НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Наталія АУШЕВА

«___»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра
спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: «Інструментальні засоби побудови логіко-імітаційної моделі розвитку небезпечних подій на енергетичних об'єктах»

Виконала: студентка 4 курсу, групи ТР-81

Маслова Єлизавета Валеріївна _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доц., к.е.н. Караєва Наталія Веніамінівна _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент доц., к.т.н. Сірий Олександр Анатолійович_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2022

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший

спеціальність 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія АУШЕВА
(підпис)

«___»_____2022р.

ЗАВДАННЯ

на дипломну роботу студенту

Масловій Єлизаветі Валеріївні _____
(прізвище, ім'я, по батькові)

1. Тема роботи ____Інструментальні засоби побудови логіко-імітаційної моделі розвитку небезпечних подій на енергетичних об'єктах _____
керівник роботи Караєва Наталія Веніамінівна, к.е.н., доц._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від «08» червня 2022р. № _965-с_____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи мови програмування C#, GDScript, рушій Godot, LiteBD

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Проаналізувати існуючі інструментальні засоби побудови логіко-імітаційних моделей, розробити архітектуру бази даних, розробити архітектуру програмного продукту, створити користувацький інтерфейс._____

5. Перелік ілюстративного матеріалу+
47 рисунків , 2 таблиці.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «30» вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	25.05.2022	
2.	Аналізування задачі	02.05.2022-03.05.2022	
3.	Розробка загальної структури системи та інтерфейсу	04.05.2022-05.05.2022	
4.	Розробка основних класів та синглтонів	06.05.2022-07.05.2022	
5.	Програмна реалізація системи	08.05.2022-15.05.2022	
6.	Оформлення пояснювальної записки	16.05.2022-22.05.2022	
7.	Захист програмного продукту	23.05.2022-27.05.2022	
8.	Передзахист	06.06.2022-09.06.2022	
9.	Захист	20.06.2022-30.06.2022	

Студент _____ Маслова Є.В.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Караєва Н.В.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою дипломної роботи є розробка системи для графічної побудови логіко-імітаційної моделі. Система надає можливість побудови діаграми дерева відмов вбудованими інструментами. Також система забезпечена компактною базою даних, яка дозволяє зберігати роботу, продовжувати її та загально аналізувати створені дерева відмов та дерева подій. Тож розроблена система може слугувати зручним інструментам аналізу ризиків на енергетичних об’єктах.

Записка складається з 4 розділів, 47 сторінок, 29 рисунків, 2 таблиці та 20 використаних джерел.

ABSTRACT

The purpose of the thesis is to develop a system for graphical construction of logic-simulation model. The system provides the ability to build a failure tree diagram and event tree diagram with built-in tools. The system is also equipped with a compact database that allows you to save work, continue it and generally analyze the created failure trees. Therefore, the developed software product can serve as a convenient tool for risk analysis at energy facilities.

The note consists of 4 sections, 47 pages, 29 figures, 2 tables and 20 sources used.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – База Даних

ПЗ, програмне забезпечення, програмні засоби – комплекс програм системи оброблення інформації та програмних документів, що забезпечують можливість експлуатації програм; сукупність вказівок, що створюють і реалізують алгоритми роботи пристрою.

Рушій – ядро програми призначене для реалізації конкретної прикладної задачі, що забезпечує її диференціацію від наповнення і зовнішніх ознак конкретної програми.

Інтерфейс – користувацький інтерфейс, сукупність методів для взаємодії користувача з системою.

Godot – відкритий багатоплатформовий 2D та 3D рушій розроблений торством Godot Engine Community та заліцензований за допомогою ліцензії MIT.

Об’єктно-орієнтоване програмування – парадигма програмування, що визначає програму як сукупність «об’єктів», які перебувають у взаємодії. Базується на чотирьох концепціях-складниках: інкапсуляції, поліморфізмові, успадкуванні та абстракції.

Ноди – вузлові точки, у яких відбувається генерація, поширення або рецепція даних.

SQL, Structured Query Language – декларативна мова програмування, призначена для генерації, модифікації та керування інформацією в реляційній базі даних.

NoSQL, Structured Query Language – термін на позначення сукупності підходів, відмінних від традиційних релятивістських СУБД.

СУБД – Система Управління Базами Даних

JSON, JavaScript Object Notation – формат текстового обміну даними, зручний для читання, використовується для представленя структур даних (масив, словник тощо) у текстовому вигляді.

BSON, Binary JSON – формат на основі JSON.

FTA, fault tree analysis, дерево відмов – діаграма аналізу ризику типу «дерево відмов».

ETA, Event Tree Analysis, дерево подій – діаграма аналізу ризику типу «дерево подій».

ВСТУП

Україна – одна з країн-лідерів Європи за об’ємами виробленої електроенергії. Але попри вражаючі обсяги виробництва, Україна не має свого зручного та дружнього до користувача програмного забезпечення для ризик-менеджменту на енергетичних об’єктах. Наявне програмне забезпечення є застарілим або ж високовартісним, що значно зменшує його доступність для кінцевого користувача. Наприклад, на одну з найбільш популярних програм у цій сфері – OpenFTA – масово надходять скарги на її не дружність до користувача, починаючи від проблем з встановленням до користувацького інтерфейсу. Тому задача розробки інструментальних засобів для ризик-менеджменту є актуальною.

При цьому всі енергетичні об’єкти можна назвати потенційно небезпечними, а непередбачувані аварії несуть значну шкоду. Наймасштабнішим прикладом є енергетична аварія у США в 2003, коли значна частина країни (50 млн осіб) лишились без електроенергії через цілу ланку подій. Наростання чинників ризику невпинно вело до аварії. Проте помилкові симуляції комп’ютери не показали цього.

У таких критичних випадках потрібно швидко приймати рішення в умовах недостатньої кількості інформації. Тому необхідна система, що спростить обробку та надання інформації особам, що розробляють сценарії мінімізації небезпечних подій на енергетичних об’єктах.

Для зниження ризиків аварій та техногенних катастроф на енергетичних об’єктах доцільно використовувати спеціалізовані системи прогнозу і мінімізації ризику. Основою таких систем є імовірнісний аналіз безпеки, який чудово себе зарекомендував у аналізі транспортних процесів [1]. Сам термін «ризик» походить з грецької мови і означає «скеля», тобто, ризикувати – це оминати скелю [2].

Оцінка небезпеки різних енергетичних об'єктів це визначення виникнення можливих надзвичайних ситуацій, впливу людського фактору, збоїв виробництва електроенергії через порушення логістики сировини, зношення обладнання,

руйнівних впливів пожеж і витоків забруднюючих речовин на об'єкти, а також впливу небезпечних факторів пожеж і забруднюючих речовин на людей. Оцінка цих небезпечних впливів на стадії проектування об'єктів дозволяє виконувати об’єкти більш стійкими до ризиків, тобто з урахуванням можливої аварійної ситуації.

Задача продукту що розробляється полягає у дружньої до користувача побудові методів аналізування сценаріїв, їх аналізу, редагування та оцінювання експертами.

У першому розділі описана мета дипломної роботи, призначення розробленого програмного забезпечення та задачі, що ним вирішуються.

В другому розділі проводиться огляд сучасних методів побудови логіко-імітаційної моделі розвитку небезпечних подій і відповідних програмних рішень.

Третій розділ містить опис розробки системи побудови логіко-імітаційної моделі розвитку небезпечних подій на енергетичних об’єктах та обґрунтування обраних засобів розробки системи.

У четвертому розділі описана програмна реалізація задачі, архітектура розробленої програмної системи, опис бази даних.

У четвертому розділі описуються системні вимоги та робота користувача в

У висновках зазначаються результати розробленої системи.

1 ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ СИСТЕМИ ПОБУДОВИ ЛОГІКО-ІМІТАЦІЙНОЇ МОДЕЛІ РОЗВИТКУ НЕБЕЗПЕЧНИХ ПОДІЙ НА ЕНЕРГЕТИЧНИХ ОБ'ЄКТАХ

Метою дослідження є розробка системи, що дозволить будувати логіко-імітаційну модель розвитку небезпечних подій у відповідності вимог ДСТУ IEC/ISO 31010:2013 «Керуванням ризиком. Методи загального оцінювання ризиків» [3]. Цей додаток має графічно відображати дерево відмов, надаючи дані у зручному для користувача вигляді, придатному для використання у документації.

Оскільки процес і результати загальної оцінки ризику повинні бути детально задокументовані, використовуючи для позначення ризиків зрозумілі позначення. Обсяг звіту залежить від обсягу та цілей загальної оцінки ризиків. Інші особливості необхідно задокументувати, за винятком випадку спрощеної оцінки ризику. Детальніше про елементи оцінювання ризику можна дізнатися на рисунку 1.1. Рекомендується оновлювати документ відповідно до вимог системи управління протягом усього терміну експлуатації системи. Тому коли виникає нова важлива інформація або інші зміни в критичній ситуації, документи слід оновлювати для подальшого полегшення загальної оцінки, а для цього вкрай необхідно мати зручне програмне забезпечення з можливістю редагування діаграм.

Також система має задовольняти ті потреби, що не можуть бути закриті поточними пропозиціями на ринку: доступність, наявність державної (української) мови для забезпечення можливості використання системи менш кваліфікованими співробітниками, зручність, орієнтованість на Україну та безпосередньо енергетичні об'єкти.

Рисунок 1.1 – Елементи загального оцінювання ризиків, що слід документувати

Для ефективної розробки системи були поставлені наступні завдання:

– дослідити поточні методи загального оцінювання ризику на енергетичних об’єктах;

– визначити методи, що найкраще підходять для моделювання розвитку небезпечних подій на енергетичних об’єктах;

– дослідити характеристики та функціонал існуючих систем побудови діаграм аналізування сценарію;

– розробити систему на основі поставленої задачі для побудови інструментальних засобів побудови логіко-імітаційної моделі розвитку небезпечних подій на енергетичних об’єктах.

Вхідні дані: ДСТУ IEC/ISO 31010:2013, дані про можливі події

Вихідні дані: інтерактивні карти та побудовані діаграми, що подалі можуть використовуватися при аналізі ризик-чинників небезпечних подій і розробці системи заходів щодо запобігання або пом’якшення небажаних наслідків

2 ОГЛЯД ІСНУЮЧИХ МЕТОДІВ ПОБУДОВИ ЛОГІКО-ІМІТАЦІЙНОЇ МОДЕЛІ РОЗВИТКУ НЕБЕЗПЕЧНИХ ПОДІЙ І ВІДПОВІДНИХ ПРОГРАМНИХ РІШЕНЬ

Сучасна енергетична індустрія вимагає актуального програмного забезпечення для запобігання техногенних катастроф. Основою для оцінки небезпеки є імовірнісний аналіз безпеки, який набув найбільшого поширення у сфері атомної енергетики та на виробничих об’єктах, де можливість викидів та вибухів небезпечних речовин найбільша [4]. Виробничі небезпеки завжди мають складну будову, і тому їх можна віднести до складних техніко-ергатичних систем, в яких завжди є потенційна небезпека, що полягає в присутності людини [5, 6]. У даному розділі будуть розглянуті існуючі методи оцінки ризиків для обґрунтування вибору методу, який було обрано для реалізації системи.

2.1 Методи і моделі оцінювання ризиків енергетичних об’єктів

Сучасні дослідники констатують наявність багатьох різноманітних підходів до унормування ризику в галузі забезпечення екологічної та промислової безпеки. Серед них виділяють такі три провідні типи:

1. детермінований;
2. імовірнісний;
3. комбінований;

Детермінований підхід спирається на певну кількісну диференціацію та розподіл надзвичайних ситуацій, їхньої кількості за ступенем небезпеки на категорії, класи і тощо, які визначаються за параметром, що характеризує параметри небезпечних процесів, кількість уражених та постраждалих, а також руйнівні наслідки надзвичайних ситуацій. При цьому призначаються конкретні кількісні межі цих категорій, класів і т. п.

Загальне представлення різних методів оцінювання ризиків надано на рисунку 2.1.

Рисунок 2.1 – Загальне представлення методів оцінювання ризиків

Імовірнісний підхід є більш прогресивним і придатним для реалізації, оскільки він дає можливість знаходження оптимального варіанта проектного рішення. Його основою є кількісна залежність між небезпечними факторами середовища, матеріальною шкодою та ймовірністю реалізації небезпечних факторів з урахуванням захисних заходів. За допомогою імовірнісних методів можна знаходити оптимальні технічні рішення для конкретних об'єктів.

Імовірнісно-статистичні методи аналізу ризику припускають як оцінку імовірності виникнення небезпеки, так і розрахунок відносної ймовірності того або іншого шляху розвитку процесу. При цьому аналізуються розгалужені ланцюжки умов і факторів, вибирається відповідний математичний апарат і

оцінюється повна імовірність ризикованої ситуації. Розрахункові математичні моделі при цьому можна істотно спростити в порівнянні з детермінованими методами. Основні обмеження методу пов’язані з недостатньою статистикою по різних факторах.

Імовірнісний підхід заснований на концепції допустимого ризику з розрахунком імовірності досягнення певного рівня безпеки і передбачає недопущення впливу на людей небезпечних факторів дорожнього середовища з ймовірністю, що перевищує нормативну.

Переваги та недоліки цих методів було проаналізовано і представлено у таблиці 2.1.

Таблиця 2.1. Переваги і недоліки методів аналізування сценарію

Метод	Переваги	Недоліки
Аналіз дерева подій (ETA)	– є помічним при встановленні обліку причинно-наслідкового зв'язку між відмовами елементів та картини фізичних процесів, які можуть стати причиною критичного стану системи; – метод надає можливість схематичного генерування сценарію розвитку подій після виникнення початкової події, проведення аналізу станів та подій, які мають влив на зменшення наслідків відмови, а також оцінки їхнього впливу на систему; – результатом застосування методу стає структуроване встановлення послідовності подій, яку неможливо дослідити за використання методу «дерево відмов».	– потреба зауваження усіх можливих початкових подій, наявність імовірного випадкового ігнорування деяких важливих подій; – обумовлення будь-якої гілки дерева подіями, які вже сталися в попередніх точках шляху, надає можливість розгляду усіх кореляцій уздовж існуючих шляхів. Однак при цьому деякі з них можуть бути випадково проігноровані чи невраховані, і це призведе до встановлення більш оптимістичної оцінки ризику; – метод надається до застосування лише для двох станів системи (безвідмовного та відновного), і як наслідок виникає проблема врахування справного стану системи із

		запізненням або її відновлення.

Таблиця 2.1 (продовження)

Аналіз дерева відмов (FTA)	– межі аналізу призводять до виявлення тільки тих елементів системи та подій, що є причиною даної конкретної відмови системи або аварії; – структурований, адаптивний підхід, що робить можливим аналіз різноманітних чинників, таких як міжлюдська взаємодія та фізичні явища; – доцільність для аналізу систем реалізації підходу «зверху вниз» та можливість концентрації на наслідках відмови, пов’язаних із завершальною подією; – графічна репрезентація сприяє повнішому розумінню функціонування системи розглянутих факторів;	– дерево відмов є схемою булевої логіки, що здатна відображати лише два стани, а саме робочий та стан відмови; – проблематичність врахування стану часткової відмови спричинена розглядом системи за застосування методу як такою, що перебуває або в справному стані, або в стані відмови; – необхідність високого рівня розуміння фахівцем системи в цілому і ситуативного щоразового розгляду лише однієї певної відмови; – дерево відмов репрезентує систему в певний момент часу, що призводить до труднощів із відображенням ланцюжка подіяй, а часом і до неможливості його передачі.

Метод «дерева відмов» набув широкого застосування у світі для аналізу ризику аварій на об'єктах підвищеної небезпеки. Цей метод застосовують як для попереднього аналізу рівня безпеки під час розроблення рекомендацій для зниження рівня ризику, так і для визначення причин аварій на небезпечних технологічних об'єктах.

2.2 Алгоритм побудови логічної схеми розвитку небезпеки у вигляді дерева відмов

Аналіз «дерева відмов» (Fault-tree analysis – FTA) було розроблено в США [4]. FTA – це метод дедуктивного мислення, де логічний зв'язок між потенційними аваріями і відповідними причинами можна репрезентувати за допомогою деревоподібних діаграм. Приклад такої діаграми можна побачити на рисунку 2.3. FTA після шляху свого становлення і доопрацювань став одним із найпопулярніших методів у системному аналізі. Він може описати динамічний процес виникнення і розвитку аварій, з'ясувати прямі й непрямі причини і логічно поєднати їх. Аналізуючи FTA якісно і кількісно, можна виокремити основні причини, створити основу для алгоритмів заходів безпеки, для прогнозування і запобігання аварій. «Дерево відмов» є спрямованим логічним деревом, яке описує виникнення аварій від результату до причини. Технологія аналізу дерева відноситься до категорії теорії графів інженерії систем.

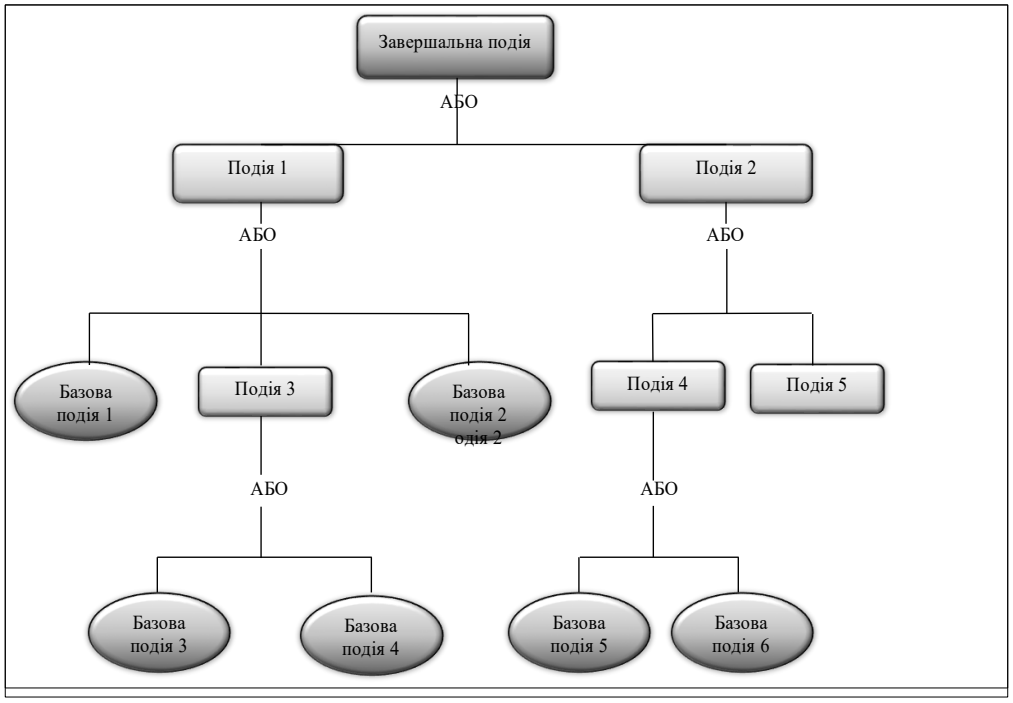

Рисунок 2.2 – Приклад діаграми дерева відмов

Для того, аби створити дерево відмов, необхідно розпочати з кінцевої події, яка стоятиме на верхівці дерева-діаграми. Усі наступні події будуть гілками цього деревами, або ж піддеревами. Така побудова дозволяє визначити, де саме відбулися пошкодження у енергетичній системі та визначити зв’язок між пошкодженнями та подією відмови й яка взаємодія відбулася чи може відбутися.

Для того, аби визначити, які вузли мають заповнювати дерево надалі, потрібно задати собі питання: «Що може призвести до виникнення даної події?». Від відповіді залежить те, які саме блоки будуть використані далі.

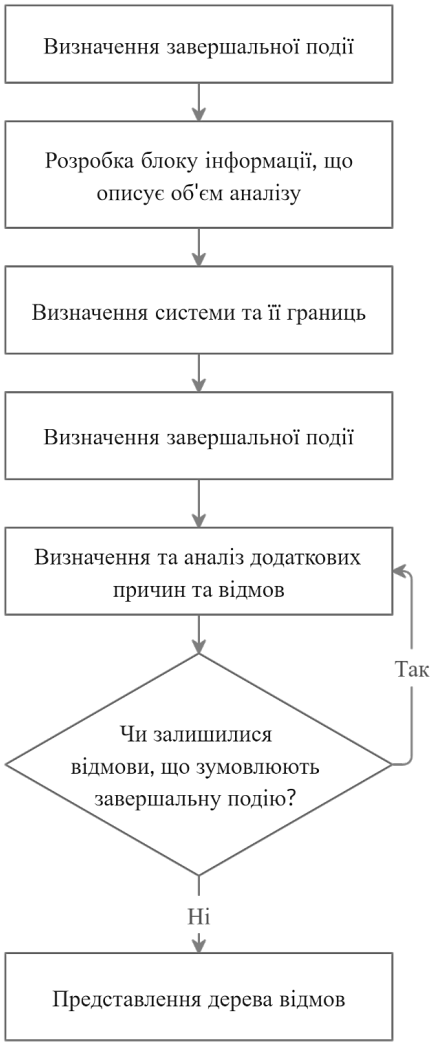

Рисунок 2.3 – Основні етапи аналізування методу «дерево відмов»

Побудова має відбуватися за використання стандартизованого графічного представлення подій, для можливості подальшого аналізу. Детальніше про те, як ілюструються ті чи інші вузли дерева відмов можна дізнатися з опису інтерфейсу розробленої системи. Вони поділяються на два типи блоків: логічні символи і символи подій.

2.3 Алгоритм побудови логічної схеми розвитку небезпеки у вигляді дерева подій

Аналіз дерева подій – це метод вивчення ланцюжка подій, починаючи з першої з них, які можуть призвести або не призвести до небажаних результатів. Таким чином, метод особливо корисний при аналізі існуючої або запланованої захисної структури в енергетичній інфраструктурі (запобігання, захист, контрзаходи). Вхідними для методу можуть бути перелік початкових подій, інформація про їх функціонування та взаємодію, а також ймовірність їх відмови.
Аналіз дерева подій має виконуватися командою експертів, які добре обізнані з проблемою, яку потрібно вирішити. Приклад схеми діаграми дерева подій зображено на рисунку 2.2. Як показано на рисунку 2.3, алгоритм побудови «дерева подій» складається з п’яти основних етапів:

1. Визначте ранні події, які можуть спричинити нещасні випадки або несприятливі наслідки.
2. Розробити початковий набір інформації, що описує обсяг аналізу. На цьому етапі додатково аналізуються обсяг і дані системи, дані проекту, процедури обліку та технічні дані.
3. Визначити проміжні події.
4. Обчислити ймовірність кожного
5. Реалізовано остаточне представлення діаграми «дерева подій».

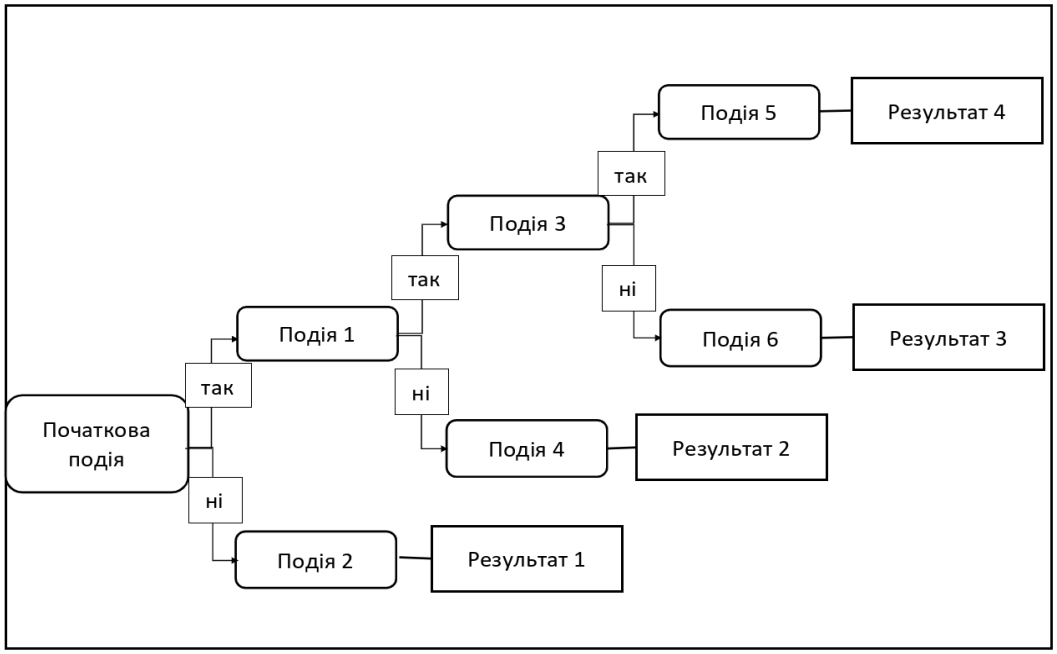

Рисунок 2.4 – Приклад дерева подій

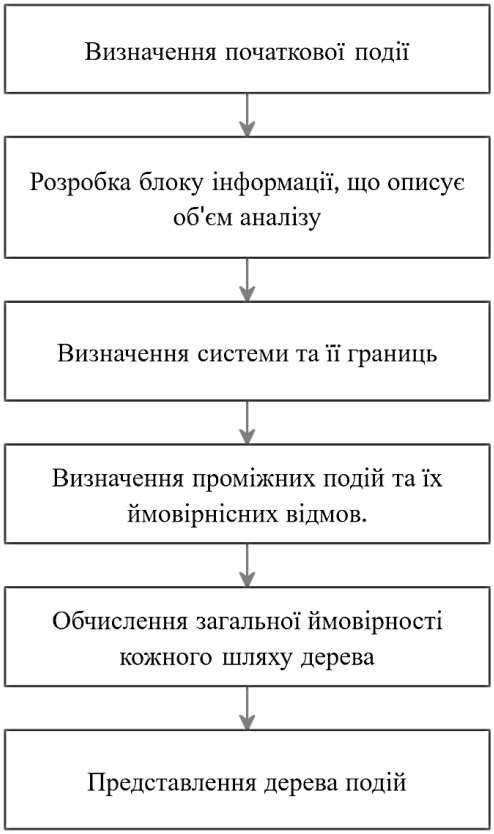

Рисунок 2.5 – Основні етапи аналізування методу «дерево подій»

На кожному рівні дерева створюється нова подія з урахуванням двох станів: так (успіх) і ні (невдача), залежно від результату конкретної події. Якщо подія вказує на успіх, логічному блоку присвоюється ймовірність Pi, а сусідньому блоку присвоюється значення 1-Pi. Кінцевий результат можна визначити, виходячи з добутку подій, що проходять по цьому шляху на всіх рівнях дерева, використовуючи наступну формулу:

$$P(A)=P(A¿¿1)\times P(A¿¿2)\times\cdots\times P(A¿¿n)¿¿¿. \quad (2.1)$$

де P – ймовірність;

A – події.

Аналіз дерева подій – це спосіб вивчити послідовність подій, які можуть призвести до неприємних наслідків, починаючи з першої дії. Тому такий підхід особливо корисний при аналізі існуючих енергетичних споруд або плануванні енергетичних об’єктів.

2.4 Функціональні можливості сучасних програмних засобів в задачах побудови логіко-імітаційних діаграм для оцінювання ризиків

Українського продукту, який давав би можливість проводити оцінювання ризиків на енергетичних об’єктах наразі немає. Тому було проаналізовано ринок іноземних програмних продуктів. Основними програмними засобами (ПЗ) на іноземному ринку побудови логіко-імітаційних діаграм є програми, зображені у таблиці 2.2.

Таблиця 2.2. Переваги і недоліки різного ПЗ

ПЗ	Ціна	Платформа	Переваги	Недоліки
OpenFTA	free	Windows	1. Підтримка повного набору символів дерева несправностей 2. Якісний та кількісний аналіз дерева відмов 3. База даних, у якій зберігається інформація про визначені події 4. Можливість розділяти базу подій між іншими користувачами та деревами.	1. Програмне забезпечення давно не оновлювалося 2. Не працює офіційний сайт 3. Проблеми сумісності з сучасним обладнанням 4. Відсутня українська мова.
ALD Fault Tree Analyzer	free	Web	1. Формування звітів 2. Групове редагування елементів діаграми 3. Якісний та кількісний аналіз дерева відмов 4. Підтримка реєстру ризиків 5. Пошук інформації у файлах та звітах 6. Можливість власного налаштування панелей інструментів 7. Експорт діаграм у PDF 8. Підтримка різних іноземних мов.	1. Відсутність можливості роботи без з’єднання з мережею Інтернет 2. Необхідна реєстрація на сайті розробника для роботи 3. Обмежена кількість інструментів 4. Відсутність української мови.

Таблиця 2.2 (продовження)

Visual Paradigm Diagrams	free	Web	1. Візуалізація задачі по управління ризику 2. Створення звітів ризику у реальному часі 3. Експорт діаграм у PDF, SVG, JPEG 4. Підтримка різних іноземних мов підтримка 5. Збереження історії редагування елементів користувачем 6. Вбудовані шаблони.	1. Відсутність можливості роботи без з'єднання з мережею Інтернет 2. Необхідна реєстрація на сайті розробника для роботи 3. Обмежена кількість інструментів 4. Обмежений функціонал аналізу ризиків через іншу спеціалізацію ПЗ. 5. Відсутність української мови.
Creately	$4-8 на місяць	Web, Windows, Mac та Linux.	1. Прив'язка інструментів для ідентифікації ризиків до аналізу методу «дерево відмов» 2. Ведення історії аналізу ризику 3. Можливість створення аудиторських перевірок на основі результатів аналізу даних 4. Редагування звітів про ризики у реальному часі 5. Збереження та перегляд звітів, можливість їх експорту в PDF.	1. Ціна 2. Необхідна реєстрація на сайті розробника для роботи 3. Обмежена кількість інструментів 4. Обмежений функціонал аналізу ризиків через іншу спеціалізацію ПЗ.

Таблиця 2.2 (продовження)

Smartdraw Fault Tree Software	$5-10 на місяць	Web	1. Гнучкі налаштування діаграми 2. Вбудовані шаблони 3. Експорт діаграм у PDF, SVG, JPEG, PNG, VSDX та у Microsoft Office 4. Вбудована функція автоматичного перекладу різними мовами, включно з українською.	1. Ціна 2. Відсутність можливості роботи без з'єднання з мережею Інтернет 3. Необхідна реєстрація на сайті розробника для роботи 4. Обмежена кількість інструментів 5. Обмежений функціонал аналізу ризиків через іншу спеціалізацію ПЗ.
Wondershare EdrawMax	$99-728 на рік	Web, Windows	1. Формування звітів 2. Вбудовані шаблони 3. Групове редагування елементів діаграми 4. Підтримка реєстру ризиків 5. Пошук інформації у файлах та звітах 6. Можливість власного налаштування панелей інструментів 7. Експорт діаграм у PDF 8. Підтримка різних іноземних мов.	1. Ціна 2. Необхідна реєстрація на сайті розробника для роботи 3. Обмежена кількість інструментів 4. Обмежений функціонал аналізу ризиків через іншу спеціалізацію ПЗ. 4. Відсутність української мови.

Таблиця 2.2 (продовження)

Draw.io	free	Web	1. Об’єднання декількох діаграм 2. Вбудовані шаблони 3. Потужний пошук по елементах діаграм 4. Одночасна робота декількох користувачів над одним документом 5. Аудіо коментарі 6. Експорт у Microsoft Word, PowerPoint, PDF 7. Можливість власного налаштування панелей інструментів.	1. Відсутність можливості роботи без з’єднання з мережею Інтернет. 3. Обмежена кількість інструментів 4. Обмежений функціонал аналізу ризиків через іншу спеціалізацію ПЗ. 5. Відсутність української мови.
Isograph Reliability Workbench	ціна не вказана	Windows	1. Підтримка повного набору символів дерева несправностей 2. Якісний та кількісний аналіз дерева відмов 3. База даних, у якій зберігається інформація про визначені події 4. Можливість створення аудиторських перевірок на основі результатів аналізу даних.	1. Закритість даних про програмне забезпечення 2. Доступність ПЗ лише для корпоративних працівників 3. Відсутність української мови.

Детальніше розглянемо деякі з цих програмних продуктів.

OpenFTA

Ймовірно, саме цей додаток використовується в Україні найчастіше. Саме на основі інтерфейсу цієї програми проводиться навчання, а на прикладі цієї програми вивчається алгоритм побудови логіко-імітаційних діаграм. Проте це програмне забезпечення є застарілим та нестабільним. Розглянемо OpenFTA детально.

OpenFTA являє собою складний інженерний інструмент для побудови, аналізу та друку дерев відмов. Завдання аналітики полегшується багатьма інструментами і особливостями, які включають:

- point-and-click графічний користувацький інтерфейс метою якого є надання користувачеві можливості оперативно будувати дерева відмов;
- підтримка повного набору символів дерева несправностей відповідно до NUREG-0492;
- база даних, у якій зберігається інформація про визначені події;
- якісний та кількісний аналіз дерева відмов;

Основні важливі особливості OpenFTA:

- розширений користувацький інтерфейс;
- забезпечення оперативного аналізу основ;
- можливість розділяти базу подій між іншими користувачами та деревами;
- відсутність обмеження щодо кількості імовірних подій, якісний аналіз їх як відгалужень одного дерева.

Власне після здійснення якісного аналізу та визначення перетинів може відбутися і кількісний аналіз, Алгоритм визначає ймовірність відмови системи, а також важливість відмови кожної коротшої події.

Дерева відмов можуть бути проаналізовані за допомогою методу Монте-Карло задля статистичного визначення впливу коротших перетинів та їхнього безпосереднього впливу на несправність системи. Оцінка імовірності збою за

допомогою методу Монте-Карло є найбільш точним у співвіднесенні з результатам, отриманим детермінованими алгоритмами.

Робоча зона OpenFTA реалізована у вигляді рухомого полотна, призначеного для створення дерева.

Draw.io

Відомий безкоштовний редактор діаграм, який також дозволяє будувати й дерево відмов. Основна перевага – можливість одночасного редагування діаграми кількома користувачами. Крім того, додаток не вимагає встановлення.

Це ПЗ відоме своєю можливістю інтеграції в інші відомі проєкти, такі як OneDrive, Google Drive, Notion тощо, а також можливістю одночасної роботи над одною діаграмою [15]. Проте через перевантаженість різними інструментами Draw.io справляє на користувачів враження складного і незрозумілого продукту, що однозначно є недоліком для тих користувачів, які ніколи раніше не займалися побудовою логіко-імітаційних діаграм та діаграм загалом у цьому програмному забезпеченні

ALD Fault Tree Analyzer

Безкоштовне програмне забезпечення, яке спеціалізується на побудові саме діаграм дерева відмов. ALD працює в галузі надійності, безпеки та якості критично важливого обладнання та обладнання (головним чином аерокосмічного, оборонного, транспортного та телекомунікаційного) більше 25 років. Тому вони підтримують просування надійності та безпеки системи та бажають зробити її більш доступною для практиків і студентів з інженерії з надійності та безпеки [4]. Дана програма доступна як через браузер, так і через завантажуваний файл.

Wondershare EdrawMax

Програмне забезпечення, яке розповсюджується з платною підпискою. Гнучкий, професійний інструмент для побудови діаграм. Окрім інших, також підтримує побудову дерева відмов [9].

3 РОЗРОБКА СИСТЕМИ ПОБУДОВИ ЛОГІКО-ІМІТАЦІЙНОЇ МОДЕЛІ РОЗВИТКУ НЕБЕЗПЕЧНИХ ПОДІЙ НА ЕНЕРГЕТИЧНИХ ОБ'ЄКТАХ

Засоби для розробки системи обиралися з оглядом на важливість ключових факторів розробленої системи: кросплатформеність, зручність для користувача, невеликий об'єм програмного продукту, можливість локалізації, ліцензії, що дозволять використання як з дослідницькою, так і з комерційною ціллю тощо.

3.1 Вибір засобів реалізації системи

Обрані мною засоби для розробки використовують ліцензію MIT. Це найпопулярніша на даний час ліцензія, що дозволяє вільно використовувати дане програмне забезпечення як для власного користування, так і з комерційною метою, за умови дотримання умов ліцензії та непривласнення програмного забезпечення, що якнайкраще підходить для використання у проєкті, який може надалі використовуватись для аналізу на енергетичних об'єктах. Ілюстрацію всіх обраних засобів можна побачити на рисунку 3.1.

Рисунок 3.1 – Обрані засоби реалізації системи

Рушій Godot 3.4.4 Mono version

Невеликий рушій загального призначення з відкритим кодом [10]. Він використовує принцип об’єктно-орієнтованого програмування. Mono version – це версія, яка дозволяє використовувати мову C# як основну або допоміжну, оскільки рушій має власні засоби програмування, такі як GDScript та VisualScript, а також наявна підтримка NativeScript (підтримка компільованих скриптів на C, C++, Python, Rust тощо). C# це сучасна й зручна мова програмування [8]. Крім того, верстка додатку може відбуватися у візуальному редакторі сцен. Такий широкий вибір інструментів дозволяє сконцентруватися на розробці безпосередньо додатку без огляду на обмеження кожного з цих інструментів, оскільки є можливість використовувати кожен з цих інструментів паралельно.

Структурно проєкт на рушії Godot представляє собою дерево нодів. Ноди – фундаментальний блок для будівництва, які представляють собою сцени. Сценою є й безпосередньо зверстана сцена, так й окремі базові елементи, які є об’єктами вбудованих та створених розробників класів.

Наприклад, основна сцена розробленого програмного забезпечення – редактора, зображена на рисунку 3.2.

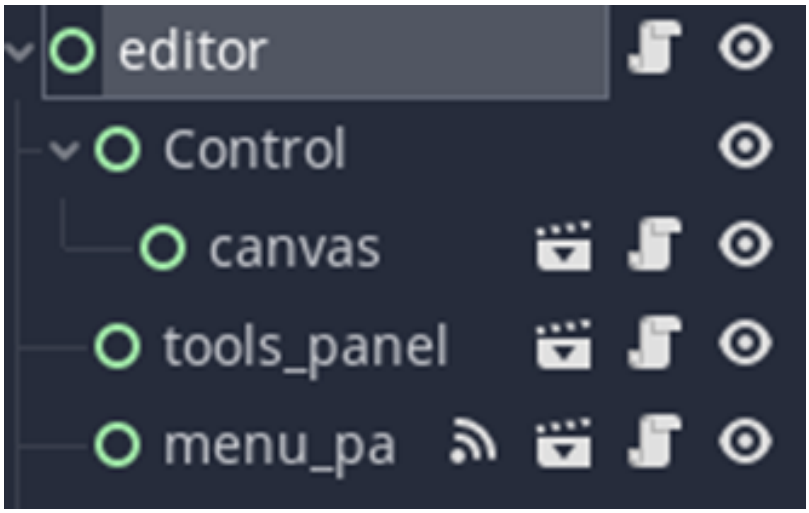

Рисунок 3.2 – Приклад нодів

Кожен з успадкованих нодів є окремою сценою, що має свої нащадки. Така структура забезпечує не тільки зручність розробки, а й можливість легкого подальшого доопрацювання програмного забезпечення. Це неймовірно важливо для підтримки додатку, додавання нових функцій, виправлення багів та інших недоліків, обробки фідбеку користувачів. Також це дає можливість безпосередньо на стороні користувачів розробити свої модулі до додатку, за умови наявності відкритого коду розробленого проєкту.

Не менш важливою функцією рушія Ґодо є вбудована функція легкої локалізації розроблюваних проєктів, показана на рисунку 3.2, що вигідно виділяє його на фоні деяких інших рушіїв. Меню знаходиться в редакторі проєкту й виглядає наступним чином:

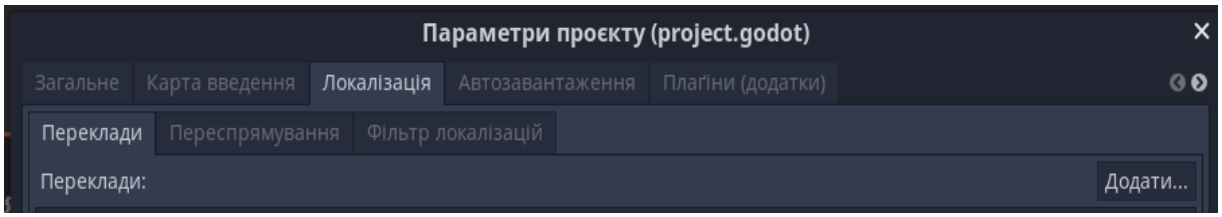

Рисунок 3.3 – Параметри локалізації

Для локалізації використовується сучасний формат gettex. Це формат, який використовується у всьому світі для перекладів різноманітного програмного забезпечення.

Використання останньої стабільної версії рушія забезпечить максимально безперебійну роботу з кінцевим програмним забезпеченням [17].

База даних LiteDB 5.0.11

Для збереження даних було використано NoSQL рішення на основі BJON документів, яка керується з допомогою мови програмування C# та SQL-like запитів. Такий вибір обумовлено кількома перевагами LiteBD [19].

По-перше, структура цієї БД. Вона безсерверна, бібліотека, що керує БД, є невеликою бібліотекою (DLL), повністю написана на dotNet та керується за допомогою мови C#. Завдяки цьому в певному роді єдиною необхідною для користувачів СУБД виступає сам розроблюваний додаток (хоча існує LiteBD Studio, яку користувач має можливість завантажити окремо й побачити БД, яку було створено), тобто для користувача немає необхідності встановлювати сервер чи інше програмне забезпечення для того, аби використовувати розроблену систему. Це є безумовно важливим фактором для забезпечення зручності розробленої системи для користувача.

По-друге, збереження даних у форматі BSON. Це менш строгий формат, ніж те, як зберігаються дані у SQL-БД. Попри наявність певних недоліків, таких як менша безпека цілісності даних, це також має значну кількість переваг. Наприклад, можливість експортувати одну окрему логіко-імітаційну модель у виділений файл за стандартом JSON, перенести, передати чи показати без необхідності передавати усю базу даних, що на мій погляд є важливим фактором як для зручності використання програмного забезпечення, так і для безпеки.

По-третє, гнучкість такого рішення надає можливість у майбутньому допрацьовувати та покращувати розроблений додаток без потреби кардинально змінювати базу даних. Головне, що має бути – сумісність оновлених даних, забезпечена у розробленій системі.

3.2 Опис програмної реалізації системи

Система має наступні складові: зверстана оболонка програми, синґлтони для керування логікою програми, таке як будування діаграми та зв’язок з БД, а також безпосередньо саму БД. Архітектуру системи можна побачити на рисунку 3.4.

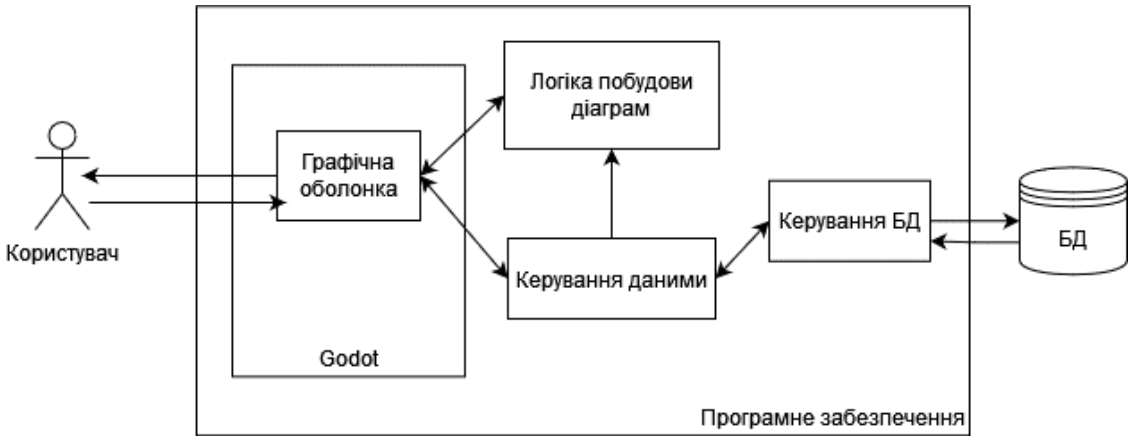

Рисунок 3.4 – Архітектура системи

Графічна оболонка програми зверстана з програмними засобами рушія Godot. Вона містить три основні елементи:

1. панель керування;
2. панель інструментів;
3. полотно;

Їхня взаємодія забезпечується скриптами сцен, класами та синґлтонами. Синґлтон – це відокремлений ресурс, який працює незалежно від того, яка сцена завантажена в графічній оболонці.

Для розробки проєкту було створені такі синґлтони:

– ToolsControl.gd

– DiagramData.gd.

Перший скрипт, ToolsControl, забезпечує коректну роботу інструментів та їхній зв’язок з тим, що користувач бачить на полотні. Саме цей скрипт відповідає за те, щоб діаграма на полотні будувалася саме тими блоками, які обрав користувач.

Другий скрипт, DiagramData, відповідає за тимчасове зберігання в пам’яті незбереженої роботи, а також задає умови формування цих даних.

Також наявні наступні користувацькі класи:

– DataBase.cs

– BlockClass.gd.

Перший клас, DataBase, забезпечує безпосередньо зв’язок програмного забезпечення з базою даних, отримує дані та з необхідними SQL-like запитами передає ці дані у БД. Також цей клас дозволяє дістати з БД необхідні дані для того аби продовжити роботу у будь-який момент.

Другий клас, BlockClass, відповідає за функціонування та заповнення блоку, який є візуальним відображенням даних з діаграми. Екземплярами цього класу можуть бути і події, і логічні елементи, оскільки функціонують на полотні вони подібно.

БД складається з трьох колекцій, які можна детально розглянути у концептуальній схемі на рисунку 3.5.

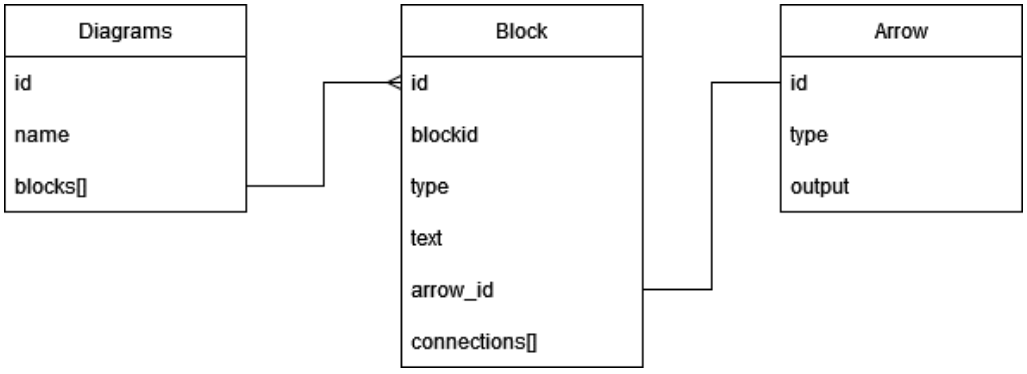

Рисунок 3.5 – Концептуальна схема БД

Сама ж база даних утворюється завдяки вбудованому у проєкт DLL-файлу, тож зв’язку з сервером наразі не передбачено, оскільки сервер відсутній. Зберігаються дані у вигляді BSON-файлів.

Завдяки правильно обраним інструментам розробки, реалізація системи вийшла лаконічною та придатною до подальших вдосконалень, якщо цього потребуватимуть користувачі.

4 МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

У цьому розділі буде розглянуто усі необхідні системні вимоги, яким має відповідати комп'ютер користувача для коректної роботи розробленої системи. Крім того, буде описаний детальний поетапний процес її використання з поясненням кожної функції.

4.1 Системні вимоги та інсталяція програмного продукту

Для коректної роботи системи наразі мають бути забезпечені наступні рекомендовані системні вимоги:

- Мінімум 100 мб вільного місця на диску;
- Операційна система Windows (7, 8, 10, 11);
- Графічний чип з підтримкою OpenGL 3.3;
- Встановлений dotNet 4.5 для максимально коректної роботи БД (не обов'язковий пункт);
- Оперативна пам'ять не менше 2GB RAM;
- Екран з роздільною здатністю екрана 1366x768 (HD), для більшої зручності створення діаграм.

Інсталяції система не вимагає. Необхідно завантажити стиснену папку .zip з програмною. Вона має містити файл EnergyFTA.exe та файли з ресурсами, які використовує програма – графіка, шрифт, бібліотека БД, файли ліцензії тощо.

Далі необхідно розпакувати цей архів у порожню папку для коректної роботи програми. Після чого потрібно запустити EnergyFTA.exe. Система самостійно створить БД та інші потрібні ресурси, участь користувача тут не потрібна.

4.2 Сценарії роботи користувача з системою

Після запуску .exe файлу користувача зустріне основний екран програми – робоча поверхня редактора побудови логіко-імітатаційної моделі. Робоча поверхня зображена на рисунку 4.1.

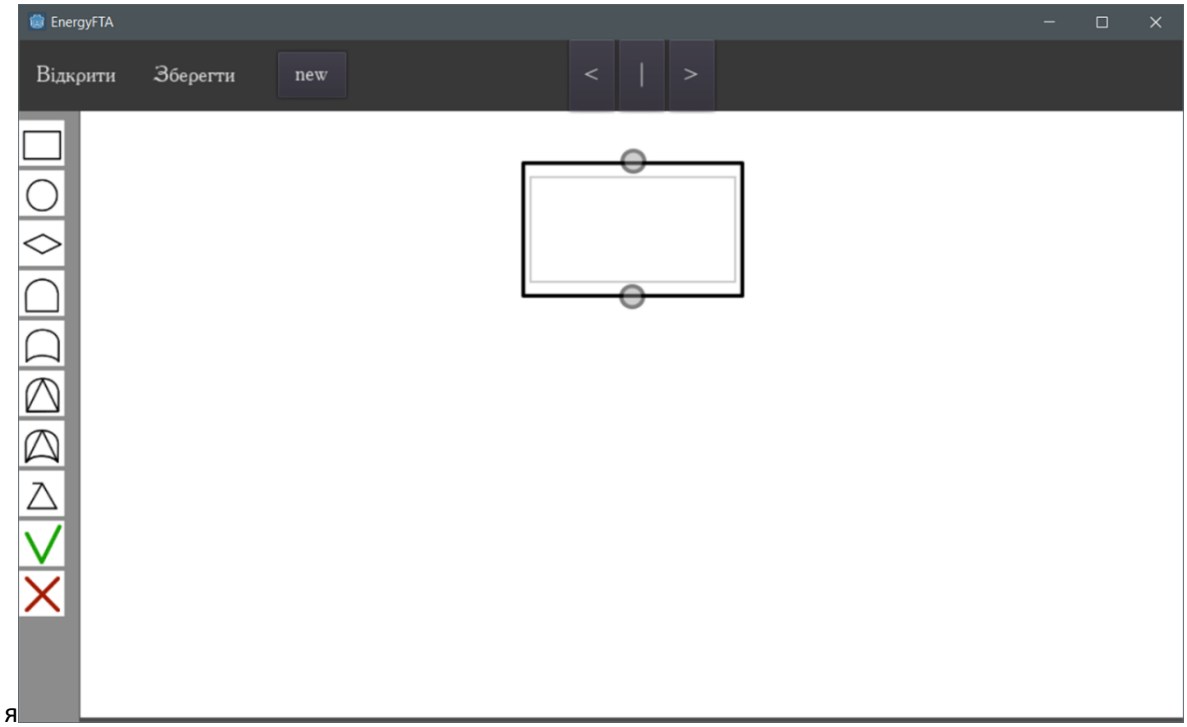

Рисунок 4.1 – Загальний вигляд вікна програми

У вікні програми можна побачити головні елементи програми: панель інструментів, панель керування, робоче полотно з діаграмою. Для керування положенням та розмірами робочого полотна необхідно скористатися гарячими клавішами. Стрілочками для пересування полотна, комбінаціями клавіш «ctrl +» та «ctrl -» для зміни масштабу зображення. Змінене положення полотна зображено на рисунку 4.2. Полотно буде автоматично збільшуватись з додаванням нових блоків.

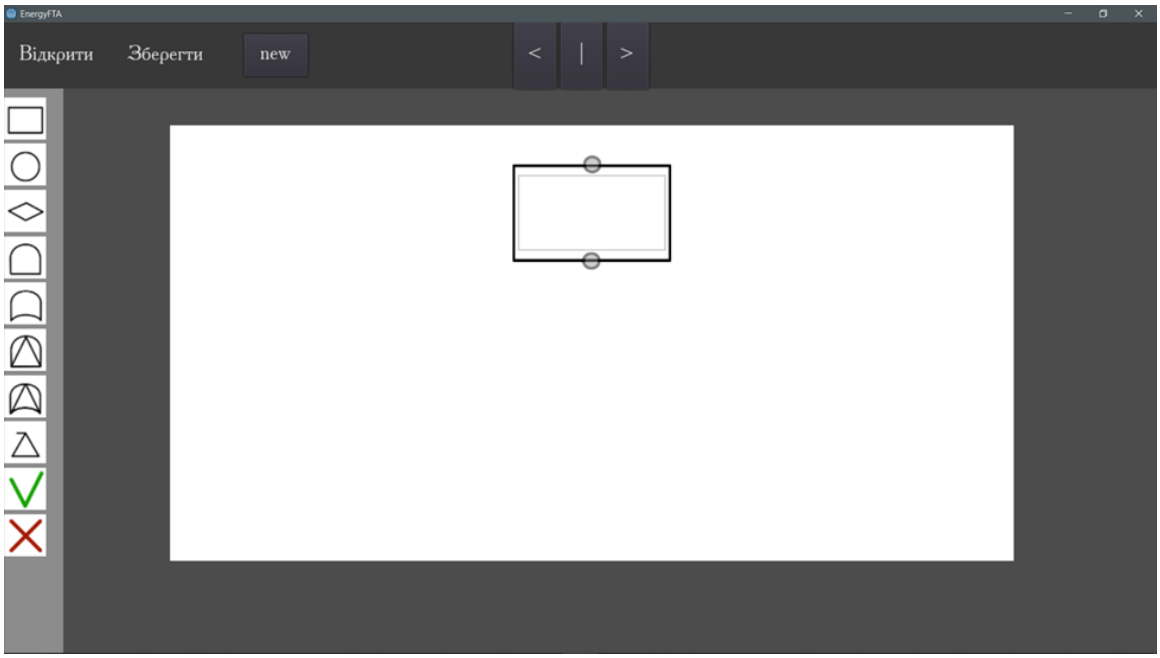

Рисунок 4.2 – Змінений масштаб полотна

Панель інструментів представлена у вигляді ряду кнопок, кожна з яких ілюструє блоки для побудови діаграми на рисунку 4.3. А саме:

– Проміжна подія. Використовується для вказівки події відмови, що відбувається через одну або декілька причин. Також є кореневим блоком діаграми.
– Основні ініціюючі події. Використовується для ілюстрації події відмови, що не вимагає розвитку, тобто є листком діаграми.
– Нерозривна подія. Використовується для вказання події, щодо якої не вистачає інформації або яка не грає значної ролі у дереві. Є листком діаграми.
– Логічний елемент «Та». Позначає, що подія, яка є батьківським елементом діаграми, відбудеться лише за умови виконання усіх вхідних подій.
– Логічний елемент «Або». Позначає, що подія вище відбудеться за умови виконання хоча б одної вхідної події.

– Логічний елемент «Пріоритетне Та». Позначає, що події вище відбудеться за умови виконання вхідних подій у певному порядку.
– Логічний елемент «Виключаюче Або». Подія вище відбудеться, якщо одна з вхідних подій точно відбудеться.
– Передача всередину. Використовується для зображення субдерева, яке зберігається у окремому файлі або як інший запис у БД.
– Блоки «так» та «ні» для побудови діаграми подій.

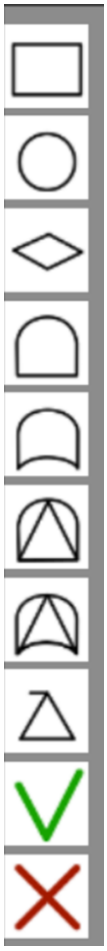

Рисунок 4.3 – Наявні інструменти

Розглянемо використання інструментів детальніше. Блок проміжної події має текстове поле, яке необхідно заповнити, показане на рисунку 4.4, для того, аби діаграма мала цінність в процесі аналізу.

Рисунок 4.4 – Заповнення текстового поля

Для додавання нових блоків на полотно діаграми необхідно обрати бажаний інструмент на панелі, зображеній на рисунку 4.5.

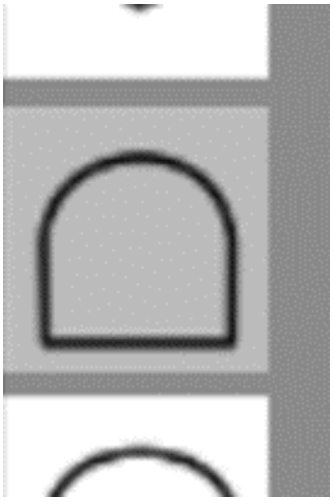

Рисунок 4.5 – Обраний інструмент

Після того потрібно обрати вузол блоку діаграми, до якої додається дочірній елемент. Якщо інструмент обрано, то відповідний блок з’явиться у дереві, що можна побачити на рисунку 4.6.

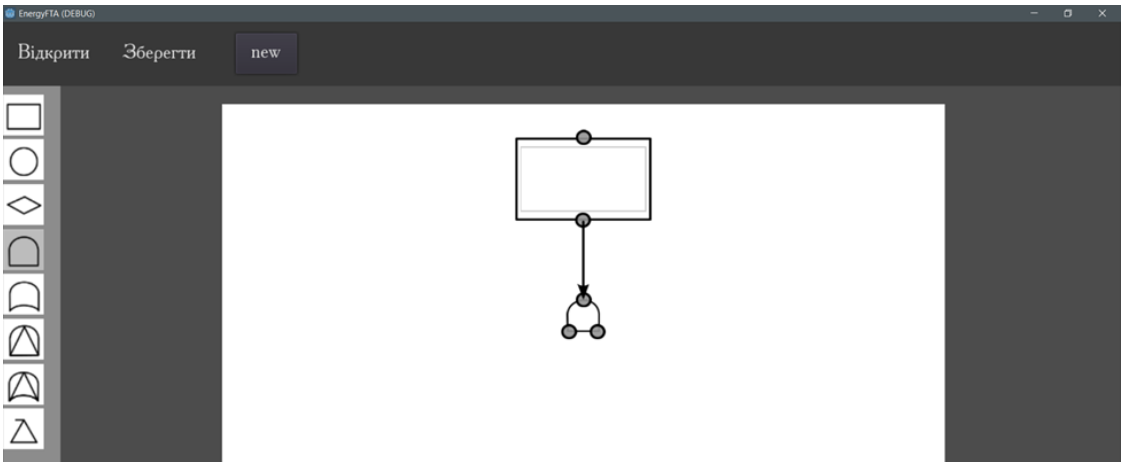

Рисунок 4.6 – Приклад доданого нащадка блоку

Для керування камерою перегляду варто використати стрілки на клавіатурі. Це дозволить пересуватися по полотну. Приклади доданих блоків-листків можна побачити на рисунку 4.7.

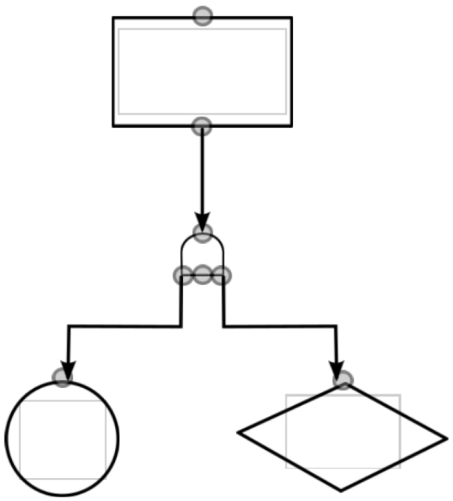

Рисунок 4.7 – Приклад блоків ініціюючої та нерозривної подій

Окремо необхідно розглянути роботу блоку «Передача всередину». Він має лише поле імені, що знаходиться біля нього, як на рисунку 4.8. Це поле має бути заповнене іменем іншої діаграми, на яку посилається цей блок, так, як це зображено на рисунку 4.9.

Рисунок 4.8 – Блок передачі всередину

Рисунок 4.9 – Приклад заповнення поля імені блоку передачі всередину

Для збереження діаграми необхідно натиснути на кнопку «Зберегти» панелі інструментів та обрати «Зберегти у БД» (рисунок 4.10), після чого відкриється вікно збереження діаграми до БД (рисунок 4.11).

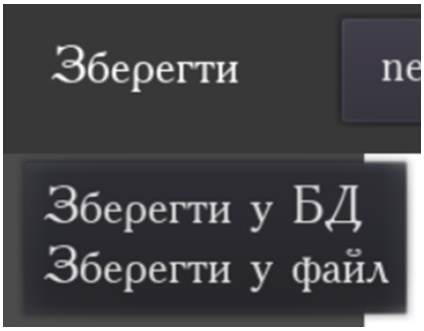

Рисунок 4.10 – Кнопка «Зберегти у БД», що випадає з панелі керування

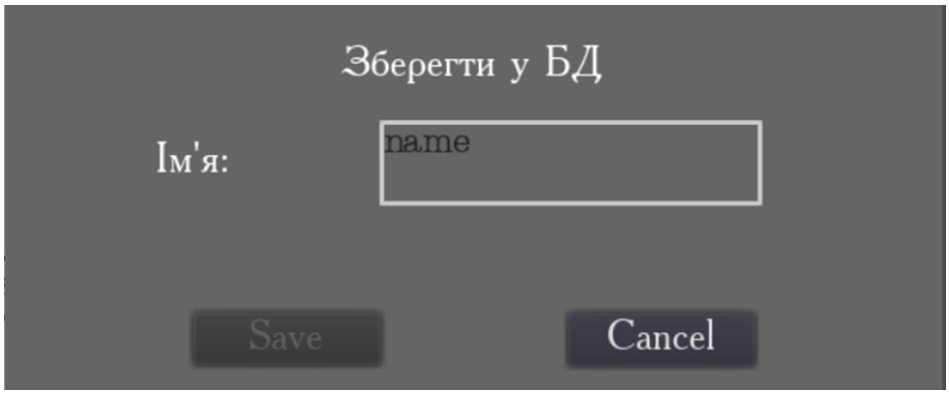

Рисунок 4.11 – Заповнене поле імені, доступна кнопка збереження

Але кнопка збереження буде недоступна доти, доки не буде заповнено поле «Ім’я». Для коректного збереження у базі даних воно має бути латинськими літерами. Після заповнення цього поля збереження стане доступне, так, як це є на рисунку 4.12.

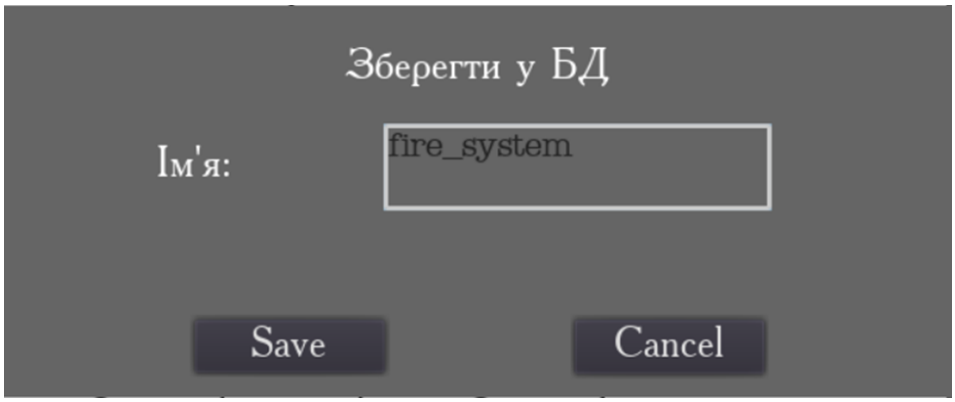

Рисунок 4.12 – Заповнене поле імені, доступна кнопка збереження

Для того, аби очистити полотно від діаграми, необхідно скористатися кнопкою «New» на панелі керування, вона поверне поле програми до початкового стану. Кнопку зображено на рисунку 4.13.

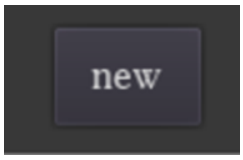

Рисунок 4.13 – Кнопка «New», що дозволяє створити нову діаграму

Для відкриття діаграми з бази даних необхідно натиснути на кнопку «Відкрити» панелі інструментів та обрати «Відкрити з БД» (рисунок 4.14), в результаті чого на екран виведеться вікно вибору діаграми (рисунок 4.15).

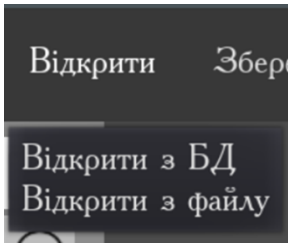

Рисунок 4.14 – Кнопка «Відкрити з БД», що випадає з панелі керування

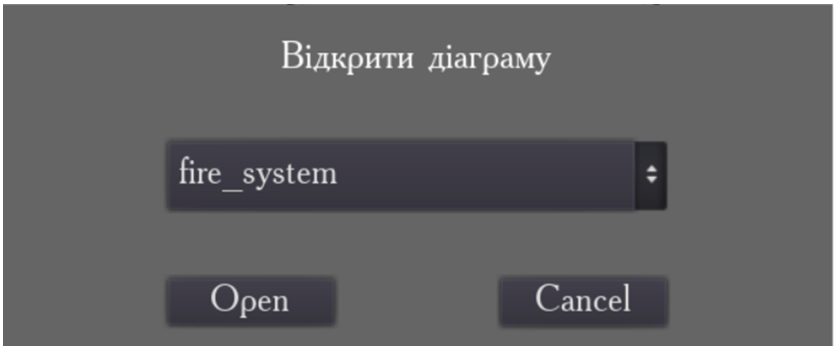

Рисунок 4.15 – Вікно відкриття діаграми, вибір з БД у вигляді списку

Приклад створеної у систем і EnergyFTA діаграми подій можна побачити на рисунку 4.16.

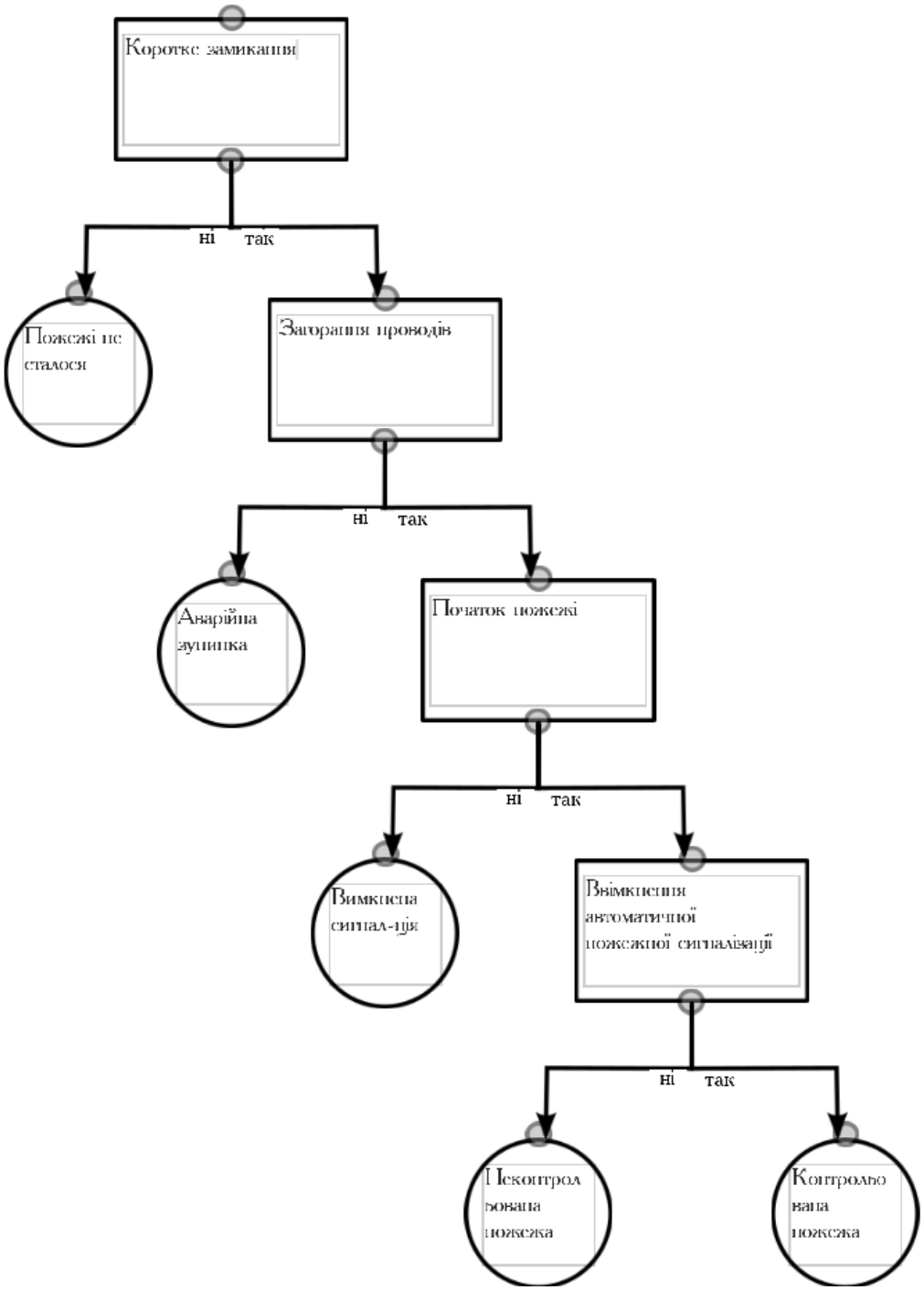

Рисунок 4.16 – Приклад діаграми подій

Приклад створеної у системі EnergyFTA діаграми відмов можна побачити на рисунку 4.17.

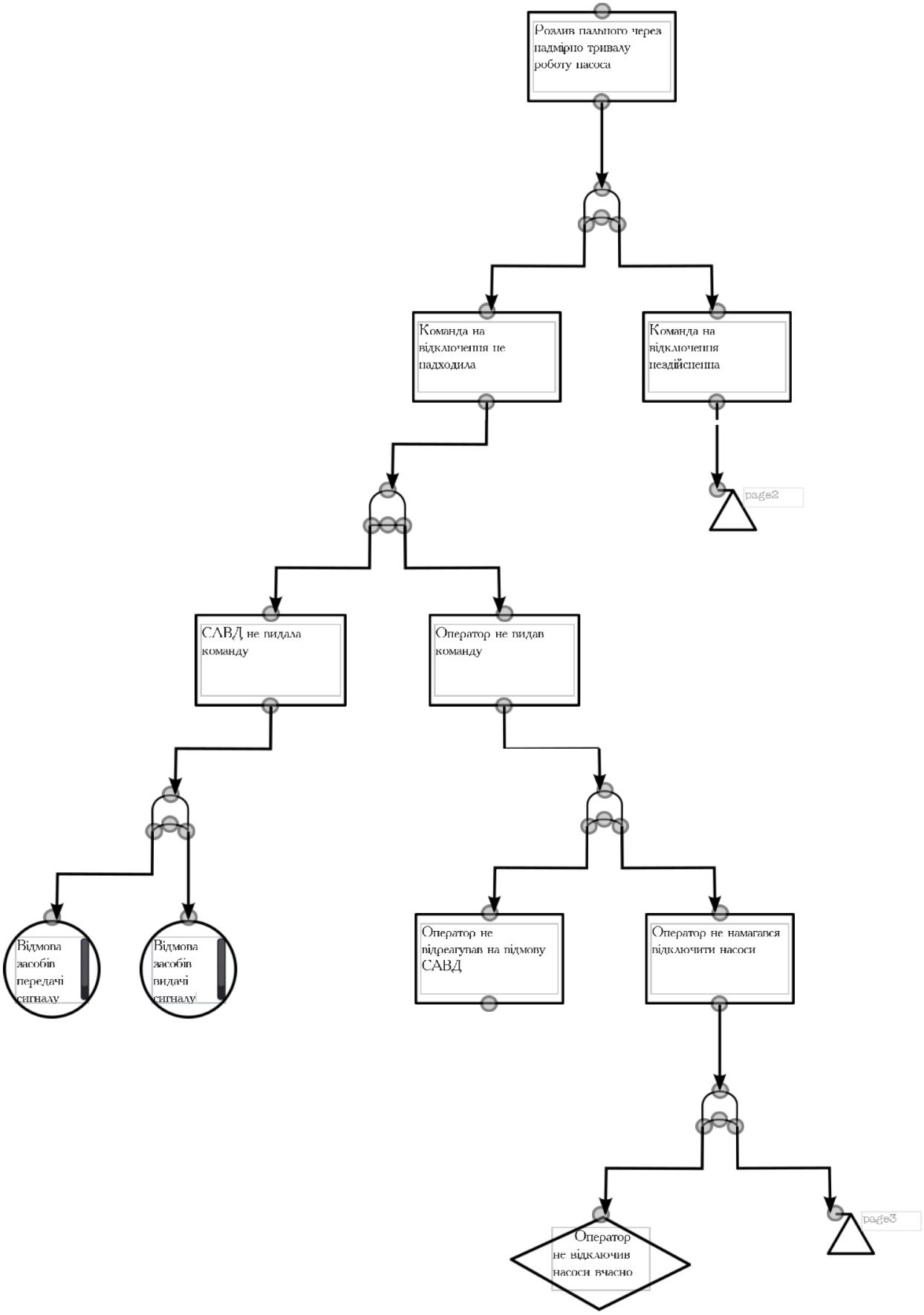

Рисунок 4.17 – Приклад діаграми відмов

Для того, аби зберегти розроблену діаграму прямо на комп’ютер, можна скористатися відповідним вікном, зображеним на рисунку 4.18.

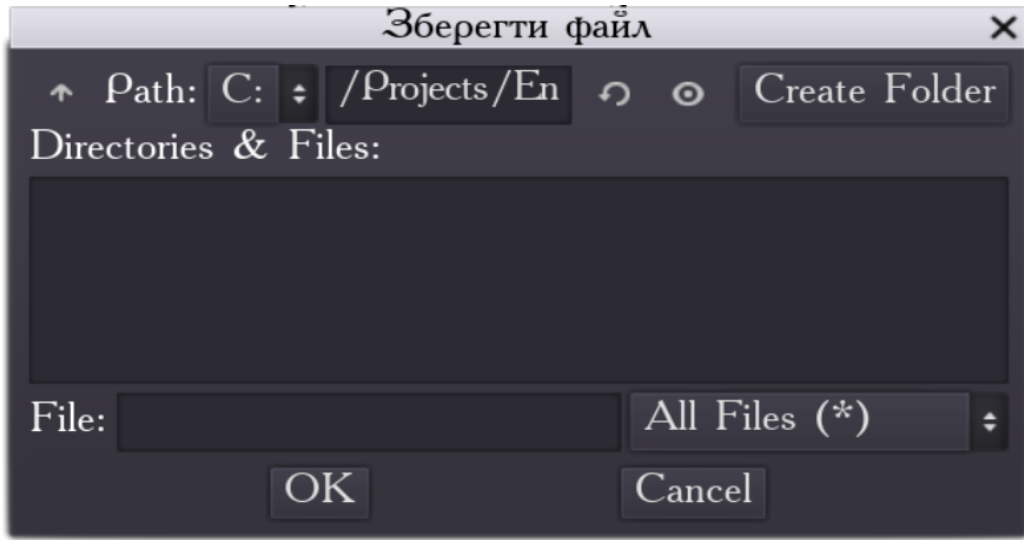

Рисунок 4.18 – Вікно збереження діаграми у файлову систему

Таким чином, система надає достатню кількість інструментів, щоби за лічену кількість кроків створити наочну діаграму дерева відмов або діаграму дерева подій.

ВИСНОВКИ

Доведено, що найбільш придатними методами побудови логіко-імітаційної моделі розвитку небезпечних подій на енергетичних об’єктах є «дерева відмов» та «дерева подій», оскільки вони мають широке застосування у сфері аналізу ризику небезпечних подій на об'єктах підвищеної небезпеки. Вищезазначені методи застосовують як для попереднього аналізу рівня безпеки під час формування рекомендацій для зниження рівня ризику, так і для розслідування причин аварій на небезпечних об'єктах.

Також досліджено обмеженість використання іноземних засобів, зокрема OpenFTA, ALD Fault Tree Analyser, Draw.io, Wondershare EdrawMax та інші, в задачах аналізу ризиків функціонування енергетичних об’єктів. Була розроблена нова система, що дозволить користувачам без додаткових встановлень ознайомитись з побудовою діаграмами «дерева відмов» та «дерева подій». Обрано рушій, а саме Godot 3.4.4 Mono version, а також NoSQL базу даних LiteDB. Створено систему, яку можна використовувати для побудови діаграм для оцінки ризиків на енергетичних об’єктах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ткаченко І. О. Ризики у транспортних процесах : навч. посібник /І. О. Ткаченко ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. Харків : ХНУМГ ім. О. М. Бекетова, 2017. 114 с.
2. Караєва Н.В., Войтко С.В., Сорокіна Л.В. Ризик-менеджмент сталого розвитку енергетики: інформаційна підтримка прийняття рішень: навчальний посібник. Київ, 2013. 308 с.
3. Керуванням ризиком. Методи загального оцінювання ризиків : (ДСТУ ISO/IEC 31010:2013, IDT) – [Чинний від 2014-07-01]. – Київ: Мінекономрозвитку України, 2015. – 73 с.
4. Бабаджанова Ф., Войтович Д. П. Використання методу FTA для аналізу небезпеки аміачних трубопроводів. Науковий вісник НЛТУ України. 2019, т. 29, № 7. С. 124-127.
5. Junfeng Wang Safety Theory and Control Technology of High-Speed Train Operation. Beijing, P.R. China, 2018. 387 p.
6. Тарадуда Д. В., Шевченко Р. І. Формування альтернативи оцінки ризику виникнення аварії на потенційних небезпекових об’єктах, до складу яких входять аміачні холодильні установки. Проблеми надзвичайних ситуацій. Харків: УЦЗУ, 2009. 161–170 с.
7. Березуцький В.В.., Адаменко М.І. Небезпечні виробничі ризики та надійність : навчальний посібник. Харків : НТУ ХПІ, 2016. 386 с.
8. C# 10 and .NET 6 – Modern Cross-Platform Development: Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code, Edition 6 / Mark J. Price. – Birmingham: Packt Publishing Ltd, 2021. 824 p.
9. Build your free fault tree now ALT Fault Tree Analysis : веб-сайт. URL: https://www.fault-tree-analysis-software.com/ (дата звернення: 06.05.2022)

10. Fault Tree Analysis (FTA) – Definition & Examples Wondershare EndrawMax : веб-сайт. URL: https://www.edrawmax.com/fault-tree-analysis/ (дата звернення: 03.05.2022)

11. Visual Paradigm Online – Infographic Maker : веб-сайт. URL: https://online.visual-paradigm.com/ (дата звернення: 07.05.2022)

12. Smartdraw – Fault Tree Analysis : веб-сайт. URL: https://www.smartdraw.com/fault-tree/ (дата звернення: 03.05.2022)

13. Creately – Create Diagram : веб-сайт. URL: https://app.creately.com/diagram/create/ (дата звернення: 04.05.2022)

14. ALD Service – Event Tree Analysis (ETA) : веб-сайт. URL: https://aldservice.com/Event-Tree-Analysis-ETA.html/ (дата звернення: 03.05.2022)

15. Draw.io – Linking content in draw.io diagrams : веб-сайт. URL: https://drawio-app.com/linking-content-in-draw-io-diagrams/ (дата звернення: 09.05.2022)

16. Isograph – Reliability Workbench : веб-сайт. URL: https://www.isograph.com/software/reliability-workbench/ (дата звернення: 10.05.2022)

17. Maintenance release: Godot 3.4.4 Godot : веб-сайт. URL: https://godotengine.org/article/maintenance-release-godot-3-4-4 (дата звернення: 04.05.2022)

18. Godot Docs – 3.4 branch Godot Docs : веб-сайт. URL: https://docs.godotengine.org/en/stable/ (дата звернення: 04.05.2022)

19. Overview LiteDB v5 - A .NET NoSQL Document Store in a single data file LiteDB : веб-сайт. URL: https://www.litedb.org/docs/ (дата звернення: 08.05.2022)

20. GeeksForGeeks – What is BSON? URL: https://www.geeksforgeeks.org/what-is-bson/ (дата звернення: 03.05.2022)

ДОДАТОК А

Текст програмного модулю

Інструментальні засоби побудови логіко-імітаційної моделі розвитку небезпечних подій на енергетичних об’єктах

УКР.НТУУ”КПІ ім. Ігоря Сікорьского”_ТЕФ_АПЕПС_ТР81231_22Б 12-1

Аркушів 11

Київ – 2022

```
#singletone DiagramData
extends Node

var template_data = {
"name" : "",
"blocks" : []
}

var template_block = {
"blockid": "0",
"type" : "",
"text" : "",
"arrow_type" : "none",
"arrow_output" : "output",
"connections": []
}

var diagrams_list = []

var file: File = File.new()
var dir: Directory = Directory.new()
var diagram_data: Dictionary
var file_exp = ".json"

func _ready():
create_new_diagram()

func save_progess(path) -> void:
file.open(path + file_exp, File.WRITE)
```

```
file.store_string( JSON.print(diagram_data) )
file.close()

func open_diagram(i : int):
if diagrams_list.size() > 0:
	diagram_data = DataBase.GetCollectionData(diagrams_list[i])

func load_progress(path) -> void:
file.open(path, File.READ)
if file.get_len() != 0:
	diagram_data = parse_json(file.get_as_text())
	if !diagram_data:
		create_new_diagram()
file.close()

func get_diagram_name() -> String:
return diagram_data.name

func get_block_count()-> int:
return diagram_data["block"].size

func create_new_diagram():
diagram_data = template_data.duplicate()

func change_diagram_name(new_name : String):
diagram_data.name = new_name

func add_new_block(blockid : String, type : String):
var new_block = template_block.duplicate()
```

```
new_block.connections = []
new_block.blockid = blockid
new_block.type = type
diagram_data.blocks.append(new_block)

func change_block_text(blockid : String, text : String):
diagram_data.blocks[blockid.to_int()].text = text

func change_arrow_type(blockid : String, arrow : String):
diagram_data.blocks[blockid.to_int()].arrow_type = arrow

func change_arrow_output(blockid : String, arrow : String):
diagram_data.blocks[blockid.to_int()].arrow_output = arrow

func add_block_connection(blockid : String, child : String):
diagram_data.blocks[blockid.to_int()].connections.append(child)

#singletone ToolsControl
extends Node

var current_tool = "none"
var prev_tool = "none"
var arrow = "straight"

func pressed_button(var button = "none"):
if button != "none":
	prev_tool = current_tool
	current_tool = button
	unpress_button(prev_tool)
```

```
func unpressed_button(var button = "none"):
if button == current_tool:
	current_tool = "none"

func unpress_button(var button = "none"):
if button != "none":
	get_tree().call_group(button, "set_pressed_no_signal", false)

#BlockClass

extends Control

class_name Block

signal output
signal output1
signal output2
signal outputleft
signal outputright

var timer_time = 1
var name_timer_time = 0.5

func get_output_pos(arrow) -> Vector2:
var pos = Vector2(0,0)
match arrow:
	"output" :
		pos = $Sprite/IOputs/Output.pos
```

```
        "output1" :
                pos = $Sprite/IOputs/Output1.pos
        "output2" :
                pos = $Sprite/IOputs/Output2.pos
        "outputleft" :
                pos = $Sprite/IOputs/OutputLeft.pos
        "outputright" :
                pos = $Sprite/IOputs/OutputRight.pos
return pos

func get_input_pos() -> Vector2:
return $Sprite/IOputs/Input.pos

func set_text(text):
$Text.text = text

func _on_Output_pressed():
emit_signal("output")

func _on_Output1_pressed():
emit_signal("output1")

func _on_Output2_pressed():
emit_signal("output2")

func _on_Text_text_changed():
$NameTimer.start(name_timer_time)

func _on_NameTimer_timeout():
```

```
DiagramData.change_block_text(name, $Text.text)


func _on_OutputLeft_pressed():
emit_signal("outputleft")


func _on_OutputRight_pressed():
emit_signal("outputright")

// class DataBase
using Godot;
using LiteDB;
using System;
using System.Collections.Generic;
using System.Linq;

public class DataBase : Godot.Node
{
    private static string ConnectionString = @"EnergyFTAdb.db";
    private LiteDatabase db;

    public class Diagrams
    {
        public int id { get; set; }
        public string name { get; set; }
        public int[] blocks_id { get; set; }
    }
```

```
public class Block
{
    public int id { get; set; }
    public string blockid { get; set; }
    public string type { get; set; }
    public string text { get; set; }
    public int arrow_id { get; set; }
    public string[] connections { get; set; }
}

public class Arrow
{
    public int id { get; set; }
    public string type { get; set; }
    public string output { get; set; }
}

public void ConnectDB()
{
    db = new LiteDatabase(ConnectionString);
}

public void SaveInDB(Godot.Collections.Dictionary data)
{
    var diagram_col = db.GetCollection<Diagrams>("diagrams");
    var blocks_col = db.GetCollection<Block>("blocks");
    var arrows_col = db.GetCollection<Arrow>("arrows");
```

```
            Godot.Collections.Array data_blocks =
(Godot.Collections.Array)data["blocks"];

            Diagrams diagram = new Diagrams();
            diagram.name = (string)data["name"];
            diagram.id = diagram_col.Count();
            diagram.blocks_id = new int[data_blocks.Count];

            for (int i = 0; i < data_blocks.Count; i++)
            {
                Block some_block = new Block();
                Arrow arrow = new Arrow();
                some_block.id = blocks_col.Count();
                Godot.Collections.Dictionary dictionary =
(Godot.Collections.Dictionary)data_blocks[i];
                some_block.blockid = (string)dictionary["blockid"];
                some_block.type = (string)dictionary["type"];
                some_block.text = (string)dictionary["text"];
                arrow.id = arrows_col.Count();
                arrow.type =  (string)dictionary["arrow_type"];
                arrow.output = (string)dictionary["arrow_output"];
                some_block.arrow_id = arrow.id;
                Godot.Collections.Array connects =
(Godot.Collections.Array)dictionary["connections"];
                some_block.connections = new string[connects.Count];
                for(int j = 0; j < connects.Count; j++)
                {
                    some_block.connections[j] = (string)connects[j];
                }
```

```
            arrows_col.Insert(arrow);
            blocks_col.Insert(some_block);
            diagram.blocks_id.Append(some_block.id);
        }

        diagram_col.Insert(diagram);
    }

    public Godot.Collections.Array GetDBList()
    {
        var diagram_col = db.GetCollection<Diagrams>("diagrams");
        Godot.Collections.Array connects = new Godot.Collections.Array();
        for (int i = 0; i < diagram_col.Count(); i++)
        {
            Diagrams some_d = (Diagrams)diagram_col.Query().Where(x =>
x.id.Equals(i));
            connects.Add(some_d.name);
        }
        return connects;
    }

    public Godot.Collections.Dictionary GetCollectionData(string diagram_name)
    {
        Godot.Collections.Dictionary dictionary = new
Godot.Collections.Dictionary();

        var diagram_col = db.GetCollection<Diagrams>("diagrams");
        var blocks_col = db.GetCollection<Block>("blocks");
        var arrows_col = db.GetCollection<Arrow>("arrows");
```

```
            dictionary["name"] = diagram_name;
            Diagrams diagram = (Diagrams)diagram_col.Query().Where(x =>
x.name.Equals(diagram_name));
            int size = diagram.blocks_id.Length;
            Godot.Collections.Array blocks = new Godot.Collections.Array();
            for(int i = 0; i< size; i++)
            {
                Godot.Collections.Dictionary dictionary1 = new
Godot.Collections.Dictionary();
                Block some_block = (Block)blocks_col.Query().Where(x =>
x.blockid.Equals(diagram.blocks_id[i]));
                Arrow arrow = (Arrow)arrows_col.Query().Where(x =>
x.id.Equals(some_block.arrow_id));
                dictionary1["blockid"] = some_block.blockid;
                dictionary1["type"] = some_block.type;
                dictionary1["text"] = some_block.text;
                dictionary1["arrow_type"] = arrow.type;
                dictionary1["arrow_output"] = arrow.output;
                Godot.Collections.Array connects = new Godot.Collections.Array();
                for (int j = 0; j < some_block.connections.Length; j++)
                {
                    connects.Add(some_block.connections[j]);
                }
                dictionary1["connections"] = connects;
            }
            dictionary["data_blocks"] = blocks;
            return dictionary;
        }
    }
```