НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету

інформаційних технологій
(назва факультету)

_____ Ігор Болбот
(підпис) (Ім'я та ПРІЗВИЩЕ)

"__"_____2025 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри

комп'ютерних наук
(назва кафедри)

_____ Белла Голуб
(підпис) (Ім'я та ПРІЗВИЩЕ)

"__"_____2025 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Система генерування ігрового світу

Спеціальність 122 «Комп'ютерні науки»
(Код і найменування)

Освітня програма Інформаційні управляючі системи та технології
(Назва)

Орієнтація освітньої програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

к.т.н., доцент _____ Белла Голуб
(науковий ступінь та вчене звання) (підпис) (Ім'я та ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи

д.т.н., професор _____ Олександр Бушма
(науковий ступінь та вчене звання) (підпис) (Ім'я та ПРІЗВИЩЕ)

Виконав _____ Сергій Земов
(підпис) (Ім'я та ПРІЗВИЩЕ)

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)
доцент, к.т.н. _____ Белла Голуб
(науковий ступінь, вчене звання) (підпис) (Ім'я та ПРІЗВИЩЕ)

«01» Листопада 2024р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Земову Сергію Олексійовичу
прізвище, ім'я, по батькові

Спеціальність 122 «Комп'ютерні науки»
(код і назва)
Освітня програма Інформаційні управляючі системи та технології
(назва)
Орієнтація освітньої програми освітньо-професійна

Тема магістерської кваліфікаційної роботи Система генерування ігрового світу
затверджена наказом ректора НУБіП України від "01" листопада 2024р. №1964 «С»
Термін подання завершеної роботи на кафедру 01 грудня 2025 р.
(рік, місяць, число)
Вихідні дані до магістерської кваліфікаційної роботи: Результати створення ігрового світу, отримані від гри власної розробки (бакалаврська кваліфікаційна робота)
Перелік питань, що підлягають дослідженню:

1. Дослідження впливу параметрів генерування на процес створення ігрового світу.
2. Дослідження доцільності та можливості застосування методів OLAP та Data Mining задля покращення процесу створення світу у ігрових проектах.
3. Дослідження та виявлення закономірностей між застосованими параметрами та фінальним часом створення світу.
4. Виявлення закономірностей між кількістю об'єктів, що було згенеровано, та алгоритмом, що було використано.

Перелік графічного матеріалу (за потреби)
Дата видачі завдання "01" листопада 2024 р.

Керівник магістерської кваліфікаційної роботи _____ Олександр Бушма
(підпис) (Ім'я та ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ Сергій Земов
(підпис) (Ім'я та ПРІЗВИЩЕ)

Календарний план

№ з/п	Назва етапів виконання бакалаврської кваліфікаційної роботи	Строк виконання етапів бакалаврської кваліфікаційної роботи	Примітка
1	Видача завдання	01.11.2024	
2	Аналіз предметної області	05.11.2024 – 26.11.2024	
3	Моделювання системи	26.11.2024 – 04.02.2025	
4	Розробка системи	04.02.2025 – 25.04.2025	
5	Аналіз результатів	26.04.2025 – 12.07.2025	
6	Оформлення записки	13.07.2025 – 03.10.2025	
7	Оформлення постеру	03.10.2025 – 20.10.2025	
8	Постерна сесія	28.10.2025 – 29.10.2025	
9	Перевірка на плагіат	13.11.2025	
10	Попередній захист	01.12.2025	
11	Захист	16.12.2025	

Студент _____ Сергій Земов
(підпис) (Ім'я та ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи _____ Олександр Бушма
(підпис) (Ім'я та ПРІЗВИЩЕ)

РЕФЕРАТ

Дана робота присвячена використанню засобів OLAP та інтелектуального аналізу даних (Data Mining) для дослідження світів, отриманих в результаті застосування генеративних алгоритмів з встановленням параметрів створення світу (розмір, алгоритм). У ході роботи проводиться аналіз параметрів, що впливають на кінцевий час генерування світу.

Об'єкт дослідження: процедурна генерація в ігрових системах.

Предмет дослідження: система створення ігрового світу на основі шуму Перліна та діаграм Вороного.

Використані методи. Сховище даних, що містить результати роботи генератора ігрового світу. Для аналізу результатів використано засоби OLAP, для пошуку нових знань та закономірностей – Data Mining.

Мета роботи. Метою даної роботи є дослідження та практичне застосування методів OLAP та інтелектуального аналізу даних (Data Mining) для аналізу результатів генерації ігрового світу задля виявлення закономірностей, що допоможуть поліпшити процес створення світу.

Наукова складова. Використано технологій аналізу (OLAP, Data Mining) для пошуку способів оптимізації процесу створення ігрового світу задля зниження часу, що витрачається на процес генерування.

Рекомендації щодо впровадження результатів. Отримані результати можуть бути використані для покращення процедури створення світу у ігровій розробці, або для покращення обізнаності у області генеративних алгоритмів для подальшого їх використання у ігрових сценаріях.

Прикладна значимість роботи. Прикладна значимість роботи полягає у підвищенні обізнаності програмістів в області генеративних ігрових алгоритмів та їх поведінці при створенні ігрового світу.

Кількість сторінок – 63.

Кількість ілюстрацій – 49.

Кількість таблиць – 1.

Кількість додатків – 4.

Кількість джерел – 28.

ABSTRACT

This work is devoted to the use of OLAP tools and Data Mining techniques for the study of worlds generated as a result of applying generative algorithms with specified world creation parameters (size, algorithm).

Object of study: procedural generation in game systems.

Subject of study: the system for creating a game world based on Perlin noise and Voronoi diagrams.

Methods used. Data warehouse containing the results of the game world generator's operation. OLAP tools were used for result analysis, and Data Mining methods were applied to discover new knowledge and patterns.

Purpose of the work. The goal of this research is to study and apply OLAP and Data Mining methods for analyzing the results of game world generation in order to identify patterns that can help improve the world creation process.

Scientific contribution. Analytical technologies (OLAP, Data Mining) were used to find ways to optimize the process of creating a game world, aiming to reduce the time required for generation.

Recommendations for implementation. Obtained results can be used to improve the world creation procedure in game development or to enhance awareness in the field of generative algorithms for their further use in game scenarios.

Practical significance. Practical significance of the work is in increasing developers' awareness of generative game algorithms and their behavior during the world creation process.

Number of pages: 63.

Number of illustrations:49.

Number of tables: 1.

Number of appendices: 4.

Number of references: 28.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

СД – сховище даних.

ШІ – штучний інтеллект.

1R – One Rule алгоритм.

СУБД – система управління базами даних.

OLAP – On-Line Analytical Processing.

JSON – JavaScript Object Notation.

SQL – Structured Query Language.

SSAS – SQL Server Analysis Services.

BI - Business intelligence

SSIS – SQL Server Integration Services.

SSRS – SQL Server Reporting Services.

ВСТУП

Актуальність теми. У сучасному світі індустрія відеоігор швидко розвивається, і одним з ключових напрямів є створення унікального ігрового досвіду шляхом генерування ігрових світів. З огляду на підвищення вимог гравців до різноманітності, правдоподібності та реіграбельності ігрового процесу, дедалі більше звертаються до процедурних методів створення контенту, в тому числі – до генерації ігрових світів.

Актуальність теми зумовлена зростаючою потребою в інструментах, які не лише генерують ігрові простори, а й надають засоби для їх аналітичної оцінки. Такий підхід дає змогу розробникам приймати обґрунтовані рішення при налаштуванні генератора, створенні ігрової логіки та забезпеченні більш захопливого ігрового процесу.

Об’єктом дослідження виступає процедурна генерація в ігрових системах.

Предметом дослідження є система створення ігрового світу на основі Шуму Перліна та Діаграм Воронного.

Метою даної роботи є дослідження та практичне застосування методів OLAP та інтелектуального аналізу даних (Data Mining) для аналізу результатів генерації ігрового світу задля виявлення закономірностей, що допоможуть поліпшити процес створення світу.

У ході виконання магістерської роботи було використано наступні **методи та засоби**: мова SQL, SSAS, Reporting Services, засоби OLAP та Data Mining. Дані були отримані безпосередньо від програми (гри), що була розроблена студентом в ході виконання бакалаврської кваліфікаційної роботи.

Наукова складова полягає у тому, що у ході виконання роботи було використано засоби OLAP та Data Mining у ігровій індустрії задля покращення результатів створення ігрових світів та пошуку нових знань у цій області.

Апробація роботи була представлена на 3-х наукових конференціях:

1. VII Всеукраїнська науково-практична конференції студентів і аспірантів "Теоретичні та прикладні аспекти розробки комп'ютерних систем `2025" (м. Київ, 2025 р.). За виступ на конференції з темою «Система генерування ігрового світу» було отримано дипломом I ступеня (фото диплому надається у Додатку А).
2. II Міжнародна науково-практична конференції «Актуальні питання розвитку науки та техніки в умовах глобалізації (м. Боярка, 2025 р.).
3. XVI Міжнародна науково-практична конференція молодих вчених «Інформаційні технології: економіка, техніка, освіта» (м. Київ, 2025р.).

Структура роботи складається з 4-х розділів. Перший розділ присвячено постановці завдання, огляду існуючих рішень та процесу створення ігрового світу. Другий розділ присвячено моделюванню: наведено його загальні положення, а також проведено моделювання системи за допомогою діаграм прецедентів, класів, послідовностей і розгортання. Третій розділ надає опис основних використаних методів (OLAP, Data Mining), а також надає опис джерела даних, з якого дані надходять до сховища даних та як саме ці дані надходять до сховища. Четвертий розділ присвячено аналізу отриманих результатів та формуванню висновків по даним.

Робота містить 63 сторінки, 49 малюнків та 28 джерел.

1 АНАЛІЗ ПРОЦЕСУ СТВОРЕННЯ ІГРОВОГО СВІТУ

1.1 Процес створення ігрового світу як об’єкт дослідження

1.1.1 Поняття ігрового світу. Ігровий світ є центральним поняттям у розробці відеоігор. Все, що відбувається у грі – відбувається всередині цього світу. Тож процес його створення та наповнення є ключовим етапом у розробці будь-якої гри.

Даний процес є поетапним, і включає роботу програмістів, геймдизайнерів та художників.

Першим етапом є концептуальне проектування. Під час цього етапу визначається жанр, художній стиль, інші особливості. На цьому етапі формується майбутня атмосфера світу.

Другим кроком є створення просторової структури. Це включає в себе планування карти світу, ландшафту, формування міст, поселень та інших природних об’єктів.

Третім кроком є наповнення світу. Відповідно до сформованої раніше структури геймдизайнер проводить наповнення ігрового світу. Він вручну розміщує ігрові об’єкти (рис.1), планує маршрути гравця, прописує сценарії взаємодії між гравцем та персонажами у світі.

Рис. 1 Робота геймдизайнера

Цей підхід роками доводив свою ефективність у створенні справді цікавих світів. Але він має й свій головний недолік – все сплановано заздалегідь. Так звана реіграбельність – можливість до повторного проходження – є доволі низькою, бо пройшовши гру гравець запам’ятовує, що де знаходиться, і з меншим шансом захоче ще раз бути у тих самих декораціях.

На відміну від цього класичного підходу все дедалі частіше застосовують автоматизоване (процедурне) створення ігрового світу. Цей підхід має на меті використання генеративного алгоритму для створення ігрового світу. Для цього використовують низку алгоритмів, які дозволяють програмісту ввести параметри, такі як початкове значення випадковості (seed), що забезпечує відтворюваність світу, а також розмір, що безпосередньо впливає на розмір ігрового середовища.

Такий підхід дозволяє значно зекономити час розробників, або за той самий час створити набагато більше об’єктів. Це все робиться задля забезпечення високої реіграбельності. Суть гри не зміниться при повторному проходженні, але світ буде іншим, і маршрути, що для себе будував гравець, стануть неактуальними, що призведе до вивчення ним нових шляхів проходження і підвищення зацікавленості.

1.1.2 Шум Перліна Цей алгоритм є одним з найпопулярніших на даний момент для створення світу через генеративні алгоритми. Ідея цього шуму

прийшла на думку математику Кену Перліну під час його роботи над фільмом «Трон». Йому не сподобалися моделі з фільму, він вважав їх надто несправжніми. Перлін казав : «Я шукав такі штуки, що допоможуть випадково, але контрольовано досягати результату.» [1]. Першу функцію шуму він представив у 1983 році. Сам шум є представником так званих градієнтних шумів, тобто таких, при створення яких генерується решітка випадкових градієнтів з подальшою інтерполяцією результатів [2].

Цей шум через свою відносну простоту успішно застосовується в ігровій розробці для створення як ігрових світів, так і матеріалів. Як найкращий приклад виступає саме створення ігрового світу, бо значення шуму лежить строго в межах від -1 до 1. Це дає змогу чітко визначити рельєф світу за відповідним значенням функції, де -1 наприклад може бути дном океану а 1 – вершиною гори (рис.2) [3].

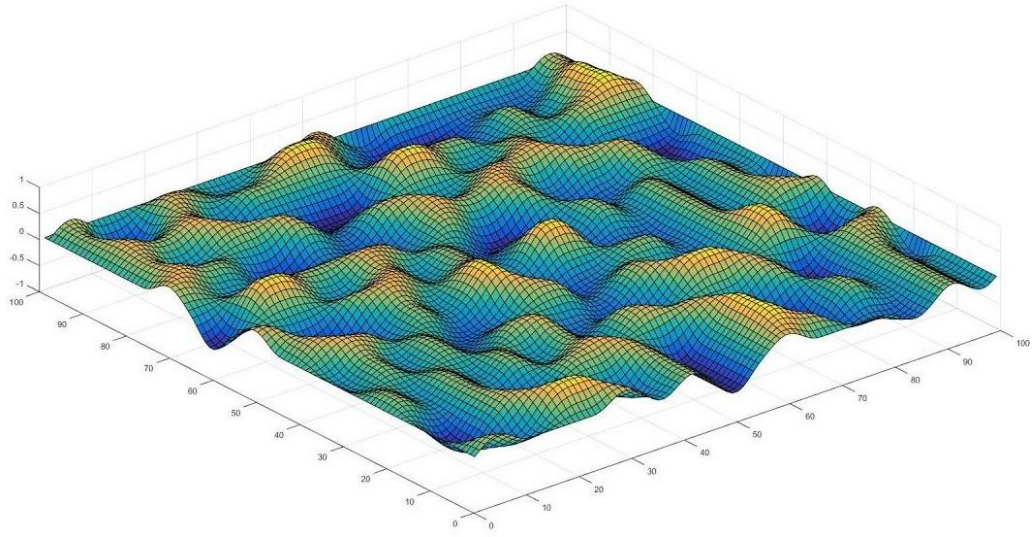


Рис. 2 Приклад світу, створеного шумом Перліна у 3-Д

На рис.2 наведено доволі простий приклад, але на ньому чітко можна побачити, де значення функції лежить ближче до одиниці (пагорби), а де значення лежить ближче до -1 (дно озера/річки).

1.1.3 Діаграми Вороного. Це алгоритм, що названо на честь українського математика Георгія Феодосійовича Вороного. Сам термін було введено у комп’ютерну науку приблизно у 1970-х роках. Ці діаграми використовувалися у великому спектрі галузей, таких як молекулярна біологія, космічні дослідження, комп’ютерна графіка та багато інших [4]. В різних країнах світу навіть

проводилися наукові симпозіуми, присвячені узагальненням діаграм Вороного (International Symposium on Voronoi Diagrams) [5].

У найпростішому вигляді діаграма являє собою множину точок на певній площині, які називаються вершинами діаграми Вороного. Кожній з вершин належить так звана комірка Вороного, що утворена з усіх точок, ближчих до цієї вершини, ніж до будь-якої іншої (рис.3) [6].

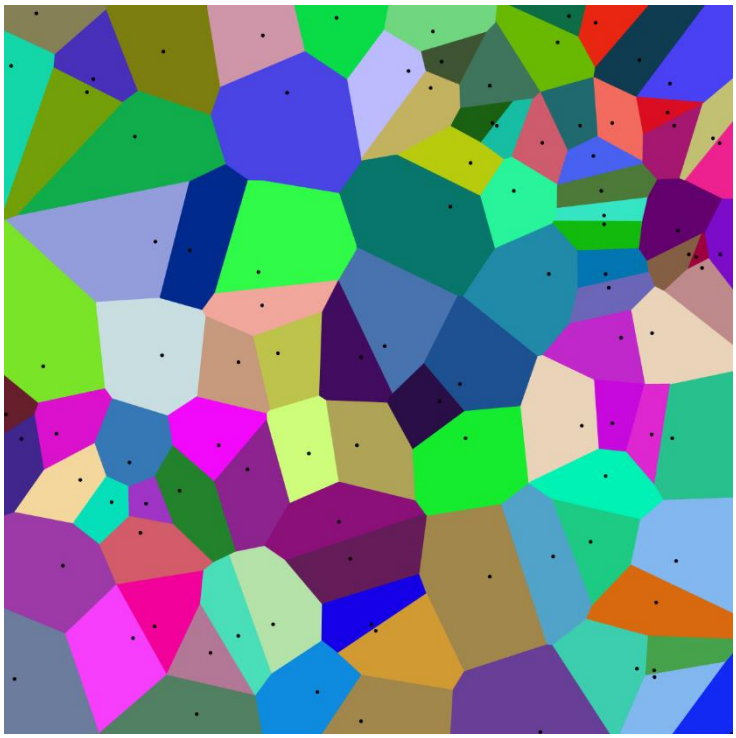


Рис. 3 Вигляд діаграм Вороного

Поглянувши на те, як виглядають діаграми Вороного, можна виявити одразу декілька варіантів використання для ігрової розробки. Першим способом є власне створення ігрового світу. Осередки діаграм можна використовувати для поділу світу на частини з подальшим їх наповненням. Місто, створене за допомогою діаграм Вороного, наведено на рис.4 [7].

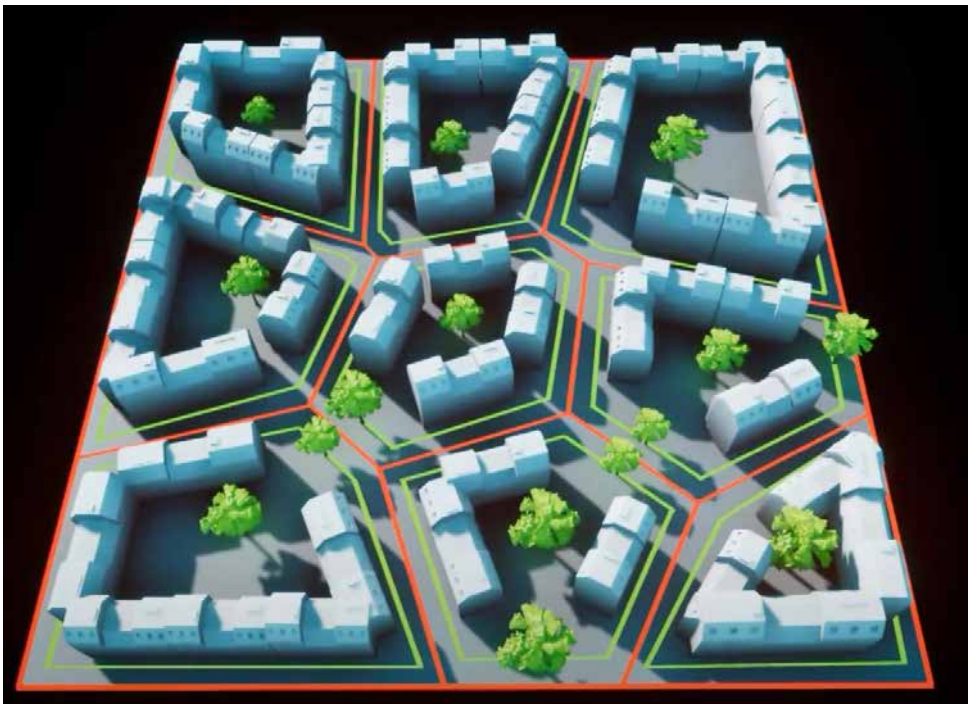


Рис. 4 Генерування міста діаграмами Вороного

Другим способом є вирішення проблем, пов’язаних з керуванням штучним інтелектом у грі. Як приклад, якщо є небезпечна місцевість – ребра можна використати для того, щоб провести ШІ такими ребрами, відстань від яких до базових точок найбільша (рис. 5) [8].

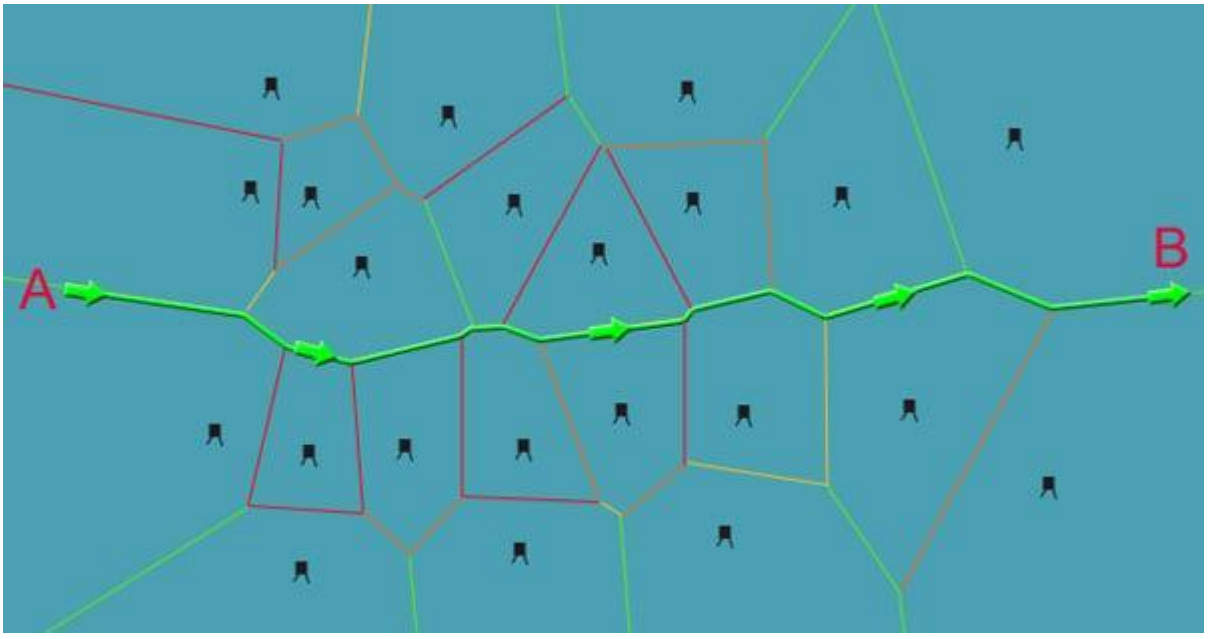


Рис. 5 Приклад безпечного маршруту

1.2 Аналіз існуючих програм-аналогів

1.2.1 Критерії відбору конкурентів. На сьогоднішній день існує багато ігор, де міститься процедурна генерація світу. Гравцеві найчастіше не дають обрати алгоритм, яким буде створено світ, а лише дозволяють обрати seed світу та інколи розмір. Для гравця жодна з цих ігор не надає аналітичних можливостей. Власне, вони гравцеві і не потрібні, йому важливо грати. Розробники ж скоріше за все використовують внутрішні засоби тестування, доступ до яких отримати не є можливим.

Було розглянуто дві гри, які мають генерування ігрового світу, а також один сервіс, що дозволяє відтворити процес створення ігрового світу. Ігри обиралися за принципом наявності аграрної складової, можливості внесення параметрів генерування ігрового світу.

1.2.2 Valheim – комп'ютерна гра в жанрі симулятора виживання з видом від третьої особи у відкритому світі, яку розробляє шведська компанія Iron Gate і видала компанія Coffee Stain. Гравець виступає у ролі воїна, якого було вбито у битві, після чого він потрапив у десятий світ. Гравцеві доведеться зіткнутися віч-на-віч з створіннями, яких за сюжетом давним-давно вигнав сюди бог Одін [9].

Гра має систему генерування ігрового світу з використанням шуму Перліна. У якості параметрів для забезпечення відтворюваності гравцеві надається можливість ввести seed світу (рис. 6)

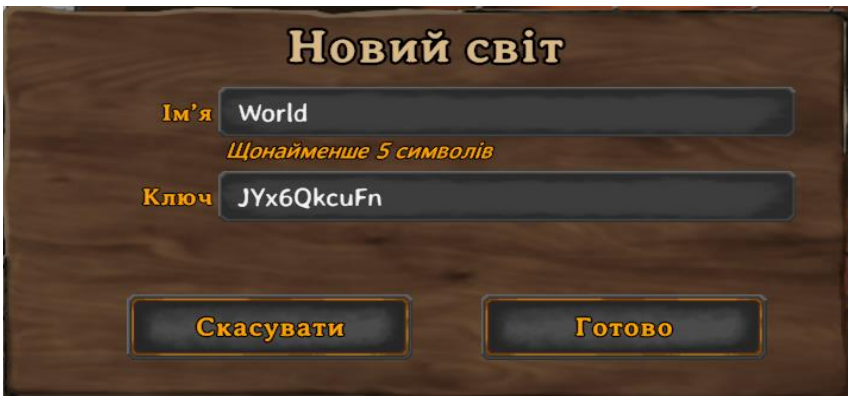


Рис. 6 Меню створення світу

Після натискання «Готово» гравець може розпочати гру у створеному світі. Розмір та наповнення є доволі великими, тож на створення в середньому йде від хвилини до двох. Увесь світ генерується випадковим чином, окрім північної та південної частини світу, бо вони завжди генеруються на цих позиціях. Інформація про способи генерування не є у відкритому доступі, тож можна лише висунути теорію про генерування суходолу шляхом поєднання діаграм Вороного для поділу його на частини, та шуму Перліна для створення загальної карти, і подальшого випадкового наповнення раніше сформованих частин. (рис.7)

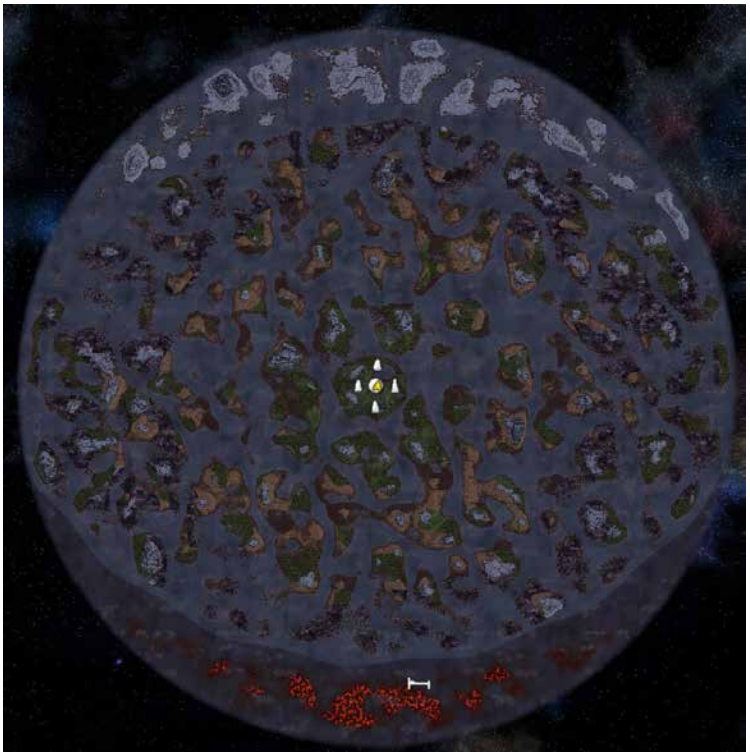


Рис. 7 Створений світ

Щодо додаткових параметрів світу – гравець може налаштувати саме ігровий процес, тобто такі його аспекти як складність бою, кількість ресурсів у світі, кількість нападів та інше (рис.8). Або просто обрати з готових шаблонів, які залишать задоволеними як гравців, що не люблять високу складність, так і тих гравців, які хочуть перевірити себе на міцність.

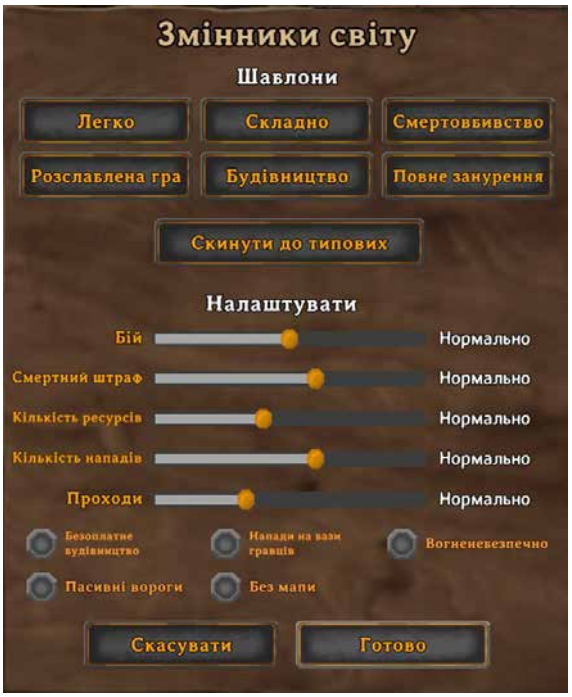


Рис. 8 Налаштування світу

Дана гра не є конкурентом, бо маючи генерування ігрового оточення не націлена на проведення аналізу результатів цього процесу.

1.2.2 Core Keeper – пригодницька гра жанру «Пісочниця», де гравець в ролі дослідника прокидається у печері біля містичного ядра у світі, що є повністю незнайомим [10].

Гра використовує генерацію світу, що створює світ у вигляді секторної діаграми. Ймовірно, що використовується модифікований алгоритм шуму Перліна, який визначає, де буде суходіл, а де, наприклад, вода чи западина.

Головною особливістю є те, що гра поділена на так звані «Chunk», тобто шматочки (рис.9). Вміст цих шматочків випадково генерується по мірі дослідження гравцем, тож це ще більше покращує реіграбельність. Найчастіше генерується одна точка інтересу для гравця (ресурс, зброя/обладунки у стилізованих локаціях, ін.).

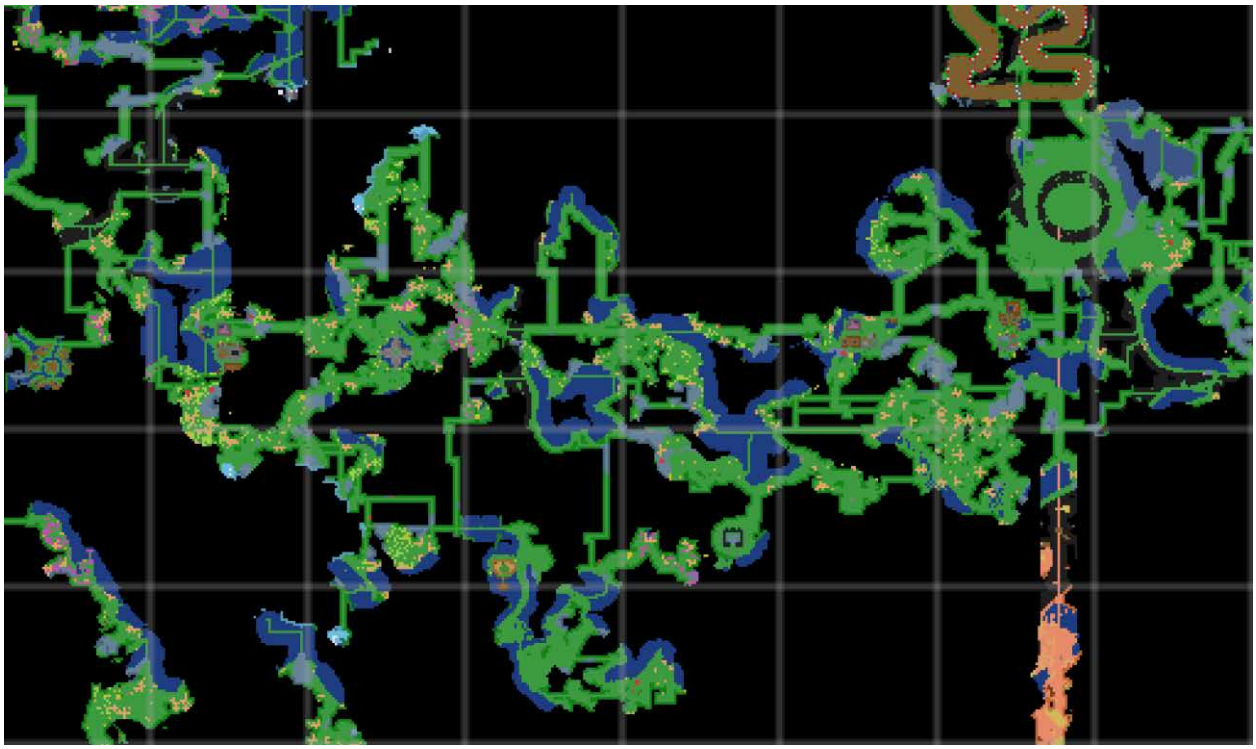


Рис. 9 Chunk Grid

Така система дозволяє створити справді щоразу унікальний ігровий досвід.

Розглянемо тепер процес створення світу. Головне меню створення світу схоже з Valheim, але має певні особливості у вигляді значка збереження (рис.10).

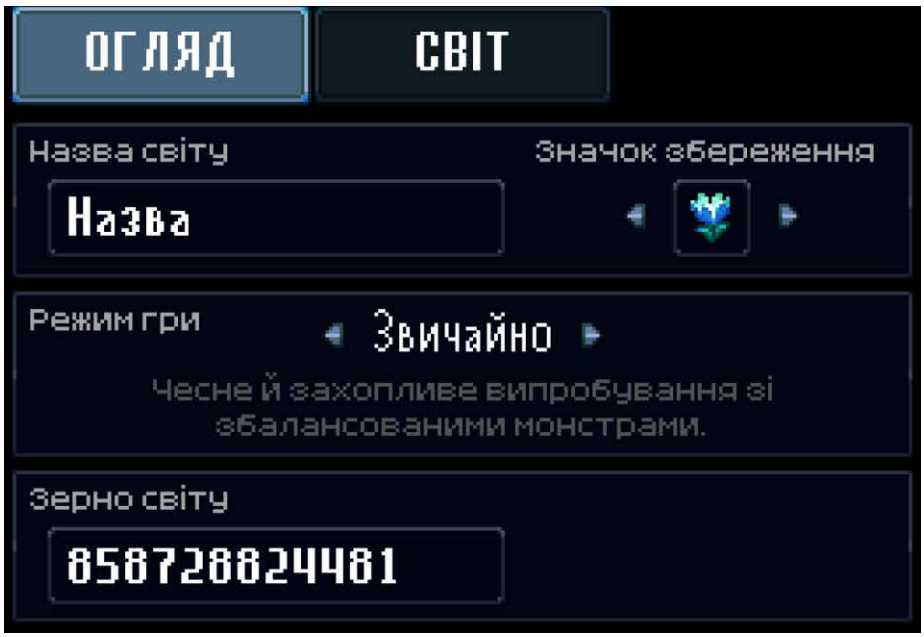


Рис. 10 Створення світу

Але обравши вкладку «Світ», можна більш детально налаштувати саме процес генерації. Обрати, як часто будуть сформовані великі та просторі кімнати

у печерах, річки, озера та інше. Цей функціонал дозволяє більш точно створити світ саме під свої потреби (рис.11).

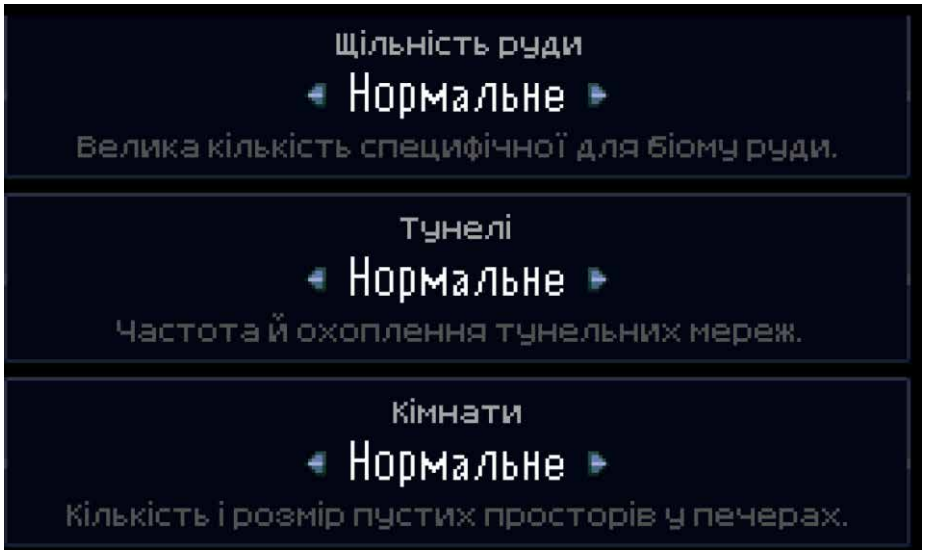


Рис. 11 Налаштування світу

Звісно ніяких засобів аналітики для гравця не передбачено, бо для нього це і не потрібно. Обрати алгоритм, за яким створюється світ також не є можливим. Гра має доволі гнучку систему налаштувань параметрів світу, але не надає жодних аналітичних можливостей.

1.2.3 Valheim Map Tool – це онлайн сервіс, який, як слідує з назви, призначений для гри Valheim. Сервіс створений для того, щоб полегшити життя гравцям, якщо вони проходять гру вдруге чи втретє та хочуть максимально оптимізувати часові затрати (рис.12). Гравець вводить seed світу або завантажує файли вже готового світу (працює точніше для пошуку ресурсів), і отримує результат.

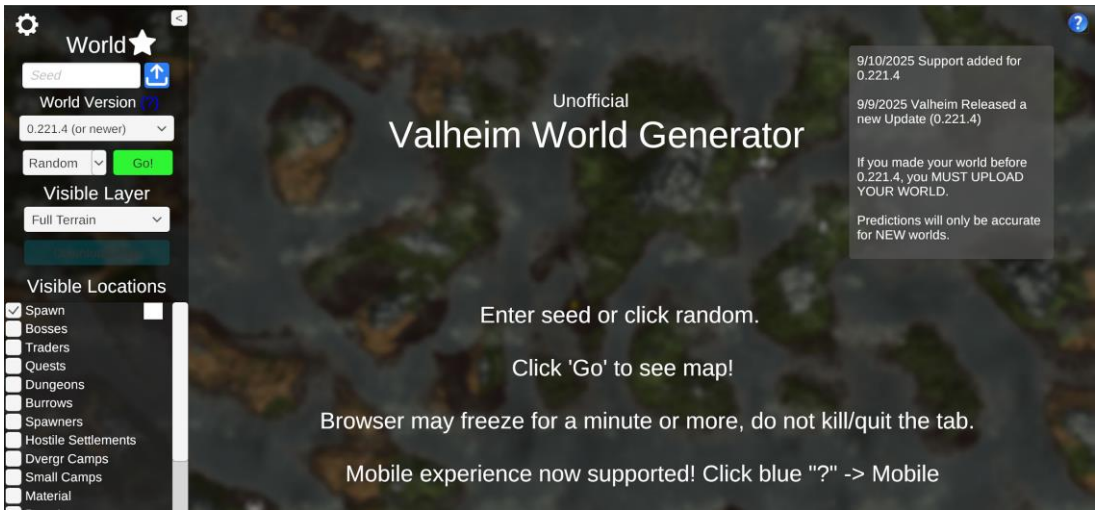


Рис. 12 Valheim World Generator

Як вже було сказано раніше, сервіс більше націлений на оптимізацію використання ігрового часу гравцем. Сервіс дозволяє ввівши seed світу побачити всю карту, дізнатися особливості рельєфу та побачити місце розташування важливих джерел ресурсів (рис.13). Сервіс використовує генерацію світу за параметром seed, але не надає інформації по часу генерації та інших параметрах, а лише використовується як так звана інтерактивна мапа, де гравець може в деталях розглянути свій світ з гри та зрозуміти, що й де знаходиться.

Якщо світ було завантажено вручну – сервіс дозволяє аналогічним чином побачити всі наявні ресурси, рельєф карти та ключові локації.

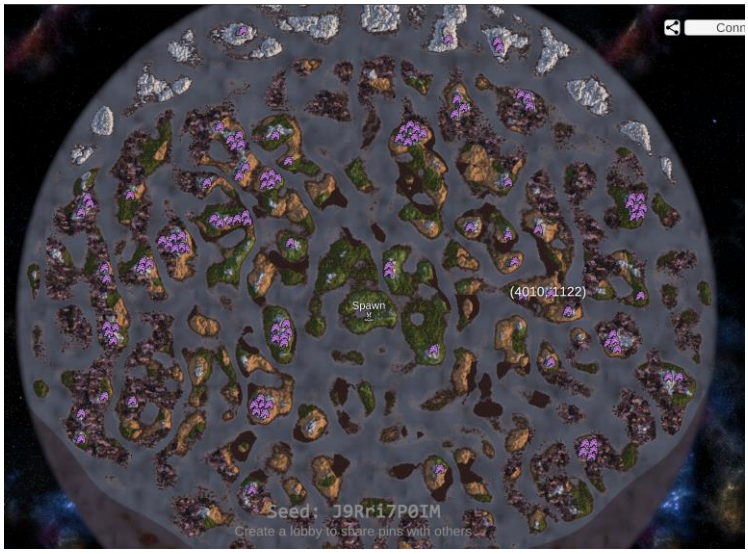


Рис. 13 Приклад пошуку печер

Даний сервіс дозволяє генерування світу на вибір користувача, хоча не надає можливості обрати алгоритм, як і гра, для якої він призначений. Сервіс надає аналітику, але суто у контексті гри та її ігрового процесу (ресурси, важливі місця, ін.) і не передбачає аналітики по самому процесу створення світу.

1.3 Постановка завдання

Необхідно розробити систему для аналізу результатів створення ігрового світу на основі гри, розробленої у ході виконання бакалаврської роботи. Для цього необхідно провести вдосконалення вже існуючої системи задля забезпечення збору необхідної інформації, розробити сховище даних та реалізувати механізм наповнення сховища даними, отриманими під час роботи системи. Після цього необхідно за допомогою засобів OLAP та інтелектуального аналізу даних (Data Mining) провести детальний аналіз даних, що були завантажені у сховище.

Система повинна надавати можливості для отримання відповідей на низку ключових питань.

1) Яка середня кількість витраченого часу на генерування за вказаний період?
2) Скільки в середньому об’єктів було згенеровано кожним алгоритмом?
3) Яка масова частка використання кожного алгоритму?
4) Скільки склав максимальний/мінімальний час генерування за вказаний період?

Розробка системи повинна проводиться у декілька етапів. Виконання нового етапу починається після повного завершення попереднього.

Першим етапом є внесення змін до існуючої системи. Метою цього етапу є модифікація системи для збереження всіх параметрів, що пов’язані з процесом генерування світу. Для цього слід внести зміни у програмний код системи, що

дозволять зберігати дані у форматі JSON. Етап вважається закінченим, коли після генерування нового світу інформація автоматично заноситься до файлу JSON.

Другим етапом є розробка сховища даних. На основі параметрів, що зберігаються на першому етапі, необхідно визначити структуру сховища даних. Використовуючи СУБД MS SQL Server та мову SQL необхідно реалізувати сховище даних відповідно до сформованої структури. Етап вважається завершеним коли сховище даних є повністю реалізованим.

Третім етапом є розробка механізму наповнення сховища даних. Для початку необхідно визначити спосіб наповнення (SSIS, застосунок власної розробки). Після цього необхідно реалізувати методи зчитування, очищення та сортування даних. Під час наповнення таблиць слід спочатку реалізувати наповнення таблиці вимірів, і на основі цих таблиць проводити наповнення таблиці фактів. Етап вважається завершеним коли дані, що містяться у файлі JSON, було повністю й безпомилково завантажено у сховище даних.

Четвертим етапом є проведення аналізу. На основі даних, що містяться у сховищі, у середовищі SSAS необхідно визначити та реалізувати основні KPI. За допомогою сервісу Reporting необхідно визначити звіти, які будуть давати відповіді на питання, що ставилися раніше. Також необхідно створити програмну реалізацію методів інтелектуального аналізу даних, а саме алгоритму 1-Rule та Naïve Bayes. Результати роботи цих методів необхідно детально дослідити, та спробувати виявити нові та практично корисні знання. Етап вважається завершеним коли KPI є сформульованими та реалізованими засобами SSAS, представлено всі необхідні звіти, а також надано детальний аналіз результатів роботи алгоритмів інтелектуального аналізу даних.

2 МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Загальні положення моделювання

Моделювання є важливим етапом будь-якої розробки. Воно дозволяє зрозуміти фундаментальні особливості як певної системи, так і наприклад певного явища або технологічного процесу.

Згідно класичного моделювання поняття моделі трактується наступним чином: Моделлю називається представлення об’єкта, системи чи поняття в деякій абстрактній формі, що є зручною для наукового дослідження. [11, с.10]. Тобто моделлю є певне поняття абстракції.

Існує два основних підходи до проведення моделювання – структурний і функціональний. Структурний підхід використовується для виявлення елементів майбутньої системи та їх зв’язків один з одним. Функціональний підхід базується на оцінці та моделюванні саме конкретних функцій системи, причому під функцією розуміється властивість, що приводить до досягнення мети. [12, с.5].

Для забезпечення стандартизації вигляду моделей в 1994-1995 роках компанією Rational Software було розроблено UML - Unified Modeling Language [13]. У 2005 році мова була офіційно опублікована Міжнародною організацією зі стандартизації (ISO) як затверджений стандарт ISO. Ця мова забезпечує стандартну нотацію для багатьох типів діаграм, які умовно можна поділити на три основні категорії:

1) Діаграми поведінки
2) Діаграми взаємодії
3) Структурні діаграми

Кожна з цих діаграм присвячена опису певного аспекту системи, що моделюється. Структурні діаграми наприклад фокусуються на відображенні структури системи, її компонентів, і часто використовуються для документування програмної архітектури системи. Діаграми поведінки в свою чергу призначені для моделювання поведінки системи та її взаємодії з

навколишнім середовищем. Діаграми взаємодії можна назвати підгрупою поведінкових діаграм, але ці діаграми націлені на моделювання потоків даних та управління між об’єктами у системі.

Для виконання першого етапу, затвердженого у постановці завдання, було використано наступні UML діаграми:

1) Діаграма прецедентів
2) Діаграма послідовності
3) Діаграма класів
4) Діаграма розгортання

2.2 Діаграма прецедентів

Діаграма прецедентів (інколи ще називають діаграма варіантів використання, Use-Case diagram) – це тип поведінкової діаграми, що спеціалізується на моделюванні взаємодії між зовнішніми користувачами (акторами) та самою системою. Ці діаграми показують, які актори та у який спосіб можуть використовувати систему, але при цьому уникаючи демонстрації роботи системи зсередини [14].

Для системи генерування ігрового світу було розроблено діаграму прецедентів, що зображена на рис. 14.

Було також виділено наступні ролі:

1) Аналітик
2) Програміст
3) Тімлід
4) Генератор світу

Детальний опис акторів наведено у таблиці 1.

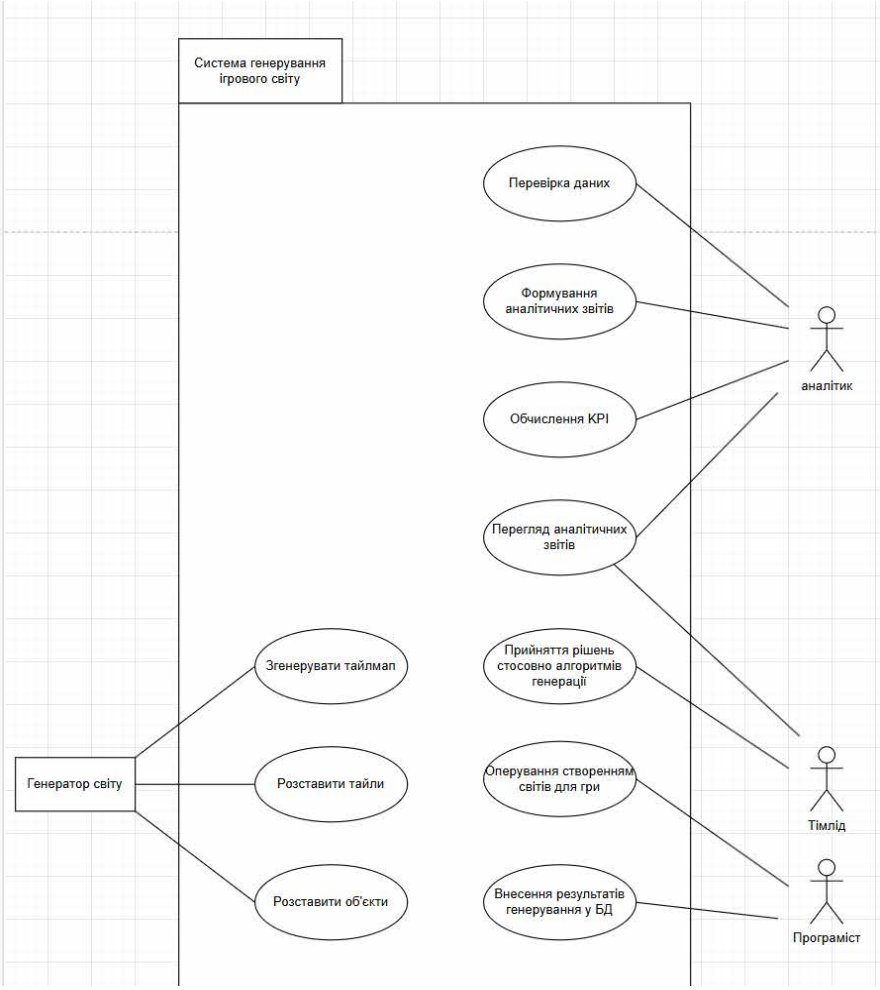


Рис. 14 Use-Case

Таблиця 1

Актори системи

Актор	Опис
Генератор світу	Системний компонент, що відповідає за створення світу
Тімлід	Керівник команди, що приймає управлінські рішення стосовно алгоритмів генерування світу.
Програміст	Оперує генератором світу та вносить дані до БД
Аналітик	Проводить аналіз результатів генерування, розраховує KPI та надає звіти керівництву.

Дана діаграма ілюструє, як різні актори взаємодіють з системою. Генератор світу виділено як окремий компонент через те, що він на основі параметрів, вказаних програмістом, генерує світ. Він, на основі даних, що були внесені програмістом, створює тайлмап, тобто карту, що вказує куди і які об’єкти необхідно встановити, і після цього виконує розстановку спочатку тайлів (елементів світу), а потім й об’єктів. Програміст, як і було сказано раніше, оперує генератором світу та вносить результати в базу даних. Аналітик, на основі внесених даних, проводить аналіз створених світів, формує звіти та розраховує KPI. Тімлід, на основі звітів, наданих аналітиком, приймає рішення стосовно гри, такі як потреба в оптимізації певних частин генерації або потреба в оптимізації використання алгоритмів.

2.3 Діаграма послідовності

Діаграма послідовності – тип діаграми взаємодії, що використовується для моделювання логіки сценаріїв використання. Цей тип діаграм показує, яка інформація передається між об’єктами в системі під час виконання певного сценарію [15].

Для моделювання системи генерування ігрового світу було розроблено діаграму, що зображена на рис. 15. В якості об’єктів було виділено програміста, генератор світу, тімліда, аналітика та саму ІС. У вигляді стрілок було зображено потоки керування між об’єктами системи.

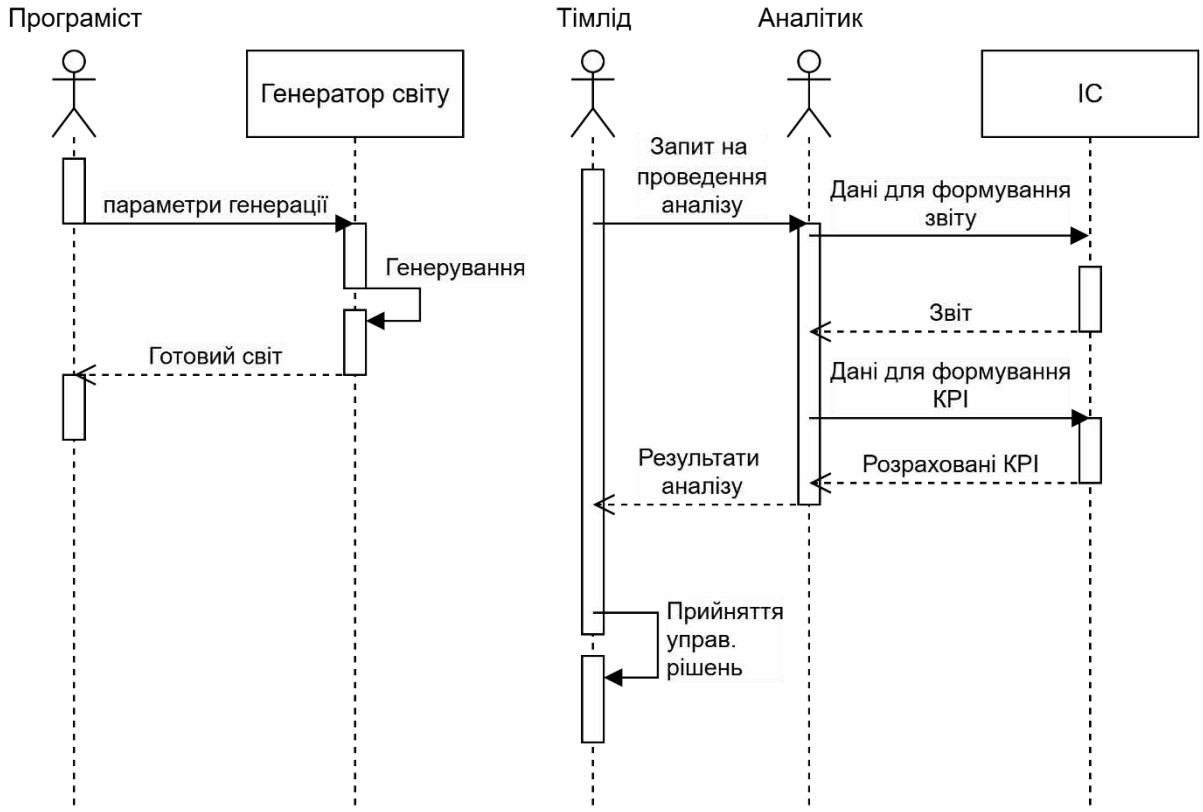


Рис. 15 Діаграма послідовності

Як видно з діаграми, програміст взаємодіє з генератором світу, передаючи йому параметри, які той (генератор) має використати для створення світу. Генератор на основі вказаних параметрів проводить створення ігрового світу. Після генерування світ повертається програмісту. Тімлід для прийняття рішень повинен мати інформацію, тож він надає запит до аналітика, щоб той (аналітик), провів дослідження результатів. Аналітик починає формувати звіти, працює з KPI і коли це все готово – повертається до тімліда з готовими звітами. На основі цих даних тімлід приймає подальші управлінські рішення. Це дозволить приймати зважені рішення відносно того, що потребує оптимізації, або де є прогалини у процесі створення світу.

2.4 Діаграма класів

Діаграма класів – діаграма, на якій представлені елементи моделі у вигляді класів з певними атрибутами, а також представлено відношення між цими об’єктами.

Клас – певний абстрактний опис множини однорідних об’єктів, що мають однакові атрибути й властивості. Клас також може бути абстрактним, тобто таким, який не може мати екземплярів [16].

Класи можуть бути класами-сутностями, що описують певні сутності, класами-інтерфейсами, тобто такими, що мають операції, але не мають атрибутів, і контрольними класами, тобто такими, які координують дії інших класів.

Типи зв’язків включать в себе відношення залежності, тобто коли зміни в визначенні одного елемента впливають на визначення іншого елемента; асоціація, тобто показ того, що об’єкти однієї сутності пов’язані з об’єктами іншої сутності; композиція, коли один об’єкт є невід’ємною частиною іншого і не може існувати окремо; агрегація – коли один елемент є частиною іншого, але може існувати самостійно.

Для системи генерування ігрового світу було розроблено діаграму класів, фрагмент якої наведено на рис.16. Повну діаграму наведено у Додатку Б. Ця діаграма є діаграмою класів гри, яка має модуль генерування ігрового світу.

На даній діаграмі представлено 3 типи класів, а саме класи управління, класи сутності та класи інтерфейси. Розглянемо детально, який клас відноситься до якого типу та чому.

Клас «Player» є класом сутністю, що репрезентує головного актора системи – гравця. Він містить основну інформацію про гравця, а також оброблює введення користувача.

Клас «WorldGenerator» є класом керування, що відповідає за створення світу на основі введених користувачем параметрів, таких як seed світу. Він має методи, що дозволяють створити ігровий світ.

Клас «SavesManager» є керуючим класом, що відповідає за обробку функціоналу збереження даних. Він дозволяє зберегти структуровані дані у файл і завантажити їх.

Клас «SaveData» є класом сутністю, що репрезентує собою дані, які необхідно записати у файл. Він містить необхідні поля, що дозволяють виконати збереження стану світу, об’єктів, гравця.

Клас «UI» є класом інтерфейсом, що репрезентує головний інтерфейс гри. Він містить методи для обробки команд користувача, а також для команд меню, таких як збереження даних та завантаження.

Клас «PlantHolder» є класом сутністю, який репрезентує об’єкт грядки. Він містить у собі статуси грядки, дані про рослину та її стадії росту.

Клас «PlantableSteps» є класом сутністю, який відображає інформацію про кроки зростання рослини. Він містить дані про те, яку культуру буде зібрано, скільки кроків буде рости рослина, скільки часу для цього буде треба, скільки об’єктів буде отримано при удобренні.

Клас «Shop» є контрольним класом, що дозволяє визначити можливість продажу речі гравцем а також оброблює випадки, коли гравець підійшов до тригеру покупки.

Клас «MoneyManager» є контрольним класом, що відповідає за керування грошима гравця. Він містить у собі поточний баланс, і має методи для його зміни.

Клас «Money» є інтерфейсом, а саме текстовим полем, де візуально відображено поточний баланс гравця.

Клас «Inventory» є сутністю, що представляє інвентар гравця. Він містить у собі список комірок інвентаря, а також методи для керування ними.

Клас «Slot» є сутністю, що представляє собою комірку інвентаря. Він містить дані про предмет у вказаній комірці, його кількість. Він також містить методи, які дозволяють перевірити можливість додавання у комірку речі, і виконати додавання.

Клас «Item» є сутністю, що містить інформацію про предмет, будь то назва, тип, зображення.

Клас «Toolbar_UI» є контрольним класом, що відповідає за візуальну взаємодію гравця з панеллю інструментів, таку як візуальне виділення обраної комірки, обрання поточної комірки.

Клас «Inventory_UI» є контрольним класом, що відповідає за відображення змін інвентаря. Він дозволяє гравцеві бачити речі, які він має, а також перетягувати їх та викидати.

Клас «InventoryPanel» є класом інтерфейсом, що відображає поточний стан інвентаря гравця. Він є елементом типу «Canvas», де розміщено об’єкти для візуалізації предметів та інших об’єктів.

Клас «ToolbarPanel» є класом інтерфейсом, що відображає поточний стан панелі інструментів гравця. Він відображає 9 комірок, а також підсвічує обрану на даний момент часу комірку.

Клас «JsonHelper» є контрольним класом, що призначений для конвертації об’єктів з формату JSON та навпаки.

Клас «ResultsSaver» є контрольним класом, що призначений для збереження інформації про згенерований світ.

Клас «GenerationData» є класом сутністю, що відповідає за структуру інформації про результати генерації, яка зберігається для подальшого аналізу.

Дана діаграма повністю описує систему генерування ігрового світу, а також супутню ігрову логіку, необхідну для коректного функціонування гри.

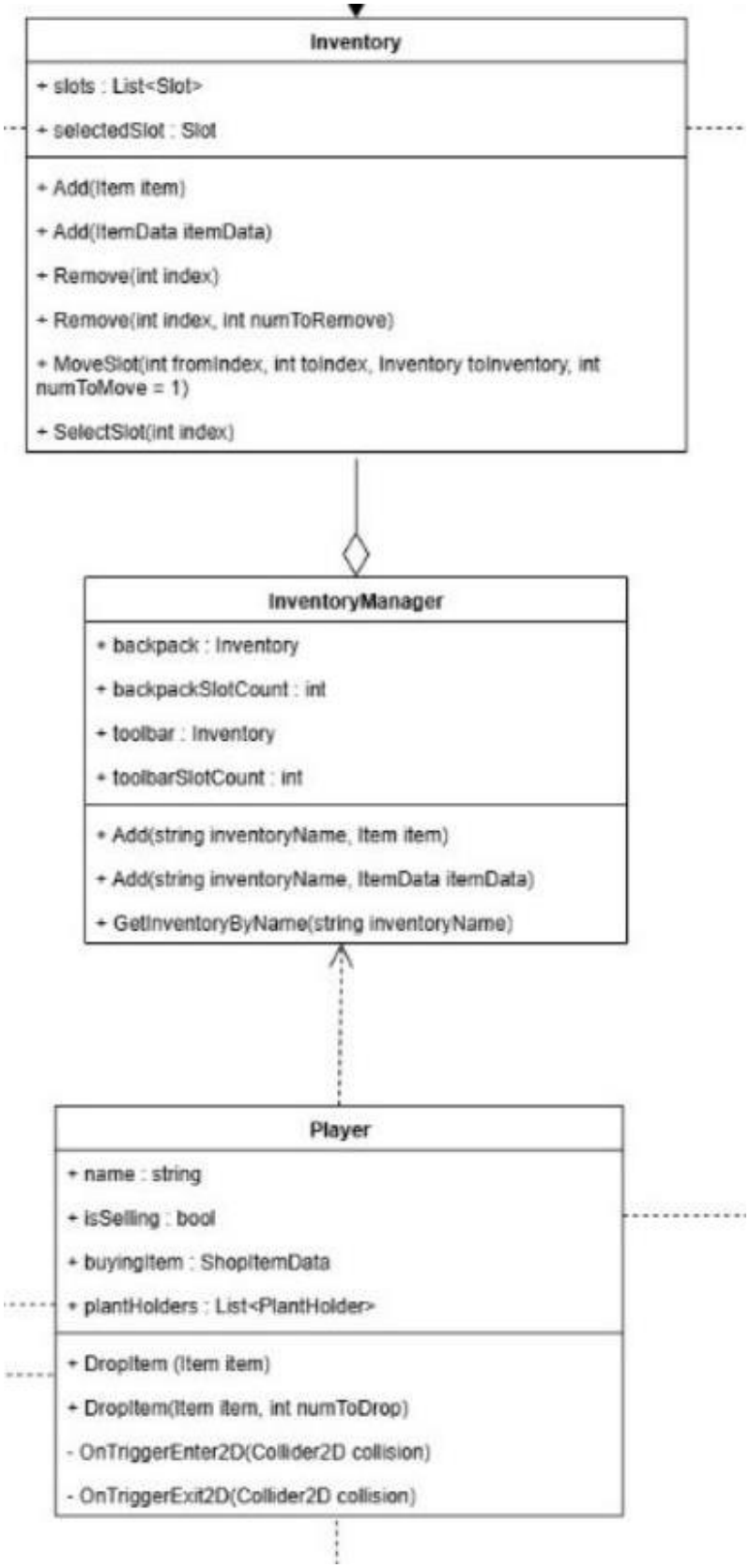


Рис. 16 Діаграма класів системи (частина)

3 РОЗРОБКА СИСТЕМИ

3.1 Діаграма розгортання системи

Діаграма розгортання — тип діаграми, яка відображає структуру системи у вигляді обчислювальних вузлів та компонентів, що містяться на цих вузлах [17].

Головне призначення таких діаграм – надати детальний опис того, як програмне забезпечення розгортається в апаратній системі. Вони візуалізують, які вузли та як саме взаємодіють між собою, а також як відбувається взаємодія між компонентами всередині одного вузла.

На рис.17 зображено діаграму розгортання, побудовану для системи генерування ігрового світу.

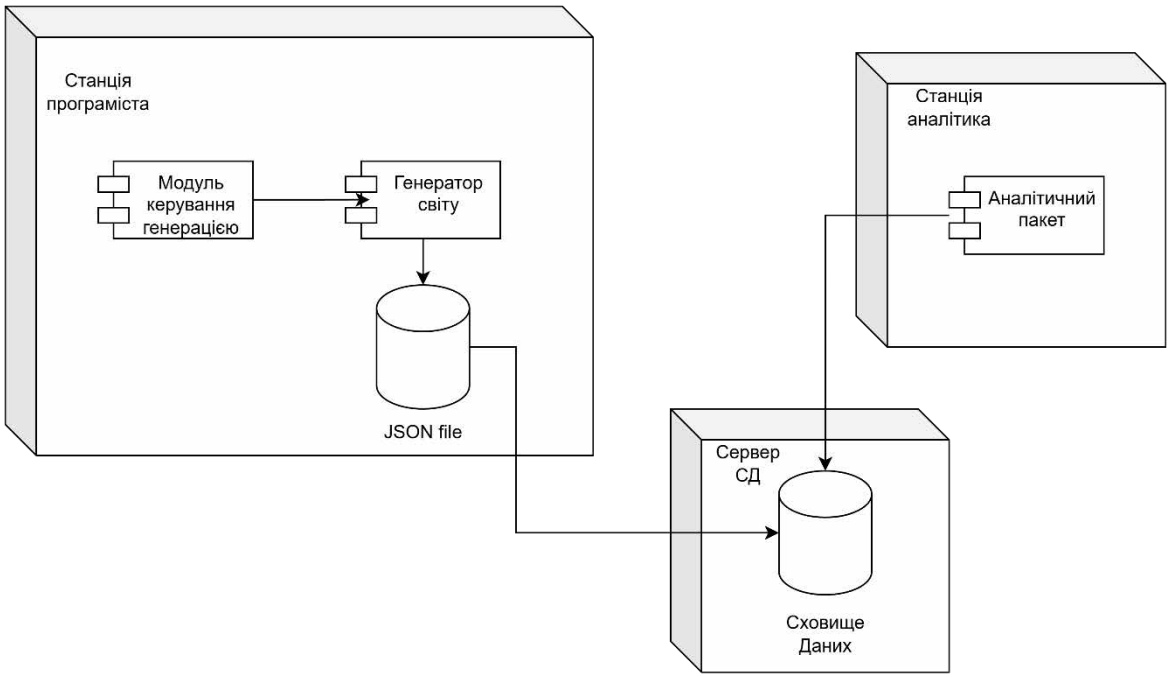


Рис. 17 Діаграма розгортання

На даній діаграмі зображено три вузли. Станція програміста є його персональним комп’ютером з грою, де він проводить генерування світу. Модуль керування генерацією отримує параметри від програміста, та запускає генератор світу, який власне й створює світ. Результати генерації зберігаються на ПК

програміста у вигляді JSON файлу з результатами. За потреби ці дані за допомогою власно розробленого застосунку завантажуються до сховища даних. Станція аналітика є його персональним комп'ютером, який він використовує для аналізу результатів. Аналітичний пакет включає в себе елементи OLAP аналізу, такі як звіти та KPI, а також засоби інтелектуального аналізу даних, такі як кластеризація, алгоритм Наївного Байеса та ін. Всі необхідні дані завантажуються з сховища даних.

3.2 Опис основних понять

В ході розробки було використано декілька ключових понять для аналізу даних, а саме OLAP та Data Mining. Було також використано метод ETL (Extract, Transform, Load) для виділення даних з файлу, їх структуризації та подальшого їх завантаження до сховища даних.

3.2.1 OLAP OLAP (Online Analytical Processing) - технологія оперативного оброблення інформації, що дозволяє в реальному часі отримувати відповіді на багатовимірні аналітичні запити. Центральним поняттям OLAP є так званий Куб даних (OLAP-куб, гіперкуб). Це куб, що складається з вимірів, які слугують осями для аналізу, а на перетині цих осей розташовуються міри – кількісні характеристики фактів, що аналізуються [18].

Залежно від того, як зберігаються дані, OLAP системи поділяються наступним чином:

1) MOLAP (Multidimensional OLAP) – система, що використовує спеціалізовані багатовимірні бази даних для зберігання даних у вигляді кубів. У такій системі дані організуються у вигляді багатовимірного масиву, де кожен вимір куба є окремою вимірною осью, а показники є значеннями, які підлягають агрегації.
2) ROLAP (Relational OLAP) – тип системи, що не використовує куби, а працює безпосередньо з реляційними базами даних, виконуючи всі

необхідні операції агрегування в реальному часі за допомогою мови SQL.

3) HOLAP (Hybrid OLAP) – тип системи, що поєднує два попередні типи. Агреговані дані зберігаються у такому випадку в багатовимірному форматі, а детальніші дані зберігаються в реляційних таблицях.

3.2.2 Data Mining (інтелектуальний аналіз даних) — це технологія, спрямована на автоматизоване виявлення раніше невідомих, нетривіальних, практично корисних і зрозумілих людині знань у великих масивах даних. Одне з перших визначень було запропоновано Григорієм Пятецьким-Шапіро у 1996 році, де підкреслюється здатність “машини” — алгоритмів або засобів штучного інтелекту — знаходити приховану інформацію, яку складно або неможливо виявити традиційними методами аналізу [19].

Основна мета Data Mining полягає в отриманні нових знань, а не підтвердженні вже відомих. Знайдені закономірності повинні бути не лише новими, але й практично корисними — тобто, такими, які можна застосувати з високим ступенем достовірності для прийняття рішень або поліпшення певних процесів. Крім того, отримані результати мають бути логічно обґрунтованими й представленими в інтерпретованій формі, що робить їх доступними для користувача.

Тобто головна відмінність Data Mining від OLAP полягає саме в призначенні. OLAP дозволяє нам перевірити коректність вже раніше відомих нам гіпотез, в той час як Data Mining дозволяє отримати нові знання та приступити до власне формування нових гіпотез. Обидва методи було використано у ході роботи, але OLAP було використано для відповіді на питання, поставлені у Постановці завдання, в той час як засоби Data Mining дозволили виявити особливості генерування та отримати краще розуміння взаємодії алгоритмів з типом світу.

3.2.3 ETL – сукупність процесів управління сховищами даних, які включають в себе вилучення даних з зовнішніх джерел, їх обробку та подальше завантаження у сховище даних. Тобто дані, отримані з джерела або декількох

джерел спочатку проходять очищення від порожніх рядків, помилок, потім форматуються відповідно до необхідного формату, і лише після цього завантажуються у сховище даних [20].

Ця методологія використовується у SSIS (SQL Server Integration Services). Це платформа, що повністю присвячена ETL. Вона дозволяє встановити, які дані та звідки необхідно брати, які дії над ними проводити (очистка, форматування, ін.) і дозволяє вказати, в яку таблицю сховища даних чи вимір куба необхідно занести ці дані. Всі операції виконуються у так званих Data Flow – тобто у потоках даних, в яких чітко вказується, які дані та як необхідно обробити (рис. 18).

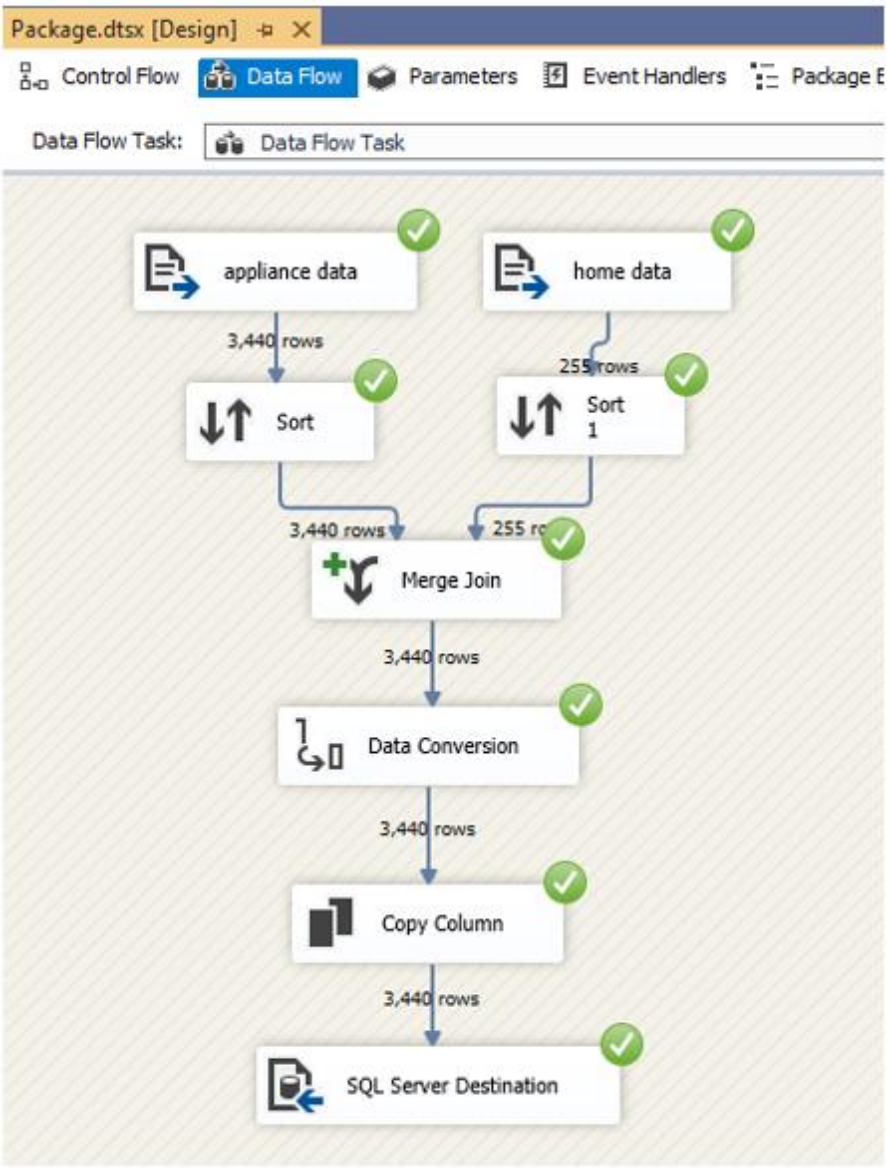


Рис. 18 Приклад Data Flow

SSIS не було використано під час роботи, хоча такий варіант розглядався. Було реалізовано програму, яка виконує читання даних з файлу, структуризацію даних та їх подальше завантаження у сховище даних. Детальніше цей процес буде розглянуто у пункті 3.6.

3.3 Опис джерела даних

Джерелом даних виступає гра, що має процедурно генерований світ. Гра дозволяє користувачеві обрати алгоритм, розмір світу та вказати seed. Головне меню зображено на рис. 19.

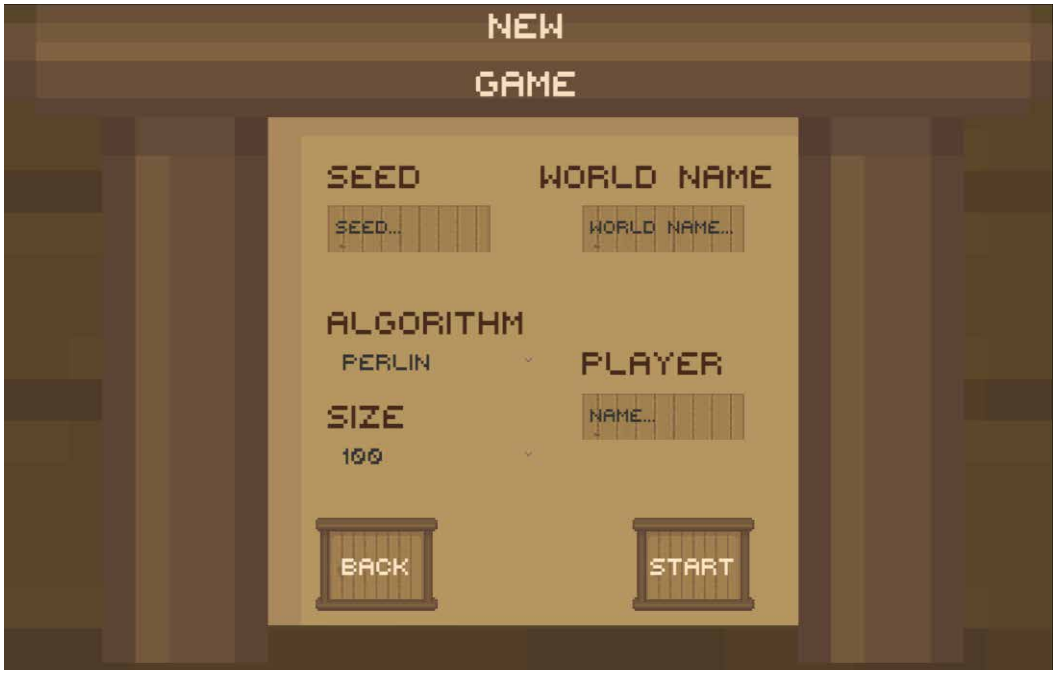
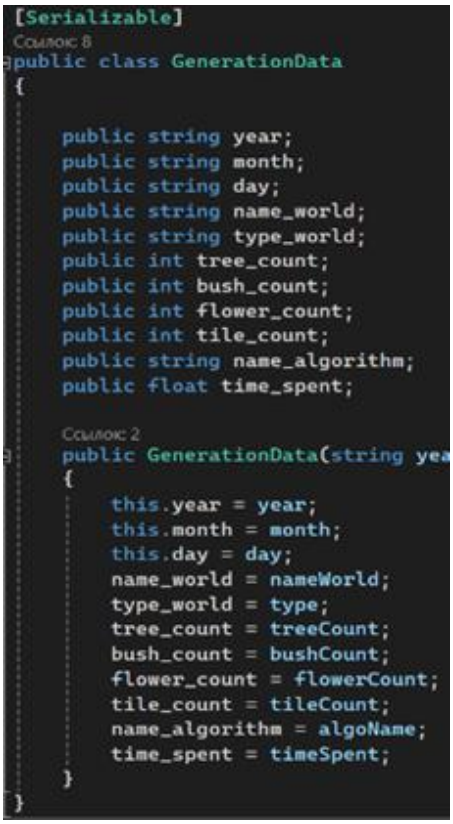


Рис. 19 Меню нової гри

Після того як світ створено, гра автоматично зберігає інформацію у файл формату JSON. Клас, що відповідає за збереження результатів наведено на рис. 20.



```
[Serializable]
Ссылок 8
public class GenerationData
{

    public string year;
    public string month;
    public string day;
    public string name_world;
    public string type_world;
    public int tree_count;
    public int bush_count;
    public int flower_count;
    public int tile_count;
    public string name_algorithm;
    public float time_spent;

    Ссылок 2
    public GenerationData(string yea
    {
        this.year = year;
        this.month = month;
        this.day = day;
        name_world = nameWorld;
        type_world = type;
        tree_count = treeCount;
        bush_count = bushCount;
        flower_count = flowerCount;
        tile_count = tileCount;
        name_algorithm = algoName;
        time_spent = timeSpent;
    }
}
```

Рис. 20 Клас для зберігання результатів

Як видно з рис. 20, зберігається доволі багато параметрів. Опишемо кожен з них детально:

1) Year – Частина дати створення світу, що зберігає рік створення.
2) Month - Частина дати створення світу, що зберігає місяць створення.
3) Day - Частина дати створення світу, що зберігає день створення.
4) Name_world – зберігає назву світу, яку користувач вказав при створенні.
5) Type_world – зберігає тип світу, що обрав користувач (зберігає словесний еквівалент розміру, де 100 – Small, 250 – Medium, 500 – Large).
6) Tree_count – змінна, що зберігає кількість дерев, які було згенеровано.
7) Bush_count – змінна, що зберігає кількість кущів, які було згенеровано.
8) Flower_count – змінна, що зберігає кількість квіток, які було згенеровано.
9) Tile_count – змінна, що зберігає кількість тайлів світу (Розмір^2)
10) Name_algorithm – назва алгоритму, яким було створено світ.

11) Time_spent – час, за який світ було створено.

Як вже було сказано раніше, було обрано формат даних JSON. Цей тип збереження має низку переваг:

1) Представляє дані у вигляді пар "ключ-значення" – Кожна змінна має свою назву, та через дві крапки вказується значення цього параметру, наприклад: "tree_count": 394
2) Підтримує типи: рядки, числа, булеві значення (true/false), масиви, об'єкти та null – У цій роботі зберігаються як числові, так і рядкові змінні, що поєднані у масив об'єктів світів.
3) Сумісний із багатьма мовами програмування, включно з JavaScript, Python, C# тощо – Даний файл створено мовою C# з використанням Unity Engine, але завдяки розповсюдженості формату цей файл можна легко відкрити іншою мовою за необхідністю.

Результуючий файл, в який занесено дані декількох світів зображено на рис. 21, де міститься інформація про декілька світів.


```
,
{
    "year": "2025",
    "month": "01",
    "day": "02",
    "name_world": "World1",
    "type_world": "medium",
    "tree_count": 345,
    "bush_count": 324,
    "flower_count": 182,
    "tile_count": 62500,
    "name_algorithm": "Combined",
    "time_spent": 0.7400000095367432
},
{
    "year": "2025",
    "month": "01",
    "day": "03",
    "name_world": "World2",
    "type_world": "small",
    "tree_count": 53,
    "bush_count": 49,
    "flower_count": 45,
    "tile_count": 10000,
    "name_algorithm": "Perlin",
    "time_spent": 0.10000000149011612
}
```


Рис. 21 Файл з даними

3.4 Опис сховища даних

Сховище даних є основним джерелом інформації для OLAP та Data Mining аналізу, який проводився в рамках виконання роботи. Сховище даних побудоване за схемою «зірка», тобто має основну центральну таблицю фактів та 4 таблиці-виміри. Схему сховища даних наведено на рис. 22. Кожна таблиця має свій унікальний ідентифікатор «id_item», а центральна таблиця фактів має складений ключ, що формується на основі інших 4 ідентифікаторів.

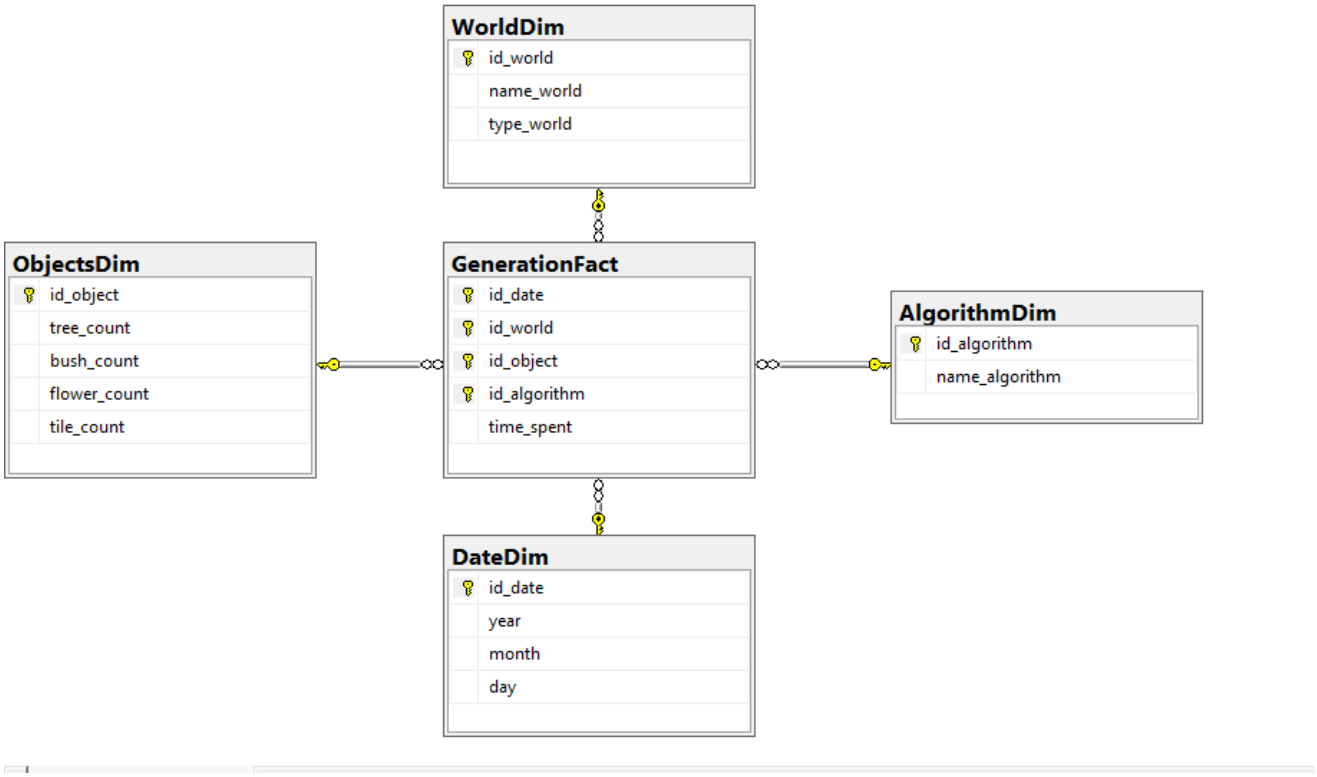


Рис. 22 Структура сховища даних

GenerationFact є таблицею фактів, і як зазначалося раніш має складений ідентифікатор, що складається з ідентифікаторів усіх чотирьох вимірів. Таблиця фактів містить час, який було витрачено на генерування світу у конкретному випадку.

ObjectsDim є виміром, що представляє кількість об’єктів, що були згенеровані у світі. Він містить кількість дерев, кущів та квітів, а також кількість тайлів, що напряму залежить від розміру світу.

AlgorithmDim є виміром, що представляє алгоритм, яким було створено світ. Містить ідентифікатор та назву алгоритму.

WorldDim є виміром, що представляє дані про світ. Містить назву світу та тип світу (маленький/середній/великий).

DateDim є виміром, що представляє дату створення світу. Є обов'язковим для аналізу даних та містить день, місяць та рік створення певного світу.

3.5 Розгортання кубу даних

Як вже було сказано у розділі 3.1 центральним поняттям OLAP є куб даних (гіперкуб). Для його розгортання було використано SSAS - SQL Server Analysis Services. Це служба, яка є частиною MS SQL Server і надає засоби для OLAP аналізу у середовищі BI.

Для початку було створено підключення до джерела даних. На рис. 23 – 26 наведено процес створення підключення до сховища даних.

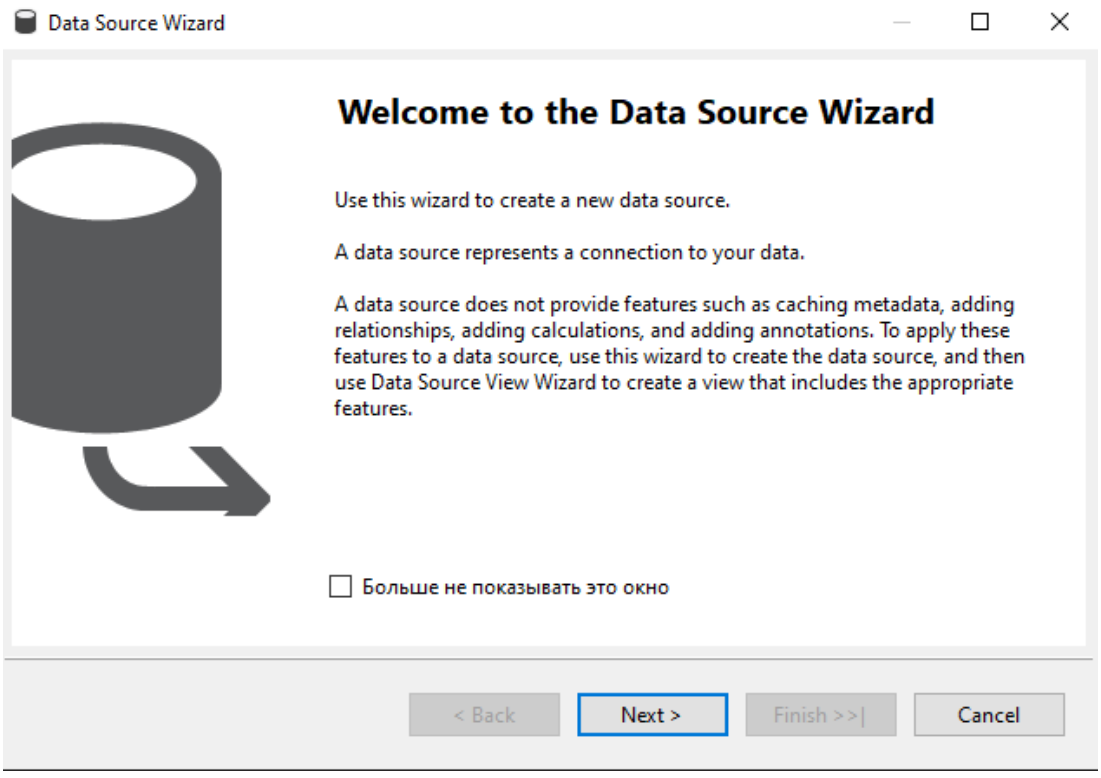


Рис. 23 Меню майстра підключень

Після натискання Next було обрано рядок підключення до сховища даних. Якщо рядка немає – його потрібно сформувати. В ході виконання роботи рядок

було сформовано для зручності використання у межах сервісів Reporting та SSAS.

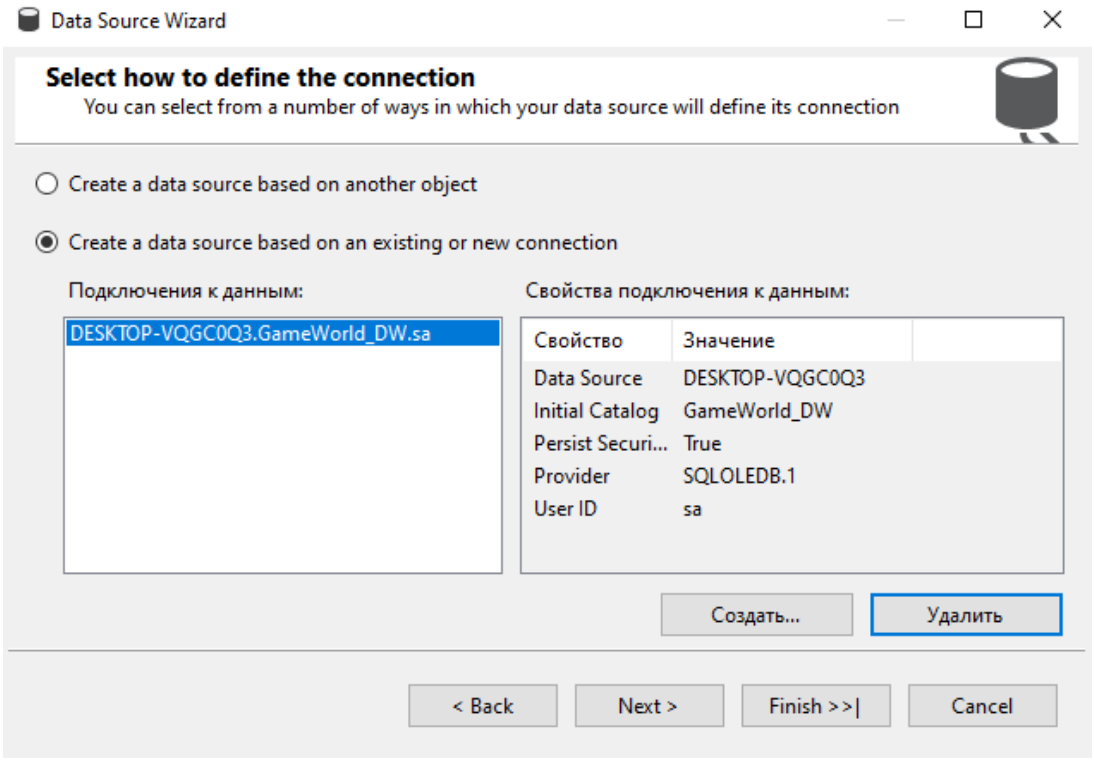


Рис. 24 Обрання підключення

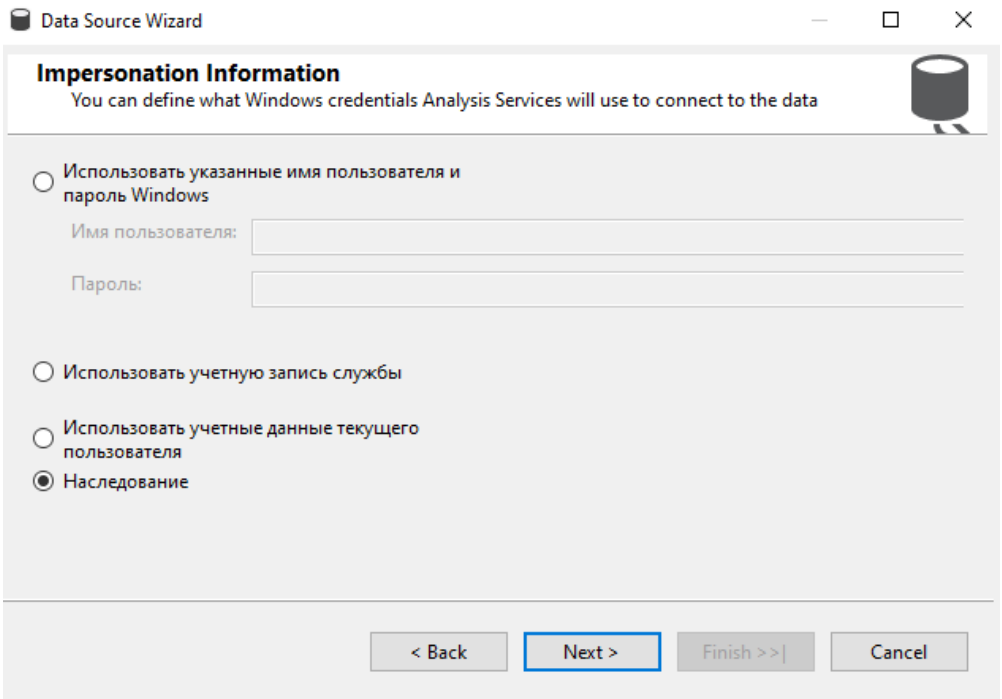


Рис. 25 Наслідування параметрів безпеки

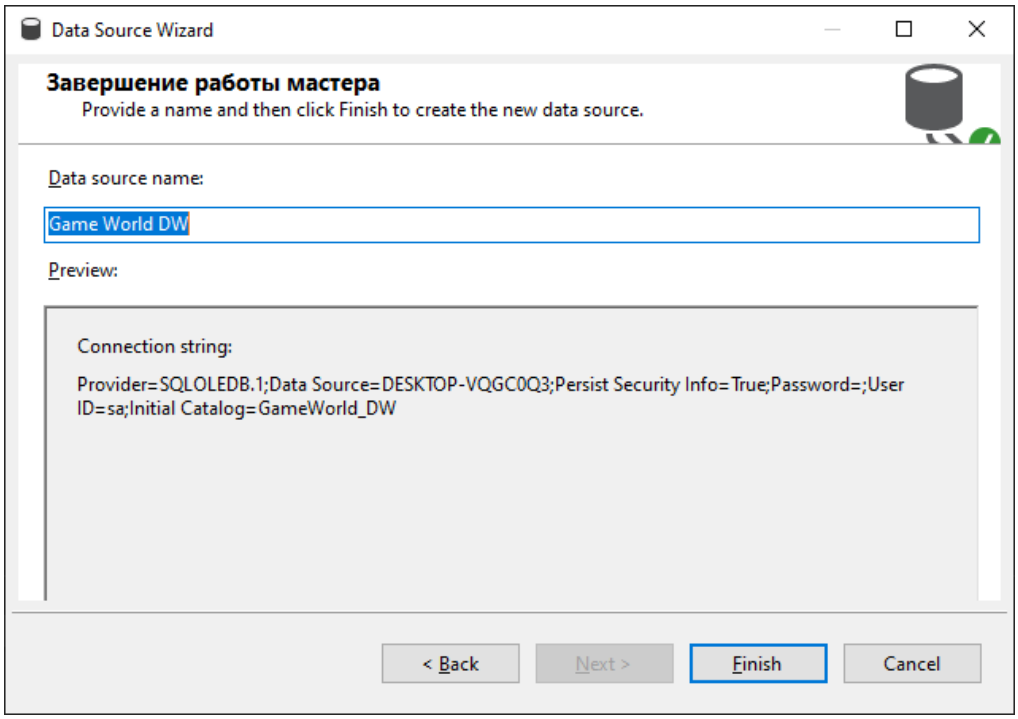


Рис. 26 Створення джерела даних

Після того як джерело даних створено на його основі було створено представлення. Для цього слід обрати джерело, що було створено раніше (рис.27).

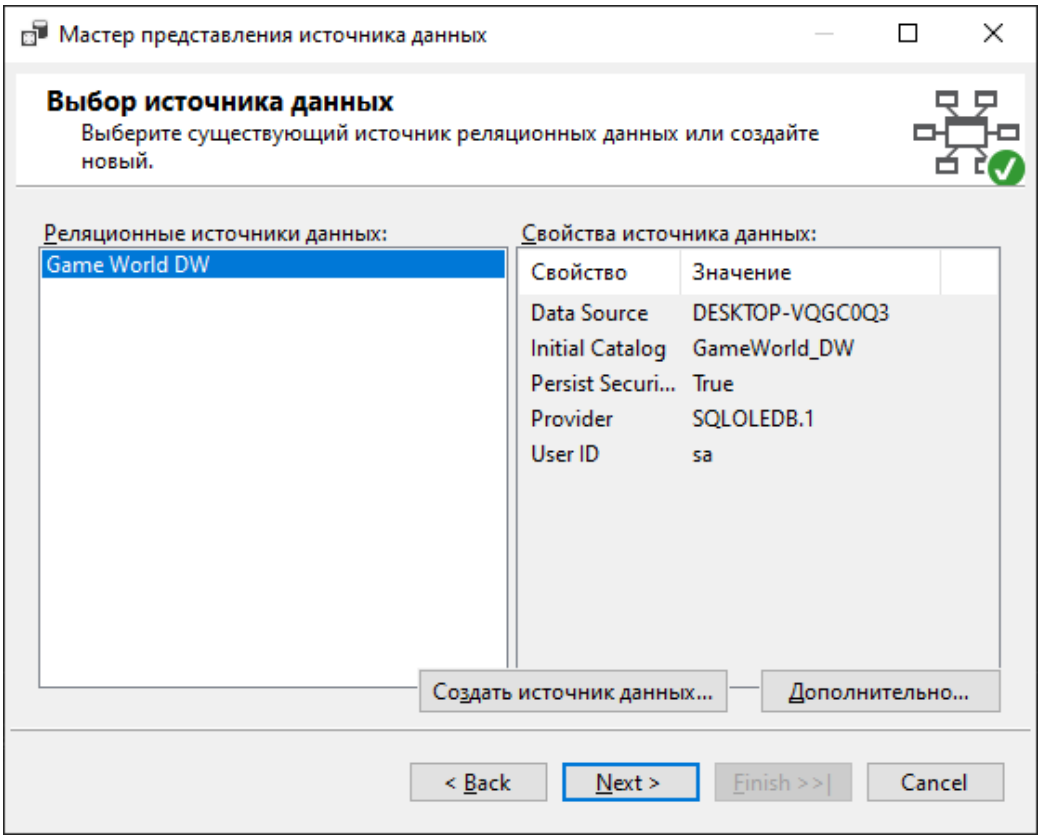


Рис. 27 Обрання джерела

Далі було обрано, які таблиці будуть використані у кубі. Таблиця системних діаграм не має відношення до теми і є сховищем схем бази даних, тож її не було включено (рис.28).

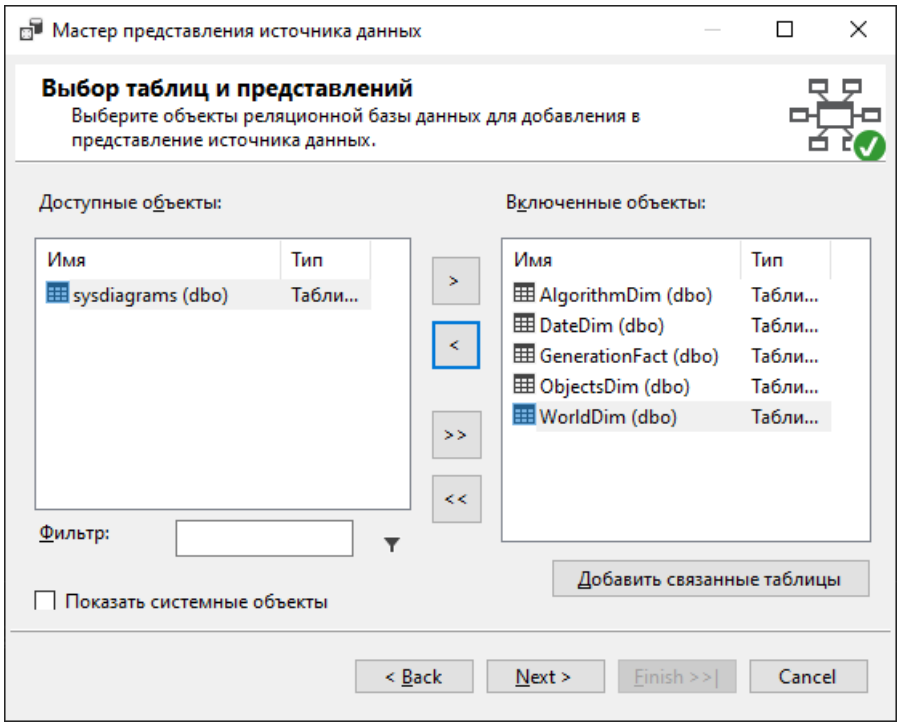


Рис. 28 Обрання таблиць

Обравши необхідні таблиці, роботу майстра було завершено (рис.29).

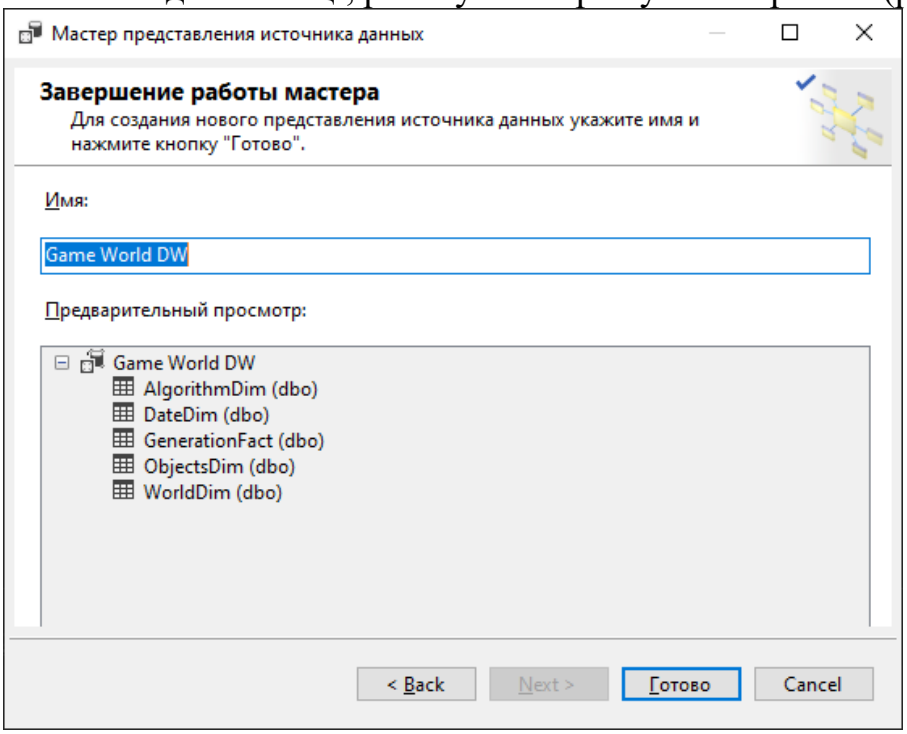


Рис. 29 Представлення створено

Для створення куба було використано раніше створене уявлення. Процес вибору таблиць зображено на рис. 30 – 31.

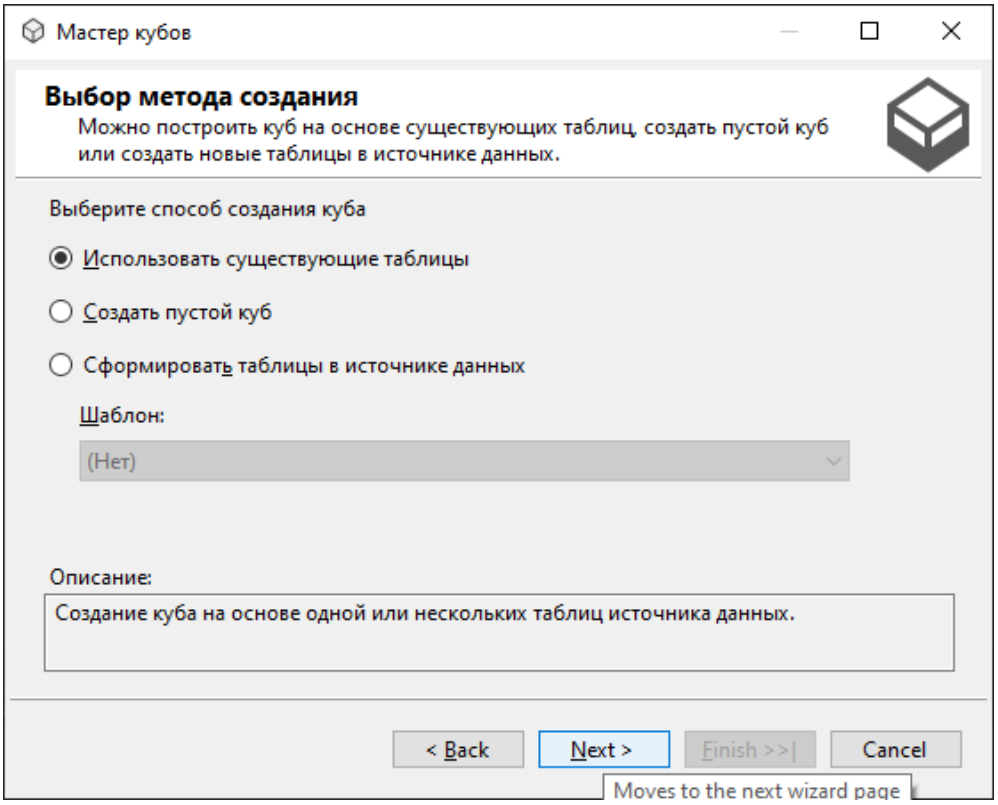


Рис. 30 Обрання вже існуючого представлення

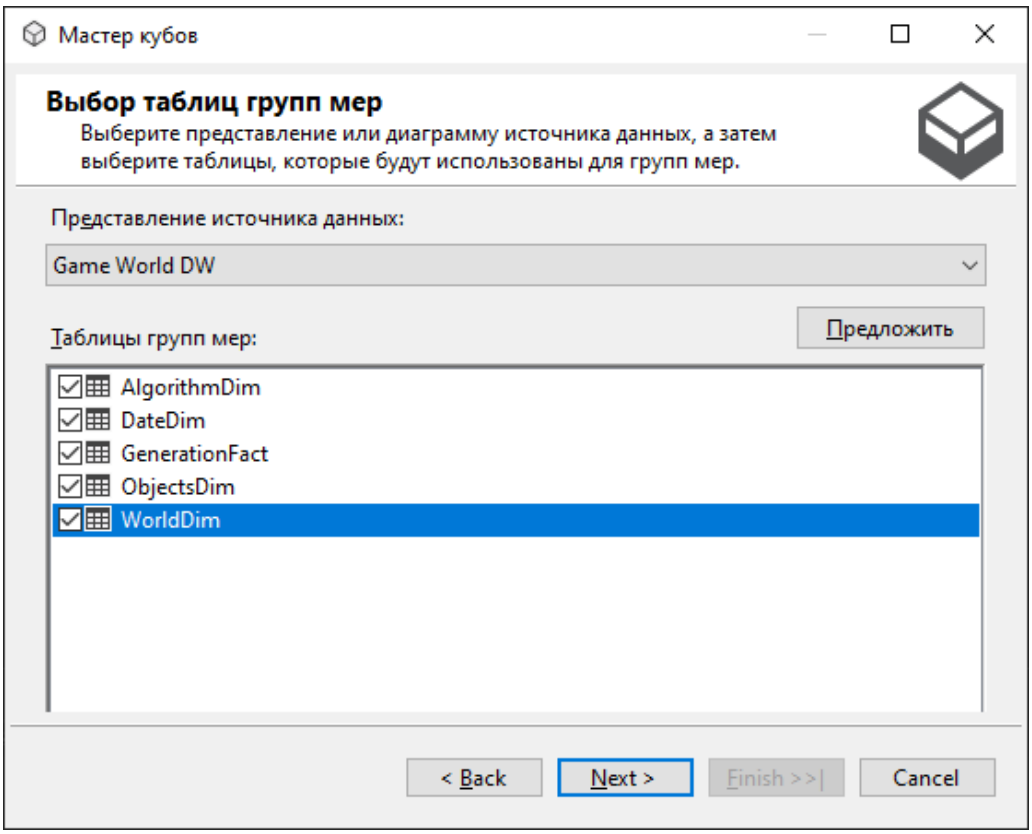


Рис. 31 Обрання таблиць

Далі було обрано виміри, які повинні бути у кубі (рис.32).

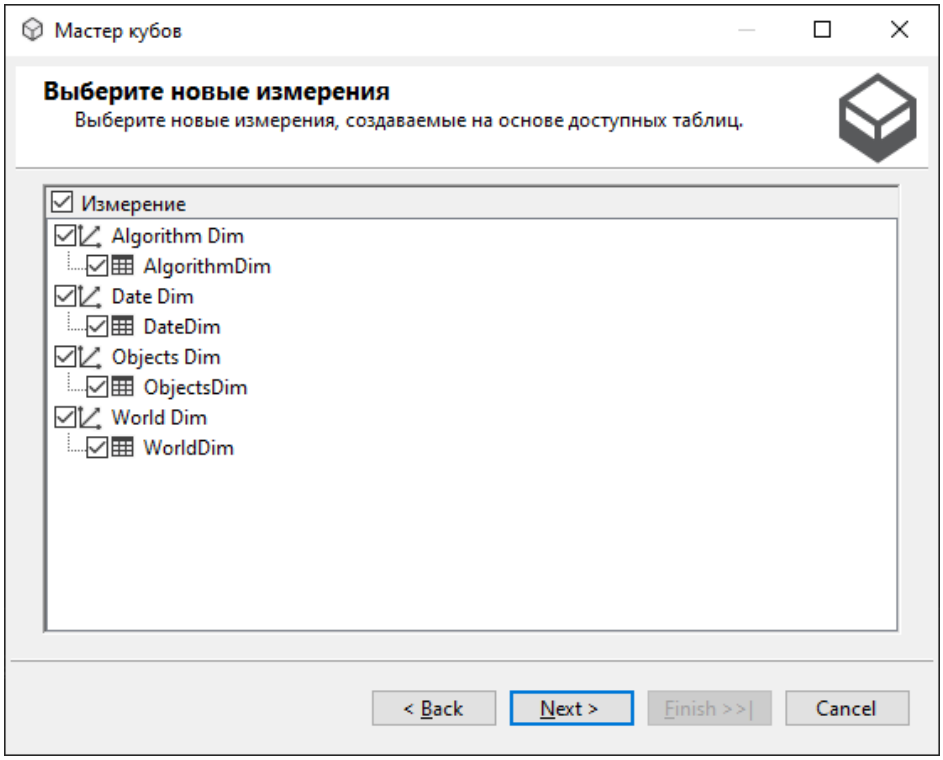


Рис. 32 Обрання вимірів кубу

На кінець куб було успішно розгорнуто. Структура куба зображена на рис. 33.

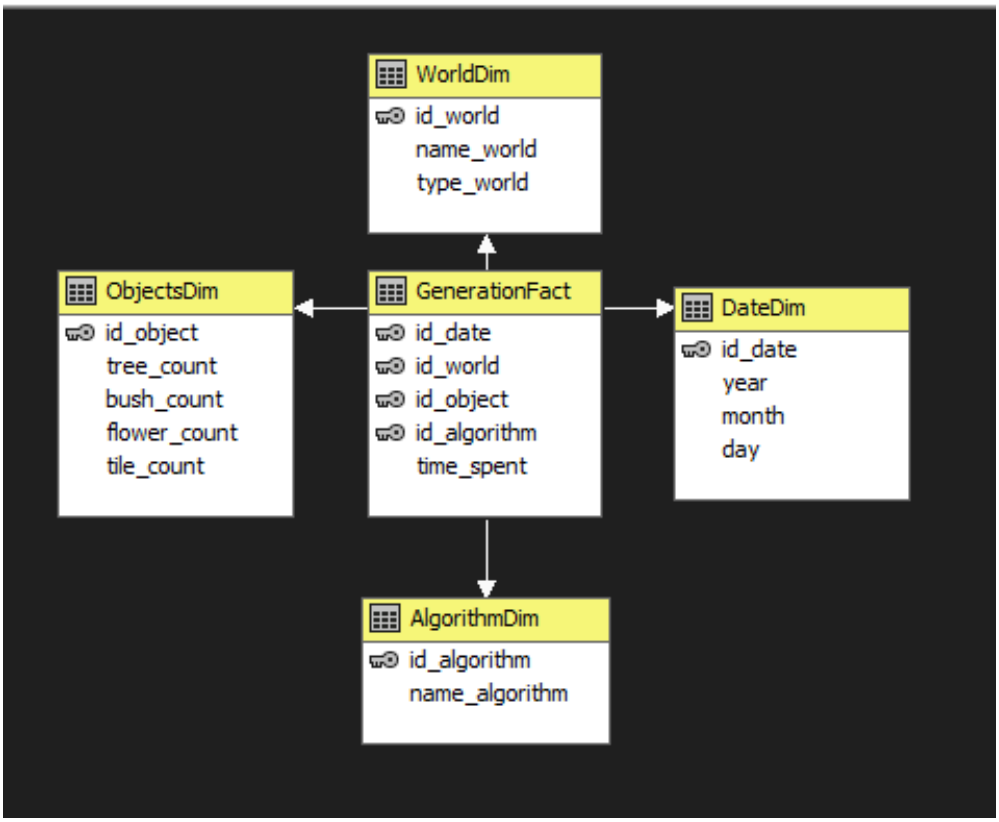


Рис. 33 Структура куба

3.6 Наповнення кубу даними

Для наповнення кубу даних було використано програму власної розробки, що забезпечує виконання принципів ETL. Можливість використання SSIS (SQL Server Integration Services) спочатку розглядалася, але в процесі роботи були виявлені його недоліки при роботі з JSON, тож вибір був зроблений на користь додатку власної розробки.

Розглянемо як саме працює додаток. Спочатку зчитуються дані з JSON файлу. Далі вибираються унікальні алгоритми (їх може бути 3 штуки), і заносяться у список, а далі вносяться як наповнення виміру «Алгоритм». Після цього аналогічним чином збираються унікальні записи дат і сортуються за зростанням (рис.34).

```
string jsonContent = File.ReadAllText(jsonFilePath);


var records = JsonConvert.DeserializeObject<List<Record>>(jsonContent);

// Збираємо унікальні назви алгоритмів із файлу
var uniqueAlgorithms = records.Select(record => record.name_algorithm).Distinct().ToList();


AddAlgorithms(uniqueAlgorithms);

// Збираємо унікальні дати та сортуємо їх за зростанням
var uniqueDates = records
    .Select(record => new { record.year, record.month, record.day })
    .Distinct()
    .OrderBy(date => new DateTime(int.Parse(date.year), int.Parse(date.month), int.Parse(date.day)))
    .ToList();
```

Рис. 34 Зчитування даних та відбір алгоритмів й дат

Далі проходячи по кожному запису вносяться відповідні дані про розмір світу, об’єкти і тд. Коли всі виміри заповнено й відповідні ідентифікатори отримано (ідентифікатор повертається як числове значення в результаті роботи функцій Add_Dim або Get_Id), заповнюється факт (рис. 35). Слід зазначити, що наповнення всіх вимірів (окрім дат й алгоритму) відбувається не повністю, а по ходу наповнення факту. Тобто зчитавши рядок, система фіксує алгоритм й дату (беручи ідентифікатор раніше внесених даних), вносить у виміри об’єктів та світу відповідні дані, і на кінець фіксує факт з відповідними ідентифікаторами.

```
foreach (var record in records)
{
    int idDate = GetDateId(record.year, record.month, record.day);

    int idWorld = AddWorldDim(record.name_world, record.type_world);
    int idObject = AddObjectsDim(record.tree_count, record.bush_count, record.flower_count, record.tile_co


    string selectedAlgorithm = record.name_algorithm;

    int idAlgorithm = GetAlgorithmId(selectedAlgorithm);

    AddGenerationFact(idDate, idWorld, idObject, idAlgorithm, record.time_spent);
}
```

Рис. 35 Наповнення інших вимірів й внесення факту

Внесення відбувається через рядок підключення та сервіс System.Data.SqlClient, який відповідає за взаємодію з SQL Server. Функція отримує параметри, за допомогою SqlCommand [21] формується відповідна команда з параметрами, і після цього успішно виконується (рис. 36).

```
// Функція для додавання запису в таблицю GenerationFact
Ссылок: 1
static void AddGenerationFact(int idDate, int idWorld, int idObject, int idAlgorithm, decimal timeSpent)
{
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        // Вставка фактного запису
        string query = "INSERT INTO GenerationFact (id_date, id_world, id_object, id_algorithm, time_spent) V
        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@idDate", idDate);
            cmd.Parameters.AddWithValue("@idWorld", idWorld);
            cmd.Parameters.AddWithValue("@idObject", idObject);
            cmd.Parameters.AddWithValue("@idAlgorithm", idAlgorithm);
            cmd.Parameters.AddWithValue("@timeSpent", timeSpent);

            cmd.ExecuteNonQuery();
        }
    }
}
```

Рис. 36 Вигляд функції AddGenerationFact

Після виконання дані будуть успішно внесені. Розглянемо як занесено дані на прикладі таблиці фактів (рис.37).

	id_date	id_world	id_object	id_algorithm	time_spent
1	1	1	1	1	3,25
2	2	2	2	2	5,73999977111816
3	3	3	3	2	4,44000005722046
4	4	4	4	2	1,30999994277954
5	5	5	5	3	3,25999999046326
6	6	6	6	1	3,35999989509583
7	7	7	7	3	3,44000005722046

Рис. 37 Внесені дані у таблицю фактів

4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Методи Data Mining

4.1.1 Кластеризація – це метод, що дозволяє проводити групування даних без попередньої інформації про їх параметри або належність. Основною ідеєю є об’єднати схожі об’єкти у кластери – тобто у групу елементів, максимально схожих між собою. В той час елементи з різних кластерів повинні бути відмінними між собою.

У межах роботи було використано алгоритм K – середніх (K – means). Цей метод базується на розділення об'єктів на кластери з певним центром - середнім значенням усіх об'єктів, що належать кластеру [22].

Особливістю методу є необхідність заздалегідь вказувати кількість кластерів для поділу. Для автоматизації пошуку оптимальної кількості кластерів було використано метод ліктя (elbow method). Основою методу є значення inertia – тобто значення суми квадратів відстаней усіх точок до відповідних центрів кластерів. Реалізацію методу зображено на рис. 38. Метод ліктя отримав свою назву через характерний перегин графіку, що чимось нагадує лікоть. Значення суми квадратів зменшується зі збільшенням кластерів, але в певний момент алгоритму темп зменшення цього значення сповільнюється. Саме ця точка, де було зафіксовано цей перегин і є оптимальним значенням кількості кластерів.

Для реалізації цього методу було використано мову Python та бібліотеку kneed. Ця бібліотека містить готову реалізацію методу, тож достатньо лише передати необхідні параметри у функцію [23].

Як видно з рис. 38 очевидний лікоть відбувається при значенні кількості кластерів рівній 2, отже саме цю кількість буде передано до методу K – середніх.

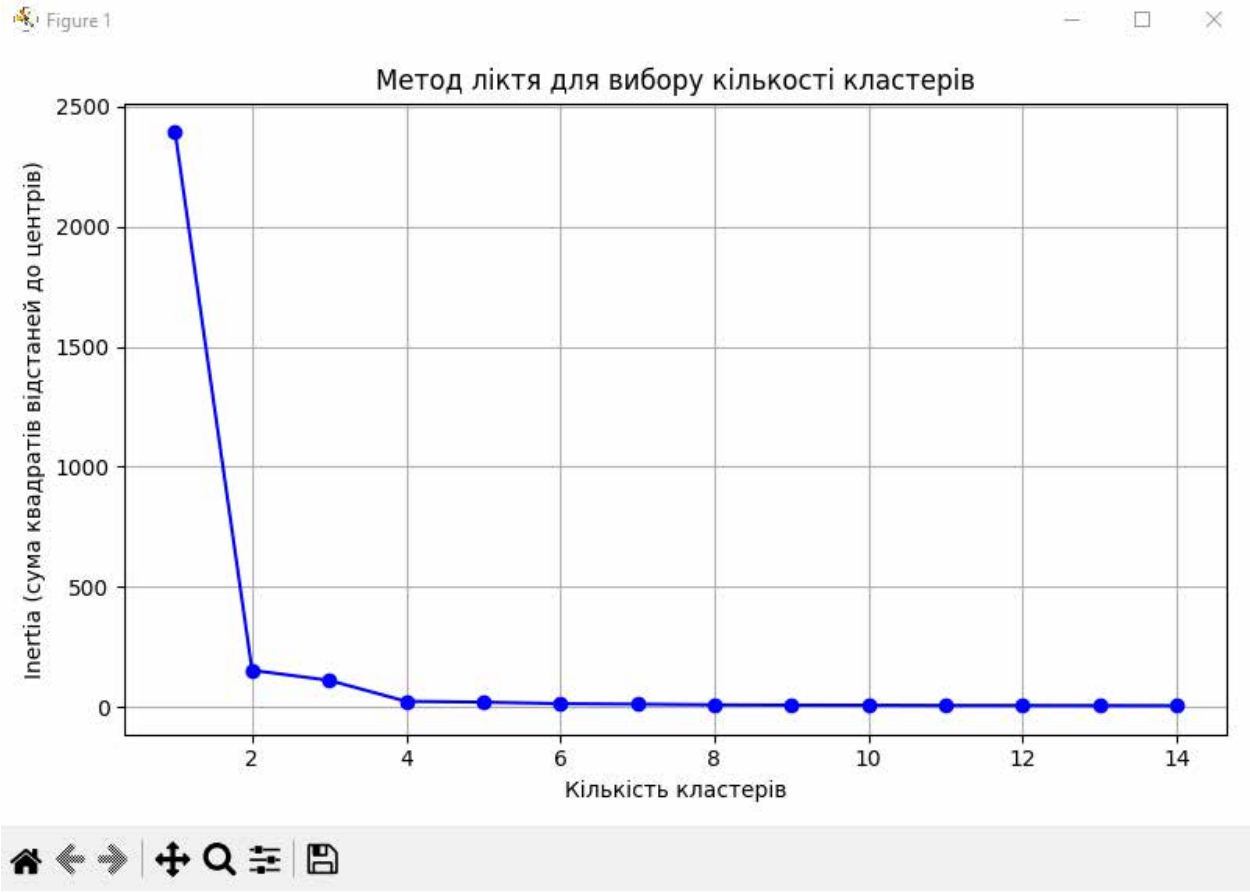


Рис. 38 Метод ліктя

Після того як кількість кластерів визначено, проводиться завантаження даних у компонент DataFrame та проводиться їх стандартизація. Після цього викликається функція що відповідає за метод К – середніх. Після цього результати кластеризації виводяться у вигляді графіка, що зображено на рис. 39. Хрестиком позначається центр кластеру, по осі оХ вказано кількість дерев, а по осу оУ кількість кущів. Два кластери позначено різними кольорами, а саме фіолетовим та жовтим.

На рис. 40 зображено більш деталізовані результати кластеризації. Показані результати кластеризації всіх світів, їх кількості дерев, кущів та тайлів, а також кластер, до якого віднесено саме цей конкретний світ. Нижче вказуються центри кластерів та їх параметри. Кластерам присвоєні цифрові коди 0 та 1, де 0 відповідає фіолетовому кольору на рис. 39, а 1 відповідає жовтому кольору.

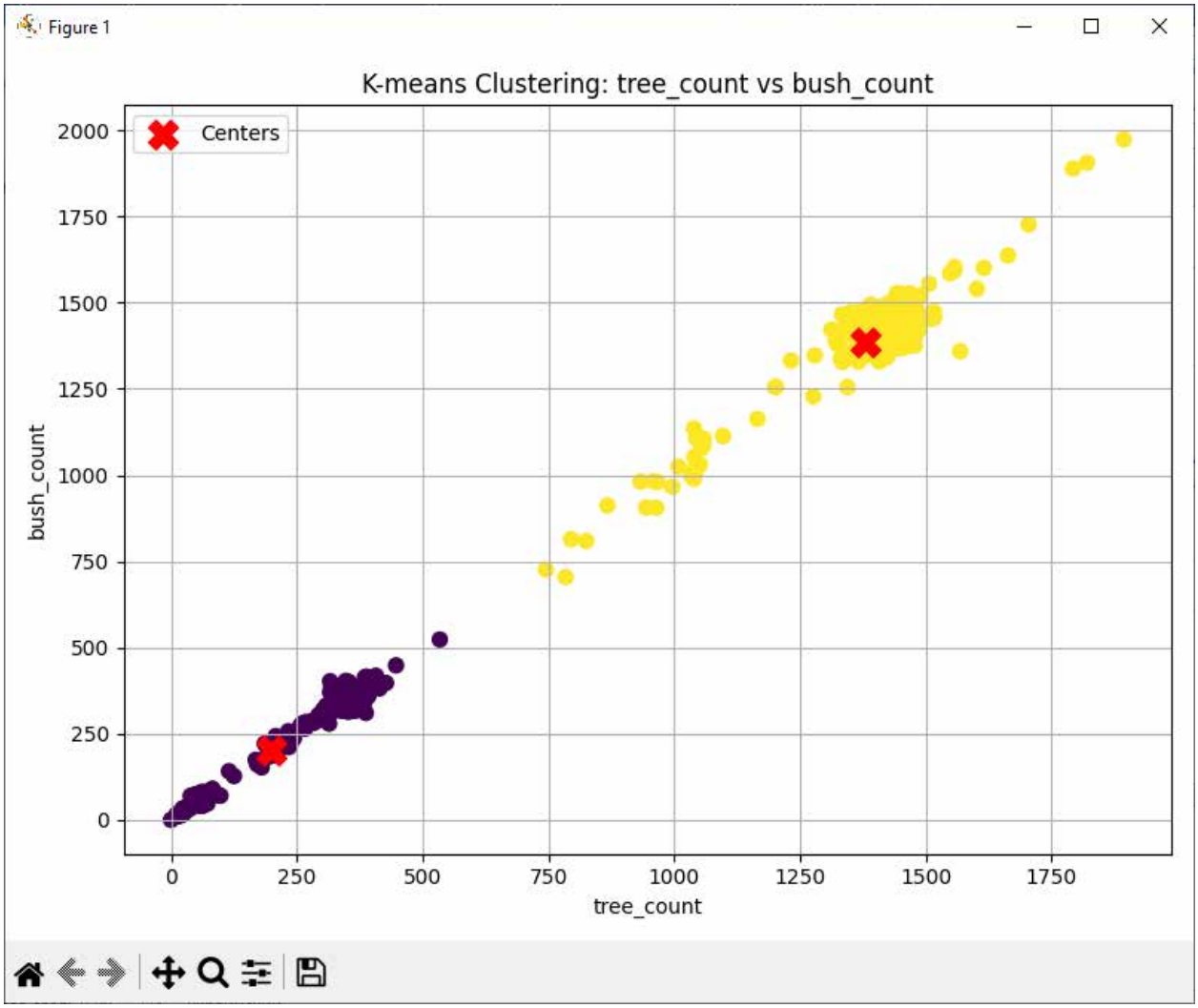


Рис. 39 Графічне представлення кластерів

```
C:\Users\Сергей\AppData\Local\Programs\Python\Python313\python.exe
575        1409        1384          712      250000        1
576        1379        1414          729      250000        1
577         347         403          166       62500        0
578        1097        1113          550      250000        1
579          54          68           20       10000        0
580          52          60           24       10000        0
581          50          53           28       10000        0
582         825         809          415      250000        1
583        1516        1473          740      250000        1
584        1058        1106          582      250000        1
585          63          57           21       10000        0
586          56          73           25       10000        0
587        1397        1411          729      250000        1
588        1435        1457          744      250000        1
589          59          80           28       10000        0
590          38          48           30       10000        0
591        1424        1489          751      250000        1
592         330         342          187       62500        0
593         336         331          192       62500        0
594          46          46           47       10000        0
595         365         341          179       62500        0
596          53          56           31       10000        0
597          57          59           32       10000        0
598        1601        1540          767      250000        1

Centers
    tree_count   bush_count  flower_count     tile_count  Cluster
0   198.380319   199.952128    101.630319   37307.180851        0
1  1378.825112  1384.632287    700.892377  250000.000000        1
```

Рис. 40 Результат кластеризації

Центр кластеру 0 має маленьку кількість кущів та інших об’єктів, у цей кластер потраплятимуть світи малого та середнього розміру (100*100 та 250*250 тайлів).

Центр кластеру 1 в свою чергу це строго світи великого розміру (500*500 тайлів), де є значна кількість об’єктів.

Результати на рис.40 з 575 світу до 598 дозволяють зробити припущення про те, що на час генерування впливає більше саме розмір світу, а не тип використаного алгоритму. Для підтвердження чи спростування цього припущення у наступних пунктах будуть застосовані методи 1Rule та Наївного Байеса.

4.1.2 Алгоритм 1Rule – це алгоритм, основна ідея якого полягає у створенні набору правил класифікації, що базуються лише на одному атрибуті. На відміну від алгоритму Наївного Байеса, який аналізує поєднання атрибутів, 1Rule аналізує строго вплив одного атрибуту [24].

На основі результатів кластеризації виділимо два класи для аналізу, а саме Висока та Низька швидкодія. Програма реалізована у вигляді Windows Forms додатка. При натисканні на кнопку Завантажити дані виконується завантаження даних за допомогою запиту SQL. Дані завантажуються в структуру у вигляді словника, де ключем є комбінація типу світу та алгоритму, а значенням — список тривалостей генерації. Також розраховується середнє значення часу генерації по всім записам.

Після цього проводиться аналіз ефективності генерації з двох боків — за алгоритмами та за типами світу. Для кожної групи (алгоритму або типу світу) підраховується кількість випадків, коли час генерації був меншим або більшим за загальний середній. На основі співвідношення цих кількостей відбувається класифікація на два класи: "Висока швидкодія" або "Низька швидкодія". Якщо переважає кількість випадків з меншою за середню тривалістю, призначається клас "Висока швидкодія", в іншому разі — "Низька швидкодія" (рис.41).

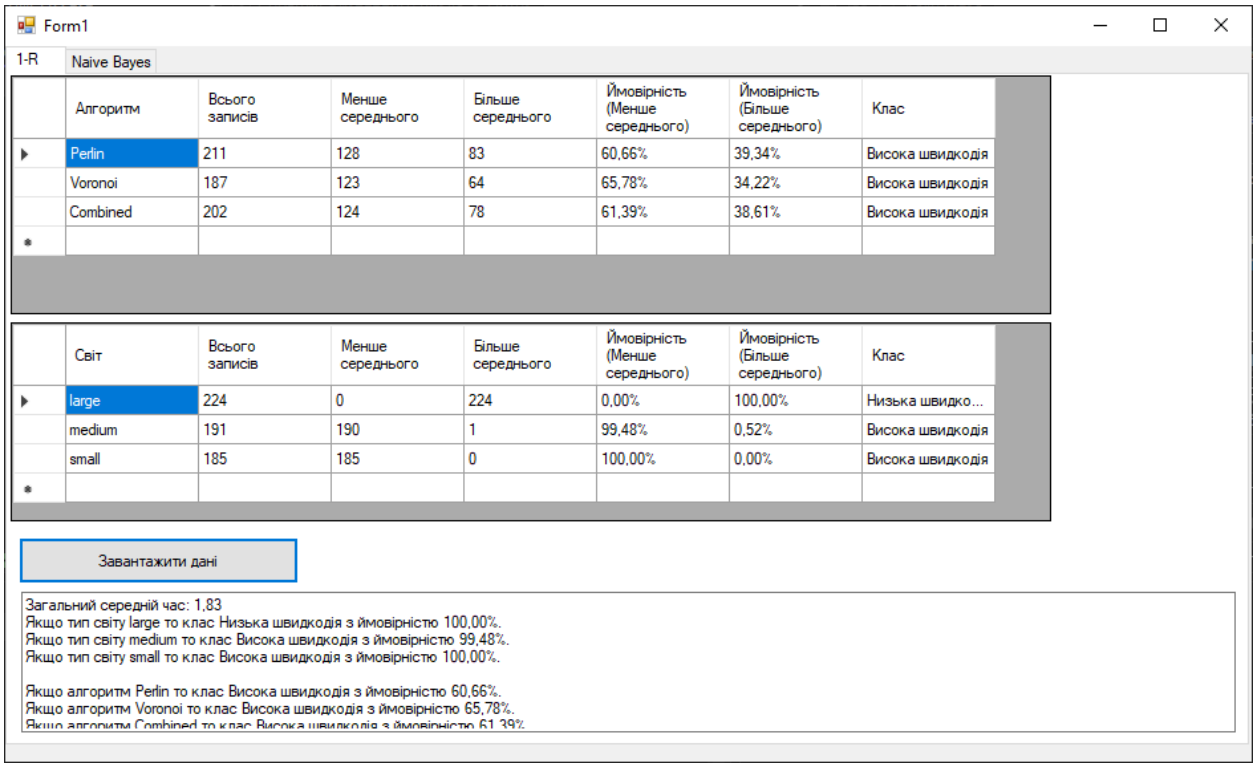


Рис. 41 Результат виконання алгоритму 1Rule

Тепер розглянемо результати з рис.41 детальніше та сформуємо правила класифікації на основі отриманих результатів. Для початку на основі інформації про світи сформуємо правила класифікації для змінної «Тип_світу»:

1) Якщо тип світу «large», то клас «Низька швидкодія» з ймовірністю 100,00%.
2) Якщо тип світу «medium», то клас «Висока швидкодія» з ймовірністю 99,48%.
3) Якщо тип світу «small», то клас «Висока швидкодія» з ймовірністю 100,00%.

Розглянувши правила можемо дійти висновку, що тип світу – основна змінна, що впливає на швидкодію. Великий розмір світу гарантовано дає нам клас Низька швидкодія, а маленький навпаки гарантує те, що клас буде «Висока швидкодія». Середній розмір світу також майже повністю гарантує клас «Висока швидкодія», але з малою (менше 1%) ймовірністю може відноситися до класу «Низька швидкодія».

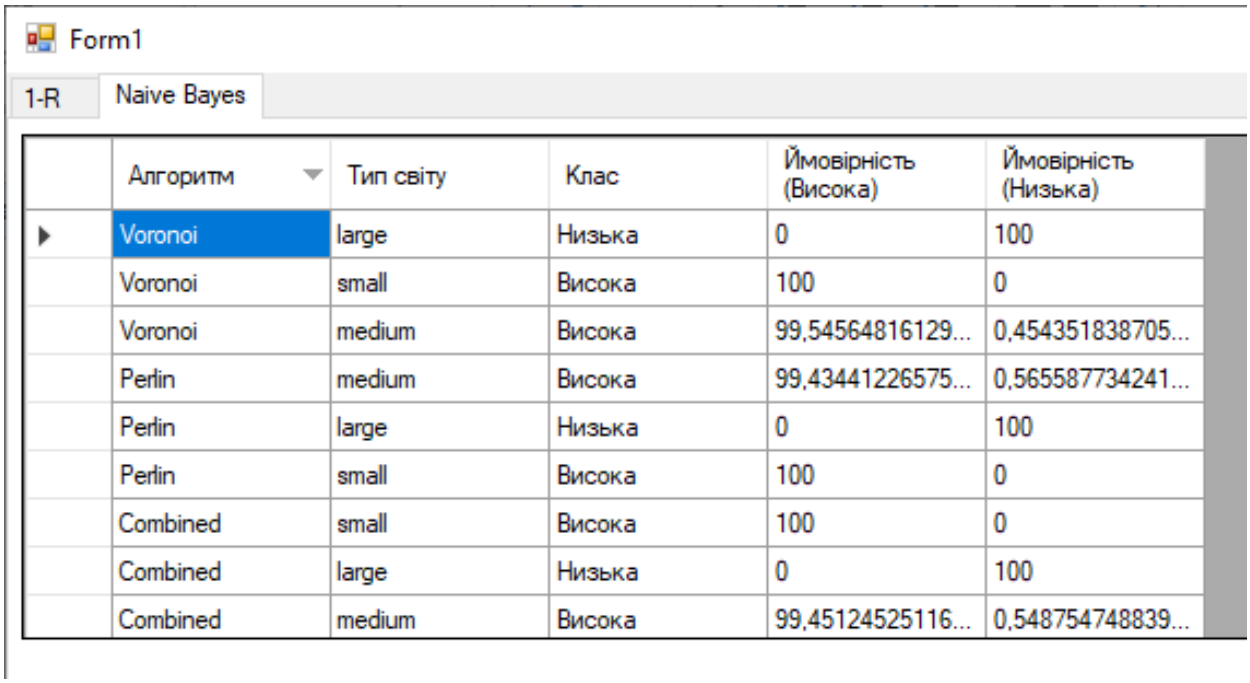
Тепер перейдемо до правил класифікації алгоритмів. Згідно отриманих результатів сформуємо наступні правила:

1) Якщо алгоритм «Perlin», то клас «Висока швидкодія» з ймовірністю 60,66%.
2) Якщо алгоритм «Voronoi», то клас «Висока швидкодія» з ймовірністю 65,78%.
3) Якщо алгоритм «Combined», то клас «Висока швидкодія» з ймовірністю 61,39%.

Як бачимо, всі алгоритми демонструють належність до класу Висока швидкодія, але з різними відсотками. Провівши аналіз цих результатів можна дійти висновку що алгоритм впливає на результат створення світу, але на такою мірою як тип світу, і порівняно з ним є набагато менше вагомою змінною, що підтверджує наше припущення про те, що саме розмір світу в першу чергу впливає на час генерування. Тож можна вже повноцінно сформувати гіпотезу: «Тип світу значно впливає на час генерування, у той час як алгоритм має набагато менший вплив».

4.1.3 Алгоритм Наївного Байеса – один з популярних алгоритмів статистичної класифікації, що базується на теоремі Байеса. Суть методу полягає у визначенні ймовірності належності об’єкту до певного класу на основі апріорних ймовірностей класів шляхом обчислення так званої апостеріорної ймовірності – належності до класу з урахуванням усіх наявних ознак [25].

Головна відмінність цього алгоритму від 1Rule полягає у тому, що аналізується не одна ознака, а комбінація ознак. У межах роботи цей метод було використано для перевірки належності вже пари «Тип_світу – Алгоритм» до класу продуктивності. Результатом роботи алгоритму є таблиця з поєднанням двох змінних (алгоритм та тип світу), та передбачений клас з ймовірністю належності (рис. 42). «Ймовірність (висока)» та «Ймовірність (низька)» – ймовірності належності комбінації змінних вказаному класу («Висока швидкодія» або «Низька швидкодія»).



Алгоритм	Тип світу	Клас	Ймовірність (Висока)	Ймовірність (Низька)
Voronoi	large	Низька	0	100
Voronoi	small	Висока	100	0
Voronoi	medium	Висока	99,54564816129...	0,454351838705...
Perlin	medium	Висока	99,43441226575...	0,565587734241...
Perlin	large	Низька	0	100
Perlin	small	Висока	100	0
Combined	small	Висока	100	0
Combined	large	Низька	0	100
Combined	medium	Висока	99,45124525116...	0,548754748839...

Рис. 42 Результати алгоритму Наївного Байеса

Тепер проведемо аналіз результатів класифікації з рис 42. Оберемо конкретну комбінацію змінних та проаналізуємо отриманий результат.

Представимо результати для діаграм Вороного та усіх типів світу. Як видно при типі світу small алгоритм з ймовірністю 100% покаже високу швидкодію (рис. 42). Це є коректним, бо середнє значення усіх записів складає приблизно 1,8 с., у той час як найбільший зафіксований час генерації малого світу є 1 секунда. І аналогічно для великого розміру світу, де найменший час склав 2,55 секунди. Для середнього розміру світу даний алгоритм показує високу швидкодію з майже 100% ймовірністю (99,5%)

Алгоритм шуму Перліна показує майже аналогічні результати з 100% ймовірністю високої швидкодії при малому розмірі світу і 100% ймовірністю низької швидкодії при великому розмірі світу.

Тобто такий результат підтверджує гіпотезу, сформовані у ході розгляду результатів 1R про те, що алгоритм не є головною змінною, що впливає на час генерування. І даний аналіз також підтвердив, що основною (значущою) змінною є саме тип світу, від якого в більшій мірі залежить час генерування та відповідно клас швидкодії.

4.2 Засоби OLAP

З OLAP аналізу для роботи було використано наступні засоби : KPI та звіти. KPI дозволять краще зрозуміти часові рамки світів, що генеруються. Звіти ж дозволять отримати відповіді на питання, що були поставлені у Постановці завдання. Для KPI було використано SSAS, а для звітів службу Reporting.

Код для всіх KPI надано у Додатку В, а коди звітів у Додатку Г.

4.2.1 KPI KPI – це кількісні метрики, що використовуються для оцінки ефективності певних процесів [26]. Для системи генерування ігрового світу було сформовано 3 основних KPI, які дозволяють оцінити ефективність генерування в розрізі часу (рис. 43).

Рис. 43 Результат виконання KPI

KPI_Max/MinTimeSpent – показники, що демонструють максимальний та мінімальний час, витрачений на генерацію за певний проміжок часу. Ці показники дозволяють зрозуміти, у яких рамках лежать часові показники створених світів та допоможуть звернути увагу на те, що саме потребує оптимізації або де наявні помилки.

KPI_AVGTime – показник, що відображає середній час генерування. Є базовим параметром, від якого розраховуються інші KPI та який дозволяє зрозуміти загальну тенденцію часових витрат.

Що можна стверджувати, провівши аналіз цих KPI? Середнє значення є меншим за 2 секунди, що можна вважати гарним результатом для маленьких ігор з генерацією світу. Мінімальний час розрахуємо як середній час відняти 1,5 секунди. Це буде найкращий результат для маленького світу, і як бачимо він рівний 0,08 секунди, що є чудовим результатом. Але максимальний час ставимо за ціль середній час додати 5 секунд, і отримуємо результат, що перевищує

очікування майже у 2 рази. Це може свідчити про певну помилку оптимізації створення великих світів, на що і треба буде звернути увагу в першу чергу.

Власне саме за допомогою цих KPI можна побачити прірву між маленьким світом, що найшвидше було згенеровано за менше ніж одну десяту секунди, і найдовшим результатом аж в 11 секунд.

4.2.2 Звіти – основа будь-якої аналітики. Вони дозволяють у зрозумілій та наочній формі відображати необхідні картинки бізнес-процесів. Саме завдяки коректно оформленим звітам керівники можуть приймати зважені й обґрунтовані рішення відносно проєкту [27].

Для системи генерування ігрового світу було створено низку звітів, кожен з яких буде детально описано пізніше. Для побудови всіх звітів було використано службу Reporting [28].

Перший звіт дає відповідь на питання «Яка масова частка використання кожного алгоритму?» (рис. 44). Даний звіт є важливим для розуміння поведінки користувача, тобто для розуміння, який алгоритм найчастіше обирали гравці. Це дозволить зрозуміти, на якому алгоритмі треба зосередитися. Алгоритм-лідер – провести аналіз на необхідність покращень та за можливості внести їх, відстаючий за часткою – провести аналіз, чому саме його обирають менш за все і внести оптимізаційні/інші правки в роботу алгоритму.

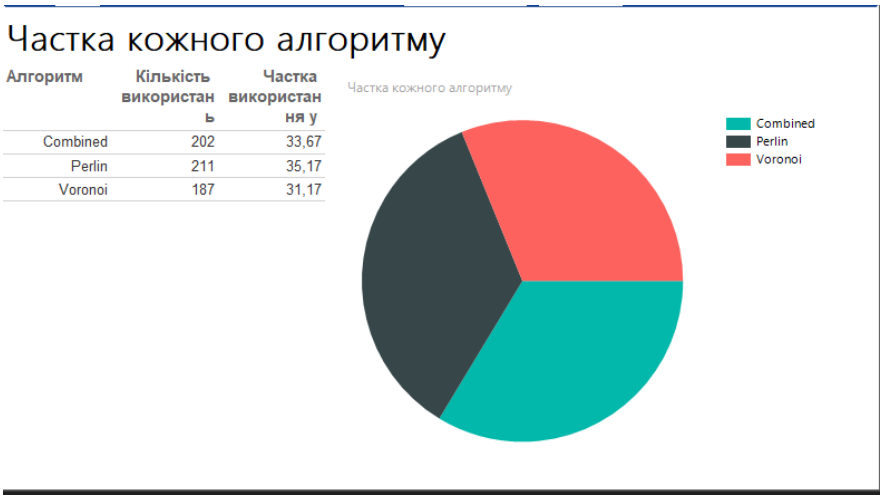


Рис. 44 Звіт по частці алгоритмів

Як видно з рис. 44, лідера або відстаючого алгоритму немає, частка лежить у межах третини для кожного з трьох алгоритмів.

Другий звіт дає відповідь на питання «Яка середня кількість витраченого часу на генерування за вказаний період?» (рис. 45). Цей звіт є також доволі важливим для розуміння середніх часових затрат кожного алгоритму. Результати також зручно виводяться у форматі діаграми.

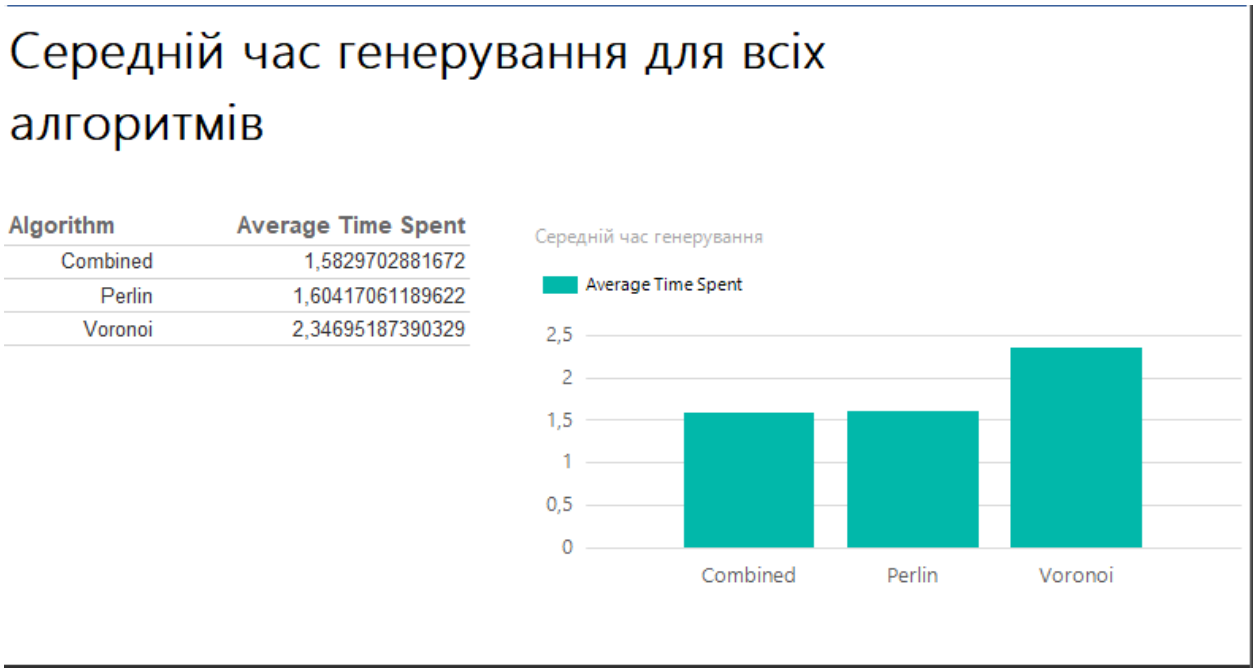


Рис. 45 Звіт про середній час генерування

Розглянемо детальніше результати цього звіту (рис.45). Видно, що комбінований алгоритм та шум Перліна мають відносно схожі результати (1.5 сек), у той час як діаграми Вороного показують середній час більше за 2 секунди (2,34). Це можна пояснити тим, що діаграми Вороного мають більшу математичну складність і по іншому створюють ландшафт, тож і вимагають більше часу.

Цей звіт підтверджує частину гіпотези про те, що вибір алгоритму впливає на результуючий час, і дозволяє оцінити відсоток впливу.

Третій звіт дає відповідь на питання «Скільки в середньому об’єктів було згенеровано кожним алгоритмом?» (рис.46). Цей звіт дозволяє зрозуміти поводження алгоритмів з об’єктами та зрозуміти, скільки в середньому об’єктів було згенеровано кожним алгоритмом.

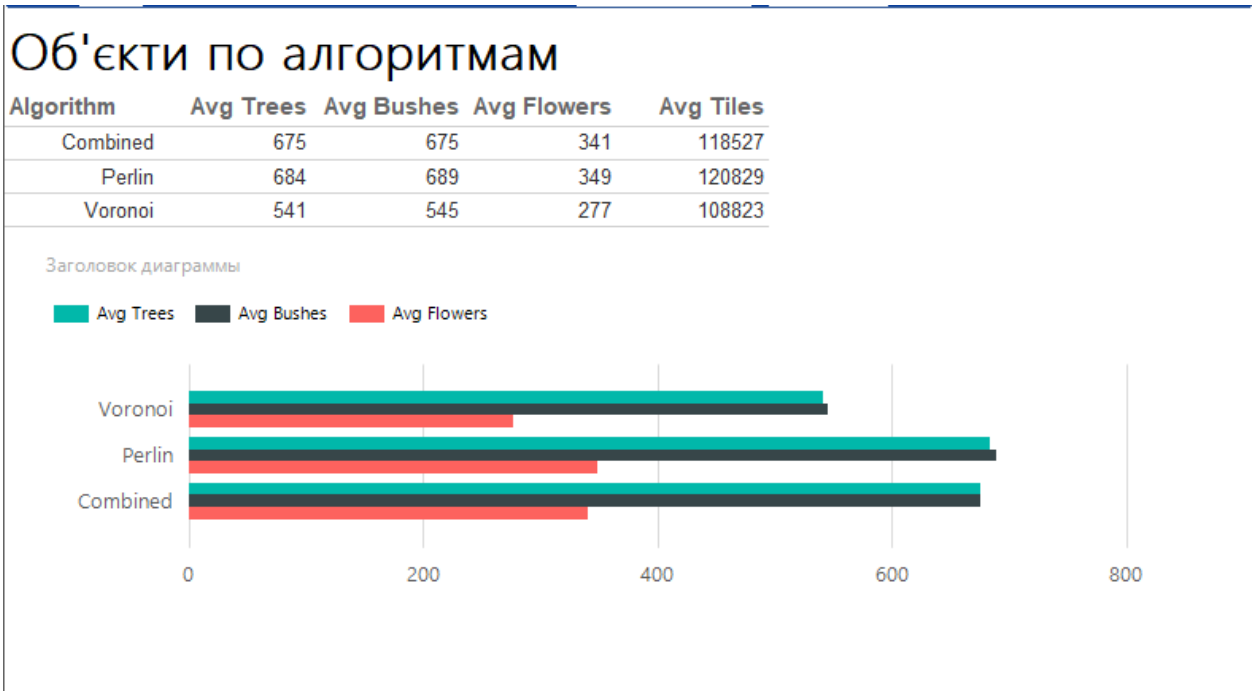


Рис. 46 Звіт по середній кількості згенерованих об'єктів

З рис. 46 видно, що по кількості об’єктів лідирує алгоритм шуму Перліна. Цей алгоритм як було виявлено у другому звіті також показує непоганий часовий результат. А ось діаграми Вороного показали найменшу кількість об’єктів. Це можна пояснити тим, як саме алгоритм генерує світ, бо даний алгоритм має набагато більшу частку місцин, де нічого не росте (пісок, вода). Але слід зазначити, що хоча об’єктів створюється найменше – час створення в середньому є більшим, тож можна дійти висновку про те, що саме особливість роботи алгоритму викликає такі результати.

Четвертий звіт схожий з минулим, але вже дозволяє оцінити загальну кількість створених об’єктів. Він доповнює розуміння того, у якій кількості алгоритми генерують об’єкти. Результати для кращого візуального сприйняття представлені у вигляді двох діаграм (рис. 47-48).

Як видно з рис. 47, тенденція до того що діаграми Вороного генерують менше об’єктів, зберігається. Різниця є доволі суттєвою, але це зумовлено саме тим, як цей алгоритм генерує світ у вигляді областей.

Algorithm	Total Trees	Total Bushes	Total Flowers	Total Tiles
Combined	136420	136531	68968	23942500
Perlin	144469	145391	73675	25495000
Voronoi	101180	102033	51869	20350000

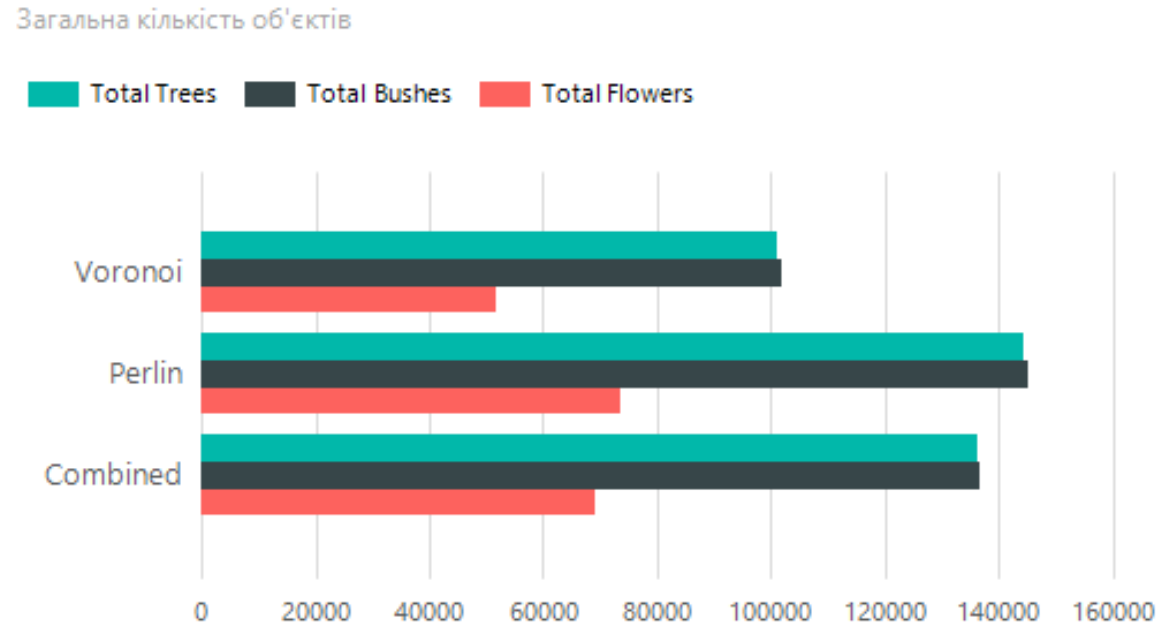


Рис. 47 Звіт по загальній кількості об'єктів (частина 1)

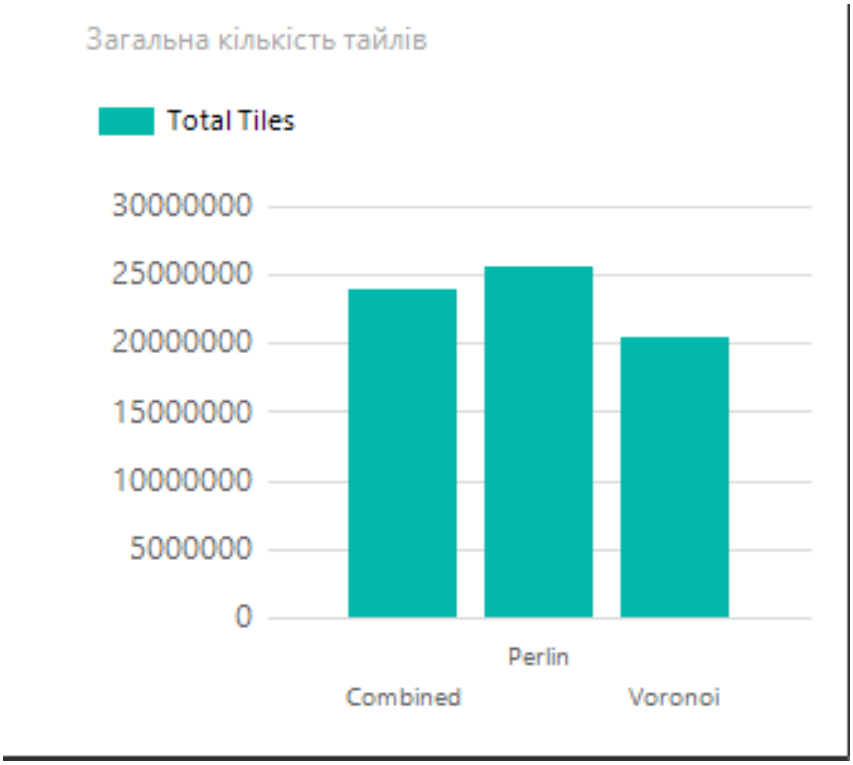


Рис. 48 Звіт по загальній кількості об'єктів (частина 2)

Останній звіт дає розуміння середнього часу генерування в контексті пари алгоритм – тип світу. Цей звіт дозволяє візуально представити знання, отримані від методу Наївного Байеса.

Рис. 49 Звіт по затраченому часу в розрізі Алгоритм - Тип світу

Як видно з рис. 49, алгоритми поєднаний та шум Перліна не дуже відрізняються по часу, в той час як діаграми Вороного справді показують себе гірше у будь якому випадку. Але цей звіт дозволяє наочно побачити, наскільки сильно розмір світу впливає на час генерування. Якщо різниця між шумом Перліна та діаграмами Вороного складає близько 30%, то в розрізі одного алгоритму різниця в залежності від розміру світу може сягати аж 700%. Діаграми Вороного в середньому витрачали 0,11с на малий тип світу, а на середній витрачалося вже 0,78с, що є в 7 разів більше.

Результати цього звіту повністю підтверджують сформовану гіпотезу: «Тип світу значно впливає на час генерування, у той час як алгоритм має набагато менший вплив». Тип світу може збільшувати час створення аж до 7 разів, в той час як алгоритм може збільшити час на приблизно 30-40%.

ВИСНОВКИ

У ході виконання магістерської дипломної роботи було проведено моделювання предметної області та розроблено систему генерування ігрового світу. Розробка проводилася у три етапи, а саме:

1) Моделювання предметної області.
2) Проектування архітектури системи.
3) Розробка системи та аналіз результатів.

Було розроблено сховище даних у середовищі MS SQL Server, яке повністю відповідало вимогам роботи.

За допомогою методу ліктя було проведено підготовку до кластеризації. Було виявлено, що оптимальною кількістю кластерів є 2 кластери.

На основі результатів методу ліктя було застосовано метод кластеризації К – середніх, яким було проведено кластеризацію світів на основі кількості об'єктів. У результаті було отримано два кластери : кластер маленьких та середніх світів та кластер великих світів. Також було висунуто припущення про те, що на час генерування впливає більше саме розмір світу, а не тип використаного алгоритму.

За допомогою алгоритму 1Rule було проведено класифікацію за двома типами змінних окремо – тип світу та алгоритм. Класифікація проводилася на основі результатів кластеризації, виділивши два класи – «Висока швидкодія» та «Низька швидкодія». Результатом дослідження стала класифікація всіх алгоритмів на Висока швидкодія, а тип світу показав наступні результати:

1) Якщо тип світу «large», то клас «Низька швидкодія» з ймовірністю 100,00%.
2) Якщо тип світу «medium», то клас «Висока швидкодія» з ймовірністю 99,48%.
3) Якщо тип світу «small», то клас «Висока швидкодія» з ймовірністю 100,00%.

Ці результати підтверджують сформоване на етапі кластеризації припущення, даючи змогу повноцінно сформувати гіпотезу : «Тип світу значно впливає на час генерування, у той час як алгоритм має набагато менший вплив».

За допомогою алгоритму Наївного Байеса було отримано додаткове підтвердження сформованої гіпотези, бо всі пари алгоритм – великий світ показували низьку швидкодію, в той час як всі пари алгоритм – малий світ показувати високу швидкодію. Це дозволило стверджувати, що саме тип світу (малий, середній, великий) є головною змінною, що впливає на час генерації.

За результатами розрахунку KPI стало відомо середній час генерування, а також максимальне та мінімальне значення часу. Розрив між максимальним і мінімальним часом є доволі великим, тож це показує як час змінюється в залежності від типу світу.

За результатами звітів було отримано відповіді на питання, що висувалися у постановці завдання. Звіти також дозволили зрозуміти, що алгоритм діаграм Вороного не тільки є повільнішим за інші, а й також генерує менше об’єктів за більший час. Також було наочно продемонстровано залежність часу генерації від розміру світу, де значення могли відрізнятися до 7 разів.

Результати проведеного дослідження дають змогу краще зрозуміти поведінку генеративних алгоритмів у ігровій розробці. На основі отриманих даних розробники можуть краще проводити оптимізацію створення ігрового світу, що дозволить знизити часові затрати на процес генерування. Отже, можна зробити висновок що мета роботи була досягнута в повному обсязі.

ДЖЕРЕЛА

1) Що таке шум Перліна та як його використовувати при створенні ігор. URL: https://gamedev.dou.ua/articles/mathematics-gamedev-perlin-noise/ (дата звернення: 14.09.2025).
2) Perlin Noise: Implementation, Procedural Generation, and Simplex Noise. URL: https://garagefarm.net/blog/perlin-noise-implementation-procedural-generation-and-simplex-noise (дата звернення: 14.09.2025).
3) Scher Y. Playing with Perlin Noise: Generating Realistic Archipelagos. *Medium*. URL: https://medium.com/@yvanscher/playing-with-perlin-noise-generating-realistic-archipelagos-b59f004d8401 (дата звернення: 14.09.2025).
4) Bellelli F. The fascinating world of Voronoi diagrams. *Medium*. URL: https://medium.com/data-science/the-fascinating-world-of-voronoi-diagrams-da8fc700fa1b (дата звернення: 15.09.2025).
5) International Symposium on Voronoi Diagrams in Science and Engineering (ISVD). URL: https://ieeexplore.ieee.org/xpl/conhome/1001201/all-proceedings (дата звернення: 15.09.2025).
6) Діаграма Вороного. *Aspose Documentation*. URL: https://docs.aspose.com/gis/uk/net/geo-tools/voronoi-diagram/ (дата звернення: 15.09.2025).
7) Математика в геймдеві. Що таке діаграми Вороного та як їх використовують при розробці ігор. URL: https://gamedev.dou.ua/articles/mathematics-gamedev-voronoi-diagrams/ (дата звернення: 15.09.2025).
8) Owens B. How to Use Voronoi Diagrams to Control AI | Envato Tuts+. *Code Envato Tuts+*. URL: https://code.tutsplus.com/how-to-use-voronoi-diagrams-to-control-ai--gamedev-11778t (дата звернення: 15.09.2025).

9) Iron Gate AB. Valheim. Coffee Stain Publishing, 2021. URL: https://store.steampowered.com/app/892970/Valheim/ (дата звернення: 15.09.2025).
10) Pugstorm. Core Keeper. Fireshine Games, 2024. URL: https://store.steampowered.com/app/1621690/Core_Keeper/?l=ukrainian (дата звернення: 15.09.2025).
11) Стеценко, І.В. Моделювання систем: навч. посіб. [Електронний ресурс, текст] / І.В. Стеценко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2010. – 399 с.
12) Основні підходи до моделювання інформаційно-вимірювальних систем. *Житомирська Політехніка*. URL: https://learn.ztu.edu.ua/pluginfile.php/114218/mod_resource/content/0/Лекція%206%20OMIBC.pdf (дата звернення: 16.09.2025).
13) Maxym Zosym. Уніфікована мова моделювання (Unified Modeling Language - UML). *Maxym Zosym*. URL: https://www.maxzosim.com/unifikovana-mova-modeluvannia/ (дата звернення: 16.09.2025).
14) Моралес Дж. In-depth Knowledge of UML Use Case Diagram: with Tutorial. *MindOnMap | Free Mind Mapping Tool to Draw Ideas Easily Online*. URL: https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram/ (дата звернення: 16.09.2025).
15) Діаграма послідовності (Sequence Diagrams). URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 16.09.2025).
16) Дідковська М. Проектування програмного забезпечення засобами UML. *Кафедра ММСА*. URL: http://mmsa.kpi.ua/sites/default/files/disciplines/Розробка%20і%20тестування%20програм/didkovska_m_v_testing_lecture_5.pdf (дата звернення: 16.09.2025).

17) Діаграми UML для моделювання процесів і архітектури проекту. *Evergreen*. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 16.09.2025).

18) Що таке OLAP. URL: https://data-life-ua.com/db/shcho-take-olap/ (дата звернення: 16.09.2025).

19) Голуб Б. Л. Data Mining - основні положення. Київ, 2021. 12 с. URL: https://elearn.nubip.edu.ua/pluginfile.php/102629/mod_resource/content/3/DATA%20MINING%20–%20початок.pdf (дата звернення: 16.09.2025).

20) Городецька В.В., Левківський В.В., Вакалюк Т.А. Що таке ETL і для чого це потрібно. *Оптимізація промальовування вебзастосунку на основі об'єктів з глибокою вкладеністю та багатозалежними зв'язками*, м. Житомир. С. 1.

21) SqlCommand Class (System.Data.SqlClient). URL: https://learn.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqlcommand?view=netframework-4.8.1 (дата звернення: 16.09.2025).

22) Стандартні методи кластеризації даних. *Факультет комп'ютерних наук та кібернетики*. URL: https://csc.knu.ua/media/study/asp/mod_probl_inf_tech_sys_analysis_ivohin/lecture/lec2.pdf (дата звернення: 16.09.2025).

23) kneed 0.8.5 documentation. URL: https://kneed.readthedocs.io/en/stable/ (дата звернення: 16.09.2025).

24) Learn-One-Rule Algorithm - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/machine-learning/learn-one-rule-algorithm/ (дата звернення: 16.09.2025).

25) Naive Bayes Classifiers - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/machine-learning/naive-bayes-classifiers/ (дата звернення: 16.09.2025).

26) Що таке ключовий показник ефективності?. URL: https://peopleforce.io/uk/hr-glossary/key-performance-indicator (дата звернення: 17.09.2025).

27) SQL Server Reporting Services (SSRS) | Dataedo documentation. *Dataedo documentation*. URL: https://docs.dataedo.com/docs/documenting-technology/supported-databases/ssrs/ (дата звернення: 18.09.2025).

28) What Is SQL Server Reporting Services? - SQL Server Reporting Services (SSRS). *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/sql/reporting-services/create-deploy-and-manage-mobile-and-paginated-reports?view=sql-server-ver17 (дата звернення: 18.09.2025).

ДОДАТКИ

ДОДАТОК А

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДИПЛОМ
І ступеню
НАГОРОДЖУЄТЬСЯ

Земов Сергій Олексійович

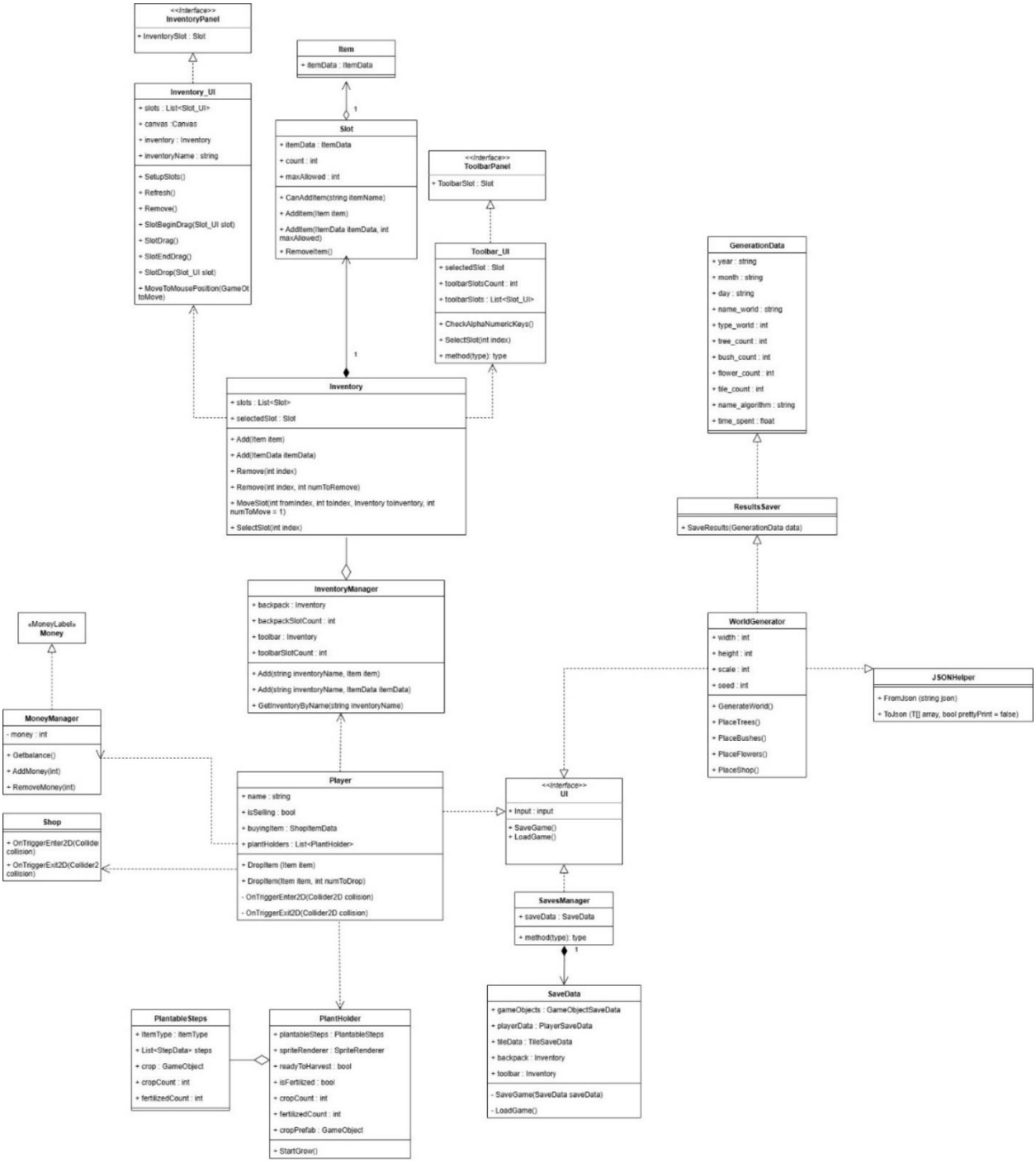
студент групи ІУСТ-24002м ОС «Магістр»

за доповідь «Система генерування ігрового світу»
на VII Всеукраїнській науково-практичній конференції студентів і аспірантів
«Теоретичні та прикладні аспекти розробки комп'ютерних систем '2025»

Завідувач кафедри комп'ютерних наук НУБіП України **Б.Л. Голуб**

24 квітня 2025 р.

ДОДАТОК Б


«Interface»
InventoryPanel
+ InventorySlot : Slot
Item
+ itemData : ItemData
Inventory_UI
+ slots : List<Slot_UI>
+ canvas :Canvas
+ inventory : Inventory
+ inventoryName : string
+ SetupSlots()
+ Refresh()
+ Remove()
+ SlotBeginDrag(Slot_UI slot)
+ SlotDrag()
+ SlotEndDrag()
+ SlotDrop(Slot_UI slot)
+ MoveToMousePosition(GameObject toMove)
Slot
+ itemData : ItemData
+ count : int
+ maxAllowed : int
+ CanAddItem(string itemName)
+ AddItem(Item item)
+ AddItem(ItemData itemData, int maxAllowed)
+ RemoveItem()
«Interface»
ToolbarPanel
+ ToolbarSlot : Slot
Toolbar_UI
+ selectedSlot : Slot
+ toolbarSlotsCount : int
+ toolbarSlots : List<Slot_UI>
+ CheckAlphaNumericKeys()
+ SelectSlot(int index)
+ method(type): type
GenerationData
+ year : string
+ month : string
+ day : string
+ name_world : string
+ type_world : int
+ tree_count : int
+ bush_count : int
+ flower_count : int
+ tile_count : int
+ name_algorithm : string
+ time_spent : float
Inventory
+ slots : List<Slot>
+ selectedSlot : Slot
+ Add(Item item)
+ Add(ItemData itemData)
+ Remove(int index)
+ Remove(int index, int numToRemove)
+ MoveSlot(int fromIndex, int toIndex, Inventory toInventory, int numToMove = 1)
+ SelectSlot(int index)
ResultsSaver
+ SaveResults(GenerationData data)
InventoryManager
+ backpack : Inventory
+ backpackSlotCount : int
+ toolbar : Inventory
+ toolbarSlotCount : int
+ Add(string inventoryName, Item item)
+ Add(string inventoryName, ItemData itemData)
+ GetInventoryByName(string inventoryName)
«MoneyLabel»
Money
WorldGenerator
+ width : int
+ height : int
+ scale : int
+ seed : int
+ GenerateWorld()
+ PlaceTrees()
+ PlaceBushes()
+ PlaceFlowers()
+ PlaceShop()
JSONHelper
+ FromJson (string json)
+ ToJson (T[] array, bool prettyPrint = false)
MoneyManager
- money : int
+ Getbalance()
+ AddMoney(int)
+ RemoveMoney(int)
Player
+ name : string
+ isSelling : bool
+ buyingItem : ShopItemData
+ plantHolders : List<PlantHolder>
+ DropItem (Item item)
+ DropItem(Item item, int numToDrop)
- OnTriggerEnter2D(Collider2D collision)
- OnTriggerExit2D(Collider2D collision)
«Interface»
UI
+ Input : input
+ SaveGame()
+ LoadGame()
Shop
+ OnTriggerEnter2D(Collider collision)
+ OnTriggerExit2D(Collider2 collision)
SavesManager
+ saveData : SaveData
+ method(type): type
SaveData
+ gameObjects : GameObjectSaveData
+ playerData : PlayerSaveData
+ tileData : TileSaveData
+ backpack : Inventory
+ toolbar : Inventory
- SaveGame(SaveData saveData)
- LoadGame()
PlantableSteps
+ itemType : itemType
+ List<StepData> steps
+ crop : GameObject
+ cropCount : int
+ fertilizedCount : int
PlantHolder
+ plantableSteps : PlantableSteps
+ spriteRenderer : SpriteRenderer
+ readyToHarvest : bool
+ isFertilized : bool
+ cropCount : int
+ fertilizedCount : int
+ cropPrefab : GameObject
+ StartGrow()

ДОДАТОК В

KPI середнього часу:

```
AVG([Date Dim].[Id Date].&[1]: [Date Dim].[Id Date].&[600],[Measures].[Time Spent])
```

KPI мінімального часу:

Вираз значення:

```
MIN([Date Dim].[Id Date].&[1]: [Date Dim].[Id Date].&[600],[Measures].[Time Spent])
```

Цільове значення:

```
AVG([Date Dim].[Id Date].&[1]: [Date Dim].[Id Date].&[600],[Measures].[Time Spent]) - 1.5
```

KPI максимального часу:

Вираз значення:

```
MAX([Date Dim].[Id Date].&[1]: [Date Dim].[Id Date].&[600],[Measures].[Time Spent])
```

Цільове значення:

```
AVG([Date Dim].[Id Date].&[1]: [Date Dim].[Id Date].&[600],[Measures].[Time Spent]) + 5
```

ДОДАТОК Г

Частота алгоритму:

```
SELECT
   a.name_algorithm AS [Алгоритм],
   COUNT(*) AS [Кількість_використань],
   CAST(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER () AS DECIMAL(5,2)) AS
[Частка_використання_у_%]
FROM GenerationFact gf
JOIN AlgorithmDim a ON gf.id_algorithm = a.id_algorithm
GROUP BY a.name_algorithm
ORDER BY [Частка_використання_у_%] DESC;
```

Середній час генерування алгоритмів:

```
SELECT
   ad.name_algorithm AS Algorithm,
   AVG(gf.time_spent) AS AverageTimeSpent
FROM
   GenerationFact gf
INNER JOIN
   AlgorithmDim ad ON gf.id_algorithm = ad.id_algorithm
GROUP BY
   ad.name_algorithm
ORDER BY
   AverageTimeSpent DESC;
```

Середня кількість згенерованих об’єктів:

```
SELECT
   ad.name_algorithm AS Algorithm,
   ROUND(AVG(od.tree_count), 3) AS AvgTrees,
   ROUND(AVG(od.bush_count), 3) AS AvgBushes,
   ROUND(AVG(od.flower_count), 3) AS AvgFlowers,
   ROUND(AVG(od.tile_count), 3) AS AvgTiles
FROM
   GenerationFact gf
INNER JOIN
   AlgorithmDim ad ON gf.id_algorithm = ad.id_algorithm
INNER JOIN
   ObjectsDim od ON gf.id_object = od.id_object
GROUP BY
   ad.name_algorithm
ORDER BY
   ad.name_algorithm;
```

Загальна кількість об’єктів:

```
SELECT
   ad.name_algorithm AS Algorithm,
   SUM(od.tree_count) AS TotalTrees,
   SUM(od.bush_count) AS TotalBushes,
   SUM(od.flower_count) AS TotalFlowers,
   SUM(od.tile_count) AS TotalTiles
FROM
   GenerationFact gf
INNER JOIN
   AlgorithmDim ad ON gf.id_algorithm = ad.id_algorithm
INNER JOIN
   ObjectsDim od ON gf.id_object = od.id_object
GROUP BY
   ad.name_algorithm
ORDER BY
   TotalTrees DESC, TotalBushes DESC;
```