НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"___" _____ 20 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему
Програмне забезпечення підсистеми побудови лабіринту на основі методів теорії графів
Спеціальність F2 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи

_____ Василюк-Зайцева С.В._
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _Пурхало Микита Олександрович_
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____"_____ 20 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту
_____ Пурхалу Микиті Олександровичу _____
(прізвище, ім'я, по батькові)

Спеціальність F2 – «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи _ Програмне забезпечення підсистеми побудови лабіринту на основі методів теорії графів_
затверджена наказом ректора НУБіП України від "16" грудня 2024 р. № 2249 "С"

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи
_____
Перелік питань, які потрібно розробити: _____
Перелік графічних документів (за потреби)
Дата видачі завдання "_____"_____20___ р.
Керівник бакалаврської кваліфікаційної роботи
_____ _____ _Василюк-Зайцева С.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Пурхало М.О.
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

DFS — (англ. Depth-First Search) алгоритм пошуку в глибину, який проходить вершини графа, заглиблюючись у кожну гілку максимально глибоко, перш ніж повернутися назад. Використовується для обходу або пошуку шляхів у графових структурах.

MVS — Microsoft Visual Studio; інтегроване середовище розробки програмного забезпечення від компанії Microsoft.

IDE — Integrated Development Environment, інтегроване середовище розробки

ІТ-технології — інформаційні технології; засоби та методи для створення, обробки, зберігання й передачі інформації за допомогою комп'ютерних систем.

ОС — операційна система; базовий набір програм, що забезпечує управління апаратними ресурсами комп'ютера та виконання прикладного ПЗ.

ПЗ — програмне забезпечення; сукупність програм і пов'язаних із ними даних, що забезпечують функціонування комп'ютерної системи.

ПК — персональний комп'ютер; універсальна електронна обчислювальна машина для індивідуального використання.

PNG (Portable Network Graphics) — растровий графічний формат для збереження зображень без втрати якості, з підтримкою прозорості.

PDF (Portable Document Format) — універсальний формат документів, що зберігає фіксоване макетування тексту, графіки та зображень.

SVG (Scalable Vector Graphics) — векторний формат на основі XML, призначений для збереження масштабованої графіки без втрати якості.

FBX (Filmbox) — формат для обміну 3D-моделями та анімацією між графічними програмами, з підтримкою матеріалів, освітлення та скелетної анімації.

Roguelike — жанр комп’ютерних ігор, що характеризується процедурною генерацією рівнів.

Puzzle — жанр ігор, що базується на вирішенні логічних або просторових задач, з акцентом на мислення, а не на швидкість чи реакцію.

Реіграбельність — здатність гри залишатися цікавою при повторному проходженні завдяки змінним сценаріям, нелінійності або процедурній генерації.

SQL (Structured Query Language) — мова структурованих запитів, яка використовується для створення, зміни та управління реляційними базами даних.

NoSQL (Not Only SQL) —назва для нереляційних баз даних, які зберігають дані у форматах, відмінних від таблиць (документи, графи, ключ-значення).

Юзабіліті (Usability) — зручність використання програмного забезпечення, що визначає ефективність, результативність та задоволеність користувача при взаємодії з інтерфейсом системи.

SDK (Software Development Kit) — набір інструментів для розробників, що включає бібліотеки, документацію, зразки коду та утиліти для створення програмного забезпечення під певну платформу.

XML (eXtensible Markup Language) — розширювана мова розмітки, призначена для зберігання та структурування даних у текстовому форматі, зручною для обміну між різними системами.

HTML (HyperText Markup Language) — мова розмітки гіпертексту, яка використовується для створення та структурування веб-сторінок у браузерах.

CSS (Cascading Style Sheets) — каскадні таблиці стилів; технологія, що дозволяє описувати зовнішній вигляд HTML-елементів: кольори, шрифти, розміри, розташування тощо.

JSON (JavaScript Object Notation) — текстовий формат обміну даними, заснований на синтаксисі JavaScript, зручний для зберігання та передачі структурованої інформації.

Python — високорівнева мова програмування загального призначення з простою синтаксичною структурою, що активно використовується в автоматизації, аналізі даних, штучному інтелекті, веб-розробці тощо.

ВСТУП

Перші згадки про лабіринти ведуть до давньогрецького міфу про Тесея і Мінотавра: за переказами, легендарний Критський лабіринт, збудований майстром Дедалом за наказом царя Міноса, слугував пасткою для чудовиська та символізував складність вибору, пошуків і випробувань, які стоять перед людиною. [21] В епоху Середньовіччя кам'яні та мозаїчні лабіринти прикрашали підлоги соборів, уособлюючи шлях до спасіння, а в Ренесансі зі зростанням інтересу до геометрії їх почали перетворювати на садові шедеври, що випробовували не лише просторову уяву, а й соціальний статус відвідувачів.

З розвитком математичної думки та особливо після формалізації теорії графів у XIX столітті лабіринти перестали бути лише декоративними візерунками. Сьогодні їх представляють як неорієнтовані або орієнтовані графи, де вершини відповідають кімнатам або клітинкам, а ребра — прохідним коридорам. Така інтерпретація відкрила шлях до строгого аналізу топологічних властивостей: зв'язності, циркуляції циклів, діаметра тощо. Цей підхід знайшов широке застосування в багатьох сучасних галузях.

Ігрова індустрія. Процедурна генерація рівнів у жанрах roguelike та puzzle забезпечує безмежну варіативність контенту й високу реіграбельність.

Робототехніка та автономна навігація. Лабіринтова постановка лежить в основі алгоритмів планування маршрутів у динамічному середовищі, зокрема для пошуково-рятувальних дронів.

Штучний інтелект і машинне навчання. Середовище лабіринту є класичним тестовим полігоном для навчання агентів методом підкріплення (reinforcement learning).

У XXI столітті використання лабіринтів вийшло за межі розваг. Вони стали ефективним інструментом для перевірки роботизованих систем навігації, тестування алгоритмів штучного інтелекту, дослідження властивостей дерев і графів, а також навчання студентів і школярів основам алгоритмізації. Їхня

структурна складність і гнучкість роблять їх зручним середовищем для моделювання різноманітних сценаріїв, особливо у випадках, коли важливо враховувати просторові обмеження, варіативність шляхів і взаємозв'язки між об'єктами.

Математичне моделювання лабіринтів на основі теорії графів дозволяє ефективно формувати топологічні структури з заданими параметрами, зокрема, розміром, типом генерації, рівнем складності тощо. За допомогою алгоритмів генерації, таких як алгоритм Прима, Крускала, глибина-перший пошук (DFS), рекурсивне ділення простору, метод Еллера та інші, можна створювати як регулярні, так і випадкові структури з контролем параметрів. Це відкриває можливості для як автоматизованого створення лабіринтів, так і подальшої інтерактивної роботи з ними.

Актуальність теми полягає в необхідності створення універсального інструменту для генерації лабіринтів, який дозволяє не тільки будувати складні структури, а й здійснювати їх візуалізацію, проходження та експорт у зручних форматах. Таке програмне забезпечення може бути застосоване в освіті, інженерії, віртуальній реальності, 3D-моделюванні та наукових дослідженнях, де моделювання просторових задач є ключовим елементом.

Дипломний проєкт спрямований на розробку системи, яка реалізує генерацію лабіринтів із використанням різних графових алгоритмів, забезпечує графічне представлення результату, його збереження у 2D- та 3D-форматах (PNG, PDF, SVG, FBX) та інтерактивне проходження лабіринту. Реалізація цього функціоналу дозволить створити потужний програмний інструмент, що поєднує в собі естетику, наочність та алгоритмічну точність.

Основні положення даної роботи були представлені у вигляді тез доповіді на VII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем» (НУБіП України, Київ, 2025) [23].

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Постановка задачі

Сьогодні в багатьох сферах діяльності активно впроваджуються системи, що потребують генерації складних структур, зокрема лабіринтів. Вони знаходять застосування у комп’ютерних іграх, навчальних платформах, симуляторах, візуалізаційних середовищах та алгоритмах побудови маршрутів. Важливо, щоб такі структури були логічно послідовними, без тупиків або циклів (де це необхідно), а також гнучко налаштовуваними. Це можливо реалізувати шляхом побудови генератора лабіринтів, заснованого на математично обґрунтованих методах, зокрема на теорії графів.

Мета даного дипломного проєкту — створення програмного інструменту, що дозволить автоматично генерувати лабіринти з урахуванням алгоритмів графів, забезпечить їх візуалізацію та надасть можливості для подальшого використання, наприклад, у навчальних або розважальних застосунках.

Теоретичне підґрунтя: роль теорії графів

Теорія графів — це розділ математики, що вивчає властивості особливих математичних структур — графів, які використовуються для моделювання парних відношень між об'єктами. У загальному вигляді граф можна уявити як конфігурацію з вершин (точок), з’єднаних ребрами (лініями). У строгому математичному формулюванні граф визначається як пара множин G = (V, E), де V — множина вершин, а E — множина ребер, тобто підмножина декартового добутку V × V. [21, 24]

Гнучкість і високий рівень абстракції дозволяють застосовувати графи для опису найрізноманітніших явищ та об’єктів реального світу. Завдяки цьому теорія графів стала універсальним інструментом для вирішення задач у сферах транспортного планування, комп’ютерних мереж, будівельного проєктування, молекулярного моделювання, геоінформаційних систем (ГІС) тощо. Наприклад,

у ГІС будинки та об'єкти інфраструктури розглядаються як вершини графа, а дороги, мережі електропередач, трубопроводи — як ребра. На основі обчислень на такому графі можна визначити найкоротший маршрут, найближчий об’єкт інфраструктури або оптимізувати логістику.

Теорія графів активно використовується у візуалізації структурованих даних, особливо там, де текстове чи табличне подання є недостатньо наочним. Її методи є основою алгоритмів обходу графа, пошуку найкоротших шляхів, побудови остовних дерев, кластеризації та оптимізації зв’язків. У програмуванні графи виступають не лише як засіб подання даних, а й як формальний механізм моделювання складних структур.

Початки формального вивчення графів закладені Леонардом Ейлером, зокрема в задачі про сім мостів Кенігсберга, що вважається першою задачею теорії графів. Попри вражаючий розвиток, ця теорія все ще містить багато невирішених задач і недоведених гіпотез, що робить її актуальною галуззю дослідження в сучасній науці.

Бізнес-вимоги

1. **Підвищення ефективності розробки лабіринтів**
   - Автоматизація рутинних операцій: система має максимально полегшити процес створення складних структур лабіринтів (наприклад, за допомогою вбудованих алгоритмів та готових шаблонів).
   - Скорочення часу розробки: завдяки зручному інтерфейсу та готовим налаштуванням система дає змогу суттєво зменшити час, необхідний на проєктування і тестування лабіринту.
   - Залучення ширшої аудиторії: оскільки створення лабіринтів стає доступнішим, ПЗ можуть використовувати не лише розробники ігор, а й викладачі, дизайнери симуляцій тощо.
2. **Гнучкість у налаштуванні**
   - Різноманітні параметри генерації: розмір, форма, тощо.

- Підтримка кастомних алгоритмів: можливість додавати власні або сторонні алгоритми формування лабіринтів.

3. **Інтеграція з існуючими системами**

- Підтримка базових стандартів: забезпечення експорту/імпорту за допомогою популярних форматів даних (PNG, PDF, тощо).

Функціональні вимоги

1. **Генерація лабіринтів на основі теорії графів**

- Реалізація кількох алгоритмів (Прима, Крускала, пошуку в ширину (BFS), пошуку в глибину (DFS), Recursive Backtracking тощо).

2. **Візуалізація лабіринтів**

- 2D-відображення: відобразити структуру лабіринту зверху (top-down view).

3. Експорт

- Графічні формати: PNG, PDF для швидкого перегляду та друку; векторні (SVG) — за потреби масштабування без втрати якості.
- Формат 3D: FBX для перенесення лабіринту у 3D простір і подальшого використання у системах редагування 3D об’єктів

Системні вимоги

1. **Платформа**

- Підтримка Windows 10/11: забезпечення сумісності з основними версіями ОС Windows.

2. **Продуктивність**

- Оптимізація візуалізації: використання ефективних методів рендерингу (кешування даних, GPU-акселерація).
- Мінімальні затримки: виконання операцій за прийнятний для користувача час (наприклад, не більше кількох секунд для генерації великого лабіринту).

3. **Мінімальні системні ресурси**

- Стандартна конфігурація ПК: 4 ГБ ОЗП, процесор з частотою 2 ГГц, інтегрована відеокарта.
- Відсутність потреби у спеціалізованому обладнанні: немає необхідності у дорогих GPU чи додаткових обладнаннях.

Нефункціональні вимоги

1. **Юзабіліті**
   - Інтуїтивний інтерфейс: логічно згруповані функції, підказки.
   - Документація та навчальні матеріали: наявність користувацьких посібників, прикладів використання.
2. **Масштабованість**
   - Архітектура, що легко розширюється: можливість додавання нових алгоритмів генерації чи типів візуалізації без суттєвого переписування коду.
   - Гнучка обробка даних: використання універсальних структур даних, що дозволяють ефективно змінювати розмір лабіринту або його форму.
3. **Надійність**
   - Стабільна робота: відсутність критичних збоїв чи вильотів за нормальних умов експлуатації.
   - Збереження даних: забезпечення цілісності й збереження проєктів лабіринтів.
   - Обробка помилок: інформативні повідомлення про помилки з поясненнями та можливими шляхами їх усунення.

Очікуваний результат

У результаті проєкту буде створено програмну підсистему, яка підтримує кілька методів генерації лабіринтів на основі теорії графів. Система дозволятиме будувати різноманітні за структурою лабіринти, легко налаштовувати параметри генерації, виводити графічне подання та експортувати результати.

1.2. Діаграма прецедентів

Діаграма прецедентів (*Use Case Diagram*) — один із основних типів діаграм у мові UML, який використовується для моделювання функціональних вимог до системи. Вона ілюструє, які дії можуть виконувати зовнішні актори (користувачі або інші системи) по відношенню до розроблюваного програмного забезпечення. Дана діаграма створюється на ранніх етапах проєктування і є корисною для узгодження очікуваної поведінки системи з потребами замовника або кінцевого користувача. [25]

Основні елементи діаграми:

- **Актор (Actor)** — зовнішній користувач або інша система, яка взаємодіє із програмним забезпеченням. Зазвичай відображається у вигляді стилізованої фігури людини.
- **Прецедент (Use Case)** — окрема функція або задача, яку може виконати користувач, відображається у вигляді еліпса.
- **Зв'язок** — лінія між актором і прецедентом, що означає ініціювання дії. Може доповнюватися спеціальними типами залежностей, такими як:
  - <<include>> — означає обов'язкове включення однієї функції в іншу;
  - <<extend>> — означає додаткову або умовну функціональність;
  - generalization — успадкування функцій або ролей.

Призначення та переваги:

- Формалізує функціональні вимоги до системи;
- Служить зрозумілою схемою для обговорення з користувачем;
- Виявляє основні сценарії використання системи;
- Дозволяє побачити ролі та повноваження різних типів користувачів.

Діаграма прецедентів на рис. 1.1, відображає основну функціональність системи генерації лабіринтів та взаємодію з нею двох типів користувачів — звичайного користувача та дизайнера лабіринтів.

Система представлена у вигляді великого прямокутника з назвою «Система генерації лабіринтів». У межах цієї системи показано сценарії (варіанти використання), які реалізує програмне забезпечення.

Актори:

- Користувач – взаємодіє з інтерфейсом системи для введення параметрів, вибору алгоритму створення лабіринту та проходження згенерованого лабіринту.
- Дизайнер лабіринту – має розширені можливості: створення лабіринту, його візуалізація у вигляді зображення або тривимірної моделі.

Основні варіанти використання:

- Ввести параметри лабіринту – користувач задає параметри, наприклад, висоту, ширину.
- Вибрати метод створення лабіринту – надається вибір одного з алгоритмів побудови: Прима, Крускала, DFS тощо.
- Пройти лабіринт – користувач може протестувати або пройти лабіринт у програмі.
- Створити лабіринт – функціональність, доступна дизайнеру, яка передбачає ручне чи автоматизоване створення лабіринтів.
- Візуалізувати лабіринт (зображення) – вивід лабіринту у графічному вигляді.
- Зображення – варіант виводу лабіринту у вигляді двовимірної схеми.
- Тривимірна модель – альтернатива зображенню, яка дозволяє експорт лабіринту у 3D-форматі.

Пунктирні стрілки між «Візуалізувати лабіринт (зображення)» та «Зображення»/«Тривимірна модель» означають розширення функціоналу, тобто система візуалізації може працювати в одному з двох режимів.

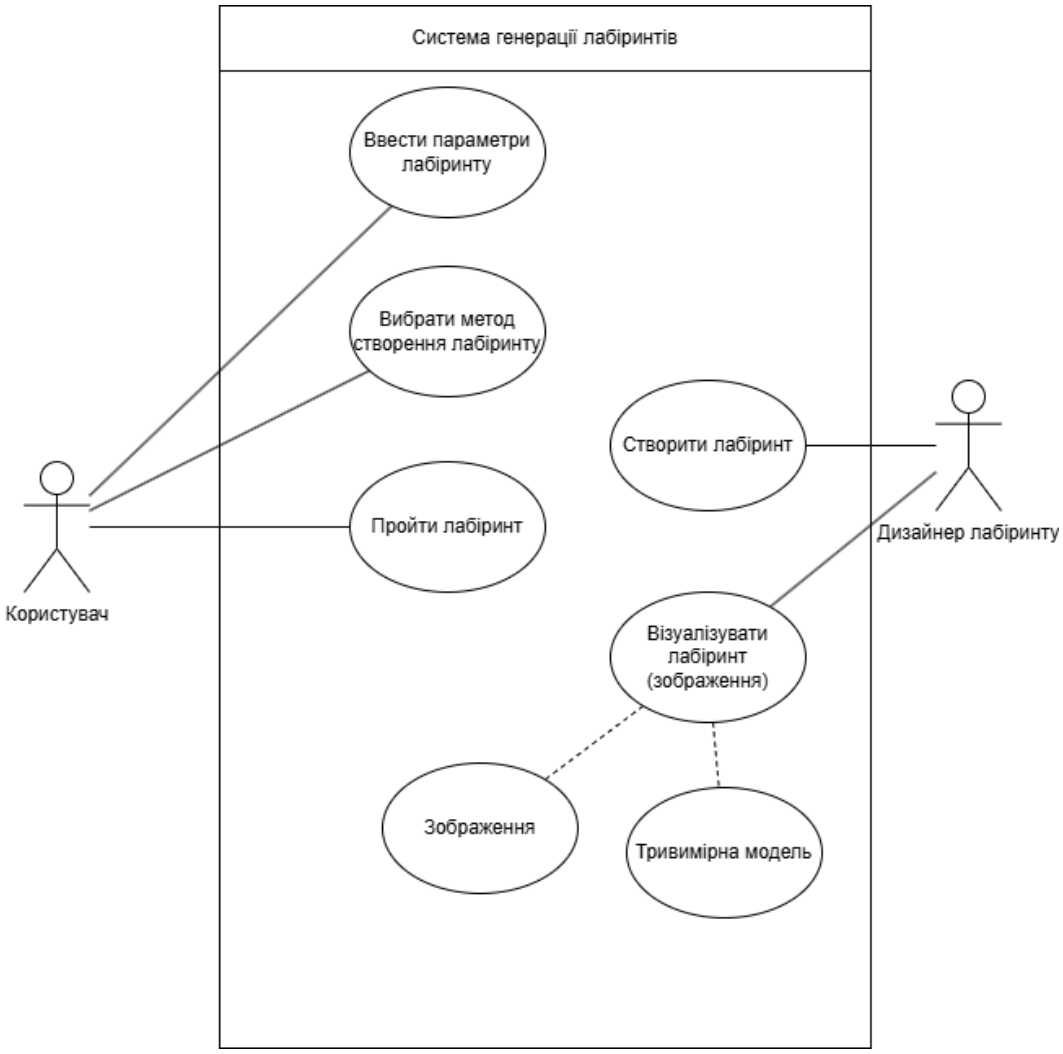

Рис. 1.1 Діаграма прецедентів

1.3. Діаграма послідовності

Діаграма послідовності — це один із типів діаграм UML, який використовується для моделювання динамічної взаємодії між об’єктами системи в часовій послідовності. Вона показує, які об’єкти беруть участь у процесі, які повідомлення вони надсилають один одному, а також у якому порядку це відбувається. [25]

Основні елементи діаграми послідовності:

- **Об’єкти (актор або компонент)** — відображаються у вигляді прямокутників із підписом у верхній частині діаграми. Вони

представляють користувачів або програмні компоненти, які беруть участь у взаємодії.

- **Лінії життя** (*lifelines*) — вертикальні штриховані лінії, що йдуть вниз від кожного об’єкта, ілюструють час життя об’єкта протягом сценарію.
- **Повідомлення (messages)** — горизонтальні стрілки між лініями життя, які позначають виклики методів, надсилання повідомлень або повернення результатів.
- **Активність (activation)** — вузький прямокутник на лінії життя, який позначає період, коли об'єкт виконує дію або обробляє повідомлення.
- **Повернення результату** — пунктирна стрілка з повідомленням про завершення дії або повернення значення.

На діаграмі послідовності, представленій на рис. 1.2, показано предметну область процесу створення лабіринту.

Учасники:

- **Користувач** — ініціює процес створення лабіринту, вводить параметри та може пройти створений лабіринт.
- **Дизайнер лабіринту** — відповідає за обробку параметрів і формування структури лабіринту, а також за його повернення користувачу.

Основні повідомлення:

1. **Ввести параметри лабіринту.** Користувач надсилає запит до дизайнера лабіринту з параметрами для генерації (розміри, алгоритм тощо).
2. **Повернення створеного лабіринту.** Дизайнер лабіринту обробляє введені параметри, формує структуру лабіринту та надсилає результат користувачеві. Це зворотне повідомлення зображено пунктирною стрілкою, що в UML означає відповідь або повернення результату.
3. **Проходження лабіринту.** Після отримання лабіринту, користувач розпочинає його проходження. Цей крок виконується вже без додаткової взаємодії з дизайнером.

Рис. 1.2 Діаграма послідовності

1.4. Діаграма активності

Діаграма активності — це один з типів діаграм UML, який використовується для моделювання потоків управління або діяльності в системі. Вона показує послідовність дій, які виконує система або користувач, і дозволяє візуалізувати логіку виконання процесу або алгоритму. [25]

Складові діаграми компонентів:

- **Початкова точка.** Чорне коло, з якого починається виконання діаграми.
- **Дія.** Прямокутник із закругленими кутами, що представляє конкретну дію або крок.
- **Перехід.** Стрілка, яка вказує напрямок переходу між діями.
- **Умовне розгалуження.** Ромб, з якого виходять кілька стрілок з умовами.
- **Злиття.** Ромб, в який входять кілька гілок, що об’єднуються в одну.
- **Паралельне розгалуження.** Товста горизонтальна або вертикальна лінія, з якої виходять паралельні потоки.

- **Паралельне злиття.** Товста лінія, в яку входять паралельні потоки і об'єднуються.
- **Кінцева точка.** Коло з темною крапкою всередині, позначає завершення активності.

Діаграма активності на рис. 1.3, відображає основний алгоритм для користування програмою.

1. **Початок процесу**
   - Початковий стан позначено жирною чорною крапкою.
2. **Перевірка введених параметрів**
   - Визначається, чи введені параметри лабіринту (розмір, алгоритм генерації).
   - Якщо параметри не введені → встановлюються стандартні параметри.
   - Якщо параметри введені → використовуються введені значення.
3. **Генерація лабіринту**
   - Після вибору параметрів запускається процес генерації лабіринту.
4. **Відображення лабіринту**
   - Згенерований лабіринт відображається користувачеві.
5. **Вибір подальших дій**
   - Користувач може або почати проходження лабіринту, або відмовитися від цього.
6. **Якщо користувач відмовляється від проходження:**
   - Є можливість експортувати лабіринт у форматі PNG, PDF тощо.
   - Якщо користувач не хоче експортувати – процес завершується.
   - Якщо лабіринт експортується – після експортування процес завершується.
7. **Якщо користувач починає проходження лабіринту:**
   - Користувач проходить лабіринт або може достроково завершити проходження.

- Якщо лабіринт повністю пройдено, повертається до етапу відображення лабіринту.

8. **Завершення процесу**
    - Процес завершується після експорту або відмови від подальших дій.

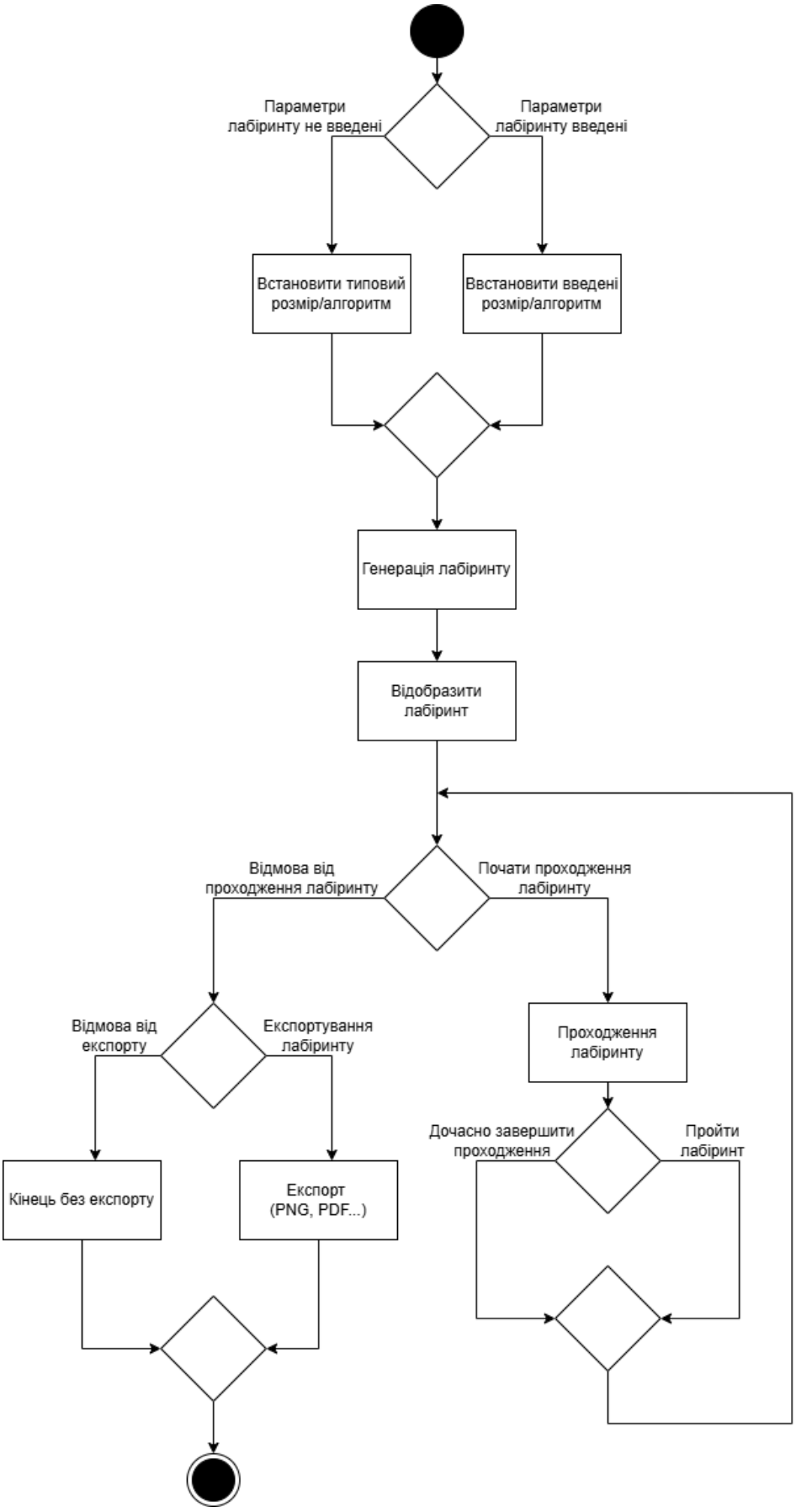

Рис. 1.3 Діаграма активності

1.5. Діаграма класів з простими кооперaціями

Діаграма класів (англ. *Class Diagram*) — один із ключових типів діаграм у мові моделювання UML, який використовується для статичного опису структури системи. Вона відображає класи, їхні атрибути, методи та взаємозв'язки між класами. Такі діаграми дозволяють формалізувати логіку взаємодії об'єктів у системі на рівні структури даних і моделі предметної області. [20]

Прості кооперації у контексті діаграми класів — це основні типи зв'язків між класами, які відображають базові способи взаємодії без ускладнень, таких як патерни проєктування або специфічні механізми зв'язку. До таких кооперацій належать:

Основні типи зв'язків у діаграмі класів:

1. **Асоціація.** Показує зв'язок між двома класами, які «знають» один про одного. Наприклад, клас Labirynth може мати зв'язок з класом Cell, що означає, що лабіринт містить клітинки.
2. **Агрегація.** Це слабкий тип зв'язку «ціле–частина». Клас може складатися з інших класів, але при цьому частини можуть існувати окремо. Наприклад, Maze агрегує Wall, але стінки можуть існувати незалежно від конкретного лабіринту.
3. **Композиція.** Це сильна форма агрегації: якщо клас-компонент знищується, то знищується і його складова частина. Наприклад, якщо об'єкт MazeGenerator знищується, усі згенеровані клітинки (Cell) також перестають існувати.
4. **Успадкування.** Відображає відношення між базовим (батьківським) класом і похідним. Наприклад, DFSGenerator, PrimsGenerator, KruskalsGenerator можуть успадковуватися від базового класу MazeGenerator.

5. **Залежність.** Показує, що один клас використовує інший у своїх методах. Наприклад, UIController може залежати від Maze, оскільки оновлює його візуалізацію.

Призначення діаграми з простими кооперaціями

- Візуалізує логічну структуру системи;
- Допомагає на етапі проєктування класифікувати сутності та їхні зв’язки;
- Сприяє створенню об’єктно-орієнтованого коду без дублювання функціоналу;
- Полегшує спілкування між розробниками, тестувальниками та іншими учасниками проєкту.

Опис основних класів та їх взаємозв’язки представлених на рис.1.4:

1. **Інтерфейс (Інтерфейс)**
    - Відповідає за створення, візуалізацію та експорт лабіринтів.
    - Має атрибут:
        - ui_elements: List<UIComponent> – список елементів інтерфейсу.
    - Методи:
        - export() – експортує лабіринт.
        - creation() – створює лабіринт.
        - visualize() – візуалізує лабіринт.
    - **Взаємозв’язки**:
        - Ініціалізується модулем проходження.
        - Використовує генератор лабіринтів, візуалізатор і модуль експорту.
        - Зв’язок із модулем проходження позначений пунктирною лінією (залежність).
2. **Модуль проходження (Модуль проходження)**
    - Відповідає за управління проходженням лабіринту.

- Має атрибут:
    - maze: Maze – лабіринт, що використовується.
- Методи:
    - display_maze() – відображає лабіринт.
    - end_game() – завершує гру.
    - KeyDown() – обробляє натискання клавіш.
- **Взаємозв’язки**:
    - Взаємодіє з Інтерфейсом, ініціалізуючи його.

3. **Генератор лабіринтів (Генератор лабіринту)**
    - Відповідає за створення лабіринтів за допомогою різних алгоритмів.
    - Атрибути:
        - algorithm: string – алгоритм генерації.
        - maze: Maze – об'єкт лабіринту.
    - Методи:
        - Prima() – алгоритм Прима.
        - Krusckala() – алгоритм Крускала.
        - BSF() – пошук у ширину.
    - **Взаємозв’язки**:
        - Використовує Клітину, оскільки лабіринт складається з клітин.
        - Взаємодіє з Інтерфейсом.
4. **Клітина (Клітина)**
    - Описує окрему клітину лабіринту.
    - Атрибути:
        - Top: bool – чи є верхня стіна.
        - Right: bool – чи є права стіна.
        - Bottom: bool – чи є нижня стіна.

- Left: bool – чи є ліва стіна.
- Visited: bool – чи була клітина відвідана.

- **Взаємозв’язки**:
  - Кожен лабіринт містить множину клітин (зв’язок 4..* з генератором лабіринтів).

5. **Візуалізатор (Візуалізатор)**
   - Відповідає за відображення лабіринту.
   - Атрибути:
     - graphics: Graphics – об'єкт для графічного відображення.
     - maze: Maze – лабіринт.
     - cell_size: int – розмір клітини.
   - Методи:
     - display_maze() – візуалізує лабіринт.
   - **Взаємозв’язки**:
     - Використовується інтерфейсом.
6. **Експортер**
   - Відповідає за збереження лабіринту у файл.
   - Атрибути:
     - format: string – формат файлу (PNG, PDF тощо).
     - path: string – шлях до файлу.
     - maze_data: Maze – збережений лабіринт.
   - Методи:
     - export() – експортує лабіринт.
   - **Взаємозв’язки**:
     - Взаємодіє з Інтерфейсом.

Зв’язки між класами

- **Композиція (суцільний ромб)**:

- Генератор лабіринту компонує Клітини, оскільки лабіринт складається з клітин.
- Візуалізатор є частиною Інтерфейсу.
- Збережені лабіринти містять Maze, тому це також композиція.

- **Агрегація (порожній ромб)**:
  - Візуалізатор агрегує Maze, оскільки може відображати різні лабіринти.
- **Асоціація (пряма лінія)**:
  - Інтерфейс пов’язаний із Генератором лабіринтів, Візуалізатором і Експортом лабіринту.
- **Залежність (пунктирна стрілка)**:
  - Інтерфейс залежить від Модуля проходження, оскільки використовує його функціональність.

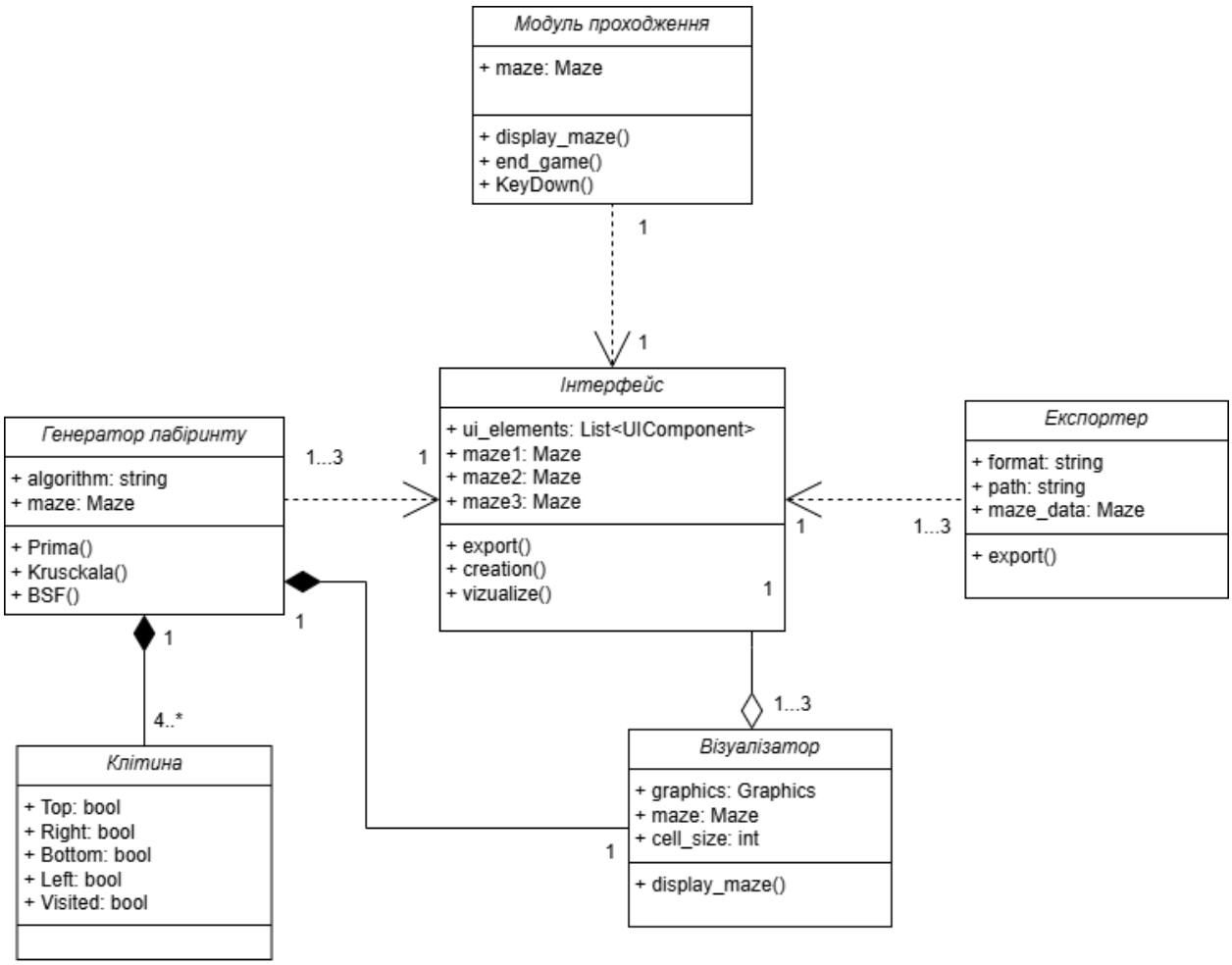

Рис. 1.4 Діаграма простих кооперацій

1.6. Діаграма пакетів

Діаграма пакетів (*Package Diagram*) — це один із типів статичних діаграм у мові UML, який використовується для відображення модульної структури програмної системи. Вона демонструє, як логічно згруповані класи, інтерфейси або інші елементи системи об'єднуються в пакети (модулі) і як ці пакети взаємодіють між собою. [19]

Призначення діаграми пакетів:

- Формалізувати архітектурну структуру проєкту;
- Визначити відповідальність окремих підсистем;
- Зменшити зв’язаність і підвищити модульність системи;
- Спрощувати супровід та розширення ПЗ;
- Надати уявлення про залежності між окремими логічними блоками.

Основні елементи діаграми:

- Пакет (package) — прямокутник із вкладеною назвою, який представляє групу класів, інтерфейсів або інших елементів.
- Залежність між пакетами — стрілка з пунктирною лінією та стрілочкою, що показує, що один пакет використовує функціональність іншого.
- Ієрархія пакетів — пакети можуть бути вкладеними, що дозволяє деталізувати структуру проєкту.

Типи залежності між пакетами

1. Import (Імпорт)

- Один пакет імпортує інший, отримуючи доступ до його публічних елементів.
- Позначається пунктирною стрілкою з відкритою стрілкою та стереотипом <<import>>.

2. Access (Доступ)

- Один пакет може використовувати публічні елементи іншого пакета, але не імпортує їх безпосередньо.
- Позначається так само, як <<import>>, але з міткою <<access>>.
- Використовується, коли пакет працює з іншим пакетом, але не включає його до своєї області видимості.

3. Merge (Об’єднання)

- Один пакет розширює або доповнює інший, додаючи або перевизначаючи елементи.
- Використовується для рефакторингу та модифікації існуючих моделей.
- Позначається пунктирною стрілкою з відкритою стрілкою та міткою <<merge>>.

4. Dependency (Залежність)

- Один пакет залежить від іншого на рівні реалізації або використання.
- Позначається пунктирною стрілкою без спеціального стереотипу.
- Використовується, коли зміна в одному пакеті може вплинути на інший.

5. Containment (Вкладеність)

- Один пакет містить інший як підпакет.
- Позначається звичайною лінією без стрілки.
- Використовується для організації проєкту у логічні групи.

Основні компоненти та їх взаємозв’язки представлених на рис.1.5:

1. **System of Maze Generation (Система генерації лабіринтів)**
    - Основний модуль, який включає всі внутрішні компоненти, необхідні для роботи програми.
2. **UI (Інтерфейс користувача)**
    - Відповідає за взаємодію користувача із системою.
    - Отримує команди від користувача та передає їх іншим компонентам.
    - Взаємодіє з:
        - Visualizer (Візуалізатор) – для відображення лабіринту.

- Export (Експорт) – для збереження лабіринту у файл.
- Maze Exploring (Проходження лабіринту) – для навігації та аналізу.

3. **Visualizer (Візуалізатор)**
    - Відповідає за графічне представлення лабіринту.
    - Отримує дані від Maze Generator (Генератора лабіринту).
    - Повертає зображення до UI.
4. **Maze Generator (Генератор лабіринту)**
    - Відповідає за створення лабіринту на основі алгоритмів.
    - Використовує Cell (Клітину) як базову структуру.
    - Отримує команди від Visualizer.
5. **Cell (Клітина)**
    - Основний елемент лабіринту.
    - Визначає структуру та кордони лабіринту.
    - Є частиною Maze Generator.
6. **Export (Експорт)**
    - Відповідає за збереження лабіринту у різних форматах (наприклад, PNG, PDF).
    - Взаємодіє з UI для отримання команд.
7. **Maze Exploring (Проходження лабіринту)**
    - Відповідає за механізми дослідження та навігації лабіринтом.
    - Взаємодіє з UI.

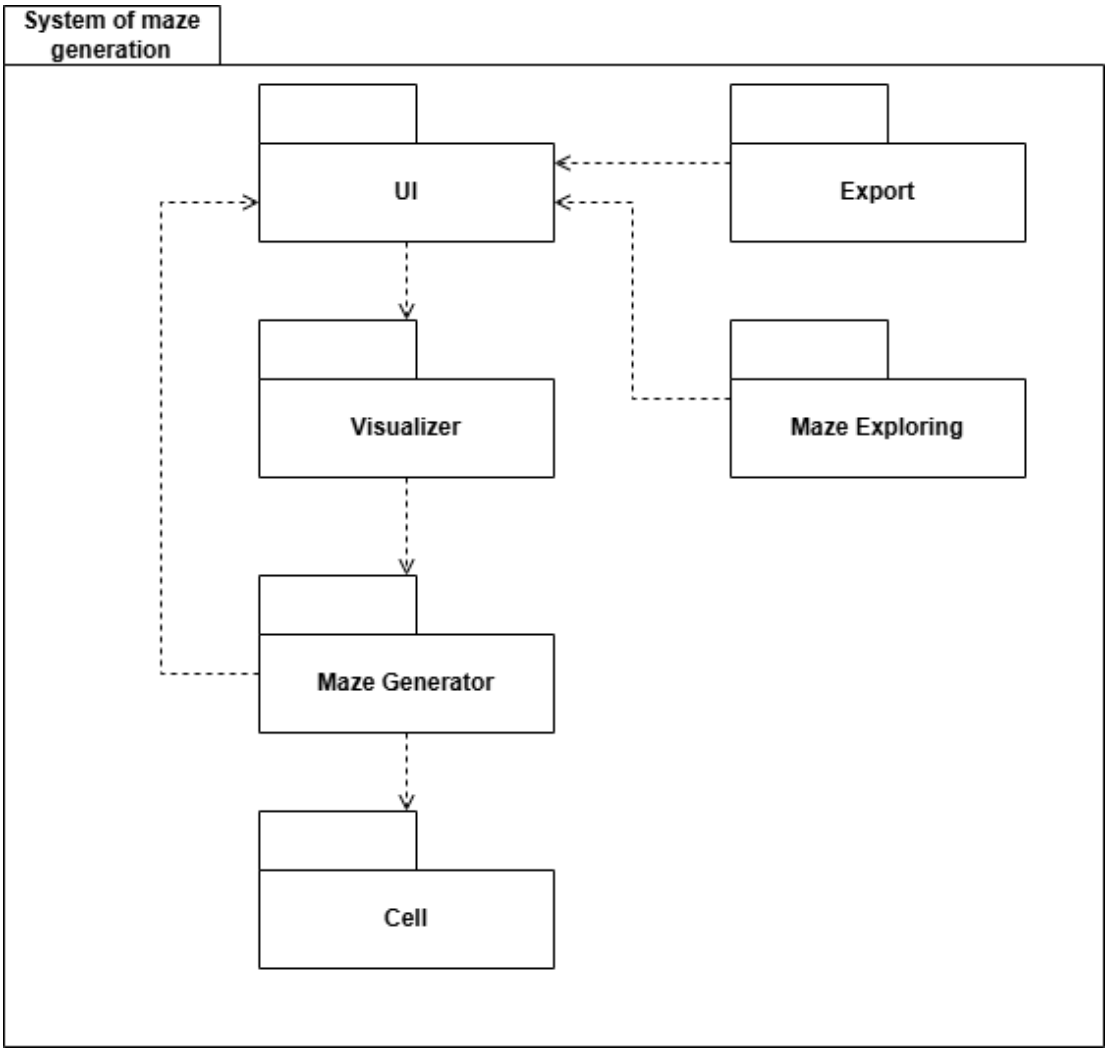

Рис. 1.5 Діаграма пакетів

1.7. Висновки до розділу

У цьому розділі було здійснено ґрунтовний аналіз предметної області, що стосується побудови, візуалізації та подальшого використання лабіринтів у програмному забезпеченні. Розглянуто теоретичні аспекти, які лежать в основі роботи системи, зокрема, застосування алгоритмів теорії графів для генерації структури лабіринтів. Детально описано основні бізнес-вимоги до майбутньої системи, які враховують необхідність автоматизації процесу генерації, забезпечення візуального представлення та зручності для користувачів різних категорій.

Також сформульовано функціональні та нефункціональні вимоги до системи, що охоплюють параметри гнучкості, масштабованості, зручності

інтерфейсу, швидкодії, надійності та стабільності. Визначено основні функції, які має виконувати система, серед яких генерація лабіринтів різними алгоритмами, візуалізація результату, його експорт у різних форматах та проходження згенерованого лабіринту користувачем.

Для відображення логіки роботи системи побудовано набір UML-діаграм:

- діаграма прецедентів відображає ролі користувача і дизайнера лабіринтів та функціональні можливості ПЗ;
- діаграма послідовності демонструє часову динаміку викликів під час створення і проходження лабіринту;
- діаграма активності формалізує типовий сценарій взаємодії із системою;
- діаграма класів із простими кооперaціями деталізує взаємозв'язки основних компонентів системи на рівні структури коду;
- діаграма пакетів ілюструє модульну архітектуру системи та її компонентів, виділяючи підсистеми за функціональною відповідальністю.

Таким чином, на базі проведеного аналізу сформовано концептуальну основу програмного продукту, яка забезпечує подальший перехід до проєктування інформаційної структури та реалізації функціональних компонентів системи.

2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. ER-діаграма

ER-діаграма (діаграма «сутність-зв'язок») — це модель, яка використовується для опису структури бази даних у термінах об'єктів (сутностей), атрибутів і зв'язків між ними. Вона широко застосовується на етапі проєктування баз даних, особливо у реляційних СУБД.

Основні компоненти ER-діаграми

1. **Сутність.** Це об'єкт реального світу або логічної моделі, який підлягає зберіганню в базі даних. На ER-діаграмі позначається прямокутником.
2. **Атрибути.** Це властивості, які описують сутність. Позначаються еліпсами, пов’язаними з сутністю.

   Види атрибутів:

   - Первинний ключ — унікальний ідентифікатор сутності.
   - Зовнішній ключ — атрибут, що встановлює зв’язок з іншою сутністю.
   - Складений атрибут — атрибут, що складається з кількох частин.
   - Багатозначний атрибут — може мати кілька значень для однієї сутності.
3. **Зв’язки.** Відображають логічний зв’язок між двома або більше сутностями. Позначаються ромбом, з'єднаним із сутностями.
4. **Кратність** Вказує, скільки об'єктів однієї сутності може бути пов’язано з об'єктами іншої. Основні типи:

   - Один до одного (1:1)
   - Один до багатьох (1:N)
   - Багато до багатьох (M:N)
5. **Слабка сутність.** Це сутність, яка не має власного первинного ключа і залежить від іншої. Позначається подвійною рамкою.

ER-діаграма на рис. 2.1, показує основні зв’язки у розроблюваному програмному забезпечені.

Сутності

1. **Animation**
   - **Ключ (PK)**: animation_id
   - **Поля**:
     - on_off: bool – вказує, чи ввімкнена/вимкнена анімація.
     - time: int – певна характеристика часу анімації
2. **Maze_size**
   - **Ключ (PK)**: maze_size_id
   - **Поля**:
     - width: int – ширина лабіринту у клітинках.
     - hight: int – висота.
3. **Maze_color**
   - **Ключ (PK)**: maze_color_id
   - **Поля**:
     - red: int
     - green: int
     - blue: int

     (три компоненти кольору, у форматі RGB)
4. **Maze**
   - **Ключ (PK)**: maze_id
   - **Зовнішні ключі (FK)**:
     - maze_size_id → Maze_size.maze_size_id
     - maze_color_id → Maze_color.maze_color_id
     - animation_id → Animation.animation_id
   - **Поля**:

- algorithm: String – назва/тип алгоритму генерування лабіринту.
- graphics: Graphics – посилання на графічний об’єкт (у коді може бути складний тип).
- cell_size: int – розмір клітинки (наприклад, у пікселях).

5. **Cell**
   - **Ключ (PK)**: cell_id
   - **Зовнішній ключ (FK)**: maze_id → Maze.maze_id
   - **Поля**:
     - top: bool – чи є стіна зверху.
     - right: bool – чи є стіна праворуч.
     - bottom: bool – чи є стіна знизу.
     - left: bool – чи є стіна ліворуч.
     - visited: bool – прапорець «клітинку вже відвідали» (корисно в алгоритмах побудови або пошуку шляху).
6. **Maze_explorer**
   - **Ключ (PK)**: maze_explorer_id
   - **Зовнішній ключ (FK)**: maze_id → Maze.maze_id
   - **Поля**:
     - player_coord_X: int – координата гравця (X).
     - player_coord_Y: int – координата гравця (Y).

Зв’язки

Всі зв’язки мають «Ідентифікуючий зв’язки»

1. **Maze** – **Maze_size**:
   - Кожен рядок у Maze має власний запис із розмірами (Maze_size), і навпаки.
2. **Maze** – **Maze_color**:
   - Кожен лабіринт має унікальний опис кольору.
3. **Maze** – **Animation**:

- Кожен лабіринт має одну анімаційну конфігурацію (animation_id).

4. **Maze** – **Cell**:
   - Maze складається з багатьох Cell. Тобто **один Maze** – **багато Cell**.
5. **Maze** – **Maze_explorer**:
   - Один лабіринт може мати 0 або кількох систем проходження, і кожен «explorer» пов’язаний рівно з одним Maze.

Загальна логіка схеми

1. **Maze** є «центральною» сутністю, яка вказує на свої параметри (Maze_size, Maze_color, Animation).
2. Список усіх клітин (Cell) зберігає посилання на конкретний Maze.
3. **Maze_explorer** – відображає позицію гравця у певному лабіринті.

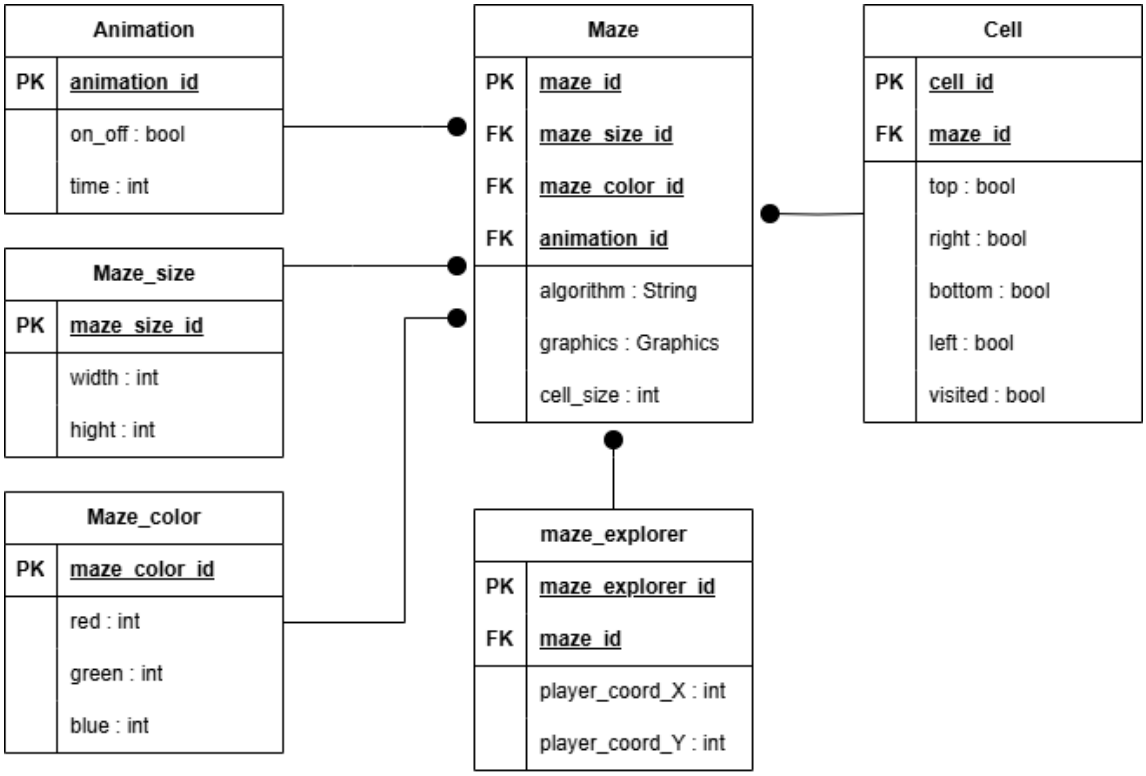

Рис. 2.1 ER-діаграма

2.2. Вибір та обґрунтування СУБД

Під час проєктування програмного забезпечення для генерації та візуалізації лабіринтів важливим аспектом є зберігання інформації про створені структури, параметри генерації, кольорові схеми, типи алгоритмів, тощо. Проте характер таких даних відрізняється від класичного облікового підходу: він є переважно сесійним, тимчасовим або візуально залежним, що суттєво впливає на вибір засобів збереження.

У рамках даного проєкту було проаналізовано доцільність використання реляційної СУБД (SQL) та нереляційних підходів (NoSQL, файлові системи). Оскільки система не потребує складного багаторівневого збереження, складної аналітики або транзакцій між сутностями, а основні дані — це згенеровані об'єкти у графічному чи 3D-форматі, було вирішено відмовитися від повноцінної СУБД на користь локального файлового зберігання.

Усі результати генерації лабіринтів зберігаються у вигляді файлів (PNG, PDF, SVG, FBX) у вибраній користувачем директорії. Такий підхід має кілька переваг:

- Мінімальна ресурсозалежність — відсутня потреба у встановленні або адмініструванні СУБД;
- Швидкий доступ до збережених результатів без проміжної обробки;
- Гнучкість у виборі форматів експорту залежно від подальшого використання (2D-друк, 3D-візуалізація, обмін файлами).

Для внутрішнього представлення параметрів (розмір, алгоритм, колір, режим анімації) використовуються структури та об’єкти на рівні коду, які зберігаються лише при потребі експорту.

Таким чином, вибір не використовувати повноцінну СУБД є повністю обґрунтованим з огляду на цільове призначення програмного забезпечення, його архітектуру та вимоги до продуктивності.

2.3. Діаграма компонентів

Діаграма компонентів (Component Diagram) — це тип структурної діаграми в UML, яка використовується для моделювання фізичної архітектури програмного забезпечення, тобто того, як система розбита на модулі (компоненти) та як вони взаємодіють між собою.

Ця діаграма показує розділення коду на функціональні частини, зокрема бібліотеки, сервіси, підсистеми, інтерфейси та залежності між ними.

Основні елементи діаграми компонентів:

1. **Компонент.** Логічна частина програмного забезпечення, яка виконує певну функцію. Позначається прямокутником із маленьким символом (дві вкладені прямі лінії) вгорі зліва.
2. **Інтерфейс.** Визначає, які операції компонент надає або використовує. Позначається кружечком (надає інтерфейс) або півколом (використовує інтерфейс).
3. **Зв'язки.** Вказують, що один компонент залежить від іншого (тобто використовує його функціональність). Зображуються пунктирною стрілкою від споживача до постачальника.
4. **Порти.** Використовуються для уточнення точки входу/виходу даних між компонентами.

Діаграма компонентів на рис. 2.2 показує, як взаємодіють різні підсистеми та компоненти між собою, демонструє створення, проходження та візуалізацію лабіринтів.

1. Підсистема інтерфейсу користувача для проходження лабіринту

- Компонент: Алгоритм обробки подій користувача
- Вхідні дані: Дані виконаної події
- Вихідні дані: передаються до Підсистеми проходження лабіринту

2. Підсистема генерації лабіринту

- Компонент: Алгоритм генерації лабіринту
- Отримує вхідні налаштування від Підсистеми інтерфейсу користувача для створення лабіринту
- Вихідні дані: Дані згенерованого лабіринту → передаються до інших підсистем (наприклад, проходження та візуалізації)

3. Підсистема інтерфейсу користувача для створення лабіринту

- Компонент: Налаштування лабіринту
- Вихід: Налаштування лабіринту → передаються до Підсистеми генерації лабіринту

4. Підсистема проходження лабіринту

- Компонент: Алгоритм проходження
- Отримує:
  - Дані згенерованого лабіринту
  - Дані з інтерфейсу користувача
- Вихід: Результат проходження

5. Підсистема управління файлами

- Компонент: Алгоритм експорту
- Вхід: отримує результати або лабіринт
- Вихід: Експортований файл

6. Підсистема візуалізації

- Компонент: Алгоритм візуалізації лабіринту
- Отримує: Дані згенерованого лабіринту
- Вихід: Візуалізований лабіринт

Взаємозв'язки:

- Всі підсистеми тісно взаємодіють між собою:
  - Дані лабіринту проходять через генерацію → проходження → візуалізацію → експорт.
  - Користувач впливає як на створення, так і на проходження лабіринту.

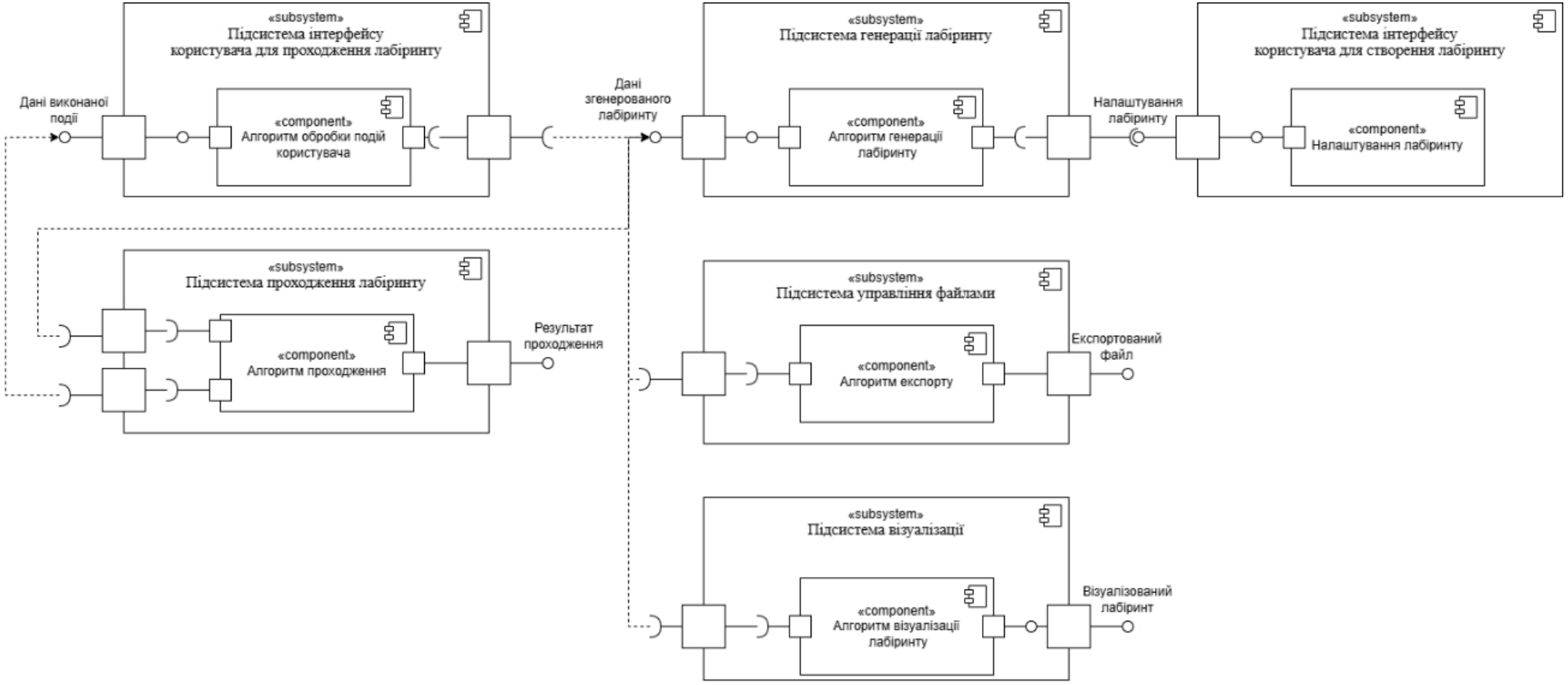

Рис. 2.2 Діаграма компонентів

2.4. Висновки до розділу

У цьому розділі розглянуто інформаційне забезпечення програмної системи, що включає структуру даних, необхідну для коректної роботи генератора лабіринтів. Зокрема, побудовано ER-діаграму, яка описує сутності (Maze, Cell, Animation, Maze_size, Maze_color, Maze_explorer) та зв’язки між ними. Дана модель дозволяє відобразити всі необхідні параметри, які формують структуру лабіринту, його кольорову гаму, розміри, спосіб візуалізації та проходження. Особливу увагу приділено використанню ідентифікуючих зв’язків між сутностями, що забезпечує логічну цілісність та уніфікацію структури збережених даних.

Проаналізовано доцільність використання бази даних: оскільки система не потребує складного збереження структурованих даних або виконання запитів, то було прийнято рішення про збереження інформації у вигляді готових файлів (PNG, PDF, SVG, FBX). Таке рішення спрощує архітектуру системи, знижує залежність від зовнішніх сервісів та забезпечує швидкий доступ до результатів генерації.

Додатково було представлено модель фізичного рівня, де зазначено, що всі згенеровані результати зберігаються у директоріях, обраних користувачем, без потреби в централізованій БД. Діаграма компонентів, представлена у розділі, дозволила сформувати уявлення про логічну архітектуру системи, взаємодію основних компонентів (генерація, проходження, візуалізація, експорт) та їхнє інтегрування через інтерфейс користувача.

Отже, в результаті аналізу інформаційного забезпечення було створено чітке формалізоване уявлення про внутрішню структуру даних, способи зберігання результатів роботи та взаємодію компонентів системи, що у подальшому дозволяє ефективно реалізувати програмний функціонал.

3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Середовище розробки Microsoft Visual Studio

На рис. 3.1. Представлено інтерфейс MVS

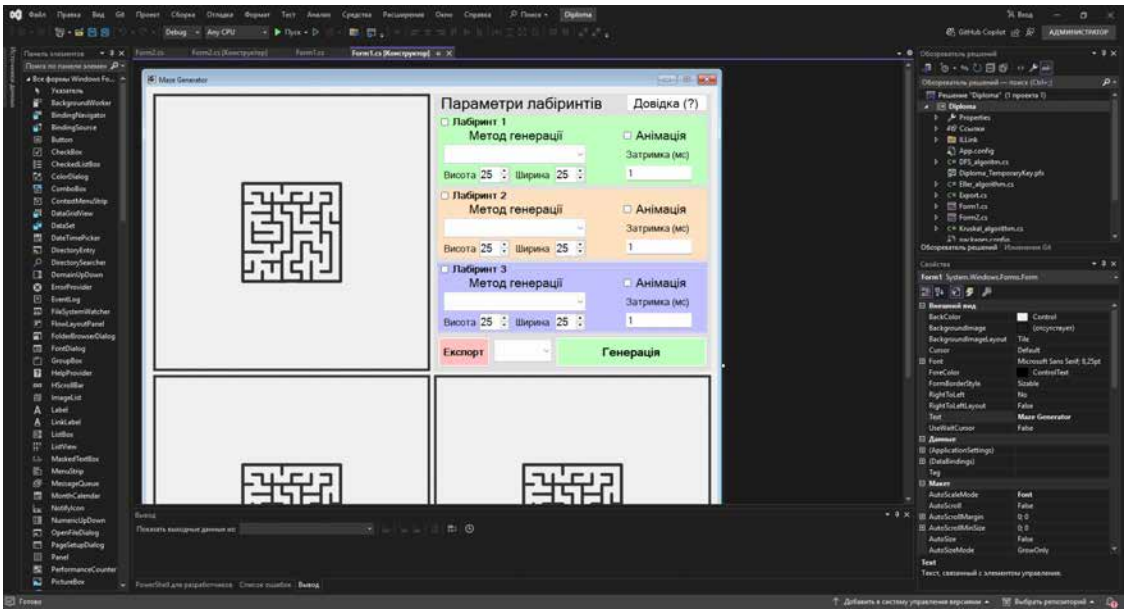

Рис. 3.1. Середовище MVS

Короткий огляд розвитку

Microsoft Visual Studio — це інтегроване середовище розробки, призначене для створення додатків різного типу: від консольних і графічних застосунків до вебсервісів та мобільних програм. Воно підтримує низку мов програмування, зокрема C#, VB.NET, C++, F# та інші, і працює на базі платформи .NET. [10]

Історія версій:

- **Visual Studio 97** — перша версія, яка об’єднала кілька мов (VB, VC++, J++, FoxPro) в одному пакеті. З'явилась підтримка веб-розробки через Visual InterDev.
- **Visual Studio 6.0 (1998)** — остання версія для Windows 9x. Була базовим середовищем до появи .NET.

- **Visual Studio .NET 2002–2003** — вперше представлена платформа .NET та мова C#. Програми компілювались у MSIL-код, який виконується у віртуальному середовищі CLR. З’явилась підтримка мобільних застосунків.
- **Visual Studio 2005–2008** — додано підтримку 64-розрядних додатків, інтеграцію з Microsoft Office, розширення LINQ, WPF, WCF, а також Windows Vista і .NET Framework до версії 3.5. Запроваджено Express-редакції для новачків.
- **Visual Studio 2010** — новий інтерфейс на базі WPF, підтримка F# та .NET Framework 4.0, вдосконалені інструменти для тестування, хмарних і мобільних застосунків.
- **Visual Studio 2012–2013** — акцент на розробці під Windows 8 і 8.1, інтеграція з Windows Runtime та нові засоби діагностики. З’явилась підтримка C++11 та GPGPU-програмування (C++ AMP).
- **Visual Studio 2015** — значне розширення можливостей: підтримка iOS, Android, Unity, оновлений механізм авторизації через Microsoft Account. Вперше реалізовано єдину платформу для багатьох ОС.
- **Visual Studio 2017–2019** — покращено інсталятор, продуктивність та підтримку сучасних вебтехнологій. Версія 2019 вийшла для Windows і macOS одночасно.
- **Visual Studio 2022** — перша повноцінна 64-бітна версія, яка дозволяє працювати з великими проєктами без обмежень у пам’яті. Підтримує .NET 6/7, розробку під MAUI, Blazor, Azure тощо. Надалі активно вдосконалюється з кожним оновленням. [11]

Стан на 2025 рік

Visual Studio залишається основним інструментом для розробників на платформі .NET. Вона постійно оновлюється, додаючи нові засоби для хмарної, мобільної, десктопної та кросплатформової розробки.

3.2. Мова програмування C Sharp (C#)

Загальний опис

C# (C Sharp, «Сі-шарп») — це об’єктно-орієнтована мова програмування, розроблена під платформу .NET корпорацією Microsoft. Вона поєднує строго типізовану систему з широкими можливостями для розробки додатків різного масштабу: від настільних і веб-додатків до мобільних і серверних рішень.

Мову було створено Андерсом Гейлсбергом, Скотом Вілтамутом і Пітером Гольде під егідою дослідницького підрозділу Microsoft. Вона успадкувала синтаксис і логіку від мов C++, Java, Object Pascal та інших, адаптуючи перевірені концепції з урахуванням безпеки, стабільності та зручності використання. [4]

Особливості синтаксису

C# має синтаксис, схожий на C++ та Java, але орієнтований на більш безпечне програмування. Мова підтримує:

- строгу статичну типізацію;
- поліморфізм, перевантаження операторів, властивості та події;
- виключення для обробки помилок;
- XML-коментарі для генерації документації;
- вказівники, але тільки в «небезпечному» (unsafe) режимі.

Однією з концептуальних відмінностей від C++ є відсутність множинного успадкування класів, що спрощує розробку та зменшує ризик конфліктів.

Походження та зв’язок із Java

C# тісно пов’язаний із Java — як за ідеологією, так і за технічним підходом. Обидві мови мають віртуальне середовище виконання, автоматичне управління пам’яттю, байт-код і незалежність від апаратної архітектури.

У 1990-х Microsoft ліцензувала Java від компанії Sun Microsystems. Однак через порушення ліцензійних умов — зокрема, спроби зробити Java залежною від Windows — Microsoft була змушена відмовитися від її використання після судового рішення. Як наслідок, було створено власну мову — C#, яка врахувала

сильні сторони Java та додала нові механізми, орієнтовані на інтеграцію з іншими мовами в межах платформи .NET.

Інтеграція з платформою .NET

C# тісно інтегрована з .NET Framework / .NET Core / .NET 6+, що дає їй перевагу в написанні додатків будь-якої складності. Платформа забезпечує:

- багату бібліотеку класів;
- автоматичне управління пам’яттю (GC);
- середовище виконання CLR (Common Language Runtime);
- сумісність між різними мовами завдяки спільному байт-коду MSIL.

Завдяки цій сумісності C# дозволяє легко взаємодіяти з кодом, написаним на інших мовах, якщо вони також працюють у середовищі .NET.

Поточний стан та розвиток

Станом на 2025 рік актуальною стабільною версією мови є C# 13.0, яка входить до складу платформи .NET 9.0. Microsoft продовжує активно розвивати C#, фокусуючись на зниженні кількості шаблонного коду, підвищенні продуктивності виконання та спрощенні написання декларативного, модульного і асинхронного коду.

Мова постійно поповнюється новими функціональними можливостями, серед яких:

- покращена підтримка патерн-матчингу;
- удосконалені спан-типи та діапазони;
- розширена робота з null-значеннями;
- покращення у source generators та метапрограмуванні;
- підтримка розширеного lambda-синтаксису та виразів зі значеннями за умовчанням.

На сьогодні C# залишається флагманською мовою Microsoft, яка найкраще реалізує усі новітні можливості екосистеми .NET. Незважаючи на наявність альтернатив, таких як VB.NET або F#, саме C# розглядається як найбільш гнучка,

широко підтримувана та рекомендована мова для розробки застосунків будь-якого рівня складності на платформі .NET.

3.3. Формат файлу PNG

Загальний опис

PNG (Portable Network Graphics) — це растровий графічний формат, створений для зберігання зображень високої якості з підтримкою прозорого та напівпрозорого фону. Його часто використовують у вебдизайні та цифровій графіці, адже формат не потребує ліцензії й відкривається у більшості графічних редакторів без обмежень. PNG-файли мають розширення .png та здатні зберігати до 16 мільйонів кольорів, що робить їх значно гнучкішими порівняно з багатьма старішими форматами. [13]

Походження формату PNG

Формат був створений у 1995 році як альтернатива формату GIF, який на той час вже мав певні недоліки, зокрема обмеження палітри до 256 кольорів і патентні обмеження на використання. Ініціатор створення формату — спеціаліст з IT Олівер Фромм (Oliver Fromme), який запропонував назву PING (згодом скорочену до PNG). Головною метою створення нового формату було надання відкритого, вільного від патентів засобу зберігання зображень із широкою кольоровою палітрою та підтримкою прозорості.

Переваги формату PNG:

- Висока деталізація. PNG підтримує мільйони кольорів, що дозволяє зберігати дуже якісні зображення з точними кольорами та градієнтами.
- Відкрита специфікація. Формат не ліцензований, тому підтримується практично всіма графічними редакторами та платформами.
- Стиснення без втрат. На відміну від JPEG, PNG не втрачає даних під час стиснення, що зберігає початкову якість файлу.

- Підтримка прозорості. Формат дозволяє використовувати альфа-канали для створення зображень із прозорим фоном, що важливо для інтерфейсів та вебконтенту.

Недоліки формату PNG:

- Великий розмір файлів. Через збереження усіх піксельних даних без втрат PNG-файли зазвичай важчі за JPEG або GIF, що впливає на швидкість завантаження.
- Обмежена придатність для поліграфії. PNG не підтримує колірну модель CMYK, яка необхідна для професійного друку, тому не є ідеальним для поліграфічних цілей.
- Низька ефективність у веб-оптимізації. Через великий розмір PNG-файли можуть повільно завантажуватись на сторінках, особливо при використанні багатьох зображень.

3.4. Формат файлу PDF

Загальний опис

PDF (Portable Document Format) — це універсальний формат електронних документів, розроблений компанією Adobe для збереження й обміну інформацією незалежно від програмного чи апаратного забезпечення. Основна ідея PDF — гарантувати, що документ виглядатиме однаково на будь-якому пристрої або операційній системі, незалежно від того, де його відкривають. [12]

На сьогодні PDF є відкритим стандартом, затвердженим Міжнародною організацією зі стандартизації (ISO). Формат підтримує не лише текст та зображення, а й інтерактивні елементи: гіперпосилання, кнопки, заповнювані форми, мультимедіа (аудіо, відео), а також можливість накладання електронного підпису.

Історія формату бере початок із 1991 року, коли один із засновників Adobe, Джон Ворнок, започаткував ініціативу «Проєкт Camelot». Її метою було створення універсального формату для зберігання та друку документів із будь-

якого застосунку, незалежно від типу пристрою. У 1992 році концепція Camelot трансформувалася у формат PDF, який з часом став одним із найнадійніших стандартів у сфері обміну документацією.

Переваги формату PDF:

- Універсальність. Документи PDF відображаються однаково на будь-якому пристрої, незалежно від операційної системи чи встановленого ПЗ.
- Підтримка складного вмісту. Формат дозволяє вбудовувати гіперпосилання, інтерактивні поля, скрипти, відео та звук.
- Безпека та підпис. Підтримує електронні підписи, захист паролем, обмеження на редагування чи копіювання.
- Довговічність. Формат широко використовується в державних установах, освіті, бізнесі та архівах.
- Сумісність. Відкривається в більшості переглядачів, зокрема Adobe Acrobat Reader, браузерах, мобільних додатках тощо.

Недоліки формату PDF:

- Редагування. Вміст PDF-документів важче редагувати порівняно з офісними форматами (наприклад, DOCX), без спеціального ПЗ.
- Об’єм. Файли з великою кількістю зображень або вбудованих шрифтів можуть мати значний розмір.
- Мало придатний для динамічних даних. Хоча PDF може містити інтерактивні елементи, він менш гнучкий у порівнянні з HTML або JSON, коли йдеться про живе оновлення інформації.

3.5. Формат файлу SVG

Загальний опис

SVG (Scalable Vector Graphics — масштабована векторна графіка) — це формат зберігання зображень, який базується на векторній моделі. На відміну від

растрових форматів (як-от JPEG чи PNG), де зображення формуються з пікселів, SVG описує графіку за допомогою математичних формул — ліній, кривих та фігур. Завдяки цьому SVG-зображення можна масштабувати до будь-якого розміру без втрати якості, що робить їх особливо зручними для логотипів, іконок, схем і діаграм. [15]

SVG-файли мають текстову структуру на основі мови розмітки XML. Це означає, що їхній вміст легко читається як людиною, так і машиною: зображення можна редагувати за допомогою текстових редакторів, а пошукові системи можуть індексувати текст, що міститься в них. Така особливість також полегшує доступність контенту для користувачів з вадами зору, які користуються програмами зчитування екрану.

Походження та розвиток формату

Формат SVG був запропонований у кінці 1990-х років під егідою Консорціуму Всесвітнього павутиння (W3C) як відкритий стандарт для опису векторної графіки в Інтернеті. Після розгляду кількох альтернатив саме SVG було обрано як основний формат. Хоча перші роки SVG залишався малопоширеним, починаючи з 2017 року він почав активно використовуватися завдяки зростанню підтримки з боку сучасних браузерів. Наразі SVG став стандартом для 2D-графіки на вебсайтах та у фронтенд-розробці.

Переваги формату SVG:

- Масштабованість без втрати якості. Завдяки векторній природі зображення зберігають чіткість незалежно від розміру або роздільної здатності пристрою.
- Малий розмір файлів. Просте SVG-зображення часто займає менше пам’яті, ніж відповідне растрове, оскільки не містить масивів пікселів.
- Текстова структура. Елементи SVG можна редагувати напряму в коді, а також читати за допомогою програм для доступності та індексувати в пошукових системах.

- Гнучкість і сумісність із HTML/CSS. SVG-файли легко вбудовуються в веб-сторінки, можуть стилізуватися та анімуватися за допомогою CSS або JavaScript.

Недоліки формату SVG:

- Непридатність для фотореалістичних зображень. SVG не підходить для зберігання детальних фотографій або складної растрової графіки — для цього краще використовувати формати JPEG або PNG.
- Обмежена підтримка в застарілих браузерах. Деякі старі версії браузерів (наприклад, Internet Explorer 8 і раніше) не підтримують повноцінне відображення SVG.
- Складність для новачків. Розуміння та редагування SVG-коду може бути складним для користувачів без досвіду роботи з XML або веб-розробкою.

3.6. Формат файлу FBX

Загальний опис

FBX (FilmBox) — це один із найпопулярніших форматів для обміну тривимірними моделями, який широко застосовується у 3D-моделюванні, анімації, візуалізації та розробці ігор. Спочатку він був розроблений компанією Kaydara для застосунку MotionBuilder, а з 2006 року належить компанії Autodesk, яка активно підтримує його розвиток як стандарт де-факто в індустрії цифрового контенту.[6]

Формат FBX підтримується більшістю 3D-редакторів, таких як Autodesk Maya, 3ds Max, Unity, Unreal Engine, Blender тощо. Він доступний у двох версіях — у двійковому форматі (компактному та оптимізованому для збереження та передачі) і в текстовому (ASCII) форматі, який має ієрархічну, дерево-подібну структуру з чіткими тегами та ідентифікаторами. У текстовій версії кожен вузол

має назву, набір властивостей (наприклад, числа, рядки) та підвузли, що створюють вкладену структуру.

FBX створено для забезпечення максимальної сумісності між різними системами цифрового моделювання, включаючи підтримку не лише геометрії, але й матеріалів, текстур, скелетної анімації, освітлення, камер і навіть часових ліній.

Технічна специфіка

FBX є закритим (пропрієтарним) форматом, специфікація його двійкової структури не є відкритою. Проте компанія Autodesk надає офіційний SDK (FBX SDK на C++), який дозволяє розробникам читати, створювати та конвертувати FBX-файли. Також у таких інструментах, як Blender, реалізовано імпорт/експорт FBX за допомогою Python-скриптів, без прямого використання офіційного SDK.

Переваги формату FBX

- Широка підтримка у професійному середовищі. FBX є стандартом обміну між найпопулярнішими 3D-редакторами й рушіями.
- Багатство інформації. Підтримує геометрію, матеріали, UV-розгортки, анімації, кістки, освітлення та інші атрибути сцени.
- Експорт у 3D-ігри та віртуальне середовище. Завдяки підтримці у таких рушіях як Unity і Unreal, формат FBX ідеально підходить для інтеграції 3D-контенту в інтерактивні середовища.
- Оптимізація. Двійковий формат забезпечує ефективне зберігання без надмірного об’єму, зберігаючи всі важливі дані.

Недоліки формату FBX

- Закритість формату. Технічна специфікація двійкової структури не є відкритою, що ускладнює пряме використання без SDK.
- Несумісність між версіями. Різні програми іноді по-різному реалізують підтримку FBX, що може призводити до помилок при обміні між ними.

- Складність розбору ASCII-структури. Хоча текстова версія доступна для перегляду, вона має складну ієрархічну структуру, що ускладнює її обробку без відповідного досвіду.
- Залежність від Autodesk. Для повноцінної роботи часто необхідне використання SDK або програмного забезпечення, сумісного з Autodesk.

3.7. Вимоги до програмного та технічного забезпечення

Для забезпечення коректної роботи програмного продукту необхідно дотримання певних вимог до програмного та апаратного середовища, а саме мінімальні та рекомендовані характеристики, які гарантують стабільну роботу системи генерації, візуалізації, проходження та експорту лабіринтів.

Програмне забезпечення

Операційна система:

- Мінімально: Windows 10 (64-bit)
- Рекомендовано: Windows 11 (64-bit)

Необхідні компоненти:

- .NET Framework 4.8 або вище (встановлюється автоматично під час інсталяції)

Середовище розробки (для модифікацій):

- MVS 2022 або новіша
- Мова програмування: C#
- Підтримка NuGet-бібліотек (за потреби: Newtonsoft.Json, System.Drawing.Common тощо)

Зовнішні засоби (опційно):

- Blender (для перегляду 3D-файлів у форматі FBX)
- Веб-браузер або PDF-рідер (для перегляду PDF/SVG)

Технічне забезпечення

Мінімальні системні вимоги:

- Процесор: 2-ядерний 64-бітний (наприклад, Intel Core i3 або еквівалент AMD) із тактовою частотою 2 ГГц
- Оперативна пам’ять: 4 ГБ
- Відеокарта: інтегрована з підтримкою OpenGL 3.0+
- Вільне місце на диску: 500 МБ
- Дисплей: роздільна здатність не менше 1366×768

Рекомендовані системні вимоги:

- Процесор: Intel Core i5 / AMD Ryzen 5 і вище
- Оперативна пам’ять: 8 ГБ і більше
- Відеокарта: дискретна з підтримкою 3D-графіки (для FBX)
- Вільне місце на диску: 1 ГБ+
- Дисплей: Full HD (1920×1080) або вища

Додаткові умови

- Права користувача повинні дозволяти запуск програм і збереження файлів у вибрані директорії.
- Для повноцінного використання функцій експорту потрібно мати доступ до директорій запису.

3.8. Висновки до розділу

У цьому розділі було охарактеризовано інструменти та засоби, використані під час розробки програмного забезпечення для генерації лабіринтів. Основним середовищем розробки виступила **MVS 2022**, що забезпечила високий рівень інтеграції між мовою програмування, графічним інтерфейсом, бібліотеками, компонентами та такими елементами керування, як: кнопки (Button), написи (Label), текстові поля (TextBox), групи (GroupBox) та інші. Завдяки широкому функціоналу та підтримці .NET Framework, це середовище надало всі необхідні засоби для створення стабільної та зручної у використанні програми.

Описано використання мови програмування **C#**, яка продемонструвала свою ефективність для реалізації алгоритмів генерації лабіринтів, опрацювання графічних структур, реалізації візуалізації та роботи з файлами різного формату. Було застосовано низку вбудованих бібліотек для обробки графіки, візуалізації та експорту результатів у різні формати.

Визначено оптимальну платформу — **Windows 10 (64-bit)** або новішу версію, мінімальну конфігурацію ПК, необхідну для стабільної роботи, а також перелік програмних компонентів, таких як .NET Framework версії 4.8 або вище.

4. ТЕХНОЛОГІЧНИЙ РОЗДІЛ

4.1. Керівництво користувача

Запустивши програму користувач отримує доступ до всіх можливих систем ПЗ, але спочатку лише до генерації лабіринтів, бо інші системи, як експорт та проходження будуть доступні, лише після створення будь-якого лабіринту.

Для створення ПЗ потрібно було виконати такий перелік дій:

- створити дизайн програми;
- створити функціональну частину програму.

Додаток має таку структуру:

- Головне вікно
  - Поле відображення лабіринту
  - Налаштування лабіринтів
  - Налаштування експорту
- Вікно проходження

У програмному коді, представленому у додатках А-К, використано такі елементи та функції:

Елементи:

- *Button* – кнопка
- *Textbox* – текстове поле
- *Label* – напис
- *GroupBox* – вікно групування об’єктів
- *NumericUpDown* – лічильник
- *ComboBox* – випадаючий список
- *CheckBox* – прапорець
- *PictureBox* – поле для відображення зображення

Функції:

- *Random* – створює змінну типу «Random»

- *Next* – випадковим чином створює ціле число з переліку
- *Show* – відкриває відповідне вікно
- *Close* – закриває відповідне вікно
- *display_maze* – відображає згенерований лабіринт
- *DrawCell* – малює частину лабіринту
- *Prima* – створює лабіринт методом Прима
- *Kruskal* – створює лабіринт методом Крускала
- *DFS* – створює лабіринт методом пошуку в глиб
- *Eller* – створює лабіринт методом Еллера
- *RecursiveDivision* – створює лабіринт рекурсивним методом
- *ExportMazeToPdf* – експорт лабіринту у формат PDF
- *ExportMazeToPng* – експорт лабіринту у формат PNG
- *ExportMazeToSvg* – експорт лабіринту у формат SVG
- *ExportMazeToFbx* – експорт лабіринту у формат FBX
- *AddWall* – допоміжна функція для генерації файлу FBX, для побудови стін лабіринту

У програмному коді використано такі функції бібліотеки *MVS:*

- *System*;
- *System.IO*;
- *System.Text*;
- *System.Collections.Generic;*
- *System.Drawing;*
- *System.Threading.Tasks;*
- *System.Windows.Forms;*
- *System.Threading;*
- *Aspose.ThreeD;*
- *Aspose.ThreeD.Utilities;*
- *Aspose.ThreeD.Entities;*

4.2. Встановлення програми

Для встановлення програмного забезпечення необхідно завантажити архів з інсталяційними файлами за посиланням на Google Диск [17]. Після завантаження слід розпакувати архів та запустити файл setup.exe.

У процесі встановлення система може запропонувати завантажити додаткові компоненти, необхідні для коректної роботи програми (наприклад, .NET Framework, бібліотеки візуалізації тощо). Рекомендується погодитися з усіма запитами інсталятора.

Після завершення встановлення для запуску програми потрібно знову виконати setup.exe.

4.3. Опис інтерфейсу програми

При запуску програми відкривається вікно «Maze Generator», де користувач може отримати доступ до всіх елементів системи. Вигляд вікна «Maze Generator» представлений на рис. 4.1

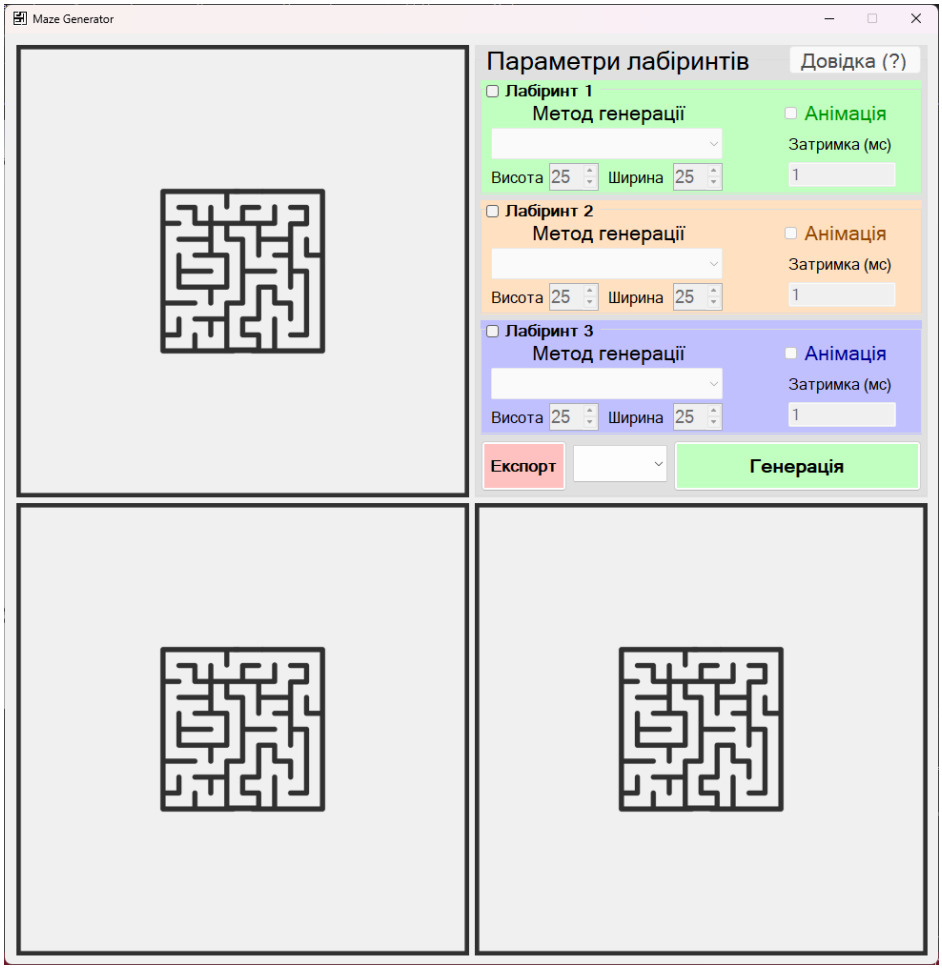

Рис. 4.1 Maze Generator

На рис. 4.2 зображено Вікно з згенерованими лабіринтами методами Прима, пошуку в глиб та Еллера

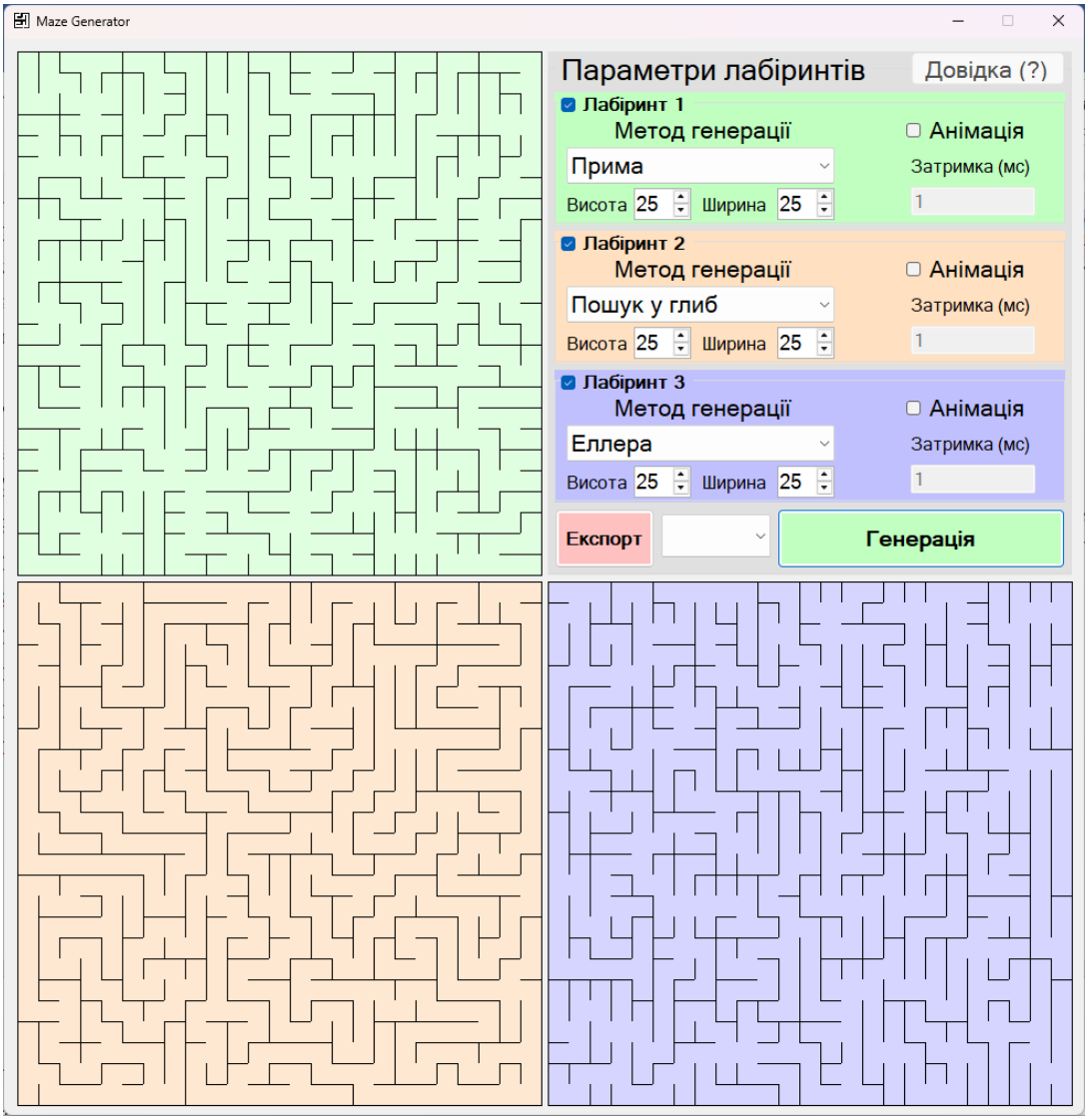

Рис. 4.2 Згенеровані лабіринти

Вікно Maze Explorer, на рис. 4.3, демонструє вибраний лабіринт для проходження в даному випадку лабіринт створений методом пошук в глиб, де зелений круг є позицією користувача, а блакитний квадрат є точкою фінішу.

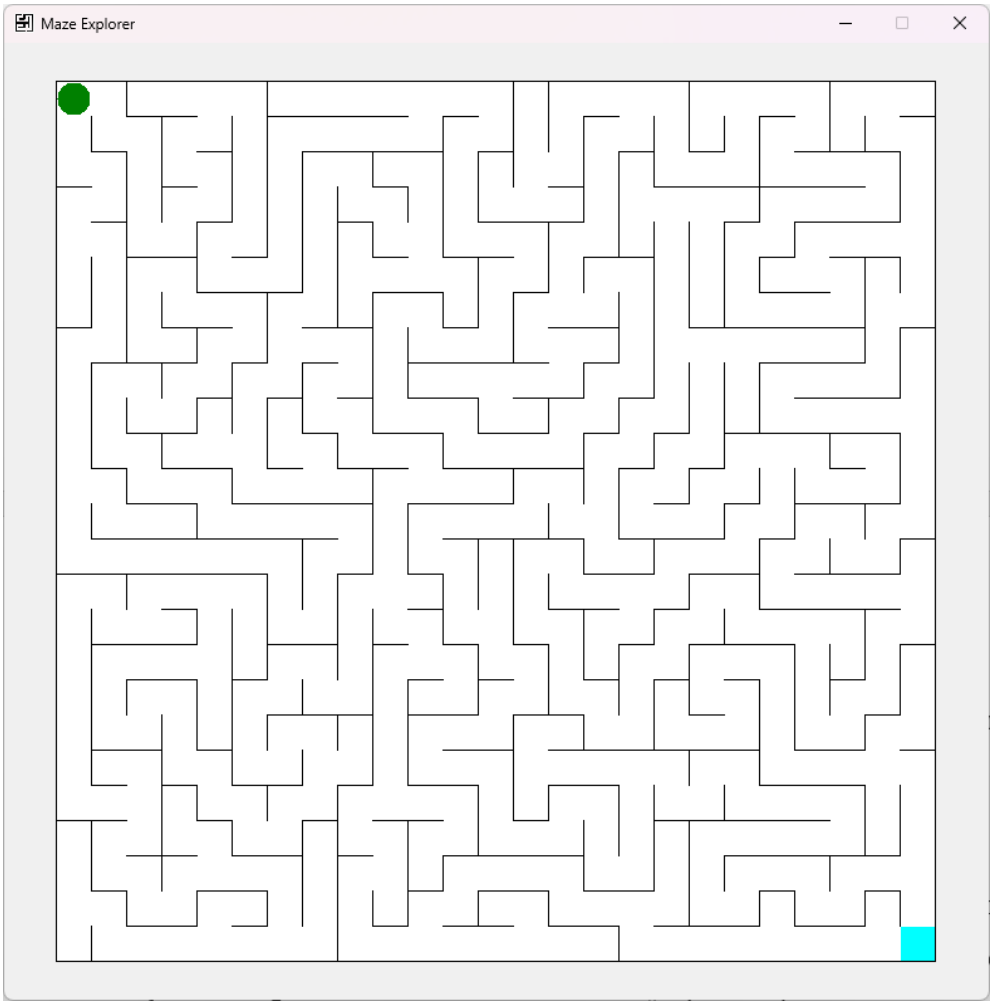

Рис. 4.3 Maze Explorer

На рис. 4.4, 4.5, 4.6 представлені результати експорту лабіринту, зокрема:

- збережені файли в файловій системі
- демонстрація експортованого файлу у форматі PNG
- демонстрація експортованого файлу у форматі FBX, у програмі Blender

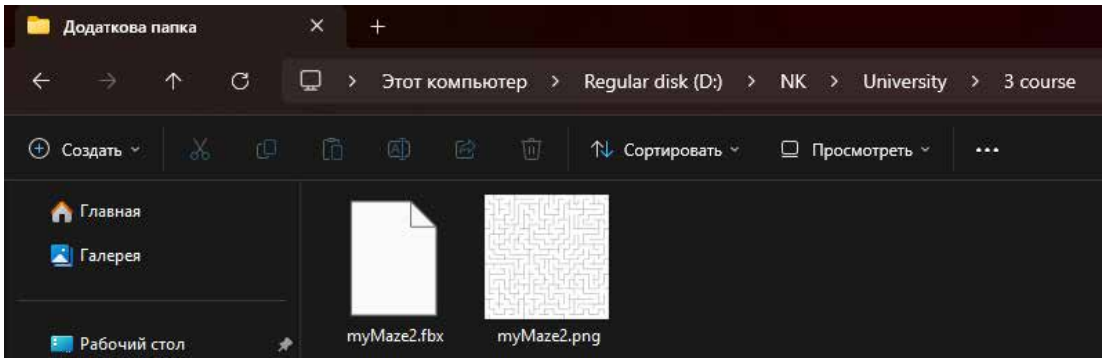

Рис. 4.4 Збережені файли

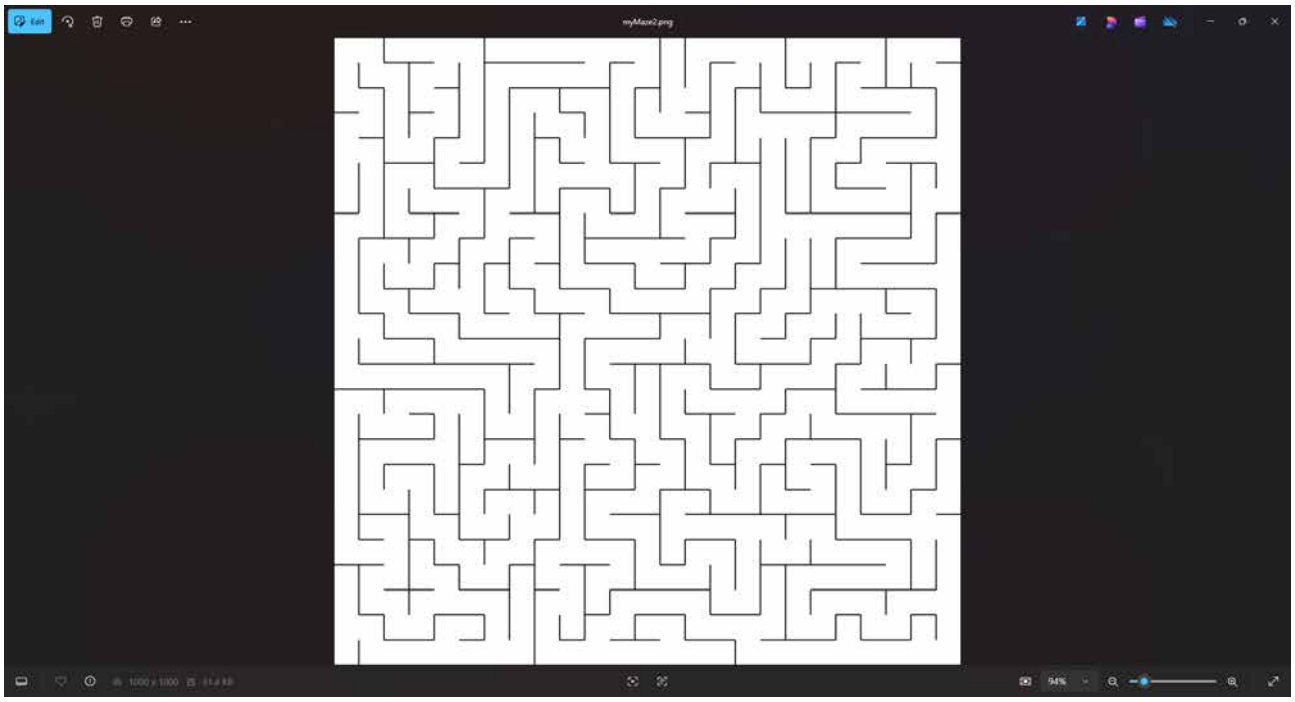

Рис. 4.5 Лабіринт формату PDF

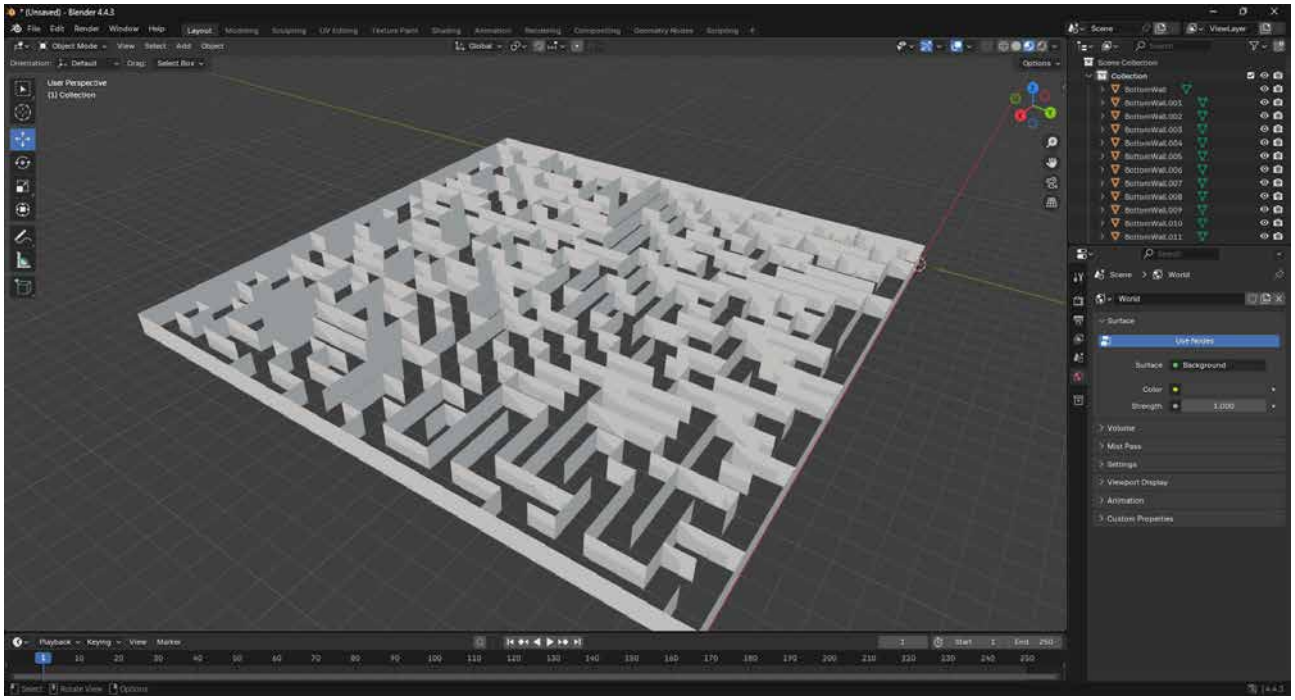

Рис. 4.6 Лабіринт формату FBX

4.4. Тестування та аналіз результатів роботи

Для забезпечення надійності, стабільності та функціональності програмного продукту було проведено повне тестування розробленої системи генерації лабіринтів. Тестування охоплювало як функціональні компоненти (генерація, візуалізація, проходження, експорт), так і нефункціональні характеристики.

1. Функціональне тестування

Під час функціонального тестування було перевірено правильність роботи кожного з реалізованих алгоритмів генерації лабіринтів:

- **Прима** – система коректно генерує остовні дерева без циклів;
- **Крускала** – побудова мінімального остовного дерева відбувається без помилок;
- **DFS (глибина-перший)** – створюються довгі вузькі проходи, що відповідає природі алгоритму;
- **Еллера** – забезпечується коректне рядкове заповнення структури лабіринту;

- **Рекурсивне ділення** – простір розділяється на підобласті відповідно до очікуваної логіки.

Кожен алгоритм протестовано з різними вхідними параметрами (розміри лабіринту, колір, наявність анімації). У всіх випадках система повертала правильний результат без помилок або зависань.

2. Візуалізація

Блок візуалізації лабіринту дозволив наочно переконатися у правильності з'єднання клітинок, наявності або відсутності стін та узгодженості кольорових параметрів.

Особливу увагу приділено правильності формування зображення при зміні розміру лабіринту. Перевірено коректність відображення як для невеликих (5×5), так і для великих (50×50) лабіринтів.

3. Проходження лабіринту

Система проходження дозволяє користувачеві керувати положенням гравця на полі лабіринту. Було протестовано правильність логіки руху, визначення стін та меж поля. Зелений маркер (позиція гравця) переміщується тільки у дозволених напрямках, а досягнення фінішної точки коректно фіксується системою.

4. Експорт

Було здійснено експорт лабіринтів у формати PNG, PDF, SVG та FBX. Кожен формат відкривався у відповідному програмному забезпеченні без спотворення структури. Особливо ретельно перевірено експорт у 3D (FBX), який був протестований у середовищі Blender — відображення стін, висоти та загальної геометрії відповідає згенерованому лабіринту.

5. Швидкодія та стабільність

У ході тестування програма демонструвала стабільну роботу навіть при високих навантаженнях — зокрема, при генерації великих лабіринтів (до 100×100 клітинок). Час очікування генерації та візуалізації не перевищував кількох секунд, що є прийнятним показником. Також не виявлено збоїв, зависань або витоків пам'яті.

6. Висновки за результатами тестування

- Усі основні компоненти програми функціонують стабільно та відповідають заявленим вимогам.
- Інтерфейс забезпечує інтуїтивну взаємодію з користувачем.
- Програма готова до використання як у навчальному, так і в демонстраційному середовищі.
- Результати експорту підтверджують відповідність створених файлів очікуваній структурі.

4.5. Висновки до розділу

У цьому розділі було детально описано технологічні аспекти взаємодії користувача з програмним забезпеченням. Розглянуто покрокову логіку роботи з інтерфейсом — від генерації лабіринтів до їх проходження та експорту результатів у популярні графічні формати. Надано опис структури додатку, яка включає головне вікно, блоки налаштувань, область візуалізації та функціонал керування.

Окрему увагу приділено інтерфейсу користувача. Продемонстровано, що інтерфейс побудовано з використанням стандартних елементів Windows Forms (Button, TextBox, ComboBox, PictureBox тощо), що забезпечує зручність у роботі навіть для недосвідченого користувача. Передбачено можливість вибору алгоритму генерації, задання параметрів, перегляду та експорту лабіринтів у формати PNG, PDF, SVG і FBX.

Додатково було розглянуто процес встановлення програми. Описано процедуру завантаження, запуску інсталятора та встановлення необхідних компонентів, що дозволяє користувачеві легко запустити застосунок на будь-якому сумісному комп’ютері без потреби в додатковому налаштуванні.

Проведено тестування всіх функціональних елементів системи. Перевірено стабільність генерації, правильність візуалізації, точність експорту та

працездатність режиму проходження. Усі функції працюють коректно, критичних помилок не виявлено.

Таким чином, програмне забезпечення є повністю готовим до використання. Зручний інтерфейс, інтуїтивна логіка взаємодії та широкий набір функцій роблять його придатним як для навчання, так і для практичного застосування у задачах, пов’язаних із генерацією та візуалізацією графових структур.

ВИСНОВКИ

У процесі виконання дипломного проєкту було реалізовано повноцінну програмну систему для автоматизованої генерації, візуалізації, проходження та експорту лабіринтів із використанням сучасних алгоритмів теорії графів. Робота охопила повний цикл розробки програмного забезпечення — від аналізу предметної області до тестування готового продукту.

На першому етапі дослідження було проаналізовано історичні та теоретичні аспекти, пов’язані з лабіринтами. Розглянуто приклади застосування лабіринтів у різних сферах: в комп’ютерних іграх, у задачах автономної навігації роботів, а також у навчанні алгоритмів штучного інтелекту. Теоретичне підґрунтя базувалося на поняттях графів, які дозволяють ефективно моделювати структуру лабіринтів та реалізовувати різні сценарії проходження.

Під час проєктування системи було визначено функціональні й нефункціональні вимоги, сформульовано сценарії використання та побудовано UML-діаграми: діаграму прецедентів, діаграму послідовності, діаграму активності, діаграму класів і діаграму пакетів. Це дозволило структурувати архітектуру програми, забезпечити масштабованість і логічну організованість коду.

Програмне забезпечення було реалізовано мовою програмування C# у середовищі Microsoft Visual Studio 2022, з використанням платформи .NET Framework. Основними реалізованими алгоритмами генерації стали:

- **Прима** — для створення розгалужених деревоподібних лабіринтів;
- **Крускала** — як приклад остовного підходу;
- **Пошук у глибину (DFS)** та інші — для створення довгих і вузьких проходів;

Графічний інтерфейс реалізовано на базі Windows Forms, з можливістю налаштування параметрів генерації і запуску режиму проходження лабіринту.

Було реалізовано інтерактивне переміщення користувача по лабіринту з виявленням досягнення фінішної точки.

Одним із важливих компонентів стала функція експорту результатів генерації у різні формати:

- **PNG** — для збереження растрових зображень;
- **PDF** — для створення документів, готових до друку;
- **SVG** — для векторної графіки, яку можна масштабувати без втрати якості;
- **FBX** — для експорту тривимірної моделі лабіринту, що відкриває можливості використання в 3D-візуалізації або ігрових рушіях.

Розроблене програмне забезпечення не вимагає підключення до зовнішньої бази даних або складних інсталяцій — усі дані зберігаються локально в обраних користувачем форматах. Це робить систему універсальною, портативною та зручною для використання в різних умовах: у навчальних аудиторіях, для домашнього експериментування або як демонстраційний інструмент для вивчення алгоритмів.

За результатами тестування програмний продукт продемонстрував стабільну роботу, швидке виконання генерації навіть при великих розмірах лабіринтів. Інтерфейс виявився зручним і зрозумілим для користувача, а гнучка система налаштувань — корисною для різних сценаріїв застосування.

Результатом виконаної роботи є функціональний, стабільний та гнучкий програмний продукт, що вирішує задачу генерації лабіринтів з можливістю інтерактивної взаємодії та експорту у кількох популярних форматах. Система може бути використана як інструмент у навчальному процесі, для вивчення алгоритмів, у демонстраційних середовищах або як база для подальшої розробки, наприклад — інтеграції у гру, 3D-проєкт чи роботизовану платформу.

Крім того, потенціал системи відкриває можливості для міждисциплінарного застосування: у медицині — для моделювання складних топологій судин чи симуляцій ендоскопічних процедур, що перегукується з

дослідженням [3], де алгоритмічні підходи розглядаються як засіб навігації в складних клінічних станах. У нейронауках — для вивчення механізмів просторової орієнтації та прийняття рішень, зокрема через застосування штучних агентів із сітчастими уявленнями простору, як це показано в роботі DeepMind [2]. У фізиці — для досліджень дифузії в середовищах із перешкодами або випадковими потенціалами, подібно до моделі «магнітного лабіринту», запропонованої у праці [7]. У архітектурі та урбаністиці — для планування евакуаційних маршрутів, оптимізації простору або реалізації генеративного дизайну, що використовує ті самі принципи алгоритмічного структурування [8]. А також у психології — для проведення експериментів із просторовою пам’яттю та поведінковими реакціями в умовах вибору, як у класичній системі лабіринтів Гебба–Вільямса [9].

Таким чином, реалізоване програмне забезпечення не лише відповідає поставленим завданням, але й має високий потенціал подальшого розвитку та застосування у різних наукових і прикладних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Comparative Analysis of Maze Generation Algorithms / J. B. Laga, C. Bouville [Електронний ресурс] // HAL Open Science. – Режим доступу: https://uca.hal.science/hal-03174952v1/document – Дата звернення: 15.05.2025.

2. Banino A., Barry C., Uria B., Blundell C., Lillicrap T., Mirowski P. та ін. Vector-based navigation using grid-like representations in artificial agents [Електронний ресурс] // Nature. – 2018. – Vol. 557, № 7705. – С. 429–433. – Режим доступу: https://www.wired.com/story/deepmind-newest-network-mimics-the-gps-cells-in-your-brain. – Дата звернення: 15.05.2025.

3. Beneki E., Dimitriadis K., Tsioufis K., Aggeli K. Can We Use an Algorithm as an "Ariadne's Thread" to Escape the Maze of Mitral Regurgitation Phenotype? [Електронний ресурс] // Journal of the American Society of Echocardiography. – 2024. – Vol. 37, № 3. – С. 373. – Режим доступу: https://pubmed.ncbi.nlm.nih.gov/37839618. – Дата звернення: 15.05.2025.

4. C# [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/C_Sharp. – Дата звернення: 15.05.2025.

5. Depth-first search – теоретичне обґрунтування [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Depth-first_search. – Дата звернення: 15.05.2025.

6. FBX – формат обміну 3D-моделями [Електронний ресурс]. – Режим доступу: https://docs.fileformat.com/uk/3d/fbx. – Дата звернення: 15.05.2025.

7. Flicker F., Singh S. The Magnetic Maze: A System With Tunable Scale Invariance [Електронний ресурс] // arXiv preprint arXiv:2409.02176. – 2024. – Режим доступу: https://arxiv.org/abs/2409.02176. – Дата звернення: 15.05.2025.

8. Generative design [Електронний ресурс] // Wikipedia – The Free Encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Generative_design. – Дата звернення: 15.05.2025.

9. Hebb–Williams maze [Електронний ресурс] // Wikipedia – The Free Encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Hebb%E2%80%93Williams_maze. – Дата звернення: 15.05.2025.

10. Microsoft Visual Studio [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio. – Дата звернення: 15.05.2025.

11. Microsoft Visual Studio 2022 [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#2022. – Дата звернення: 15.05.2025.

12. PDF – що таке формат PDF [Електронний ресурс]. – Режим доступу: https://www.adobe.com/ua/acrobat/about-adobe-pdf.html. – Дата звернення: 15.05.2025.

13. PNG-файли – опис формату [Електронний ресурс]. – Режим доступу: https://www.adobe.com/ua/creativecloud/file-types/image/raster/png-file.html. – Дата звернення: 15.05.2025.

14. S. Y. Verma, S. Panigrahi. A Survey on Maze Solving Algorithms for Robotics [Електронний ресурс] // IPSI Transactions on Internet Research. – 2019. – Режим доступу: https://ipsitransactions.org/journals/papers/tir/2019jan/p5.pdf – Дата звернення: 15.05.2025.

15. SVG – масштабована векторна графіка [Електронний ресурс]. – Режим доступу: https://www.adobe.com/ua/creativecloud/file-types/image/vector/svg-file.html. – Дата звернення: 15.05.2025.

16. Алгоритм створення лабіринту [Електронний ресурс] – Вікіпедія. Вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/Алгоритм_створення_лабіринту – Дата звернення: 15.05.2025.

17. Архів інсталяційних файлів до дипломного проекту [Електронний ресурс] – Google Drive. – Режим доступу:

https://drive.google.com/drive/folders/1UOFJrMw3Ku7IB37AiWhfe17JuMYrYA4r – Дата звернення: 15.05.2025.

18. Глибина-перший пошук (DFS) для графів [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph. – Дата звернення: 15.05.2025.

19. Діаграма пакетів [Електронний ресурс] – Вікіпедія. Вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/Діаграма_пакетів. – Дата звернення: 15.05.2025.

20. Дідковська М. В. Тестування програмного забезпечення. Лекція 5 [Електронний ресурс] / НТУУ «КПІ ім. Ігоря Сікорського» – Режим доступу: http://mmsa.kpi.ua/sites/default/files/disciplines/Розробка%20і%20тестування%20програм/didkovska_m_v_testing_lecture_5.pdf – Дата звернення: 15.05.2025.

21. Міф про Тесея і Мінотавра [Електронний ресурс] – Довідка.biz.ua. – Режим доступу: https://dovidka.biz.ua/mif-pro-teseya-i-minotavra/ – Дата звернення: 15.05.2025.

22. Огляд алгоритмів генерації лабіринтів / Електронний ресурс НТУУ "КПІ" – Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/fb0a4251-74d9-470b-88da-71abb4e85f93/content. – Дата звернення: 15.05.2025.

23. Пурхало М. О. Програмне забезпечення системи побудови лабіринтів на основі методів теорії графів. Збірник наукових праць за матеріалами VII Всеукраїнської науково-практичної конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем 2025», м. Київ, 24 квітня. 2025 р. НУБІП України. Київ, 2025.

24. Теорія графів [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Теорія_графів. – Дата звернення: 15.05.2025.

25. Як будувати UML-діаграми [Електронний ресурс]. – Режим доступу: https://dou.ua/forums/topic/40575. – Дата звернення: 15.05.2025.

ДОДАТКИ

ДОДАТОК А

Код головного вікна програми

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static Diploma.Prima_algorithm;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;
using System.IO;
using PdfSharp.Pdf;
using PdfSharp.Drawing;
using System.Threading;
using System.Diagnostics;
namespace Diploma{
   public partial class Form1 : Form{
      public static Form1 Instance { get; private set; }
      private CancellationTokenSource cts1;
      private CancellationTokenSource cts2;
      private CancellationTokenSource cts3;
      public Form1(){
         InitializeComponent();
         Instance = this;}
      private static Random globalRand = new Random();
      private List<(int x, int y)> walls = new List<(int x, int y)>();
      Cell[,] maze1, maze2, maze3;
      private void Form1_FormClosed(object sender, FormClosedEventArgs e){
         try {Environment.Exit(0);}
         catch { }}
      public class Cell{public bool Top { get; set; }
         public bool Right { get; set; }
public bool Bottom { get; set; } public bool Left { get; set; }
         public bool Visited { get; set; }
         public Cell(){Top = true; Right = true;
            Bottom = true; Left = true; Visited = false;}}
      public void display_maze(Cell[,] cells, Bitmap bitmap, int picBox, int rgb_r = 255, int rgb_g = 255, int rgb_b = 255){
         int canvasWidth = 476; int canvasHeight = 476;
         int w = cells.GetLength(0); int h = cells.GetLength(1);
         // Визначаємо PictureBox за його номером
         PictureBox selectedPictureBox = null;
         if (picBox == 1) selectedPictureBox = pictureBox1;
else if (picBox == 2) selectedPictureBox = pictureBox2;
else if (picBox == 3) selectedPictureBox = pictureBox3;
         if (selectedPictureBox == null) return;
int cellSize = Math.Min(canvasWidth / w, canvasHeight / h);
         using (Graphics g = Graphics.FromImage(bitmap)){
            g.Clear(Color.White); // Фон білий
            SolidBrush brush = new SolidBrush(Color.FromArgb(rgb_r, rgb_g, rgb_b));
            Pen blackPen = new Pen(Color.Black, 1);
            // Малюємо комірки
            for (int i = 0; i < w; i++){ for (int j = 0; j < h; j++){
                  int xPos = i * cellSize; int yPos = j * cellSize;
// Якщо у клітинки всі чотири стінки - зафарбовуємо сірою
                  if (cells[i, j].Top && cells[i, j].Left &&
                     cells[i, j].Bottom && cells[i, j].Right){
                     g.FillRectangle(Brushes.Gray, xPos, yPos, cellSize, cellSize);}
                  else{
                     // Малюємо фон комірки
g.FillRectangle(brush, xPos, yPos, cellSize, cellSize);
                     // Домальовуємо стіни
                     if (cells[i, j].Top)
g.DrawLine(blackPen, xPos, yPos, xPos + cellSize, yPos);
                     if (cells[i, j].Left)
g.DrawLine(blackPen, xPos, yPos, xPos, yPos + cellSize);
                     if (cells[i, j].Bottom)
                        g.DrawLine(blackPen, xPos, yPos + cellSize, xPos + cellSize, yPos + cellSize);
                     if (cells[i, j].Right)
                        g.DrawLine(blackPen, xPos + cellSize, yPos, xPos + cellSize, yPos + cellSize);}}}}
         // Встановлюємо `Bitmap` у відповідний PictureBox
         selectedPictureBox.Image = bitmap;
         selectedPictureBox.SizeMode = PictureBoxSizeMode.AutoSize;}
      public void DrawCell(Bitmap bitmap, int picBox, Cell[,] cells, int cx, int cy, int rgb_r = 255, int rgb_g = 255, int rgb_b = 255){int canvasWidth = bitmap.Width;
         int canvasHeight = bitmap.Height;
         int w = cells.GetLength(0); int h = cells.GetLength(1);
         using (Graphics g = Graphics.FromImage(bitmap)){
            SolidBrush brush = new SolidBrush(Color.FromArgb(rgb_r, rgb_g, rgb_b));
            Pen blackPen = new Pen(Color.Black, 1);
int cellSize = Math.Min(canvasWidth / w, canvasHeight / h);
// Координати верхнього лівого кута комірки (cx, cy)
            int xPos = cx * cellSize; int yPos = cy * cellSize;
            if (cells[cx, cy].Top && cells[cx, cy].Left &&
               cells[cx, cy].Bottom && cells[cx, cy].Right){
// Заповнюємо сірим кольором, якщо вона має всі стіни
               g.FillRectangle(Brushes.Gray,xPos,yPos,cellSize, cellSize);}
            else{
g.FillRectangle(brush, xPos, yPos, cellSize, cellSize);
               if (cells[cx, cy].Top)
g.DrawLine(blackPen, xPos, yPos, xPos + cellSize, yPos);
               if (cells[cx, cy].Left)
g.DrawLine(blackPen, xPos, yPos, xPos, yPos + cellSize);
               if (cells[cx, cy].Bottom)
                  g.DrawLine(blackPen, xPos, yPos + cellSize, xPos + cellSize, yPos + cellSize);
               if (cells[cx, cy].Right)
                  g.DrawLine(blackPen, xPos + cellSize, yPos, xPos + cellSize, yPos + cellSize);}}
         // Відображення у відповідному PictureBox
```

```
        PictureBox selectedPictureBox = null;
        if (picBox == 1) selectedPictureBox = pictureBox1;
else if (picBox == 2) selectedPictureBox = pictureBox2;
else if (picBox == 3) selectedPictureBox = pictureBox3;
        if (selectedPictureBox != null){
           selectedPictureBox.Image = bitmap;
           selectedPictureBox.SizeMode =
PictureBoxSizeMode.AutoSize;}}
     private async void button3_Click(object sender,
EventArgs e){
        Prima_algorithm algoPrima = new Prima_algorithm();
        Kruskal_algorithm algoKruskal = new
Kruskal_algorithm();
        DFS_algoritm algoDFS = new DFS_algoritm();
        Eller_algorithm algoEller = new Eller_algorithm();
        RecursiveDivision_algorithm algoRD = new
RecursiveDivision_algorithm();
        Task<Cell[,]> t1 = null; Task<Cell[,]> t2 = null;
        Task<Cell[,]> t3 = null;
        if (!checkBox2.Checked && !checkBox5.Checked &&
!checkBox7.Checked){
MessageBox.Show("Не вибран лабіринт для генерації");}
        if (checkBox2.Checked){
           string method = comboBox1.Text;
if (method != "" && method != "Метод генерації"){
              if (cts1 != null) { cts1.Cancel(); }
              cts1 = new CancellationTokenSource();
              maze1 = null; await Task.Delay(100);
              int delay = Convert.ToInt32(textBox1.Text);
int w = Convert.ToInt32(numericUpDown1.Value);
int h = Convert.ToInt32(numericUpDown2.Value);
              int flag = checkBox1.Checked ? 1 : 0;
              int r = 224, g = 255, b = 224;
              Bitmap bitmap = new Bitmap(476, 476);
              int picBox = 1;
if (method == "Прима") t1 = algoPrima.Prima(bitmap,
cts1.Token, picBox, w, h, flag, delay, r, g, b);
              else if (method == "Крускала")
                 t1 = algoKruskal.Kruskal(bitmap, cts1.Token,
picBox, w, h, flag, delay, r, g, b);
              else if (method == "Пошук у глиб")
                 t1=algoDFS.DFS(bitmap,cts1.Token,picBox,w,
h, flag, delay, r, g, b);
              else if (method == "Еллера")
                 t1=algoEller.Eller(bitmap,cts1.Token,picBox,w,
h, flag, delay, r, g, b);
              else if (method == "Рекурсивний поділ")
                 t1 = algoRD.RecursiveDivision(bitmap,
cts1.Token, picBox, w, h, flag, delay, r, g, b);}
           else { MessageBox.Show("Не вибран метод
генерації лабіринту 1"); }}
        if (checkBox5.Checked){
           string method = comboBox2.Text;
if (method != "" && method != "Метод генерації"){
              if (cts2 != null) { cts2.Cancel(); }
              cts2 = new CancellationTokenSource();
              maze2 = null; await Task.Delay(100);
              int delay = Convert.ToInt32(textBox2.Text);
int w = Convert.ToInt32(numericUpDown3.Value);
int h = Convert.ToInt32(numericUpDown4.Value);
              int flag = checkBox4.Checked ? 1 : 0;
int r = 255, g = 232, b = 209; Bitmap bitmap = new
Bitmap(476, 476); int picBox = 2;
if (method == "Прима") t2 = algoPrima.Prima(bitmap,
cts2.Token, picBox, w, h, flag, delay, r, g, b);
              else if (method == "Крускала")
                 t2 = algoKruskal.Kruskal(bitmap, cts2.Token,
picBox, w, h, flag, delay, r, g, b);
              else if (method == "Пошук у глиб")
                 t2=algoDFS.DFS(bitmap,cts2.Token,picBox,w,
h, flag, delay, r, g, b);
              else if (method == "Еллера")
                 t2=algoEller.Eller(bitmap,cts2.Token,picBox,w,
h, flag, delay, r, g, b);
              else if (method == "Рекурсивний поділ")
                 t2 = algoRD.RecursiveDivision(bitmap,
cts2.Token, picBox, w, h, flag, delay, r, g, b);}
           else { MessageBox.Show("Не вибран метод
генерації лабіринту 2"); }}
if (checkBox7.Checked){ string method = comboBox3.Text;
if (method != "" && method != "Метод генерації"){
              if (cts3 != null) { cts3.Cancel(); }
              cts3 = new CancellationTokenSource();
              maze3 = null; await Task.Delay(100);
              int delay = Convert.ToInt32(textBox3.Text);
int w = Convert.ToInt32(numericUpDown5.Value);
int h = Convert.ToInt32(numericUpDown6.Value);
              int flag = checkBox6.Checked ? 1 : 0;
int r = 213, g = 213, b = 255; Bitmap bitmap = new
Bitmap(476, 476); int picBox = 3;
              if (method == "Прима")
                 t3 = algoPrima.Prima(bitmap, cts3.Token,
picBox, w, h, flag, delay, r, g, b);
              else if (method == "Крускала")
                 t3 = algoKruskal.Kruskal(bitmap, cts3.Token,
picBox, w, h, flag, delay, r, g, b);
              else if (method == "Пошук у глиб")
                 t3=algoDFS.DFS(bitmap,cts3.Token,picBox,w,
h, flag, delay, r, g, b);
              else if (method == "Еллера")
                 t3=algoEller.Eller(bitmap,cts3.Token,picBox,w,
h, flag, delay, r, g, b);
              else if (method == "Рекурсивний поділ")
                 t3 = algoRD.RecursiveDivision(bitmap,
cts3.Token, picBox, w, h, flag, delay, r, g, b);}
           else { MessageBox.Show("Не вибран метод
генерації лабіринту 3"); }}
        // Збираємо всі НЕ-null задачі в один список
        List<Task<Cell[,]>> tasks = new List<Task<Cell[,]>>();
        if (t1 != null) tasks.Add(t1); if (t2 != null) tasks.Add(t2);
if (t3 != null) tasks.Add(t3);
        if (tasks.Count > 0) { await Task.WhenAll(tasks);
           if (t1 != null) maze1 = t1.Result;
           if (t2 != null) maze2 = t2.Result;
           if (t3 != null) maze3 = t3.Result;}}
     private void button2_Click(object sender, EventArgs e){
        string file_type; file_type = comboBox4.Text;
        Export export = new Export();
        if ((!checkBox2.Checked && !checkBox5.Checked
&& !checkBox7.Checked) || comboBox4.Text == ""){
           MessageBox.Show("Не вибран лабіринт для
експорту або формат експорту");}
        if (checkBox2.Checked){
           if (file_type != "" && maze1 != null){
if (file_type == "PNG") { export.ExportMazeToPng(maze1,
"myMaze1.png", 255, 255, 255); }
              else if (file_type == "PDF") {
export.ExportMazeToPdf(maze1, "myMaze1.pdf", 255, 255,
255); }
              else if (file_type == "SVG") {
export.ExportMazeToSvg(maze1, "myMaze1.svg", 255, 255,
255); }
              else if (file_type == "FBX") {
export.ExportMazeToFbx(maze1, "myMaze1.fbx"); }}
           else { MessageBox.Show("Створіть лабіринт 1 та
виберіть формат збереження файлу"); }}
        if (checkBox5.Checked){
           if (file_type != "" && maze2 != null){
```

```
            if (file_type == "PNG") { export.ExportMazeToPng(maze2, "myMaze2.png", 255, 255, 255); }
            else if (file_type == "PDF") { export.ExportMazeToPdf(maze2, "myMaze2.pdf", 255, 255, 255); }
            else if (file_type == "SVG") { export.ExportMazeToSvg(maze2, "myMaze2.svg", 255, 255, 255); }
            else if (file_type == "FBX") { export.ExportMazeToFbx(maze2, "myMaze2.fbx"); }}
        else { MessageBox.Show("Створіть лабіринт 2 та виберіть формат збереження файлу"); }}
        if (checkBox7.Checked){
            if (file_type != "" && maze3 != null){
if (file_type == "PNG") { export.ExportMazeToPng(maze3, "myMaze3.png", 255, 255, 255); }
else if (file_type == "PDF") { export.ExportMazeToPdf (maze3, "myMaze3.pdf", 255, 255, 255); }
else if (file_type == "SVG") { export.ExportMazeToSvg (maze3, "myMaze3.svg", 255, 255, 255); }
else if (file_type == "FBX") { export.ExportMazeToFbx (maze3, "myMaze3.fbx"); }}
            else { MessageBox.Show("Створіть лабіринт 3 та виберіть формат збереження файлу"); }}}
    private void checkBox2_CheckedChanged(object sender, EventArgs e){
if (checkBox2.Checked){comboBox1.Enabled = true; numericUpDown1.Enabled = true;numericUpDown2.Enabled = true;checkBox1.Enabled = true;}
else{comboBox1.Enabled = false; numericUpDown1.Enabled = false; numericUpDown2.Enabled = false;
            checkBox1.Enabled = false;}}
    private void checkBox5_CheckedChanged(object sender, EventArgs e){
        if (checkBox5.Checked){comboBox2.Enabled = true; numericUpDown3.Enabled = true;numericUpDown4.Enabled = true;checkBox4.Enabled = true;}
else{comboBox2.Enabled = false;numericUpDown3.Enabled = false;numericUpDown4.Enabled = false;checkBox4.Enabled = false;}}
    private void checkBox7_CheckedChanged(object sender, EventArgs e){
        if (checkBox7.Checked){comboBox3.Enabled = true; numericUpDown5.Enabled = true;numericUpDown6.Enabled = true;checkBox6.Enabled = true;}
else{comboBox3.Enabled = false;numericUpDown5.Enabled = false;numericUpDown6.Enabled = false;
            checkBox6.Enabled = false;}}
    private void checkBox1_CheckedChanged(object sender, EventArgs e){
        if (checkBox1.Checked){textBox1.Enabled = true;}
        else{textBox1.Enabled = false;}}
    private void checkBox4_CheckedChanged(object sender, EventArgs e){
        if (checkBox4.Checked){textBox2.Enabled = true;}
        else{textBox2.Enabled = false;}}
    private void checkBox6_CheckedChanged(object sender, EventArgs e){
        if (checkBox6.Checked){textBox3.Enabled = true;}
        else{textBox3.Enabled = false;}}
    private void pictureBox1_DoubleClick(object sender, EventArgs e){
        if (maze1 != null) { Form2 form2 = new Form2(maze1); form2.Show(); }}
    private void pictureBox2_DoubleClick(object sender, EventArgs e){if (maze2 != null) { Form2 form2 = new Form2(maze2); form2.Show(); } }
    private void pictureBox3_DoubleClick(object sender, EventArgs e){if (maze3 != null) { Form2 form2 = new Form2(maze3); form2.Show(); }}
    private void textBox1_KeyPress(object sender, KeyPressEventArgs e){if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar)){e.Handled = true;}}
    private void textBox2_KeyPress(object sender, KeyPressEventArgs e){if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar)){e.Handled = true;}}
    private void textBox3_KeyPress(object sender, KeyPressEventArgs e){if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar)){e.Handled = true;}}
    private void button1_Click(object sender, EventArgs e){
        string exeDirectory = AppDomain.CurrentDomain.BaseDirectory;
        string fileName = "Довідка до програми.pdf";
string filePath = Path.Combine(exeDirectory, fileName);
        if (File.Exists(filePath)){try{
            Process.Start(new ProcessStartInfo{
                FileName=filePath,UseShellExecute=true});}
catch (Exception ex){
            MessageBox.Show("Не вдалося відкрити файл: " + ex.Message);}}
        else{MessageBox.Show("Файл не знайдено: " + filePath);}}}}
```

ДОДАТОК Б

Код функції для генерації лабіринту методом «Прима»

```
public async Task<Cell[,]> Prima(Bitmap bitmap, CancellationToken token, int picBox, int w, int h, int flag, int delay = 0, int rgb_r = 255, int rgb_g = 255, int rgb_b = 255){
   // Пензель для кольору тла клітинки
   SolidBrush brush = new SolidBrush(Color.FromArgb(rgb_r, rgb_g, rgb_b));
   // Створюємо двовимірний масив комірок
   Cell[,] cells = new Cell[w, h];
   for (int x = 0; x < w; x++){
      for (int y = 0; y < h; y++){
         cells[x, y] = new Cell();}}
   Random rand = globalRand;
   // 1. Випадкова стартова комірка
   int startX = rand.Next(w);
   int startY = rand.Next(h);
   cells[startX, startY].Visited = true;
   int visitedCount = 1;
   // Список ребер (між відвіданими та невідвіданими клітинками)
   List<(int x1, int y1, int x2, int y2)> edges = new List<(int, int, int, int)>();
   // Додавання суміжних невідвіданих клітинок
   void AddEdgesToFrontier(int cx, int cy){
      if (cy - 1 >= 0 && !cells[cx, cy - 1].Visited)
         edges.Add((cx, cy, cx, cy - 1));
      if (cy + 1 < h && !cells[cx, cy + 1].Visited)
         edges.Add((cx, cy, cx, cy + 1));
      if (cx - 1 >= 0 && !cells[cx - 1, cy].Visited)
         edges.Add((cx, cy, cx - 1, cy));
      if (cx + 1 < w && !cells[cx + 1, cy].Visited)
         edges.Add((cx, cy, cx + 1, cy));}
   // Додаємо початкові ребра
   AddEdgesToFrontier(startX, startY);
   // 2. Генерація лабіринту
   while (visitedCount < w * h && edges.Count > 0){
      if (token.IsCancellationRequested){ return null;}
      // Випадкове ребро
      int edgeIndex = rand.Next(edges.Count);
      var (x1, y1, x2, y2) = edges[edgeIndex];
      edges.RemoveAt(edgeIndex);
      // Якщо друга комірка ще не відвідана
      if (!cells[x2, y2].Visited){
         // Ламаємо стіни
         if (x2 == x1 && y2 == y1 - 1){
            cells[x1, y1].Top = false;
            cells[x2, y2].Bottom = false;
         }
         else if (x2 == x1 && y2 == y1 + 1){
            cells[x1, y1].Bottom = false;
            cells[x2, y2].Top = false;
         }
         else if (x2 == x1 - 1 && y2 == y1){
            cells[x1, y1].Left = false;
            cells[x2, y2].Right = false;}
         else if (x2 == x1 + 1 && y2 == y1){
            cells[x1, y1].Right = false;
            cells[x2, y2].Left = false;}
         // Позначаємо (x2, y2) відвіданою
         cells[x2, y2].Visited = true;
         visitedCount++;
         // Додаємо суміжні ребра нової комірки
         AddEdgesToFrontier(x2, y2);
         if (flag == 1){
            // Затримка між кроками
            await Task.Delay(delay);
            // Перемальовуємо дві змінені комірки
            using (Graphics gStep = Form1.Instance.CreateGraphics()){
               Form1.Instance.DrawCell(bitmap, picBox, cells, x1, y1, rgb_r, rgb_g, rgb_b);
               Form1.Instance.DrawCell(bitmap, picBox, cells, x2, y2, rgb_r, rgb_g, rgb_b);}}}}
   // Якщо після завершення треба показати весь лабіринт одним кадром (flag == 0)
   if (flag == 0){
Form1.Instance.display_maze(cells, bitmap, picBox, rgb_r, rgb_g, rgb_b);}
   return cells;}
```

ДОДАТОК В

Код функції для генерації лабіринту методом «Курскала»

```
public class DSU{
    private int[] parent;
    private int[] rank;
    public DSU(int n)    {
        parent = new int[n]; rank = new int[n];
        for (int i = 0; i < n; i++){parent[i] = i; rank[i] = 0;}}
    public int Find(int x){
        if (parent[x] != x)
            parent[x] = Find(parent[x]);
        return parent[x];}
    public void Union(int a, int b){
        a = Find(a); b = Find(b);
        if (a == b) return;
        if (rank[a] < rank[b]) parent[a] = b;
        else if (rank[a] > rank[b]) parent[b] = a;
        else{parent[b] = a; rank[a]++;}}
    public bool SameSet(int a, int b){
        return Find(a) == Find(b);}}
public async Task<Cell[,]> Kruskal(Bitmap bitmap,
CancellationToken token, int picBox, int w, int h, int flag, int
delay = 0, int rgb_r = 255, int rgb_g = 255, int rgb_b = 255){
// 1. Створюємо масив клітинок
    Cell[,] cells = new Cell[w, h];
    for (int x = 0; x < w; x++){
        for (int y = 0; y < h; y++){
            cells[x, y] = new Cell();}}
// 2. Ініціалізуємо DSU (Union-Find) на w*h елементів
    DSU dsu = new DSU(w * h);
// Допоміжна функція для перетворення (x,y) -> індекс
    int Index(int cx, int cy) => cx + cy * w;
// 3. Збираємо всі “ребра” (між суміжними клітинками)
    var edges = new List<(int x1, int y1, int x2, int y2)>();
// Горизонтальні
    for (int y = 0; y < h; y++){
        for (int x = 0; x < w - 1; x++){
            edges.Add((x, y, x + 1, y));}}
    // Вертикальні
    for (int x = 0; x < w; x++){
        for (int y = 0; y < h - 1; y++){
            edges.Add((x, y, x, y + 1));}}
// 4. Перемішуємо список стін випадковим чином (shuffle)
    Random rand = new Random();
    for (int i = edges.Count - 1; i > 0; i--){
        int rIndex = rand.Next(i + 1); var tmp = edges[i];
        edges[i] = edges[rIndex]; edges[rIndex] = tmp;}
    // 5. Перебираємо випадково усі стіни
    int totalCells = w * h; int addedEdges = 0;
    foreach (var (x1, y1, x2, y2) in edges){
        if (token.IsCancellationRequested) { return null; }
        int idx1 = Index(x1, y1); int idx2 = Index(x2, y2);
// Якщо клітинки в різних множинах – “ламаємо” стіну
        if (!dsu.SameSet(idx1, idx2)){
            dsu.Union(idx1, idx2); addedEdges++;
            // Визначаємо, як ламаємо стіну
            if (x2 == x1 + 1 && y2 == y1){
                cells[x1, y1].Right = false;
                cells[x2, y2].Left = false;}
            else if (x2 == x1 - 1 && y2 == y1){
                cells[x1, y1].Left = false;
                cells[x2, y2].Right = false;}
            else if (y2 == y1 + 1 && x2 == x1){
                cells[x1, y1].Bottom = false;
                cells[x2, y2].Top = false;}
            else if (y2 == y1 - 1 && x2 == x1){
                cells[x1, y1].Top = false;
                cells[x2, y2].Bottom = false;}
// перемалюємо тільки дві оновлені комірки
            if (flag == 1){using (Graphics g =
Form1.Instance.CreateGraphics()){
                await Task.Delay(delay);
// Перемальовуємо комірки (x1, y1) і (x2, y2)
                Form1.Instance.DrawCell(bitmap, picBox, cells,
x1, y1, rgb_r, rgb_g, rgb_b);
                Form1.Instance.DrawCell(bitmap, picBox, cells,
x2, y2, rgb_r, rgb_g, rgb_b);}}
// Якщо є w*h - 1 ребро, можна завершити
            if (addedEdges == totalCells - 1) break;}}
    if (flag == 0){Form1.Instance.display_maze(cells, bitmap,
picBox, rgb_r, rgb_g, rgb_b);}
                return cells;}
```

ДОДАТОК Д

Код функції для генерації лабіринту методом «Пошук в глиб»

```
public async Task<Cell[,]> DFS(Bitmap bitmap, CancellationToken token, int picBox, int w, int h, int flag, int delay = 0, int rgb_r = 255, int rgb_g = 255, int rgb_b = 255){
   // 1) Створюємо сітку клітин
   Cell[,] cells = new Cell[w, h];
   for (int x = 0; x < w; x++){
      for (int y = 0; y < h; y++){cells[x, y] = new Cell();}}
   // 2) Стартова клітинка (випадкова)
   Random rand = new Random();
   int startX = rand.Next(w);
   int startY = rand.Next(h);
   cells[startX, startY].Visited = true;
   // 4) Стек для DFS
   Stack<(int x, int y)> stack = new Stack<(int, int)>();
   stack.Push((startX, startY));
   // Локальна функція: зняти спільну стіну між поточною й сусідом
   void RemoveWall(int x1, int y1, int x2, int y2)    {
      if (x2 == x1 && y2 == y1 - 1)       {
         // Сусід угорі
         cells[x1, y1].Top = false;
         cells[x2, y2].Bottom = false;}
      else if (x2 == x1 && y2 == y1 + 1){
         // Сусід унизу
         cells[x1, y1].Bottom = false;
         cells[x2, y2].Top = false;}
      else if (x2 == x1 - 1 && y2 == y1){
         // Сусід ліворуч
         cells[x1, y1].Left = false;
         cells[x2, y2].Right = false;}
      else if (x2 == x1 + 1 && y2 == y1){
         // Сусід праворуч
         cells[x1, y1].Right = false;
         cells[x2, y2].Left = false;}}
   // 5) Поки в стеці ще є клітинки
   while (stack.Count > 0){
      if (token.IsCancellationRequested) { return null; }
      // Поточна клітинка (верхівка стеку)
      var (cx, cy) = stack.Peek();
      // Збираємо список невідвіданих сусідів
      List<(int nx, int ny)> neighbors = new List<(int nx, int ny)>();
      if (cy - 1 >= 0 && !cells[cx, cy - 1].Visited)
         neighbors.Add((cx, cy - 1));
      if (cy + 1 < h && !cells[cx, cy + 1].Visited)
         neighbors.Add((cx, cy + 1));
      if (cx - 1 >= 0 && !cells[cx - 1, cy].Visited)
         neighbors.Add((cx - 1, cy));
      if (cx + 1 < w && !cells[cx + 1, cy].Visited)
         neighbors.Add((cx + 1, cy));
      if (neighbors.Count > 0){
         // Випадковий сусід
         var (nx, ny) = neighbors[rand.Next(neighbors.Count)];
         // Позначаємо його відвіданим, знімаємо стіну між поточним і сусідом
         cells[nx, ny].Visited = true;
         RemoveWall(cx, cy, nx, ny);
         // Кладемо сусіда в стек (поглиблюємо пошук)
         stack.Push((nx, ny));
         if (flag == 1){using (Graphics g = Form1.Instance.CreateGraphics()){await Task.Delay(delay);
               // Перемалювати поточну
               Form1.Instance.DrawCell(bitmap, picBox, cells, cx, cy, rgb_r, rgb_g, rgb_b);
               // Перемалювати сусіда
               Form1.Instance.DrawCell(bitmap, picBox, cells, nx, ny, rgb_r, rgb_g, rgb_b);}}}
      else{
         // Якщо немає невідвіданих сусідів, повертаємось назад (pop)
         stack.Pop();}}
   // 6) Якщо flag == 0, малюємо весь лабіринт одразу по завершенні
   if (flag == 0){
      Form1.Instance.display_maze(cells, bitmap, picBox, rgb_r, rgb_g, rgb_b);}
   return cells;}
```

ДОДАТОК Е

Код функції для генерації лабіринту методом «Ейлера»

```
public async Task<Cell[,]> Eller(Bitmap bitmap, CancellationToken token, int picBox, int w, int h, int flag, int delay = 0, int rgb_r = 255, int rgb_g = 255, int rgb_b = 255){
   if (token.IsCancellationRequested) { return null; }
   // 1) Створюємо сітку клітин
   Cell[,] cells = new Cell[w, h];
   for (int x = 0; x < w; x++){
      for (int y = 0; y < h; y++){cells[x, y] = new Cell();}}
   Random rand = globalRand;
   int[] setID = new int[w]; int currentSetId = 1;
   for (int x = 0; x < w; x++){ setID[x] = currentSetId; currentSetId++;}
   async Task DrawStep(int x1, int y1, int x2, int y2){
      if (flag == 1){ if (token.IsCancellationRequested) { return; } await Task.Delay(delay);
         using (Graphics g = Form1.Instance.CreateGraphics()){
            Form1.Instance.DrawCell(bitmap, picBox, cells, x1, y1, rgb_r, rgb_g, rgb_b);
            Form1.Instance.DrawCell(bitmap, picBox, cells, x2, y2, rgb_r, rgb_g, rgb_b);}}}
   void MergeSetsHorizontal(int row, int x1, int x2){
      int oldSet = setID[x2];
      int newSet = setID[x1];
      for (int i = 0; i < w; i++){
         if (setID[i] == oldSet){
            setID[i] = newSet;}}
      if (x2 == x1 + 1){
         cells[x1, row].Right = false;
         cells[x2, row].Left = false;}
      else{
         cells[x1, row].Left = false;
         cells[x2, row].Right = false;}}
   for (int row = 0; row < h; row++){
      if (token.IsCancellationRequested) { return null; }
      // 2) Об'єднання (горизонтальне) всередині рядка
      if (row < h - 1){
         for (int x = 0; x < w - 1; x++){
            if (setID[x] != setID[x + 1]){
               if (rand.Next(2) == 0){
                  int oldSet = setID[x + 1];
                  MergeSetsHorizontal(row, x, x + 1);
                  await DrawStep(x, row, x + 1, row);}}}}
      else{for (int x = 0; x < w - 1; x++){
            if (token.IsCancellationRequested) { return null; }
            if (setID[x] != setID[x + 1]){
               MergeSetsHorizontal(row, x, x + 1);
               await DrawStep(x, row, x + 1, row);}}}
      if (row < h - 1){
         var setsInRow = new Dictionary<int, List<int>>();
         for (int x = 0; x < w; x++){
            if (token.IsCancellationRequested) { return null; }
            int s = setID[x];
            if (!setsInRow.ContainsKey(s))
               setsInRow[s] = new List<int>();
            setsInRow[s].Add(x);}
         foreach (var kvp in setsInRow){
            if (token.IsCancellationRequested) { return null; }
            var listX = kvp.Value;
            int chosen = listX[rand.Next(listX.Count)];
            // Прорізаємо вниз
            cells[chosen, row].Bottom = false;
            cells[chosen, row + 1].Top = false;
            await DrawStep(chosen, row, chosen, row + 1);
            foreach (var x in listX){
               if (token.IsCancellationRequested) { return null;}
               if (x != chosen){
                  if (rand.Next(2) == 0){
                     cells[x, row].Bottom = false;
                     cells[x, row + 1].Top = false;
                     await DrawStep(x, row, x, row + 1);}}}}
         int[] newSetID = new int[w];
         for (int x = 0; x < w; x++){
            if (token.IsCancellationRequested) { return null; }
            if (!cells[x, row].Bottom){newSetID[x] = setID[x];}
            else{ newSetID[x] = currentSetId; currentSetId++;}}
setID = newSetID;}}
   if (flag == 0){ Form1.Instance.display_maze(cells, bitmap, picBox, rgb_r, rgb_g, rgb_b);}
   return cells;}
```

ДОДАТОК Ж

Код функції для генерації лабіринту методом «Рекурсивним»

```
private static Random globalRand = new Random();
public async Task<Cell[,]> RecursiveDivision(Bitmap bitmap, CancellationToken token, int picBox, int w, int h, int flag, int delay = 0, int rgb_r = 224, int rgb_g = 255, int rgb_b = 224){if (token.IsCancellationRequested) return null;
Color passageColor = Color.FromArgb(rgb_r, rgb_g, rgb_b);
   // Створюємо двовимірний масив комірок.
   Cell[,] cells = new Cell[w, h];
   for (int x = 0; x < w; x++){for (int y = 0; y < h; y++){
         cells[x, y] = new Cell();
         cells[x, y].Top = false; cells[x, y].Bottom = false;
         cells[x, y].Left = false; cells[x, y].Right = false;}}
   // 1) Створюємо “зовнішню” рамку (стіни по периметру)
   for (int x = 0; x < w; x++){cells[x, 0].Top = true; cells[x, h - 1].Bottom = true;}
   for (int y = 0; y < h; y++){cells[0, y].Left = true;
      cells[w - 1, y].Right = true;}
   if (flag == 1){if (token.IsCancellationRequested) return null;
await Task.Delay(delay);
      using (Graphics g = Form1.Instance.CreateGraphics()){
         for (int x = 0; x < w; x++){
            if (token.IsCancellationRequested) return null;
            Form1.Instance.DrawCell(bitmap, picBox, cells, x, 0, passageColor.R, passageColor.G, passageColor.B);
            Form1.Instance.DrawCell(bitmap, picBox, cells, x, h - 1, passageColor.R, passageColor.G, passageColor.B);}
         for (int y = 0; y < h; y++){
            if (token.IsCancellationRequested) return null;
            Form1.Instance.DrawCell(bitmap, picBox, cells, 0, y, passageColor.R, passageColor.G, passageColor.B);
            Form1.Instance.DrawCell(bitmap, picBox, cells, w - 1, y, passageColor.R, passageColor.G, passageColor.B);}}
      Form1.Instance.display_maze(cells, bitmap, picBox, passageColor.R, passageColor.G, passageColor.B);}
   // 2) Основна рекурсивна функція поділу
   async Task Divide(int x1, int y1, int x2, int y2){
      if ((x2 - x1 < 2) || (y2 - y1 < 2)) {return;}
      bool horizontal = (globalRand.Next(2) == 0);
      if (horizontal){
         int yWall = globalRand.Next(y1 + 1, y2);
         int holeX = globalRand.Next(x1, x2 + 1);
         for (int xx = x1; xx <= x2; xx++){
            if (xx != holeX){cells[xx, yWall - 1].Bottom = true;
               cells[xx, yWall].Top = true;
               if (flag == 1){await Task.Delay(delay);
using (Graphics g = Form1.Instance.CreateGraphics()){
                  if (token.IsCancellationRequested) return;
Form1.Instance.DrawCell(bitmap, picBox, cells, xx, yWall - 1, passageColor.R, passageColor.G, passageColor.B);
Form1.Instance.DrawCell(bitmap, picBox, cells, xx, yWall, passageColor.R, passageColor.G, passageColor.B);}}}}
         if (token.IsCancellationRequested) return;
         await Divide(x1, y1, x2, yWall - 1);
         if (token.IsCancellationRequested) return;
         await Divide(x1, yWall, x2, y2);}
      else{int xWall = globalRand.Next(x1 + 1, x2);
         int holeY = globalRand.Next(y1, y2 + 1);
         for (int yy = y1; yy <= y2; yy++){
            if (yy != holeY){cells[xWall - 1, yy].Right = true;
               cells[xWall, yy].Left = true;
               if (flag == 1){await Task.Delay(delay);
using (Graphics g = Form1.Instance.CreateGraphics()){
                  if (token.IsCancellationRequested) return;
Form1.Instance.DrawCell(bitmap, picBox, cells, xWall - 1, yy, passageColor.R, passageColor.G, passageColor.B);
                  Form1.Instance.DrawCell(bitmap, picBox, cells, xWall, yy, passageColor.R, passageColor.G, passageColor.B);}}}}
         if (token.IsCancellationRequested) return;
         await Divide(x1, y1, xWall - 1, y2);
         if (token.IsCancellationRequested) return;
         await Divide(xWall, y1, x2, y2);}}
   // 3) Викликаємо Divide для усієї (внутрішньої) області
   if (token.IsCancellationRequested) return null;
   await Divide(0, 0, w - 1, h - 1);
   if (flag == 0){Form1.Instance.display_maze(cells, bitmap, picBox, passageColor.R, passageColor.G, passageColor.B);}
   if (token.IsCancellationRequested) return null;
   return cells;}
```

ДОДАТОК И

Код функцій експорту лабіринту у формати: PNG, PDF, SVG, FBX

```
public void ExportMazeToPdf(Cell[,] maze, string defaultFileName = "maze.pdf", int rgb_r = 255, int rgb_g = 255, int rgb_b = 255){

    if (maze == null){MessageBox.Show("Лабіринт ще не сформовано!");return;}

    int w = maze.GetLength(0); int h = maze.GetLength(1);

    using (SaveFileDialog sfd = new SaveFileDialog()){

        sfd.Filter = "PDF file|*.pdf";

        sfd.Title = "Оберіть, куди зберегти PDF";

        sfd.FileName = defaultFileName;

        if (sfd.ShowDialog() == DialogResult.OK){

            PdfDocument doc = new PdfDocument();

            doc.Info.Title = "Maze Export (PDF)";

            PdfPage page = doc.AddPage();

            double maxWidth = 600; double maxHeight = 600;
double cellSize = Math.Min(maxWidth / w, maxHeight / h);

            if (cellSize < 1) cellSize = 1; // щоб не було 0
page.Width = cellSize *w +50;page.Height = cellSize *h+50;

            XGraphics gfx = XGraphics.FromPdfPage(page);

            XBrush whiteBrush = XBrushes.White;

            XBrush passageBrush = new XSolidBrush(XColor.FromArgb(rgb_r, rgb_g, rgb_b));

            XPen blackPen = new XPen(XColors.Black, 1);

            gfx.DrawRectangle(whiteBrush, 0, 0, page.Width, page.Height);

            double offsetX = 25; double offsetY = 25;
for (int cx = 0; cx < w; cx++){for (int cy = 0; cy < h; cy++){ double xPos = offsetX + cx * cellSize; double yPos = offsetY + cy * cellSize;gfx.DrawRectangle(passageBrush, xPos, yPos, cellSize, cellSize);

                    if (maze[cx, cy].Top){gfx.DrawLine(blackPen, xPos, yPos, xPos + cellSize, yPos);}if (maze[cx, cy].Left){gfx.DrawLine(blackPen, xPos, yPos, xPos, yPos + cellSize);}if (maze[cx, cy].Bottom){gfx.DrawLine(blackPen,xPos, yPos + cellSize,xPos + cellSize, yPos + cellSize);}if (maze[cx, cy].Right){ gfx.DrawLine(blackPen,xPos + cellSize, yPos,xPos + cellSize, yPos + cellSize);}}}

try{doc.Save(sfd.FileName); MessageBox.Show($"Лабіринт збережено у:\n{sfd.FileName}");}

            catch (Exception ex){MessageBox.Show("Помилка при збереженні PDF: " + ex.Message);}}}}
```

```
  public void ExportMazeToPng(Cell[,] maze, string defaultFileName = "maze.png", int rgb_r = 255, int rgb_g = 255, int rgb_b = 255){

    if (maze == null){MessageBox.Show("Лабіринт ще не сформовано!");return;}

    int w = maze.GetLength(0); int h = maze.GetLength(1);
    using (SaveFileDialog sfd = new SaveFileDialog()){
        sfd.Filter = "PNG Image|*.png";
        sfd.Title = "Оберіть куди зберегти лабіринт";
        sfd.FileName = defaultFileName;
        if (sfd.ShowDialog() == DialogResult.OK){
            int maxWidth = 1024; int maxHeight = 1024;
int cellSize = Math.Min(maxWidth / w, maxHeight / h);
            if (cellSize < 1){cellSize = 1;}
            int canvasWidth = cellSize * w;
            int canvasHeight = cellSize * h;

            using (Bitmap bmp = new Bitmap(canvasWidth, canvasHeight))

            using (Graphics g = Graphics.FromImage(bmp)){

                g.Clear(Color.White);
Color passColor = Color.FromArgb(rgb_r, rgb_g, rgb_b);

                for (int cx = 0; cx < w; cx++){
                    for (int cy = 0; cy < h; cy++){
int xPos = cx * cellSize; int yPos = cy * cellSize;

using (SolidBrush brush = new SolidBrush(passColor)){

g.FillRectangle(brush, xPos, yPos, cellSize, cellSize);}

if (maze[cx, cy].Top){g.FillRectangle(Brushes.Black, xPos, yPos, cellSize, 1);}if (maze[cx, cy].Left){g.FillRectangle(Brushes.Black, xPos, yPos, 1, cellSize);}if (maze[cx, cy].Bottom){g.FillRectangle(Brushes.Black,xPos, yPos + (cellSize - 1), cellSize, 1);}if (maze[cx, cy].Right){g.FillRectangle(Brushes.Black,xPos + (cellSize - 1), yPos, 1, cellSize);}}}

                try{bmp.Save(sfd.FileName, System.Drawing.Imaging.ImageFormat.Png);

                    MessageBox.Show($"Лабіринт збережено у:\n{sfd.FileName}\n");}

                catch (Exception ex){MessageBox.Show($"Помилка збереження файлу:\n{ex.Message}");}}}}}
```

```
  public void ExportMazeToSvg(Cell[,] maze, string defaultFileName = "maze.svg", int rgb_r = 255, int rgb_g = 255, int rgb_b = 255){

if (maze == null){MessageBox.Show("Лабіринт ще не сформовано!");return;}

    int w = maze.GetLength(0); int h = maze.GetLength(1);

    using (SaveFileDialog sfd = new SaveFileDialog()){

        sfd.Filter = "SVG File|*.svg";
```

```
        sfd.Title = "Оберіть, куди зберегти SVG";
        sfd.FileName = defaultFileName;
        if (sfd.ShowDialog() == DialogResult.OK){
            string svgPath = sfd.FileName;
            int maxWidth = 1024; int maxHeight = 1024;
int cellSize = Math.Min(maxWidth / w, maxHeight / h);
            if (cellSize < 1) cellSize = 1;
int svgWidth = cellSize * w; int svgHeight = cellSize * h;
            using (StreamWriter sw = new
StreamWriter(svgPath, false, System.Text.Encoding.UTF8)){
                sw.WriteLine($"<svg version=\"1.1\"
baseProfile=\"full\" " +
$"width=\"{svgWidth}\" height=\"{svgHeight}\" " +
$"xmlns=\"http://www.w3.org/2000/svg\">");
                for (int cx = 0; cx < w; cx++){ for (int cy = 0; cy
< h; cy++){ int xPos = cx * cellSize; int yPos = cy * cellSize;
sw.WriteLine($"  <rect x=\"{xPos}\" y=\"{yPos}\" " +
$"width=\"{cellSize}\" height=\"{cellSize}\" " +
$"fill=\"rgb({rgb_r},{rgb_g},{rgb_b})\" />");
                    if (maze[cx, cy].Top){sw.WriteLine(
$"  <line x1=\"{xPos}\" y1=\"{yPos}\" " +
$"x2=\"{xPos + cellSize}\" y2=\"{yPos}\" " +
"stroke=\"black\" stroke-width=\"1\" />");}
                    if (maze[cx, cy].Left){sw.WriteLine(
$"  <line x1=\"{xPos}\" y1=\"{yPos}\" " +
$"x2=\"{xPos}\" y2=\"{yPos + cellSize}\" " +
"stroke=\"black\" stroke-width=\"1\" />");}
                    if (maze[cx, cy].Bottom){sw.WriteLine(
$"  <line x1=\"{xPos}\" y1=\"{yPos + cellSize}\" " +
$"x2=\"{xPos + cellSize}\" y2=\"{yPos + cellSize}\" " +
"stroke=\"black\" stroke-width=\"1\" />");}
if (maze[cx, cy].Right){sw.WriteLine($"  <line x1=\"{xPos +
cellSize}\" y1=\"{yPos}\" " +
$"x2=\"{xPos + cellSize}\" y2=\"{yPos + cellSize}\" " +
"stroke=\"black\" stroke-width=\"1\" />");}}}
                sw.WriteLine("</svg>");}
            MessageBox.Show($"Лабіринт збережено
y:\n{svgPath}\n");}}}
```

```
  public void ExportMazeToFbx(Cell[,] maze, string
FileName = "maze.fbx"){if (maze == null){
        MessageBox.Show("Лабіринт ще не сформовано!");
        return;}
    using (SaveFileDialog saveFileDialog = new
SaveFileDialog()){
        saveFileDialog.Filter = "FBX файли (*.fbx)|*.fbx";
        saveFileDialog.Title = "Збереження лабіринту у
форматі FBX";
        saveFileDialog.DefaultExt = "fbx";
        saveFileDialog.FileName = FileName;
if (saveFileDialog.ShowDialog() == DialogResult.OK){
            string filePath = saveFileDialog.FileName;
int width=maze.GetLength(0);int height=maze.GetLength(1);
Scene scene=new Scene(); Node rootNode=scene.RootNode;
            float cellSize = 10f;
for (int x = 0; x < width; x++){for (int y = 0; y < height;
y++){Cell cell = maze[x, y];
Vector3 position = new Vector3(x * cellSize, 0, y * cellSize);
                if (cell.Top)AddWall(rootNode, position,
"TopWall", cellSize, true);if (cell.Right)AddWall(rootNode,
position + new Vector3(cellSize, 0, 0), "RightWall", cellSize,
false);if (cell.Bottom)AddWall(rootNode, position + new
Vector3(0, 0, cellSize), "BottomWall", cellSize, true);if
(cell.Left)AddWall(rootNode, position, "LeftWall", cellSize,
false);}}
            scene.Save(filePath, FileFormat.FBX7500Binary);
            MessageBox.Show($"Лабіринт збережено
y:\n{filePath}", "Збереження FBX",
MessageBoxButtons.OK, MessageBoxIcon.Information);}}}
  private static void AddWall(Node rootNode, Vector3
position, string name, float size, bool horizontal){
Mesh wallMesh = new Mesh(); float height = 10f; float
thickness = 1f;
    Vector4[] vertices; if (horizontal){vertices = new[]{
    new Vector4(0, 0, 0, 1),new Vector4(size, 0, 0, 1),
    new Vector4(size, height, 0, 1),new Vector4(0, height, 0,
1)};}
    else{vertices = new[]{new Vector4(0, 0, 0, 1),new
Vector4(0, 0, size, 1),new Vector4(0, height, size, 1),
new Vector4(0, height, 0, 1)};}
    wallMesh.ControlPoints.Add(vertices[0]);
    wallMesh.ControlPoints.Add(vertices[1]);
    wallMesh.ControlPoints.Add(vertices[2]);
    wallMesh.ControlPoints.Add(vertices[3]);
    int v0 = 0;int v1 = 1;int v2 = 2;int v3 = 3;
    wallMesh.CreatePolygon(v0, v1, v2, v3);
    Node wallNode = rootNode.CreateChildNode(name,
wallMesh);
    wallNode.Transform.Translation = position;}
```

ДОДАТОК К

Код вікна для проходження лабіринту

```
using System; using System.Collections.Generic;
using System.ComponentModel; using System.Data;
using System.Drawing; using System.Linq;
using System.Reflection.Emit; using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static Diploma.Form1;
namespace Diploma{
   public partial class Form2 : Form{
      private Cell[,] mazeData;   // Лабіринт
      private int mazeWidth, mazeHeight;
      private int playerX = 0; private int playerY = 0;
      public Form2(Cell[,] maze){
         InitializeComponent(); this.mazeData = maze;
         this.mazeWidth = maze.GetLength(0);
         this.mazeHeight = maze.GetLength(1);}
      private void Form2_Load(object sender, EventArgs e){
         display_maze(mazeData);}
      void end_game(){if (playerX == mazeWidth - 1 &&
playerY == mazeHeight - 1){ MessageBox.Show("Вітаю,
лабіринт пройдено!"); this.Close();}}
      private void Form2_KeyDown(object sender,
KeyEventArgs e){
         // W = вгору, A = ліворуч, S = вниз, D = праворуч
         int newX = playerX; int newY = playerY;
         switch (e.KeyCode){case Keys.W:
if (!mazeData[playerX, playerY].Top && playerY > 0){
               newY = playerY - 1;} break;
            caseKeys.S:if(!mazeData[playerX,playerY].Bottom
&& playerY < mazeHeight - 1){
               newY = playerY + 1;} break;
            case Keys.A: if (!mazeData[playerX, playerY].Left
&& playerX > 0){newX = playerX - 1;} break;
            case Keys.D: if (!mazeData[playerX, playerY].Right
&& playerX < mazeWidth - 1){ newX = playerX + 1;}break;
            case Keys.Up: if (!mazeData[playerX, playerY].Top
&& playerY > 0) newY = playerY - 1; break;
            case Keys.Down: if (!mazeData[playerX,
playerY].Bottom && playerY < mazeHeight - 1) newY =
playerY + 1; break;
            caseKeys.Left:if(!mazeData[playerX,playerY].Left
&& playerX > 0) newX = playerX - 1; break;
            case Keys.Right: if (!mazeData[playerX,
playerY].Right && playerX < mazeWidth - 1)
newX = playerX + 1; break;}
         if (newX != playerX || newY != playerY){
            playerX = newX; playerY = newY;
            display_maze(mazeData); end_game();}}
      public void display_maze(Cell[,] cells){
         int canvasWidth = 701; int canvasHeight = 701;
         int w = cells.GetLength(0); int h = cells.GetLength(1);
int cellSize = Math.Min(canvasWidth / w, canvasHeight / h);
Bitmap bitmap = new Bitmap(canvasWidth, canvasHeight);
using (Graphics g = Graphics.FromImage(bitmap)){g.Clear
(Color.White); Pen blackPen = new Pen(Color.Black, 1);
            for (int i = 0; i < w; i++){for (int j = 0; j < h; j++){
               int xPos = i * cellSize; int yPos = j * cellSize;
               if (cells[i, j].Top)
g.DrawLine(blackPen, xPos, yPos, xPos + cellSize, yPos);
               if (cells[i, j].Left)
g.DrawLine(blackPen, xPos, yPos, xPos, yPos + cellSize);
               if (cells[i, j].Bottom)g.DrawLine(blackPen,
xPos, yPos + cellSize, xPos + cellSize, yPos + cellSize);
               if (cells[i, j].Right)g.DrawLine(blackPen, xPos
+ cellSize, yPos, xPos + cellSize, yPos + cellSize);
               if (i == playerX && j == playerY){
                  RectangleFplayerRect=newRectangleF(xPos+
1, yPos + 1, cellSize - 2, cellSize - 2);
                  g.FillEllipse(Brushes.Green, playerRect);
// Використовуємо ellipse для "кружечка", можна rect }
if (i == mazeWidth - 1 && j == mazeHeight - 1){
// Малюємо залитий прямокутник (або коло) всередині
// Щоб був відступ від країв, можна зробити 1 px padding
               Rectangle finish_pos = new Rectangle(xPos
+ 1, yPos + 1, cellSize - 1, cellSize - 1);
g.FillRectangle(Brushes.Cyan, finish_pos);}}}}
```

```
pictureBox1.Image = bitmap;}}}
```