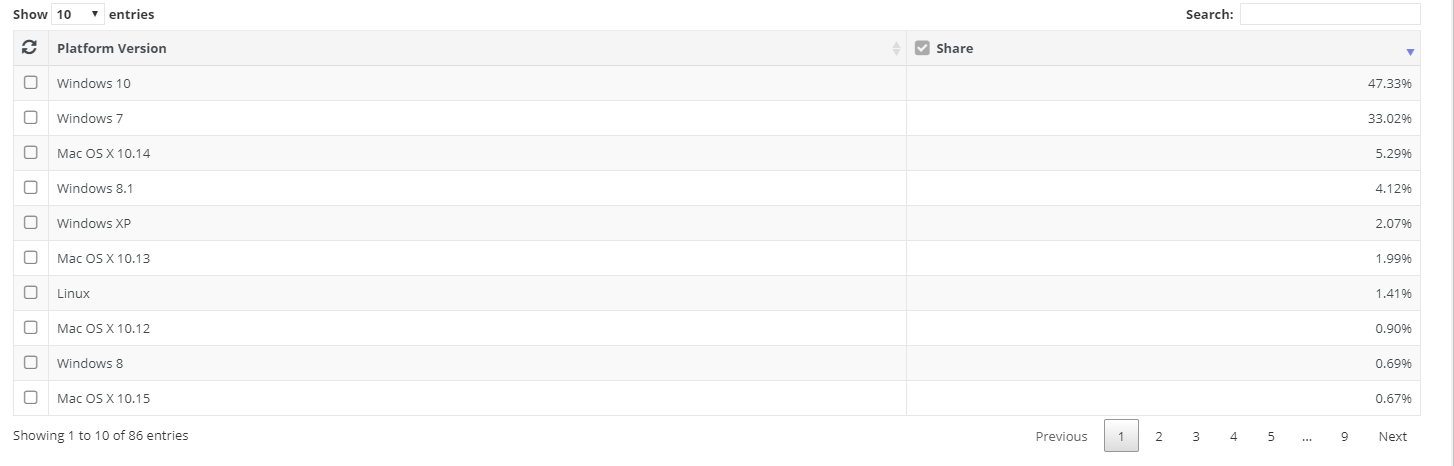


Рисунок 2.3 – Рейтинг операційних систем за популярністю (2021 р.)

Під час вибору мови програмування (МП) необхідно враховувати наступні чинники:

* наявність готових програмних рішень (бібліотеки і фреймворки, що дають змогу реалізувати необхідну функціональність);
* надійність (можливість мови програмування уберегти розробника від прийняття невдалих рішень);
* досвід розробника (вивчення нових бібліотек, фреймворків або навіть мов програмування потребує значних затрат у часі і шкідливо впливає на якість отриманого продукту).

Враховуючи усе вищезазначене, вибір мови програмування для реалізації Desktop-застосунку підпадає на C# (CSharp). Її авторами є компанія Microsoft, яка також розробили ОС Windows, а тому має багато готових рішень для роботи саме з цією операційною системою. Програми, написані з використанням С#, мають власний «збірник сміття» [6], що забезпечує постійне очищення невикористовуваної оперативної пам’яті. Дана мова програмування має насичений синтаксис, що в певній мірі вберігає програміста від небажаних ефектів (наприклад, небажане перевантаження функції, тощо).

Графічний інетрфейс застосунку буде виконано з використанням WPF (англ. «Windows Presentation Foundation»). Нижче приведені деякі переваги WPF перед іншими технологіями (наприклад, Windows Forms):

* Апаратне прискорення DirectX. Завдяки цьому все, що пов’язане з інтерфейсом і графічною частиною використовує вашу відеокарту, а не процесор, що значно пришвидшує роботу вашої програми
* Веб-подібна модель компонування. Замість того, щоб фіксувати елементи керування на місці з певними координатами, WPF підтримує гнучкий потік, що розміщує елементи керування на основі їхнього вмісту. В результаті виходить інтерфейс користувача, який може бути адаптований для відображення високодинамічного вмісту або до різних мов.
* Багата модель малювання. Замість малювання пікселів у WPF ви маєте справу з примітивами — базовими фігурами, блоками тексту та іншими графічними інгредієнтами. Крім того, доступні такі нові засоби, як дійсно прозорі елементи управління, можливість укладання один на одного безлічі рівнів з різним ступенем прозорості, а також вбудована підтримка тривимірної графіки.
* Розвинена текстова модель. Після багатьох років нестандартної обробки текстів WPF нарешті надає Windows-додаткам можливість відображення розширеного стилізованого тексту в будь-якому місці інтерфейсу користувача.
* Анімація. У WPF немає необхідності використовувати таймер для того, щоб змусити форму перемалювати себе. Натомість доступна анімація - невід'ємна частина платформи. Анімація визначається декларативними дескрипторами і WPF запускає її в дію автоматично.
* Стилі та шаблони. Стилі дозволяють стандартизувати форматування та багаторазово використовувати його по всьому додатку. Шаблони дають можливість змінити спосіб відображення елементів, навіть таких основних, як кнопки.
* Команди. Більшості користувачів відомо, що немає значення, звідки вони ініціюють команду відкриття (Open) — через меню чи панель інструментів; кінцевий результат той самий. Тепер ця абстракція доступна коду - можна визначати команди програми в одному місці і прив'язувати їх до багатьох елементів керування.
* Декларативний інтерфейс користувача. Хоча можна конструювати вікно WPF у коді, Visual Studio використовується інший підхід. Вміст кожного вікна серіалізується у вигляді дескрипторів XML у документі XAML. Перевага полягає в тому, що інтерфейс користувача повністю відокремлюється від коду, і дизайнери графіки можуть використовувати професійні інструменти для редагування файлів XAML, покращуючи зовнішній вигляд всієї програми. (XAML – це скорочення від Extensible Application Markup Language (розширювана мова розмітки додатків).)

Так як основна складність при розробці програми полягала саме у реалізації зручного і сучасного інтерфейсу з можливістю графічного моделювання структури ГВСС і WPF – один з небагатьох таких надпотужних фреймворків, то я обрав саму цю технологію.

2.3 Функціональна структура програми

Визначення послідовності дій користувача має велике значення під час проектування графічного інтерфейсу програмного забезпечення. Це дає змогу визначити моменти інтересу користвача до тієї чи іншої активності та наділити його лише потрібною функціональністю у данний момент часу, не перевантажуючи інтерфейс. З цією метою було розроблено функціональну блок-схему алгоритму роботи програми, зображену на рисунку 2.4.



Рисунок 2.4 – Функціональна блок схема

З функціональної діаграми можна побачити, що робота користувача з програмним забезпеченням поділена на п’ять етапів.

Для більш детального аналізу активностей користувача була розроблена діаграма прецедентів (рис. 2.5), що описує взаємодію користувача з програмним забезпеченням без прив’язки до часу.

З цієї діаграми можна побачити зв’язаність окремих функцій системи, що також становить великий інтерес при проектуванні програмного забезпечення, наприклад, для вдалого вибору і розміщення елементів керування на формах.



Рисунок 2.5 – Розроблена діаграма прецедентів

З діаграми прецедентів видно, що робота зі структурою ГВСС і отримання часу виявлення порушення є двома взаємопов’язаними процесами, бо неможливо отримати час виявлення порушення не визначивши структуру. А ось «відображення результатів MathLab» є повністю незалежним прецедентом і щодб побачити результати користувачеві необхідно просто завантажити файл зі спеціальним розширенням.

3. розробка програми

3.1 Розробка Desktop-застосунку

3.1.1 Визначення класів Desktop-застосунку

Оскільки обрана мова програмування С# є виключно об’єктно-орієнтованою, то все функціонування програмного забезпечення виражається у взаємодії окремих класів, сутностей предметної області.

Згідно розробленої архітектури програмного забезпечення, розроблюваний додаток містить класи, які відповідають за бізнес-логіку програми і за зовнішний вид (інтерфейс).

Класи/структури, які відповідають за бізнес логіку:

* class GVSS – основний клас програми, відповідає за структуру ГВСС і включає методи взаємодії з нею;
* class Vertex – клас, який представляє вершину у ГВСС. Є включеним у клас GVSS і не доступний ззовні;
* class Edge – клас, який представляє зв’язок (ребро) між двома вершинами у ГВСС. Є включеним у клас Vertex і не доступний ззовні;
* struct Correspondence – структура, який представляє собою відповідність між певною вершиною ГВСС і ознакою, яку представляє дана вершина. Є включеною у клас GVSS і не доступна ззовні;
* class Sensor – клас, який представляє ознаку і містить всю необхідну інформацію про неї. У більшості випадків використовується класом GVSS, але є окремою сутністю.
* class Event – клас, який представляє подію (подія = порушення + ланцюг ознак). Містить повну інформацію про подію і за допомогою методів цього класу ми можемо визначити час виявлення порушення і інші результуючі величини. Є окремою сутністю, але у більшості випадків використовується саме класом GVSS;
* class VertextAndNumOfSubvertices – клас, який являє собою структуру даних, яка визначає певну вершину і кількість вершин у піддереві. Є включеним у клас Event і використовується ним для розрахунків, але є доступним ззовні;
* class DefaultSettings – статичний клас, який представляє собою збірку всіх налаштувань системи за замовчуванням: тут містяться всі розрахункові дані, а також переклади всіх написів, які представлені у графічному інтерфейсі користувача;
* class BinarySerilization – статичний клас, який представляє собою сервіс для збереження будь-яких класів у мові програмування С# у бінарному форматі;
* Класи виключень (є спадкоємцями класу Exception): InvalidArgument, InvalidSensor, UnexpectedError, NotFoundException, SensorRepeatsException – використовуються для обробки помилок;

Для визначених класів було створено діаграму класів, зображену на рисунку 3.1.

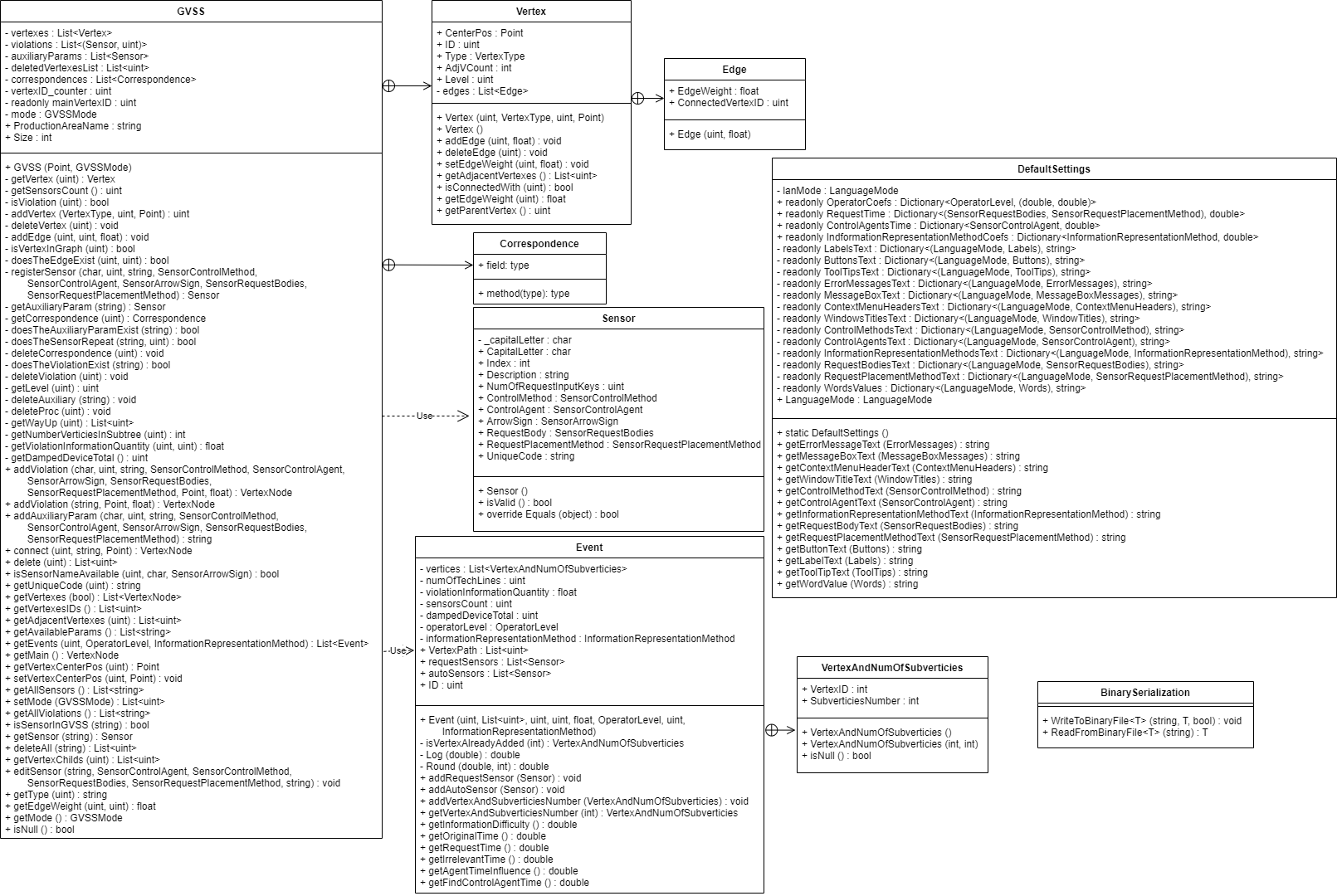


Рисунок 3.1 – Діаграма класів моделі

Класи/структури, які відповідають за графічний інтерфейс:

* class ParameterValueHelper – статичний клас, який допомагаю ініціалізувати поля з випадаючими списками (комбобокси) у графічному інтерфейсі;
* class InputCorrector – статичний клас, який відповідає за динамічне реагування інтерфейсу на користувацьке введення;
* struct ErrorWhileInput – допоміжна структура, яка є включеною у клас InputCorrector і не доступна ззовні. Використовується для специфікації помилок при користувацькому введенні;
* class RenderVisualService – статичний клас, який являє собою сервіс, який використовується для отриманная зображення робочої обласі з ГВСС структурою у програмі;
* class MainWindow – клас, який являє собою головне вікно програми. Насичений функціоналом і об’єднує всі інші вікна. Відповідає також за графічне моделювання структури ГВСС;
* class EditDialog – клас, який являє собою вікно програми, у якому користувач може змінювати певну ознаку;
* class MathLABResultWindow – клас, який є вікном з результатами, отриманими за допомогою MathLAB;
* class ShowTime – клас, який є вікном і, що містить інформацію щодо отриманих результатів обчислень;
* class VertexDialog – клас, який є вікном і дозволяє користувачеві додавати вершини до структури ГВСС.
* class VertexNode – клас, який використовується як посередник між класом GVSS і іншою логікою програми (наприклад, графічним інтерфейсом). Містить в собі стислу інформацію про певну вершину;
* class Computations – допоміжний статичний клас, який використовується для генерації текстового представлення назви ознаки відповідно до переданий параметрів;
* class EventDifficulty – допоміжний клас, який використовується класом MathLABResultWindow для позначення події і часу її виявлення;
* struct Connection – допоміжна стуктура, яка використовується в класі MainWindow і позначає з’єднання двох вершин (ребро) ГВСС структури;

Розроблена діаграма вищезазначених класів зображена на рисунку 3.2.

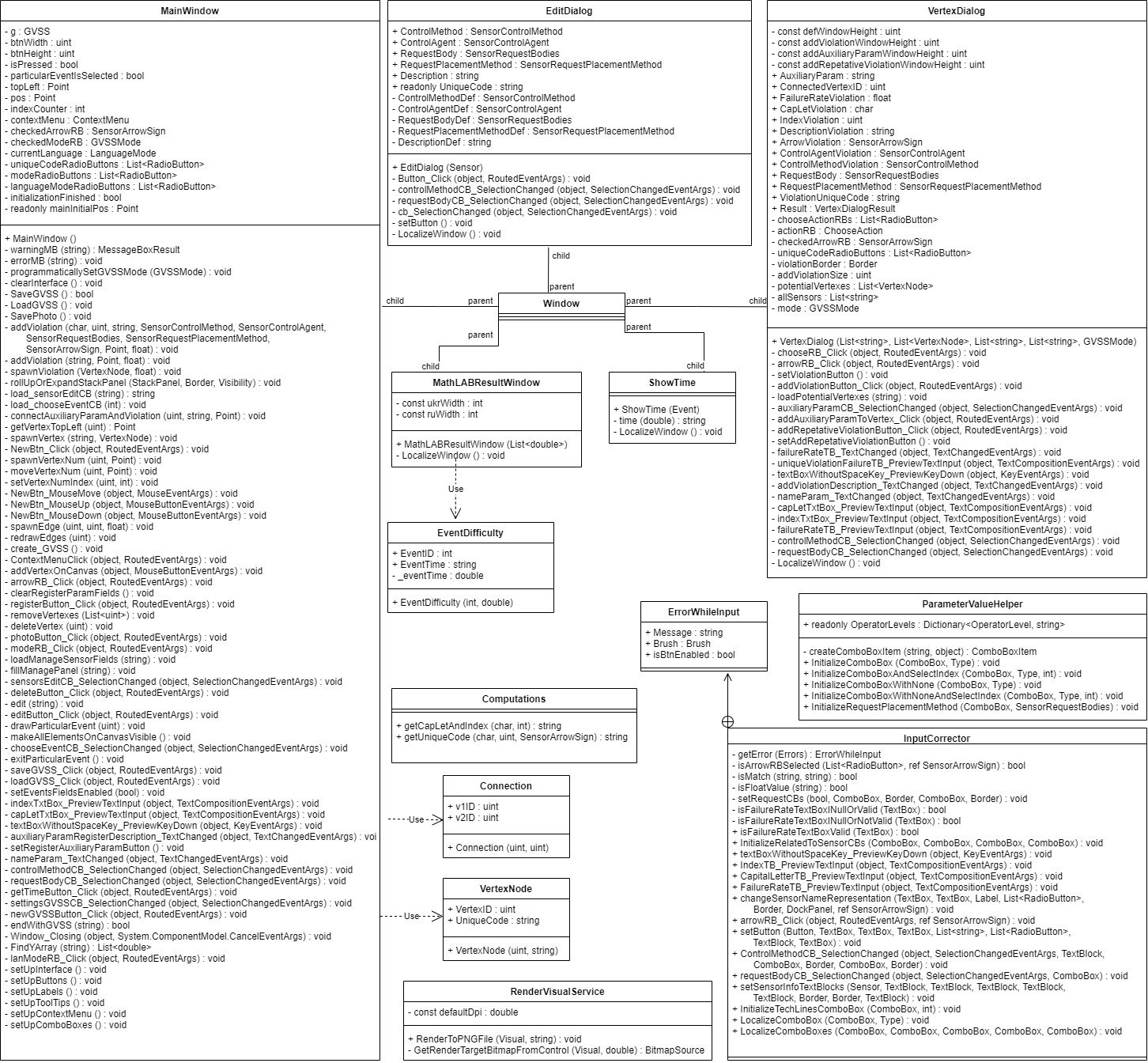


Рисунок 3.2 – Діаграма класів інтерфейсу

3.1.2 Розробка графічного інтерфейсу користувача

Будь який програмний інтерфейс має містити окремний контролер або групу контролерів, що є абстакцією до виконуваної програмою функції. Відповідно до функцій системи, специфікованих у програмі були розроблені:

* початкової форми, що надає доступ до інших компонентів системи;
* форма додавання нової вершини на робочу область ГВСС (мається на увазі графічна зміна структури ГВСС);
* форми редагування ознаки ГВСС;
* форма з результатами програмних обчислень;
* форма з результатами обчислень MathLab.

При проектуванні форм були використані наступні елементи керування:

* кнопки;
* поля для вводу;
* поля з випадаючими списками;
* кнопки-перемикачі.

Всі форми побудовані відповідно до загальних правил побудови ергономічних [7] інтерфейсів.

Усі форми виконані з використанням одного набору кольорів, подібних шрифтів та повторюваних геометричних форм елементів керування. Дизайн виконано у теплих синіх тонах, що створюють відчуття комфорту та не напруженності інтерфейсу.

Далі будуть детально розглянуті всі ці форми.

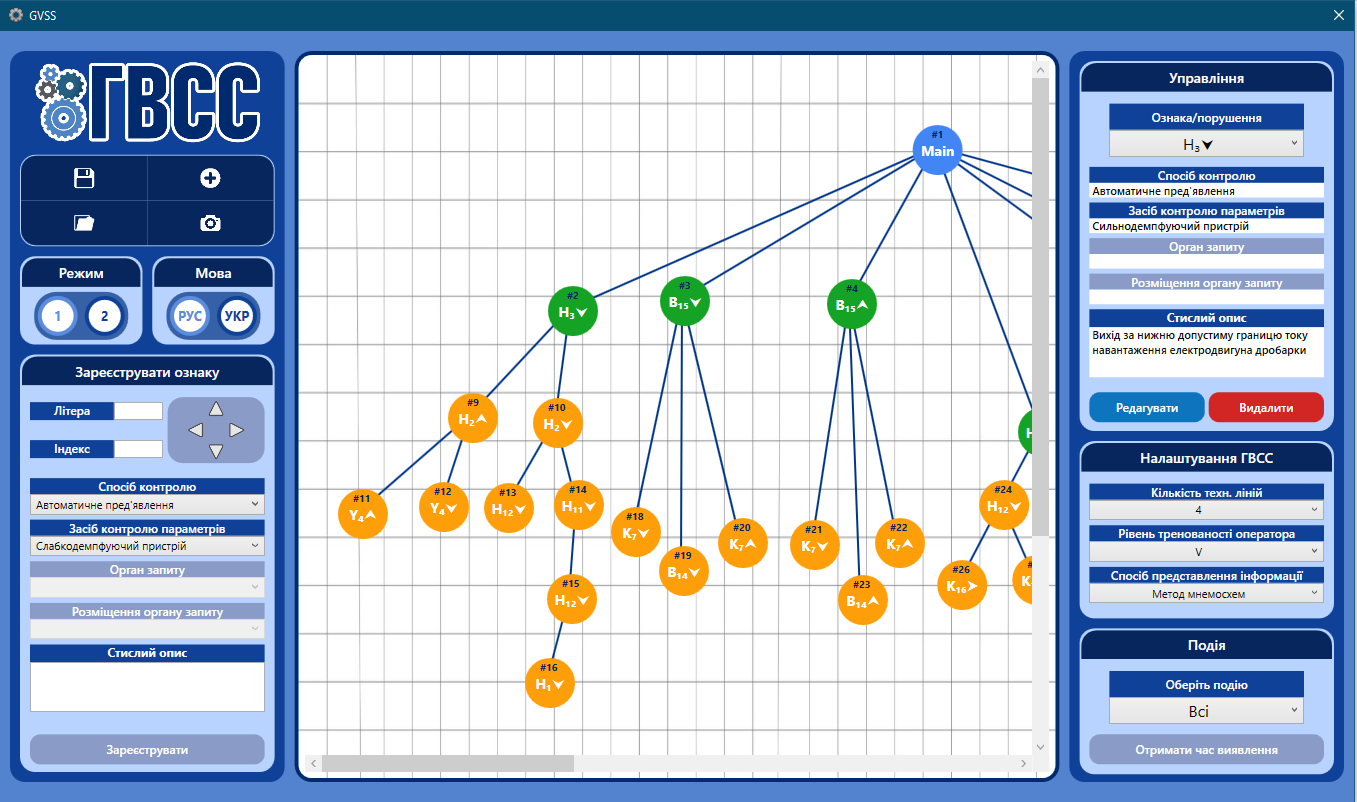
3.1.2.1 Головна форма

Рисунок 3.3 – Головна форма Desktop-застосунку

Головна форма (рис. 3.3) представляє собою вікно, розподілене на 8 блоків:

1. Блок з кнопками управління (рис 3.4)

Містить 4 кнопки: «Зберегти», «Завантажити», «Створити нову ГВСС», «Зберегти фото структури».

Кожна з цих кнопок виконує відповідну функцію зі структурою ГВСС

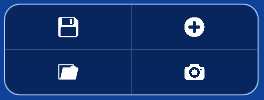


Рисунок 3.4 – Блок з кнопками управління

1. Блок вибору режиму (рис 3.5)

Містить дві кнопки перемикання режимів. Далі у керівництві оператора будуть описаний кожний з цих режимів.

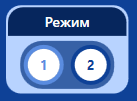


Рисунок 3.5 – Блок вибору режиму

1. Блок вибору мови інтерфейсу (рис 3.6)

Містить дві кнопки перемикання мови інтерфейсу (українську і російську).



Рисунок 3.6 – Блок вибору мови

1. Блок реєстрації ознаки (рис 3.7)

Містить поля, які необхідно запонити для реєстрації ознаки. У заповненому вигляді має вигляд як на рисунку 3.8.

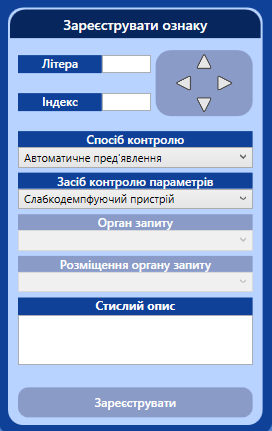
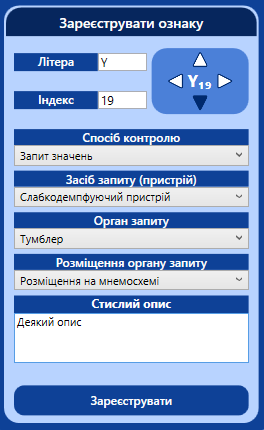
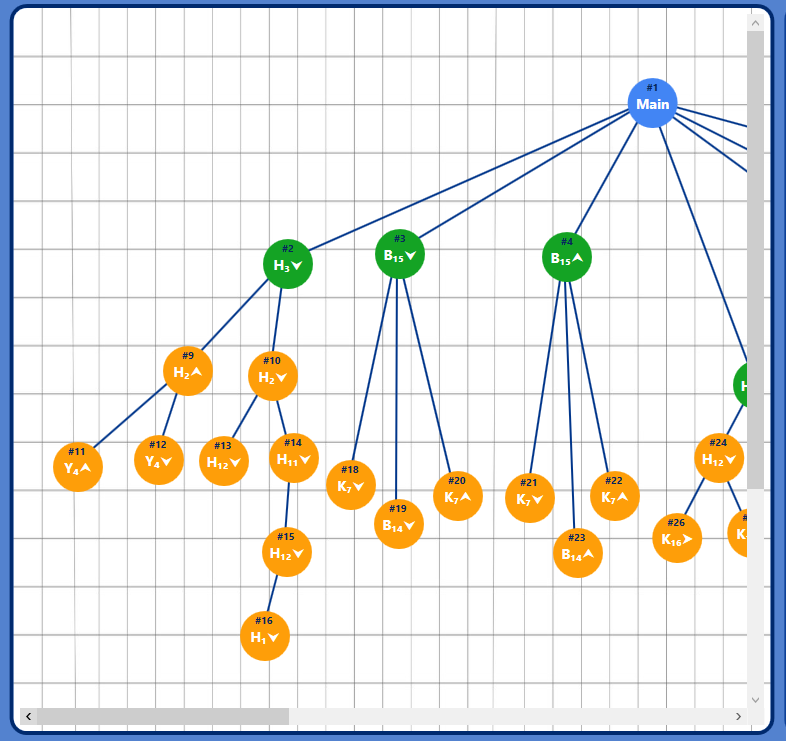


Рисунок 3.7 – Блок реєстрації ознаки Рисунок 3.8 – Заповнений блок реєстрації

1. Блок графічного моделювання ГВСС (рис 3.9)

Представляє собою робочу область, на якій користувач може у явному вигляді створювати структуру ГВСС. Переміщення по ній відбувається за допомогою «скрол-барів». Кожна вершина ГВСС, розташована у даній обласі підтримує контекстне меню (рис 3.10)

Рисунок 3.9 – Блок графічного моделювання

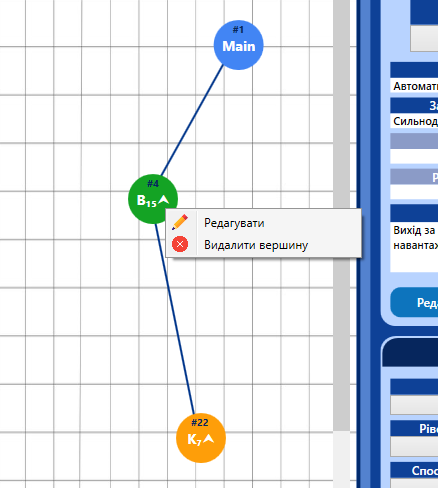


Рисунок 3.10 – Контекстне меню у блоці графічного моделювання

1. Блок управління ознаками (рис 3.11)

Дозволяє переглянути інформацію стосовно певної ознаки, а також викликати форму редагування, а також видалити ознаку.

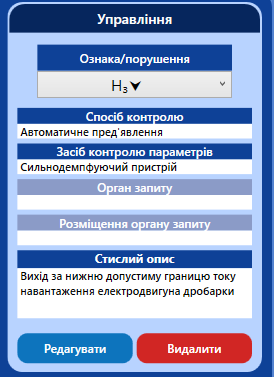


Рисунок 3.11 – Блок управління ознаками

1. Блок налаштування ГВСС (рис 3.12)

Містить загальну інформацію щодо ГВСС і дозволяє її змінювани

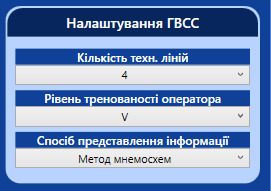


Рисунок 3.12 – Блок налаштування ГВСС

1. Блок подій (рис 3.13)

У цьому блоці користувач може обрати подію і отримати час виявлення порушення, яке позначає дана подія

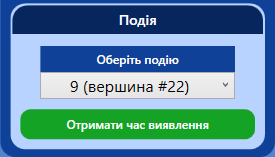


Рисунок 3.13 – Блок подій

***3.1.2.2 Форма додавання нової вершини на робочу область ГВСС***

Ця форма викликається при натисканні користувача лівою кнопкою миші на робочу область (блок графічного моделювання головного вікна). І має виглядає як на рисунку 3.14

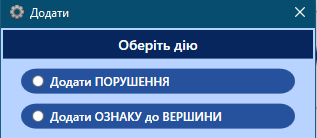


Рисунок 3.14 – Форма додавання

Вона містить дві кнопки-перемикачі: «Додати порушення» і «Додати ознаку до порушення». Відповідно до обраної опції форма буде пропонувати певний функціонал користувачеві.

Якщо користувач обере опцію «Додати порушення» і буде виставлений перший режим ГВСС, то форма буде виглядати відповідно до рисунка 3.15

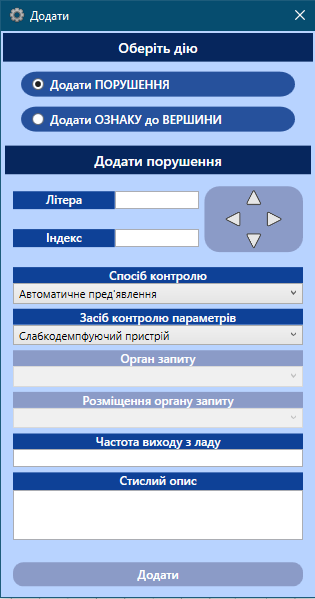


Рисунок 3.15 – «Додати порушення» (режим ГВСС №1)

Якщо користувач обере опцію «Додати порушення» і буде виставлений другий режим ГВСС, то форма буде виглядати відповідно до рисунка 3.16

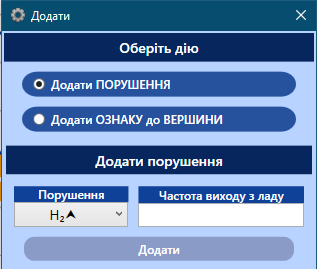


Рисунок 3.16 – «Додати порушення» (режим ГВСС №2)

Якщо користувач обере опцію «Додати ознаку до вершини», то форма буде виглядати відповідно до рисунка 3.17

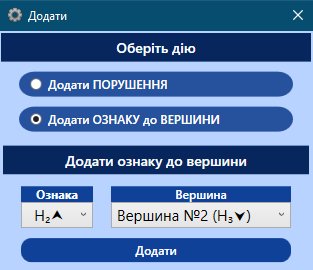


Рисунок 3.17 – «Додати ознаку до вершини»

***3.1.2.3 Форма редагування ознаки ГВСС***

Викликається при натисканні на кнопку «Редагувати» у блоці управління на головній формі, або при виборі опції «Редагувати» у контекстному меню блоку графічного моделювання. Дозволяє користувачеві змінювати властивості певної ознаки ГВСС. Зовнішній вигляд форми показаний на рисунку 3.18

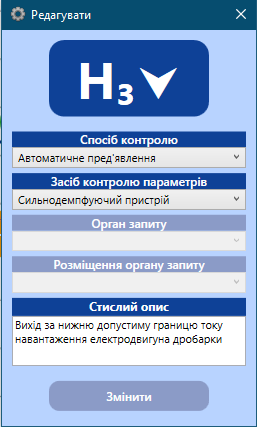


Рисунок 3.18 – Форма редагування ознаки ГВСС

***3.1.2.4 Форма з результатами програмних обчислень***

Викликається, коли користувач обирає подію о блоці подій головної форми і натискає кнопку «Отримати час виявлення». Ця форма відображає інформацію щодо часу виявлення порушення певної події і його складових. Зовнішній вигляд наведений на рисунку 3.19



Рисунок 3.19 – Форма з результатами програмних обчислень

***3.1.2.5*** ***Форма з результатами обчислень MathLAB***

Викликається, коли користувач натискає на кнопку «Завантажити» у блоці з кнопками програмного управління на головній формі і обирає файл з розширенням .asv. Ця форма є виключно інформативною і відображає результати обчислень, отриманих за допомогою MathLAB . Зовнішній вигляд наведений на рисунку 3.20

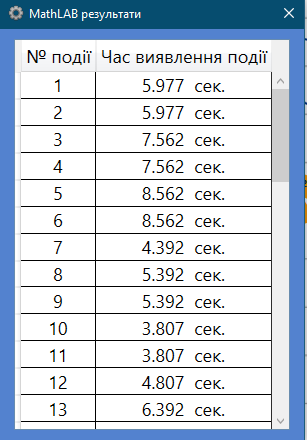


Рисунок 3.20 – Форма з результатами обчислень MathLAB

4. Керівництво програміста

4.1 Призначення та умови застосування програми

Дане програмне забезпечення призначене моделювання структури ГВСС і автоматичного обчислення часу виявлення порушення оператором.

Для коректного функціонування Desktop-застосунку необхідна версія ОС Windows XP та вище.

4.2 Характеристики програми

Програмне забезпечення відловлює можливі помилки в роботі оператора, запобігаючи від можливих помилок в їх роботі, наприклад, при спробі користувача зареєструвати ознаку, яка має не унікальне ім’я, кнопка зареєструвати змінить свій колір на сіро-синій і буде вимкнутою, а також користувач побачить відповідне повідомлення червоним кольором у блоці реєстрації ознак на головній формі. Усі повідомлення, отримувані оператором під час виключних ситуацій описані у пункті 4.4.

Для запуску програмного забезпечення необхідно виконати файл GVSS.exe. Програма не потребує та не підтримує виклик з додатковими аргументами з командного рядка.

4.3 Початкові та вихідні дані

Початковими даними, необхідними для функціонування програмного продукту, є структура ГВСС, яка представляє собою граф, де вершинами є порушення і їх ознаки. Кожна ознака має такі властивості: унікальна назва, спосіб контролю, засіб контролю параметрів/засіб запиту, орган запиту, метод розміщення запиту на мнемосхемі, стислий опис. Порушення мають бути визначені за допомогою таких самих параметрів, а також для них повинна бути відома частота виходу з ладу. Додатково повинні бути визначені такі параметри для ГВСС: кількість технологічних ліній, рівень тренованості оператора, спосіб представлення інформації.

Вихідними даними є орієнтовний час виявлення порушення оператором, а також: кількість інформації в події (відповідно до теорії інформації вимірюється у двійкових одиницях), початковий час, час витрачений на запити, затримка (залежність від кількості іррелевантної інформації), вплив типу засобів представлення (сек.), час витрачений на пошук засобів представлення.

4.4 Модифікація програми

В програмі дуже багато класів, які визначають її роботу, структуру і зовнішній вигляд. Але рушієм програми є по більшій мірі всього два класи: GVSS і Event. GVSS – представляє так би мовиги структуру даних ГВСС і визначає всі дії, які можна з нею робити. Event – клас, який уособлює певну подію у структурі ГВСС і відповідає за розрахунок часу виялення порушення (результат роботи програми і мета, з якою вона створювалась). Тобто, кажучи простими словами, якщо ми візьмемо і дістанемо ці два класи з програми і під’єднаємо їх до консольного застосунку, то програма буде працювати так само добре. Тому ми бачимо, що взаємодія безпосередньо з програмою відбувається за допомогою інтерфейсу саме цих двох класів, а інші класи програми визначають її графічну реалізацію.

Нижче у таблиці 4.1 буде наведено детальну структуру інтерфейсу (приватні методи) цих класів.

Таблиця 4.1 – Перелік атрибутів та методів

|  |  |  |  |
| --- | --- | --- | --- |
| Клас | Атрибут /  Метод | Тип даних | Короткий опис |
| 1 | 2 | 3 | 4 |
| GVSS | Size | int | Розмір структури ГВСС |
| addViolation | VertexNode | Додати порушення |
| addAuxiliaryParam | string | Зареєструвати ознаку |
| connect | VertexNode | З’єднати дві вершини |
| delete | List<uint> | Видалити вершину |
| isSensorNameAvailable | bool | Чи доступне ім’я ознаки для реєстрації |
| getUniqueCode | string | Отримати унікальне ім’я певної вершини |
| 1 | 2 | 3 | 5 |
| GVSS | getVertexes | List<VertexNode> | Отримати список вершин |
| getVertexesIDs | List<uint> | Отримати список унікальних номерів вершин |
| getAdjacentVertexes | List<uint> | Отримати список суміжних вершин певній вершині |
| getEvents | List<Event> | Отримати події поточної ГВСС |
| getMain | VertexNode | Отримати інформацію щодо головної вершини ГВСС |
| getVertexCenterPos | Point | Отримати координати вершини ГВСС |
| setVertexCenterPos | Void | Визначити координати певної вершини ГВСС |
| getAllSensors | List<string> | Отпримати список всіх зареєстрованих ознак |
| setMode | List<uint> | Змінити режим ГВСС |
| getAllViolations | List<string> | Отримати список всіх порушень |
| isSensorInGVSS | bool | Чи знаходиться певна ознака в структурі ГВСС |
| getSensor | Sensor | Отримати ознаку за її унікальним ім’ям |
| deleteAll | List<uint> | Видалити ознаку (видалити всі вершини, які представлюють дану ознаку) |
| editSensor | void | Редагувати ознаку |
| 1 | 2 | 3 | 4 |
| GVSS | getType | string | Отримати тип вершини |
| getEdgeWeight | float | Отримати вагу ребра (в нашому випадку це частота порушення) |
| getMode | GVSSMode | Отпримати поточний режим ГВСС |
| isNull | bool | Чи є має ГВСС порожню структуру? |
| Event | getInformationDifficulty | double | Отримати інформаційну складність події |
| getOriginalTime | double | Отримати початковий час події |
| getRequestTime | double | Отримати час витрачений на запити |
| getIrrelevantTime | double | Отримати час затримки, що залежить ві кількості іррелевантної інформації в системі |
| getAgentTimeInfluence | double | Отримати час, який залежить від типу засобу представлення |
| getFindControlAgentTime | double | Отримати час, витрачений на пошук засобу представлення |

5 Керівництво оператора

5.1 Призначення та умови виконання програми

Дане програмне забезпечення призначене моделювання структури ГВСС і автоматичного обчислення часу виявлення порушення оператором.

Для коректного функціонування Desktop-застосунку необхідна версія ОС Windows XP та вище.

5.2 Виконання програми

Для запуску програмного забезпечення необхідно виконати файл GVSS.exe. Програма не потребує та не підтримує виклик з додатковими аргументами з командного рядка.

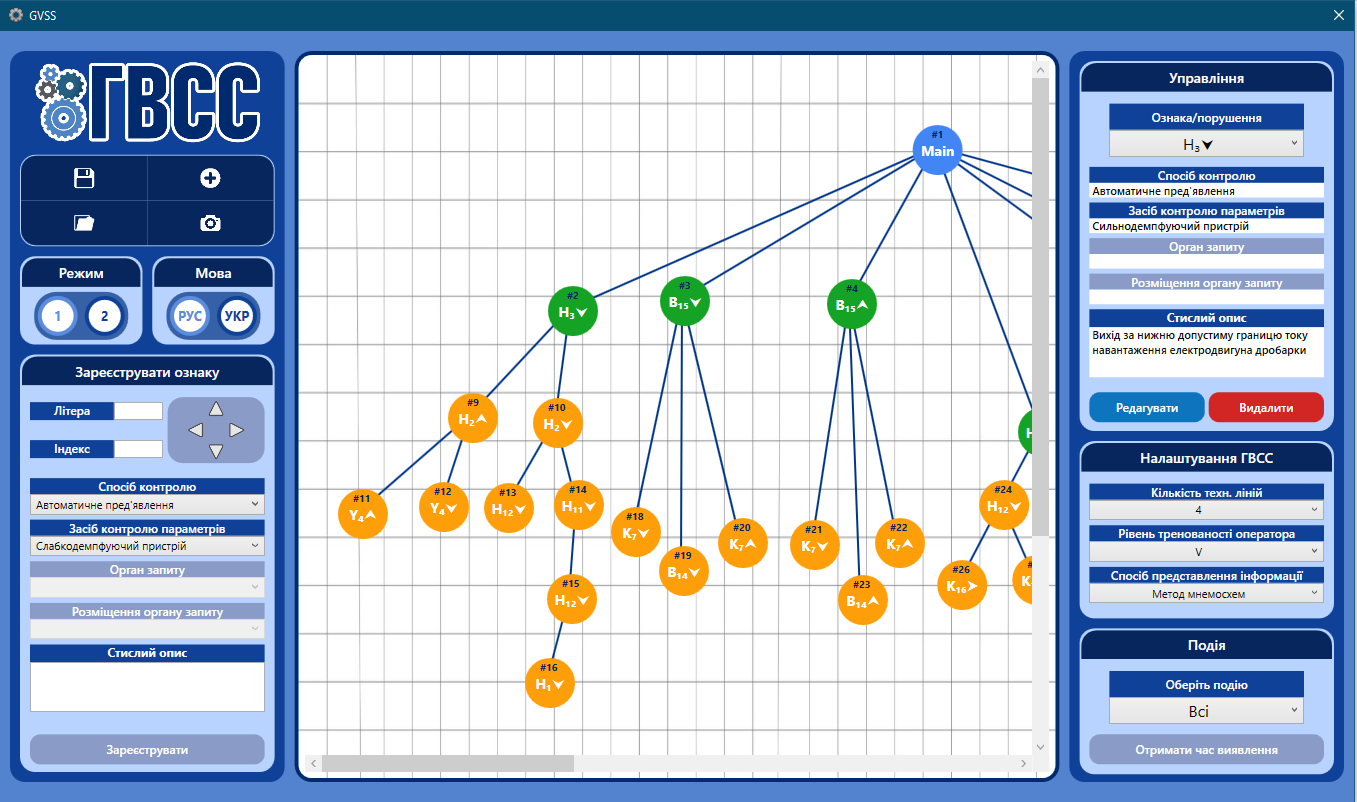
Після запуску програми оператор потрапляє на її головну форму (рисунок 5.1).

Рис 5.1 – Головна форма програми

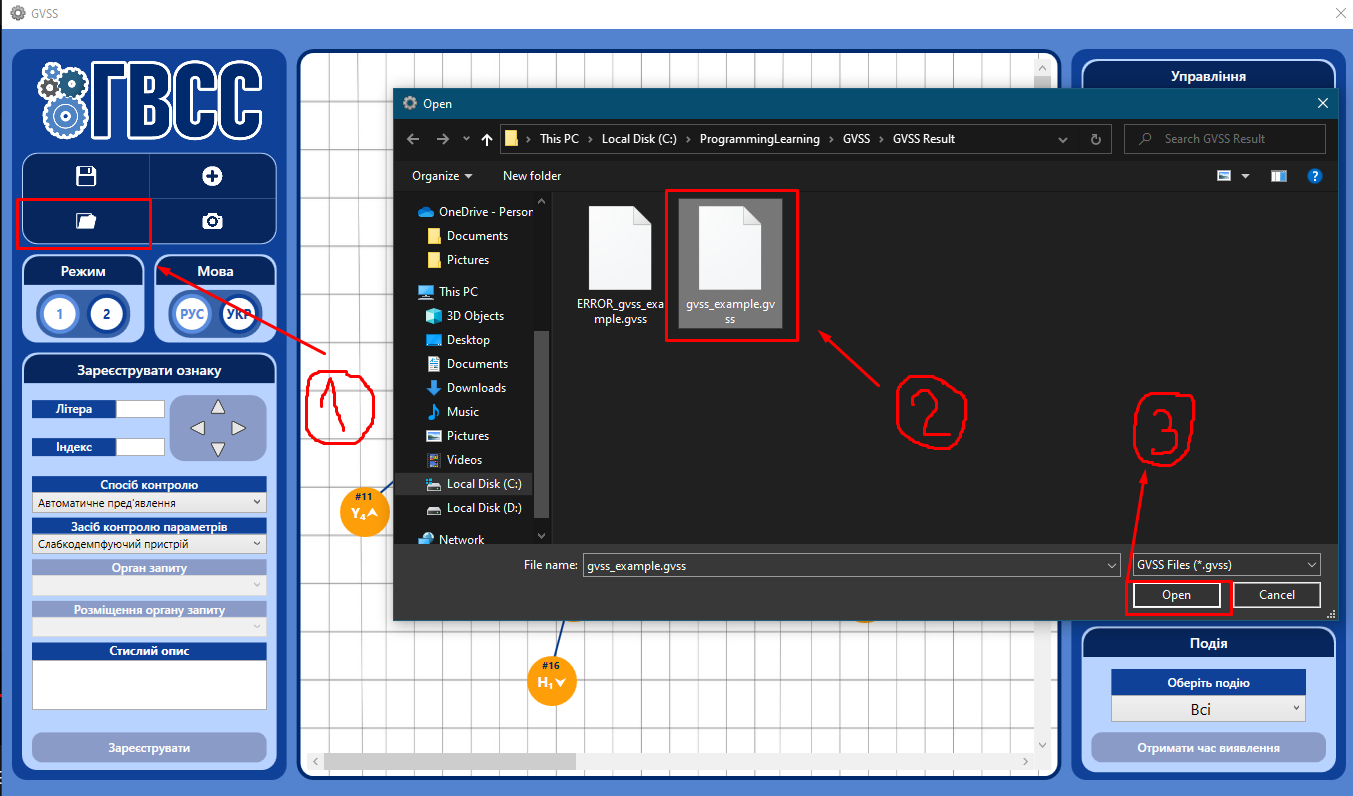
Програма реалізує певний функціонал взаємодії з ГВСС і тому у користувача можуть виникнути такі потреби:

1. Робота з системою:
   1. Зберегти ГВСС;
   2. Завантажити ГВСС з файлу;
   3. Створити нову ГВСС;
   4. Зберегти фото ГВСС;
   5. Переглянути таблицю результатів, отриманих за допомогою MathLAB;
   6. Змінити мову інтерфейсу
2. Робота зі структурою ГВСС:
   1. Змінити режим ГВСС;
   2. Змінити налаштування ГВСС;
   3. Зареєструвати ознаку;
   4. Додати вершину на робочу область;
   5. Перміщення вершини;
   6. Редагувати ознаку/вершину;
   7. Переглянути інформацію про вершину/ознаку;
   8. Видалити ознаку;
   9. Видалити вершину;
   10. Отримати час виявлення порушення.

Далі будуть розглянуті всі ці функції, а точніше шляхи їх досягнення (виконання).

***5.2.1 Зберегти ГВСС***

Для того, щоб зберегти ГВСС необхідно натиснути на кнопку «Зберегти» у блоці управління програмою на головній формі і визначити шлях для збереження файлу. Детальніше показано на рис 5.2

Рис 5.2 – Процес збереження ГВСС

***5.2.2 Завантажити ГВСС з файлу***

Для того, щоб завантажити ГВСС необхідно натиснути на кнопку «Завантажити» у блоці управління програмою на головній формі і визначити шлях до файлу. Детальніше показано на рис 5.3

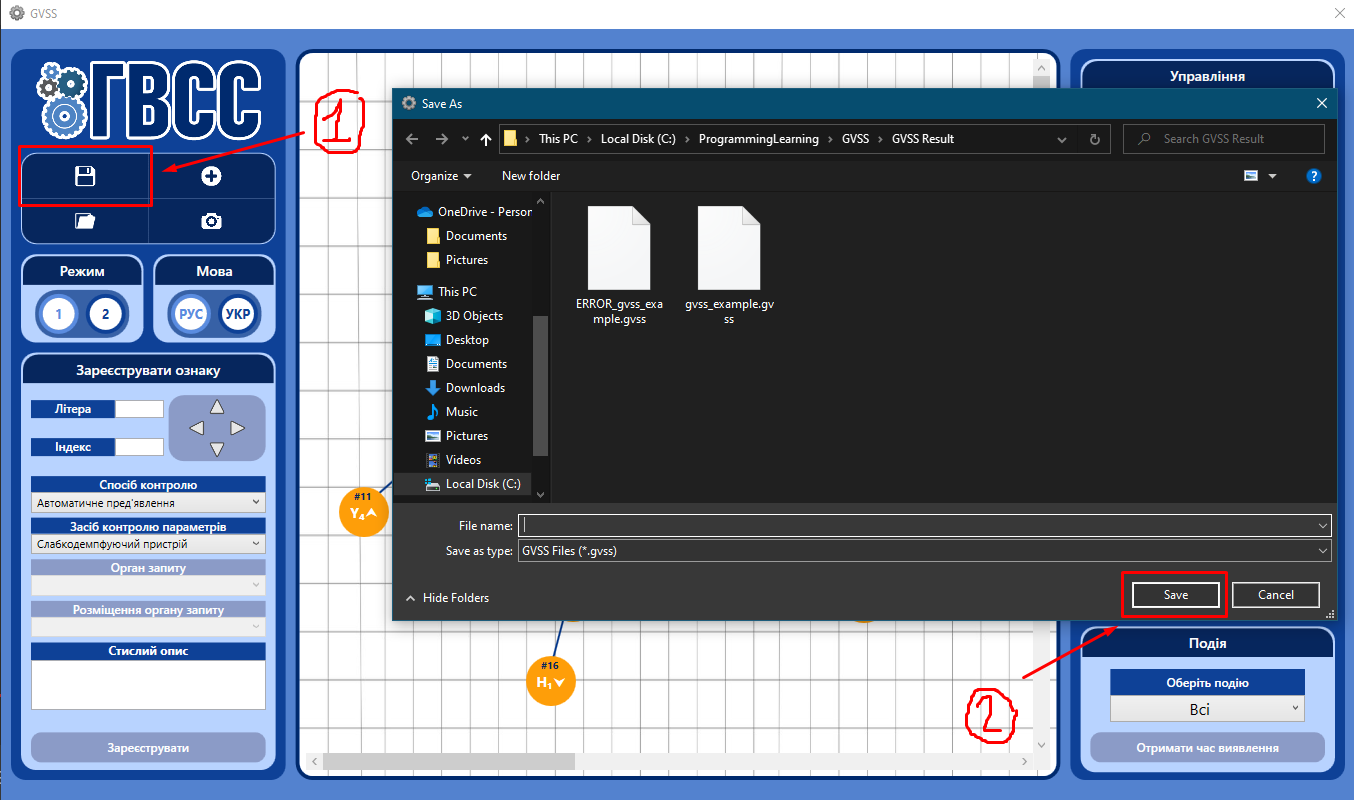


Рис 5.3 – Процес завантаження ГВСС

***5.2.3 Створити нову ГВСС***

Для того, щоб створити нову ГВСС необхідно натиснути на кнопку «Створити нову ГВСС» у блоці управління програмою на головній формі як показано на рис 5.4



Рис 5.4 – Процес створення нової ГВСС

***5.2.4 Зберегти фото ГВСС***

Для того, щоб зберегти фото ГВСС необхідно натиснути на кнопку «Зберегти фото ГВСС» у блоці управління програмою на головній формі і визначити шлях до файлу. Детальніше показано на рис 5.5

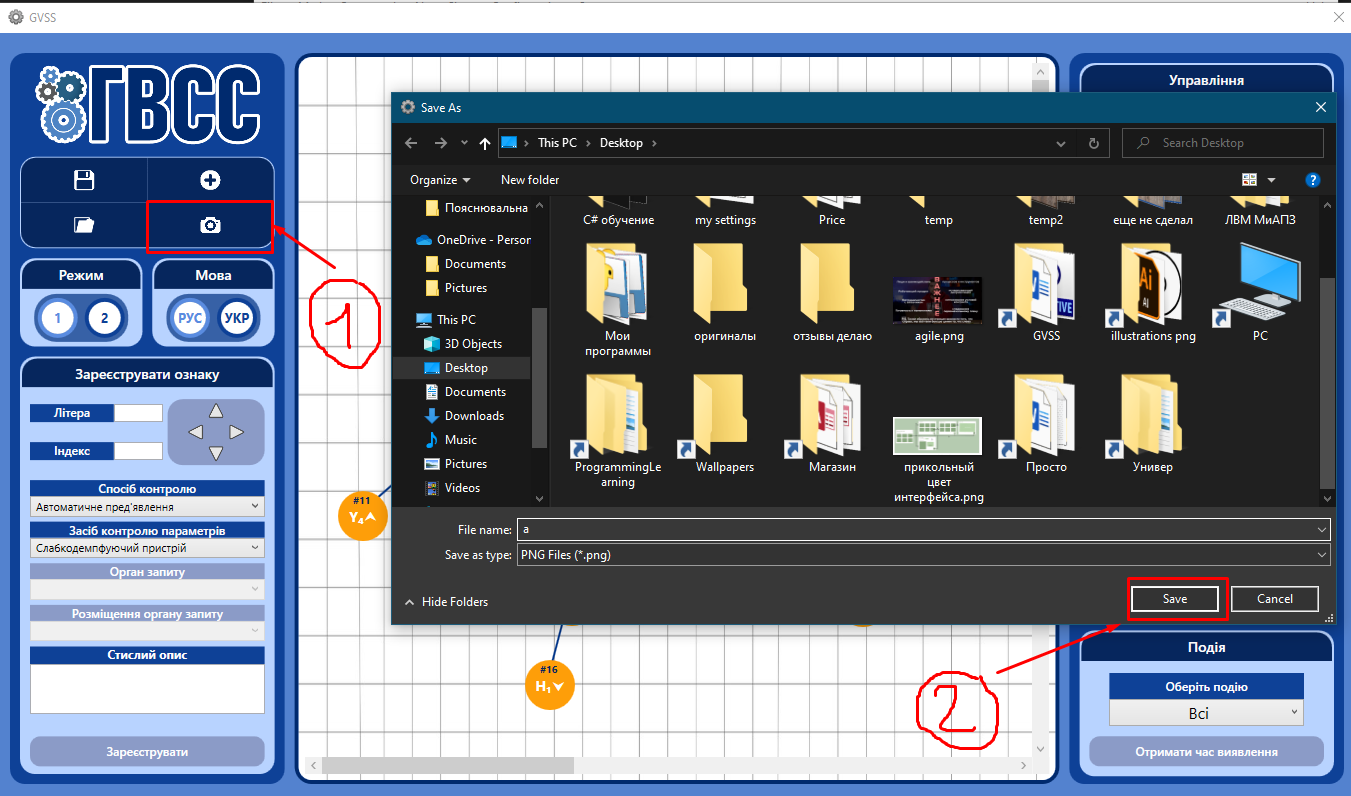
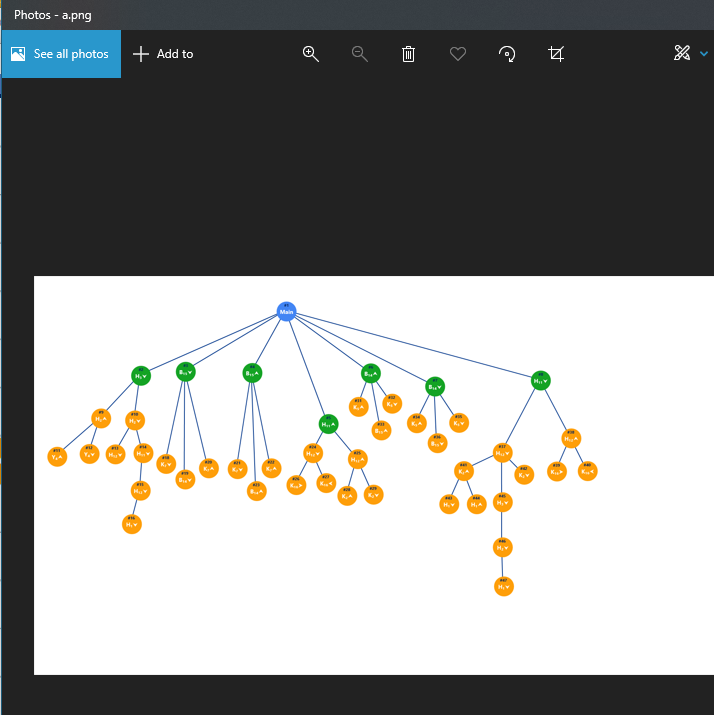


Рис 5.5 – Процес збереження фото структури ГВСС



***\***

Рис 5.6 – Збережене фото структури ГВСС

***5.2.5 Переглянути таблицю результатів, отриманих за допомогою MathLAB***

Для того, щоб переглянути результати, отримані за допомогою MathLAB необхідно натиснути на кнопку «Завантажити» у блоці управління програмою на головній формі і визначити шлях до файлу з розширенням .asv. Детальніше показано на рис 5.7

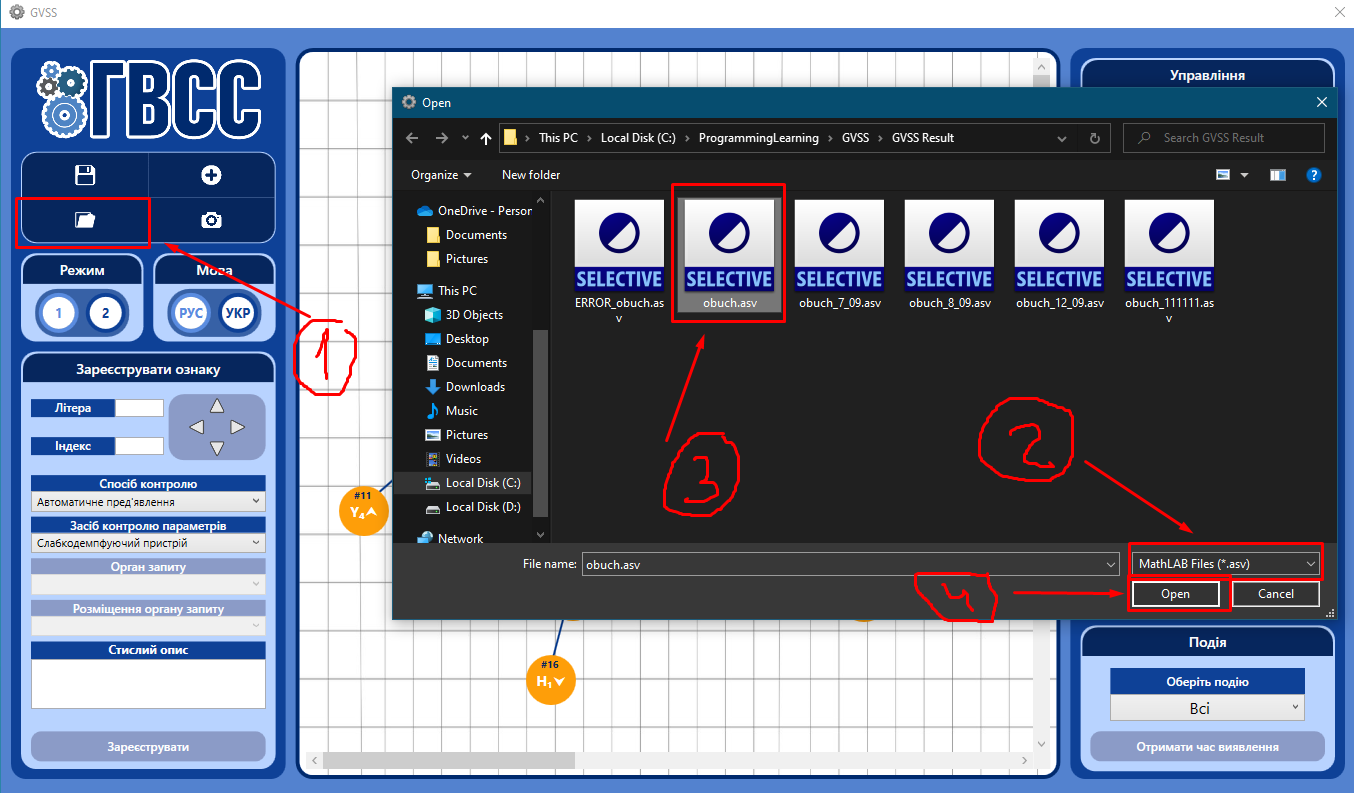


Рис 5.7 – Перегляд результатів, отриманих за допомогою MathLAB

***5.2.6 Змінити мову інтерфейсу***

Для того, щоб змінити мову інтерфейсу необхідно обрати необхідну мову у блоці вибору мови на головній формі. Детальніше показано на рис 5.8

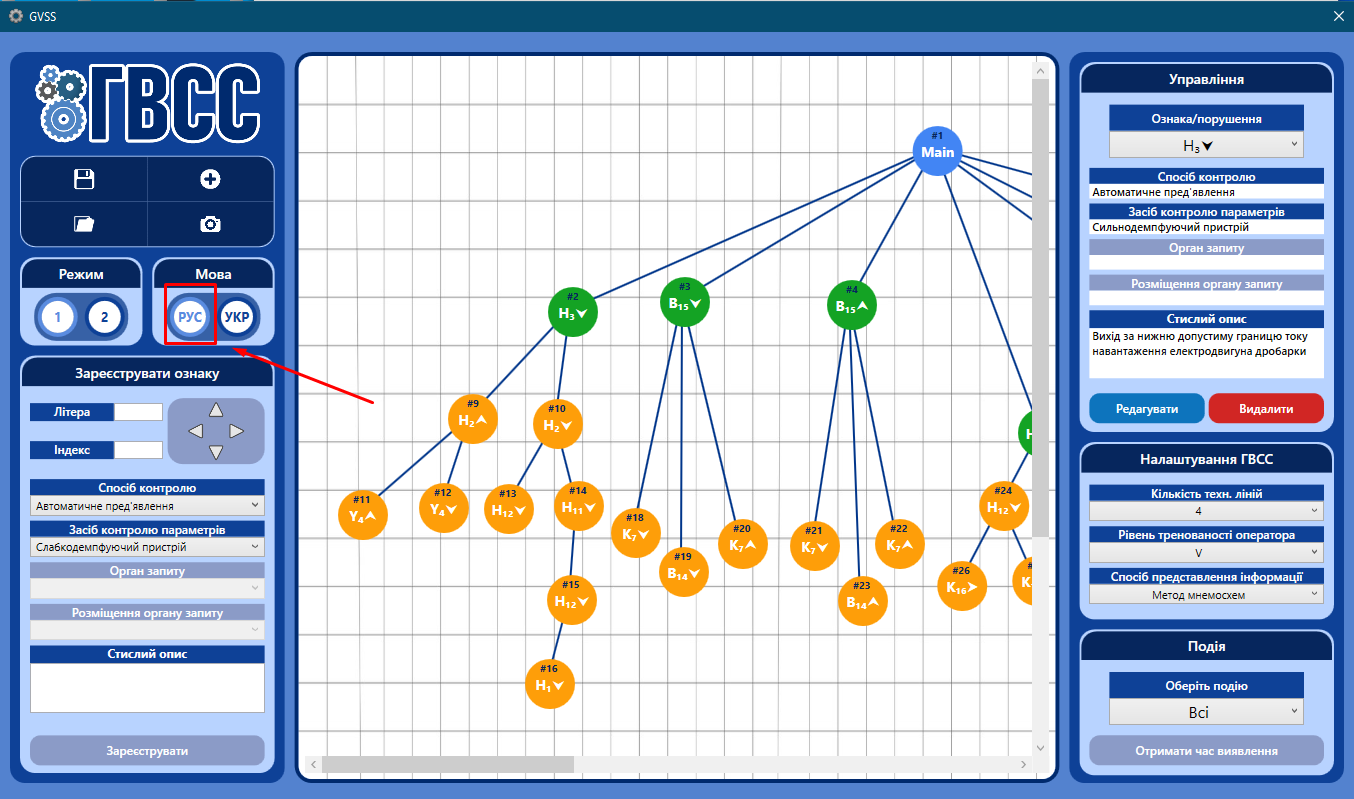
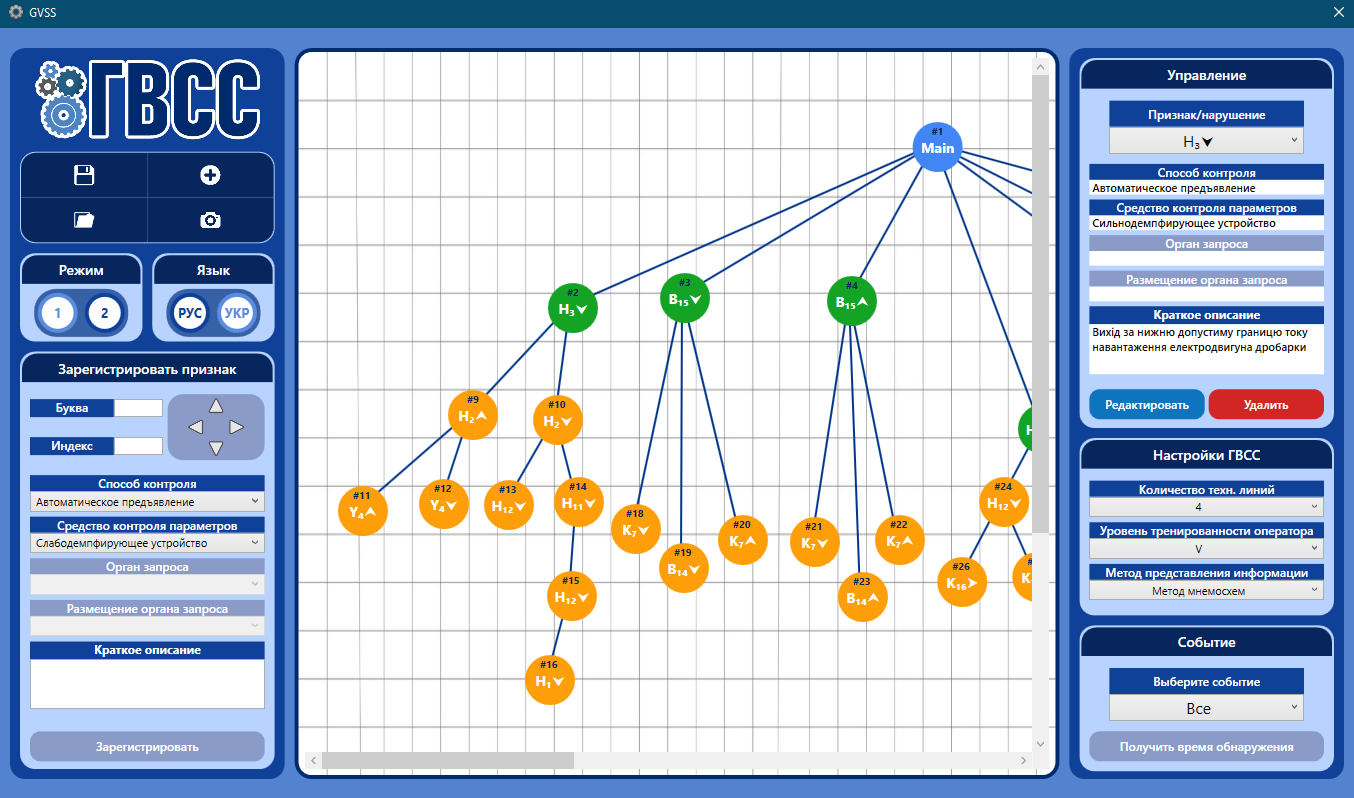
 Рис 5.8 – Процес зміни мови інтерфейсу

Рис 5.9 – Змінена мова інтерфейсу

***5.2.7 Змінити режим ГВСС***

Треба зазначити окремо, що програма має два режими роботи. Перший режим: «Порушення – унікальні і реєструються окремо». Це означає, що для реєстрації порушень призначений спеціальний блок на формі додавання вершини і те, що користувач не може використовувати порушення як ознаки до інших порушень. Другий режим: «Порушення можуть бути ознаками інших порушень». Це означає, що порушення можуть бути ознаками так само як і ознаки порушеннями, тому обидві ці сутності реєструються в одному місці, а саме у блоці реєстрації ознак на головній формі. Зміна режиму з другого на перший може призвести до втрати даних: деякі вершини, які не відповідають критеріям нового режиму – будуть видалені.

Щоб змінити режим треба обрати необхідний режим у блоці вибору режиму на головній формі програми.

***5.2.8 Змінити налаштування ГВСС***

У програмі є три опції ГВСС доступні до зміни: «Кількість технологічних ліній», «Рівень тренованості оператора», «Спосіб представлення інформації». Змінити ці параметри ви можете за допомогою вибору необхідних параметрів у полях з випадаючими списками блоку налаштувань ГВСС на головній формі програми.

***5.2.9 Зареєструвати ознаку***

Щоб зареєструвати ознаку необхідні заповнити всі поля, які відповідають за властивості ознаки, у блоці реєстрації на головній формі і натиснути кнопку «Зареєструвати».

***5.2.10 Додати вершину на робочу область***

Щоб додати вершину необхідно натиснути на необхідну точку на робочій області і далі у діалоговому вікні «Додати вершину» вибрати всі необхідні параметри нової вершини і натиснути кнопку «Додати».

***5.2.11 Переміщення вершини***

Щоб переміщувати вершину по робочій області необхідно натиснути на вершину лівою кнопкою миші і утримуючи її переміщувати її.

***5.2.12 Редагувати ознаку/вершину***

Щоб редагувати ознаку необхідно обрати ознаку у блоці управління ознаками на головній формі або у обрати таку функції у контекстному меню робочої області, і у діалоговому вікні «Змінити ознаку» по-новому визначити параметри ознаки і натиснути «Змінити».

***5.2.13 Переглянути інформацію про вершину/ознаку***

Щоб переглянути інформацію про вершину або ознаку необхідно відповідно натиснути на вершину, що знаходиться на робочій області, правою кнопкою миші, і обрати відповідну функцію у контекстному меню або обрати цю ознаку у полі з випадаючим списком у блоці управління ознаками на головній формі. Інформацію ви побачите у блоці управління ознаками на головній формі.

***5.2.14 Видалити ознаку***

Щоб видалити ознаку необхідно обрати цю ознаку у полі з випадаючим списком у блоці управління ознаками на головній формі і натиснути «Видалити». Будуть видалені всі вершини, які містять цю ознаку і всі їх дочірні вершини.

***5.2.15 Видалити вершину***

Щоб видалити вершину необхідно натиснути на вершину, що знаходиться на робочій області, правою кнопкою миші, і натиснути «Видалити». Буде видалена відповідна вершина і всі її дочірні вершини.

***5.2.16 Отримати час виявлення порушення***

Щоб отримати час виявлення порушення необхідно спочатку визначити чи завантажити структуру ГВСС, потім обрати необхідну подію у блоці «Події» і натиснути на кнопку «Отримати час виявлення». В результаті буде відкрите відповідне вікно з результатами. Весь процес відображений на рис. 5.10.

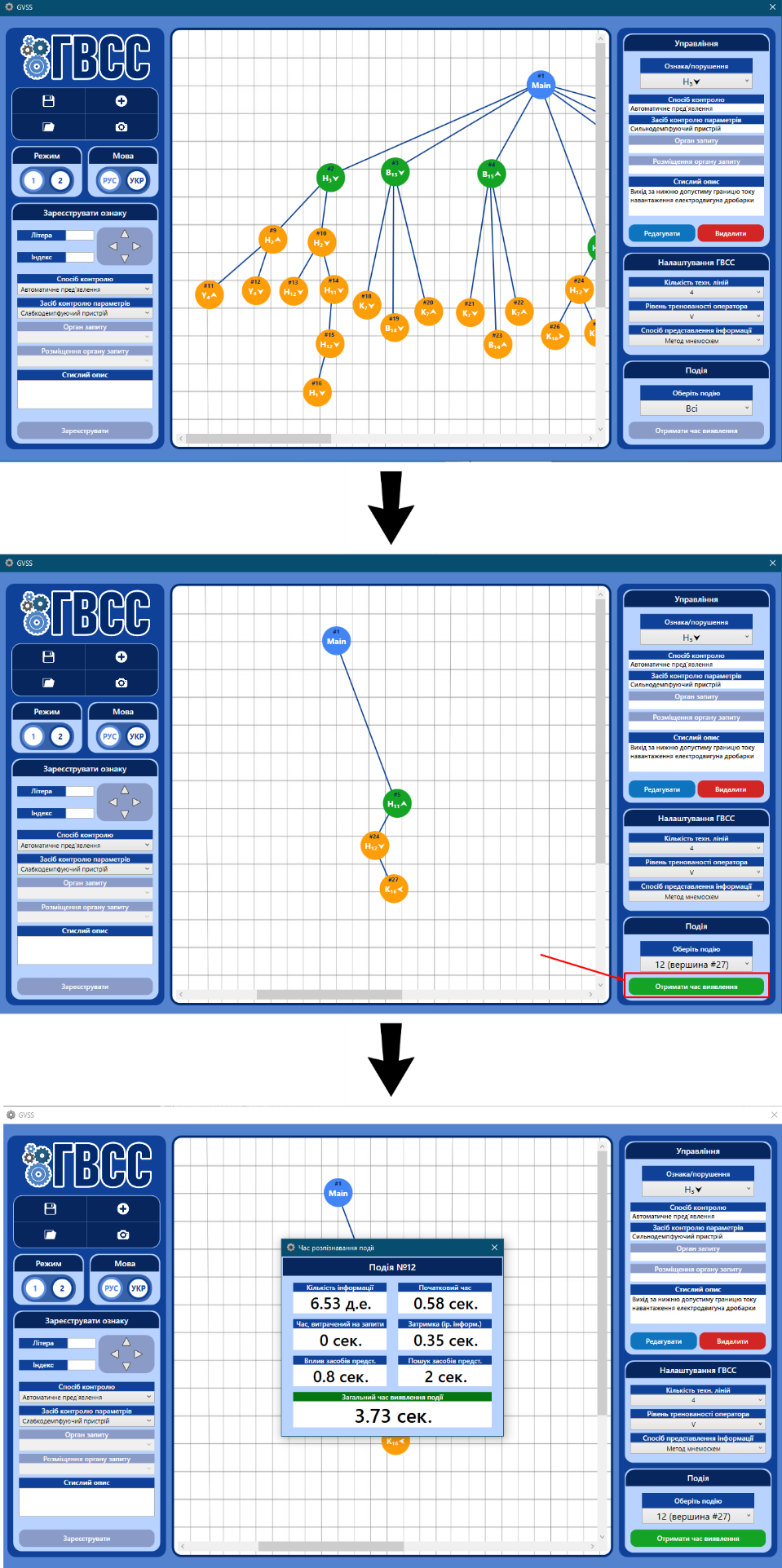


Рис 5.10 – Процес отримання часу виявлення певної події

5.3 Повідомлення оператору

Всі повідомлення, що може отримати оператор під час роботи з програмним забезпеченням описані у таблиці 5.1.

Таблиця 5.1 – Повідомлення оператору

|  |  |  |
| --- | --- | --- |
| Текст | Причина появи | Подальші дії |
| Ім’я недоступне | При реєстрації події користувач визначає ім’я для ознаки і ознака з таким ім’ям вже є у системі. | Визначити нове ім’я |
| Незаповнені поля | При реєстрації ознаки ви не заповнюєте всі необхідні поля і вам не дозволено зареєструвати таку ознаку | Заповнити всі поля |
| Неможливо відкрити файл | При спробі відкрити файл, внутрішня структура якого є пошкодженою. | Відкрити інший файл |
| Неможливо додати вершину у піддерево | Виникає при спробі додати вершину у піддерево структури ГВСС, якщо у піддереві вже є така ж ознака | Додати іншу ознаку |

Висновки

В ході виконання роботи було проведено аналіз технологій створення програмного забезпечення для ОС Windows, проведено аналіз предметної області, а також вже наявних програм-аналогів, створено технічне завдання з переліком вимог до проекту.

Після етапу проектування, виконанного з урахуванням усіх вимог, висунутих у технічному завданні, було проведено реалізацію Desktop-застосунку для захоплення та редагування зображень з екрану.

Готовий програмний продукт відповідає усім вимогам, поставленим у технічному завданні. Отримане програмне забезпечення є мультифункціональним і задовольняє всі потреби користувачів, які можуть виникнути в ході роботи.

Перелік посилань

1. Представление информации оператору (исследование деятельности человека – оператора производственных процессов / Галактіонов А. І. Москва «Энергия», 1969. 136 с. з іл.

2. Алгоритм пошуку в глибину – Вікіпедія [Електронний ресурс] – Режим доступу: <https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA_%D1%83_%D0%B3%D0%BB%D0%B8%D0%B1%D0%B8%D0%BD%D1%83>

3. Теорія інформації – Вікіпедія [Електронний ресурс] - <https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97>

4. Event-driven architecture (EDA) – Wikipedia [Електронний ресурс] – Режим доступу:

[https://en.wikipedia.org/wiki/Event-driven\_architecture#:~:text=Event%2Ddriven%20architecture%20(EDA),sale%22%20to%20%22sold%22.](https://en.wikipedia.org/wiki/Event-driven_architecture%23:~:text=Event%2Ddriven%20architecture%20(EDA),sale%22%20to%20%22sold%22.)

5. Operating systems market share – NetMarketShare [Електронний ресурс] – Режим доступу: <https://netmarketshare.com/operating-system-market-share.aspx?options=%7B%22filter%22%3A%7B%22%24and%22%3A%5B%7B%22deviceType%22%3A%7B%22%24in%22%3A%5B%22Desktop%2Flaptop%22%5D%7D%7D%5D%7D%2C%22dateLabel%22%3A%22Custom%22%2C%22attributes%22%3A%22share%22%2C%22group%22%3A%22platformVersion%22%2C%22sort%22%3A%7B%22share%22%3A-1%7D%2C%22id%22%3A%22platformsDesktopVersions%22%2C%22dateInterval%22%3A%22Monthly%22%2C%22dateStart%22%3A%222019-01%22%2C%22dateEnd%22%3A%222019-11%22%2C%22segments%22%3A%22-1000%22%7D>

6. Garbage collector (computer science) – Вікіпедія [Електронний ресурс] – Режим доступу: <https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)>

7. Human factors and ergonomics – Вікіпедія [Електронний ресурс] – Режим доступу: <https://en.wikipedia.org/wiki/Human_factors_and_ergonomics>

8. Представление информации оператору (исследование деятельности человека – оператора производственных процессов. с. 95

Додаток А. Технічне завдання

А.1 Підстава для розробки

Підставою для розробки служить завдання на курсовий проект з дисципліни «Операційні системи» на тему «Розробка програми автоматизації визначення часу виявлення порушення технологічного процесу на виробництві (ГВСС)».

А.2 Призначення розробки

Дане програмне забезпечення призначене для графічного моделювання системи порушень на виробництві і автоматичного визначення часу виявлення порушення оператором.

А.3 Вимог до програми або програмного продукту

А.3.1 Функціональні вимоги

Розроблюване програмне забезпечення має надавати користувачеві наступні можливості:

* графічне моделювання структури ГВСС;
* створення нової вершини у заданій точці робочої області;
* відображення зв’язків між вершинами
* переміщення вершин ГВСС в межах робочої області;
* видалення вершини;
* два типи вершин: вершини-порушення і вершини-ознаки;
* реєстрація ознак;
* редагування ознак;
* видалення ознак;
* два режими роботи: порушення є унікальними і реєструються окремо; порушення можуть бути ознаками інших порушень, реєструються як ознаки;
* відображення результатів отриманих за допомогою MathLab;
* автоматичний розрахунок часу виявлення порушення;
* збереження поточної ГВСС;
* завантаження ГВСС з файлу;
* можливість експортувати структуру ГВСС (.png файл)

Вхідними даними, необхідними для функціонування програмного продукту, є структура ГВСС, яка представляє собою граф, де вершинами є порушення і їх ознаки. Кожна ознака має такі властивості: унікальна назва, спосіб контролю, засіб контролю параметрів/засіб запиту, орган запиту, метод розміщення запиту на мнемосхемі, стислий опис. Порушення мають бути визначені за допомогою таких самих параметрів, а також для них повинна бути відома частота виходу з ладу. Додатково повинні бути визначені такі параметри для ГВСС: кількість технологічних ліній, рівень тренованості оператора, спосіб представлення інформації. Вихідними даними є орієнтовний час виявлення порушення оператором, а також: кількість інформації в події (відповідно до теорії інформації вимірюється у двійкових одиницях), початковий час, час витрачений на запити, затримка (залежність від кількості іррелевантної інформації), вплив типу засобів представлення (сек.), час витрачений на пошук засобів представлення.

А.3.2 Вимоги до надійності

Розроблюване програмне забезпечення повинно проводити обробку виключних ситуацій, повідомляючи оператора про некоректність наданих їм вхідних даних або проведених операцій, обмежувати оператора у можливості введення вхідних даних (явно вказувати тип даних, який очікується, проводити їх валідацію).

А.3.3 Умови експлуатації

Умови експлуатації повинні відповідати вимогам, які пред’являються до технічних засобів. Показники температури, відносної вологості і швидкості руху повітря повинні відповідати оптимальним і допустимим значенням в робочій зоні виробничих приміщень.

А.3.3 Вимоги до складу та параметрів технічних засобів

Серед технічних засобів повинні бути наявні:

* ОС Windows;
* вільне місце на жорсткому диску, достатнє для встановлення програми;
* ОЗУ 512 мб або більше;
* необхідне для функціонування встановлення ПЗ;
* монітору;
* маніпулятори вводу (клавіатура, миша).

А.3.4 Вимоги до інформаційної та програмної сумісності

Розроблюване ПЗ повинно підтримувати спільний режим роботи з ОС Windows. Програма повинна бути підготовлена до можливої модернізації.

А.3.5 Вимоги до програмної документації

Включно з розроблюваним програмним забезпеченням повинно бути розроблено керівництва програміста та оператора, а також коротка довідка по роботі даного ПЗ.

А.4 Стадії та етапи розробки

Розробка даного програмного забезпечення повинна включати наступні етапи:

* аналіз аналогів, формалізація вимог до ПЗ;
* проектування, опис архітектури, вибір технічних і програмних засобів реалізації;
* реалізація;
* тестування.

А.5 Порядок приймання і тестування

Тестування даного ПЗ проводиться у вигляді виконання усіх можливих сценаріїв роботи оператора.

Дане ПЗ вважається прийнятним, якщо воно задовольняє усім висунутим вимогам до нього і успішно пройшло етап тестування.

Додаток Б. Текст програми

Б.1 Текст Desktop-застосунку

\*не містить файлів розмітки графічного інтерфейсу .xaml\*

***BinarySerialization.cs:***

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace gvss\_project

{

public static class BinarySerialization

{

public static void WriteToBinaryFile<T>(string filePath, T objectToWrite, bool append = false)

{

using (Stream stream = File.Open(filePath, append ? FileMode.Append : FileMode.Create))

{

var binaryFormatter = new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();

binaryFormatter.Serialize(stream, objectToWrite);

}

}

public static T ReadFromBinaryFile<T>(string filePath)

{

using (Stream stream = File.Open(filePath, FileMode.Open))

{

var binaryFormatter = new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();

return (T)binaryFormatter.Deserialize(stream);

}

}

}

}

***Computations.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace gvss\_project

{

public static class Computations

{

public static string getCapLetAndIndex(char capLetter, int index)

{

if (Char.IsLetter(capLetter))

{

capLetter = Char.ToUpper(capLetter);

string str = capLetter.ToString();

foreach (char c in index.ToString())

{

str += ((char)(0x2080 + Int32.Parse(c.ToString()))).ToString();

}

return str;

}

else

{

throw new InvalidArgument();

}

}

public static string getUniqueCode(char capLetter, uint index, SensorArrowSign arrowSign)

{

int u = 0x02b9b + (int)arrowSign;

string str = capLetter.ToString();

foreach (char c in index.ToString())

{

str += ((char)(0x2080 + Int32.Parse(c.ToString()))).ToString();

}

str += ((char)u).ToString();

return str;

}

}

}

***EditDialog.xaml.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace gvss\_project

{

/// <summary>

/// Interaction logic for EditDialog.xaml

/// </summary>

///

public partial class EditDialog : Window

{

public SensorControlMethod ControlMethod { get; private set; }

public SensorControlAgent ControlAgent { get; private set; }

public SensorRequestBodies RequestBody { get; set; }

public SensorRequestPlacementMethod RequestPlacementMethod { get; set; }

public string Description { get; private set; }

public readonly string UniqueCode;

private SensorControlMethod ControlMethodDef { get; set; }

private SensorControlAgent ControlAgentDef { get; set; }

private SensorRequestBodies RequestBodyDef { get; set; }

private SensorRequestPlacementMethod RequestPlacementMethodDef { get; set; }

private string DescriptionDef { get; set; }

public EditDialog(Sensor sensor)

{

InitializeComponent();

LocalizeWindow();

UniqueCode = sensor.UniqueCode;

ControlAgentDef = sensor.ControlAgent;

ControlMethodDef = sensor.ControlMethod;

DescriptionDef = sensor.Description;

RequestPlacementMethodDef = sensor.RequestPlacementMethod;

RequestBodyDef = sensor.RequestBody;

InputCorrector.InitializeRelatedToSensorCBs(controlMethodCB, controlAgentCB, requestBodyCB, requestPlacementMethodCB);

uniqueCodeLabel.Content = sensor.UniqueCode;

foreach (ComboBoxItem cbi in controlMethodCB.Items) {

if((SensorControlMethod)(cbi.Tag) == sensor.ControlMethod){

controlMethodCB.SelectedItem = cbi;

break;

}

}

foreach (ComboBoxItem cbi in controlAgentCB.Items){

if ((SensorControlAgent)(cbi.Tag) == sensor.ControlAgent){

controlAgentCB.SelectedItem = cbi;

break;

}

}

if (sensor.ControlMethod == SensorControlMethod.Request) {

foreach (ComboBoxItem cbi in requestBodyCB.Items)

{

if ((SensorRequestBodies)(cbi.Tag) == sensor.RequestBody)

{

requestBodyCB.SelectedItem = cbi;

break;

}

}

foreach (ComboBoxItem cbi in requestPlacementMethodCB.Items)

{

if ((SensorRequestPlacementMethod)(cbi.Tag) == sensor.RequestPlacementMethod)

{

requestPlacementMethodCB.SelectedItem = cbi;

break;

}

}

}

descriptionTB.Text = sensor.Description;

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

ControlMethod = (SensorControlMethod)(((ComboBoxItem)(controlMethodCB.SelectedItem)).Tag);

ControlAgent = (SensorControlAgent)(((ComboBoxItem)(controlAgentCB.SelectedItem)).Tag);

Description = descriptionTB.Text;

RequestBody = (SensorRequestBodies)(((ComboBoxItem)(requestBodyCB.SelectedItem)).Tag);

RequestPlacementMethod = (SensorRequestPlacementMethod)(((ComboBoxItem)(requestPlacementMethodCB.SelectedItem)).Tag);

this.DialogResult = true;

this.Close();

}

private void controlMethodCB\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

InputCorrector.ControlMethodCB\_SelectionChanged(sender, e,

controlAgentTBlock,

requestBodyCB,

requestBodyBorder,

requestPlacementMethodCB,

requestPlacementMethodBorder);

setButton();

}

private void requestBodyCB\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

InputCorrector.requestBodyCB\_SelectionChanged(sender, e, requestPlacementMethodCB);

setButton();

}

private void cb\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

setButton();

}

private void setButton() {

if (controlMethodCB.SelectedItem != null &&

controlAgentCB.SelectedItem != null &&

requestBodyCB.SelectedItem != null &&

requestPlacementMethodCB.SelectedItem != null) {

bool isControlMethodNew = ((SensorControlMethod)((ComboBoxItem)(controlMethodCB.SelectedItem)).Tag) != ControlMethodDef;

bool isControlAgentNew = ((SensorControlAgent)((ComboBoxItem)(controlAgentCB.SelectedItem)).Tag) != ControlAgentDef;

bool isRequestBodyNew = ((SensorRequestBodies)((ComboBoxItem)(requestBodyCB.SelectedItem)).Tag) != RequestBodyDef;

bool isRequestPlacementMethodNew = ((SensorRequestPlacementMethod)((ComboBoxItem)(requestPlacementMethodCB.SelectedItem)).Tag) != RequestPlacementMethodDef;

bool isDescriptionNew = DescriptionDef != descriptionTB.Text;

if ((isControlMethodNew || isControlAgentNew || isRequestBodyNew || isRequestPlacementMethodNew || isDescriptionNew) && descriptionTB.Text.Length != 0)

{

changeButton.IsEnabled = true;

}

else

{

changeButton.IsEnabled = false;

}

}

}

private void LocalizeWindow()

{

this.Title = DefaultSettings.getWindowTitleText(WindowTitles.Edit);

changeButton.Content = DefaultSettings.getButtonText(Buttons.Change);

controlMethodLabel.Text = DefaultSettings.getLabelText(Labels.ControlMethod);

requestBodyLabel.Text = DefaultSettings.getLabelText(Labels.RequestBody);

requestPlacementMethodLabel.Text = DefaultSettings.getLabelText(Labels.RequestPlacementMethod);

descriptionLabel.Text = DefaultSettings.getLabelText(Labels.Description);

}

private void descriptionTB\_TextChanged(object sender, TextChangedEventArgs e)

{

setButton();

}

}

}

***Enumerations.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace gvss\_project

{

public enum LanguageMode { Russian, Ukrainian};

public enum Labels { LanguageMode, RegisterAuxiliaryParam, Letter, Index,

ControlMethod, ControlAgentAuto, ControlAgentRequest, RequestBody, RequestPlacementMethod,

Description, Managment, AuxiliaryParamAndViolation,

GVSSSettings, NumOfTechLines, OperatorLevel,

InformationRepresentationMethod, Event, ChooseEvent,

InformationQuantity, OriginalTime, RequestTime, IrrelevantInformationTime,

AgentTimeInfluence, FindControlAgentTime, TotalTime, AddViolation,

Violation, ChooseAction, FailureRate, AuxiliaryParam, AddAuxiliaryParamToVertex};

public enum Buttons { Register, Edit, Delete, GetTime, Add, Change};

public enum ToolTips { ViolationsAreUnique, ViolationsCanBeParams, CreateNewGVSS, LoadGVSS, SaveGVSS, SavePhoto};

public enum ErrorMessages { WhileAddRepetative, WhileGVSSFileCorrupted, WhileASVFileCorrupted, TheNameExistsAlready, NotAllFieldsAreFilled};

public enum MessageBoxMessages { DoYouWantToSaveGVSSBeforeQuit, DeleteVertex, SaveGVSSBeforeLoad, ChangeMode, FullDeleteParam};

public enum ContextMenuHeaders { DeleteVertex, Edit};

public enum WindowTitles { ErrorMB, WarningMB, Edit, Add, EventDetectionTime, MathLABResults};

public enum Words { All, EventID, AddViolation, AddAuxiliaryParamToVertex, EventTime};

//mat lab ne zabitb

public enum OperatorLevel { Second = 0, Third, Forth, Fifth/\*, Sixth\*/ };

public enum SensorControlMethod { Auto = 0, Request, /\*Forecasting,\*/ None };

public enum SensorControlAgent { WeakDampedDevice = 0, HighlyDampedDevice, DigitalThreeDigitIndicator, SignalingDevice, None }

public enum SensorArrowSign { None = 0, Left, Up, Right, Down };

public enum SensorRequestPlacementMethod { UnorganizedMethod = 0, MnemonicDiagram, Combination, BlockDiagramMethod, Separately, GroupMethod, None };

public enum SensorRequestBodies { ToggleSwitch = 0, Button, Dialer, KeyboardDevice, None }

public enum InformationRepresentationMethod { /\*UnorganizedMethod = 0,\*/ MnemonicDiagram, GroupMethod, /\*AutoMethod,\*/ BlockDiagramMethod, /\*GeneralizedMethod,\*/ None }

}

***Event.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace gvss\_project

{

public class Event

{

public class VertexAndNumOfSubverticies

{

public int VertexID { get; private set; }

public int SubverticiesNumber { get; private set; }

public VertexAndNumOfSubverticies(int vertexID, int subvertexesNumber)

{

if (vertexID <= 0 || subvertexesNumber < 0)

{

throw new InvalidArgument();

}

VertexID = vertexID;

SubverticiesNumber = subvertexesNumber;

}

public VertexAndNumOfSubverticies()

{

VertexID = -1;

SubverticiesNumber = -1;

}

public bool isNull()

{

if (VertexID == -1 && SubverticiesNumber == -1)

return true;

return false;

}

}

private List<VertexAndNumOfSubverticies> vertices;

private uint numOfTechLines;

private float violationInformationQuantity;

private uint sensorsCount;

private uint dampedDeviceTotal;

private OperatorLevel operatorLevel;

private InformationRepresentationMethod informationRepresentationMethod;

public List<uint> VertexPath { get; private set; }

public List<Sensor> requestSensors;

public List<Sensor> autoSensors;

public uint ID { get; private set; }

private VertexAndNumOfSubverticies isVertexAlreadyAdded(int vID)

{

foreach (VertexAndNumOfSubverticies vertex in vertices)

{

if (vertex.VertexID == vID)

{

return vertex;

}

}

throw new NotFoundException();

}

private double Log(double value)

{

return -Math.Log(value, 2);

}

private double Round(double value, int decimalNumbers = 2) {

return Math.Round(value, decimalNumbers);

}

public Event(uint id, List<uint> vertexPath, uint numOfTechLines, uint sensorsCount, float violationInformationQuantity, OperatorLevel operatorLevel, uint dampedDeviceTotal, InformationRepresentationMethod informationRepresentationMethod)

{

vertices = new List<VertexAndNumOfSubverticies>();

requestSensors = new List<Sensor>();

autoSensors = new List<Sensor>();

this.numOfTechLines = numOfTechLines;

this.operatorLevel = operatorLevel;

this.sensorsCount = sensorsCount;

this.violationInformationQuantity = violationInformationQuantity;

this.dampedDeviceTotal = dampedDeviceTotal;

this.informationRepresentationMethod = informationRepresentationMethod;

ID = id;

VertexPath = vertexPath;

}

public void addRequestSensor(Sensor s)

{

if (s.ControlMethod == SensorControlMethod.Request)

requestSensors.Add(s);

}

public void addAutoSensor(Sensor s) {

if (s.ControlMethod == SensorControlMethod.Auto)

autoSensors.Add(s);

}

public void addVertexAndSubverticiesNumber(VertexAndNumOfSubverticies node)

{

VertexAndNumOfSubverticies v = new VertexAndNumOfSubverticies();

try

{

v = isVertexAlreadyAdded(node.VertexID);

throw new InvalidArgument();

}

catch (NotFoundException)

{

vertices.Add(node);

}

}

public VertexAndNumOfSubverticies getVertexAndSubverticiesNumber(int vID)

{

return isVertexAlreadyAdded(vID);

}

public double getInformationDifficulty()

{

double techLineInfo = Log(1.0 / numOfTechLines);

double sum = techLineInfo + Log(violationInformationQuantity);

foreach (VertexAndNumOfSubverticies v in vertices)

{

sum -= Log((float)v.SubverticiesNumber);

}

return Round(sum);

}

public double getOriginalTime()

{

double informationDifficulty = getInformationDifficulty();

(double, double) ab = DefaultSettings.OperatorCoefs[operatorLevel];

double time = ab.Item1 + ab.Item2 \* informationDifficulty;

return Round(DefaultSettings.IndformationRepresentationMethodCoefs[informationRepresentationMethod] \* time);

}

public double getRequestTime() {

double time = 0;

double t1, t2;

foreach (Sensor s in requestSensors) {

t1 = DefaultSettings.RequestTime[(s.RequestBody, s.RequestPlacementMethod)];

t2 = DefaultSettings.ControlAgentsTime[s.ControlAgent];

if (s.RequestBody == SensorRequestBodies.Dialer || s.RequestBody == SensorRequestBodies.KeyboardDevice)

{

t1 \*= s.NumOfRequestInputKeys;

}

time += t1 + t2;

}

return Round(time);

}

public double getIrrelevantTime() {

return Math.Round(0.2\*(sensorsCount \* 0.075),2,MidpointRounding.AwayFromZero);

}

public double getAgentTimeInfluence() {

double time = 0;

foreach (Sensor s in autoSensors) {

time += DefaultSettings.ControlAgentsTime[s.ControlAgent];

}

return Round(time);

}

public double getFindControlAgentTime() {

double time = 0;

uint dampedDevicesInEvent = 0;

foreach (Sensor s in requestSensors) {

if (s.ControlAgent == SensorControlAgent.HighlyDampedDevice || s.ControlAgent == SensorControlAgent.WeakDampedDevice)

dampedDevicesInEvent++;

}

foreach (Sensor s in autoSensors)

{

if (s.ControlAgent == SensorControlAgent.HighlyDampedDevice || s.ControlAgent == SensorControlAgent.WeakDampedDevice)

dampedDevicesInEvent++;

}

if (dampedDevicesInEvent != 0) {

time = 0.02 \* dampedDeviceTotal + 1.7 + (0.1 \* (dampedDevicesInEvent - 1));

}

return Round(time, 1);

}

}

}

***Exceptions.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace gvss\_project

{

class InvalidArgument : Exception

{

public InvalidArgument() : base("Invalid argument!") { }

}

class InvalidSensor : Exception {

public InvalidSensor() : base("Invalid sensor") { }

}

class UnexpectedError : Exception {

public UnexpectedError() : base("UnexpectedError") { }

}

class NotFoundException: Exception

{

public NotFoundException() : base("Not found") { }

}

class SensorRepeatsException: Exception {

public SensorRepeatsException() : base("The sensor repeats") { }

}

}

***GVSS.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace gvss\_project

{

[Serializable]

public enum GVSSMode { UniqueViolations, RepetativeViolations }

[Serializable]

public class VertexNode

{

public uint VertexID { get; private set; }

public string UniqueCode { get; private set; }

public VertexNode(uint vID, string uniqueCode) {

VertexID = vID;

UniqueCode = uniqueCode;

}

}

[Serializable]

public class GVSS

{

[Serializable]

private enum VertexType { Main, Violation, AuxiliaryParam };

[Serializable]

private class Vertex

{

[Serializable]

private class Edge

{

public float EdgeWeight { get; set; }

public uint ConnectedVertexID { get; set; }

public Edge(uint conVerID, float edgeWeight)

{

EdgeWeight = edgeWeight;

ConnectedVertexID = conVerID;

}

}

//properties and fields

public Point CenterPos { get; set; }

public uint ID { get; private set; }

public VertexType Type { get; private set; }

public int AdjVCount {

get {

return edges.Count;

}

}

public uint Level { get; private set; }

private List<Edge> edges;

//additional information corresponding to the task

public Vertex() { }

public Vertex(uint vID, VertexType type, uint level, Point centerPos)

{

ID = vID;

edges = new List<Edge>();

Type = type;

Level = level;

CenterPos = centerPos;

}

public void addEdge(uint anotherVertexID, float edgeWeight)

{

if (anotherVertexID != this.ID)

edges.Add(new Edge(anotherVertexID, edgeWeight));

}

public void deleteEdge(uint anotherVertexID)

{

if (anotherVertexID != this.ID)

{

foreach (Edge edge in edges)

{

if (edge.ConnectedVertexID == anotherVertexID)

{

edges.Remove(edge);

break;

}

}

}

}

public void setEdgeWeight(uint anotherVertexID, float edgeWeight)

{

foreach (Edge edge in edges)

{

if (edge.ConnectedVertexID == anotherVertexID)

{

edges[edges.IndexOf(edge)].EdgeWeight = edgeWeight;

break;

}

}

}

public List<uint> getAdjacentVertexes()

{

List<uint> adjVs = new List<uint>();

foreach (Edge edge in edges)

{

adjVs.Add(edge.ConnectedVertexID);

}

return adjVs;

}

public bool isConnectedWith(uint anotherVertexID)

{

if (anotherVertexID != this.ID)

{

foreach (Edge edge in edges)

{

if (anotherVertexID == edge.ConnectedVertexID)

{

return true;

}

}

}

return false;

}

public float getEdgeWeight(uint anotherVertex)

{

foreach (Edge edge in edges)

{

if (edge.ConnectedVertexID == anotherVertex)

{

return edge.EdgeWeight;

}

}

return -1;

}

public uint getParentVertex() {

if (ID != 1)

{

return edges[0].ConnectedVertexID;

}

else

throw new UnexpectedError();

}

}

[Serializable]

private struct Correspondence

{

public string SensorCode { get; private set; }

public uint VertexID { get; private set; }

public uint VertexIndex { get; private set; }

public Correspondence(string sensorCode, uint vertexID, uint vertexIndex)

{

SensorCode = sensorCode;

VertexID = vertexID;

VertexIndex = vertexIndex;

}

}

//fields and properties

private List<Vertex> vertexes;

private List<(Sensor, uint)> violations;

private List<Sensor> auxiliaryParams;

private List<uint> deletedVertexesList;

private List<Correspondence> correspondences;

private uint vertexID\_counter;

private readonly uint mainVertexID;

private GVSSMode mode;

public string ProductionAreaName { get; private set; }

public int Size { get { return vertexes.Count; } }

//private methods

private Vertex getVertex(uint vID) {

foreach (Vertex v in vertexes)

{

if (v.ID == vID)

return v;

}

throw new NotFoundException();

}

private uint getSensorsCount() {

if (mode == GVSSMode.RepetativeViolations)

{

return (uint)auxiliaryParams.Count;

}

else if (mode == GVSSMode.UniqueViolations)

{

return (uint)(auxiliaryParams.Count + violations.Count);

}

else throw new UnexpectedError();

}

private bool isViolation(uint vID) {

if (getVertex(vID).Type == VertexType.Violation)

{

return true;

}

else {

return false;

}

}

private uint addVertex(VertexType type, uint level, Point centerPos) {

uint id = ++vertexID\_counter;

vertexes.Add(new Vertex(id, type, level, centerPos));

return id;

}

private void deleteVertex(uint vertexID) {//

if (isVertexInGraph(vertexID))

{

List<uint> connectedVertexes = new List<uint>();

foreach(Vertex v in vertexes)

{

if (v.ID == vertexID)

{

connectedVertexes = v.getAdjacentVertexes();

deletedVertexesList.Add(v.ID);

vertexes.Remove(v);

break;

}

}

if (connectedVertexes.Count > 0) {

foreach (Vertex v1 in vertexes)

{

foreach (uint v2 in connectedVertexes)

{

if (v1.ID == v2)

{

v1.deleteEdge(vertexID);

}

}

}

}

}

}

private void addEdge(uint firstV, uint secondV, float edgeWeight) {

if (isVertexInGraph(firstV) && isVertexInGraph(secondV) && !doesTheEdgeExist(firstV, secondV))

{

uint counter = 0;

foreach (Vertex v in vertexes)

{

if (v.ID == firstV)

{

v.addEdge(secondV, edgeWeight);

counter++;

}

else if (v.ID == secondV)

{

v.addEdge(firstV, edgeWeight);

counter++;

}

if (counter == 2)

break;

}

}

}

private bool isVertexInGraph(uint vID) {

foreach (Vertex vertex in vertexes)

{

if (vertex.ID == vID)

return true;

}

return false;

}

private bool doesTheEdgeExist(uint firstV, uint secondV) {

foreach(Vertex v in vertexes)

{

if ((v.ID == firstV && v.isConnectedWith(secondV))||

(v.ID == secondV && v.isConnectedWith(firstV)))

{

return true;

}

}

return false;

}

private Sensor registerSensor(char captionLetter, uint index, string desription, SensorControlMethod controlMethod, SensorControlAgent controlAgent, SensorArrowSign arrowSign, SensorRequestBodies requestBody, SensorRequestPlacementMethod requestPlacementMethod) {

Sensor s = new Sensor();

s.CapitalLetter = captionLetter;

s.Index = (int)index;

s.Description = desription;

s.ControlMethod = controlMethod;

s.ArrowSign = arrowSign;

s.ControlAgent = controlAgent;

s.RequestBody = requestBody;

s.RequestPlacementMethod = requestPlacementMethod;

if (s.isValid() && isSensorNameAvailable(index, s.CapitalLetter, s.ArrowSign))

return s;

else throw new InvalidSensor();

}

private Sensor getAuxiliaryParam(string code) {

foreach (Sensor s in auxiliaryParams) {

if (s.UniqueCode == code)

return s;

}

throw new NotFoundException();

}

private Correspondence getCorrespondence(uint vID) {

foreach (Correspondence c in correspondences) {

if (c.VertexID == vID)

return c;

}

throw new NotFoundException();

}

private bool doesTheAuxiliaryParamExist(string code) {

foreach (Sensor s in auxiliaryParams)

{

if (s.UniqueCode == code)

return true;

}

return false;

}

private bool doesTheSensorRepeat(string sensorCode, uint connectedVertexID) {

List<uint> path = getWayUp(connectedVertexID);

foreach (uint v in path) {

if (getUniqueCode(v) == sensorCode)

return true;

}

List<uint> vertexesAround = getAdjacentVertexes(connectedVertexID);

uint level = getLevel(connectedVertexID);

foreach (uint v in vertexesAround) {

if (getLevel(v) > level && sensorCode == getUniqueCode(v))

return true;

}

return false;

}

private void deleteCorrespondence(uint vID) {

foreach(Correspondence c in correspondences){

if (c.VertexID == vID) {

correspondences.Remove(c);

return;

}

}

throw new NotFoundException();

}

private bool doesTheViolationExist(string uniqueCode) {

foreach ((Sensor,uint) v in violations)

{

if (v.Item1.UniqueCode == uniqueCode) {

return true;

}

}

return false;

}

private void deleteViolation(uint vID) {

foreach ((Sensor, uint) node in violations) {

if (node.Item2 == vID) {

violations.Remove(node);

return;

}

}

throw new NotFoundException();

}

private uint getLevel(uint vID) {

foreach (Vertex v in vertexes) {

if (v.ID == vID) {

return v.Level;

}

}

throw new NotFoundException();

}

private void deleteAuxiliary(string sensorCode) {

if (doesTheAuxiliaryParamExist(sensorCode)) {

foreach (Sensor s in auxiliaryParams) {

if (s.UniqueCode == sensorCode) {

auxiliaryParams.Remove(s);

break;

}

}

}

}

private void deleteProc(uint curVertexID) {

List<uint> adjV = getAdjacentVertexes(curVertexID);

foreach (uint v in adjV) {

if (getLevel(v) > getLevel(curVertexID)) {

deleteProc(v);

}

}

if (getVertex(curVertexID).Type == VertexType.Violation) {

string aP = getCorrespondence(curVertexID).SensorCode;

if (mode == GVSSMode.RepetativeViolations && !doesTheAuxiliaryParamExist(aP)) {

deleteAuxiliary(aP);

}

deleteViolation(curVertexID);

}

deleteCorrespondence(curVertexID);

deleteVertex(curVertexID);

return;

}

private List<uint> getWayUp(uint startVertexID) {

uint curVertex = startVertexID;

List<uint> path = new List<uint>();

bool completed = false;

while (!completed) {

path.Add(curVertex);

List<uint> adjVertexes = getAdjacentVertexes(curVertex);

uint level = getLevel(curVertex);

foreach (uint v in adjVertexes) {

if (level > getLevel(v)) {

if (v == mainVertexID)

completed = true;

else {

curVertex = v;

}

break;

}

}

}

return path;

}

private int getNumberVerticiesInSubtree(uint vID) {

if (isVertexInGraph(vID) && vID != mainVertexID)

{

uint parentID = getVertex(vID).getParentVertex();

uint parentLevel = getLevel(parentID);

List<uint> verticies = getAdjacentVertexes(parentID);

int counter = 0;

foreach (uint v in verticies) {

if (getLevel(v) > parentLevel) {

++counter;

}

}

return counter;

}

else {

throw new InvalidArgument();

}

}

private float getViolationInformationQuantity(uint vID, uint numOfTechLines) {

if (isViolation(vID))

{

float sum = 0;

foreach ((Sensor, uint) violation in violations) {

sum += getEdgeWeight(mainVertexID, violation.Item2);

}

return getEdgeWeight(mainVertexID, vID) / sum;

}

else

throw new InvalidArgument();

}

private uint getDampedDeviceTotal() {

uint count = 0;

foreach (Sensor s in auxiliaryParams) {

if (s.ControlAgent == SensorControlAgent.HighlyDampedDevice || s.ControlAgent == SensorControlAgent.WeakDampedDevice) {

count++;

}

}

if (mode == GVSSMode.UniqueViolations) {

foreach ((Sensor, uint) v in violations) {

if (v.Item1.ControlAgent == SensorControlAgent.HighlyDampedDevice || v.Item1.ControlAgent == SensorControlAgent.WeakDampedDevice) {

count++;

}

}

}

return count;

}

//public methods

public GVSS(Point mainCenterPos, GVSSMode mode = GVSSMode.RepetativeViolations)

{

this.mode = mode;

this.vertexes = new List<Vertex>();

this.violations = new List<(Sensor,uint)>();

this.auxiliaryParams = new List<Sensor>();

this.correspondences = new List<Correspondence>();

this.vertexID\_counter = 0;

this.ProductionAreaName = "Прозводственная система";

//creating main vertex

mainVertexID = this.addVertex(VertexType.Main, 0, mainCenterPos);

correspondences.Add(new Correspondence("Main", mainVertexID, 0));

}

public VertexNode addViolation(char captionLetter, uint index, string description, SensorControlMethod controlMethod, SensorControlAgent controlAgent, SensorArrowSign arrowSign, SensorRequestBodies requestBody, SensorRequestPlacementMethod requestPlacementMethod, Point centerPos, float probability) {

Sensor s = registerSensor(captionLetter, index, description, controlMethod, controlAgent, arrowSign, requestBody, requestPlacementMethod);

uint level = 1;

uint vID = addVertex(VertexType.Violation, level, centerPos);

violations.Add((s, vID));

if (mode == GVSSMode.RepetativeViolations)

auxiliaryParams.Add(s);

correspondences.Add(new Correspondence(s.UniqueCode, vID, (uint)(vertexes.Count - 1)));

addEdge(mainVertexID, vID, probability);

return new VertexNode(vID, s.UniqueCode);

}

public VertexNode addViolation(string auxiliaryCode, Point centerPos, float probability) {

if (mode == GVSSMode.RepetativeViolations)

{

if (doesTheAuxiliaryParamExist(auxiliaryCode) && !doesTheViolationExist(auxiliaryCode))

{

uint level = 1;

uint vID = addVertex(VertexType.Violation, level, centerPos);

Sensor s = getAuxiliaryParam(auxiliaryCode);

violations.Add((s, vID));

correspondences.Add(new Correspondence(s.UniqueCode, vID, (uint)(vertexes.Count - 1)));

addEdge(mainVertexID, vID, probability);

return new VertexNode(vID, s.UniqueCode);

}

else

{

throw new InvalidArgument();

}

}

else {

throw new UnexpectedError();

}

}

public string addAuxiliaryParam(char captionLetter, uint index, string description, SensorControlMethod controlMethod, SensorControlAgent controlAgent, SensorArrowSign arrowSign, SensorRequestBodies requestBody, SensorRequestPlacementMethod requestPlacementMethod) {

Sensor s = registerSensor(captionLetter, index, description, controlMethod, controlAgent, arrowSign,requestBody, requestPlacementMethod);

auxiliaryParams.Add(s);

return s.UniqueCode;

}

public VertexNode connect(uint upperVertex, string auxiliaryVertexCode, Point centerPos) {

if (upperVertex == mainVertexID

|| !isVertexInGraph(upperVertex)

|| !doesTheAuxiliaryParamExist(auxiliaryVertexCode))

throw new InvalidArgument();

else if (doesTheSensorRepeat(auxiliaryVertexCode, upperVertex))

throw new SensorRepeatsException();

Sensor s = getAuxiliaryParam(auxiliaryVertexCode);

uint level = vertexes[(int)getCorrespondence(upperVertex).VertexIndex].Level+1;

uint vID = addVertex(VertexType.AuxiliaryParam, level, centerPos);

correspondences.Add(new Correspondence(s.UniqueCode, vID, (uint)(vertexes.Count - 1)));

float probability = 1.0f; //count probability

addEdge(upperVertex, vID, probability);

return new VertexNode(vID,s.UniqueCode);

}

public List<uint> delete(uint vID) {

if (vID != mainVertexID && isVertexInGraph(vID)) {

deletedVertexesList = new List<uint>();

deleteProc(vID);

return deletedVertexesList;

}

else

throw new InvalidArgument();

}

public bool isSensorNameAvailable(uint index, char capitalLetter, SensorArrowSign arrowSign) {

Sensor s = new Sensor();

s.Index = (int)index;

s.CapitalLetter = capitalLetter;

s.ArrowSign = arrowSign;

foreach ((Sensor, uint) value in violations) {

if (value.Item1 == s)

return false;

}

foreach (Sensor value in auxiliaryParams) {

if (value == s)

return false;

}

return true;

}

public string getUniqueCode(uint vID) {

if (isVertexInGraph(vID)) {

foreach (Correspondence c in correspondences)

{

if (c.VertexID == vID)

return c.SensorCode;

}

throw new UnexpectedError();

}

else

throw new InvalidArgument();

}

public List<VertexNode> getVertexes(bool withoutMain = true)

{

List<VertexNode> vList = new List<VertexNode>();

foreach (Correspondence c in correspondences)

{

if (withoutMain && c.SensorCode == "Main") {

continue;

}

vList.Add(new VertexNode(c.VertexID, c.SensorCode));

}

return vList;

}

public List<uint> getVertexesIDs() {

List<uint> vList = new List<uint>();

foreach (Correspondence c in correspondences)

{

vList.Add(c.VertexID);

}

return vList;

}

public List<uint> getAdjacentVertexes(uint vertexID)

{

foreach (Vertex v in vertexes)

{

if (v.ID == vertexID)

{

return v.getAdjacentVertexes();

}

}

return new List<uint>();

}

public List<string> getAvailableParams() {

List<string> list = new List<string>();

foreach (Sensor s in auxiliaryParams) {

list.Add(s.UniqueCode);

}

return list;

}

public List<Event> getEvents(uint numOfTechLines = 1, OperatorLevel operatorLevel = OperatorLevel.Third, InformationRepresentationMethod informationRepresentationMethod = InformationRepresentationMethod.MnemonicDiagram) {

if (numOfTechLines < 1)

throw new InvalidArgument();

List<Event> events = new List<Event>();

List<uint> path = new List<uint>();

uint eventCounter = 1;

foreach (Vertex v in vertexes) {

if (v.AdjVCount == 1 && v.Type != VertexType.Main && v.Type != VertexType.Violation) {

path = getWayUp(v.getAdjacentVertexes()[0]);

path.Insert(0, v.ID);

path.Add(mainVertexID);

Event e = new Event(eventCounter++, path, numOfTechLines, getSensorsCount(), getViolationInformationQuantity(path[path.Count - 2],numOfTechLines), operatorLevel, getDampedDeviceTotal(), informationRepresentationMethod);

foreach (uint tempV in path) {

if (tempV != mainVertexID) {

Sensor s = getSensor(getCorrespondence(tempV).SensorCode);

if (s.ControlMethod == SensorControlMethod.Request)

{

e.addRequestSensor(s);

}

else if (s.ControlMethod == SensorControlMethod.Auto) {

e.addAutoSensor(s);

}

if (tempV != path[path.Count - 2]) {

e.addVertexAndSubverticiesNumber(new Event.VertexAndNumOfSubverticies((int)tempV, getNumberVerticiesInSubtree(tempV)));

}

}

}

events.Add(e);

}

}

return events;

}

public VertexNode getMain() {

foreach (Correspondence c in correspondences) {

if (c.VertexID == mainVertexID) {

return new VertexNode(c.VertexID, c.SensorCode);

}

}

throw new UnexpectedError();

}

public Point getVertexCenterPos(uint vID) {

foreach (Vertex v in vertexes) {

if (v.ID == vID)

return v.CenterPos;

}

throw new NotFoundException();

}

public void setVertexCenterPos(uint vID, Point pos) {

foreach (Vertex v in vertexes)

{

if (v.ID == vID) {

v.CenterPos = pos;

return;

}

}

throw new NotFoundException();

}

public List<string> getAllSensors() {

List<string> allSensors = new List<string>();

foreach ((Sensor, uint) value in violations)

{

allSensors.Add(value.Item1.UniqueCode);

}

foreach (Sensor value in auxiliaryParams)

{

allSensors.Add(value.UniqueCode);

}

if (allSensors.Count == 0)

return allSensors;

return allSensors.Distinct().ToList();

}

public List<uint> setMode(GVSSMode newMode) {

if (newMode == this.mode) {

return new List<uint>();

}

List<uint> vertexesToRemove = new List<uint>();

if (newMode == GVSSMode.RepetativeViolations)

{

//были уникальные - стали повторяющиеся

foreach ((Sensor, uint) violation in violations) {

auxiliaryParams.Add(violation.Item1);

}

}

else if (newMode == GVSSMode.UniqueViolations)

{

//были повторяющиеся - стали уникальными

// remove from auxiliary params, delete vertexes

foreach ((Sensor, uint) violation in violations) {

foreach (Sensor auxiliary in auxiliaryParams) {

if (violation.Item1.UniqueCode == auxiliary.UniqueCode) {

auxiliaryParams.Remove(auxiliary);

break;

}

}

}

List<uint> toRemoveVertexes = new List<uint>();

foreach((Sensor, uint) violation in violations) {

foreach (Correspondence c in correspondences)

{

if (c.SensorCode == violation.Item1.UniqueCode && getVertex(c.VertexID).Type == VertexType.AuxiliaryParam) {

toRemoveVertexes.Add(c.VertexID);

}

}

}

foreach (uint v in toRemoveVertexes) {

vertexesToRemove.AddRange(delete(v));

}

}

this.mode = newMode;

if (vertexesToRemove.Count != 0)

{

return vertexesToRemove.Distinct().ToList();

}

else return new List<uint>();

}

public List<string> getAllViolations() {

List<string> v = new List<string>();

foreach ((Sensor, uint) viol in violations) {

v.Add(viol.Item1.UniqueCode);

}

return v;

}

public bool isSensorInGVSS(string uniqueCode) {

bool isIn = false;

foreach (Sensor s in auxiliaryParams)

{

if (s.UniqueCode == uniqueCode) {

isIn = true;

break;

}

}

if (!isIn && mode == GVSSMode.UniqueViolations) {

foreach ((Sensor, uint) v in violations) {

if (v.Item1.UniqueCode == uniqueCode) {

isIn = true;

break;

}

}

}

return isIn;

}

public Sensor getSensor(string uniqueCode) {

if (isSensorInGVSS(uniqueCode))

{

foreach (Sensor sens in auxiliaryParams) {

if (sens.UniqueCode == uniqueCode)

return sens;

}

if (mode == GVSSMode.UniqueViolations) {

foreach ((Sensor, uint) v in violations) {

if (v.Item1.UniqueCode == uniqueCode) {

return v.Item1;

}

}

}

}

throw new InvalidArgument();

}

public List<uint> deleteAll(string uniqueCode) {

if (isSensorInGVSS(uniqueCode))

{

List<uint> toRemove = new List<uint>();

foreach (Correspondence c in correspondences) {

if (c.SensorCode == uniqueCode) {

toRemove.Add(c.VertexID);

}

}

List<uint> deletedVertexesList = new List<uint>();

foreach (uint v in toRemove) {

deletedVertexesList.AddRange(delete(v));

}

deleteAuxiliary(uniqueCode);

if (deletedVertexesList.Count != 0)

return deletedVertexesList.Distinct().ToList();

return deletedVertexesList;

}

else {

throw new InvalidArgument();

}

}

public List<uint> getVertexChilds(uint vID) {

if (isVertexInGraph(vID))

{

Vertex v = getVertex(vID);

uint vLevel = v.Level;

List<uint> adjVertexes = v.getAdjacentVertexes();

foreach (uint adjV in adjVertexes) {

if (getLevel(adjV) < vLevel) {

adjVertexes.Remove(adjV);

break;

}

}

return adjVertexes;

}

else {

throw new InvalidArgument();

}

}

public void editSensor(string uniqueCode, SensorControlAgent controlAgent, SensorControlMethod controlMethod, SensorRequestBodies requestBody, SensorRequestPlacementMethod requestPlacementMethod, string description) {

foreach ((Sensor, uint) v in violations) {

if (v.Item1.UniqueCode == uniqueCode) {

v.Item1.ControlMethod = controlMethod;

v.Item1.ControlAgent = controlAgent;

v.Item1.Description = description;

v.Item1.RequestBody = requestBody;

v.Item1.RequestPlacementMethod = requestPlacementMethod;

break;

}

}

foreach (Sensor a in auxiliaryParams)

{

if (a.UniqueCode == uniqueCode)

{

a.ControlMethod = controlMethod;

a.ControlAgent = controlAgent;

a.Description = description;

a.RequestBody = requestBody;

a.RequestPlacementMethod = requestPlacementMethod;

}

}

}

public string getType(uint vID) {

if (isVertexInGraph(vID))

{

if (vID == mainVertexID)

return "Main";

foreach ((Sensor, uint) v in violations)

{

if (v.Item2 == vID)

return "Violation";

}

return "Auxiliary";

}

else throw new InvalidArgument();

}

public float getEdgeWeight(uint firstV, uint secondV)

{

if (doesTheEdgeExist(firstV, secondV))

{

foreach (Vertex v in vertexes)

{

if (v.ID == firstV)

{

return v.getEdgeWeight(secondV);

}

else if (v.ID == secondV)

{

return v.getEdgeWeight(firstV);

}

}

}

return -1;

}

public GVSSMode getMode() {

return mode;

}

public bool isNull() {

return vertexes.Count == 1 && correspondences.Count == 1 && auxiliaryParams.Count == 0 && violations.Count == 0;

}

}

}

***InputCorrector.cs:***

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

namespace gvss\_project

{

//public enum TextBoxType { Description, FailureRate, None};

public static class InputCorrector

{

private enum Errors { Name, NotFilled, AllEmpty, None}

private static ErrorWhileInput getError(Errors error) {

SolidColorBrush redBrush = new SolidColorBrush((Color)ColorConverter.ConvertFromString("#9c0b4a"));

if (error == Errors.Name)

return new ErrorWhileInput { Message = DefaultSettings.getErrorMessageText(ErrorMessages.TheNameExistsAlready), Brush = redBrush, isBtnEnabled = false };

else if(error == Errors.NotFilled)

return new ErrorWhileInput { Message = DefaultSettings.getErrorMessageText(ErrorMessages.NotAllFieldsAreFilled), Brush = redBrush, isBtnEnabled = false };

else if (error == Errors.AllEmpty)

return new ErrorWhileInput { Message = "", isBtnEnabled = false };

else

return new ErrorWhileInput { Message = "", isBtnEnabled = true };

}

private struct ErrorWhileInput {

public string Message;

public Brush Brush;

public bool isBtnEnabled;

}

private static bool isArrowRBSelected(List<RadioButton> arrowRadioButtons, ref SensorArrowSign resultArrowSign) {

foreach (RadioButton rb in arrowRadioButtons)

{

if (rb.IsChecked == true)

{

resultArrowSign = (SensorArrowSign)rb.Tag;

return true;

}

}

return false;

}

private static bool isMatch(string text, string pattern) {

return System.Text.RegularExpressions.Regex.IsMatch(text, pattern);

}

private static bool isFloatValue(string text)

{

if (text.Length == 1 && text[0] == '0')

return false;

if (text.Length >= 2 && text[0] == '0' && text[1] != '.')

return false;

System.Text.RegularExpressions.Regex regex = new System.Text.RegularExpressions.Regex(@"^\d\*\.?\d+$");

return regex.IsMatch(text);

}

private static void setRequestCBs(bool isEnabled, ComboBox requestBodyCB, Border requestBodyBorder, ComboBox requestPlacementMethodCB, Border requestPlacementMethodBorder) {

Brush brush;

if (isEnabled)

{

brush = (SolidColorBrush)App.Current.TryFindResource("defBlue");

ParameterValueHelper.InitializeComboBoxAndSelectIndex(requestBodyCB, typeof(SensorRequestBodies));

}

else {

brush = (SolidColorBrush)App.Current.TryFindResource("disabledBlue");

ParameterValueHelper.InitializeComboBoxWithNoneAndSelectIndex(requestBodyCB, typeof(SensorRequestBodies));

}

requestBodyBorder.Background = brush;

requestPlacementMethodBorder.Background = brush;

requestBodyCB.IsEnabled = isEnabled;

requestPlacementMethodCB.IsEnabled = isEnabled;

}

private static bool isFailureRateTextBoxINullOrValid(TextBox textBox) {

return (textBox == null) || (textBox.Text.Length != 0 && isFloatValue(textBox.Text));

}

private static bool isFailureRateTextBoxINullOrNotValid(TextBox textBox)

{

return (textBox == null) || !(textBox.Text.Length != 0 && isFloatValue(textBox.Text));

}

public static bool isFailureRateTextBoxValid(TextBox textBox) {

return textBox.Text.Length != 0 && isFloatValue(textBox.Text);

}

public static void InitializeRelatedToSensorCBs(ComboBox controlMethodCB, ComboBox controlAgentCB, ComboBox requestBodyCB, ComboBox requestPlacementMethodCB) {

ParameterValueHelper.InitializeComboBox(controlMethodCB, typeof(SensorControlMethod));

ParameterValueHelper.InitializeComboBox(controlAgentCB, typeof(SensorControlAgent));

ParameterValueHelper.InitializeComboBoxWithNone(requestBodyCB, typeof(SensorRequestBodies));

ParameterValueHelper.InitializeComboBoxWithNone(requestPlacementMethodCB, typeof(SensorRequestPlacementMethod));

controlAgentCB.SelectedIndex = 0;

controlMethodCB.SelectedIndex = 0;

requestBodyCB.SelectedIndex = 0;

requestPlacementMethodCB.SelectedIndex = 0;

}

public static void textBoxWithoutSpaceKey\_PreviewKeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Space)

e.Handled = true;

}

public static void IndexTB\_PreviewTextInput(object sender, TextCompositionEventArgs e) {

if ((e.Text == "0" && ((TextBox)sender).Text.Length == 0) || !isMatch(e.Text, "^[0-9]"))

{

e.Handled = true;

}

}

public static void CapitalLetterTB\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

if (!isMatch(e.Text, "^[АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзийклмнопрстуфхцчшщъыьэюяІіЇїa-zA-Z]"))

{

e.Handled = true;

}

else {

((TextBox)sender).Text = e.Text[0].ToString().ToUpper();

}

}

public static void FailureRateTB\_PreviewTextInput(object sender, TextCompositionEventArgs e) {

if (!isMatch(e.Text, "^[.0-9]"))

{

e.Handled = true;

}

}

public static void changeSensorNameRepresentation(TextBox capitalLetterTextBox, TextBox indexTextBox, Label namePreviewLabel, List<RadioButton> arrowRadioButtons, Border backgroundBorder, DockPanel uniqueCodePreviewDock, ref SensorArrowSign checkedArrowRB) {

if (capitalLetterTextBox.Text.Length > 0 && indexTextBox.Text.Length > 0)

{

foreach (RadioButton rb in arrowRadioButtons)

{

rb.IsEnabled = true;

}

backgroundBorder.Background = (SolidColorBrush)App.Current.TryFindResource("blueBorderBrush");

int length = indexTextBox.Text.Length;

int width = 0;

int paddingLeftOffset = 0;

if (length == 1)

{

paddingLeftOffset = 8;

width = 58;

}

else if (length == 2)

{

paddingLeftOffset = 5;

width = 64;

}

else if (length == 3)

{

paddingLeftOffset = 2;

width = 70;

}

uniqueCodePreviewDock.Width = width;

namePreviewLabel.Padding = new Thickness(paddingLeftOffset, 0, 0, 0);

namePreviewLabel.Content = Computations.getCapLetAndIndex(capitalLetterTextBox.Text[0], Int32.Parse(indexTextBox.Text));

}

else

{

int width = 56;

namePreviewLabel.Content = "";

uniqueCodePreviewDock.Width = width;

backgroundBorder.Background = (SolidColorBrush)App.Current.TryFindResource("disabledBlue");

foreach (RadioButton rb in arrowRadioButtons)

{

rb.IsChecked = false;

rb.IsEnabled = false;

}

checkedArrowRB = SensorArrowSign.None;

}

}

public static void arrowRB\_Click(object sender, RoutedEventArgs e, ref SensorArrowSign checkedArrowRB)

{

RadioButton rb = (RadioButton)sender;

SensorArrowSign rbArrow = (SensorArrowSign)rb.Tag;

if (rbArrow == checkedArrowRB)

{

rb.IsChecked = false;

checkedArrowRB = SensorArrowSign.None;

}

else

{

checkedArrowRB = (SensorArrowSign)rb.Tag;

}

}

public static void setButton(Button button, TextBox indexTextBox, TextBox capitalLetterTextBox, TextBox descriptionTextBox, List<string> sensors,List<RadioButton> arrowRadioButtons, TextBlock resultTextBlock, TextBox failureTextBox = null) {

SensorArrowSign arrowSign = SensorArrowSign.None;

Errors error = Errors.NotFilled;

bool isArrowSelected = isArrowRBSelected(arrowRadioButtons, ref arrowSign);

bool isIndexTBValid = indexTextBox.Text.Length != 0;

bool isCapitalLetterTBValid = capitalLetterTextBox.Text.Length != 0;

bool isDescriptionTBValid = descriptionTextBox.Text.Length != 0;

bool isFailureRateTBNullOrValid = isFailureRateTextBoxINullOrValid(failureTextBox);

bool isFailureRateTBNullOrNotValid = isFailureRateTextBoxINullOrNotValid(failureTextBox);

bool isAllEmpty = !isArrowSelected && !isIndexTBValid && !isCapitalLetterTBValid && !isDescriptionTBValid;

if (isArrowSelected && isIndexTBValid && isCapitalLetterTBValid)

{

bool isSensorNameAvailable = true;

string curCode = Computations.getUniqueCode(capitalLetterTextBox.Text[0], UInt16.Parse(indexTextBox.Text), arrowSign);

foreach (string ucode in sensors)

{

if (ucode == curCode)

{

isSensorNameAvailable = false;

break;

}

}

if (!isSensorNameAvailable)

error = Errors.Name;

else if (isDescriptionTBValid && isFailureRateTBNullOrValid)

error = Errors.None;

else if (!isDescriptionTBValid || !isFailureRateTBNullOrValid)

error = Errors.NotFilled;

}

else if (isAllEmpty && isFailureRateTBNullOrNotValid)

error = Errors.AllEmpty;

ErrorWhileInput errorWhileInput = getError(error);

resultTextBlock.Text = errorWhileInput.Message;

resultTextBlock.Foreground = errorWhileInput.Brush;

button.IsEnabled = errorWhileInput.isBtnEnabled;

}

public static void ControlMethodCB\_SelectionChanged(

object sender,

SelectionChangedEventArgs e,

TextBlock labelTextBlock,

ComboBox requestBodyCB,

Border requestBodyBorder,

ComboBox requestPlacementMethodCB,

Border requestPlacementMethodBorder) {

SensorControlMethod cm = (SensorControlMethod)(((ComboBoxItem)(((ComboBox)sender).SelectedItem)).Tag);

string labelText = "";

bool isRequest = false;

if (cm == SensorControlMethod.Auto)

{

labelText = DefaultSettings.getLabelText(Labels.ControlAgentAuto);

}

else if (cm == SensorControlMethod.Request)

{

labelText = DefaultSettings.getLabelText(Labels.ControlAgentRequest);

isRequest = true;

}

else

throw new UnexpectedError();

labelTextBlock.Text = labelText;

setRequestCBs(isRequest, requestBodyCB, requestBodyBorder, requestPlacementMethodCB, requestPlacementMethodBorder);

}

public static void requestBodyCB\_SelectionChanged(object sender, SelectionChangedEventArgs e, ComboBox requestPlacementMethodCB) {

if (((ComboBox)sender).Items.Count != 0)

{

SensorRequestBodies rb = (SensorRequestBodies)(((ComboBoxItem)(((ComboBox)sender).SelectedItem)).Tag);

ParameterValueHelper.InitializeRequestPlacementMethod(requestPlacementMethodCB, rb);

}

else {

ParameterValueHelper.InitializeComboBoxWithNoneAndSelectIndex(requestPlacementMethodCB, typeof(SensorRequestPlacementMethod));

}

}

public static void setSensorInfoTextBlocks(

Sensor s,

TextBlock controlMethodInfo,

TextBlock controlAgentInfo,

TextBlock requestBodyInfo,

TextBlock requestPlacementMethodInfo,

TextBlock descriptionInfo,

Border requestBodyInfoBorder,

Border requestPlacementMethodInfoBorder,

TextBlock controlAgentInfoTBlock)

{

controlAgentInfo.Text = DefaultSettings.getControlAgentText(s.ControlAgent);

controlMethodInfo.Text = DefaultSettings.getControlMethodText(s.ControlMethod);

if (s.ControlMethod == SensorControlMethod.Request)

{

Brush b = (SolidColorBrush)App.Current.TryFindResource("defBlue");

controlAgentInfoTBlock.Text = DefaultSettings.getLabelText(Labels.ControlAgentRequest);

requestBodyInfoBorder.Background = b;

requestPlacementMethodInfoBorder.Background = b;

requestBodyInfo.Text = DefaultSettings.getRequestBodyText(s.RequestBody);

requestPlacementMethodInfo.Text = DefaultSettings.getRequestPlacementMethodText(s.RequestPlacementMethod);

}

else {

Brush b = (SolidColorBrush)App.Current.TryFindResource("disabledBlue");

controlAgentInfoTBlock.Text = DefaultSettings.getLabelText(Labels.ControlAgentAuto);

requestBodyInfoBorder.Background = b;

requestPlacementMethodInfoBorder.Background = b;

requestBodyInfo.Text = "";

requestPlacementMethodInfo.Text = "";

}

descriptionInfo.Text = s.Description;

}

public static void InitializeTechLinesComboBox(ComboBox techLinesCB, int numOfTechLines)

{

if (numOfTechLines > 0)

{

techLinesCB.Items.Clear();

foreach (uint i in Enumerable.Range(1, numOfTechLines))

{

ComboBoxItem cbi = new ComboBoxItem();

cbi.Tag = i;

cbi.Content = i.ToString();

techLinesCB.Items.Add(cbi);

}

techLinesCB.SelectedIndex = 0;

}

}

public static void LocalizeComboBox(ComboBox cb, Type innerType) {

if (cb.Items.Count == 0)

return;

if (innerType == typeof(SensorControlMethod))

{

foreach (ComboBoxItem cbi in cb.Items) {

cbi.Content = DefaultSettings.getControlMethodText((SensorControlMethod)cbi.Tag);

}

}

else if (innerType == typeof(SensorControlAgent))

{

foreach (ComboBoxItem cbi in cb.Items)

{

cbi.Content = DefaultSettings.getControlAgentText((SensorControlAgent)cbi.Tag);

}

}

else if (innerType == typeof(SensorRequestPlacementMethod))

{

foreach (ComboBoxItem cbi in cb.Items)

{

cbi.Content = DefaultSettings.getRequestPlacementMethodText((SensorRequestPlacementMethod)cbi.Tag);

}

}

else if (innerType == typeof(SensorRequestBodies))

{

foreach (ComboBoxItem cbi in cb.Items)

{

cbi.Content = DefaultSettings.getRequestBodyText((SensorRequestBodies)cbi.Tag);

}

}

else if (innerType == typeof(InformationRepresentationMethod))

{

foreach (ComboBoxItem cbi in cb.Items)

{

cbi.Content = DefaultSettings.getInformationRepresentationMethodText((InformationRepresentationMethod)cbi.Tag);

}

}

}

public static void LocalizeComboBoxes(ComboBox controlMethod, ComboBox controlAgent, ComboBox requestBody, ComboBox requestPlacementMethod, ComboBox informationRepresentationMethod) {

LocalizeComboBox(controlMethod, typeof(SensorControlMethod));

LocalizeComboBox(controlAgent, typeof(SensorControlAgent));

LocalizeComboBox(requestBody, typeof(SensorRequestBodies));

LocalizeComboBox(requestPlacementMethod, typeof(SensorRequestPlacementMethod));

LocalizeComboBox(informationRepresentationMethod, typeof(InformationRepresentationMethod));

}

}

}

***MainWindow.xaml.cs:***

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace gvss\_project

{

public struct Connection {

public uint v1ID { get; private set; }

public uint v2ID { get; private set; }

public Connection(uint v1, uint v2) {

v1ID = v1;

v2ID = v2;

}

}

public partial class MainWindow : Window

{

private GVSS g;

private uint btnWidth = 50;

private uint btnHeight = 50;

private bool isPressed = false;

private bool particularEventIsSelected = false;

private Point topLeft;

private Point pos;

private int indexCounter = 3;

private ContextMenu contextMenu;

private SensorArrowSign checkedArrowRB;

private GVSSMode checkedModeRB;

private LanguageMode currentLanguage;

private List<RadioButton> uniqueCodeRadioButtons;

private List<RadioButton> modeRadioButtons;

private List<RadioButton> languageModeRadioButtons;

private bool initializationFinished;

private readonly Point mainInitialPos;

private MessageBoxResult warningMB(string text) {

return MessageBox.Show(text, DefaultSettings.getWindowTitleText(WindowTitles.WarningMB),

MessageBoxButton.YesNo, MessageBoxImage.Warning, MessageBoxResult.No);

}

private void errorMB(string text) {

MessageBox.Show(text, DefaultSettings.getWindowTitleText(WindowTitles.ErrorMB), MessageBoxButton.OK, MessageBoxImage.Error);

}

private void programmaticallySetGVSSMode(GVSSMode mode) {

foreach (RadioButton modeRB in modeRadioButtons)

{

if ((GVSSMode)(modeRB.Tag) == mode)

{

checkedModeRB = mode;

modeRB.IsChecked = true;

break;

}

}

}

private void clearInterface() {

load\_sensorEditCB();

load\_chooseEventCB();

clearRegisterParamFields();

}

private bool SaveGVSS() {

SaveFileDialog sfd = new SaveFileDialog();

sfd.Filter = "GVSS Files (\*.gvss)|\*.gvss";

sfd.InitialDirectory = Environment.CurrentDirectory;

if (sfd.ShowDialog() == true) {

BinarySerialization.WriteToBinaryFile<GVSS>(sfd.FileName, g);

return true;

}

return false;

}

private void LoadGVSS() {

if (endWithGVSS(DefaultSettings.getMessageBoxText(MessageBoxMessages.SaveGVSSBeforeLoad)) == true) {

OpenFileDialog ofd = new OpenFileDialog();

ofd.Filter = "GVSS Files (\*.gvss)|\*.gvss| MathLAB Files (\*.asv)|\*.asv";

ofd.InitialDirectory = Environment.CurrentDirectory;

if (ofd.ShowDialog() == true)

{

string fileName = ofd.FileName;

string fileExt = System.IO.Path.GetExtension(fileName);

if (fileExt == ".asv")

{

List<double> doubleNumbers = new List<double>();

try

{

doubleNumbers = FindYArray(fileName);

}

catch (Exception)

{

errorMB(DefaultSettings.getErrorMessageText(ErrorMessages.WhileASVFileCorrupted));

return;

}

MathLABResultWindow window = new MathLABResultWindow(doubleNumbers);

window.Owner = this;

window.WindowStartupLocation = WindowStartupLocation.CenterOwner;

window.ShowDialog();

}

else if (fileExt == ".gvss") {

List<uint> oldVertexes = g.getVertexesIDs();

try

{

GVSS newGVSS = BinarySerialization.ReadFromBinaryFile<GVSS>(fileName);

this.g = newGVSS;

}

catch (Exception)

{

errorMB(DefaultSettings.getErrorMessageText(ErrorMessages.WhileGVSSFileCorrupted));

return;

}

removeVertexes(oldVertexes);

clearInterface();

GVSSMode newMode = g.getMode();

if (newMode != checkedModeRB)

{

programmaticallySetGVSSMode(newMode);

}

List<VertexNode> newVertexes = g.getVertexes(false);

foreach (VertexNode v in newVertexes)

{

spawnVertex(g.getType(v.VertexID), v);

List<uint> childVertexes = g.getVertexChilds(v.VertexID);

foreach (uint cv in childVertexes)

{

float probability = -1f;

if (v.VertexID == 1)

{

probability = g.getEdgeWeight(v.VertexID, cv);

}

spawnEdge(v.VertexID, cv, probability);

}

}

}

}

}

}

private void SavePhoto()

{

SaveFileDialog sfd = new SaveFileDialog();

sfd.Filter = "PNG Files (\*.png)|\*.png";

sfd.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);

if (sfd.ShowDialog() == true)

{

RenderVisualService.RenderToPNGFile(canvas, sfd.FileName);

}

}

private void addViolation(char capLet, uint index, string description, SensorControlMethod controlMethod, SensorControlAgent controlAgent, SensorRequestBodies requestBody, SensorRequestPlacementMethod requestPlacementMethod, SensorArrowSign arrowSign, Point centerPos, float probability) {

VertexNode v = g.addViolation(capLet, index, description, controlMethod, controlAgent, arrowSign, requestBody, requestPlacementMethod, centerPos,probability);

spawnViolation(v, probability);

fillManagePanel();

load\_chooseEventCB();

}

private void addViolation(string auxiliaryCode, Point centerPos, float probability) {

VertexNode v = g.addViolation(auxiliaryCode, centerPos,probability);

spawnViolation(v, probability);

fillManagePanel();

load\_chooseEventCB();

}

private void spawnViolation(VertexNode v, float probability) {

spawnVertex("Violation", v);

spawnEdge(g.getMain().VertexID, v.VertexID, probability);

}

private void rollUpOrExpandStackPanel(StackPanel s, Border b, Visibility v) {

uint rolledUpHeight;

uint expandedHeight;

if (s == manageStackPanel)

{

rolledUpHeight = 33;

expandedHeight = 370;

}

else

throw new UnexpectedError();

if (v == Visibility.Collapsed)

{

s.Visibility = Visibility.Collapsed;

b.Height = rolledUpHeight;

b.CornerRadius = new CornerRadius(15, 15, 0, 0);

}

else if (v == Visibility.Visible) {

s.Visibility = Visibility.Visible;

b.Height = expandedHeight;

b.CornerRadius = new CornerRadius(15, 15, 15, 15);

}

}

private string load\_sensorEditCB(string uniqueCode = "") {

bool selectExplicit = false;

ComboBoxItem cbi = new ComboBoxItem();

if (uniqueCode != "" && g.isSensorInGVSS(uniqueCode))

selectExplicit = true;

sensorsEditCB.Items.Clear();

List<string> allSensors = g.getAllSensors();

foreach (string sensor in allSensors) {

ComboBoxItem c = new ComboBoxItem();

c.Tag = sensor;

c.Content = sensor;

sensorsEditCB.Items.Add(c);

if (selectExplicit && uniqueCode == sensor)

cbi = c;

}

if (selectExplicit)

sensorsEditCB.SelectedItem = cbi;

else

sensorsEditCB.SelectedIndex = 0;

if (sensorsEditCB.Items.Count == 0)

{

editButton.IsEnabled = false;

deleteButton.IsEnabled = false;

rollUpOrExpandStackPanel(manageStackPanel,manageBorder,Visibility.Collapsed);

return "";

}

else {

editButton.IsEnabled = true;

deleteButton.IsEnabled = true;

rollUpOrExpandStackPanel(manageStackPanel, manageBorder, Visibility.Visible);

return (string)(((ComboBoxItem)(sensorsEditCB.SelectedItem)).Tag);

}

}

private void load\_chooseEventCB(int eID = 0) {

bool selectExplicit = false;

if (eID > 0 && eID <= chooseEventCB.Items.Count - 1) {

selectExplicit = true;

}

chooseEventCB.Items.Clear();

uint numOfTechLines = (uint)((ComboBoxItem)techLinesCB.SelectedItem).Tag;

OperatorLevel operatorLevel = (OperatorLevel)((ComboBoxItem)operatorLevelCB.SelectedItem).Tag;

InformationRepresentationMethod informationRepresentationMethod = (InformationRepresentationMethod)((ComboBoxItem)informationRepresentationMethodCB.SelectedItem).Tag;

List <Event> events = g.getEvents(numOfTechLines, operatorLevel, informationRepresentationMethod);

ComboBoxItem c = new ComboBoxItem();

c.Content = DefaultSettings.getWordValue(Words.All);

c.Tag = (int)0;

chooseEventCB.Items.Add(c);

ComboBoxItem cbi = new ComboBoxItem();

foreach (Event e in events)

{

c = new ComboBoxItem();

c.Tag = e;

c.Content = e.ID + " (вершина #" + e.VertexPath[0]+")";

chooseEventCB.Items.Add(c);

if (selectExplicit && eID == e.ID)

cbi = c;

}

if (selectExplicit)

chooseEventCB.SelectedItem = cbi;

else

chooseEventCB.SelectedIndex = 0;

}

private void connectAuxiliaryParamAndViolation(uint violationID, string auxiliaryParamCode, Point centerPosOfAuxiliaryParamVertex) {

VertexNode v2 = g.connect(violationID, auxiliaryParamCode, centerPosOfAuxiliaryParamVertex);

spawnVertex("Auxiliary", v2);

spawnEdge(violationID, v2.VertexID);

fillManagePanel();

load\_chooseEventCB();

}

private Point getVertexTopLeft(uint vID) {

Point vPos = g.getVertexCenterPos(vID);

vPos.X -= btnWidth / 2;

vPos.Y -= btnHeight / 2;

return vPos;

}

private void spawnVertex(string style, VertexNode tag) {

Point vertexPosition = getVertexTopLeft(tag.VertexID);

Button newBtn = new Button { Width = btnWidth, Height = btnHeight };

newBtn.Style = Resources[style] as Style;

newBtn.PreviewMouseDown += NewBtn\_MouseDown;

newBtn.PreviewMouseUp += NewBtn\_MouseUp;

newBtn.PreviewMouseMove += NewBtn\_MouseMove;

newBtn.Tag = tag;

newBtn.Content = tag.UniqueCode;

if (tag.UniqueCode != "Main") {

newBtn.ContextMenu = contextMenu;

}

if (tag.UniqueCode != "Main")

newBtn.Click += NewBtn\_Click;

Panel.SetZIndex(newBtn, 2);

Canvas.SetLeft(newBtn, vertexPosition.X);

Canvas.SetTop(newBtn, vertexPosition.Y);

canvas.Children.Add(newBtn);

spawnVertexNum(tag.VertexID, new Point(vertexPosition.X, vertexPosition.Y));

}

private void NewBtn\_Click(object sender, RoutedEventArgs e)

{

fillManagePanel(((VertexNode)(((Button)sender).Tag)).UniqueCode);

}

private void spawnVertexNum(uint vID, Point pos) {

Label num = new Label();

num.Content = '#' + vID.ToString();

int textLen = num.Content.ToString().Length;

Panel.SetZIndex(num, 3);

num.Foreground = (SolidColorBrush)this.TryFindResource("mainButtonBlue");

num.FontWeight = FontWeights.Bold;

num.Tag = vID;

num.FontSize = 10;

num.Width = btnWidth;

num.Padding = new Thickness(btnWidth / 2 - 3 - ((textLen-2)\*4), 5, 0, 0);

num.IsHitTestVisible = false;

Canvas.SetLeft(num, pos.X);

Canvas.SetTop(num, pos.Y);

canvas.Children.Add(num);

}

private void moveVertexNum(uint vID, Point pos) {

Label l = new Label();

foreach (UIElement uI in canvas.Children) {

if (uI.GetType() == typeof(Label)) {

if ((uint)(((Label)uI).Tag) == vID) {

l = (Label)uI;

break;

}

}

}

if (l == new Label())

throw new NotFoundException();

Canvas.SetLeft(l, pos.X);

Canvas.SetTop(l, pos.Y);

}

private void setVertexNumIndex(uint vID, int index)

{

Label l = new Label();

foreach (UIElement uI in canvas.Children)

{

if (uI.GetType() == typeof(Label))

{

if ((uint)(((Label)uI).Tag) == vID)

{

l = (Label)uI;

break;

}

}

}

if (l == new Label())

throw new NotFoundException();

Panel.SetZIndex(l, index);

}

private void NewBtn\_MouseMove(object sender, MouseEventArgs e)

{

if (isPressed) {

Button b = (Button)sender;

Point pos1 = e.GetPosition(canvas);

double canvasHeight = canvas.ActualHeight;

double canvasWidth = canvas.ActualWidth;

double left = topLeft.X + (pos1.X - pos.X);

double top = topLeft.Y + (pos1.Y - pos.Y);

if (left + b.Width > canvasWidth)

left = canvasWidth - b.Width;

else if (left < 0)

left = 0;

if (top + b.Height > canvasHeight)

top = canvasHeight - b.Height;

else if (top < 0)

top = 0;

Canvas.SetLeft(b, left);

Canvas.SetTop(b, top);

uint vID = ((VertexNode)b.Tag).VertexID;

g.setVertexCenterPos(vID, new Point(left + (btnWidth / 2), top + (btnHeight / 2)));

redrawEdges(vID);

moveVertexNum(vID, new Point(left, top));

}

}

private void NewBtn\_MouseUp(object sender, MouseButtonEventArgs e)

{

if (e.LeftButton == MouseButtonState.Released)

isPressed = false;

}

private void NewBtn\_MouseDown(object sender, MouseButtonEventArgs e)

{

if (e.LeftButton == MouseButtonState.Pressed) {

if (particularEventIsSelected)

exitParticularEvent();

isPressed = true;

Button b = (Button)sender;

Panel.SetZIndex(b, ++indexCounter);

setVertexNumIndex(((VertexNode)b.Tag).VertexID, ++indexCounter);

topLeft = new Point(Canvas.GetLeft(b), Canvas.GetTop(b));

pos = e.GetPosition(canvas);

}

}

private void spawnEdge(uint v1ID, uint v2ID, float probability = -1f) {

bool setToolTip = false;

if (probability != -1f && v1ID == g.getMain().VertexID) {

setToolTip = true;

}

Line line = new Line();

Point v1Pos = g.getVertexCenterPos(v1ID);

Point v2Pos = g.getVertexCenterPos(v2ID);

line.X1 = v1Pos.X;

line.Y1 = v1Pos.Y;

line.X2 = v2Pos.X;

line.Y2 = v2Pos.Y;

line.Cursor = Cursors.Arrow;

line.StrokeThickness = 2;

line.Stroke = (Brush)new BrushConverter().ConvertFrom("#00358a");

line.Tag = new Connection(v1ID, v2ID);

Panel.SetZIndex(line, 1);

if (setToolTip) {

line.ToolTip = new ToolTip { Content = probability.ToString() };

}

canvas.Children.Add(line);

}

private void redrawEdges(uint vertexThatHaveNewPos) {

Line l;

Connection c;

foreach (UIElement uIElement in canvas.Children) {

if (uIElement.GetType() == typeof(Line)) {

l = (Line)uIElement;

c = (Connection)l.Tag;

if (c.v1ID == vertexThatHaveNewPos)

{

Point p = g.getVertexCenterPos(vertexThatHaveNewPos);

l.X1 = p.X;

l.Y1 = p.Y;

}

else if (c.v2ID == vertexThatHaveNewPos) {

Point p = g.getVertexCenterPos(vertexThatHaveNewPos);

l.X2 = p.X;

l.Y2 = p.Y;

}

}

}

}

private void create\_GVSS() {

g = new GVSS(mainInitialPos);

spawnVertex("Main", g.getMain());

programmaticallySetGVSSMode(g.getMode());

}

public MainWindow()

{

initializationFinished = false;

InitializeComponent();

particularEventIsSelected = false;

InputCorrector.InitializeRelatedToSensorCBs(controlMethodCB, controlAgentCB, requestBodyCB, requestPlacementMethodCB);

InputCorrector.InitializeTechLinesComboBox(techLinesCB, 4);

ParameterValueHelper.InitializeComboBoxAndSelectIndex(operatorLevelCB, typeof(OperatorLevel), 3);

ParameterValueHelper.InitializeComboBoxAndSelectIndex(informationRepresentationMethodCB, typeof(InformationRepresentationMethod));

techLinesCB.SelectedIndex = 3;

uniqueCodeRadioButtons = new List<RadioButton>();

uniqueCodeRadioButtons.Add(arrowUpRB);

uniqueCodeRadioButtons.Add(arrowDownRB);

uniqueCodeRadioButtons.Add(arrowLeftRB);

uniqueCodeRadioButtons.Add(arrowRightRB);

arrowUpRB.Tag = SensorArrowSign.Up;

arrowRightRB.Tag = SensorArrowSign.Right;

arrowLeftRB.Tag = SensorArrowSign.Left;

arrowDownRB.Tag = SensorArrowSign.Down;

checkedArrowRB = SensorArrowSign.None;

modeRadioButtons = new List<RadioButton>();

uniqueViolationsModeRB.Tag = GVSSMode.UniqueViolations;

repetativeViolationModeRB.Tag = GVSSMode.RepetativeViolations;

modeRadioButtons.Add(uniqueViolationsModeRB);

modeRadioButtons.Add(repetativeViolationModeRB);

repetativeViolationModeRB.IsChecked = true;

checkedModeRB = (GVSSMode)repetativeViolationModeRB.Tag;

languageModeRadioButtons = new List<RadioButton>();

currentLanguage = DefaultSettings.LanguageMode;

languageModeRadioButtons.Add(ruLanRB);

ruLanRB.Tag = LanguageMode.Russian;

languageModeRadioButtons.Add(ukrLanRB);

ukrLanRB.Tag = LanguageMode.Ukrainian;

foreach (RadioButton rb in languageModeRadioButtons) {

if ((LanguageMode)rb.Tag == currentLanguage) {

rb.IsChecked = true;

break;

}

}

borderAroundUniqueCodePreview.Background = (SolidColorBrush)this.TryFindResource("disabledBlue");

contextMenu = new ContextMenu();

MenuItem edit = new MenuItem();

edit.Header = DefaultSettings.getContextMenuHeaderText(ContextMenuHeaders.Edit);

edit.Click += ContextMenuClick;

edit.Icon = new Image { Source = new BitmapImage(new Uri("/edit.png", UriKind.Relative)), Stretch = Stretch.Fill };

contextMenu.Items.Add(edit);

MenuItem delete = new MenuItem();

delete.Header = DefaultSettings.getContextMenuHeaderText(ContextMenuHeaders.DeleteVertex);

delete.Click += ContextMenuClick;

delete.Icon = new Image { Source = new BitmapImage(new Uri("/delete.png", UriKind.Relative)), Stretch = Stretch.Fill };

contextMenu.Items.Add(delete);

mainInitialPos = new Point(370, 50);

create\_GVSS();

load\_chooseEventCB();

setUpInterface();

initializationFinished = true;

}

private void ContextMenuClick(object sender, RoutedEventArgs e)

{

MenuItem mi = sender as MenuItem;

if (mi != null)

{

ContextMenu cm = mi.Parent as ContextMenu;

if (cm != null)

{

Button btn = cm.PlacementTarget as Button;

if (btn != null)

{

exitParticularEvent();

if ((string)mi.Header == DefaultSettings.getContextMenuHeaderText(ContextMenuHeaders.DeleteVertex))

{

if (warningMB(DefaultSettings.getMessageBoxText(MessageBoxMessages.DeleteVertex)) == MessageBoxResult.Yes)

{

deleteVertex(((VertexNode)(btn.Tag)).VertexID);

}

}

else if ((string)mi.Header == DefaultSettings.getContextMenuHeaderText(ContextMenuHeaders.Edit)) {

edit(((VertexNode)(btn.Tag)).UniqueCode);

}

}

}

}

}

private void addVertexOnCanvas(object sender, MouseButtonEventArgs e)

{

if (particularEventIsSelected)

{

canvas.Cursor = Cursors.Cross;

exitParticularEvent();

}

else {

Point curCursorPos = e.GetPosition(canvas);

VertexDialog vd = new VertexDialog(g.getAllSensors(), g.getVertexes(), g.getAvailableParams(), g.getAllViolations(), this.checkedModeRB);

vd.Owner = this;

vd.WindowStartupLocation = WindowStartupLocation.CenterOwner;

if (vd.ShowDialog() == true)

{

VertexDialog.VertexDialogResult res = vd.Result;

if (res == VertexDialog.VertexDialogResult.AddViolation)

{

if (this.checkedModeRB == GVSSMode.UniqueViolations)

{

addViolation(vd.CapLetViolation, vd.IndexViolation, vd.DescriptionViolation, vd.ControlMethodViolation, vd.ControlAgentViolation, vd.RequestBody, vd.RequestPlacementMethod,vd.ArrowViolation, curCursorPos, vd.FailureRateViolation);

}

else if (this.checkedModeRB == GVSSMode.RepetativeViolations)

{

addViolation(vd.ViolationUniqueCode, curCursorPos,vd.FailureRateViolation);

}

}

else if (res == VertexDialog.VertexDialogResult.AddAuxiliaryParam)

{

try

{

connectAuxiliaryParamAndViolation(vd.ConnectedVertexID, vd.AuxiliaryParam, curCursorPos);

}

catch (Exception)

{

errorMB(DefaultSettings.getErrorMessageText(ErrorMessages.WhileAddRepetative));

}

}

fillManagePanel();

}

}

}

private void arrowRB\_Click(object sender, RoutedEventArgs e)

{

InputCorrector.arrowRB\_Click(sender, e, ref checkedArrowRB);

setRegisterAuxiliaryParamButton();

}

private void clearRegisterParamFields() {

capLetTxtBox.Text = "";

indexTxtBox.Text = "";

auxiliaryParamRegisterDescription.Text = "";

controlMethodCB.SelectedIndex = 0;

controlAgentCB.SelectedIndex = 0;

}

private void registerButton\_Click(object sender, RoutedEventArgs e)

{

SensorArrowSign arrowSign = SensorArrowSign.None;

foreach (RadioButton rb in uniqueCodeRadioButtons) {

if (rb.IsChecked == true) {

arrowSign = (SensorArrowSign)rb.Tag;

break;

}

}

SensorControlMethod cm = (SensorControlMethod)(((ComboBoxItem)(controlMethodCB.SelectedItem)).Tag);

SensorControlAgent ca = (SensorControlAgent)(((ComboBoxItem)(controlAgentCB.SelectedItem)).Tag);

SensorRequestBodies requestBodies = (SensorRequestBodies)(((ComboBoxItem)(requestBodyCB.SelectedItem)).Tag);

SensorRequestPlacementMethod rpm = (SensorRequestPlacementMethod)(((ComboBoxItem)(requestPlacementMethodCB.SelectedItem)).Tag);

g.addAuxiliaryParam(capLetTxtBox.Text[0], UInt16.Parse(indexTxtBox.Text), auxiliaryParamRegisterDescription.Text,

cm, ca, arrowSign, requestBodies, rpm);

fillManagePanel();

clearRegisterParamFields();

}

private void removeVertexes(List<uint> listOfVertexesToDelete) {

List<UIElement> toRemove = new List<UIElement>();

foreach (UIElement element in canvas.Children)

{

if (element.GetType() == typeof(Button))

{

VertexNode node = (VertexNode)(((Button)element).Tag);

foreach (uint v in listOfVertexesToDelete)

{

if (v == node.VertexID)

{

toRemove.Add(element);

break;

}

}

}

else if (element.GetType() == typeof(Line))

{

Connection connection = (Connection)(((Line)element).Tag);

foreach (uint v in listOfVertexesToDelete)

{

if (v == connection.v1ID)

{

toRemove.Add(element);

}

else if (v == connection.v2ID)

{

toRemove.Add(element);

}

}

}

else if (element.GetType() == typeof(Label))

{

uint id = (uint)(((Label)element).Tag);

foreach (uint v in listOfVertexesToDelete)

{

if (v == id)

{

toRemove.Add(element);

break;

}

}

}

}

foreach (UIElement o in toRemove)

{

canvas.Children.Remove(o);

}

fillManagePanel();

load\_chooseEventCB();

}

private void deleteVertex(uint vertexID) {

List<uint> list = g.delete(vertexID);

removeVertexes(list);

}

private void photoButton\_Click(object sender, RoutedEventArgs e)

{

exitParticularEvent();

SavePhoto();

}

private void modeRB\_Click(object sender, RoutedEventArgs e)

{

GVSSMode newMode = (GVSSMode)((RadioButton)sender).Tag;

if (newMode != checkedModeRB)

{

exitParticularEvent();

bool ifNeedToChangeMode = true;

if (newMode == GVSSMode.UniqueViolations && !g.isNull())

{

if (warningMB(DefaultSettings.getMessageBoxText(MessageBoxMessages.ChangeMode)) == MessageBoxResult.No)

{

ifNeedToChangeMode = false;

}

}

if (ifNeedToChangeMode)

{

List<uint> deletedVertexes = g.setMode(newMode);

if (newMode == GVSSMode.UniqueViolations && deletedVertexes.Count != 0)

{

removeVertexes(deletedVertexes);

}

checkedModeRB = newMode;

}

else

{

foreach (RadioButton radio in modeRadioButtons)

{

if (radio.IsChecked == false)

{

radio.IsChecked = true;

break;

}

}

}

}

}

private void loadManageSensorFields(string uniqueCode) {

InputCorrector.setSensorInfoTextBlocks(

g.getSensor(uniqueCode),

controlMethodInfo,

controlAgentInfo,

requestBodyInfo,

requestPlacementMethodInfo,

descriptionInfo,

requestBodyInfoBorder,

requestPlacementMethodInfoBorder,

controlAgentInfoTBlock

);

}

private void fillManagePanel(string uniqueCode = "") {

string code = uniqueCode;

if (uniqueCode == "" && sensorsEditCB.Items.Count != 0) {

code = (string)(((ComboBoxItem)(sensorsEditCB.SelectedItem)).Tag);

}

code = load\_sensorEditCB(code);

if (code != "") {

loadManageSensorFields(code);

}

}

private void sensorsEditCB\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (((ComboBox)sender).Items.Count != 0)

loadManageSensorFields((string)(((ComboBoxItem)(((ComboBox)sender).SelectedItem)).Tag));

}

private void deleteButton\_Click(object sender, RoutedEventArgs e)

{

exitParticularEvent();

if (warningMB(DefaultSettings.getMessageBoxText(MessageBoxMessages.FullDeleteParam)) == MessageBoxResult.Yes) {

removeVertexes(g.deleteAll((string)(((ComboBoxItem)(sensorsEditCB.SelectedItem)).Tag)));

fillManagePanel("---");

setRegisterAuxiliaryParamButton();

}

}

private void edit(string uniqueCode)

{

EditDialog ed = new EditDialog(g.getSensor(uniqueCode));

ed.Owner = this;

ed.WindowStartupLocation = WindowStartupLocation.CenterOwner;

if (ed.ShowDialog() == true)

{

g.editSensor(ed.UniqueCode, ed.ControlAgent, ed.ControlMethod, ed.RequestBody, ed.RequestPlacementMethod, ed.Description);

fillManagePanel();

}

}

private void editButton\_Click(object sender, RoutedEventArgs e)

{

exitParticularEvent();

edit((string)(((ComboBoxItem)(sensorsEditCB.SelectedItem)).Tag));

}

private void drawParticularEvent(uint eventID) {

List<uint> vList = new List<uint>();

if (eventID == 0) {

makeAllElementsOnCanvasVisible();

return;

}

foreach (ComboBoxItem cbi in chooseEventCB.Items) {

if (cbi.Tag.GetType() == typeof(Event)) {

Event e = (Event)(cbi.Tag);

if (e.ID == eventID)

{

vList.AddRange(e.VertexPath);

break;

}

}

}

if (vList == new List<uint>()) {

throw new UnexpectedError();

}

foreach (UIElement element in canvas.Children) {

bool makeHidden = true;

if (element.GetType() == typeof(Button))

{

VertexNode node = (VertexNode)(((Button)element).Tag);

foreach (uint v in vList)

{

if (v == node.VertexID)

{

makeHidden = false;

break;

}

}

}

else if (element.GetType() == typeof(Line))

{

Connection connection = (Connection)(((Line)element).Tag);

foreach (uint v in vList)

{

bool entered = false;

if (v == connection.v1ID)

{

entered = true;

foreach (uint v2 in vList) {

if (v2 == connection.v2ID) {

makeHidden = false;

break;

}

}

}

else if (v == connection.v2ID)

{

entered = true;

foreach (uint v2 in vList)

{

if (v2 == connection.v1ID)

{

makeHidden = false;

break;

}

}

}

if (entered) {

break;

}

}

}

else if (element.GetType() == typeof(Label))

{

uint id = (uint)(((Label)element).Tag);

foreach (uint v in vList)

{

if (v == id)

{

makeHidden = false;

break;

}

}

}

if (makeHidden)

{

element.Visibility = Visibility.Collapsed;

}

else {

element.Visibility = Visibility.Visible;

}

}

particularEventIsSelected = true;

canvas.Cursor = Cursors.Arrow;

}

private void makeAllElementsOnCanvasVisible() {

canvas.Cursor = Cursors.Cross;

particularEventIsSelected = false;

foreach (UIElement element in canvas.Children) {

element.Visibility = Visibility.Visible;

}

}

private void chooseEventCB\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

ComboBoxItem cbi = (ComboBoxItem)(chooseEventCB.SelectedItem);

uint eventID;

if (cbi == null)

return;

if (cbi.Tag.GetType() == typeof(int))

{

eventID = 0;

setEventsFieldsEnabled(false);

}

else {

eventID = ((Event)cbi.Tag).ID;

setEventsFieldsEnabled(true);

}

drawParticularEvent(eventID);

}

private void exitParticularEvent() {

chooseEventCB.SelectedIndex = 0;

particularEventIsSelected = false;

}

private void saveGVSS\_Click(object sender, RoutedEventArgs e)

{

exitParticularEvent();

SaveGVSS();

}

private void loadGVSS\_Click(object sender, RoutedEventArgs e)

{

exitParticularEvent();

LoadGVSS();

}

private void setEventsFieldsEnabled(bool value) {

getTimeButton.IsEnabled = value;

}

private void indexTxtBox\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

InputCorrector.IndexTB\_PreviewTextInput(sender, e);

}

private void capLetTxtBox\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

InputCorrector.CapitalLetterTB\_PreviewTextInput(sender, e);

}

private void textBoxWithoutSpaceKey\_PreviewKeyDown(object sender, KeyEventArgs e)

{

InputCorrector.textBoxWithoutSpaceKey\_PreviewKeyDown(sender, e);

}

private void auxiliaryParamRegisterDescription\_TextChanged(object sender, TextChangedEventArgs e)

{

setRegisterAuxiliaryParamButton();

}

private void setRegisterAuxiliaryParamButton()

{

InputCorrector.setButton(registerButton,

indexTxtBox,

capLetTxtBox,

auxiliaryParamRegisterDescription,

g.getAllSensors(),

uniqueCodeRadioButtons,

registerAuxiliaryParamInfo);

}

private void nameParam\_TextChanged(object sender, TextChangedEventArgs e)

{

InputCorrector.changeSensorNameRepresentation(capLetTxtBox,

indexTxtBox,

namePreviewLabel,

uniqueCodeRadioButtons,

borderAroundUniqueCodePreview,

uniqueCodePreviewDock, ref checkedArrowRB);

setRegisterAuxiliaryParamButton();

}

private void controlMethodCB\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

InputCorrector.ControlMethodCB\_SelectionChanged(sender, e,

controlAgentTBlock,

requestBodyCB,

requestBodyBorder,

requestPlacementMethodCB,

requestPlacementMethodBorder);

}

private void requestBodyCB\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

InputCorrector.requestBodyCB\_SelectionChanged(sender, e, requestPlacementMethodCB);

}

private void getTimeButton\_Click(object sender, RoutedEventArgs e)

{

Event ev = (Event)(((ComboBoxItem)(chooseEventCB.SelectedItem)).Tag);

ShowTime show = new ShowTime(ev);

show.Owner = this;

show.WindowStartupLocation = WindowStartupLocation.CenterOwner;

show.ShowDialog();

}

private void settingsGVSSCB\_SelectionChanged(object sender, SelectionChangedEventArgs e){

if (initializationFinished) {

load\_chooseEventCB(chooseEventCB.SelectedIndex);

}

}

private void newGVSSButton\_Click(object sender, RoutedEventArgs e)

{

if (endWithGVSS("Хотите ли вы сохранить эту ГВСС перед созданием новой?") == true) {

removeVertexes(g.getVertexesIDs());

create\_GVSS();

clearInterface();

}

}

private bool endWithGVSS(string warningMessageText) {

if (!g.isNull())

{

if (warningMB(warningMessageText) == MessageBoxResult.Yes)

{

if (SaveGVSS() == false)

return false;

}

}

return true;

}

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

if (endWithGVSS(DefaultSettings.getMessageBoxText(MessageBoxMessages.DoYouWantToSaveGVSSBeforeQuit)) == false)

e.Cancel = true;

}

private List<double> FindYArray(string filePath)

{

if (File.Exists(filePath))

{

string txt = File.ReadAllText(filePath);

txt = txt.Replace("\r\n", " ");

Regex resultPattern = new Regex(@"y1=\[.\*?\]");

Regex numPattern = new Regex(@"(\s\d+\.\d+)");

List<double> doubleValues = new List<double>();

foreach (Match m in numPattern.Matches(resultPattern.Match(txt).Value))

{

doubleValues.Add(Double.Parse(m.Value));

}

return doubleValues;

}

throw new UnexpectedError();

}

private void lanModeRB\_Click(object sender, RoutedEventArgs e)

{

LanguageMode newLan = (LanguageMode)((RadioButton)sender).Tag;

if (newLan != currentLanguage)

{

currentLanguage = newLan;

DefaultSettings.LanguageMode = newLan;

setUpInterface();

}

}

private void setUpInterface() {

setUpButtons();

setUpLabels();

setUpToolTips();

setUpContextMenu();

setUpComboBoxes();

fillManagePanel();

setRegisterAuxiliaryParamButton();

}

private void setUpButtons() {

registerButton.Content = DefaultSettings.getButtonText(Buttons.Register);

editButton.Content = DefaultSettings.getButtonText(Buttons.Edit);

deleteButton.Content = DefaultSettings.getButtonText(Buttons.Delete);

getTimeButton.Content = DefaultSettings.getButtonText(Buttons.GetTime);

}

private void setUpLabels() {

lanModeLabel.Content = DefaultSettings.getLabelText(Labels.LanguageMode);

registerAuxiliaryParamLabel.Content = DefaultSettings.getLabelText(Labels.RegisterAuxiliaryParam);

letterLabel.Text = DefaultSettings.getLabelText(Labels.Letter);

indexLabel.Text = DefaultSettings.getLabelText(Labels.Index);

requestBodyLabel.Text = DefaultSettings.getLabelText(Labels.RequestBody);

requestPlacementMethodLabel.Text = DefaultSettings.getLabelText(Labels.RequestPlacementMethod);

descriptionLabel.Text = DefaultSettings.getLabelText(Labels.Description);

managmentLabel.Content = DefaultSettings.getLabelText(Labels.Managment);

auxiliaryParamOrViolationLabel.Content = DefaultSettings.getLabelText(Labels.AuxiliaryParamAndViolation);

controlMethodInfoLabel.Text = DefaultSettings.getLabelText(Labels.ControlMethod);

controlMethodLabel.Text = DefaultSettings.getLabelText(Labels.ControlMethod);

requestBodyInfoLabel.Text = DefaultSettings.getLabelText(Labels.RequestBody);

requestPlacementMethodInfoLabel.Text = DefaultSettings.getLabelText(Labels.RequestPlacementMethod);

descriptionInfoLabel.Text = DefaultSettings.getLabelText(Labels.Description);

GVSSSetUpLabel.Content = DefaultSettings.getLabelText(Labels.GVSSSettings);

numOfTechLinesLabel.Text = DefaultSettings.getLabelText(Labels.NumOfTechLines);

operatorLevelLabel.Text = DefaultSettings.getLabelText(Labels.OperatorLevel);

informationRepresentationMethodLabel.Text = DefaultSettings.getLabelText(Labels.InformationRepresentationMethod);

eventLabel.Content = DefaultSettings.getLabelText(Labels.Event);

chooseEventLabel.Content = DefaultSettings.getLabelText(Labels.ChooseEvent);

}

private void setUpToolTips() {

uniqueViolationsModeRB.ToolTip = new TextBlock { Text = DefaultSettings.getToolTipText(ToolTips.ViolationsAreUnique), Width = 180, FontSize = 14, FontWeight = FontWeights.DemiBold, TextWrapping = TextWrapping.Wrap};

repetativeViolationModeRB.ToolTip = new TextBlock { Text = DefaultSettings.getToolTipText(ToolTips.ViolationsCanBeParams), Width = 170, FontSize = 14, FontWeight = FontWeights.DemiBold, TextWrapping = TextWrapping.Wrap };

saveGVSS.ToolTip = DefaultSettings.getToolTipText(ToolTips.SaveGVSS);

loadGVSS.ToolTip = DefaultSettings.getToolTipText(ToolTips.LoadGVSS);

newGVSSButton.ToolTip = DefaultSettings.getToolTipText(ToolTips.CreateNewGVSS);

photoButton.ToolTip = DefaultSettings.getToolTipText(ToolTips.SavePhoto);

}

private void setUpContextMenu() {

foreach (UIElement uIElement in canvas.Children) {

if (uIElement.GetType() == typeof(Button)) {

Button b = (Button)uIElement;

if (b.Style == Resources["Main"] as Style)

continue;

((MenuItem)b.ContextMenu.Items[0]).Header = DefaultSettings.getContextMenuHeaderText(ContextMenuHeaders.Edit);

((MenuItem)b.ContextMenu.Items[1]).Header = DefaultSettings.getContextMenuHeaderText(ContextMenuHeaders.DeleteVertex);

}

}

}

private void setUpComboBoxes() {

InputCorrector.LocalizeComboBoxes(controlMethodCB, controlAgentCB, requestBodyCB, requestPlacementMethodCB, informationRepresentationMethodCB);

((ComboBoxItem)chooseEventCB.Items[0]).Content = DefaultSettings.getWordValue(Words.All);

}

}

}

***MathLABResultWindow.xaml.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace gvss\_project

{

public class EventDifficulty {

public int EventID { get; set; }

public string EventTime {

get {

return \_eventTime.ToString() + " сек.";

} }

private double \_eventTime;

public EventDifficulty(int id, double infDif) {

EventID = id;

\_eventTime = infDif;

}

}

public partial class MathLABResultWindow : Window

{

private const int ukrWidth = 320;

private const int ruWidth = 420;

public MathLABResultWindow(List<double> doubleNumbers)

{

InitializeComponent();

LocalizeWindow();

List<EventDifficulty> ed = new List<EventDifficulty>();

int id = 1;

foreach (double num in doubleNumbers)

{

ed.Add(new EventDifficulty(id++, num));

}

resultDataGrid.ItemsSource = ed;

}

private void LocalizeWindow()

{

this.Title = DefaultSettings.getWindowTitleText(WindowTitles.MathLABResults);

this.Width = (DefaultSettings.LanguageMode == LanguageMode.Russian) ? ruWidth : ukrWidth;

((DataGridTextColumn)resultDataGrid.Columns[0]).Header = "№ " + DefaultSettings.getWordValue(Words.EventID);

((DataGridTextColumn)resultDataGrid.Columns[1]).Header = DefaultSettings.getWordValue(Words.EventTime);

}

}

}

***ParameterValueHelper.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

namespace gvss\_project

{

public static class ParameterValueHelper

{

public static readonly Dictionary<OperatorLevel, string> OperatorLevels = new Dictionary<OperatorLevel, string>() {

{ OperatorLevel.Second, "II"},

{ OperatorLevel.Third, "III"},

{ OperatorLevel.Forth, "IV"},

{ OperatorLevel.Fifth, "V"}

};

private static ComboBoxItem createComboBoxItem(string content, object tag)

{

ComboBoxItem cbi = new ComboBoxItem();

cbi.Tag = tag;

cbi.Content = content;

return cbi;

}

public static void InitializeComboBox(ComboBox cb, Type enumType)

{

if (enumType == typeof(SensorControlMethod))

{

cb.Items.Clear();

foreach (SensorControlMethod cm in Enum.GetValues(typeof(SensorControlMethod)))

{

if (cm == SensorControlMethod.None)

continue;

cb.Items.Add(createComboBoxItem(DefaultSettings.getControlMethodText(cm), cm));

}

}

else if (enumType == typeof(SensorControlAgent))

{

cb.Items.Clear();

foreach (SensorControlAgent ca in Enum.GetValues(typeof(SensorControlAgent)))

{

if (ca == SensorControlAgent.None)

continue;

cb.Items.Add(createComboBoxItem(DefaultSettings.getControlAgentText(ca), ca));

}

}

else if (enumType == typeof(InformationRepresentationMethod))

{

cb.Items.Clear();

foreach (InformationRepresentationMethod irm in Enum.GetValues(typeof(InformationRepresentationMethod)))

{

if (irm == InformationRepresentationMethod.None)

continue;

cb.Items.Add(createComboBoxItem(DefaultSettings.getInformationRepresentationMethodText(irm), irm));

}

}

else if (enumType == typeof(SensorRequestBodies))

{

cb.Items.Clear();

foreach (SensorRequestBodies rb in Enum.GetValues(typeof(SensorRequestBodies)))

{

if (rb == SensorRequestBodies.None)

continue;

cb.Items.Add(createComboBoxItem(DefaultSettings.getRequestBodyText(rb), rb));

}

}

else if (enumType == typeof(OperatorLevel)) {

cb.Items.Clear();

foreach (OperatorLevel ol in Enum.GetValues(typeof(OperatorLevel)))

{

cb.Items.Add(createComboBoxItem(OperatorLevels[ol], ol));

}

}

else

{

throw new InvalidArgument();

}

}

public static void InitializeComboBoxAndSelectIndex(ComboBox cb, Type enumType, int index = 0)

{

InitializeComboBox(cb, enumType);

cb.SelectedIndex = index;

}

public static void InitializeComboBoxWithNone(ComboBox cb, Type enumType)

{

if (enumType == typeof(SensorRequestBodies))

{

cb.Items.Clear();

cb.Items.Add(createComboBoxItem("", SensorRequestBodies.None));

}

else if (enumType == typeof(SensorRequestPlacementMethod))

{

cb.Items.Clear();

cb.Items.Add(createComboBoxItem("", SensorRequestPlacementMethod.None));

}

else

{

throw new InvalidArgument();

}

}

public static void InitializeComboBoxWithNoneAndSelectIndex(ComboBox cb, Type enumType, int index = 0)

{

InitializeComboBoxWithNone(cb, enumType);

cb.SelectedIndex = index;

}

public static void InitializeRequestPlacementMethod(ComboBox cb, SensorRequestBodies requestBody)

{

if (requestBody != SensorRequestBodies.None)

{

cb.Items.Clear();

}

else

{

return;

}

if (requestBody == SensorRequestBodies.ToggleSwitch || requestBody == SensorRequestBodies.Button)

{

cb.Items.Add(createComboBoxItem(DefaultSettings.getRequestPlacementMethodText(SensorRequestPlacementMethod.MnemonicDiagram), SensorRequestPlacementMethod.MnemonicDiagram));

cb.Items.Add(createComboBoxItem(DefaultSettings.getRequestPlacementMethodText(SensorRequestPlacementMethod.GroupMethod), SensorRequestPlacementMethod.GroupMethod));

cb.Items.Add(createComboBoxItem(DefaultSettings.getRequestPlacementMethodText(SensorRequestPlacementMethod.BlockDiagramMethod), SensorRequestPlacementMethod.BlockDiagramMethod));

cb.Items.Add(createComboBoxItem(DefaultSettings.getRequestPlacementMethodText(SensorRequestPlacementMethod.UnorganizedMethod), SensorRequestPlacementMethod.UnorganizedMethod));

cb.SelectedIndex = 0;

}

else if (requestBody == SensorRequestBodies.Dialer || requestBody == SensorRequestBodies.KeyboardDevice)

{

cb.Items.Add(createComboBoxItem(DefaultSettings.getRequestPlacementMethodText(SensorRequestPlacementMethod.Separately), SensorRequestPlacementMethod.Separately));

cb.Items.Add(createComboBoxItem(DefaultSettings.getRequestPlacementMethodText(SensorRequestPlacementMethod.GroupMethod), SensorRequestPlacementMethod.GroupMethod));

cb.SelectedIndex = 0;

}

}

}

}

***RenderVisualService.cs:***

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace gvss\_project

{

public static class RenderVisualService

{

private const double defaultDpi = 96.0;

public static void RenderToPNGFile(Visual targetControl, string filename)

{

var renderTargetBitmap = GetRenderTargetBitmapFromControl(targetControl);

var encoder = new PngBitmapEncoder();

encoder.Frames.Add(BitmapFrame.Create(renderTargetBitmap));

var result = new BitmapImage();

try

{

using (var fileStream = new FileStream(filename, FileMode.Create))

{

encoder.Save(fileStream);

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine($"There was an error saving the file: {ex.Message}");

}

}

private static BitmapSource GetRenderTargetBitmapFromControl(Visual targetControl, double dpi = defaultDpi)

{

if (targetControl == null) return null;

var bounds = VisualTreeHelper.GetDescendantBounds(targetControl);

var renderTargetBitmap = new RenderTargetBitmap((int)(bounds.Width \* dpi / 96.0),

(int)(bounds.Height \* dpi / 96.0),

dpi,

dpi,

PixelFormats.Pbgra32);

var drawingVisual = new DrawingVisual();

using (var drawingContext = drawingVisual.RenderOpen())

{

var visualBrush = new VisualBrush(targetControl);

if (targetControl.GetType() == typeof(Canvas))

{

drawingContext.DrawRectangle(Brushes.White, null, new Rect(new Point(), bounds.Size));

}

drawingContext.DrawRectangle(visualBrush, null, new Rect(new Point(), bounds.Size));

}

renderTargetBitmap.Render(drawingVisual);

return renderTargetBitmap;

}

}

}

***Sensor.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace gvss\_project

{

[Serializable]

public class Sensor

{

private char \_capitalLetter;

public char CapitalLetter

{

get

{

return \_capitalLetter;

}

set

{

if (Char.IsLetter(value))

{

\_capitalLetter = Char.ToUpper(value);

}

else

{

throw new InvalidArgument();

}

}

}

public int Index { get; set; }

public string Description { get; set; }

public uint NumOfRequestInputKeys { get; set; }

public SensorControlMethod ControlMethod { get; set; }

public SensorControlAgent ControlAgent { get; set; }

public SensorArrowSign ArrowSign { get; set; }

public SensorRequestBodies RequestBody { get; set; }

public SensorRequestPlacementMethod RequestPlacementMethod{ get; set; }

public string UniqueCode {

get {

int u = 0x02b9b + (int)ArrowSign;

string str = CapitalLetter.ToString();

foreach (char c in Index.ToString()) {

str += ((char)(0x2080 + Int32.Parse(c.ToString()))).ToString();

}

str += ((char)u).ToString();

return str;

}

}

public bool isValid() {

return \_capitalLetter != '0' &&

Index != -1 &&

ArrowSign != SensorArrowSign.None &&

ControlMethod != SensorControlMethod.None &&

ControlAgent != SensorControlAgent.None;

}

public static bool operator ==(Sensor s1, Sensor s2) {

return (s1.CapitalLetter == s2.CapitalLetter)

&& (s1.Index == s2.Index)

&& (s1.ArrowSign == s2.ArrowSign);

}

public static bool operator !=(Sensor s1, Sensor s2) {

return (s1.CapitalLetter != s2.CapitalLetter)

|| (s1.Index != s2.Index)

|| (s1.ArrowSign != s2.ArrowSign);

}

public override bool Equals(object obj)

{

if ((obj == null) || !this.GetType().Equals(obj.GetType()))

{

return false;

}

else {

Sensor s = (Sensor)obj;

return (CapitalLetter == s.CapitalLetter)

&& (Index == s.Index)

&& (ArrowSign == s.ArrowSign);

}

}

public Sensor() {

\_capitalLetter = '0';

Index = -1;

Description = "";

ControlMethod = SensorControlMethod.None;

ArrowSign = SensorArrowSign.None;

ControlAgent = SensorControlAgent.None;

RequestBody = SensorRequestBodies.None;

RequestPlacementMethod = SensorRequestPlacementMethod.None;

NumOfRequestInputKeys = 3;

}

}

}

***Settings.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace gvss\_project

{

public static class DefaultSettings {

private static LanguageMode lanMode;

public static readonly Dictionary<OperatorLevel, (double, double)> OperatorCoefs = new Dictionary<OperatorLevel, (double, double)> {

{OperatorLevel.Second, (0.69,0.18) },

{OperatorLevel.Third, (0.58,0.09) },

{OperatorLevel.Forth, (0.46,0.03) },

{OperatorLevel.Fifth, (0.42,0.025) }

};

public static readonly Dictionary<(SensorRequestBodies, SensorRequestPlacementMethod), double> RequestTime = new Dictionary<(SensorRequestBodies, SensorRequestPlacementMethod), double>() {

{ (SensorRequestBodies.ToggleSwitch, SensorRequestPlacementMethod.MnemonicDiagram), 0.2},

{ (SensorRequestBodies.ToggleSwitch, SensorRequestPlacementMethod.GroupMethod), 0.24},

{ (SensorRequestBodies.ToggleSwitch, SensorRequestPlacementMethod.BlockDiagramMethod), 0.15},

{ (SensorRequestBodies.ToggleSwitch, SensorRequestPlacementMethod.UnorganizedMethod), 0.3},

{ (SensorRequestBodies.Button, SensorRequestPlacementMethod.MnemonicDiagram), 0.12},

{ (SensorRequestBodies.Button, SensorRequestPlacementMethod.GroupMethod), 0.16},

{ (SensorRequestBodies.Button, SensorRequestPlacementMethod.BlockDiagramMethod), 0.07},

{ (SensorRequestBodies.Button, SensorRequestPlacementMethod.UnorganizedMethod), 0.23},

{ (SensorRequestBodies.Dialer, SensorRequestPlacementMethod.Separately), 1.8},

{ (SensorRequestBodies.KeyboardDevice, SensorRequestPlacementMethod.Separately), 0.15},

{ (SensorRequestBodies.Dialer, SensorRequestPlacementMethod.GroupMethod), 1.8},

{ (SensorRequestBodies.KeyboardDevice, SensorRequestPlacementMethod.GroupMethod), 0.15}

};

public static readonly Dictionary<SensorControlAgent, double> ControlAgentsTime = new Dictionary<SensorControlAgent, double>() {

{ SensorControlAgent.SignalingDevice, 0},

{ SensorControlAgent.WeakDampedDevice, 1},

{ SensorControlAgent.HighlyDampedDevice, 0.4},

{ SensorControlAgent.DigitalThreeDigitIndicator, 0.35}

};

public static readonly Dictionary<InformationRepresentationMethod, double> IndformationRepresentationMethodCoefs = new Dictionary<InformationRepresentationMethod, double>() {

{ InformationRepresentationMethod.MnemonicDiagram, 1},

{ InformationRepresentationMethod.GroupMethod, 0.85},

{ InformationRepresentationMethod.BlockDiagramMethod, 1.2}

};

private static readonly Dictionary<(LanguageMode, Labels), string> LabelsText = new Dictionary<(LanguageMode, Labels), string>() {

{ (LanguageMode.Russian, Labels.LanguageMode), "Язык"},

{ (LanguageMode.Ukrainian, Labels.LanguageMode), "Мова"},

{ (LanguageMode.Russian, Labels.RegisterAuxiliaryParam), "Зарегистрировать признак"},

{ (LanguageMode.Ukrainian, Labels.RegisterAuxiliaryParam), "Зареєструвати ознаку"},

{ (LanguageMode.Russian, Labels.Letter), "Буква"},

{ (LanguageMode.Ukrainian, Labels.Letter), "Літера"},

{ (LanguageMode.Russian, Labels.Index), "Индекс"},

{ (LanguageMode.Ukrainian, Labels.Index), "Індекс"},

{ (LanguageMode.Russian, Labels.ControlMethod), "Способ контроля"},

{ (LanguageMode.Ukrainian, Labels.ControlMethod), "Спосіб контролю"},

{ (LanguageMode.Russian, Labels.ControlAgentAuto), "Средство контроля параметров"},

{ (LanguageMode.Ukrainian, Labels.ControlAgentAuto), "Засіб контролю параметрів"},

{ (LanguageMode.Russian, Labels.ControlAgentRequest), "Средство запроса (прибор)"},

{ (LanguageMode.Ukrainian, Labels.ControlAgentRequest), "Засіб запиту (пристрій)"},

{ (LanguageMode.Russian, Labels.RequestBody), "Орган запроса"},

{ (LanguageMode.Ukrainian, Labels.RequestBody), "Орган запиту"},

{ (LanguageMode.Russian, Labels.RequestPlacementMethod), "Размещение органа запроса"},

{ (LanguageMode.Ukrainian, Labels.RequestPlacementMethod), "Розміщення органу запиту"},

{ (LanguageMode.Russian, Labels.Description), "Краткое описание"},

{ (LanguageMode.Ukrainian, Labels.Description), "Стислий опис"},

{ (LanguageMode.Russian, Labels.Managment), "Управление"},

{ (LanguageMode.Ukrainian, Labels.Managment), "Управління"},

{ (LanguageMode.Russian, Labels.AuxiliaryParamAndViolation), "Признак/нарушение"},

{ (LanguageMode.Ukrainian, Labels.AuxiliaryParamAndViolation), "Ознака/порушення"},

{ (LanguageMode.Russian, Labels.GVSSSettings), "Настройки ГВСС"},

{ (LanguageMode.Ukrainian, Labels.GVSSSettings), "Налаштування ГВСС"},

{ (LanguageMode.Russian, Labels.NumOfTechLines), "Количество техн. линий"},

{ (LanguageMode.Ukrainian, Labels.NumOfTechLines), "Кількість техн. ліній"},

{ (LanguageMode.Russian, Labels.OperatorLevel), "Уровень тренированности оператора"},

{ (LanguageMode.Ukrainian, Labels.OperatorLevel), "Рівень тренованості оператора"},

{ (LanguageMode.Russian, Labels.InformationRepresentationMethod), "Метод представления информации"},

{ (LanguageMode.Ukrainian, Labels.InformationRepresentationMethod), "Спосіб представлення інформації"},

{ (LanguageMode.Russian, Labels.Event), "Событие"},

{ (LanguageMode.Ukrainian, Labels.Event), "Подія"},

{ (LanguageMode.Russian, Labels.ChooseEvent), "Выберите событие"},

{ (LanguageMode.Ukrainian, Labels.ChooseEvent), "Оберіть подію"},

{ (LanguageMode.Russian, Labels.InformationQuantity), "Количество информации"},

{ (LanguageMode.Ukrainian, Labels.InformationQuantity), "Кількість інформації"},

{ (LanguageMode.Russian, Labels.OriginalTime), "Исходное время"},

{ (LanguageMode.Ukrainian, Labels.OriginalTime), "Початковий час"},

{ (LanguageMode.Russian, Labels.RequestTime), "Время потр. на запросы"},

{ (LanguageMode.Ukrainian, Labels.RequestTime), "Час, витрачений на запити"},

{ (LanguageMode.Russian, Labels.IrrelevantInformationTime), "Задержка (ир. информ.)"},

{ (LanguageMode.Ukrainian, Labels.IrrelevantInformationTime), "Затримка (ір. інформ.)"},

{ (LanguageMode.Russian, Labels.AgentTimeInfluence), "Влияние средств предст."},

{ (LanguageMode.Ukrainian, Labels.AgentTimeInfluence), "Вплив засобів предст."},

{ (LanguageMode.Russian, Labels.FindControlAgentTime), "Поиск средств предст."},

{ (LanguageMode.Ukrainian, Labels.FindControlAgentTime), "Пошук засобів предст."},

{ (LanguageMode.Russian, Labels.TotalTime), "Общее время обнаружения события"},

{ (LanguageMode.Ukrainian, Labels.TotalTime), "Загальний час виявлення події"},

{ (LanguageMode.Russian, Labels.AddViolation), "Добавить нарушение"},

{ (LanguageMode.Ukrainian, Labels.AddViolation), "Додати порушення"},

{ (LanguageMode.Russian, Labels.Violation), "Нарушение"},

{ (LanguageMode.Ukrainian, Labels.Violation), "Порушення"},

{ (LanguageMode.Russian, Labels.ChooseAction), "Выберите действие"},

{ (LanguageMode.Ukrainian, Labels.ChooseAction), "Оберіть дію"},

{ (LanguageMode.Russian, Labels.FailureRate), "Частота выхода из строя"},

{ (LanguageMode.Ukrainian, Labels.FailureRate), "Частота виходу з ладу"},

{ (LanguageMode.Russian, Labels.AuxiliaryParam), "Признак"},

{ (LanguageMode.Ukrainian, Labels.AuxiliaryParam), "Ознака"},

{ (LanguageMode.Russian, Labels.AddAuxiliaryParamToVertex), "Добавить признак к вершине"},

{ (LanguageMode.Ukrainian, Labels.AddAuxiliaryParamToVertex), "Додати ознаку до вершини"},

};

private static readonly Dictionary<(LanguageMode, Buttons), string> ButtonsText = new Dictionary<(LanguageMode, Buttons), string>() {

{ (LanguageMode.Russian, Buttons.Register), "Зарегистрировать"},

{ (LanguageMode.Ukrainian, Buttons.Register), "Зареєструвати"},

{ (LanguageMode.Russian, Buttons.Edit), "Редактировать"},

{ (LanguageMode.Ukrainian, Buttons.Edit), "Редагувати"},

{ (LanguageMode.Russian, Buttons.Delete), "Удалить"},

{ (LanguageMode.Ukrainian, Buttons.Delete), "Видалити"},

{ (LanguageMode.Russian, Buttons.GetTime), "Получить время обнаружения"},

{ (LanguageMode.Ukrainian, Buttons.GetTime), "Отримати час виявлення"},

{ (LanguageMode.Russian, Buttons.Add), "Добавить"},

{ (LanguageMode.Ukrainian, Buttons.Add), "Додати"},

{ (LanguageMode.Russian, Buttons.Change), "Изменить"},

{ (LanguageMode.Ukrainian, Buttons.Change), "Змінити"}

};

private static readonly Dictionary<(LanguageMode, ToolTips), string> ToolTipsText = new Dictionary<(LanguageMode, ToolTips), string>() {

{ (LanguageMode.Russian, ToolTips.CreateNewGVSS), "Создать новую ГВСС"},

{ (LanguageMode.Ukrainian, ToolTips.CreateNewGVSS), "Створити нову ГВСС"},

{ (LanguageMode.Russian, ToolTips.LoadGVSS), "Загрузить"},

{ (LanguageMode.Ukrainian, ToolTips.LoadGVSS), "Завантажити"},

{ (LanguageMode.Russian, ToolTips.SaveGVSS), "Сохранить"},

{ (LanguageMode.Ukrainian, ToolTips.SaveGVSS), "Зберегти"},

{ (LanguageMode.Russian, ToolTips.SavePhoto), "Сохранить фото структуры"},

{ (LanguageMode.Ukrainian, ToolTips.SavePhoto), "Зберегти фото структури"},

{ (LanguageMode.Russian, ToolTips.ViolationsAreUnique), "Нарушения - уникальны. Регистрируются отдельно"},

{ (LanguageMode.Ukrainian, ToolTips.ViolationsAreUnique), "Порушення - унікальні. Реєструються окремо"},

{ (LanguageMode.Russian, ToolTips.ViolationsCanBeParams), "Нарушения могут быть признаками других нарушений"},

{ (LanguageMode.Ukrainian, ToolTips.ViolationsCanBeParams), "Порушення можуть бути ознаками інших порушень"}

};

private static readonly Dictionary<(LanguageMode, ErrorMessages), string> ErrorMessagesText = new Dictionary<(LanguageMode, ErrorMessages), string>() {

{ (LanguageMode.Russian, ErrorMessages.WhileAddRepetative), "Невозможно добавить признак в это поддерево"},

{ (LanguageMode.Ukrainian, ErrorMessages.WhileAddRepetative), "Неможливо додати ознаку в це піддерево"},

{ (LanguageMode.Russian, ErrorMessages.WhileASVFileCorrupted), "Файл .avs поврежден или внутренняя структура нарушена"},

{ (LanguageMode.Ukrainian, ErrorMessages.WhileASVFileCorrupted), "Файл .avs пошкоджений або внутрішня структура порушена"},

{ (LanguageMode.Russian, ErrorMessages.WhileGVSSFileCorrupted), "Файл .gvss поврежден или внутренняя структура нарушена"},

{ (LanguageMode.Ukrainian, ErrorMessages.WhileGVSSFileCorrupted), "Файл .gvvs пошкоджений або внутрішня структура порушена"},

{ (LanguageMode.Russian, ErrorMessages.TheNameExistsAlready), "\* имя недоступно"},

{ (LanguageMode.Ukrainian, ErrorMessages.TheNameExistsAlready), "\* ім'я вже зайняте"},

{ (LanguageMode.Russian, ErrorMessages.NotAllFieldsAreFilled), "\* не все поля заполненны"},

{ (LanguageMode.Ukrainian, ErrorMessages.NotAllFieldsAreFilled), "\* не всі поля заповнені"}

};

private static readonly Dictionary<(LanguageMode, MessageBoxMessages), string> MessageBoxText = new Dictionary<(LanguageMode, MessageBoxMessages), string>() {

{ (LanguageMode.Russian, MessageBoxMessages.ChangeMode), "Изменение режима может привести к удалению некоторых вершин. Продолжить?"},

{ (LanguageMode.Ukrainian, MessageBoxMessages.ChangeMode), "Зміна режиму може привести до видалення деяких вершин. Продовжити?"},

{ (LanguageMode.Russian, MessageBoxMessages.DeleteVertex), "Удаление этой вершины приведет к удалению всех дочерних вершин. Продолжить?"},

{ (LanguageMode.Ukrainian, MessageBoxMessages.DeleteVertex), "Видалення цієї вершини призведе до видалення всіх дочірніх вершин. Продовжити?"},

{ (LanguageMode.Russian, MessageBoxMessages.DoYouWantToSaveGVSSBeforeQuit), "Хотите ли вы сохранить эту ГВСС перед выходом?"},

{ (LanguageMode.Ukrainian, MessageBoxMessages.DoYouWantToSaveGVSSBeforeQuit), "Чи бажаєте ви зберегти цю ГВСС перед виходом?"},

{ (LanguageMode.Russian, MessageBoxMessages.FullDeleteParam), "Все вершины и признаки с таким уникальным кодом будут удалены. Продолжить?"},

{ (LanguageMode.Ukrainian, MessageBoxMessages.FullDeleteParam), "Всі вершини і ознаки з таким унікальним кодом будуть видалені. Продовжити?"},

{ (LanguageMode.Russian, MessageBoxMessages.SaveGVSSBeforeLoad), "Хотите ли вы сохранить эту ГВСС перед загрузкой?"},

{ (LanguageMode.Ukrainian, MessageBoxMessages.SaveGVSSBeforeLoad), "Чи бажаєте ви зберегти цю ГВСС перед завантаженням?"}

};

private static readonly Dictionary<(LanguageMode, ContextMenuHeaders), string> ContextMenuHeadersText = new Dictionary<(LanguageMode, ContextMenuHeaders), string>() {

{ (LanguageMode.Russian, ContextMenuHeaders.DeleteVertex), "Удалить вершину"},

{ (LanguageMode.Ukrainian, ContextMenuHeaders.DeleteVertex), "Видалити вершину"},

{ (LanguageMode.Russian, ContextMenuHeaders.Edit), "Редактировать"},

{ (LanguageMode.Ukrainian, ContextMenuHeaders.Edit), "Редагувати"}

};

private static readonly Dictionary<(LanguageMode, WindowTitles), string> WindowsTitlesText = new Dictionary<(LanguageMode, WindowTitles), string>() {

{ (LanguageMode.Russian, WindowTitles.Add), "Добавить"},

{ (LanguageMode.Ukrainian, WindowTitles.Add), "Додати"},

{ (LanguageMode.Russian, WindowTitles.Edit), "Редактировать"},

{ (LanguageMode.Ukrainian, WindowTitles.Edit), "Редагувати"},

{ (LanguageMode.Russian, WindowTitles.ErrorMB), "Ошибка"},

{ (LanguageMode.Ukrainian, WindowTitles.ErrorMB), "Помилка"},

{ (LanguageMode.Russian, WindowTitles.EventDetectionTime), "Время обнаружения события"},

{ (LanguageMode.Ukrainian, WindowTitles.EventDetectionTime), "Час розпізнавання події"},

{ (LanguageMode.Russian, WindowTitles.WarningMB), "Предупреждение"},

{ (LanguageMode.Ukrainian, WindowTitles.WarningMB), "Попередження"},

{ (LanguageMode.Russian, WindowTitles.MathLABResults), "MathLAB результаты"},

{ (LanguageMode.Ukrainian, WindowTitles.MathLABResults), "MathLAB результати"}

};

private static readonly Dictionary<(LanguageMode, SensorControlMethod), string> ControlMethodsText = new Dictionary<(LanguageMode, SensorControlMethod), string>(){

{ (LanguageMode.Russian, SensorControlMethod.Auto), "Автоматическое предъявление" },

{ (LanguageMode.Ukrainian, SensorControlMethod.Auto), "Автоматичне пред'явлення" },

{ (LanguageMode.Russian, SensorControlMethod.Request), "Запрос значений"},

{ (LanguageMode.Ukrainian, SensorControlMethod.Request), "Запит значень"}

/\*{ SensorControlMethod.Forecasting, "Контроль с прогназированием"}\*/

};

private static readonly Dictionary<(LanguageMode, SensorControlAgent), string> ControlAgentsText = new Dictionary<(LanguageMode, SensorControlAgent), string>(){

{ (LanguageMode.Russian, SensorControlAgent.SignalingDevice), "Сигнализатор"},

{ (LanguageMode.Ukrainian, SensorControlAgent.SignalingDevice), "Сигналізатор"},

{ (LanguageMode.Russian, SensorControlAgent.WeakDampedDevice), "Слабодемпфирующее устройство" },

{ (LanguageMode.Ukrainian, SensorControlAgent.WeakDampedDevice), "Слабкодемпфуючий пристрій" },

{ (LanguageMode.Russian, SensorControlAgent.HighlyDampedDevice), "Сильнодемпфирующее устройство"},

{ (LanguageMode.Ukrainian, SensorControlAgent.HighlyDampedDevice), "Сильнодемпфуючий пристрій"},

{ (LanguageMode.Russian, SensorControlAgent.DigitalThreeDigitIndicator), "Цифровой трехразрядный индикатор" },

{ (LanguageMode.Ukrainian, SensorControlAgent.DigitalThreeDigitIndicator), "Цифровий трьохрозрядний індикатор" }

};

private static readonly Dictionary<(LanguageMode, InformationRepresentationMethod), string> InformationRepresentationMethodsText = new Dictionary<(LanguageMode, InformationRepresentationMethod), string>() {

{ (LanguageMode.Russian, InformationRepresentationMethod.MnemonicDiagram), "Метод мнемосхем"},

{ (LanguageMode.Ukrainian, InformationRepresentationMethod.MnemonicDiagram), "Метод мнемосхем"},

{ (LanguageMode.Russian, InformationRepresentationMethod.BlockDiagramMethod), "Метод структурных схем"},

{ (LanguageMode.Ukrainian, InformationRepresentationMethod.BlockDiagramMethod), "Метод структурних схем"},

{ (LanguageMode.Russian, InformationRepresentationMethod.GroupMethod), "Группирование по ГВСС на панели"},

{ (LanguageMode.Ukrainian, InformationRepresentationMethod.GroupMethod), "Групування по ГВСС на панелі"}

//{ InformationRepresentationMethod.GeneralizedMethod, "Метод обобщенного образа"},

//{ InformationRepresentationMethod.UnorganizedMethod, "Неорганизованное размещение"},

//{ InformationRepresentationMethod.AutoMethod, "Автоматическое предъявление"},

};

private static readonly Dictionary<(LanguageMode, SensorRequestBodies), string> RequestBodiesText = new Dictionary<(LanguageMode, SensorRequestBodies), string>()

{

{ (LanguageMode.Russian, SensorRequestBodies.ToggleSwitch), "Тумблер"},

{ (LanguageMode.Ukrainian, SensorRequestBodies.ToggleSwitch), "Тумблер"},

{ (LanguageMode.Russian, SensorRequestBodies.Button), "Кнопка"},

{ (LanguageMode.Ukrainian, SensorRequestBodies.Button), "Кнопка"},

{ (LanguageMode.Russian, SensorRequestBodies.Dialer), "Номеронабиратель"},

{ (LanguageMode.Ukrainian, SensorRequestBodies.Dialer), "Номеронабирач"},

{ (LanguageMode.Russian, SensorRequestBodies.KeyboardDevice), "Клавишное устройство"},

{ (LanguageMode.Ukrainian, SensorRequestBodies.KeyboardDevice), "Клавішний пристрій"},

{ (LanguageMode.Russian, SensorRequestBodies.None), ""},

{ (LanguageMode.Ukrainian, SensorRequestBodies.None), ""}

};

private static readonly Dictionary<(LanguageMode, SensorRequestPlacementMethod), string> RequestPlacementMethodText = new Dictionary<(LanguageMode, SensorRequestPlacementMethod), string>() {

{ (LanguageMode.Russian, SensorRequestPlacementMethod.BlockDiagramMethod), "Метод структурных схем"},

{ (LanguageMode.Ukrainian, SensorRequestPlacementMethod.BlockDiagramMethod), "Метод структурних схем"},

{ (LanguageMode.Russian, SensorRequestPlacementMethod.GroupMethod), "В агрегатной группе"},

{ (LanguageMode.Ukrainian, SensorRequestPlacementMethod.GroupMethod), "В агрегатній групі"},

{ (LanguageMode.Russian, SensorRequestPlacementMethod.MnemonicDiagram), "Размещение на мнемосхеме"},

{ (LanguageMode.Ukrainian, SensorRequestPlacementMethod.MnemonicDiagram), "Розміщення на мнемосхемі"},

{ (LanguageMode.Russian, SensorRequestPlacementMethod.Combination), "Метод совмещения с приборами"},

{ (LanguageMode.Ukrainian, SensorRequestPlacementMethod.Combination), "Метод суміщення з приладами"},

{ (LanguageMode.Russian, SensorRequestPlacementMethod.UnorganizedMethod), "Неорганизованное размещение"},

{ (LanguageMode.Ukrainian, SensorRequestPlacementMethod.UnorganizedMethod), "Неорганізоване розміщення"},

{ (LanguageMode.Russian, SensorRequestPlacementMethod.Separately), "Отдельно"},

{ (LanguageMode.Ukrainian, SensorRequestPlacementMethod.Separately), "Окремо"},

{ (LanguageMode.Russian, SensorRequestPlacementMethod.None), ""},

{ (LanguageMode.Ukrainian, SensorRequestPlacementMethod.None), ""}

};

private static readonly Dictionary<(LanguageMode, Words), string> WordsValues = new Dictionary<(LanguageMode, Words), string>() {

{ (LanguageMode.Russian, Words.All), "Все"},

{ (LanguageMode.Ukrainian, Words.All), "Всі"},

{ (LanguageMode.Russian, Words.EventID), "события"},

{ (LanguageMode.Ukrainian, Words.EventID), "події"},

{ (LanguageMode.Russian, Words.AddViolation), "Добавить НАРУШЕНИЕ"},

{ (LanguageMode.Ukrainian, Words.AddViolation), "Додати ПОРУШЕННЯ"},

{ (LanguageMode.Russian, Words.AddAuxiliaryParamToVertex), "Добавить ПРИЗНАК к ВЕРШИНЕ"},

{ (LanguageMode.Ukrainian, Words.AddAuxiliaryParamToVertex), "Додати ОЗНАКУ до ВЕРШИНИ"},

{ (LanguageMode.Russian, Words.EventTime), "Время обнаружения события"},

{ (LanguageMode.Ukrainian, Words.EventTime), "Час виявлення події"}

};

public static LanguageMode LanguageMode {

get { return lanMode; }

set { lanMode = value; }

}

static DefaultSettings() {

lanMode = LanguageMode.Ukrainian;

}

public static string getErrorMessageText(ErrorMessages errorMessage) {

return ErrorMessagesText[(LanguageMode, errorMessage)];

}

public static string getMessageBoxText(MessageBoxMessages message)

{

return MessageBoxText[(LanguageMode, message)];

}

public static string getContextMenuHeaderText(ContextMenuHeaders contextMenuHeader) {

return ContextMenuHeadersText[(LanguageMode, contextMenuHeader)];

}

public static string getWindowTitleText(WindowTitles windowTitle) {

return WindowsTitlesText[(LanguageMode, windowTitle)];

}

public static string getControlMethodText(SensorControlMethod controlMethod) {

return ControlMethodsText[(LanguageMode, controlMethod)];

}

public static string getControlAgentText(SensorControlAgent controlAgent) {

return ControlAgentsText[(LanguageMode, controlAgent)];

}

public static string getInformationRepresentationMethodText(InformationRepresentationMethod method) {

return InformationRepresentationMethodsText[(LanguageMode, method)];

}

public static string getRequestBodyText(SensorRequestBodies requestBody) {

return RequestBodiesText[(LanguageMode, requestBody)];

}

public static string getRequestPlacementMethodText(SensorRequestPlacementMethod requestPlacementMethod) {

return RequestPlacementMethodText[(LanguageMode, requestPlacementMethod)];

}

public static string getButtonText(Buttons button) {

return ButtonsText[(LanguageMode, button)];

}

public static string getLabelText(Labels label) {

return LabelsText[(LanguageMode, label)];

}

public static string getToolTipText(ToolTips toolTip) {

return ToolTipsText[(LanguageMode, toolTip)];

}

public static string getWordValue(Words word) {

return WordsValues[(LanguageMode, word)];

}

}

}

***ShowTime.xaml.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace gvss\_project

{

/// <summary>

/// Interaction logic for ShowTime.xaml

/// </summary>

public partial class ShowTime : Window

{

private string time(double value) {

return value.ToString() + " сек.";

}

public ShowTime(Event e)

{

InitializeComponent();

LocalizeWindow();

header.Content = DefaultSettings.getLabelText(Labels.Event) + " №" + e.ID.ToString();

double originalTime = e.getOriginalTime();

double requestTime = e.getRequestTime();

double irrelevantTime = e.getIrrelevantTime();

double agentTimeInfluence = e.getAgentTimeInfluence();

double findControlAgentTime = e.getFindControlAgentTime();

double totalTime = originalTime + requestTime + irrelevantTime + agentTimeInfluence + findControlAgentTime;

informationQuantityTB.Text = e.getInformationDifficulty().ToString() + " д.e.";

originalTimeTB.Text = time(originalTime);

requestTimeTB.Text = time(requestTime);

irrelevantInformationTimeTB.Text = time(irrelevantTime);

agentInfluenceTimeTB.Text = time(agentTimeInfluence);

findControlAgentTimeTB.Text = time(findControlAgentTime);

totalTimeTB.Text = time(totalTime);

}

private void LocalizeWindow() {

this.Title = DefaultSettings.getWindowTitleText(WindowTitles.EventDetectionTime);

informationQuantityLabel.Text = DefaultSettings.getLabelText(Labels.InformationQuantity);

originalTimeLabel.Text = DefaultSettings.getLabelText(Labels.OriginalTime);

requestTimeLabel.Text = DefaultSettings.getLabelText(Labels.RequestTime);

irrelevantInformationLabel.Text = DefaultSettings.getLabelText(Labels.IrrelevantInformationTime);

agentTimeInfluenceLabel.Text = DefaultSettings.getLabelText(Labels.AgentTimeInfluence);

findControlAgentTimeLabel.Text = DefaultSettings.getLabelText(Labels.FindControlAgentTime);

totalTimeLabel.Text = DefaultSettings.getLabelText(Labels.TotalTime);

}

}

}

***VertexDialog.xaml.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace gvss\_project

{

public partial class VertexDialog : Window

{

public enum VertexDialogResult {JustClosed, AddViolation, AddAuxiliaryParam};

private const uint defWindowHeight = 150;

private const uint addViolationWindowHeight = 605;

private const uint addAuxiliaryParamWindowHeight = 280;

private const uint addRepetativeViolationWindowHeight = 280;

public string AuxiliaryParam { get; private set; }

public uint ConnectedVertexID { get; private set; }

public float FailureRateViolation { get; private set; }

public char CapLetViolation { get; private set; }

public uint IndexViolation { get; private set; }

public string DescriptionViolation { get; private set; }

public SensorArrowSign ArrowViolation { get; private set; }

public SensorControlAgent ControlAgentViolation { get; private set; }

public SensorControlMethod ControlMethodViolation { get; private set; }

public SensorRequestBodies RequestBody { get; set; }

public SensorRequestPlacementMethod RequestPlacementMethod { get; set; }

public string ViolationUniqueCode { get; private set; }

public VertexDialogResult Result { get; private set; }

private enum ChooseAction { Violation, AuxiliaryParam, None};

private List<RadioButton> chooseActionRBs;

private ChooseAction actionRB;

private SensorArrowSign checkedArrowRB;

private List<RadioButton> uniqueCodeRadioButtons;

private Border violationBorder;

private uint addViolationSize;

private List<VertexNode> potentialVertexes;

private List<string> allSensors;

private GVSSMode mode;

public VertexDialog(List<string> allSensors, List<VertexNode> potentialVertexes, List<string> allAuxiliaryParams, List<string> allViolations, GVSSMode mode)

{

InitializeComponent();

LocalizeWindow();

this.mode = mode;

this.Height = defWindowHeight;

this.allSensors = allSensors;

this.potentialVertexes = potentialVertexes;

if (mode == GVSSMode.RepetativeViolations)

{

violationBorder = addRepetativeViolationBorder;

addViolationSize = addRepetativeViolationWindowHeight;

}

else if (mode == GVSSMode.UniqueViolations) {

violationBorder = addViolationBorder;

addViolationSize = addViolationWindowHeight;

}

violationRB.Tag = ChooseAction.Violation;

auxiliaryParamRB.Tag = ChooseAction.AuxiliaryParam;

actionRB = ChooseAction.None;

chooseActionRBs = new List<RadioButton>();

chooseActionRBs.Add(violationRB);

chooseActionRBs.Add(auxiliaryParamRB);

foreach (string auxiliary in allAuxiliaryParams) {

bool isAlreadyExist = false;

foreach (string violation in allViolations)

{

if (violation == auxiliary) {

isAlreadyExist = true;

break;

}

}

if (!isAlreadyExist) {

ComboBoxItem sensor = new ComboBoxItem();

sensor.Tag = auxiliary;

sensor.Content = auxiliary;

repetativeViolationsCB.Items.Add(sensor);

}

}

repetativeViolationsCB.SelectedIndex = 0;

setAddRepetativeViolationButton();

foreach (string sensorName in allAuxiliaryParams)

{

ComboBoxItem sensor = new ComboBoxItem();

sensor.Tag = sensorName;

sensor.Content = sensorName;

auxiliaryParamCB.Items.Add(sensor);

}

auxiliaryParamCB.SelectedIndex = 0;

if (auxiliaryParamCB.Items.Count == 0)

{

addAuxiliaryParamToVertex.IsEnabled = false;

loadPotentialVertexes("");

}

else {

loadPotentialVertexes((string)(((ComboBoxItem)(auxiliaryParamCB.SelectedItem)).Tag));

}

InputCorrector.InitializeRelatedToSensorCBs(controlMethodCB, controlAgentCB, requestBodyCB, requestPlacementMethodCB);

uniqueCodeRadioButtons = new List<RadioButton>();

uniqueCodeRadioButtons.Add(arrowUpRB);

uniqueCodeRadioButtons.Add(arrowDownRB);

uniqueCodeRadioButtons.Add(arrowLeftRB);

uniqueCodeRadioButtons.Add(arrowRightRB);

arrowUpRB.Tag = SensorArrowSign.Up;

arrowRightRB.Tag = SensorArrowSign.Right;

arrowLeftRB.Tag = SensorArrowSign.Left;

arrowDownRB.Tag = SensorArrowSign.Down;

checkedArrowRB = SensorArrowSign.None;

borderAroundUniqueCodePreview.Background = (SolidColorBrush)this.TryFindResource("disabledBlue");

}

private void chooseRB\_Click(object sender, RoutedEventArgs e)

{

RadioButton rb = (RadioButton)sender;

if ((ChooseAction)rb.Tag == actionRB) {

rb.IsChecked = false;

actionRB = ChooseAction.None;

violationBorder.Visibility = Visibility.Collapsed;

addAuxiliaryParamBorder.Visibility = Visibility.Collapsed;

this.Height = defWindowHeight;

}

else

{

actionRB = (ChooseAction)rb.Tag;

if (actionRB == ChooseAction.Violation)

{

this.Height = addViolationSize;

violationBorder.Visibility = Visibility.Visible;

addAuxiliaryParamBorder.Visibility = Visibility.Collapsed;

}

else if (actionRB == ChooseAction.AuxiliaryParam) {

this.Height = addAuxiliaryParamWindowHeight;

violationBorder.Visibility = Visibility.Collapsed;

addAuxiliaryParamBorder.Visibility = Visibility.Visible;

}

}

}

private void arrowRB\_Click(object sender, RoutedEventArgs e)

{

InputCorrector.arrowRB\_Click(sender, e, ref checkedArrowRB);

setViolationButton();

}

private void setViolationButton()

{

InputCorrector.setButton(addViolationButton,

indexTxtBox,

capLetTxtBox,

addViolationDescription,

allSensors,

uniqueCodeRadioButtons,

addViolationInfo,

uniqueViolationFailureTB);

}

private void addViolationButton\_Click(object sender, RoutedEventArgs e)

{

SensorArrowSign arrowSign = SensorArrowSign.None;

foreach (RadioButton rb in uniqueCodeRadioButtons)

{

if (rb.IsChecked == true)

{

arrowSign = (SensorArrowSign)rb.Tag;

break;

}

}

ControlMethodViolation = (SensorControlMethod)(((ComboBoxItem)(controlMethodCB.SelectedItem)).Tag);

ControlAgentViolation = (SensorControlAgent)(((ComboBoxItem)(controlAgentCB.SelectedItem)).Tag);

RequestBody = (SensorRequestBodies)(((ComboBoxItem)(requestBodyCB.SelectedItem)).Tag);

RequestPlacementMethod = (SensorRequestPlacementMethod)(((ComboBoxItem)(requestPlacementMethodCB.SelectedItem)).Tag);

CapLetViolation = capLetTxtBox.Text[0];

IndexViolation = UInt16.Parse(indexTxtBox.Text);

DescriptionViolation = addViolationDescription.Text;

ArrowViolation = arrowSign;

FailureRateViolation = float.Parse(uniqueViolationFailureTB.Text);

Result = VertexDialogResult.AddViolation;

this.DialogResult = true;

this.Close();

}

private void loadPotentialVertexes(string exceptUniqueCode) {

string selectedItemVertexID = "----";

if (potentialVertexesCB.Items.Count != 0) {

selectedItemVertexID = ((VertexNode)((ComboBoxItem)potentialVertexesCB.SelectedItem).Tag).VertexID.ToString();

}

potentialVertexesCB.Items.Clear();

int selectedIndex = 0;

foreach (VertexNode v in potentialVertexes) {

if (v.UniqueCode != exceptUniqueCode) {

ComboBoxItem vertex = new ComboBoxItem();

vertex.Tag = v;

vertex.Content = "Вершина №" + v.VertexID + " (" + v.UniqueCode + ")";

potentialVertexesCB.Items.Add(vertex);

if (v.VertexID.ToString() == selectedItemVertexID) {

selectedIndex = potentialVertexesCB.Items.Count - 1;

}

}

}

potentialVertexesCB.SelectedIndex = selectedIndex;

if (potentialVertexesCB.Items.Count == 0 || auxiliaryParamCB.Items.Count == 0)

{

addAuxiliaryParamToVertex.IsEnabled = false;

}

else if (potentialVertexesCB.Items.Count > 0 && auxiliaryParamCB.Items.Count > 0) {

addAuxiliaryParamToVertex.IsEnabled = true;

}

}

private void auxiliaryParamCB\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

loadPotentialVertexes((string)((ComboBoxItem)e.AddedItems[0]).Tag);

}

private void addAuxiliaryParamToVertex\_Click(object sender, RoutedEventArgs e)

{

AuxiliaryParam = (string)(((ComboBoxItem)auxiliaryParamCB.SelectedItem).Tag);

ConnectedVertexID = ((VertexNode)(((ComboBoxItem)potentialVertexesCB.SelectedItem).Tag)).VertexID;

Result = VertexDialogResult.AddAuxiliaryParam;

this.DialogResult = true;

this.Close();

}

private void addRepetativeViolationButton\_Click(object sender, RoutedEventArgs e)

{

ViolationUniqueCode = (string)((ComboBoxItem)(repetativeViolationsCB.SelectedItem)).Tag;

FailureRateViolation = float.Parse(failureRateTB.Text);

Result = VertexDialogResult.AddViolation;

this.DialogResult = true;

this.Close();

}

private void setAddRepetativeViolationButton() {

if (repetativeViolationsCB.Items.Count != 0 && InputCorrector.isFailureRateTextBoxValid(failureRateTB))

addRepetativeViolationButton.IsEnabled = true;

else

addRepetativeViolationButton.IsEnabled = false;

}

private void failureRateTB\_TextChanged(object sender, TextChangedEventArgs e)

{

string name = ((TextBox)(sender)).Name;

if (name == "failureRateTB")

{

setAddRepetativeViolationButton();

}

else if (name == "uniqueViolationFailureTB") {

setViolationButton();

}

}

private void uniqueViolationFailureTB\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

InputCorrector.FailureRateTB\_PreviewTextInput(sender, e);

setViolationButton();

}

private void textBoxWithoutSpaceKey\_PreviewKeyDown(object sender, KeyEventArgs e)

{

InputCorrector.textBoxWithoutSpaceKey\_PreviewKeyDown(sender, e);

}

private void addViolationDescription\_TextChanged(object sender, TextChangedEventArgs e)

{

setViolationButton();

}

private void nameParam\_TextChanged(object sender, TextChangedEventArgs e)

{

InputCorrector.changeSensorNameRepresentation(capLetTxtBox,

indexTxtBox,

namePreviewLabel,

uniqueCodeRadioButtons,

borderAroundUniqueCodePreview,

uniqueCodePreviewDock, ref checkedArrowRB);

setViolationButton();

}

private void capLetTxtBox\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

InputCorrector.CapitalLetterTB\_PreviewTextInput(sender, e);

}

private void indexTxtBox\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

InputCorrector.IndexTB\_PreviewTextInput(sender, e);

}

private void failureRateTB\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

InputCorrector.FailureRateTB\_PreviewTextInput(sender, e);

setAddRepetativeViolationButton();

}

private void controlMethodCB\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

InputCorrector.ControlMethodCB\_SelectionChanged(sender, e,

controlAgentTBlock,

requestBodyCB,

requestBodyBorder,

requestPlacementMethodCB,

requestPlacementMethodBorder);

}

private void requestBodyCB\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

InputCorrector.requestBodyCB\_SelectionChanged(sender, e, requestPlacementMethodCB);

}

private void LocalizeWindow()

{

this.Title = DefaultSettings.getWindowTitleText(WindowTitles.Add);

addAuxiliaryParamToVertex.Content = DefaultSettings.getButtonText(Buttons.Add);

addViolationButton.Content = DefaultSettings.getButtonText(Buttons.Add);

addRepetativeViolationButton.Content = DefaultSettings.getButtonText(Buttons.Add);

chooseActionLabel.Content = DefaultSettings.getLabelText(Labels.ChooseAction);

violationRB.Content = DefaultSettings.getWordValue(Words.AddViolation);

auxiliaryParamRB.Content = DefaultSettings.getWordValue(Words.AddAuxiliaryParamToVertex);

addViolationLabel.Content = DefaultSettings.getLabelText(Labels.AddViolation);

violationLabel.Text = DefaultSettings.getLabelText(Labels.Violation);

failureRateLabel.Text = DefaultSettings.getLabelText(Labels.FailureRate);

addUniqueViolationLabel.Content = DefaultSettings.getLabelText(Labels.AddViolation);

letterLabel.Text = DefaultSettings.getLabelText(Labels.Letter);

indexLabel.Text = DefaultSettings.getLabelText(Labels.Index);

requestBodyLabel.Text = DefaultSettings.getLabelText(Labels.RequestBody);

requestPlacementMethodLabel.Text = DefaultSettings.getLabelText(Labels.RequestPlacementMethod);

failureRateUniqueLabel.Text = DefaultSettings.getLabelText(Labels.FailureRate);

descriptionLabel.Text = DefaultSettings.getLabelText(Labels.Description);

addAuxiliaryParamToVertexLabel.Content = DefaultSettings.getLabelText(Labels.AddAuxiliaryParamToVertex);

auxiliaryParamLabel.Text = DefaultSettings.getLabelText(Labels.AuxiliaryParam);

controlMethodLabel.Text = DefaultSettings.getLabelText(Labels.ControlMethod);

}

}

}