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

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Підсистема створення 2D карти для ігрового застосунку»

Виконав (-ла):
студент (-ка) IV курсу, групи ІС-91
Куренна Анна Вадимівна _____

Керівник:
доцент, кандидат технічних наук,
Жураковська Оксана Сергіївна _____

Рецензент:
Доцент кафедри інформатики та програмної інженерії, к.т.н.,
Олійник Юрій Олександрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Куренній Анні Вадимівні

1. Тема проєкту «Підсистема створення 2D карти для ігрового застосунку», керівник проєкту Жураковська Оксана Сергіївна, к. т. н. затверджені наказом по університету від «31» травня 2023 р. №2101-с

2. Термін подання студентом проєкту: «12» червня 2023 року

3. Вихідні дані до проєкту: Технічне завдання

4. Зміст пояснювальної записки:

1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд наявних аналогів, постановка задачі;
2. Інформаційне забезпечення: вхідні та вихідні дані, опис структури бази даних;
3. Математичне забезпечення: змістовна та математична постановка задачі, обгрунтування та опис методу розв'язання;
4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів;

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

1. Схема варіантів використань системи;
2. Схема класів програмного забезпечення;
3. Схема послідовностей;
4. Схема компонентів.

7. Дата видачі завдання 13 квітня 2023 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Аналіз предметної області	17 квітня 2023	
2	Формалізація функціональної моделі	21 квітня 2023	
3	Огляд наявних аналогів	23 квітня 2023	
4	Розробка і опис структури бази даних	25 квітня 2023	
5	Опис математичної постановки задачі	5 травня 2023	
6	Огляд варіантів рішення задачі	8 травня 2023	
7	Розробка і опис технічного рішення	19 травня 2023	
8	Тестування готового проєкту	26 травня 2023	
9	Опис керівництва користувача	31 травня 2023	

Студент Анна КУРЕННА

Керівник Оксана ЖУРАКОВСЬКА

АНОТАЦІЯ

Структура та обсяг роботи. Дипломний проєкт на тему «Підсистема створення 2D карти для ігрового застосунку» складається з п’яти розділів, містить 21 рисунок, 19 таблиць, 1 додаток, 15 джерел.

Дипломний проєкт присвячений розробці комплексу задач, які стосуються генерації 2D карти та пошуку найкоротшого шляху між її клітинами.

У розділі інформаційного забезпечення описано вхідні та вихідні данні, розроблено схема таблиць бази даних.

Розділ математичного забезпечення присвячений формулюванню математичної задачі та огляду існуючих методів її розв’язання.

У розділі програмного забезпечення описуються засоби розробки програмного забезпечення, обгрунтовуються обрані архітектурні рішення та розробляються специфікації функцій.

Технологічний розділ визначає мету проведення випробувань програмного продукту та описує їх результати.

Ігровий застосунок, 2D карта, пошук найкоротшого шляху

ABSTRACT

Structure and scope of work. The diploma project on the topic "2D map creation subsystem for game application" consists of five sections, contains 21 drawings, 19 tables, 1 application, 15 sources.

The diploma project is devoted to the development of a set of problems related to the generation of a 2D map and the search for the shortest path between its cells.

In the information support section, input and output data are described, and a scheme of database tables is developed.

The chapter on mathematical support is devoted to the formulation of a mathematical problem and an overview of existing methods of solving it.

The software section describes software development tools, justifies selected architectural solutions, and develops function specifications.

The technological section defines the purpose of testing the software product and describes their results.

Game application, 2D map, find the shortest path

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1	А4	ІС91.140БАК.004 ПЗ	Пояснювальна записка	60		
2	А3	ІС91.140БАК.004 Д1	Підсистема створення 2D	1		
			карти для ігрового застосунку.			
			Схема варіантів використань			
			системи			
3	А3	ІС91.140БАК.004 Д2	Підсистема створення 2D	1		
			карти для ігрового застосунку.			
			Схема класів програмного			
			забезпечення			
4	А3	ІС91.140БАК.004 Д3	Підсистема створення 2D	1		
			карти для ігрового застосунку.			
			Схема послідовностей			
5	А3	ІС91.140БАК.004 Д4	Підсистема створення 2D	1		
			карти для ігрового застосунку.			
			Схема компонентів			

Зм.	Аркуш	№ докум.	Підпис	Дата	ІС91.140БАК.004 ТП		
Розроб.		Куренна А.В.			Підсистема створення 2D карти для ігрового застосунку. Відомість проєкту	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Жураковська О.С.				Т	
						КПІ ім. Ігоря Сікорського Група ІС-91	
Затв.							

Пояснювальна записка
до дипломного проєкту
на тему: «Підсистема створення 2D карти для ігрового застосунку»

Київ – 2023 року

ЗМІСТ

ВСТУП

Ігрова індустрія постійно розвивається та змінюється, створюючи непередбачувані можливості для розвитку як в комерційному, так і в технологічному аспектах. Одним з ключових елементів, які роблять ігровий досвід захоплюючим і привабливим, є графіка.

Швидкий розвиток комп'ютерної графіки, зростання потужності обчислювальних систем та поширення доступу до інтернету сприяли виникненню нових можливостей для створення багатокористувацьких ігор. Сьогодні ігрова індустрія стикається з постійним завданням залучення нових гравців та забезпечення їм захоплюючого та цікавого геймплею. Відповідно, виникає потреба у зручних та ефективних засобах створення вражаючих 2D карт, які б відповідали потребам сучасних гравців.

Генерація 2D карт відіграє ключову роль у створенні ігрових світів, дозволяючи розширити можливості геймдизайнерів та підвищити якість графічного виконання. Вона дає можливість автоматизувати процес створення деталей, що дозволяє розробникам більше часу сконцентруватись на створенні унікального геймплею та інноваційних ідей.

Метою розробки є підвищення ефективності процесу ігрової розробки за рахунок генерації 2D карт з заданого переліку ігрових компонентів, заданим наповненням ресурсами, та побудовою оптимальних маршрутів між клітинами, оминаючи перешкоди.

Для досягнення необхідної мети треба реалізувати комплекс наступних задач:

– розробка модуля генерації клітин карти, що в залежності від заданої позиції клітини у світі буде повертати її тип;

– розробка модуля розстановки на згенерованій карті міст гравців а також різнорівневих точок ресурсів;

– розробка модуля пошуку шляхів між клітинами на карті;

– розробка серверної частини додатку для підтримки багатокористувацького режиму;

– розробка графічного модуля для відображення згенерованої карти та візуалізації отриманої інформації із сервера.

Практичне значення полягає у тому, що розроблена система може бути впроваджена як самостійний модуль генерації карти, а також послужити базою для розширення ігрового функціоналу. Це дасть можливість автоматизовано розширювати ігрову карту і підлаштовуватись під будь-яку кількість гравців. Така генерація може допомогти створити реалістичні ігрові світи, що покращує іммерсивність та втягнення гравців. Система пошуку найкоротших шляхів у свою чергу дозволить ефективно використовувати системні ресурси та надавати візуалізацію побудованого шляху.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Ігрова індустрія розвивається стрімкими кроками, і на сьогоднішній день існує багато різновидів ігор, спрямованих на різні категорії користувачів. Окреме місце серед таких застосунків займає жанр багатокористувацьких ігор, адже він включає в собі різні аспекти соціалізації, які залучають нових гравців, і за рахунок цього довше утримують їх у грі.

Існує багато різних способів, за рахунок яких можна створити для гравця відчуття занурення в онлайн спільноту. Простим рішенням виступають звичайні турнірні дошки, змагання з друзями, можливість створювати онлайн спільноти, наприклад клани чи гільдії. Але, набагато цікавішими рішенням для користувачів є саме ігрові механіки, які дозволяють зануритись глибше у ігровий всесвіт. Саме таким елементом гри може стати глобальна карта, вона же – карта світу.

Глобальна карта – це часто використовувана механіка у мультиплеєрних іграх, що дозволяє гравцям взаємодіяти зі світом гри та іншими гравцями на великих масштабах. Така концепція проста у розумінні та охоплює багато соціальних аспектів, які приваблюють користувачів.

Режими глобальної карти можуть відрізнятися залежно від конкретної гри, але зазвичай вони мають декілька спільних рис:

– карта світу: це велика карта, яка представляє світ гри. Вона може містити різні типи місцевості а також конкретні об’єкти, такі як міста, фортеці, руїни та інші об'єкти;

– ресурси: глобальна карта може містити різні ресурси, які гравці можуть збирати та використовувати для розвитку своїх міст та армій;

– армії: гравці можуть створювати армії та відправляти їх на різні місця на карті, щоб воювати з іншими гравцями або захищати свої міста;

– клани та альянси: гравці можуть об'єднуватися в разом, щоб спільно воювати з іншими кланами або захищати свої території;

– завдання та події: глобальна карта може містити різні завдання та події, які гравці можуть виконувати, щоб отримати бонуси та нагороди.

Окрім цього, режими глобальної карти можуть мати різні правила та обмеження, що залежить від конкретної гри. Наприклад, деякі ігри можуть мати обмеження на кількість армій, які гравці можуть відправляти на карту, або максимальну кількість кланів або альянсів, які можуть об'єднуватися.

1.1.1 Опис процесу діяльності

Процес створення глобальної карти має дуже велике значення, оскільки у багатокористувацькому застосунку карта має бути масштабованою та підтримувати одночасно велику кількість гравців. Основною проблемою такої системи виступає оптимізація використовуваних ресурсів, адже зі збільшенням кількості гравців зростає навантаження на сервер. Для досягнення найкращої оптимізації, необхідно виділити ті компоненти, які залишатимуться статичними протягом усієї гри, і ті елементи, які треба тримати в пам’яті, бо вони мають поточний стан і можуть бути змінюваними.

Найменшою одиницею карти виступає її клітина. Вона представляє собою певну структуру, яка включає в себе координати цієї точки у 2D просторі. Також, ця структура містить у собі інформацію про проходимось клітини і тип її терену – ліс, галявина, горі і тд. Ще, для візуального відображення вона має деякі змінні, які зможуть додати різноманітності нашій карті. Очевидно, що така структура даних є статичною, і не буде змінюватись ніколи. Отже, необхідно розробити сервіс, який зможе детерміновано генерувати клітини, отримуючи вхідними параметрами координати цієї клітини на карті. Такий підхід генерації карти називається процедурним[1].

Також, окремим питанням під час генерації клітини стає визначення її координат. У нашій роботі буде розглянуто створення гексогональної карти, де кожна клітина позначається двома координатами (r;q), де r – рядок, q – колонка. Карти, які будуть створюватись, мають декілька варіантів

направлення клітин та власні правила зміни координат, які наведені на рисунку 1.1.

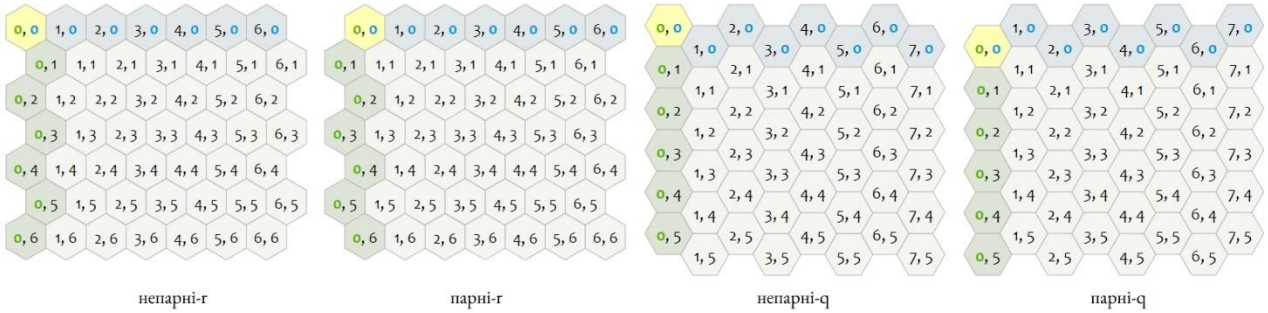


Рисунок 1.1 – Варіанти направлення координат клітин

Можемо помітити певні особливості для кожного із наведених типів карт:

– непарні-r – зсувають непарні рядки праворуч;

– парні-r – зсувають парні рядки праворуч;

– непарні-q – зсувають непарні колонки вниз;

– парні-q – зсувають парні колонки вниз.

У роботі буде використано шаблон «непарні-r». Для нього існує певний набір правил того, як визначаються сусідні клітини для поточної, який наведено на рисунку 1.2. Цей механізм буде потрібним для пошуку шляху і руху між координатами.

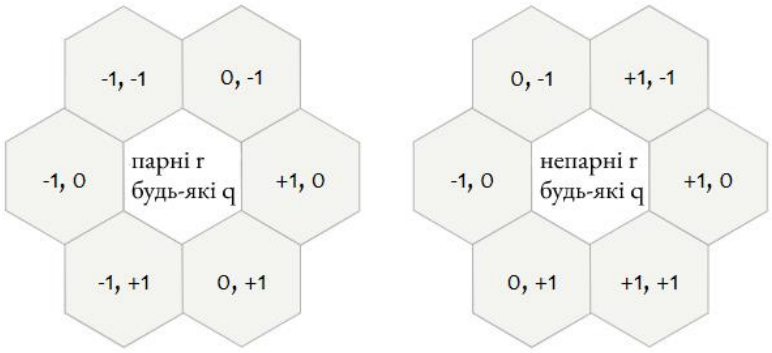


Рисунок 1.2 – Правила визначення координат для поточної клітини

Наступним кроком, коли система може створювати окремі клітини, постає питання генерації ресурсних вузлів і точок гравців. Гра передбачає, що користувачі можуть взаємодіями між собою та збирати ресурси. Це означає, що є необхідність зберігати у пам’яті системи інформацію про згенеровані

сутності і їх поточний стан. Саме тут ми і стикаємось з підпроцесом сегментації карти та її масштабуванням.

Нехай, сегментом або регіоном карти буде називатись її певна ділянка, яку можна описати певною кількістю клітин у ньому. Для полегшення подальшої розробки сегменти будемо робити квадратними, тоді нам треба буде задати певне число n, яке описуватиме довжину сторони цього квадрата, відповідно вся кількість клітин у сегменті дорівнюватиме n*n. Таких сегментів на карті може бути безліч, але генерувати нові ми будемо лише за необхідності – коли буде недостатньо створених вузлів для розміщення нових користувачів. Для всіх сегментів мають бути прописані загальні рекомендації створення ігрових вузлів на них. У нашому випадку такими рекомендаціями будемо вважати фіксовану кількість певних вузлів на сегменті. Наприклад, кожна ділянка карти має містити в собі 3 точки розташування гравців і 4 шахти з рудою. Також, кожен вузол сегменту має свої власні правила розташування на карті. Наприклад, руда може бути згенерованою лише в горах, а гравець – лише на галявині. Зрозуміло, що виконання цих рекомендацій є бажаною, але не обов’язковою умовою, адже генерація карти має випадковий характер, і тому система у деяких сегментах може просто не мати можливості виконати всі прописані умови.

Повний процес генерації карти є прихованим для всіх користувачів системи. Вже маючи згенеровані сегменти, гравці можуть взаємодіяти з вузлами на них. Також, створення гравців має певний вплив на стан карти.

Першої дією, яку виконує система, коли у ній з’являється новий користувач – є призначення йому його поточної позиції. Як було зазначено раніше, така позиція вибирається з наявних пустих вузлів у вже згенерованих сегментах карти. Якщо таких нема, то тоді генерується новий сегмент і вже на ньому вибирається перший вільний вузол.

Тепер, коли гравець має свою позицію на карті, він може взаємодіяти з навколишнім середовищем. Варто пам’ятати, що гра передбачає велику

кількість користувачів, а значить і велику кількість сегментів. Саме тому має бути розроблена система, яка буде загружати користувачу лише необхідну йому інформацію. На запуску гри такою інформацією є сегмент карти, де розташований гравець, і всі його ресурсні вузли. Також, користувач може рухатись картою, а значить має існувати механізм, який буде у реальному часі загружати нові ділянки карти і видаляти попередні для оптимізації ресурсів.

Взаємодія з ресурсними вузлами передбачає процес побудови маршрутів від користувача до цих вузлів. Гравець може відправляти армії, щоб зібрати ресурси, а армії, в свою чергу, обиратимуть найкоротший проходимий шлях до точки призначення.

Для створення ігрових подій у системі передбачено інший тип користувачів – адміністратори. Їх задача полягає у ручному коригуванні згенерованої карти та можливості створювати на певних позиціях спеціальні точки-події.

1.1.2 Опис функціональної моделі

Спочатку розглянемо деякі бізнес процеси, які відбуваються в нашій системі. Найцікавішими для нас будуть ті, які стосуються генерації карти. Наприклад, на рисунку 1.3 наведена схема генерації нового користувача. А на рисунку 1.4 детально розглянуто процес генерації сегменту, який згадується у попередній. Варто зазначити, що ці процеси відбуваються виключно у програмному коді зі сторони системи.

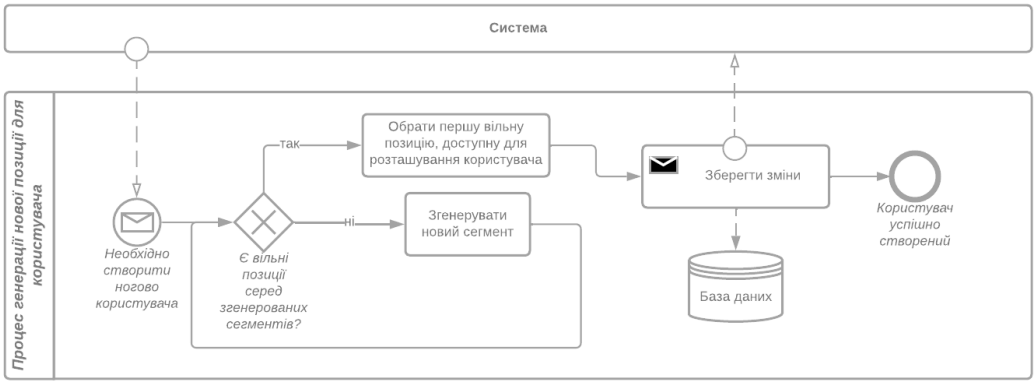


Рисунок 1.3 – Процес генерацій позиції для нового користувача

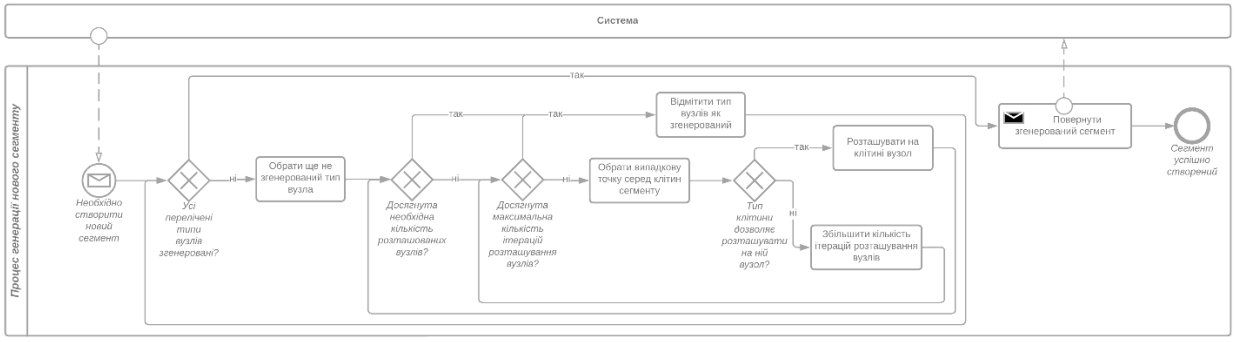


Рисунок 1.4 – Процес генерацій нового сегменту

Тепер, коли ми знаємо як відбуваються основні процеси генерації карти, можна говорити про акторів системи та функції, які вони виконують. Почнемо з акторів:

– користувач – звичайний гравець, який має базовий функціонал у грі. Для використання цього функціоналу він має авторизуватись або зареєструватись на сервісі;

– адміністратор – користувач з розширеними можливостями, такими як редагування позицій на карті та створенням нових ресурсних точок.

Схеми варіантів використання для користувача та адміністратора наведені у графічних матеріалах. А у таблиці 1.1 наведено детальний опис кожної із зазначених функцій.

Таблиця 1.1 – Опис варіантів використань

Актор	Варіант використання	Опис
Користувач	Авторизація	Користувач може ввести свої авторизаційні данні для того, щоб отримати доступ до свого профілю у грі
	Реєстрація	Користувач може зареєструватись у грі з унікальними реєстраційними даними
	Перегляд користувачів поблизу	Після авторизації користувач отримує свою позицію на карті і йому показуються гравці, які розташовані навколо нього
	Перегляд інформації про конкретного користувача	Натиснувши на конкретного користувача, гравець може отримати ігрову інформацію про вибраного користувача, наприклад, його рівень у грі
	Атака конкретного користувача	Гравець може атакувати конкретного користувача, відправивши свої армій
	Перегляд ресурсів поблизу	Після авторизації користувач отримує свою позицію на карті і йому показуються ресурси, які розташовані навколо нього
	Збір ресурсів	Гравець може відправити свої армії для того, щоб зібрати ресурси поблизу

Адміністратор	Авторизація	Адміністратор може зайти в систему зі своїми авторизаційними даними
	Зміна позицій користувача	Адміністратор може вибрати конкретного гравця на карті та змінити його позицію на будь яку іншу вільну клітину, на якій може розміщуватись точка з гравцем
	Додавання нових ресурсів на карту	Адміністратор може створити на карті новий ресурсний вузол
	Створення нового ресурсного вузла	Якщо серед наявних ресурсних нема такого, який задовільнить потреби адміністратора, він може створити свій власний та додати його на карту
	Використання готових ресурсних вузлів	Адміністратор може вибрати вузол для створення із переліку вже наявних вузлів
	Додавання конкретного вузла на карту	Після того, як було вибрано вузол необхідний для створення, адміністратор може розташувати його на карті
Система	Створення нового гравця	При реєстрації нового користувача, система обирає ще не зайняту позицію для розташування нового гравця. Якщо таких позицій нема, система може створити новий регіон

	Побудова найкоротшого маршруту	Коли гравець вибирає користувача для атаки або клітину з ресурсами для їх збору, система обраховує найкоротший шлях, який буде оминати перешкоди на карті

Для кожної з розглянутих функцій у таблиці 1.2 також наведені функціональні вимоги з відповідними пріоритетами.

Таблиця 1.2 – Функції акторів на їх пріоритети

Актор	Варіант використання	Функціональні вимоги	Пріоритет
Користувач	Авторизація	1. Система надає можливість вводити свої авторизаційні данні; 2. Система перевіряє коректність введених даних; 2.1. Система надає доступ до гри якщо авторизаційних дані наявні у системі; 2.2. Система повідомляє про помилку, якщо авторизаційних даних немає у системі;	Високий
	Реєстрація	3. Система надає можливість створювати облікові записи; 4. Система перевіряє коректність введених даних; 4.1 Система перевіряє унікальність електронної	Високий

		адреси, щоб уникнути дублікатів облікових записів; 4.2 Система повідомляє користувача про будь-які помилки, що виникли під час реєстрації;	
	Перегляд користувачів поблизу	5. Система виявляє та збирає дані про користувачів, які знаходяться у певному радіусі від поточного користувача; 5.1 Система регулярно оновлює інформацію про користувачів поблизу, щоб забезпечити точність та актуальність даних;	Високий
	Перегляд інформації про конкретного користувача	5.2 Користувач має можливість вибрати конкретного користувача, чию інформацію він бажає переглянути; 5.2.1 Система відображає детальну інформацію про обраного користувача, таку як нік у грі, рівень, досягнення тощо;	Високий
	Атака конкретного користувача	5.3 Система надає можливість атакувати вибраного користувача; 5.3.1 Система перевіряє наявність вільних армій у	Середній

		користувача перед дозволом на атаку; 5.3.2 Система враховує механіку битви та інші правила, що визначають результат атаки; 5.3.3 Після атаки система відображає результати битви між користувачами, такі як втрати армій, здобуті ресурси;	
	Перегляд ресурсів поблизу	6. Система виявляє та збирає дані про ресурсні вузли, які знаходяться у певному радіусі від поточного користувача; 6.1 Система регулярно оновлює інформацію про ресурсні вузли поблизу, щоб забезпечити точність та актуальність даних;	Високий
	Збір ресурсів	6.2 Система надає можливість зібрати ресурси з конкретного ресурсного вузла; 6.2.1 Система перевіряє наявність вільних армій у користувача перед дозволом на збір ресурсів; 6.2.2 Система враховує механіку збору ресурсів та інші правила, що визначають кількість та тип зібраних ресурсів;	Високий

		6.2.3 Ресурсні вузли можуть відновлювати свої ресурси з плином часу;	
Адміністратор	Авторизація	7. Система надає можливість вводити свої авторизаційні данні; 7. Система перевіряє коректність введених даних; 7.1. Система надає доступ до функцій адміністратора якщо авторизаційних дані наявні у системі;	Високий
	Зміна позицій користувача	8. Система надає відображення актуальних позицій користувачів у режимі реального часу 8.1 Вибір користувача може здійснюватися шляхом натискання на його іконку на карті 8.2 Система оновлює дані про позицію користувача в базі даних після зміни. 8.3 Після зміни позиції користувача система надсилає відповідне повідомлення користувачу	Високий
	Додавання нових	9. Адміністратор повинен мати можливість вибрати конкретну	Середній

	ресурсів на карту	клітину на карті, де він бажає створити новий ресурсний вузол	
	Створення нового ресурсного вузла	9.1 Система дозволяє створити новий ресурсний вузол 9.1.1 Система враховує особливості нового вузла, такі як його параметри 9.1.2 Система додає новий вузол у базу конфігурацій вузлів	Низький
	Використання готових ресурсних вузлів	9.2 Система надає перелік доступних типів ресурсних вузлів 9.2.1 Система перевіряє відповідність типу клітини до обраного типу ресурсу 9.2.2 Система зберігає вибір адміністратора	Середні
	Додавання конкретного вузла на карту	9.3 Система зберігає дані про новий ресурсний вузол, включаючи його тип, координати та параметри	Середній
Система	Створення нового гравця	10 Система створює позицію для нового гравця 10.1 Система створює новий сегмент карти, якщо немає вільної клітини для розміщення гравця	Високий

		10.2 Система зберігає розташування гравця	
	Побудова найкоротшого маршруту	11. Система будує найкоротший маршрут 11.1 Система враховує проходимось клітин і оминає перешкоди	Високий

1.2 Огляд наявних аналогів

Багато ігор використовують механіку глобальної карти. Для аналізу наявних аналогів розглянемо найбільш популярні застосунку[2], де ця механіка є ключовою:

- Civilization IV;
- Lords Mobile;
- Clash of kings.

Civilization IV – це покрокова стратегічна гра, де гравець створює та розвиває цивілізацію на глобальній карті, керуючи її економікою, науковими дослідженнями, дипломатією та військовими справами.

Рисунок 1.5 – Знімок екрану гри Civilization IV

Переваги:

– деталізована гексагональна карта світу з різними регіонами та ресурсами;

– автоматизована побудова маршрутів для переміщень армій на основі проходимості клітин карти;

– глибокі стратегічні можливості.

Недоліки:

– карта не є нескінченою, вона має фіксований розмір;

– карта може підтримувати лише конкретну кількість гравців;

– застосунок не підтримується мобільними приладами.

Lords Mobile – це стратегічна мобільна гра, в якій гравець будує своє власне королівство, формує армію та воює з іншими гравцями в режимі реального часу на карті.

Рисунок 1.6 – Знімок екрану гри Lords Mobile

Переваги:

– різноманітність режимів, яка дозволяє гравцям проводити як PvP так і PvE бої;

– наявність різнорівневих точок з ресурсами.

Недоліки:

– щільна забудова карти;

– можливість вільного переміщення користувачами свого міста на будь-яку іншу клітину;

– лінійні примітивні маршрути відправки армій для нападу.

Clash of Kings – це мобільна стратегічна гра з елементами MMO, де гравці змагаються за контроль над фантастичним світом, збираючи ресурси, ведучи бойові дії та будуючи власну імперію.

Рисунок 1.7 – Знімок екрану гри Clash of kings

Переваги:

– фіксована позиція гравця на карті, що змушує його використовувати навколишнє оточення у стратегічних рішеннях.

Недоліки:

– застарілий графічний дизайн;

– неможливість вимкнути відображення маршрутів усіх армій, що може робити інтерфейс переповненим.

Розглянувши наявні аналоги, їх переваги та недоліки, можна сформулювати загальні положення і найкращі сторони кожної з ігор, які необхідно буде розробити у подальшому. Результати наведені у таблиці 1.3.

Таблиця 1.3 – Порівняльна таблиця

Функція	Civilization IV	*Lords Mobile*	*Clash of Kings*	Розроблювана гра
Мультиплеєр	+	+	+	+
Підтримка необмеженої кількості гравців	-	+	+	+
Гексагональна карта	+	-	-	+
Побудова маршрутів з обминанням перешкод	+	-	-	+
Різноманітність ландшафту	+	-	-	+
Необмеженість часу ігрової сесії	-	+	+	+
Функції адміністратора	-	+	+	+

1.3 Постановка задачі

1.3.1 Призначення розробки

Призначенням розробки є підтримка процесу створення 2D карт з урахуванням особливостей наповнення їх ігровим контентом та можливістю побудови найкоротших маршрутів між довільними клітинами карти.

1.3.2 Цілі та задачі розробки

Метою дипломного проєкту є підвищення ефективності процесу ігрової розробки за рахунок генерації псевдовипадкових 2D карт з заданого переліку

ігрових компонентів, заданим наповненням ресурсами, та побудовою оптимальних маршрутів між клітинами, оминаючи перешкоди.

Для досягнення необхідної мети треба реалізувати комплекс наступних задач:

– розробка модуля генерації клітин карти, що в залежності від заданої позиції клітини у світі буде повертати її тип;

– розробка модуля розстановки на згенерованій карті міст гравців а також різнорівневих точок ресурсів;

– розробка модуля пошуку шляхів між клітинами на карті;

– розробка серверної частини додатку для підтримки багатокористувацького режиму;

– розробка графічного модуля для відображення згенерованої карти та візуалізації отриманої інформації із сервера.

Висновок до розділу

У цьому розділі було проведено детальне проектування підсистеми для створення 2D карт. Були описані предметне середовище та процес її діяльності, визначені актори та їх функції. Для встановлення пріоритетів функцій був використаний показник рівня важливості.

Також були проаналізовані існуючі аналоги з їх перевагами та недоліками. Можна зробити висновок, що існує багато багатокористувацьких ігор, які мають механіку глобальної карти, оскільки вона дозволяє гравцеві отримувати великий обсяг ігрової інформації та контролювати свою територію.. Але із розглянутих ігор мають застарілий дизайн або занадто важку ігрову логіку, які відлякує потенційних користувачів.

Було визначено призначення та мета програмного продукту, описані задачі для її реалізації, побудовані діаграма діяльності та таблиця функцій акторів з пріоритетами.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні данні

Розроблювана нами система має дві обширні функції:

– безпосередня генерація карти;

– можливість користувачів взаємодіяти зі згенерованою картою, відправляючи армії за ресурсами, таким чином будуючи найкоротший маршрут.

Як було згадано у функціональних вимогах, карта генерується сегментами, отже є певний набір інформації, який має бути відомо про сегмент, а саме:

– сторона (висота або ширина) для згенерованого сегменту: int;

– час у годинах, необхідний для переміщення через одну клітину: float;

– сід для генерації типів клітин карти: int;

– сід для генерації візуальних варіантів клітин карти: int;

– правила генерації окремих клітин карти: MapNodeClasses;

– правила генерації ресурсних вузлів на карті: ResourceNodeClassRequirements;

Для пошуку шляху вхідними даними система будуть:

– початкова позиція маршруту – координати гравця;

– кінцева позиція маршруту – координати пункту призначення;

– згенерована карта.

2.2 Вихідні данні

Вихідними даними карти є наступні структури:

– окремі клітини карти (рисунок 2.1);

– ігрові вузли карти – точки розташування гравців або ресурсні вузли (рисунок 2.2);

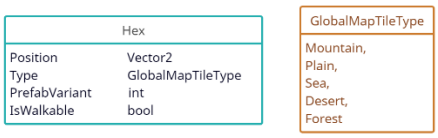


Рисунок 2.1 – Вміст об’єкту клітини карти

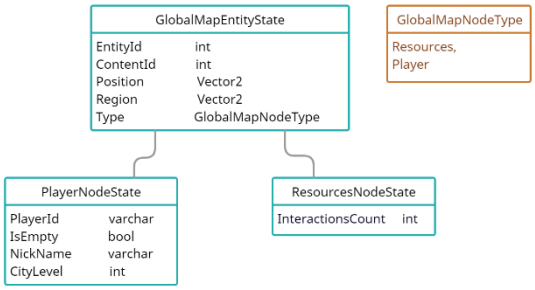


Рисунок 2.2 – Вміст об’єкту вузла карти

Також до вихідних даних системи можна віднести згенеровані маршрути, і армії, відправлені гравцями, які прямують цими маршрутами.

2.3 Опис структури бази даних

У таблиця бази даних ми будемо зберігати інформацію, частину конфігураційної інформації та ігрові стани гравців та сутностей, які пов’язані з нашою картою. На рисунку 2.3 наведена структура розроблюваної бд.

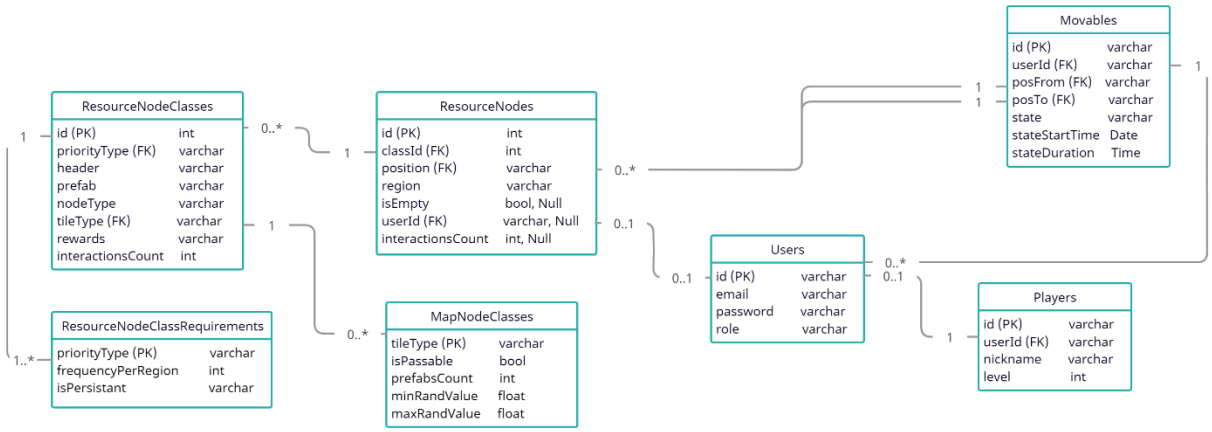


Рисунок 2.3 – Структура схема бази даних

Таблиця 2.1 – Опис таблиць бази даних

Таблиця	Призначення
Users	Таблиця, у якій зберігається інформація для авторизації користувачів
Players	Таблиця, у якій зберігається інформація про конкретних гравців, їх поточний ігровий стан
Movables	Таблиця, яка зберігає інформацію про відправлені гравцями армії та їх стан
ResourceNodeClasses	Таблиця, у якій є загальна інформація про ресурсні вузли та особливості їх розташування
ResourceNodes	Таблиця, у якій зберігається інформація про вже згенеровані ресурсні вузли на карті
ResourceNode ClassRequirements	Таблиця з рекомендаціями стосовно того, скільки яких вузлів має бути розташовано на сегменті
MapNodeClasses	Таблиця з описом правил генерації окремих клітин карти

Таблиці ResourceNodeClasses, ResourceNodeClassRequirements та MapNodeClasses відносяться до конфігураційних таблиць. Вони зберігають у собі інформацію необхідну для генерації карти, відображення клітин тощо. Їх основне призначення – описати правила гри.

У свою чергу, таблиці Users, Players, Movables та ResourceNodes відповідають конкретним згенерованим сутностям ігри, які мають свій поточний стан і можуть бути змінені у процесі гри.

Таблиця 2.2 – Опис полів баз даних

Таблиця	Поле	Опис
Users	id	Первинний ключ таблиці Users
	email	Адреса поштової скриньки
	password	Пароль

	role	Роль: Player або Admin
Players	id	Первинний ключ таблиці Players
	userId	Зовнішній ключ таблиці Players та таблицю Users
	nickname	Ігровий нік
	level	Ігровий рівень
Movables	id	Первинний ключ таблиці Movables
	userId	Зовнішній ключ таблиці Movables та таблицю Users
	nodeFrom	Точка початку шляху
	nodeTo	Точка закінчення шляху
	state	Поточний стан армії
	stateStartTime	Час, у який армія перейшла в поточний стан
	stateDuration	Тривалість стану, в якому знаходиться армія
ResourceNode Classes	id	Первинний ключ таблиці ResourceNodeClasses
	priorityType	Приоритет вузла, зовнішній ключ таблиці ResourceNodeClasses на ResourceNodeClassRequirements
	header	Назва вузла
	prefab	Посилання на візуальні ресурси в юніті, яка мають відтворювати зображення
	nodeType	Тип ресурсного вузла: Resources або Player
	tileType	Тип клітини, на якому може бути розміщений вузол

	rewards	Перелік нагород, які користувач може отримати
	interactionsCount	Максимальна кількість взаємодій з одним вузлом
	operatingDuration	Тривалість взаємодії з вузлом
ResourceNodes	id	Первинний ключ таблиці ResourceNodes
	classId	Зовнішній ключ таблиці ResourceNodes та таблицю ResourceNodeClasses
	position	Координати клітини на карті
	region	Координати сегменту, до якого відноситься клітина
	isEmpty	Булеве значення чи розташований на клітині гравець: true або false
	userId	Якщо гравець розташований, то тут має бути зовнішній ключ таблиці ResourceNodes та таблицю Users
	interactionsCount	Якщо це ресурсний вузол, то тут зберігається залишок кількості взаємодій з клітиною
ResourceNode Class Requirements	priorityType	Первинний ключ таблиці ResourceNodeClassRequirements
	frequency PerRegion	Максимальна кількість згенерованих клітин на весь сегмент
	isPersistant	Булеве зачення чи може бути клітина видалена і згенерована заново: true або false
MapNodeClass	tileType	Первинний ключ таблиці MapNodeClass
	isPassable	Булеве значення чи проходима клітина: true або false

	prefabsCount	Кількість візуальних варіантів вузла
	minRandvalue	Мінімальне випадкове число для генерації клітини
	maxRandValue	Максимальне випадкове число для генерації клітини

Висновок до розділу

В даному розділі була розроблена структура бази даних для системи, включаючи схему таблиць, їхнє призначення та опис змінних. Вірна та оптимізована структура бази даних є ключовим аспектом для ефективної роботи системи. Вона забезпечує ефективну обробку даних, швидкий доступ до інформації та забезпечує цілісність та надійність даних.

Правильна структура бази даних є основою для успішної роботи системи. Вона забезпечує ефективне управління даними, зручний доступ до інформації та сприяє високій якості результатів. Також, важливість точності та цілісності даних не може бути недооцінена, оскільки вона має вирішальний вплив на коректну роботу системи та задоволення потреб користувачів.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

В результаті генерації карти ми отримуємо певний масив клітин. Кожна така клітина може бути проходимою та непроходимою. Також, ми маємо гексогональний простір, а отже кожна клітина має 6 сусідів. Приклад наведено на рисунку 3.1. Тут білим кольором виділено проходимі клітини, і чорним – недоступні.

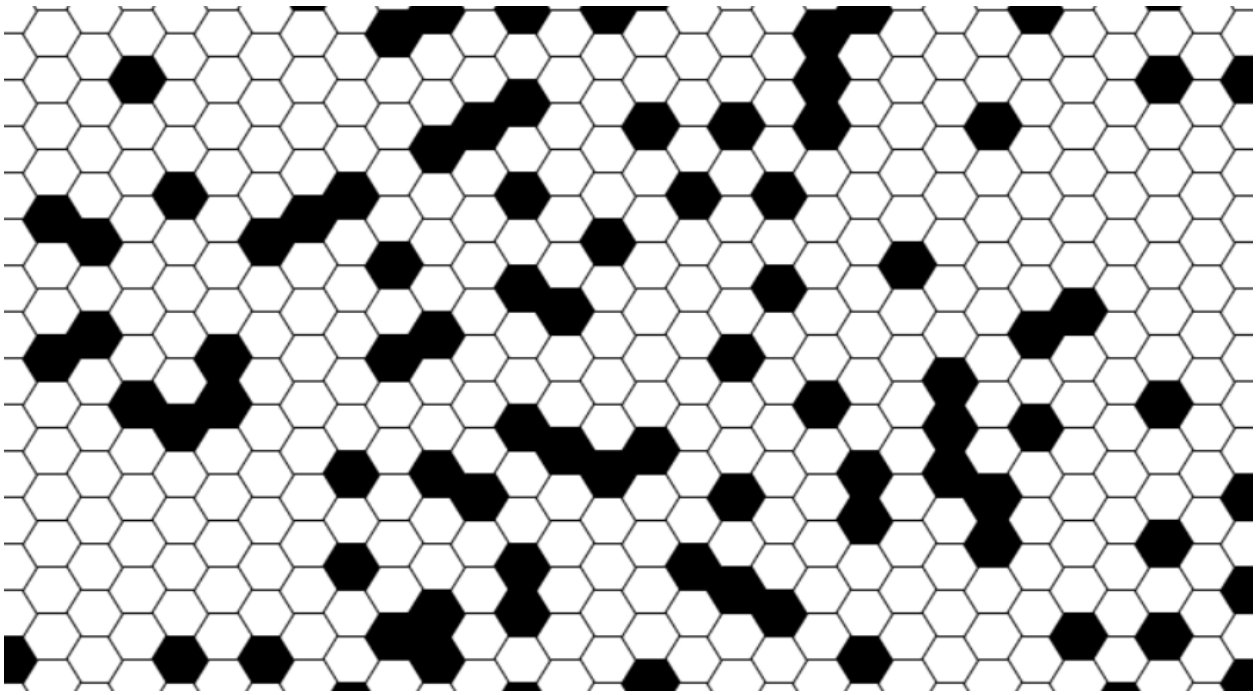


Рисунок 3.1 – Приклад згенерованої карти

Задача полягає у знаходженні найкоротшого маршруту між двома довільними проходимими клітинами, якщо хоча б один маршрут взагалі існує.

3.2 Математична постановка задачі

Змінні

Згенеровану карту ми будемо представляти як граф *D*. Відповідно, кожна клітина карти буде вершиною d_i цього графу, i = 1..n, n – кількість згенерованих клітин (3.1).

$$D = \{d_1, d_2, .., d_n\}, \tag{3.1}$$

Кожна клітина має таку характеристику, як проходимості v_i. Проходимі клітини будемо позначати 1, непроходимі – 0 (3.2).

$$v_i = \begin{cases} 1, \text{ якщо клітина проходима,} \\ 0, \text{ якщо клітина непроходима,} \end{cases} \quad (3.2)$$

Враховуючи, що створена нами карта має гексогональний вигляд, то кожна клітина має 6 сусідів, тобто 6 ребер у графі (3.3).

$$u_i = \{u_{i1}, u_{i2}, .., u_{i6}\}, \quad (3.3)$$

Шляхом буде називатись послідовність точок P на карті, яка сполучає точку початку d_x з точкою кінця d_y (3.4).

$$P = \{d_x, .., d_i, .., d_y\}, \quad (3.4)$$

Обмеження

1) Шлях має включати в себе початкову вершину і кінцеву:

$$P[1] = d_x, \qquad P[n] = d_y \quad (3.5)$$

2) Будь-які дві послідовні клітини шляху мають бути сполучені між собою:

$$\forall d_i \in P = u_{d_{i+1}} \in u_i \quad (3.6)$$

3) Всі клітини шляху мають бути проходимими:

$$\forall v_i \in P = 1 \quad (3.7)$$

Цільова функція

Необхідно побудувати маршрут, довжина якого буде мінімальною:

$$\sum_{v_i \in P} v_i \to min \quad (3.8)$$

3.3 Обґрунтування методу розв’язання

Проблема пошуку найкоротшого шляху є достатньо актуальною на сьогодняшній день. Її варіації трапляються майже у всіх областях програмування, як то у транспортних системах, геоінформатиці, мережевих протоколах і, звісно, в ігровій розробці.

Перший алгоритм пошуку шляху на графі був запропонований у 1959 році у статті "A Note on Two Problems in Connexion with Graphs"[3]. Цей алгоритм, відомий як алгоритм Дейкстри, знаходить найкоротший шлях на зваженому графі з не від'ємними вагами ребер.

На сьогодні, однією з найвідоміших книг з алгоритмів пошуку найкоротшого шляху на графах є "Introduction to Algorithms"[4]. У книзі розглядаються різні алгоритми пошуку найкоротшого шляху, включаючи алгоритми Дейкстри та Беллмана-Форда, а також алгоритм A*. Іншою добре відомою книгою є "Shortest Path Algorithms: Theory and Experimental Evaluation"[5]. Автор розглядає різні алгоритми пошуку найкоротшого шляху, включаючи алгоритми Дейкстри, Беллмана-Форда та Флойда-Уоршелла. Крім того, у книзі описується практичне застосування алгоритмів.

Враховуючи велику кількість наявних алгоритмів пошуку шляхів на графі, можна виділити дві основні стратегії:

– стратегія неінформованого (сліпого) пошуку – алгоритми такого типу можуть лише створювати нові стани на базі вже існуючих і перевіряти їх, чи відповідає стан шляху з точки A в точку B;

– стратегія інформованого (евристичного) пошуку – такі алгоритми не тільки дозволяють знайти шлях між двома точками, а і використовують інформацію з попередніх станів для аналізу потенційно найкоротших шляхів.

Єдиним алгоритмом сліпого пошуку, який може бути використаний у нашому випадку – це пошук в ширину. Специфіка нашої задачі така, що всі переходи між клітинами, тобто ребра графа, мають вагу 1, отже перший знайдений варіант одразу ж буде найкращим. Але вагомим недоліком цього алгоритму є те, що одночасно у пам’яті треба зберігати велику кількість недосяжних станів, які будуть розвиватись у майбутньому в пошуках найкращого рішення.

Використовуючи евристичний пошук, ми створюємо функцію оцінки стану, щоб можна було обрати серед них оптимальний і продовжити розвивати

саме його. У нашому випадку функція оцінки – це відстань до точки призначення. Чим менше відстань залишилось пройти, тим краще стан.

Тепер розглянемо, як алгоритми працюють на конкретній задачі. Нехай ми маємо певну згенеровану карту (рисунок 3.2), на якій білим кольором позначені звичайні проходимі клітини, а чорним – недоступні клітини з перешкодами. Початком маршруту позначена зелена клітина, кінцем – червона.

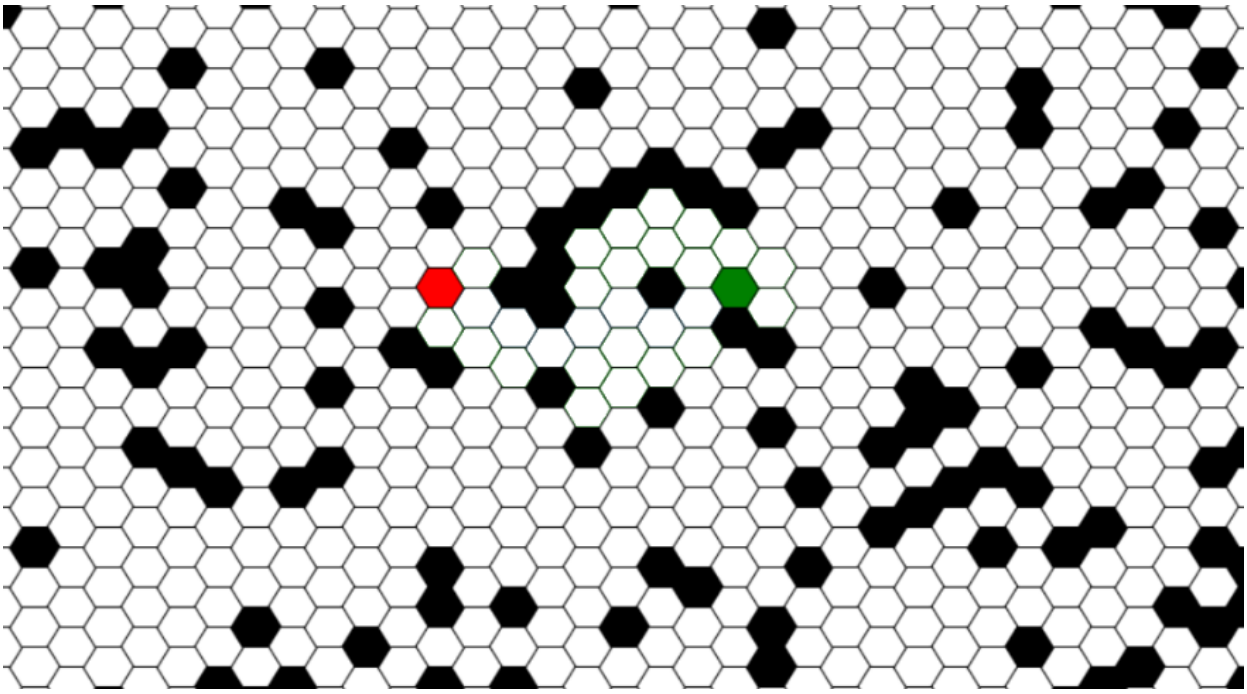


Рисунок 3.2 – Умова задачі

Тепер запустимо алгоритм пошуку (рисунок 3.3, рисунок 3.4). В результаті ми отримаємо послідовність блакитних клітин – найкоротший шлях. Блідо зеленим кольором позначені клітини, які були розглянуті в процесі пошуку шляху. Блідо жовтим – ті, які у момент знаходження найкращого шляху все ще знаходяться в оперативній пам’яті.

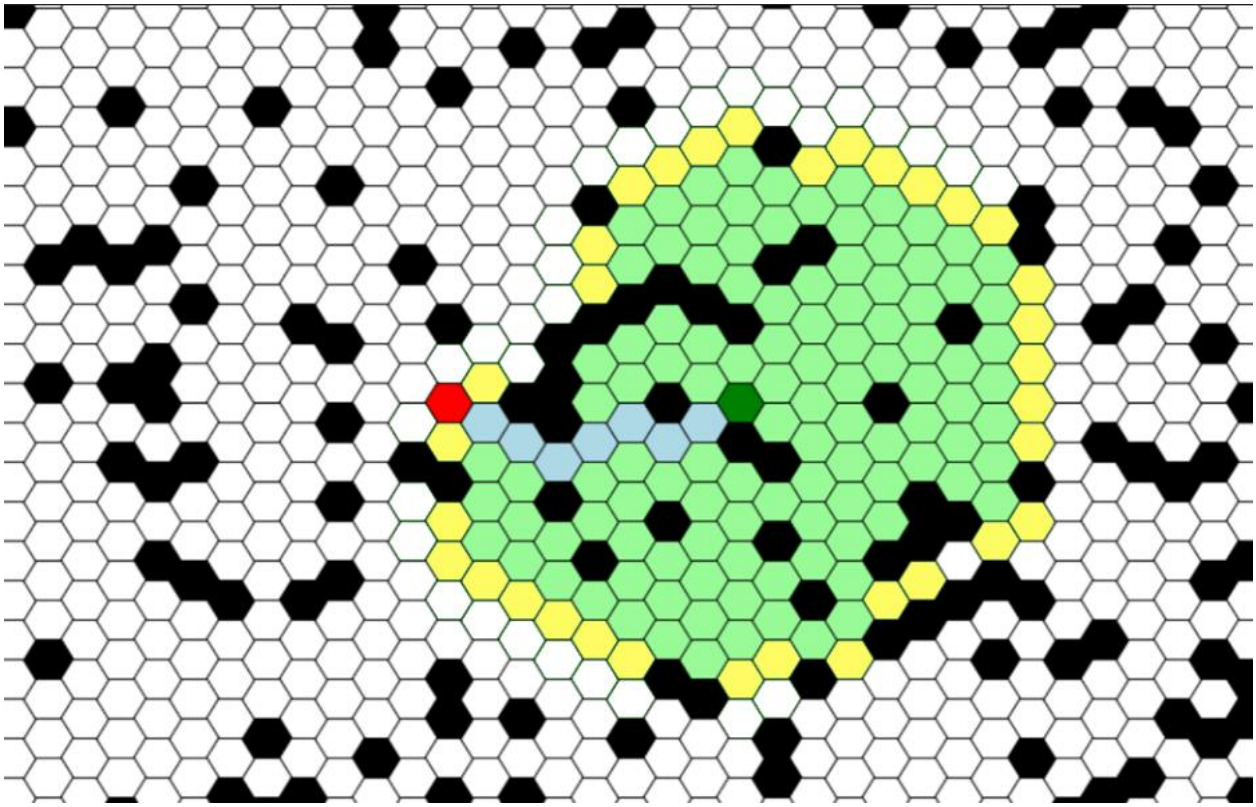


Рисунок 3.3 – Пошук найкращого шляху алгоритмом пошуку вглиб

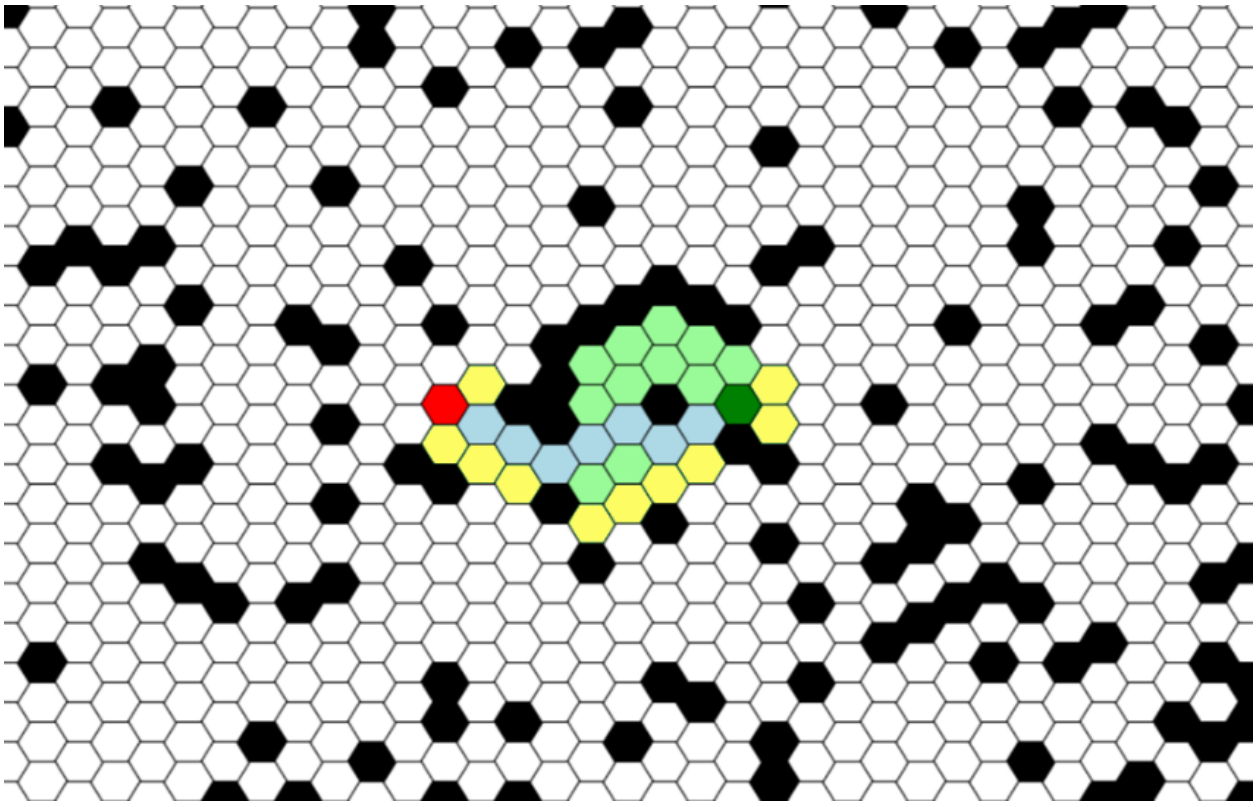


Рисунок 3.4 – Пошук найкращого шляху алгоритмом A*

Очевидно, що алгоритм A* показує кращі результати. За допомогою вибору кращого стану серед наявних, алгоритм швидше приходить до шуканого результату і потребує значно менше ресурсів.

3.4 Опис методу розв’язання

Основна ідея алгоритму A* полягає в поєднанні двох компонент: пошуку в ширину (BFS) та оцінки відстані до кінцевої точки. Закритими клітинами будемо вважати такі точки, для яких вже було знайдено найкраще рішення, відкритими – стани, які ще зберігаються у пам’яті і потенційно можуть бути розглянуті для пошуку. Нижче наведено опис алгоритму A* для поточної задачі:

а) ініціалізувати список відкритих клітин із початковою клітиною. Встановити відстань від початкової клітини до себе як 0;

б) ініціалізувати список закритих клітин;

в) поки список відкритих клітин не порожній, повторювати наступні дії:

1) вибрати клітину з найменшою загальною вартістю (вартість шляху до цієї клітини + оцінка вартості до кінцевої точки);

2) перемістити обрану клітину зі списку відкритих клітин у список закритих клітин;

3) перевірити, чи ця клітина є кінцевою точкою. Якщо так, то алгоритм завершується і знайдений шлях є оптимальним;

4) для кожної сусідньої проходимої клітини, що не знаходиться в списку закритих клітин:

– обчислити вартість шляху до сусідньої клітини;

– якщо сусідня клітина вже знаходиться у списку відкритих клітин і нова вартість шляху до неї менша, оновити її вартість шляху та батьківську клітину;

– якщо сусідня клітина не знаходиться у списку відкритих клітин, додати її туди і встановити вартість шляху та батьківську клітину;

г) якщо список відкритих клітин став порожнім і кінцева точка не досяжна, то немає шляху між початковою та кінцевою точками;

д) побудувати шлях, переходячи від кінцевої клітини до початкової по батьківським клітинам.

Цей алгоритм A* дозволяє знайти найкоротший шлях між двома заданими точками на гексогональній карті з урахуванням проходимості клітин. Він використовує покриття в ширину для обходу графа та евристичну оцінку для визначення найкращого напрямку до кінцевої точки.

Висновок до розділу

У цьому розділі були розглянуті різноманітні підходи пошуку найкоротшого шляху у графі, і визначено, що найкращим алгоритмом виявляється A*. У порівнянні з іншими алгоритмами, A* відрізняється тим, що він використовує інформацію про те, як далеко поточна вершина від цільової вершини, що робить його більш ефективним і швидким.

Алгоритм A* є оптимальним, бо він знаходить найкоротший шлях від стартової до цільової вершини за найменшу кількість кроків. Він забезпечує ефективний і точний пошук шляху у графі, враховуючи ваги кожної вершини і покращує продуктивність в порівнянні з іншими алгоритмами, що забезпечує його популярність серед різних задач.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Реалізація програмного продукту складається з двох глобальних частин: розробки серверної частини гри для підтримки багатокористувацького режиму, та створення візуалізації з використанням готового ігрового рушія.

Найкращим поєднанням засобів розробки для такого випадку буде створення серверної частини за допомогою Asp.Net Web Api[6], а для ігрового клієнта – на Unity Engine[7].

Значною перевагою такої комбінації є те, що обидва сервіси використовують одну мову програмування – C#[8]. Це спрощує інтеграцію бізнес логіки гри (її правил) у візуальне представлення рушія. Також з’являється можливість створити спільну кодову базу, адже зникає необхідність конвертувати код з одної мови в іншу. Зменшення дублювання коду спрошує підтримку та розробку проекту, дозволяючи робити зміни лише в одному місці.

У свою чергу, розділення бекенду та візуалізації дозволяє краще управляти та масштабувати проект. Це дає можливість оптимізувати та розподілити ресурси між серверною та клієнтською сторонами, що поліпшує швидкість гри. Крім того, це дозволяє зручно змінювати або модифікувати логіку гри без повного перекомпілювання її ігрового клієнту.

Сама по собі, ASP.Net[9] — технологія створення вебзастосунків і вебсервісів від компанії Майкрософт. Фреймворк Web Api відкриває доступ для інтерфейсу програми, через який клієнт, не залежно від його реалізації, може комунікувати з бізнес логікою. Він дозволяє створювати масштабовані та взаємодіючі веб-сервіси з використанням стандартних HTTP методів та підтримки форматів даних[10], таких як JSON та XML.

Технологія ASP.Net включає в себе також компонент Entity Framework[11], що є структурою об'єктно-реляційного відображення (ORM) з відкритим кодом. Він забезпечує зручний спосіб взаємодії з базами даних,

використовуючи об'єктну модель та мову запитів LINQ. Entity Framework автоматично відображає структуру бази даних на об'єктну модель, що спрощує роботу з даними як зі звичайними об'єктами. Він підтримує автоматичне створення та оновлення схеми бази даних, керування транзакціями та безпекою, а також підтримує різні типи баз даних.

Рушій Unity надає великий функціонал та набір інструментів для створення ігрової логіки, який включає графічний та фізичний рушії, анімацію, та звук. Його велика перевага полягає в тому, він дозволяє створювати ігри з відносно невеликим обсягом скомпільованого результуючого файлу проекту.

Окрему увагу у функціоналі Unity необхідно приділити компоненту PerlinNoise. Цей вже реалізований використовує алгоритм Перліна для створення псевдовипадкового шуму. Потім цей шум буде використаний для визначення типу клітини на карті. На вхід він отримує двовимірні координати клітини (x;y), потім як результат певних обрахунків видає псевдописадкове число. Результат роботи алгоритму буде завжди повертати одні і ті самі числа для введених однакових координат, що дозволить дотримуватись детермінізму при генерації карти.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Програмна реалізація додатку складається з трьох великих компонентів:

- ігрового клієнту на Unity;
- веб-сервісу;
- бази даних.

Рушій Unity дозволяє компілювати ігри під велику кількість платформ, будь то веб-браузер, консоль, мобільний телефон або комп’ютер. У нашому випадку розглянемо вимоги до найпопулярніших платформ.

PC:

– ОС Windows 7 / 8 / 8.1 / 10 (64-розрядна) і вище;
– процесор Intel Celeron G1820 | AMD A4-7300;
– відеокарта NVIDIA GeForce GT 730 | Radeon R7 240;
– дозвіл екрану: 1280 x 720 або вище;
– мережа: швидкісний доступ в інтернет;
– оперативна пам'ять: 2 ГБ;
– вільне місце на диску: 1 ГБ;
– звукова карта: сумісна з DirectX.

iOS:

– версія ОС: iOS 14 і вище;
– пристрої: iPad 4 / iPhone 7 /iPad mini 3 і новіше.

Android OS:

– версія ОС: Android 5.2 і вище;
– процесор: від 1200 МГц (рекомендується 1500 МГц), 2 ядра;
– оперативна пам'ять: 512 Мб і більше.

Розгортання веб-сервісу та бази даних необхідно реалізовувати на базі Azure Web Service та Azure SQL Database відповідно.

Системні вимоги для Azure Web App:

– операційна система: Windows Server;
– .NET Framework 4.8;
– веб-сервер: IIS (Internet Information Services) у версії 8.0 або вище;
– CPU: 1 віртуальний процесор або більше;
– оперативна пам’ять: 1Гб і більше;
– дисковий простір: 1Гб і більше;
– інтернет-підключення: система має доступ до інтернету.

Системні вимоги для Azure SQL Database:

– DTUs (одиниці транзакці бази даних): 5 DTU і більше;
– об’єм пам’яті бази даних: 2Гб і більше;

4.2.2 Опис локальної обчислюваної мережи

Для реалізації застосунку буде використовуватись трирівнева архітектура[12], наведена на рисунку 4.1

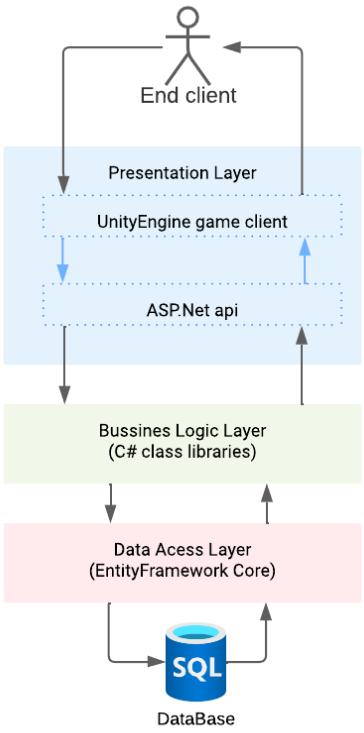


Рисунок 4.1 – Схема архітектури застосунку

На рівні представлення знаходиться ігровий клієнт, реалізований на базі рушія Unity. Гравець взаємодіє з інтерфейсом гри, після цього застосунок відсилає запити на ASP.Net api. У свою чергу, апі лише перенаправляє запити на внутрішні сервіси, не виконуючи валідації чи інших сторонніх функцій. Його основною метою є маршрутизація, тому він також відносить до рівню представлення.

Рівень бізнес логіки визначає основні правила, логіку та функціональність, що регулюють поведінку гри та взаємодію гравців. Він керує такими аспектами як механіками гри, ігровими об’єкти та середовищем, мережевою взаємодію і тд. Програмна реалізація розроблюється мовою C#.

На рівні доступу до бази даних використовується вбудований компонент технології ASP.Net Entity Framework для спрощення підключення до бази даних та обробки і оновлення інформації, яка у ній зберігається.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма класів

Застосунок складається з великої кількості різноманітних класів, але серед них можна виділити основні, які безпосередньо стосуються генерації карти та взаємодії з нею. Відповідна схема класів та відношень між ними наведена у графічних матеріалах. А у таблиці 4.1 наведено опис цих класів.

Таблиця 4.1 – Опис основних класів застосунку

Клас	Призначення
GlobalMapTileType	Перелік основних типів ландшафту клітини
GlobalMapNodeType	Перелік основних типів ресурсних вузлів
CustomVector2	Структура для представлення вектора у двовимірному просторі
GlobalMapEntityState	Абстрактний клас з загальними змінними вузлів на карті
PlayerNodeState	Вузол для представлення розміщення гравця на карті
ResourceNodeState	Вузол для представлення розміщення ресурсів на карті
ResourceNodeClass	Конфігураційний клас з загальною інформацію про конкретні вузли
MapNodeClass	Конфігураційний клас з описом правил генерації окремих клітин карти
ResourceNode ClassRequirement	Конфігураційний клас з описом правил розміщення вузлів на карті
ConfigurationService	Клас-сервіс, який надає доступ до конфігурації
Hex	Клас представлення клітини карти

GlobalMapGenerator	Клас-сервіс, який відповідає за створення окремих клітин карти
GlobalMapRegion Genirator	Клас-сервіс, який відповідає за створення регіонів карти
GlobalMapService	Клас-сервіс, який оброблює запити з АПІ
PathFinding	Клас-сервіс для пошуку найкоротшого шляху
MapQuery	Клас-сервіс, який допомагає визначити сусідні клітини для поточної

4.3.2 Діаграма послідовностей

Тепер розглянемо послідовності авторизації та реєстрації користувача, які наведені у графічних матеріалах.

Для авторизації користувач має отримати від вед застосунку декілька послідовних відповідей:

– результат авторизації – тривіальна операція, яка перевіряє наявність користувача в системі;

– загружену конфігурацію – конфігураційну інформацію про усі існуючі типи клітин та ресурсні вузли;

– поточний вибраний регіон – при відкритті застосунку, користувач загружається у тему регіоні, де розташований вузол з його замком;

– інформація про вузли, розташовані у регіоні – перелік усіх вузлів, що розташовані на конкретному регіоні. Важливо, що користувач може рухатись картою, отже регіони будуть змінюватись і загружатись у процесі гри. І навіть якщо гравець не робить жодних рухів, він все одно буде з деяким проміжком часу отримувати оновлену інформацію про вузли на поточному регіоні.

Процес реєстрації нового користувача схожий до авторизації. Єдина різниця у тому, що перевірка авторизації розширена. Після успішної перевірки, що користувач є унікальним, запускається процес створення нового гравця. Він передбачає вибір першої вільного вузла під замок і розташування

там гравця. У випадку, коли всі вузли зайняті, відбувається генерація нового регіону, допоки не з’явиться вільний вузол для розташування гравця.

4.3.3 Діаграма компонентів

Уся ігрова логіка генерації карти та роботи з нею описана 4 основними високорівневими компонентами[13]:

– конфігураційний сервіс (ConfigurationService) – надає доступ до об’єктів конфігурації;

– сервіс генерації карти (GlobalMapGenerator) – займається створенням клітин карти та вузлів на них, враховуючи особливості конфігурації;

– сервіс пошуку найкоротшого шляху (PathFinding) – знаходить найкоротший проходимий шлях в залежності від заданих координах;

– сервіс роботи з картою (GlobalMapService) – дає можливість взаємодіяти з ігровою логікою, додавати нових гравців та генерувати нові сегменти карти.

У свою чергу, кожен такий компонент складається з різних окремих компонентів та систем компонентів, які мають між собою певні взаємозв’язки. Відповідна діаграма компонентів наведена у графічних матеріалах.

4.3.4 Специфікація функцій

З діаграми класів, наведеної у графічних матеріалах, видно, що не всі класи мають функціонал. Насправді, при створенні застосунку був вибраний data-driven design[14], метою якого є розділення класів на ті, що зберігають інформацію, і ті, які її опрацьовують – тобто є сервісами. У таблицях 4.2-4.7 наведено опис основних класів-сервісів.

Таблиця 4.2 – Опис функцій класу CustomVector2

Назва функції	Опис функції
Distance	Метод знаходження відстані між двома точками

TryParse	Метод конвертації строки в CustomVector2. Якщо не було помилок при конвертації, то результат методу буде true, в іншому разі false
ToString	Метод конвертації CustomVector2 в строку. Строка створюється у форматі «x;y», де x і y – координати вектору

Таблиця 4.3 – Опис функцій класу MapQuery

Назва функції	Опис функції
GetNeighbourTiles	Метод, який повертає сусідні клітини від заданої
GetNeighbourPositions	Метод, який повертає координати сусідніх клітин від заданої

Таблиця 4.4 – Опис функцій класу PathFinding

Назва функції	Опис функції
Find	Метод пошуку найкоротшого шляху від початкової точки до кінцевої
FitFunction	Метод оцінки стану, який повертає відстань від поточного стану до результуючої точки. Чим менша відстань – тим кращий стан
GetShortestPath	Метод для конвертації результатів пошукового алгоритму у конкретний шлях, який починається початковою вершиною і закінчується кінцевою

Таблиця 4.5 – Опис функцій класу GlobalMapGenerator

Назва функції	Опис функції
Get	Метод створення клітини в залежності від її (x,y) координат
GetTileType	Метод генерації типу ландшафту клітини в залежності від її координати

GetPrefabVariant	Метод генерації індексу візуалізації клітини в залежності від її координати
GetQuantumIndex	Допоміжний метод, квантизації випадкового числа у конкретний індекс підгрупи

Таблиця 4.6 – Опис функцій класу GlobalMapRegionGenerator

Назва функції	Опис функції
GenerateNew	Метод генерації нового регіону за заданою позицією
Regenerate	Метод відновлення вже існуючого регіону, який прибирає ресурсні вузли, в яких закінчились ресурси
Generate	Загальний метод генерації регіону
GeneratePositions	Метод генерації випадкових позицій для заданого типу вузлів, якій враховує вже існуючі вузли
GenerateNodes	Метод генерації вузлів на основі випадкових позиції
CheckGenerated Position	Метод перевірки позиції вузла, який перевіряє чи вільна клітина і чи відповідає її тип доступним вузлам
GetAvailableTypes	Метод який повертає доступні конкретні вузли в залежності від заданого правила генерації

Таблиця 4.7 – Опис функцій класу GlobalMapService

Назва функції	Опис функції
AddPlayer	Метод для створення нового гравця і його розташування на карті
GetLookAtRegion	Метод, який повертає регіон, у якому був створений певний гравець
GetEntities	Метод, який повертає усі вузли, які знаходяться на заданому регіону. Якщо регіону не існує, то він створюється
GenerateRegion	Метод генерації регіону

GetRandomRegion	Метод вибору випадкового регіону для генерації у випадках, коли мапу треба розширити у випадкову сторону

Висновок до розділу

У цьому розділі були розглянуті підходи до проектування застосунків, вибрані найкращі рішення для реалізації під поставлену проблему– Unity Engine та ASP.Net.

Було встановлено, що використання трьохрівневої архітектури є ефективним підходом для розробки багатокомпонентних ігрових систем. Розділення додатку на окремі шари - презентаційний, бізнес-логіки та доступу до даних - сприяє модульності, розширюваності та підтримці додатку.

Для додатку були описані основні компоненти системи, створені взаємозв’язки між ними та описані послідовності взаємодій. Також були розроблені діаграма класів. І для кожного з класів описані їх функції.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Розглянемо функції користувача, які були реалізовані в системі. По-перше, користувач має можливість авторизуватись в системі, екран авторизації наведено на рисунку 5.1. Для авторизації необхідно ввести логін та пароль у відповідні поля «Login» та «Password». Кнопка «Log in» стане доступною після того, як у кожному з полів буде введено хоча б по одному символу.

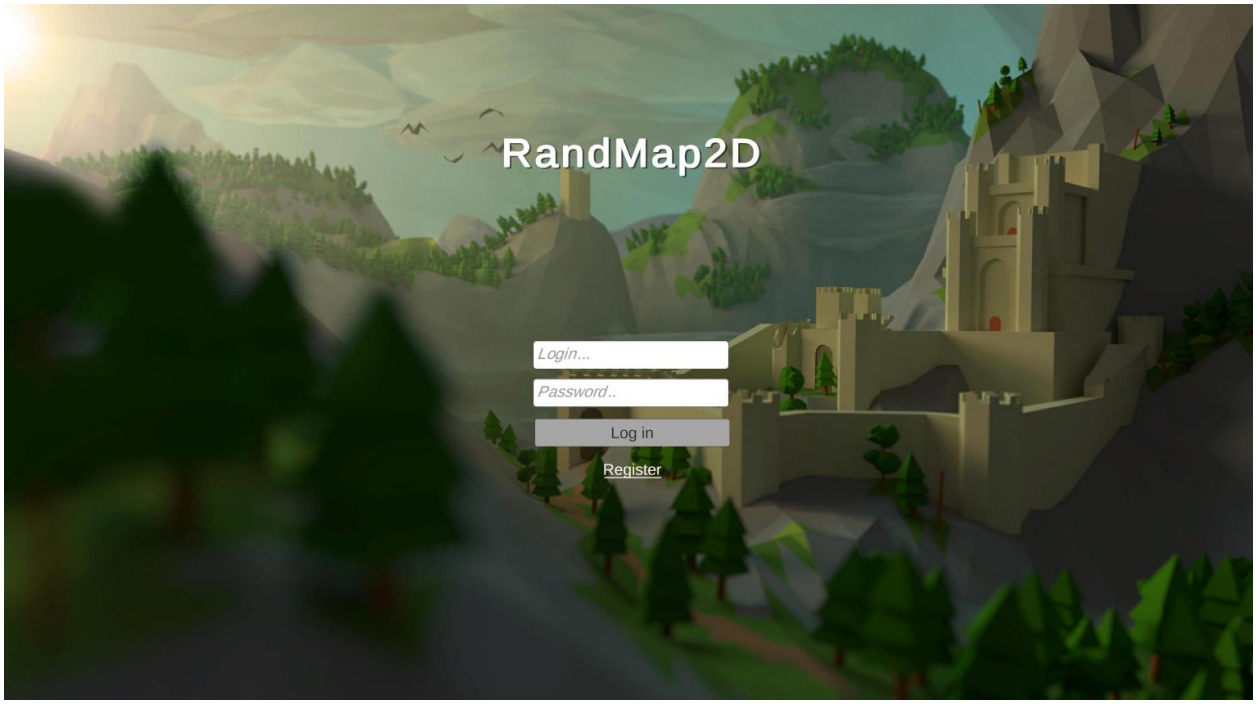


Рисунок 5.1 – Екран авторизація у грі

Також, гравець має можливість зареєструватись у грі. Для цього необхідно натиснути кнопку «Register» і відкриється меню реєстрації, наведене на рисунку 5.2.

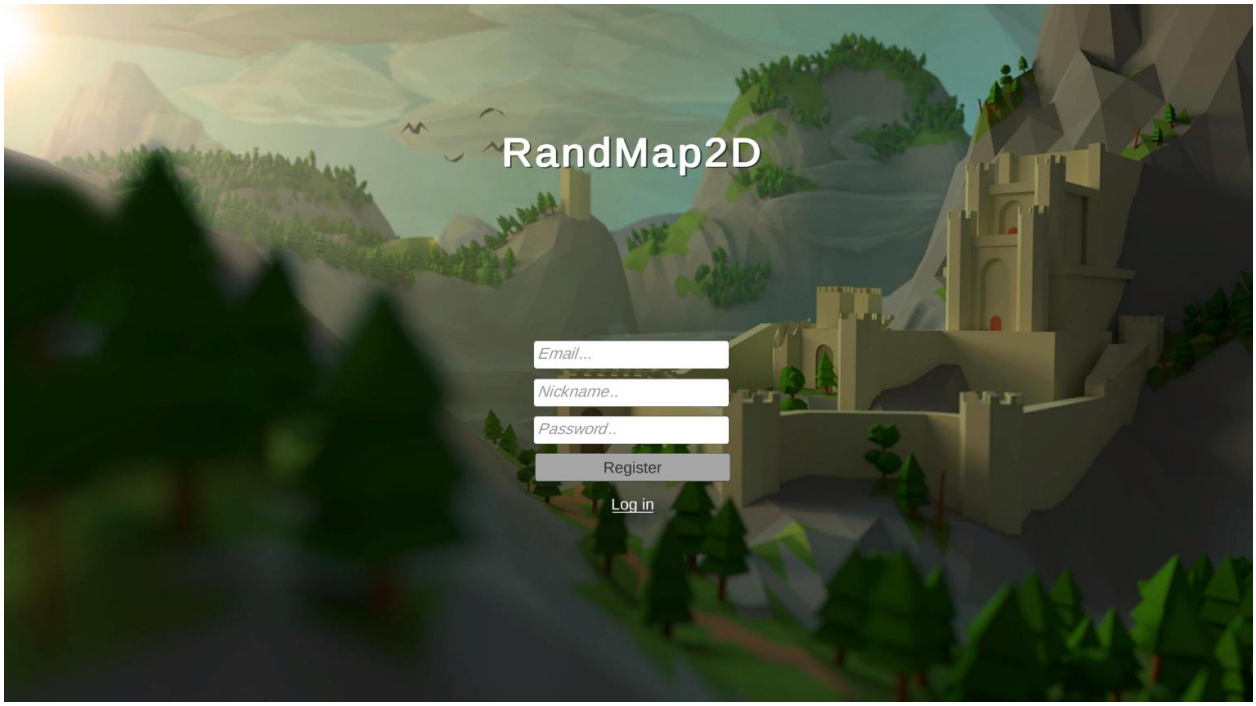


Рисунок 5.2 – Екран реєстрації у грі

Коли гравець авторизується у грі, для нього загружається карта з його поточною позицією, як показано на рисунку 5.3. У таблицях 5.1 та 5.2 наведений детальний опис всіх об’єктів, які представлені на карті.

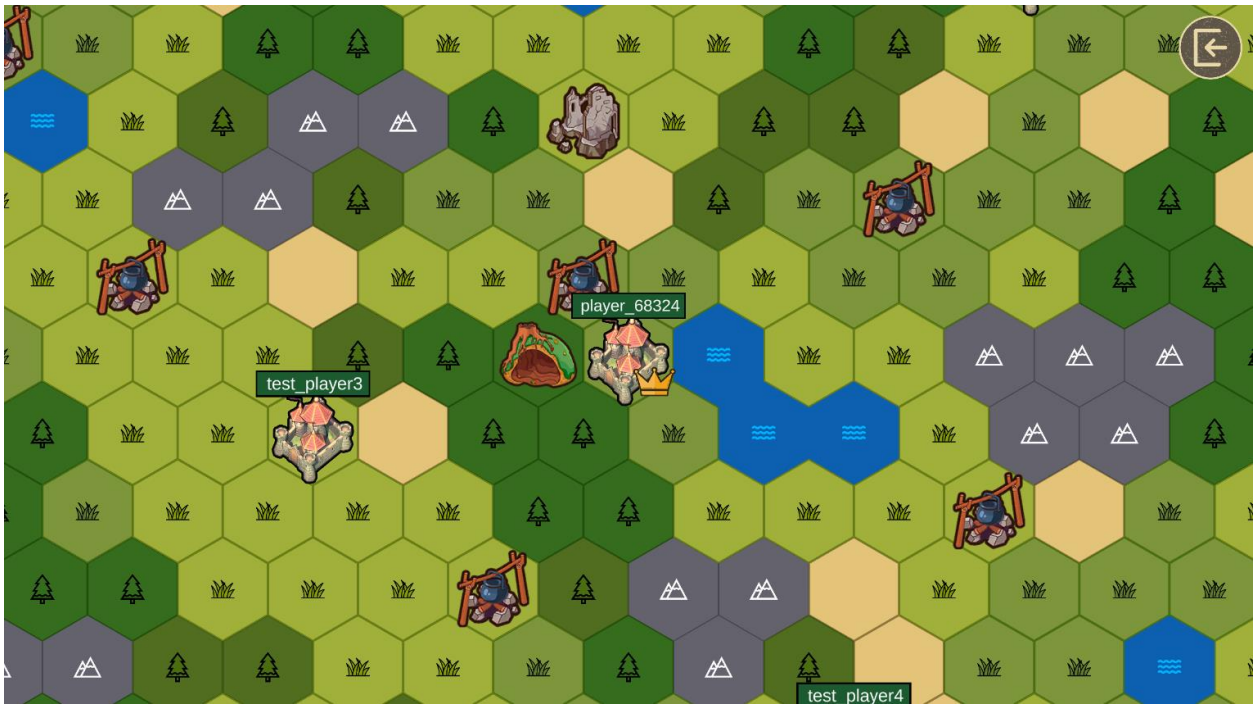
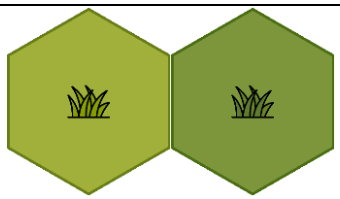
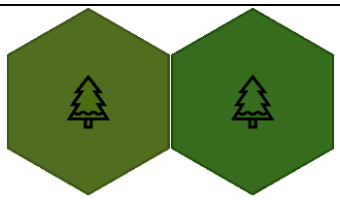


Рисунок 5.3 – Приклад згенерованої карти

Таблиця 5.1 – Опис клітин карти

Тип клітини	Візуальний вигляд	Кількість варіантів	Чи проходима клітина?
Галявина (Plain)		2	Так
Ліс (Forest)		2	Так
Море (Sea)		1	Ні
Пустеля (Desert)		1	Так
Гори (Mountains)		1	Ні

Таблиця 5.2 – Опис вузлів на карті

Тип вузла	Необхідний тип клітини	Візуальний вигляд	Примітки
PlayerNode	Plain		Індикатор корони з'являється лише поруч із замком авторизованого гравця

ResourceNode	Forest		Нема
ResourceNode	Plain		Нема
ResourceNode	Plain		Нема

Будь-який вузол на карті доступний для того, щоб взаємодіяти з ним. Для цього користувач має натиснути на той вузол, який його цікавить. Автоматично буде побудовано та показано найкоротший маршрут та відкриється вікно взаємодії (рисунки 5.4 та 5.5).

Рисунок 5.4 – Взаємодія з гравцем

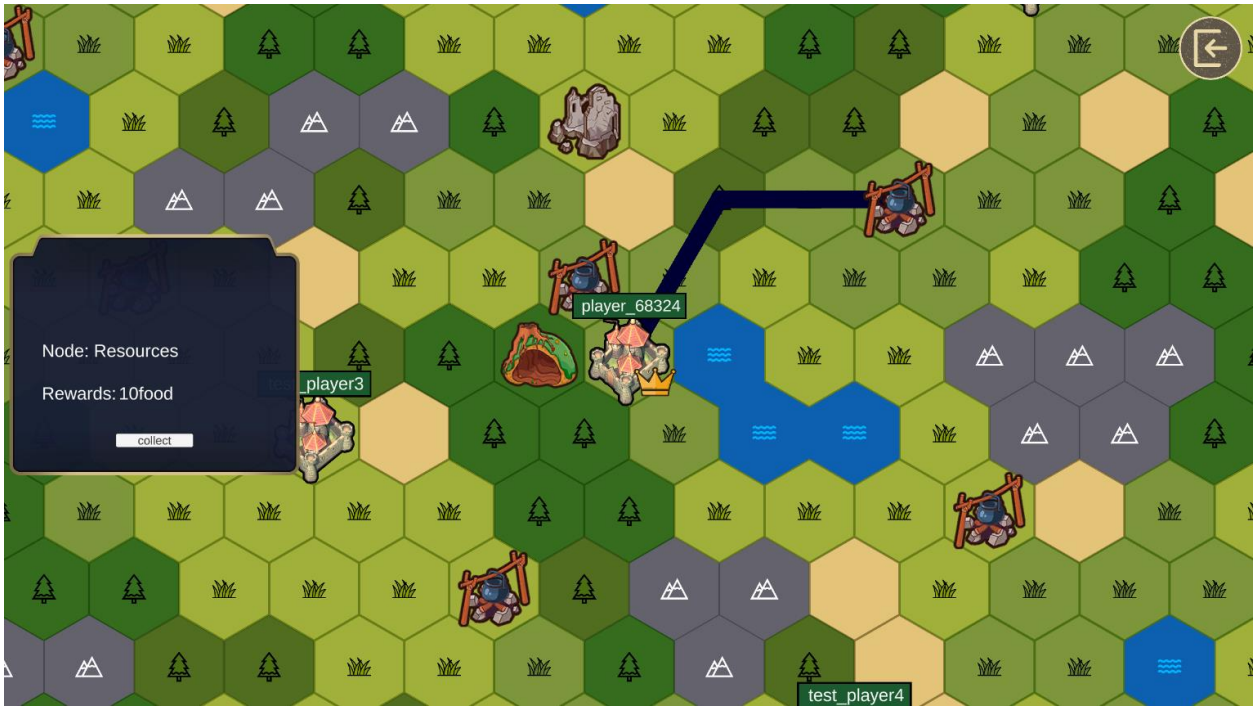


Рисунок 5.5 – Взаємодія з ресурсним вузлом

Для адміністратора реалізована можливість переміщати гравців по карті. Для цього необхідно клікнути курсором по необхідному гравцю. Після цього поточна позиція гравця на карті стане більш прозорою і вузол буде переміщатись за курсором адміністратора. Для того, щоб поставити гравця на нову клітину, адміністратору необхідно натиснути на обрану клітину.

Адміністратору варто враховувати, що клітини гравців можуть бути розташовані лише на певних типах клітин, а отже не можна розташувати гравця на горах чи у морі. У випадку, коли адміністратор намагається перемістити гравця на некоректну позицію, адміністратор буде отримувати сповіщення про помилку під час операції.

Гравець, у свою чергу, отримує сповіщення, що його позиція була змінена, як тільки адміністратор закінчить процес переміщення.

Приклад переміщення наведено на рисунку 5.6.

Рисунок 5.6 – Переміщення гравця по карті адміністратором

5.2 Випробовування програмного продукту

5.2.1 Мета випробувань

Метою випробувань є перевірка відповідності функцій комплексу задач генерації 2D карти на пошуку найкоротшого шляху між її клітинами вимогам технічного завдання.

Для досягнення мети необхідно визначити та описати набір тестових сценаріїв, які охоплюються весь розроблений функціонал. Згідно з розробленими тестами, потрібно провести їх на скомпільованому проєкті і переконатись, що система видає коректний результат у кожному із запропонованих тестів.

5.2.2 Загальні положення

Для ефективного тестування системи генерації 2D-карти та пошуку найкоротшого шляху на ній рекомендується використовувати комбінацію наступних підходів тестування[15]:

Функціональне тестування – перевіряє, чи працюють основні функції системи правильно. Необхідно виконати тести, щоб перевірити, чи генерує

система коректні карти, чи знаходить найкоротший шлях між точками та чи враховує вона обмеження та правила.

Тестування продуктивності – такі тести необхідні для оцінки швидкості генерації карт та пошуку шляху на них. Перевіряється, що система працює ефективно навіть з великими картами та великою кількістю точок.

Тестування реальних сценаріїв: мають бути розроблені тестові сценарії, які відповідають реальному використанню системи. Вони відтворюють типові ситуації, з якими користувачі можуть зіткнутися, і перевіряє, чи працює система так, як очікується у реальних умовах.

5.2.3 Результати випробувань

У цьому розділі будуть розглянуті розроблені сценарії тестування взаємодії з системою. Результати випробувань наведені у таблицях 5.3-5.7.

Таблиця 5.3 – Тестування реєстрації користувача у системі

Ціль тесту	Перевірка реєстрації користувача
Передумова	Користувач знаходиться у меню реєстрації
Вхідні дані	Інформація про користувача
Послідовність кроків	– у полі «Email» ввести унікальну пошту англійськими літерами, наприклад «anna.kurenna@gmail.com»; – у полі «Password» ввести унікальний пароль; – у полі «Nickname» ввести бажаний нік, наприклад «tteaman».
Очікуваний результат	Користувач зареєстрований у системі
Система після тестування	– користувач знаходиться на ігровій карті; – дані користувача збережені в системі; – користувачу присвоєна унікальна позиція на карті;

	– за необхідності, згенеровано додатковий регіон для розташування користувача.

Таблиця 5.4 – Тестування оновлення карти з часом

Ціль тесту	Перевірка оновлення вузлів карти
Передумова	Користувач знаходиться на ігровій карті
Вхідні дані	Координати вибраного регіону користувачем
Послідовність кроків	Користувач очікує 30 секунд на карті
Очікуваний результат	Карта оновлюється в залежності від змін, які відбулись за час очікування
Система після тестування	Користувач знаходиться на ігровій карті

Таблиця 5.5 – Тестування пошуку найкоротшого шляху

Ціль тесту	Перевірка побудови найкоротшого маршруту
Передумова	Користувач знаходиться на ігровій карті
Вхідні дані	Координати розташування гравця і кінцевої точки
Послідовність кроків	Натиснути на обраний вузол на карті
Очікуваний результат	– відображається найкоротший шлях до вузла на карті, оминаючи перешкоди; – відкривається вікно взаємодії з вузлом.
Система після тестування	Користувач має вибраний ігровий вузол на карті

Таблиця 5.6 – Тестування переміщення гравця адміністратором на коректну клітину

Ціль тесту	Перевірка переміщення гравця
Передумова	Адміністратор знаходиться на ігровій карті
Вхідні дані	Координати розташування гравця
Послідовність кроків	– натиснути на вузол гравця;

	– перемістити курсор на довільну вільну клітину такого ж типу, як і та, на якій був розташований гравець; – натиснути курсором на обрану вільну клітину.
Очікуваний результат	Гравець переміщений на нову позицію
Система після тестування	– система зберігає зміни про перестановку користувача; – користувач отримає сповіщення, що його позиція була змінена.

Таблиця 5.7 – Тестування переміщення гравця адміністратором на не коректну клітину

Ціль тесту	Перевірка переміщення гравця
Передумова	Адміністратор знаходиться на ігровій карті
Вхідні дані	Координати розташування гравця
Послідовність кроків	– натиснути на вузол гравця; – перемістити курсор на довільну вільну клітину відмінного типу від тої, на якій був розташований гравець; – натиснути курсором на обрану вільну клітину.
Очікуваний результат	Гравець залишається на поточній позиції
Система після тестування	– система не дозволяє зробити зміни про перестановку користувача; – адміністратор отримає сповіщення про помилку вибору позиції.

Висновки до розділу

У цьому розділі було розроблено керівництво користувача. Були описані основні ігрові елементи та розроблені інструкції для гравців та адміністраторів, як взаємодіяти із застосунком.

Були розглянуті популярні методи тестування і серед них виокремлені ті, які найкращим чином підійдуть для перевірки коректності функціонування підсистеми.

На основі 1 розділу були описані тест-кейси для основних функцій застосунків. Ці тест кейси були виконані на працюючій програмі та описані результати цих випробувань.

Створена підсистема показала очікуванні результати, отже її роботу можна вважати повністю коректною.

ВИСНОВКИ

Розробка багатокористувацькї гри з глобальною картою є складним завданням, що вимагає використання різноманітних компетенцій та технологій. В такому проекті ставиться за мету створення захоплюючої гри, яка дозволить користувачам насолоджуватись унікальним досвідом і взаємодіяти з величезним світом. Проект спрямований на створення ігрового процесу, що запропонує користувачам можливість взаємодії з іншими гравцями.

Головною задачею проекту є створення рішення, яке забезпечить генерацію карти відповідно до всіх правил її створення. Це включає в себе розміщення гравців та ресурсів на карті, враховуючи задані умови розстановки. Основним вимогам є забезпечення максимальної оптимізації використаних ресурсів під час процесу генерації карти. Це означає, що система повинна здійснювати ефективний розподіл обчислювальної потужності та пам'яті для швидкого та безперебійного створення карт. Крім того, рішення повинно дотримуватись правил, встановлених для генерації карт, забезпечуючи відповідну розмітку регіонів, територій і ресурсів на карті. Проєкт має на меті створити ефективне та оптимізоване рішення, яке генерує карту з точністю до правил та забезпечує ефективне використання ресурсів під час цього процесу.

На першому етапі розробки дипломного проекту проведено аналіз аналогів та визначено функціональні вимоги до системи. Шляхом вивчення схожих проектів вдалося ідентифікувати їх переваги, недоліки та особливості. Це дало змогу уточнити потреби користувачів та виявити можливості для покращення. На основі цього аналізу були сформульовані функціональні вимоги, які визначають необхідні функції та обмеження системи.

Наступним етапом було розроблено модель бази даних та виокремлено основні дата-об’єкти застосунку. Були продумані та описані ключові параметри кожної із сутностей бази даних.

Важливим етапом було розв’язання математичної задачі. У цьому розділі були розглянуті популярні варіанти вирішення поставленої проблеми. Найкращим рішенням було обрано і обгрунтовано використовувати алгоритм A* для пошуку найкоротшого шляху гра графі клітин карти.

На завершальному етапі розробки дипломного проекту було описано основні компоненти та класи застосунку. Створено діаграми послідовності, щоб показати черговість дії і, сервіси, які ці дії виконують. Також було безпосередньо реалізовано підсистему інструментами ігрового рушія Unity Engine та веб-сервісу ASP.Net.

Розроблена підсистема може бути використана як окремий і незалежний модуль. Вона реалізує базовий функціонал створення карти за певними правилами та взаємодію з цією картою. Підсистема має великий потенціал для розширення. Її можна розширювати різноманітними способами, наприклад додаючи ігровий контент та правила його генерації, або створення нових правих генерації чи пошуку шляху.

ПЕРЕЛІК ПОСИЛАНЬ

1. Maung, D. Tile-based Method of Procedural Content Generation. 2016.
2. 12 Best Games like Civilization for Android & iOS. [Електронний ресурс]. – Режим доступу https://freeappsforme.com/games-like-civilization/. 26.04.2023
3. Dijkstra E. W. A Note on Two Problems in Connexion with Graphs. 1959. P. 269–272.
4. Cormen T. H. , Leiserson C. E. , Rivest R. L. , Stein C. Introduction to Algorithms, 3rd Edition. 2009. P. 199
5. Cherkassky B., Goldberg A., Radzik T. Shortest paths algorithms: Theory and experimental evaluation. 1994. P 84-86
6. APIs with ASP.NET Core. [Електронний ресурс]. – Режим доступу https://dotnet.microsoft.com/en-us/apps/aspnet/apis. 13.05.2023
7. Dealessandri M. What is the best game engine: is Unity right for you?. [Електронний ресурс]. – Режим доступу https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you. 12.05.2023
8. Troelsen A. Professional C# 5.0 and .NET 4.5. 2013. P. 1440
9. Freeman A. Pro ASP.NET MVC 5 Platform. 2014. P. 736
10. Методи передачі даних (GET і POST). [Електронний ресурс]. – Режим доступу https://uk.php.brj.cz/metodi-peredaci-danih-get-i-post. 13.05.2023
11. Entity Framework documentation hub. [Електронний ресурс]. – Режим доступу https://learn.microsoft.com/en-us/ef/. 14.05.2023
12. Three-Layered Services Application [Електронний ресурс]. – Режим доступу https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648105(v=pandp.10).13.05.2023
13. Повне розуміння діаграми компонентів UML за допомогою легкого методу [Електронний ресурс]. – Режим доступу https://www.mindonmap.com/uk/blog/uml-component-diagram/. 14.05.2023

14. Data-Driven Programming. [Електронний ресурс]. – Режим доступу https://web.archive.org/web/20220609080948/https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch09s01.html. 14.05.2023
15. Crispin L. Gregory J. Agile Testing: A Practical Guide for Testers and Agile Teams. 2009. P. 464

Додаток А

Тексти програмного коду

GlobalMapGenerator

```
using BusinessLogic.GlobalMap;
using DatabaseLayer;
using System.Linq;

public class GlobalMapGenerator
{
    private static IGlobalMapGenerationConfig Config { get; set; }
    private static float Modifier = 3;

    private static PerlinNoise TypeNoise;
    private static PerlinNoise PrefabVariantNoise;

    public static void Initialize(IGlobalMapGenerationConfig config)
    {
        Config = config;

        TypeNoise = new PerlinNoise(Config.GenerationSeed);
        PrefabVariantNoise = new PerlinNoise(Config.PrefabGenerationSeed);
    }

    public static IHex Get(int x, int y)
    {
        var x0 = x / Modifier;
        var y0 = y / Modifier;

        var resType = GetTileType(x0, y0);

        var config = Config.GetConfig(resType);

        var prefabVariant = GetPrefabVariant(config, x0, y0);

        var isWalkable = config.isPassable;

        var hex = new Hex(x, y, resType, prefabVariant, isWalkable);

        return hex;
    }

    private static GlobalMapTileType GetTileType(float x, float y)
    {
        var randomValue = TypeNoise.Noise(x, y, 0) + 0.5f;

        if (randomValue < 0)
            randomValue = 0;

        if (randomValue >= 1)
            randomValue = 0.9999f;

        var config = Config.TileConfigs.FirstOrDefault(c => c.minRandValue <=
randomValue && randomValue < c.maxRandValue);

        var targetType = config.tileType;
        return targetType;
    }

    private static int GetPrefabVariant(MapNodeClass config, float x, float y)
    {
        var maxVariantsCount = config.prefabsCount;
        if (maxVariantsCount == 1)
```

```
            return 0;

        var randomValue = PrefabVariantNoise.Noise(x, y, 0) + 0.5f;

        var prefabVariant = GetQuantumIndex(randomValue, config.prefabsCount);
        return prefabVariant;
    }

    private static int GetQuantumIndex(double randomValue, int categoriesCount)
    {
        if (categoriesCount == 1)
            return 0;

        var step = (float)1 / categoriesCount;
        var currentValue = 0f;

        for (var i = 0; i < categoriesCount; i++)
        {
            currentValue += step;

            if (randomValue <= currentValue)
            {
                return i;
            }
        }

        return 0;
    }
}
GlobalMapRegionGenirator
using DatabaseLayer;
using System;
using System.Collections.Generic;
using System.Linq;
using Utillities;

namespace BusinessLogic
{
    public class GlobalMapRegionGenirator
    {
        private Random Random { get; } = new Random();
        private ConfigurationService ConfigurationService { get; set; }

        public GlobalMapRegionGenirator(ConfigurationService configurationService)
        {
            ConfigurationService = configurationService;


GlobalMapDatabase.Instance.Initialize(ConfigurationService.LoadConfiguration());
            GlobalMapGenerator.Initialize(GlobalMapDatabase.Instance);
        }

        public List<GlobalMapEntityState> GenerateNew(CustomVector2 region)
        {
            var generatedEntities = new List<GlobalMapEntityState>();

            return Generate(generatedEntities, true, region);
        }

        public List<GlobalMapEntityState> Regenerate(CustomVector2 region,
List<GlobalMapEntityState> allNodes)
        {
            var generatedEntities = new List<GlobalMapEntityState>();
```

```
            return Generate(generatedEntities, true, region);
        }

        private List<GlobalMapEntityState> Generate(
            List<GlobalMapEntityState> existingNodes,
            bool isNew,
            CustomVector2 region)
        {
            var results = new List<GlobalMapEntityState>();
            var regionDimention = StaticValues.RegionTilesCount;
            var regionCenter = region * regionDimention;

            var regionLeftTop = regionCenter - new CustomVector2((regionDimention /
2), (regionDimention / 2));
            var regionRightLower = regionCenter + new CustomVector2((regionDimention /
2), (regionDimention / 2));

            foreach (var requirementsConfig in
ConfigurationService.ResourceRequirements)
            {
                if (requirementsConfig.isPersistent && isNew == false)
                {
                    continue;
                }

                var positions = GeneratePositions(requirementsConfig, existingNodes,
regionLeftTop, regionRightLower);
                GenerateNodes(requirementsConfig, positions, region, existingNodes,
results);
            }

            return results;
        }

        private List<CustomVector2> GeneratePositions(
            ResourceNodeClassRequirement requirementsConfig,
            List<GlobalMapEntityState> existingNodes,
            CustomVector2 regionLeftTop,
            CustomVector2 regionRightLower)
        {
            var availableTypes = GetAvailableTypes(requirementsConfig);
            if (availableTypes.Any() == false)
            {
                return new List<CustomVector2>();
            }

            var pointsToGenerate = requirementsConfig.frequencyPerRegion;

            var alreadyGeneratedPoints = existingNodes
                .Where(n => availableTypes.Any(t => t.id == n.ContentId))
                .Select(n => n.Position);

            var positions = alreadyGeneratedPoints.ToList();

            var iterationsCount = 0;
            var maxIterationsCount = 5;

            while (positions.Count < requirementsConfig.frequencyPerRegion)
            {
                var pointsToGenerateCount = requirementsConfig.frequencyPerRegion -
positions.Count;

                for (var i = 0; i < pointsToGenerateCount; i++)
                {
```

```
                    var randomPosX = Random.Next((int)regionLeftTop.x,
(int)regionRightLower.x);
                    var randomPosY = Random.Next((int)regionLeftTop.y,
(int)regionRightLower.y);

                    var newPos = new CustomVector2(randomPosX, randomPosY);

                    var isPosRequirementsCompleted = CheckGeneratedPosition(newPos,
positions, existingNodes, requirementsConfig);

                    if (isPosRequirementsCompleted == false)
                        continue;

                    positions.Add(newPos);
                }

                iterationsCount++;

                if (iterationsCount > maxIterationsCount)
                    break;
            }

            return positions;
        }

        private void GenerateNodes(
            ResourceNodeClassRequirement requirementsConfig,
            List<CustomVector2> positions,
            CustomVector2 region,
            List<GlobalMapEntityState> existingNodes,
            List<GlobalMapEntityState> results)
        {
            foreach (var point in positions)
            {
                GlobalMapEntityState pointState = null;

                var tile = GlobalMapGenerator.Get((int)point.x, (int)point.y);

                var availableTypes = GetAvailableTypes(requirementsConfig);

                var allSuitableConfigs = availableTypes.Where(t => t.tileType ==
tile.Type);

                var randomConfigIndex = new Random().Next(0,
allSuitableConfigs.Count());
                var randomConfig =
allSuitableConfigs.Skip(randomConfigIndex).FirstOrDefault();

                if (randomConfig == null)
                {
                    continue;
                }

                if (randomConfig.nodeType == GlobalMapNodeType.PlayerNode)
                {
                    pointState = new PlayerNodeState();
                }
                else if (randomConfig.nodeType == GlobalMapNodeType.ResourceNode)
                {
                    pointState = new ResourceNodeState
                    {
                        ResourcesCapacity = randomConfig.interactionsCount,
                    };
                }
```

```
                if (pointState != null)
                {
                    pointState.Region = region;
                    pointState.Position = point;

                    pointState.ContentId = randomConfig.id;

                    results.Add(pointState);
                    existingNodes.Add(pointState);
                }
            }
        }

        private bool CheckGeneratedPosition(
            CustomVector2 newPos,
            List<CustomVector2> positions,
            List<GlobalMapEntityState> existingNodes,
            ResourceNodeClassRequirement requirementsConfig)
        {
            // Check pos requirements.
            var isAlreadyExisting = existingNodes.Any(n => n.Position == newPos) ||
positions.Contains(newPos);
            if (isAlreadyExisting)
                return false;

            var availableTypes = GetAvailableTypes(requirementsConfig);

            var tile = GlobalMapGenerator.Get((int)newPos.x, (int)newPos.y);

            var isAvailableToPlace = availableTypes.Any(t => t.tileType == tile.Type);
            if (isAvailableToPlace == false)
                return false;

            return true;
        }

        private IEnumerable<ResourceNodeClass>
GetAvailableTypes(ResourceNodeClassRequirement requirementsConfig)
        {
            return ConfigurationService.ResourceClasses.Where(c => c.priorityType ==
requirementsConfig.priorityType);
        }
    }
}

GlobalMapDatabase
using BusinessLogic;
using DatabaseLayer;
using System.Collections.Generic;
using System.Linq;

public class GlobalMapDatabase : IGlobalMapGenerationConfig
{
    #region Singleton
    private static GlobalMapDatabase _instance;
    public static GlobalMapDatabase Instance
    {
        get
        {
            if (_instance == null)
            {
                _instance = new GlobalMapDatabase();
            }
```

```
            return _instance;
        }
    }
    # endregion

    private List<MapNodeClass> _tileConfigs;

    public List<MapNodeClass> TileConfigs => _tileConfigs;

    public int GenerationSeed => 135;

    public int PrefabGenerationSeed => 268;

    public void Initialize(ConfigurationResponse configuration)
    {
        _tileConfigs = configuration.MapNodeClasses;
    }

    public MapNodeClass GetConfig(GlobalMapTileType type)
    {
        return TileConfigs.FirstOrDefault(c => c.tileType == type);
    }
}
```

PathFinding

```
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using Utillities;

public class PathFinding
{
    public static List<IHex> Find(CustomVector2 startPos, CustomVector2 endPos)
    {
        if (startPos == endPos)
            return null;

        var start = GlobalMapGenerator.Get((int)startPos.x, (int)startPos.y);
        var end = GlobalMapGenerator.Get((int)endPos.x, (int)endPos.y);

        if (end == null || start == null)
            return null;

        if (!end.IsWalkable)
            return null;

        bool isEndFound = false;
        Dictionary<IHex, IHex> previous = new Dictionary<IHex, IHex>();
        Queue<IHex> queue = new Queue<IHex>();

        var currentNode = start;
        queue.Enqueue(currentNode);

        while (queue.Count > 0)
        {
            currentNode = queue.Dequeue();

            if (currentNode.Position == end.Position)
            {
                isEndFound = true;
                break;
            }

            var neighbours = MapQuery.GetNeighborTiles(currentNode.Position);
```

```
            var orderedNeighbours = neighbours.OrderBy(n => FitFunction(n.Position,
endPos));

            foreach (var neighbor in orderedNeighbours)
            {
                var needToSkip = false;

                // If contains key.
                foreach (var key in previous.Keys)
                {
                    if (key.Position.Equals(neighbor.Position) == false)
                        continue;

                    needToSkip = true;
                    break;
                }

                if (needToSkip || (!neighbor.IsWalkable && !neighbor.Equals(end)))
                    continue;

                previous[neighbor] = currentNode;
                queue.Enqueue(neighbor);
            }
        }

        if (isEndFound == false)
            return null;

        return ShortestPath(previous, start, end);
    }

    private static List<IHex> ShortestPath(Dictionary<IHex, IHex> previous, IHex from,
IHex to)
    {
        var path = new List<IHex>();
        var current = to;

        while (current.Equals(from) == false)
        {
            path.Add(current);

            foreach (var key in previous.Keys)
            {
                if (key.Position.Equals(current.Position) == false)
                    continue;

                current = previous[key];
                break;
            }
        }

        path.Add(from);
        path.Reverse();

        return path;
    }

    private static float FitFunction(CustomVector2 currentPosition, CustomVector2
targetPosition)
    {
        int x1 = (int)currentPosition.x;
        int y1 = (int)currentPosition.y;
        int x2 = (int)targetPosition.x;
        int y2 = (int)targetPosition.y;
```

```
        int deltaX = Mathf.Abs(x2 - x1);
        int deltaY = Mathf.Abs(y2 - y1);
        int distance = deltaX + Mathf.Max(0, (deltaY - deltaX) / 2);

        return distance;
    }
}MapQuery
using System.Collections.Generic;
using Utillities;

public class MapQuery
{
    private static readonly CustomVector2[] _lowerCellDirections = new CustomVector2[]
                                                                   {
                                                                       new CustomVector2(0, 1),
                                                                       new CustomVector2(1, 1),
                                                                       new CustomVector2(1, 0),
                                                                       new CustomVector2(1, -1),
                                                                       new CustomVector2(0, -1),
                                                                       new CustomVector2(-1, 0)
                                                                   };

    private static readonly CustomVector2[] _upperCellDirections = new CustomVector2[]
                                                                  {
                                                                      new CustomVector2(-1, 1),
                                                                      new CustomVector2(0, 1),
                                                                      new CustomVector2(1, 0),
                                                                      new CustomVector2(0, -1),
                                                                      new CustomVector2(-1, -1),
                                                                      new CustomVector2(-1, 0)
                                                                  };

    public static CustomVector2[] GetNeighborPositions(CustomVector2 cellPosition)
    {
        CustomVector2[] currentDirections;

        if (cellPosition.y % 2 == 0 || cellPosition.y == 0)
        {
            currentDirections = _lowerCellDirections;
        }
        else
        {
            currentDirections = _upperCellDirections;
        }

        var neighborTilesPosition = new CustomVector2[]
                                    {
                                        cellPosition + currentDirections[0],
                                        cellPosition + currentDirections[1],
                                        cellPosition + currentDirections[2],
                                        cellPosition + currentDirections[3],
                                        cellPosition + currentDirections[4],
                                        cellPosition + currentDirections[5]
                                    };

        return neighborTilesPosition;
    }

    public static List<IHex> GetNeighborTiles(CustomVector2 cellPosition)
    {
        List<IHex> neighborTiles = new List<IHex>();
        var neighborTilesPosition = GetNeighborPositions(cellPosition);
```

```
        foreach (var neighborPosition in neighborTilesPosition)
        {
            var hex = GlobalMapGenerator.Get((int)neighborPosition.x,
(int)neighborPosition.y);
            neighborTiles.Add(hex);
        }

        return neighborTiles;
    }

}
IHex
using Utillities;

public interface IHex
{
    CustomVector2 Position { get; }
    GlobalMapTileType Type { get; }
    int PrefabVariant { get; }
    bool IsWalkable { get; }
}
Hex
using Utillities;

public class Hex : IHex
{
    public CustomVector2 Position { get; set; }
    public GlobalMapTileType Type { get; set; }
    public int PrefabVariant { get; set; }

    public bool IsWalkable { get; set; }

    public Hex(int column, int row, GlobalMapTileType type, int prefabVariant, bool
isWalkable)
    {
        Position = new CustomVector2(column, row);
        Type = type;
        PrefabVariant = prefabVariant;
        IsWalkable = isWalkable;
    }

    Hex()
    {
    }
}
GlobalMapEntityState
using System;
using System.Xml.Serialization;
using Utillities;

[Serializable()]
[XmlInclude(typeof(PlayerNodeState))]
[XmlInclude(typeof(ResourceNodeState))]
public abstract class GlobalMapEntityState
{
    // Is uniq and is used only for identification items on global map.
    public int EntityId { get; set; }

    // Not unic id, should have apropriate configuration.
    public int ContentId { get; set; }

    public CustomVector2 Region { get; set; }

    public CustomVector2 Position { get; set; }
```

```
    public abstract GlobalMapNodeType Type { get; }

    public DateTime LastUpdate { get; set; }
}
PlayerNodeState
using System;

public class PlayerNodeState : GlobalMapEntityState
{
    public string PlayerId { get; set; }

    public bool IsEmpty => string.IsNullOrEmpty(PlayerId);

    public override GlobalMapNodeType Type => GlobalMapNodeType.PlayerNode;

    public string NickName { get; set; }
    public long CityLevel { get; set; }
}
ResourceNodeState
public class ResourceNodeState : GlobalMapEntityState
{
    public override GlobalMapNodeType Type => GlobalMapNodeType.ResourceNode;

    public int ResourcesCapacity { get; set; }

    public bool IsNeedToBeRecovered => ResourcesCapacity <= 0;
}
GlobalMapNodeConfig
public class GlobalMapNodeConfig
{
    public int Id { get; set; }
    public float OperationDurationHours { get; set; }

    public GlobalMapNodeType Type { get; set; }
    public GlobalMapNodeGenerationPriority Priority { get; set; }

    public GlobalMapTileType[] RequiredTileTypes { get; set;
```