Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра обчислювальної техніки
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«МЕТОД СТВОРЕННЯ КРОСПЛАТФОРМЕННИХ КАЗУАЛЬНИХ ІГРОВИХ ЗАСТОСУНКІВ»

08-54.МКР.009.00.000 ПЗ

Виконав: студент 2-го курсу, групи
1КІ-23м спеціальності 123 «Комп'ютерна інженерія »
(шифр і назва напряму підготовки, спеціальності)
Крейчі В.Б.
(прізвище та ініціали)

Керівник: к. т. н., доц. каф. ОТ
Савицька Л.А.
(прізвище та ініціали)
«_____»_____2024 р.

Опонент: д.т.н., доц. каф.
Шиян А.А
(прізвище та ініціали)
«_____»_____2024 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
(прізвище та ініціали)
«_____»_____2024 р.

Вінниця ВНТУ- 2024 рік

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 123 « Комп'ютерна інженерія »

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., професор Азаров О.Д.
"18" вересня 2024 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТА

Крейчі Владислава Богдановича
(прізвище, ім'я, по батькові)

1 Тема роботи «Метод створення кросплатформенних казуальних ігрових застосунків», керівник роботи к. т. н., доц. кафедри ОТ Савицька Л.А., затверджені наказом вищого навчального закладу від "17" вересня 2024 року № 310.

2 Строк подання студентом роботи: 16.12.2024

3 Вихідні дані до роботи: ігровий рушій Unity, параметри рівня (розмір 9 на 18), операційна система Windows.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз предметної області, проєктування інструменту для побудови рівнів, програмна реалізація інструменту для побудови рівнів, економічна частина, література.

5 Перелік ілюстративного матеріалу: архітектура інструменту створення та побудови ігрових рівнів, блок-схема алгоритму роботи, лістинг модуля створення шаблонів, лістинг модуля побудови ігрових рівнів, протокол перевірки кваліфікаційної роботи на наявність текстових запозичень.

6 Консультанти розділів роботи представлено в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийня
1–3	Савицька Л.А., к. т. н., доцент каф. ОТ		
4	Небава М. І., доцент, к.е.н., професор каф. ЕПВМ		

7 Дата видачі завдання: 18.09.2024

8 Календарний план роботи зазначений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	19.09.2024	виконано
2	Аналіз предметної області	20.09.2024–25.09.2024	виконано
3	Проєктування інструменту для побудови рівнів	26.09.2024 – 05.10.2024	виконано
4	Програмна реалізація інструменту для побудови рівнів	06.10.2024 – 26.10.2024	виконано
5	Розрахунок економічної частини	11.11.2024	виконано
6	Оформлення матеріалів до захисту МКР	12.11.2024	виконано
7	Перевірка якості виконання магістерської роботи та усунення недоліків	13.11.2024 – 14.11.2024	виконано
8	Підписи супроводжувальних документів у нормоконтролера, керівника, опонента	15.11.2024 – 16.11.2024	виконано
9	Перевірка на антиплагіат та ШІ	17.11.2024	виконано
10	Попередній захист роботи	18.11.2024	виконано

Студент _____ Крейчі В.Б.
(підпис)

Керівник роботи _____ Савицька Л.А
(підпис)

АНОТАЦІЯ

УДК 004.9

Крейчі В.Б. Методи створення кросплатформенних казуальних ігрових застосунків. Магістерська кваліфікаційна робота зі спеціальності 123 – Комп’ютерна Інженерія,. Вінниця: ВНТУ, 2024. 116 с.

На укр. мові. Бібліогр. назв: 41; рис.: 24 ; табл.6.

Магістерська кваліфікаційна робота описує розробку інструменту для створення та побудови ігрових рівнів у казуальних кросплатформенних ігрових застосунках. У роботі використано комбінацію процедурної та шаблонної генерації для забезпечення варіативності та контролю над якістю рівнів. Програмна реалізація виконана з використанням C#, Unity та ScriptableObject, що забезпечує зменшення обсягу збережених даних та оптимізацію пам’яті. Інструмент протестовано на практичних прикладах, що підтвердило його ефективність у зменшенні часу розробки рівнів і використання ресурсів пристроїв. Результати роботи можуть бути використані у розробці казуальних ігор, мобільних застосунків та інтерактивного контенту. Робота містить 116 сторінок, 24 рисунків, 6 таблиць.

Ключові слова: процедурна генерація, 2d рівні, unity, оптимізація пам’яті, динамічний дизайн ігрового процесу.

ABSTRACT

Kreichi V.B. Methods for creating cross-platform casual gaming applications. Master's qualification thesis in the specialty 123 – Computer Engineering, Vinnytsia: VNTU, 2024. 116 pages.

In Ukrainian. Bibliography: 41 titles; figures: 24; tables: 6.

The master's qualification thesis describes the development of a tool for creating and constructing game levels in cross-platform casual gaming applications. The work utilizes a combination of procedural and template generation to ensure variability and quality control of levels. The software implementation was carried out using C#, Unity, and ScriptableObject, which provides reduced data storage volume and memory optimization. The tool was tested on practical examples, confirming its efficiency in reducing level development time and resource consumption of devices. The results of the work can be used in the development of casual games, mobile applications, and interactive content. The thesis contains 116 pages, 24 figures, and 6 tables.

Keywords: procedural generation, 2D levels, Unity, memory optimization, dynamic game process design.

ЗМІСТ

ВСТУП

Казуальні ігри, **актуальність** яких не зменшується з часом, залишаються одними з найпопулярніших серед гравців завдяки своїй доступності, простоті та можливості швидкого розроблення. Вони орієнтовані на широкий загал гравців і не вимагають значних зусиль для освоєння, що робить їх ідеальними для коротких перерв і дозвілля. Прості механіки, привабливий візуальний стиль та легкість у грі сприяють масовому поширенню цих ігор, зокрема на мобільних платформах. Завдяки своїй універсальності, казуальні ігри стали популярними не лише серед гравців, а й серед розробників, особливо малих команд, що прагнуть створити кросплатформенні продукти із залученням мінімальних ресурсів.

Проте створення таких ігор, зокрема розробка та інтеграція ігрових рівнів, є одним з найдовших і трудомістких процесів. Створенні рівні визначають загальну привабливість гри, оскільки вони формують ігровий досвід користувача. Традиційні підходи до збереження рівнів, такі як використання префабів або інструментів на кшталт TileMap Editor у Unity, стикаються з проблемами, пов'язаними з надмірним використанням ресурсів та обмеженою гнучкістю. Це, своєю чергою, створює значні труднощі для розробників, які мають працювати в умовах обмежених ресурсів, особливо коли йдеться про створення кросплатформенних ігор.

Враховуючи виклики, пов'язані зі збереженням рівнів та оптимізацією ресурсів, постала необхідність розробки нових підходів до автоматизованої генерації рівнів, які могли б знизити навантаження на систему та полегшити процес розробки.

Все це пояснює актуальність вдосконалення методу створення казуальних кросплатформенних ігрових застосунків, а саме створення інструменту для створення і генерації рівнів.

В даній роботі буде розглянута програмна реалізація інструменту для створення та генерації рівнів. Цей інструмент зможе знайти попит серед розробників, які працюють над створенням кросплатформенних казуальних ігрових застосунків.

Метою дослідження є зменшення використання ресурсів пристрою та часу на побудову ігрових рівнів за рахунок створення інструменту для побудови, зберігання та динамічного створення ігрових рівнів. Для досягнення поставленої мети потрібно виконати наступні завдання:

— проаналізувати сучасні кросплатформенні казуальні ігрові застосунки та методи їх створення;

— дослідити існуючи методи створення ігрових рівнів;

— обрати оптимальні засоби створення ігрових рівнів;

— розробити прототип інструменту для створення та побудови ігрових рівнів;

— протестувати роботу інструменту для створення та побудови ігрових рівнів;

— провести тестування розробленого інструменту.

Об’єкт дослідження процес побудови ігрових рівнів у контексті розробки кросплатформенних казуальних ігор.

Предмет дослідження інструменти і методи побудови ігрових рівнів кросплатформенних казуальних ігор.

Для вирішення поставлених завдань було застосовано **методи дослідження** подання знань, математичного моделювання, тестування програмних засобів та об’єктно-орієнтованого програмування.

Наукова новизна роботи полягає у вдосконалені методу побудови ігрових рівнів, який поєднує шаблонний та процедурний підходи для автоматизованої генерації рівнів, що дозволяє зменшити час розробки та оптимізувати використання пам'яті.

Практичне значення роботи полягає у створенні інструменту для ігрового рушія, який автоматизує побудову рівнів та забезпечує швидке створення шаблонів, що суттєво скорочує час розробки та зменшує вимоги до системних ресурсів.

Запропонований інструмент сприяє підвищенню ефективності процесу створення ігрових рівнів:

— розроблено алгоритм створення і редагування шаблонів рівнів, який дозволяє ефективно створювати основу ігрових рівнів і зберігати їх для подальшої генерації ігровим рушієм;

— розроблено алгоритм побудови ігрових рівнів по вказаній черзі шаблонів, що дозволяє створювати ігрові рівні під час ігрового процесу не зберігаючи при цьому зайві компоненти;

— програмний комплекс для спрощення побудови ігрових рівнів.

Апробація результатів роботи було здійснено на міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи» та LIV Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії.

За результатами виконання магістерської кваліфікаційної роботи було **опубліковано** тезу доповіді та наукову статтю на науково-технічних конференціях:

Крейчі, В.; Савицька, Л. МЕТОД СТВОРЕННЯ КРОСПЛАТФОРМЕННИХ КАЗУАЛЬНИХ ІГРОВИХ ЗАСТОСУНКІВ. Молодь в науці: дослідження, проблеми, перспективи, Ukraine, may. 2024. Available at: <https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21706>. [1]

Л. А. Савицька, В. Б. Крейчі, Л. В. Крупельницький. Інформаційна технологія автоматизованої генерації 2D рівнів у середовищі Unity. //[Електронний ресурс] / Л. А. Савицька // Матеріали LIV Всеукраїнської науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії, Вінниця, 24-27 березня 2025 р. – Електрон. текст. дані. – 2024. – Режим доступу: <https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/22730>. [2]

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАДАЧІ СТВОРЕННЯ ІНТЕЛЕКТУАЛЬНОГО МОДУЛЯ ДЛЯ ГРИ У ЖАНРІ СТРАТЕГІЯ В РЕАЛЬНОМУ ЧАСІ

1.1 Історичний розвиток казуальних ігор та їхня популярність

Казуальні ігри, відомі своєю доступністю та простотою, існували задовго до появи цифрових технологій. Першими прикладами таких розваг були паперові головоломки та настільні ігри, що слугували засобом швидкого відпочинку та інтелектуального розвитку для широкого кола людей. Вони не вимагали спеціальних навичок чи значних часових інвестицій, що робило їх популярними серед різних соціальних груп.

З появою електронних технологій у другій половині XX століття концепція казуальних ігор плавно перейшла у цифровий світ. У 1970-1980-х роках на ринок вийшли аркадні ігри на зразок Pac-Man (рис. 1.1), які стали символами нового етапу розвитку ігрової індустрії. Ці ігри характеризувалися простою, але захопливою механікою, інтуїтивно зрозумілим керуванням та поступово зростаючою складністю, що робило їх привабливими для широкої аудиторії, незалежно від віку та досвіду.

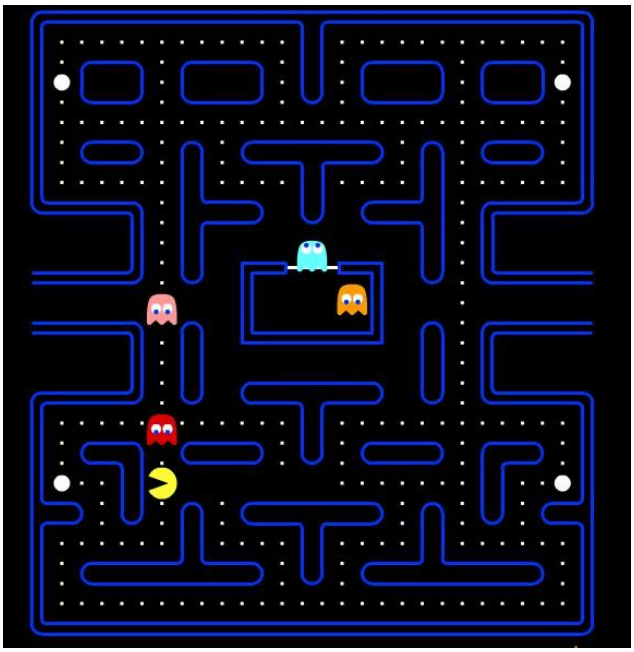

Рисунок 1.1 — Ігровий процес гри «Pac-Man».

З розвитком ігрової індустрії стало зрозуміло, що казуальні ігри змінили уявлення про відеоігри та їхніх гравців. Казуальні ігри стали свого роду "революцією", розширивши аудиторію за рахунок залучення людей, які раніше не вважали себе геймерами. Це змінило і підходи в дизайні ігор — на перший план вийшли простота, доступність та можливість швидко отримати задоволення від гри.

Поява інтернету та мобільних технологій на початку XXI століття значно вплинула на популяризацію казуальних ігор. Смартфони та планшети зробили ці ігри доступними будь-де та будь-коли, що ідеально вписувалося в сучасний стиль життя, орієнтований на мобільність та швидкий доступ до розваг. Ігри на зразок Angry Birds (рис. 1.2) стали справжніми глобальними феноменами, залучаючи мільйони гравців по всьому світу.

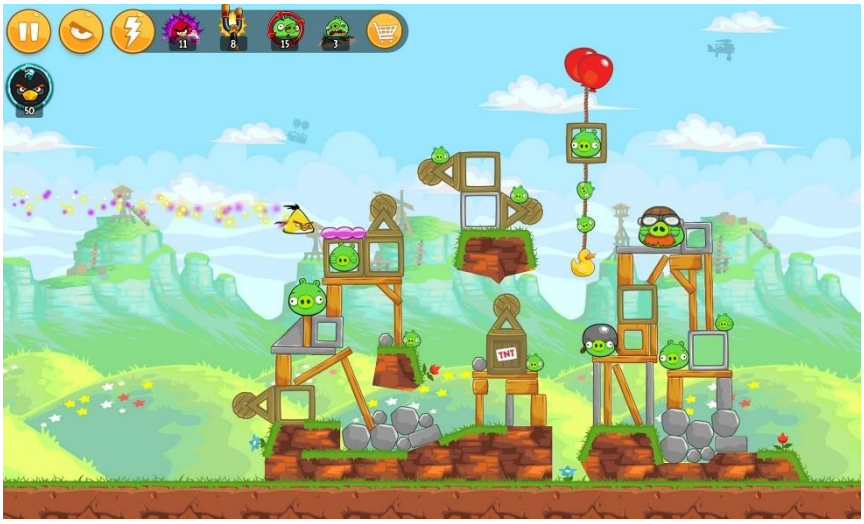

Рисунок 1.2 — Ігровий процес гри «Angry Birds».

Соціальні мережі також відіграли важливу роль у поширенні казуальних ігор. Інтеграція ігор із соціальними платформами перетворила їх на соціальний досвід, де гравці могли взаємодіяти, змагатися між собою і ділитися досягненнями. Це не лише підвищило залученість гравців, але й сприяло вірусному поширенню ігор завдяки особистим рекомендаціям та соціальній взаємодії.

Казуальні ігри відіграли ключову роль у тому, щоб зробити відеоігри популярними серед масової аудиторії. Вони допомогли подолати стереотипи про відеоігри як нішеву розвагу, доступну лише для молоді та технічно обізнаних користувачів. Сучасні казуальні ігри продовжують інтегрувати новітні технології, такі як доповнена реальність, що демонструє, як ці ігри можуть поєднувати фізичний та цифровий світи, створюючи унікальний ігровий досвід. До прикладу, гра Pokémon GO не лише розважає, але й стимулює фізичну активність та соціальну взаємодію між гравцями.

Рисунок 1.3 — Ігровий процес гри «Pokémon GO».

Казуальні ігри продовжують залишатися вагомим сегментом ігрової індустрії, який об'єднує різноманітні категорії гравців та використовує інноваційні підходи для покращення ігрового досвіду. Тому створення рівнів для таких ігор — це вагома частина роботи, яку необхідно постійно вдосконалювати, особливо в контексті оптимізації ресурсів та автоматизованої генерації.

1.2 Аналіз проблем і викликів при створенні казуальних ігрових застосунків

Сучасний ринок казуальних ігор зазнав істотних змін під впливом технологічних інновацій, соціальних механізмів та широкого розповсюдження

мобільних платформ. Ці зміни не лише вплинули на сам процес розробки таких ігор, а й створили нові вимоги до розробників, які тепер повинні забезпечувати ефективну соціальну інтеграцію, підтримку кросплатформенності та динамічну генерацію унікальних ігрових рівнів. Еволюція казуальних ігор від простих браузерних програм до складних мобільних інтегрованих систем, що взаємодіють із соціальними мережами, демонструє їхню здатність адаптуватися до швидких змін в уподобаннях користувачів. Ця еволюція сприяла також розширенню аудиторії: низький поріг для входу дозволяє казуальним іграм залучати широку та різноманітну аудиторію, від дітей до літніх людей. Однак, їхня популярність породжує і значні виклики, зокрема питання збереження простоти та доступності гри при одночасній інтеграції сучасних технологій, а також забезпеченні постійного потоку цікавого контенту для утримання інтересу гравців.

Однією з ключових тенденцій у сучасних казуальних іграх є інтеграція соціальних механізмів, що стимулюють взаємодію між гравцями та формування спільнот. Соціальні мережі, такі як Facebook, перетворили казуальні ігри на платформи для соціальної взаємодії, де гравці можуть змагатися, ділитися досягненнями, а також запрошувати друзів до гри. Такі механіки відіграють ключову роль у підтримці високого рівня активності серед користувачів, а також у сприянні вірусному поширенню ігор через їхню популяризацію серед соціальних контактів. Наприклад, гра Candy Crush Saga (рис. 1.4) активно використовує соціальні інтеграції для підвищення зацікавленості гравців — механізми, що дозволяють змагатися у рейтингах чи надсилати бонуси друзям, сприяють розширенню аудиторії шляхом залучення нових гравців. Вірусний маркетинг у поєднанні з механікою соціальної взаємодії є потужним засобом залучення і утримання користувачів, що підтверджує важливість інтеграції соціальних компонентів у казуальні ігри для їхнього успіху.

Рисунок 1.4 — Ігровий процес гри «Candy Crush Saga».

Кросплатформенність є ще однією важливою характеристикою, яка визначає сучасний розвиток казуальних ігор. Гравці очікують можливості продовжувати гру на різних пристроях, забезпечуючи безперервність ігрового процесу. Кросплатформенність підвищує зручність користування і дозволяє розробникам економити ресурси, створюючи єдину версію гри, придатну для різних платформ. Це не тільки полегшує процес розробки, але й розширює аудиторію, оскільки гра стає доступною для більшої кількості пристроїв. У цьому контексті Candy Crush Saga є яскравим прикладом успішної реалізації кросплатформенності, оскільки забезпечує можливість почати гру на одному пристрої та продовжити її на іншому, зберігаючи ігровий прогрес, що значно підвищує зручність для користувачів. Це дозволяє забезпечити тісніший зв’язок між гравцем і грою, оскільки гравці мають можливість грати там, де їм зручно, без прив'язки до одного конкретного пристрою. Кросплатформенність, таким чином, стає важливим фактором конкурентоспроможності сучасних казуальних ігор, забезпечуючи їм перевагу у задоволенні потреб аудиторії.

Мобільні платформи стали основним середовищем для казуальних ігор завдяки широкій доступності смартфонів та їх інтеграції в повсякденне життя користувачів. Мобільні пристрої не лише стали основним засобом доступу до

казуальних ігор, але й змінили сам підхід до їх розробки, створюючи ідеальні умови для коротких ігрових сесій. Це дозволяє користувачам взаємодіяти з ігровим контентом у будь-який зручний час і місце, що робить такі ігри особливо привабливими. Мобільні ігри стали невід'ємною частиною повсякденного життя, що значно розширило охоплення казуальних ігор. Однак розробка для мобільних платформ вимагає врахування специфічних особливостей, таких як обмежені ресурси пам'яті, процесорної потужності та різні характеристики екранів, що ускладнює процес розробки. Для успішної реалізації ігор на мобільних пристроях важливо забезпечити оптимізацію графічних ресурсів, зменшити обсяг пам'яті, який займає гра, а також стабільну роботу на широкому спектрі пристроїв, включаючи менш потужні моделі. Мінімізація обсягу пам'яті, яку займає гра, а також забезпечення стабільної роботи є важливими завданнями, оскільки значна частина аудиторії використовує бюджетні пристрої з обмеженими ресурсами. Це вимагає ретельного підходу до дизайну та тестування, зокрема адаптації інтерфейсу та оптимізації обчислювальних процесів.

Серед основних викликів, з якими стикаються розробники казуальних ігор, слід також виділити необхідність створення унікального контенту, особливо ігрових рівнів, що забезпечують новизну і різноманітність. Гравці очікують постійного розвитку, цікавих викликів та нових рівнів, а повторюваність може швидко призвести до втрати інтересу. З метою забезпечення цікавості та утримання аудиторії, розробники дедалі більше використовують процедурну генерацію рівнів, яка дозволяє швидко створювати нові варіації контенту. Це особливо актуально для гіперказуальних і гібридних казуальних ігор, де процедурна генерація допомагає створювати нові сценарії з мінімальними затратами часу і ресурсів, що дозволяє утримувати гравців зацікавленими протягом тривалого часу. Процедурна генерація не лише сприяє різноманітності контенту, але й оптимізує витрати на його створення, що робить цей підхід критично важливим для розробників, які прагнуть забезпечити динамічний і захоплюючий ігровий процес. Процедурні методи дозволяють автоматично генерувати нові рівні на основі певних правил або алгоритмів, що надає

розробникам можливість швидко створювати новий контент без необхідності ручної роботи. Цей підхід є особливо цінним у контексті казуальних ігор, де оновлюваність і динамічність є ключовими аспектами успіху.

1.3 Огляд рушіїв для розробки казуальних ігор

У сучасному процесі розробки казуальних ігор розробники стикаються з викликом вибору оптимальних інструментів, які забезпечують ефективність, гнучкість, швидкість створення та можливість масштабування проекту. Вибір рушія є критично важливим рішенням, яке визначає успішність усього процесу розробки, оскільки впливає на технічні характеристики, продуктивність команди, вартість розробки та подальшу підтримку гри. У цьому підрозділі буде представлено детальний огляд сучасних рушіїв, які найбільш поширені серед розробників казуальних ігор: Unity, Godot, GameMaker Studio, Unreal Engine та Construct.

Unity залишається одним із найпопулярніших рушіїв для розробки казуальних ігор завдяки своїй гнучкості, потужному інструментарію та підтримці широкого спектру платформ. Цей рушій дозволяє створювати як прості 2D, так і складні 3D проекти, що робить його універсальним інструментом як для самостійних розробників, так і для великих команд. Unity забезпечує кросплатформенну розробку, що дозволяє випускати гри одночасно для мобільних пристроїв, ПК, консолей та веб-браузерів, значно розширюючи охоплення аудиторії та підвищуючи рентабельність (рис. 1.5).

Рисунок 1.5 — Приклад використання рушія Unity.

Однією з ключових переваг Unity є його розширена екосистема, яка включає Unity Asset Store — платформу, на якій розробники можуть купувати або продавати готові моделі, текстури, скрипти та інші ресурси. Це істотно скорочує час на розробку, дозволяючи команді зосередитися на інноваційних аспектах проєкту. Інтеграція з такими інструментами, як Tilemap, спрощує створення ігрових рівнів у 2D-форматі, що є вкрай важливим для казуальних ігор, де різноманітність рівнів і швидкість їх створення відіграють ключову роль.

Крім того, Unity підтримує велику кількість розширень та додатків, які можуть використовуватися для процедурної генерації контенту, інтеграції соціальних функцій, монетизації та тестування. Це дозволяє розробникам розширювати функціональні можливості своїх ігор без необхідності розробляти все з нуля. Unity також пропонує інтеграцію з основними рекламними мережами, що полегшує монетизацію казуальних проектів. Важливим аспектом є і кросплатформені можливості Unity, які дозволяють легко адаптувати гру для різних платформ і пристроїв, зокрема для мобільних телефонів, консолей, ПК та навіть VR-пристроїв. Завдяки цьому Unity стає універсальним рішенням для широкого кола проектів.

Втім, Unity має певні недоліки. Одним із них є високі системні вимоги, які можуть ускладнювати роботу на слабких пристроях, особливо при розробці складних проектів з великою кількістю активів. Крім того, повне опанування всіх можливостей рушія вимагає значного часу та зусиль, що може стати бар'єром для новачків. Однак завдяки активній спільноті, великій кількості навчальних матеріалів і ресурсів, розробники можуть відносно швидко подолати ці труднощі. Також Unity має велику кількість форумів, де можна отримати відповіді на запитання та знайти рішення для складних технічних проблем.

Godot — це відкритий рушій, що стрімко набирає популярності серед розробників казуальних ігор завдяки своїй простоті у використанні та відкритій природі. Godot підтримує як 2D, так і 3D розробку, проте його особливо цінують за можливості роботи з 2D-графікою. Цей рушій

оптимізований для розробки двовимірних ігор, що робить його привабливим для малих студій і незалежних розробників (рис. 1.6).

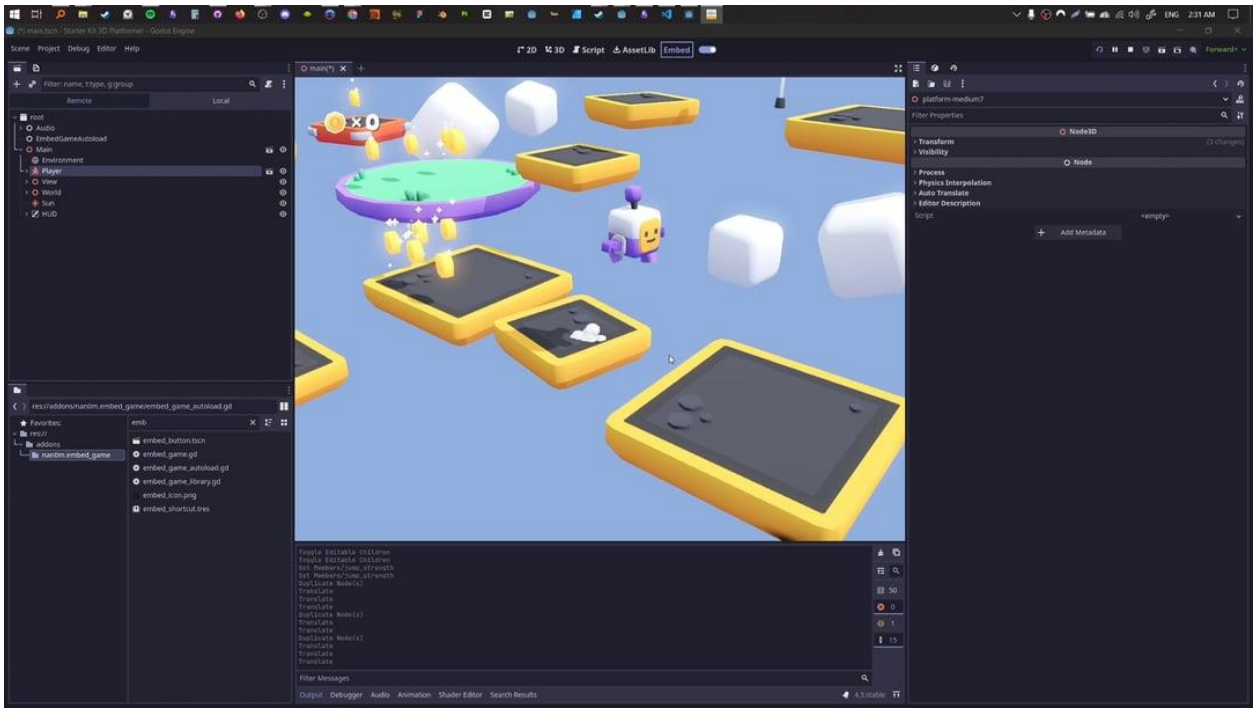

Рисунок 1.6 — Приклад використання рушія Godot.

Однією з найбільших переваг Godot є його відкритий код, який дозволяє розробникам змінювати рушій відповідно до своїх потреб. Це забезпечує більший контроль над процесом розробки та дозволяє реалізовувати індивідуальні рішення, що можуть бути критичними для специфічних проектів. Крім того, Godot не вимагає ліцензійних відрахувань, що робить його привабливим для проектів із обмеженим бюджетом. Godot також використовує власну скриптову мову GDScript, яка є схожою на Python і легкою для вивчення, що спрощує процес освоєння рушія для новачків.

Godot також підтримує вузлову систему, яка дозволяє структурувати проекти у вигляді деревоподібної ієрархії компонентів. Це робить процес розробки більш організованим і прозорим, дозволяючи швидко вносити зміни і масштабувати проект. Важливою особливістю є і можливість роботи над 2D-проектами з мінімальними витратами ресурсів, що робить Godot особливо ефективним для мобільних ігор та простих казуальних проектів.

Серед недоліків Godot варто зазначити відносно невелику кількість доступних плагінів і модулів у порівнянні з Unity, що може створити труднощі під час реалізації складних функціональних можливостей. Інфраструктура підтримки 3D-проектів також не така розвинена, як у конкурента, що обмежує використання Godot для масштабніших проектів. Крім того, деякі складніші функції, такі як реалізація розширеної фізики чи інтеграція з певними рекламними мережами, можуть вимагати додаткових зусиль і часу на реалізацію.

GameMaker Studio є одним із найбільш відомих рушіїв для створення 2D ігор. Він відомий своєю простотою та інтуїтивно зрозумілим інтерфейсом, що дозволяє швидко створювати прототипи ігор навіть розробникам без досвіду програмування. Мова програмування GML (GameMaker Language) є легкою у використанні, що робить GameMaker Studio популярним серед новачків та невеликих команд, які прагнуть швидко реалізувати свої ідеї (рис. 1.7).

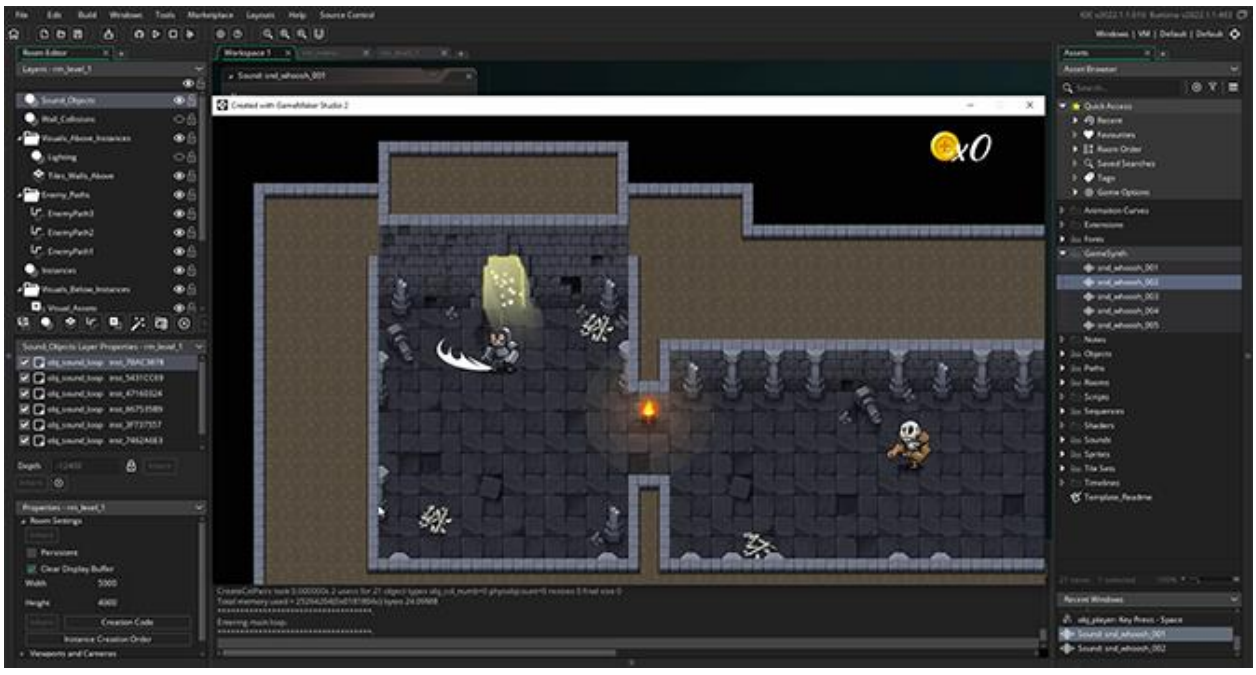

Рисунок 1.7 — Приклад використання рушія GameMaker Studio.

Основною перевагою GameMaker Studio є можливість швидкої розробки прототипів та простих ігор. Завдяки своїй візуальній системі створення ігрової логіки, розробники можуть легко зосередитися на творчих аспектах проекту,

а не на складних технічних деталях. GameMaker також пропонує різноманітні інструменти для роботи з анімацією, що дозволяє швидко створювати персонажів та інтерактивні елементи. Це робить GameMaker ідеальним вибором для створення невеликих казуальних проектів, які потребують швидкого впровадження ідей.

Проте GameMaker Studio має свої обмеження, зокрема в контексті масштабованості та реалізації складних механік. Цей рушій не підтримує багатьох можливостей, таких як процедурна генерація складних рівнів або розширена фізика, які можуть бути необхідні для більших або складніших проектів. Також підтримка кросплатформенності є менш гнучкою у порівнянні з Unity, що може стати проблемою для розробників, які хочуть розширити гру на кілька платформ. Крім того, ліцензійні витрати можуть швидко зростати, якщо команда планує підтримку декількох платформ, що робить цей рушій менш привабливим для великих проектів з обмеженим бюджетом.

Unreal Engine, попри свою відомість як рушія для AAA-проектів, також може використовуватися для розробки казуальних ігор, хоча це не є його основною перевагою. Найсильніша сторона Unreal Engine — це його потужні графічні можливості, які дозволяють створювати візуально вражаючі ігри. Також варто відзначити систему Blueprints, яка дає змогу створювати ігрову логіку без написання коду, що може бути корисним для прототипування казуальних проектів (рис. 1.8).

Рисунок 1.8 — Приклад використання рушія Unreal Engine.

Unreal Engine має багатий набір інструментів для роботи з тривимірною графікою, а його потужний рушій фізики дозволяє створювати дуже реалістичні ігрові світи. Це робить Unreal ідеальним для проектів, які потребують високоякісної візуальної складової. Крім того, Unreal Engine має розширену екосистему, включаючи Unreal Marketplace, де розробники можуть купувати або продавати активи, що значно спрощує процес розробки.

Однак цей рушій має високі вимоги до ресурсів, що ускладнює його використання для проектів, орієнтованих на менш потужні пристрої. Крім того, Unreal Engine є досить складним у вивченні, що робить його малопридатним для невеликих команд або інді-розробників, які мають обмежені ресурси. Використання Unreal Engine для казуальних ігор часто є надмірним, оскільки цей рушій розроблений насамперед для проектів із високими вимогами до графіки та продуктивності. Також вартість ліцензії може стати перешкодою для команд із обмеженим бюджетом.

Construct є спеціалізованим рушієм для розробки 2D ігор, який орієнтований на користувачів без досвіду програмування. Використовуючи візуальну систему створення ігрової логіки, Construct дозволяє швидко розробляти прототипи та невеликі проекти, що робить його популярним серед початківців та інді-розробників. Основна привабливість Construct полягає у його простоті та швидкості реалізації (рис. 1.9).

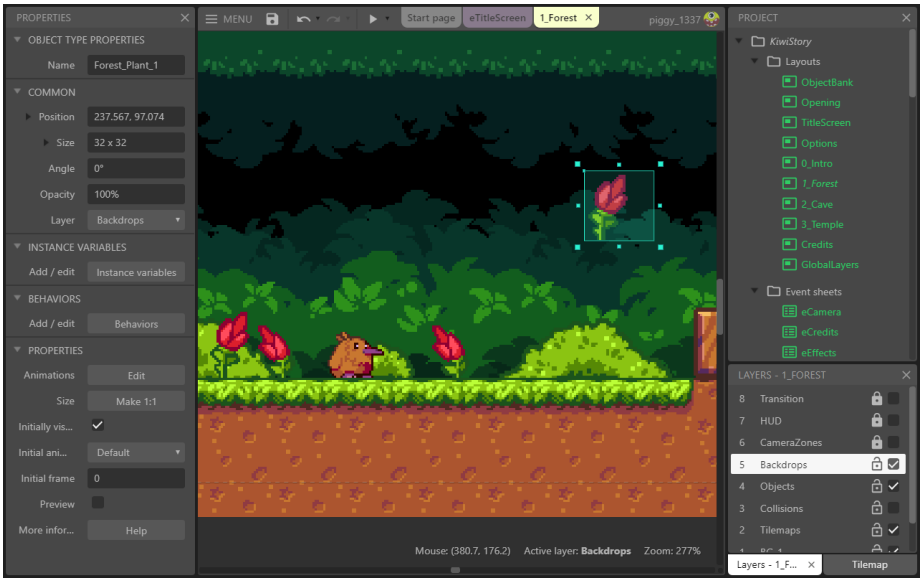

Рисунок 1.9 — Приклад використання рушія Construct.

Construct забезпечує легкість у використанні та можливість створення ігор без необхідності програмування, що дозволяє творцям зосередитися на креативних аспектах гри. Інтерфейс Construct є інтуїтивно зрозумілим, що дозволяє швидко освоїти основні принципи створення ігор. Construct також має підтримку HTML5, що дозволяє запускати ігри безпосередньо у веббраузерах, що робить його зручним для розробки простих ігор з широким охопленням.

Основними обмеженнями Construct є недостатня гнучкість для реалізації масштабних проектів та обмежені можливості щодо процедурної генерації контенту. Це робить Construct менш привабливим для тих, хто прагне створювати складні ігри або розширювати їх на різні платформи. У порівнянні з іншими рушіями, можливості кросплатформенності та автоматизації у Construct є обмеженими, що звужує його потенційне використання для амбітніших проектів. Крім того, Construct має обмеження в роботі з додатковими розширеннями та інтеграцією з зовнішніми сервісами, що може ускладнити процес розробки більш функціональних проектів.

Проведений огляд рушіїв для розробки казуальних ігор демонструє, що кожен із них має свої специфічні переваги та недоліки, які впливають на вибір розробників залежно від потреб проекту. Unity виявляється найбільш універсальним рішенням, завдяки своїй гнучкості, підтримці різних платформ та наявності великої спільноти, що забезпечує доступ до численних ресурсів і підтримки. Він є відмінним вибором для тих, хто прагне поєднати швидкість розробки та універсальність з широкими можливостями кросплатформенності та підтримкою інноваційних функцій. Unity також дозволяє швидко впроваджувати рекламні рішення та соціальні механіки, що є ключовими для казуальних ігор.

Godot приваблює відкритим кодом та зручністю використання, хоча й поступається Unity у плані масштабованості та наявності готових ресурсів. Він є чудовим вибором для тих, хто прагне повного контролю над кодом і віддає перевагу роботі з відкритими технологіями. GameMaker Studio ідеально підходить для простих 2D ігор, однак має значні обмеження для складніших проектів, особливо з точки зору кросплатформенності та процедурної

генерації контенту. Unreal Engine є надзвичайно потужним інструментом, але його використання для казуальних ігор є не завжди доцільним через високі системні вимоги та складність у освоєнні. Construct є чудовим вибором для новачків, проте не забезпечує достатньої гнучкості для реалізації масштабних та складних проектів, що робить його більш придатним для невеликих і швидких проектів.

Таким чином, вибір рушія залежить від специфіки проекту, бюджету, наявного досвіду команди та цілей розробки. Unity, завдяки своїй гнучкості та підтримці кросплатформенності, залишається найпопулярнішим рушієм для розробки казуальних ігор, особливо тих, які потребують швидкого впровадження інновацій та роботи на різних платформах. Розробники мають враховувати всі особливості кожного рушія для забезпечення ефективності, оптимізації ресурсів і досягнення поставлених цілей у рамках своїх проектів.

1.4 Огляд інструментів для створення рівнів

Процес створення ігрових рівнів є фундаментальним аспектом розробки казуальних ігор, який значною мірою визначає не лише захопливість і унікальність ігрового досвіду, але й комерційний успіх гри. В умовах зростаючої конкуренції в індустрії мобільних та казуальних ігор розробники змушені використовувати передові інструменти, які забезпечують ефективність, гнучкість та високу швидкість розробки. На сучасному етапі існує великий вибір інструментів для створення рівнів, серед яких є як вбудовані рішення в межах рушіїв, так і сторонні розробки, що дозволяють розширити можливості автоматизації і кастомізації процесу.

Одним із найпоширеніших інструментів для створення 2D-рівнів у Unity є Tilemap Editor (рис. 1.10). Цей інструмент забезпечує розробникам можливість ефективно використовувати плитки (tile) для створення складних і детальних рівнів. Його інтеграція з екосистемою Unity дозволяє значно скоротити час на розробку та забезпечує високу ефективність роботи з проєктами різного рівня складності. Tilemap Editor підтримує багатошаровість, що дозволяє додати глибину до рівнів, створюючи

багатогранний ігровий процес. Однак, варто зазначити, що створення великої кількості рівнів вручну вимагає значного часу, що може стати недоцільним для великих проєктів з великою кількістю контенту, особливо коли потрібна процедурна генерація.

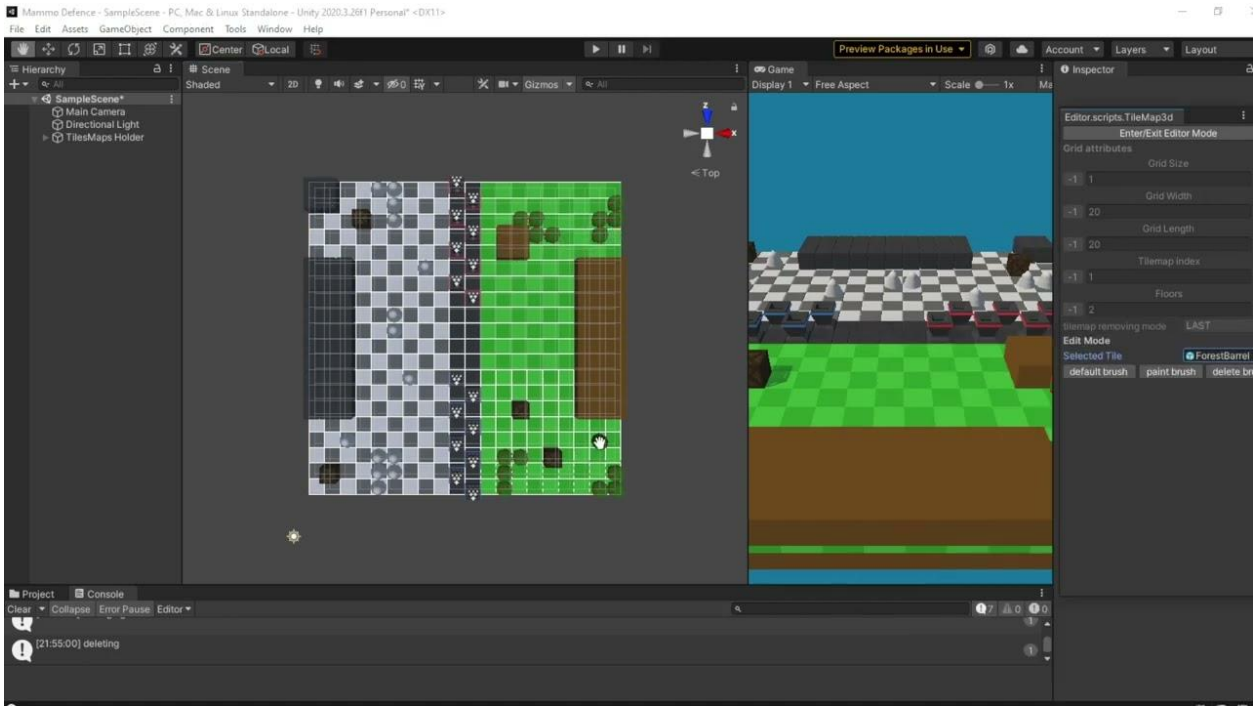

Рисунок 1.10 — Приклад використання Tilemap Editor.

Схожий підхід до створення рівнів реалізований у Godot через інструмент GridMap (рис. 1.11). Це потужний інструмент для створення рівнів, який використовує вокселі як основні будівельні блоки. В основі GridMap лежить система сітки, де кожна клітинка може бути заповнена певним елементом з бібліотеки MeshLibrary. Це дозволяє швидко створювати рівні шляхом розміщення об'єктів на визначених координатах.

Однією з основних переваг GridMap є можливість зберігати і повторно використовувати створені конфігурації, що значно спрощує роботу над великими проєктами. Інструмент також підтримує взаємодію з фізичними об'єктами та матеріалами, що робить його універсальним для різних типів ігор, таких як платформери, головоломки чи шутери.

Гнучкість GridMap дозволяє легко змінювати рівень за допомогою коду, а інтеграція з іншими інструментами Godot, такими як навігаційні сітки чи

світлові карти, забезпечує ще більшу функціональність. Однак для ефективного використання GridMap потрібно мати базові знання про структуру MeshLibrary і підготовку ресурсів для оптимізації продуктивності.

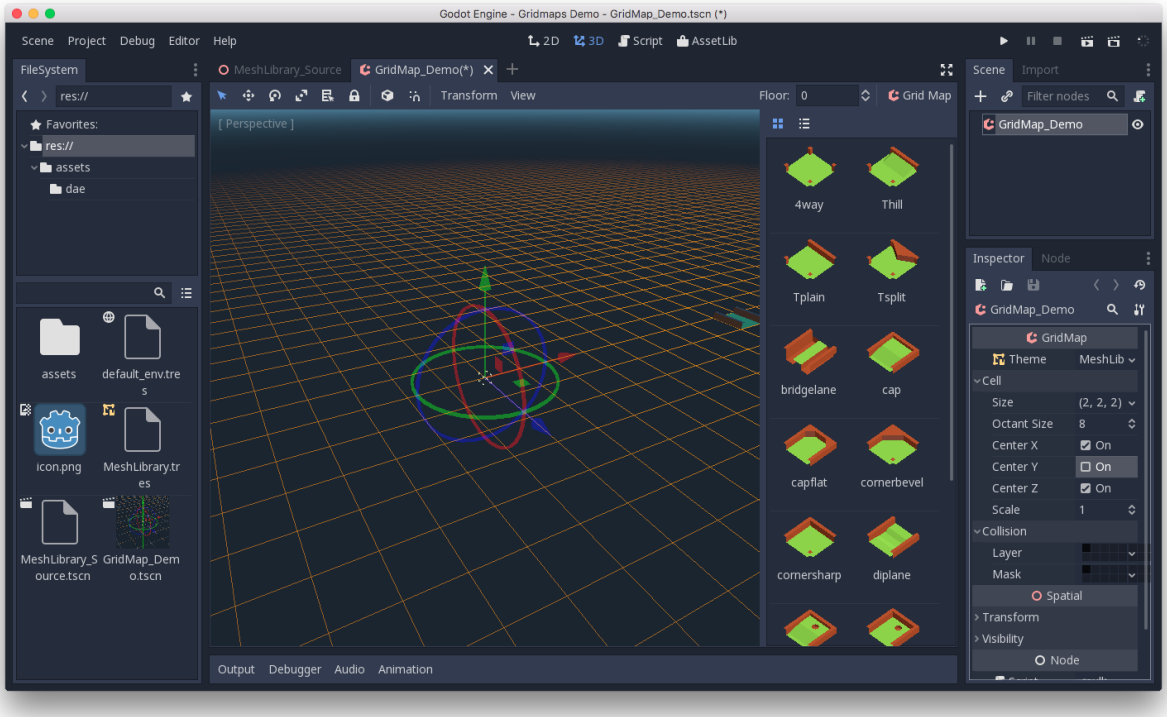

Рисунок 1.11 — Приклад використання GridMap.

Досить потужним інструментом є Paper2D в Unreal Engine. Він дозволяє створювати 2D-ігри, використовуючи потужності тривимірного рушія. Інструмент побудований на концепції інтеграції 2D-графіки в 3D-середовище, що відкриває можливості для розробки як традиційних 2D-ігор, так і 2.5D-проєктів. Використання Paper2D забезпечує доступ до фізики, освітлення, та інших систем рушія, які зазвичай недоступні в стандартних 2D-редакторах.

Центральним елементом Paper2D є система Sprite, яка дозволяє використовувати текстури як базові графічні елементи. Кожен спрайт може мати власні налаштування, такі як точки прив’язки, колізійні області та матеріали. Цей підхід дозволяє застосовувати освітлення та створювати візуальні ефекти, зберігаючи водночас характерний для 2D стиль. Редактор спрайтів забезпечує простоту у створенні і налаштуванні окремих елементів, що полегшує роботу над дизайном (рис. 1.12).

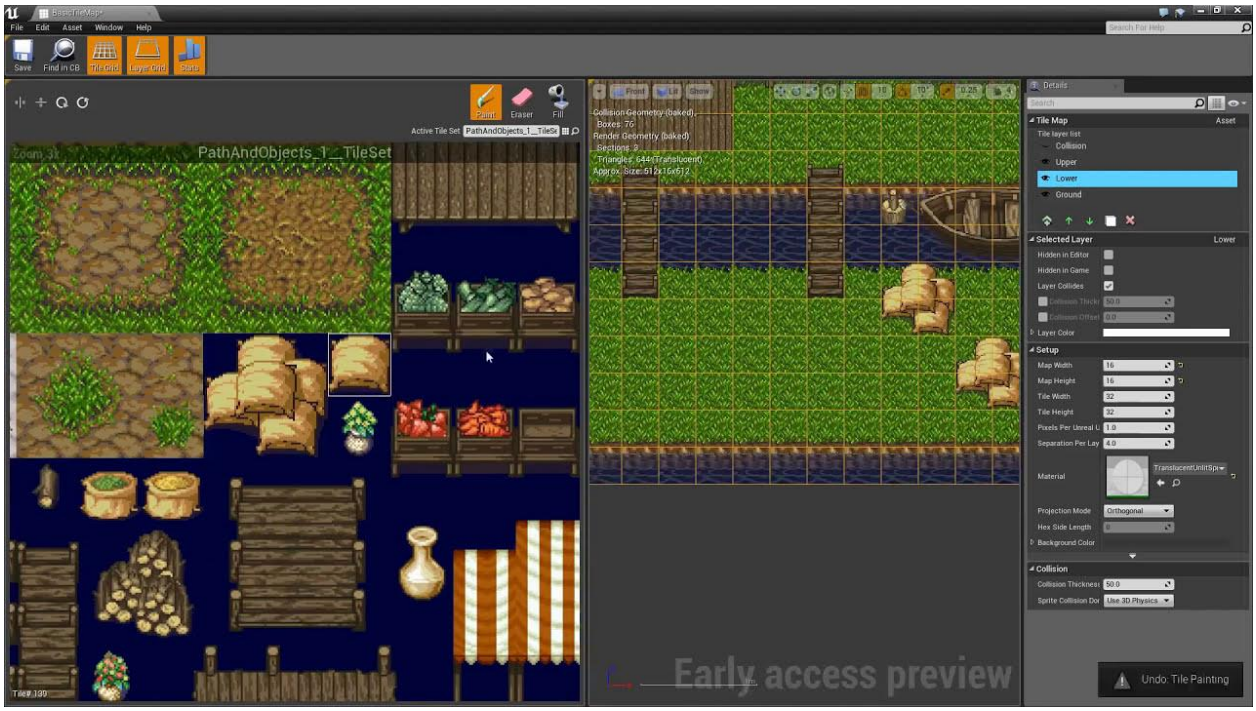

Рисунок 1.12 — Приклад використання Paper2D

Окремою важливою складовою є TileMap Editor, який підтримує створення рівнів шляхом організації графічних елементів у вигляді сітки. Користувач може задавати колізії, шари, та інші властивості для тайлів, що дозволяє створювати складні та детальні 2D-рівні. Інтеграція з фізичним рушієм Unreal Engine забезпечує точну симуляцію взаємодії об’єктів, що є важливим для ігор, які залежать від фізичних механік.

Інструмент Flipbook Editor розширює можливості Paper2D, забезпечуючи просте створення покадрових анімацій. Це дозволяє створювати динамічних персонажів, ефекти вибухів, та інші візуальні елементи. Гнучкі налаштування Flipbook, такі як зміна швидкості анімації або додавання кастомних подій, роблять цей інструмент універсальним для різних жанрів ігор.

Paper2D також підтримує повну інтеграцію з системою Blueprint, яка дозволяє створювати поведінку об’єктів без написання коду. Це забезпечує доступ до складних механік Unreal Engine, таких як інтерактивні об’єкти, звукові ефекти та UI-компоненти. Завдяки цьому інструмент є потужним рішенням для невеликих команд і окремих розробників.

Однак Paper2D має певні обмеження, зокрема менш активну підтримку з боку спільноти та обмежений розвиток порівняно з тривимірними інструментами Unreal Engine. Це може ускладнити використання інструмента для великих проєктів або для ігор, що потребують нестандартних рішень. Водночас його інтеграція з іншими системами рушія робить Paper2D надійним вибором для розробки проєктів середньої складності.

Крім вбудованих інструментів, існує низка сторонніх рішень, які надають додаткові можливості для автоматизації та підвищення варіативності рівнів. Одним з найбільш відомих інструментів є Tiled (рис. 1.13), незалежний редактор для створення рівнів на основі плиток. Його головною перевагою є гнучкість і можливість інтеграції з різними рушіями, включаючи Unity та Godot. Це дозволяє створювати багатошарові рівні, які можна експортувати у зручні формати для подальшої інтеграції. Водночас інтеграція з рушіями часто вимагає додаткових технічних знань, що може бути перешкодою для початківців.

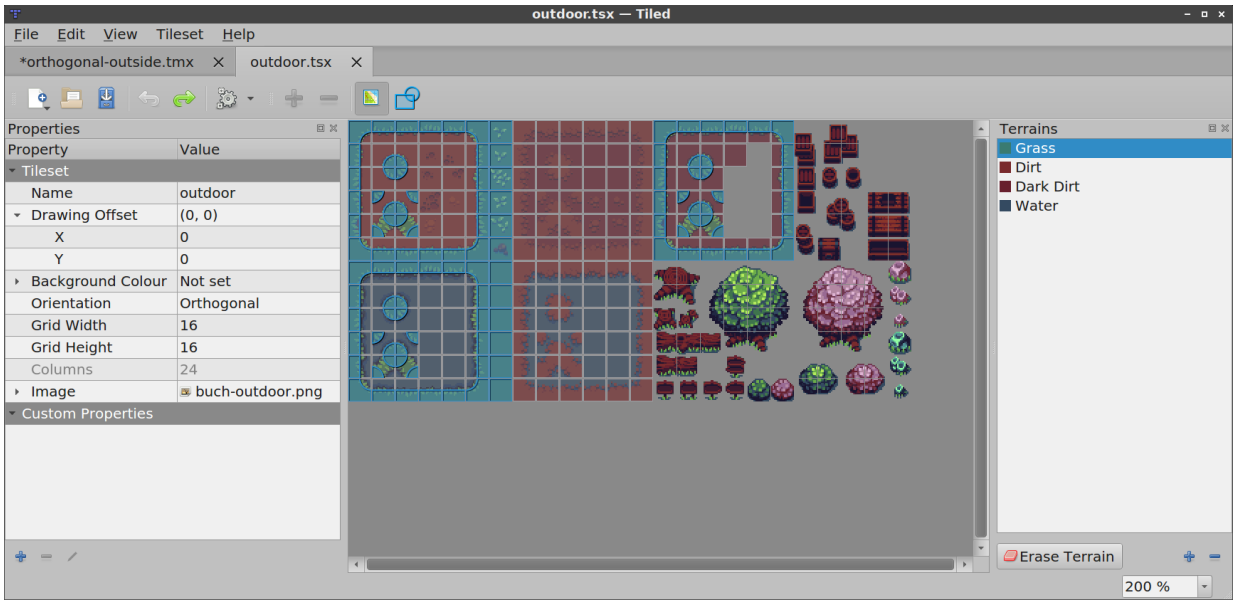

Рисунок 1.13 — Приклад використання Tiled.

Інструмент AutoTileGen (рис. 1.14) є прикладом рішення, яке дозволяє автоматизувати процес створення рівнів за допомогою алгоритмів процедурної генерації. Це особливо корисно для розробників, які прагнуть скоротити час на створення контенту та підвищити його варіативність.

Інтеграція з Unity та іншими рушіями робить AutoTileGen універсальним рішенням, проте використання цього інструменту вимагає глибокого розуміння принципів процедурної генерації, що може стати бар’єром для ефективного застосування. Помилки в налаштуванні шаблонів можуть призводити до зниження якості рівнів, що безпосередньо впливає на ігровий досвід.

Рисунок 1.14 — Приклад використання AutoTileGen.

Ще одним популярним інструментом для автоматизації створення рівнів є DunGen, який є спеціалізованим рішенням для процедурної генерації рівнів у Unity. DunGen (рис. 1.15) особливо популярний серед розробників жанру рогалик, оскільки дозволяє створювати складні підземелля та лабіринти з високим рівнем варіативності. Головною перевагою цього інструменту є здатність створювати логічно пов'язані та структуровані рівні, що забезпечують унікальний ігровий досвід під час кожного проходження. Проте, налаштування сегментів вимагає значних зусиль і часу, що може бути складним для команд, які прагнуть швидко створити прототип гри.

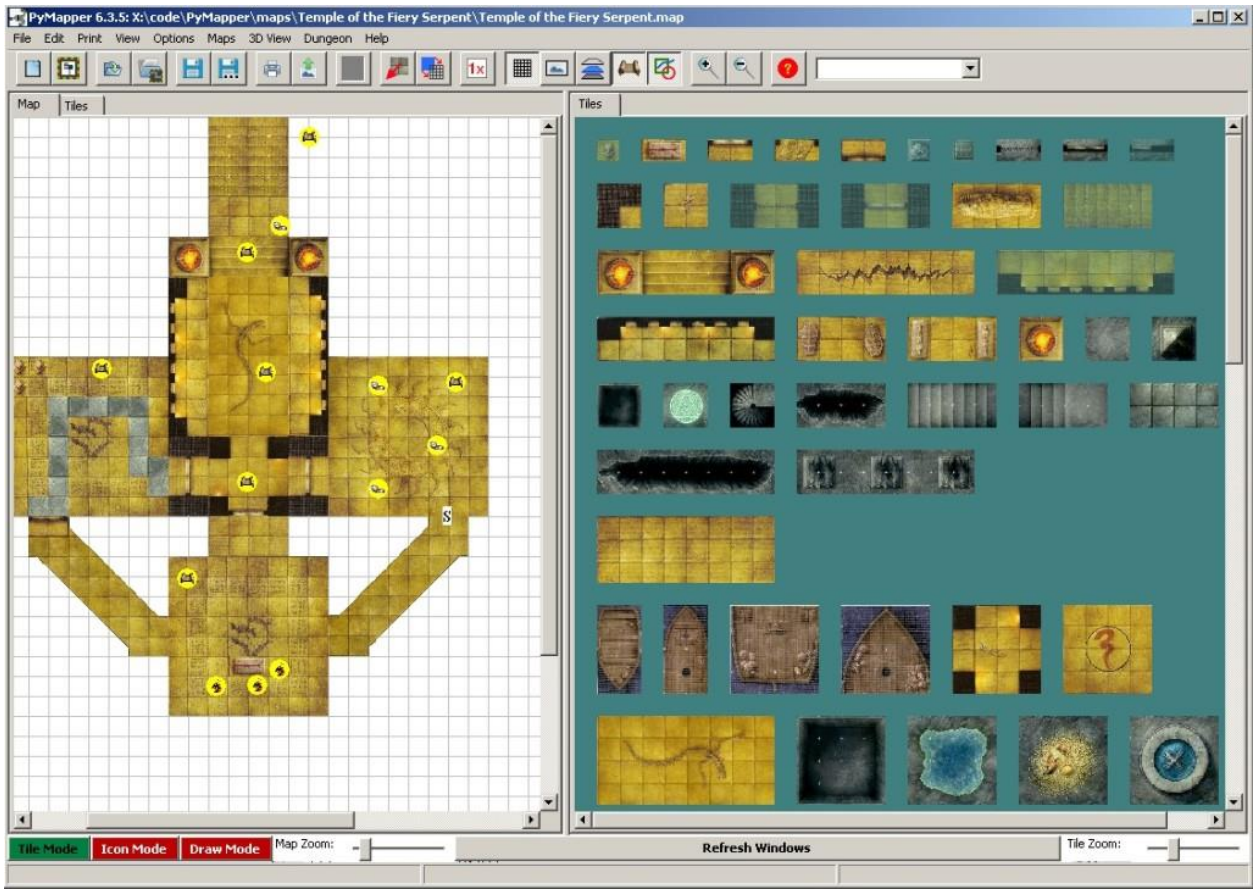

Рисунок 1.15 — Приклад використання DunGen.

Отже, аналіз існуючих інструментів для створення ігрових рівнів демонструє, що вибір конкретного рішення залежить від специфіки проєкту, масштабу, наявних ресурсів та рівня технічної підготовки команди. Вбудовані інструменти, такі як Unity Tilemap Editor, Godot TileMap або Unreal Engine Paper2D, підходять для швидкого початку роботи та забезпечують інтеграцію в межах рушія, але вони вимагають значного часу на створення великої кількості рівнів вручну. Сторонні інструменти, такі як Tiled, AutoTileGen і DunGen, надають більше можливостей для автоматизації та індивідуального налаштування, однак їх інтеграція може вимагати значних технічних знань і ресурсів.

1.5 Постанова задачі

З метою спрощення масштабування ігрових застосунків і підвищення ефективності розробки, постає необхідність розробки інструменту, який

поєднує шаблонний та процедурний підходи для автоматизованої генерації рівнів. Цей інструмент має забезпечити можливість розробникам створювати шаблони рівнів, що потім використовуються для динамічної генерації ігрових просторів, зменшуючи час розробки, підвищуючи варіативність контенту та знижуючи вимоги до системних ресурсів.

Постановку задачі можна сформулювати як створення інструменту для побудови, зберігання та автоматичної генерації ігрових рівнів, що дозволяє розробникам зменшити час на створення контенту та підвищити ефективність використання пам'яті.

2 ПРОЕКТУВАННЯ ІНСТРУМЕНТУ ДЛЯ ПОБУДОВИ ІГРОВИХ РІВНІВ У ІГРОВИХ ЗАСТОСУНКАХ

2.1 Структура та архітектура інструменту для побудови рівнів

Завданням цієї роботи є створення інструменту для побудови, зберігання та автоматичної генерації ігрових рівнів. Перш за все потрібно чітко визначити архітектуру інструменту, щоб точно розуміти складові модулі, необхідні для його роботи.

Проаналізувавши необхідні складові для роботи інструменту, було визначено його архітектуру на рис. 2.1.

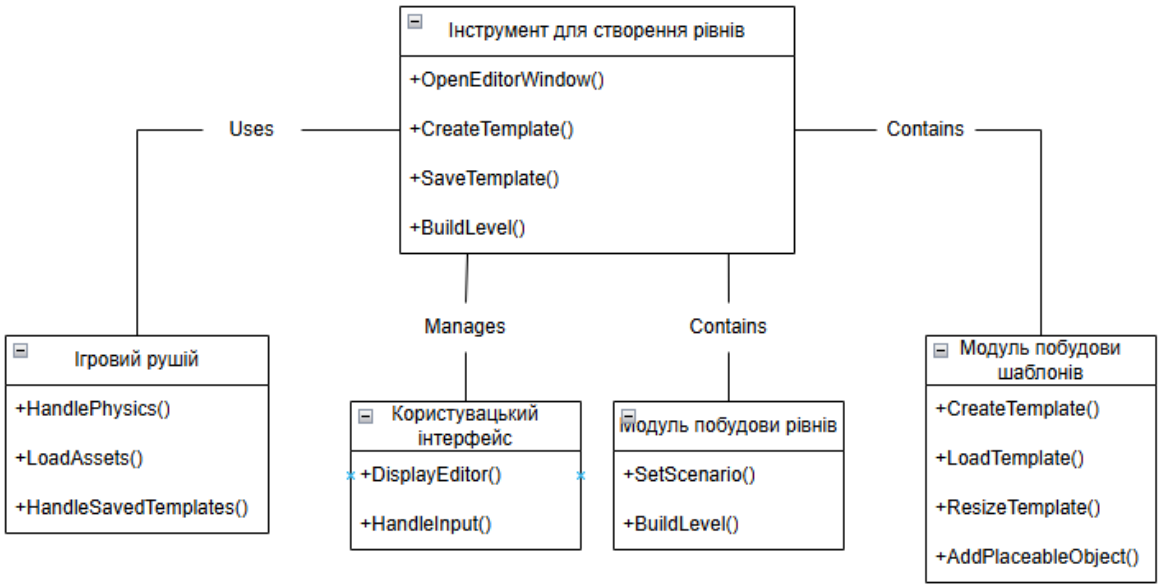

Рисунок 2.1 — Архітектура інструменту для створення рівнів.

Архітектура включає наступні частини: користувацький інтерфейс, модуль створення шаблонів модуль побудови рівнів, ігровий рушій. Кожний модуль виконує свою функцію:

— інтерфейс користувача необхідний для взаємодії користувача із інструментом, він дозволяє задавати початкові параметри рівня, наприклад

розмір, крім цього інтерфейс надає інтуїтивно зрозумілі опції для побудови рівнів;

— модуль шаблонів рівнів опрацьовує зміни, вносимі користувачем через інтерфейс, він встановлює вказаний користувачем тип кожної комірки шаблону і зберігає його у необхідному форматі;

— база даних шаблонів представляє собою механізм збереження і доступу до даних використовуючи внутрішні можливості ігрових рушіїв;

— модуль побудови рівнів приймає собі як вхідні данні певний шаблон, і для кожної комірки шаблону створює ігровий об’єкт по її локальним координатам, таким чином відбувається створення рівня в ігровому середовищі;

— ігровий рушій слугує як ядро інструменту на основі його будується функціонал інструменту, крім того, для збереження шаблонів будуть використовуватися внутрішні інструменти рушія.

Після визначення архітектури видно, що структура інструменту для побудови ігрових рівнів складається з кількох ключових компонентів, кожен з яких виконує свою унікальну функцію у забезпеченні ефективного створення та управління рівнями. Кожен компонент має своє призначення, яке взаємодоповнює інші елементи системи, забезпечуючи безперебійну і гнучку роботу інструменту в цілому.

Першим елементом структури є інтерфейс користувача (UI), який надає розробнику всі необхідні інструменти для початку роботи. Завдяки цьому інтерфейсу користувач може ініціювати процес створення нових шаблонів рівнів або редагування вже існуючих. Він є початковою точкою взаємодії, де користувач визначає основні параметри майбутнього рівня, такі як його розмір, розташування об’єктів, типи елементів тощо. Інтерфейс є простим у використанні і дозволяє задавати початкові параметри рівня інтуїтивно зрозумілими опціями. Крім того, двонаправлений зв'язок з іншими модулями забезпечує ефективний процес редагування та внесення змін до вже існуючих шаблонів.

Далі у структурі інструменту знаходиться модуль створення шаблонів рівнів. Основне завдання цього модуля полягає в обробці введених даних, отриманих через інтерфейс користувача. Він забезпечує можливість задавати тип для кожної комірки рівня, визначаючи, що саме буде розташовано на кожній частині ігрового поля. Модуль створює повноцінні шаблони, які згодом можуть бути використані як основа для генерування рівнів. Важливою особливістю цього модуля є можливість структурувати інформацію в необхідному форматі для подальшої роботи з базою даних шаблонів.

База даних шаблонів представляє собою центральний компонент, що забезпечує організоване зберігання всіх створених шаблонів ігрових рівнів. Вона дозволяє систематизувати роботу з шаблонами, спрощуючи їх пошук, збереження та обробку. Завдяки використанню бази даних розробники мають можливість швидко отримати доступ до необхідного шаблону, відновлювати його після внесення змін або використовувати повторно у нових проєктах.

Наступним компонентом є модуль побудови рівнів. Він обробляє шаблони, що зберігаються у базі даних, та на їх основі генерує повноцінні ігрові рівні. Як вхідні дані модуль приймає готові шаблони, які містять параметри для кожної комірки, зокрема тип об'єкта та його розташування. На основі цих даних модуль автоматично будує рівень, створюючи необхідні ігрові об'єкти у віртуальному середовищі.

Ігровий рушій є фінальним компонентом цієї структури. Саме рушій виконує всю обчислювальну роботу з реалізації згенерованих рівнів в ігровому середовищі. На основі шаблонів, що пройшли всі етапи підготовки, ігровий рушій, наприклад Unity, відповідає за створення рівнів у вигляді інтерактивного ігрового контенту. Він забезпечує можливість тестування, налагодження, верифікації і оптимізації рівнів, створених за допомогою інструменту.

Структуру інструменту наведено на рис. 2.2.

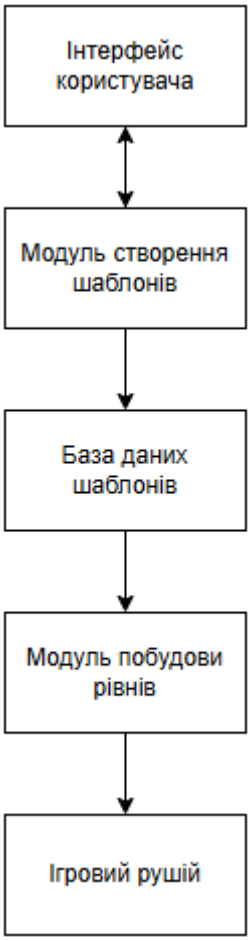

Рисунок 2.2 — Структура інструменту для створення рівнів.

Отже, структуру інструменту для побудови ігрових рівнів сформовано, потрібно обрати алгоритми для його реалізації.

2.2 Аналіз алгоритмів для генерації рівнів

Процес генерації рівнів є фундаментальним аспектом для досягнення цікавого, динамічного і різноманітного ігрового досвіду в казуальних іграх. Це завдання є водночас і творчим, і технічним викликом, адже вимагає ефективних підходів, що дозволяють автоматизувати створення складного і багатошарового контенту, зменшуючи потребу в ручній праці та оптимізуючи ресурси розробників. У цьому розділі розглядаються основні алгоритми, що використовуються в процесі автоматизованої генерації рівнів, а також їхні сильні та слабкі сторони.

Процедурна генерація (Procedural Generation) є універсальним підходом, що забезпечує створення безлічі різних варіантів рівнів з використанням обмеженого набору параметрів. Її головна перевага полягає у здатності створювати контент динамічно, знижуючи вимоги до пам'яті за рахунок зменшення розміру збережених даних. Це дозволяє, наприклад, таким іграм, як Spelunky або No Man's Sky, створювати неповторні світи, що забезпечують кожного гравця унікальним досвідом.

Проте успішна процедурна генерація вимагає великої кількості тестування та ретельного налаштування, аби забезпечити необхідний баланс між складністю ігрового процесу та цікавістю контенту. Неправильне налаштування алгоритму може призвести до одноманітності рівнів або, навпаки, до занадто хаотичної структури. Таким чином, ефективне використання процедурної генерації вимагає значних ресурсів на етапі розробки та тестування, що може стати викликом для невеликих команд.

Алгоритми на основі графів (Graph-based Algorithms) формують рівень шляхом зв'язку між вузлами, забезпечуючи чітку і логічну структуру. Вони ідеально підходять для створення лабіринтів або мережі взаємопов'язаних кімнат. Найвідоміший приклад такого підходу — алгоритм A* (A-Star), який широко використовується для пошуку оптимальних шляхів між точками у грі, надаючи гравцеві чітко визначені цілі та маршрути для їх досягнення.

Однією з переваг алгоритмів на основі графів є здатність створювати взаємопов'язані простори з кількома шляхами до мети, що забезпечує нелінійний ігровий процес. Однак надмірна структурованість таких рівнів може знижувати інтерес до повторного проходження, оскільки певні аспекти стають передбачуваними. До того ж, складність налаштування графових алгоритмів може значно зрости, якщо необхідно врахувати багато можливих варіантів зв'язків між вузлами, що підвищує складність і час розробки.

Клітинні автомати (Cellular Automata) використовуються для моделювання органічних структур, таких як печери або підземелля, шляхом застосування набору правил, які визначають зміну стану кожної комірки

залежно від стану її сусідів. Це дозволяє створювати нерегулярні, але природні середовища, що забезпечують візуальну варіативність і непередбачуваність. Зокрема, вони добре підходять для створення підземних лабіринтів, де кожна комірка відповідає за певну частину структури, як-от стіни або проходи.

Ключова перевага клітинних автоматів — це здатність створювати природні і нерегулярні рівні, які важко досягти іншими методами. Проте великим недоліком є відсутність гарантованої логіки або передбачуваності у згенерованих рівнях, що може бути проблематичним для жанрів, де важлива структурована навігація. Крім того, змінювання навіть одного параметра може призвести до суттєвих змін у структурі рівня, що потребує додаткового тестування і налаштування.

Методи розділення простору (Space Partitioning), такі як Binary Space Partitioning (BSP), використовуються для побудови логічної структури рівня шляхом поділу простору на менші області. Цей підхід часто використовується у шутерах, де важливо мати чітко організовану структуру з приміщеннями та коридорами, що направляють гравця до конкретних цілей. BSP забезпечує логічну і послідовну структуру, що допомагає гравцеві орієнтуватися в середовищі.

Перевага методу розділення простору полягає в тому, що він надає рівню зрозумілу і логічну структуру, забезпечуючи навігацію гравця і легкість розуміння. Проте такий підхід може призвести до надмірної одноманітності, якщо його не доповнювати іншими елементами, що забезпечують варіативність. Це особливо важливо у казуальних іграх, де гравець очікує на непередбачувані й цікаві ситуації.

Шаблонні алгоритми (Template-based Algorithms) полягають у використанні попередньо створених елементів (шаблонів), що комбінуються для створення рівнів. Цей підхід дозволяє зберегти високий рівень контролю над якістю ігрового контенту, адже всі елементи розробляються вручну і відповідають визначеним стандартам. Він добре підходить для створення

складних рівнів із високим рівнем деталізації, де важлива кожна деталь, як, наприклад, у платформерах або головоломках.

Головна перевага шаблонних алгоритмів полягає в тому, що вони дозволяють уникнути випадкових помилок, що можуть виникнути в процесі повної автоматизації. Кожен елемент створюється вручну, що дозволяє досягти високої якості та відповідності задуму дизайнера. Проте підготовка шаблонів вимагає значних витрат часу, а варіативність рівнів може бути обмеженою через використання обмеженого набору шаблонів. У результаті гра може втрачати свою новизну при повторних проходженнях.

Кожен з розглянутих алгоритмів має свої переваги та недоліки, і вибір підходу залежить від конкретних вимог до гри. Процедурна генерація забезпечує унікальність і варіативність, але вимагає значних ресурсів для налаштування і тестування. Алгоритми на основі графів створюють передбачувані та логічні рівні, але можуть втратити елемент випадковості. Клітинні автомати дозволяють генерувати органічні середовища, проте потребують ретельного налагодження. BSP ефективно працює для структурованих ігор, але може призводити до одноманітності. Шаблонні алгоритми гарантують якість, але обмежують варіативність.

Проаналізувавши алгоритми, було вирішено використовувати для роботи інструменту комбінацію процедурної та шаблонної генерації, щоб забезпечити баланс між гнучкістю, варіативністю і контролем над якістю створюваних рівнів. Такий підхід дозволяє максимально ефективно використовувати час і ресурси розробників, одночасно забезпечуючи унікальний і захоплюючий ігровий досвід для гравців.

2.3 Математичні моделі для автоматизованої генерації рівнів

Необхідно побудувати та дослідити математичні моделі для розуміння того, як можна вдосконалити процес створення рівнів і зменшити вимоги до

системних ресурсів. Розробка таких моделей дозволяє формалізувати процес генерації ігрових рівнів, оптимізувати використання пам'яті та часу розробки.

Для вирішення цих проблем пропонується розробити нову математичну модель, яка поєднує можливості традиційних редакторів карт із процедурною генерацією. Основний підхід до генерації рівнів базується на використанні статичних шаблонів та процедурної генерації для їх доповнення.

Нехай множина всіх можливих рівнів позначається як G, яка складається з n різних підрівнів або компонентів. Кожен рівень можна представити у вигляді:

$$G = \bigcup_{i=1}^{n}(S_i + P_i), \quad (2.1)$$

де S_i— статичний шаблон i-ої частини рівня;

P_i— процедурно згенеровані елементи, що додаються до кожного шаблону;

n—загальна кількість шаблонів, що використовуються для створення рівня.

Цей підхід дозволяє комбінувати статичні та динамічні компоненти, забезпечуючи гнучкість у процесі створення рівнів та уникаючи надмірної повторюваності. Алгоритм генерації забезпечує випадковість вибору певних елементів, що додає унікальності кожному рівню, зберігаючи при цьому контроль над базовою структурою.

Математична модель споживання пам'яті.

При порівнянні ефективності інструментів важливо враховувати кількість споживаної пам'яті. Нехай $M_{traditional}$ та $M_{proposed}$ позначають обсяги використаної пам'яті для традиційних інструментів та запропонованого підходу відповідно. У традиційних підходах обсяг пам'яті для збереження рівня визначається як сума пам'яті для всіх об'єктів на рівні:

$$M_{traditional} = \sum_{j=1}^{m} O_j, \quad (2.2)$$

де O_j— обсяг пам'яті, що використовується для j -го об'єкта;

m—кількість об'єктів у рівні.

Ця модель показує, що зі збільшенням кількості об'єктів лінійно зростає і обсяг використаної пам'яті, що може стати проблемою для складних рівнів.

У запропонованому підході обсяг пам'яті визначається як:

$$M_{proposed} = S_{base} + P_{gen}, \quad (2.3)$$

де S_{base}— обсяг пам'яті для збереження статичних шаблонів;

P_{gen}— обсяг пам'яті для процедурних алгоритмів та параметрів генерації.

Оскільки процедурно згенеровані елементи не потребують збереження всіх деталей наперед, загальний обсяг пам'яті суттєво зменшується.

Для кількісної оцінки зменшення обсягу пам'яті вводимо коефіцієнт K_M:

$$K_M = \frac{M_{traditional}}{M_{proposed}}, \quad (2.4)$$

Цей коефіцієнт відображає відношення між обсягами пам'яті двох підходів.

Час, необхідний для створення рівня, є важливим показником ефективності розробки. Для традиційного методу час створення рівня можна виразити як:

$$T_{traditional} = t_{design} + t_{integration}, \quad (2.5)$$

де t_{design}— час на розробку та дизайн рівня;

$t_{integration}$ — час на інтеграцію об'єктів у гру.

У традиційному підході ці часові витрати можуть бути значними, особливо для рівнів зі складною структурою та великою кількістю об'єктів.

Для запропонованого методу час створення рівня визначається як:

$$T_{proposed} = t_{template} + t_{procedural}, \quad (2.6)$$

де $t_{template}$ — час на створення та налаштування шаблонів;

$t_{procedural}$— час на розробку та налаштування алгоритмів процедурної генерації.

Оскільки процедурна генерація автоматизує значну частину процесу, загальний час створення рівня зменшується.

Коефіцієнт зменшення часу створення рівня вводиться як:

$$K_T = \frac{T_{traditional}}{T_{proposed}}, \quad (2.7)$$

Для більш глибокого розуміння залежності обсягу пам'яті та часу створення рівня від складності рівня, яку позначимо параметром c, можна записати наступні вирази.

Для традиційного методу:

$$M_{traditional}(c) = m(c) \cdot O_{avg}, \quad (2.8)$$

$$T_{traditional}(c) = t_{design}(c) + t_{integration}(c), \quad (2.9)$$

де $m(c)$ — кількість об'єктів, що залежить від складності рівня c;

O_{avg} — середній обсяг пам'яті одного об'єкта.

Зі збільшенням складності рівня обидва показники зростають, що може негативно впливати на продуктивність та час розробки.

Для запропонованого методу:

$$M_{proposed}(c) = M_{S_{base}} + M_{P_{gen}}(c), \quad (2.10)$$

$$T_{proposed}(c) = t_{template} + t_{procedural}(c), \quad (2.11)$$

Тут зростання обсягу пам'яті та часу створення рівня зі збільшенням складності є менш стрімким завдяки ефективності процедурної генерації.

Модель варіативності рівнів.

Кількість можливих унікальних рівнів N можна оцінити за формулою:

$$N = \prod_{i=1}^{n} V_i , \quad (2.12)$$

де V_i— кількість варіацій для i -го компоненту;

n — загальна кількість компонентів або шаблонів.

Ця модель показує експоненційне зростання кількості можливих рівнів зі збільшенням кількості компонентів та їх варіацій. Це забезпечує високу варіативність ігрового досвіду без необхідності ручного створення великої кількості рівнів.

Оптимізація використання ресурсів.

Для оцінки ефективності використання ресурсів вводиться функція ефективності E:

$$E = \frac{N}{M_{proposed}}, \quad (2.13)$$

Ця функція відображає кількість унікальних рівнів, які можна згенерувати на одиницю пам'яті. Максимізація E є бажаною, оскільки дозволяє отримати багатий ігровий контент при мінімальному використанні системних ресурсів.

Для оцінки різноманітності згенерованих рівнів можна застосувати поняття ентропії H:

$$H = -\sum_{k=1}^{K} p_k \log_2 p_k \,, \tag{2.14}$$

де p_k — ймовірність появи певної конфігурації k;

K — загальна кількість можливих конфігурацій.

Високе значення ентропії свідчить про високу різноманітність рівнів, що підвищує інтерес гравців та їхнє залучення.

Для контролю складності рівнів вводиться функція складності C:

$$C = \sum_{i=1}^{n} w_i \cdot f_i(P_i) \,, \tag{2.15}$$

де w_i — ваговий коефіцієнт для iii-го компоненту.

$f_i(P_i)$ — функція, що оцінює складність процедурно згенерованих елементів P_i;

Такий підхід дозволяє налаштовувати рівні відповідно до цільової аудиторії та забезпечувати баланс між викликом для гравця та задоволенням від гри.

Приклад застосування моделей.

Розглянемо гру, яка складається з трьох компонентів ($n = 3$):

- платформи (S_1) з $V_1 = 5$ варіаціями;
- перешкоди (S_2) з $V_2 = 4$ варіаціями;
- вороги (S_3) з $V_3 = 6$ варіаціями.

Загальна кількість можливих рівнів становитиме:

$$N = V_1 \cdot V_2 \cdot V_3 = 5 \cdot 4 \cdot 6 = 120\ , \quad (2.16)$$

Якщо обсяг пам'яті для запропонованого методу $M_{proposed}$ дорівнює 10 МБ, функція ефективності буде:

$$E = \frac{N}{M_{proposed}} = \frac{120}{10} = 12 \text{ рівнів на 1 МБ пам}'\text{яті}\ , \quad (2.17)$$

Це демонструє високу ефективність використання пам'яті при значній варіативності рівнів.

Побудовані математичні моделі показують, що поєднання статичних шаблонів з процедурною генерацією дозволяє суттєво оптимізувати процес створення ігрових рівнів. Запропонований підхід забезпечує ефективне використання пам'яті та часу розробки, підвищує варіативність та унікальність рівнів.

2.4 Принципи оптимізації пам'яті та ресурсів при побудові рівнів

Оптимізація використання пам'яті та обчислювальних ресурсів є критичним аспектом розробки ігор, особливо для проєктів з великим обсягом контенту. Ефективне використання пам'яті не лише забезпечує безперебійну роботу гри, але й суттєво впливає на розмір кінцевого продукту, що має особливе значення для мобільних платформ та пристроїв із обмеженими ресурсами. Один із ключових моментів у цьому контексті — це спосіб збереження та відтворення ігрових рівнів.

Традиційно в ігрових рушіях, таких як Unity, рівні зберігаються за допомогою префабів (prefabs) — попередньо налаштованих об'єктів, які включають компоненти та їхні властивості. Префаби дозволяють зберігати комплексні об'єкти з усіма їхніми залежностями, включаючи ієрархію дочірніх

об'єктів, компоненти, матеріали, скрипти та інші властивості. Цей підхід дозволяє ефективно повторно використовувати створені елементи, що пришвидшує розробку та значно полегшує керування контентом. На рівні внутрішньої реалізації, префаб є серіалізованою структурою даних, яка включає інформацію про всі компоненти об'єкта та їх стан, що робить його ідеальним для збереження статичних об'єктів, але не дуже ефективним для великих або динамічно змінюваних структур. Однак, використання префабів для збереження цілих рівнів або великих їхніх сегментів має низку недоліків.

По-перше, кожен префаб, що представляє рівень, містить повну інформацію про всі об'єкти, їхні позиції, властивості та взаємозв'язки. Це призводить до значного обсягу збережених даних, особливо якщо рівні містять велику кількість об'єктів. Як наслідок, зберігання великої кількості рівнів збільшує розмір кінцевого продукту та потребує більше пам'яті для обробки. Крім того, використання префабів не забезпечує достатньої гнучкості в динамічній генерації рівнів — кожен префаб є статичним набором об'єктів, і будь-які зміни в дизайні рівня вимагають ручного редагування очевидно, що зі збільшенням кількості рівнів, загальний обсяг пам'яті $M_{traditional}$ стає пропорційним кількості рівнів. Це створює значне навантаження на систему, особливо на мобільних платформах з обмеженими ресурсами, що вимагає оптимізації процесу збереження даних.

Очевидними недоліками цього підходу є:

— великий обсяг збережених даних: кожен рівень зберігається як окремий префаб із повною інформацією про всі об'єкти, що призводить до значного збільшення розміру гри та вимог до пам'яті;

— обмежена гнучкість у внесенні змін: будь-які зміни в структурі рівня вимагають редагування префаба, що стає вкрай незручним при великій кількості рівнів або складних ігрових світах;

— повторюваність контенту: при використанні статичних префабів важко досягти високої варіативності рівнів без значних витрат ресурсів на створення різноманітних версій.

Для вирішення цих проблем пропонується використовувати альтернативний підхід до збереження рівнів, що базується на використанні двовимірних масивів та ScriptableObject (SO) у Unity. У цьому підході кожен рівень представляється як двовимірний масив, де кожна комірка містить інформацію про об'єкт, який повинен бути створений під час виконання гри.

Двовимірний масив рівня може бути описаний як:

$$L = \begin{bmatrix} c_{1,1} & \cdots & c_{1,j} \\ \vdots & \ddots & \vdots \\ c_{i,1} & \cdots & c_{i,j} \end{bmatrix}, \quad (2.18)$$

де $c_{i,j}$ — елемент, що містить інформацію про тип об'єкта та його властивості на позиції (i , j);

ScriptableObject (SO) у Unity — це спеціальний тип об'єктів, який дозволяє зберігати дані незалежно від сцен та префабів. На рівні внутрішньої реалізації, SO є серіалізованими об'єктами, що зберігаються окремо від сцени і можуть бути зчитані або модифіковані в будь-який момент. Вони оптимізовані для зберігання даних, що використовуються повторно, наприклад конфігураційні налаштування або метадані рівнів. SO має легковагову структуру, що мінімізує обсяг необхідної пам'яті та підвищує ефективність обробки даних під час виконання. Крім того, використання SO дозволяє уникнути дублювання даних, оскільки вони існують як окремі ресурси, які можуть бути спільно використані декількома об'єктами в грі. Це дозволяє значно зменшити кількість копій і покращити управління ресурсами, зокрема при роботі зі складними структурами даних. SO є серіалізованими об'єктами, що можуть бути створені та збережені як окремі файли у проєкті. Цей підхід має низку переваг для збереження рівнів:

Легка вага — SO займають мінімум пам'яті, що дозволяє створювати їх у великій кількості без значного впливу на продуктивність. Наприклад, для типової гри використання SO може зменшити споживання пам'яті на 30-50%

порівняно з традиційними методами зберігання даних, такими як префаби, оскільки SO не зберігають дублювання структури та ієрархії об'єктів, а лише ключову інформацію, необхідну для процедурної генерації.

Розділення даних і логіки — SO дозволяють відокремити дані від логіки, що значно спрощує підтримку та розширення коду. Це дає змогу розробникам робити зміни в даних без необхідності модифікації логіки, а також додавати нові функціональні можливості, не змінюючи основну структуру даних. Такий підхід сприяє модульності системи, полегшує тестування і дозволяє зменшити кількість помилок при внесенні змін, оскільки кожен компонент є незалежним і чітко визначеним.

Підтримка серіалізації — дані в SO автоматично серіалізуються Unity за допомогою вбудованого механізму серіалізації. Цей процес дозволяє перетворювати об'єкти на послідовність байтів, що спрощує збереження та передачу даних. Порівняно з іншими підходами до збереження даних, серіалізація в Unity забезпечує ефективне збереження ресурсів і швидкий доступ до них, зменшуючи витрати на операції вводу-виводу та покращуючи загальну продуктивність.

Використання SO дозволяє суттєво зменшити обсяг збережених даних, оскільки зберігається лише необхідна інформація про об'єкти — тип, позиція тощо, а не повні префаби. Це також підвищує гнучкість і масштабованість, оскільки додавання нових рівнів зводиться до створення нових SO з відповідними масивами даних. Дані з SO можуть бути використані для процедурної генерації рівнів під час виконання, що підвищує варіативність гри.

ScriptableObject можна розглядати як реалізацію патерну "Легковаговик" (Flyweight), який спрямований на мінімізацію споживання пам'яті шляхом повторного використання спільних об'єктів. У контексті розробки рівнів, SO використовуються для збереження спільних даних, таких як характеристики об'єктів, що зустрічаються в різних частинах гри. Це дозволяє значно скоротити обсяг пам'яті, оскільки кожен екземпляр рівня

використовує посилання на вже існуючі об'єкти, а не створює нові. Наприклад, якщо один і той самий тип об'єкта повторюється в декількох рівнях, SO дозволяє зберегти лише одну копію даних про цей об'єкт, забезпечуючи доступ до них для всіх рівнів, що використовують його. Таким чином, дані ефективно розділяються між рівнями, зменшуючи загальний обсяг пам'яті, який необхідний для зберігання рівнів, і підвищуючи продуктивність за рахунок уникнення дублювання ресурсів. У даному інструменті, SO зберігають загальні дані про типи об'єктів, які можуть використовуватися у різних рівнях без необхідності створення окремих екземплярів для кожного випадку.

Використання двовимірних масивів та SO для збереження рівнів узгоджується з математичними моделями оптимізації, розглянутими у попередньому підрозділі. Згідно з формулою (10) видно що, $M_{S_{base}}$ є відносно невеликим і не залежить лінійно від кількості рівнів, загальний обсяг пам'яті зростає повільніше, ніж у традиційних методах.

Припустимо, що кожен рівень має розмір 50×50 клітинок. Для збереження інформації про кожну клітинку потрібен обсяг пам'яті m_c. Загальний обсяг пам'яті для одного рівня буде:

$$M_{level} = 50 \times 50 \times m_c = 2500 \times m_c\,, \tag{2.19}$$

Якщо m_c невеликий (наприклад, 4 байти), то обсяг пам'яті для одного рівня складає лише 10 КБ. Це набагато менше, ніж обсяг префаба, який може містити повну інформацію про всі об'єкти.

Отже, використання двовимірних масивів у зв’язці з ScriptableObject для збереження рівнів дозволяє суттєво зменшити обсяг використаної пам'яті та підвищити ефективність розробки. Цей підхід забезпечує гнучкість, масштабованість та підтримку процедурної генерації, що є важливим для створення сучасних казуальних ігор з великою кількістю контенту.

2.5 Розробка алгоритму роботи інструменту для створення ігрових рівнів

На основі всіх вищеописаних у цьому розділі етапів, було створено граф-схему алгоритму роботи інструменту для побудови ігрових рівнів на рисунку 2.3.

При запуску інструменту виконується ініціалізація всіх основних частин вікна, включаючи налаштування інтерфейсу та підготовку функціональних елементів до роботи. Після завершення ініціалізації користувач має можливість вибору між створенням нового шаблону або завантаженням існуючого. Якщо вибір не було зроблено, користувач повертається у головне меню. У разі створення нового шаблону відкривається вікно, яке дозволяє додати базові елементи, визначити їх розташування та налаштувати основні параметри. Якщо обрано завантаження існуючого шаблону, відкривається редактор, який дозволяє змінювати структуру та властивості рівня. Після завершення редагування чи створення шаблон зберігається у проекті як ScriptableObject (SO). Далі, під час побудови рівнів, модуль отримує чергу шаблонів, які необхідно послідовно опрацювати. Якщо у черзі є шаблон, рівень будується на його основі. Якщо ж черга порожня, інструмент завершує свою роботу.

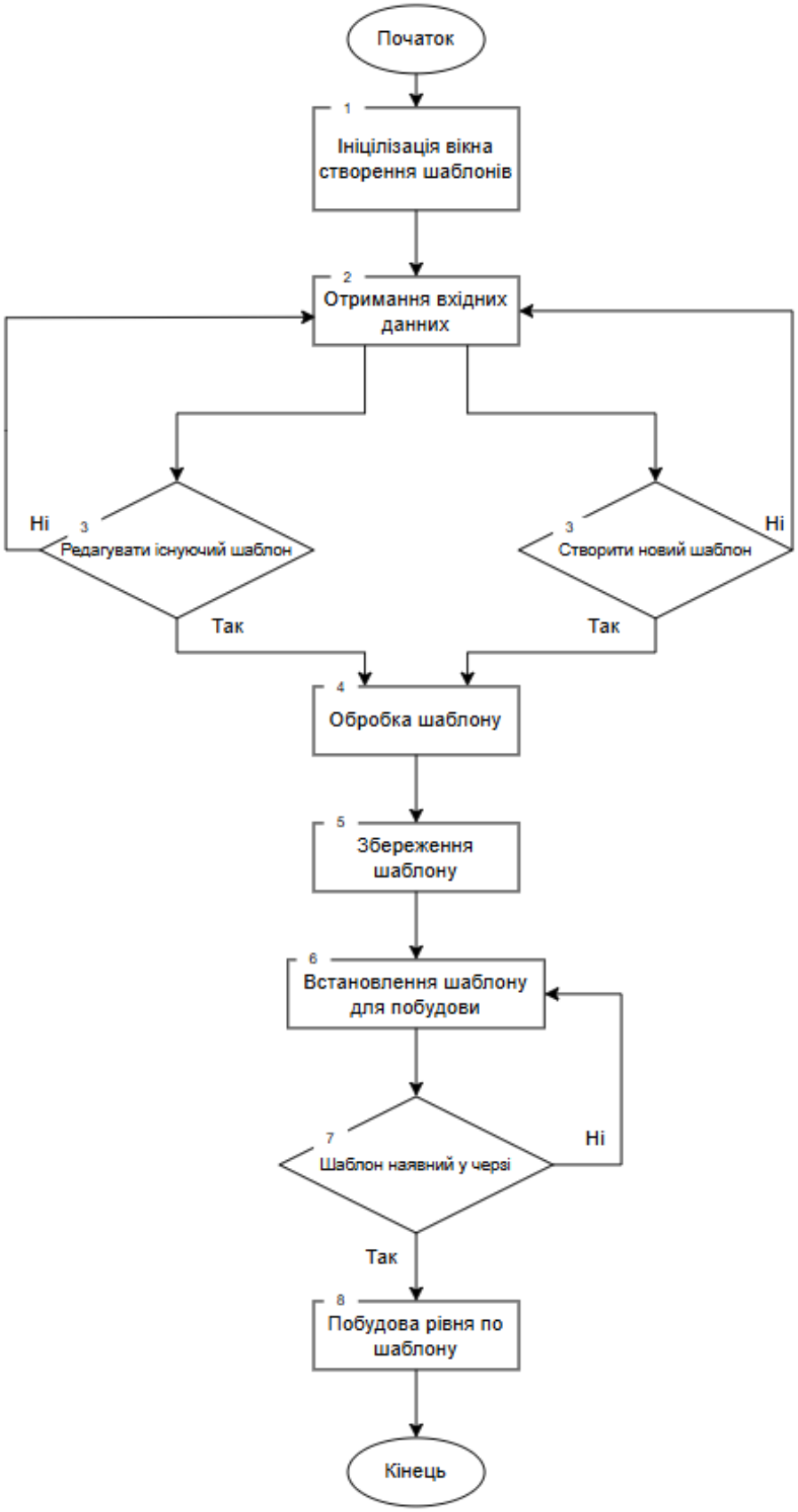

Рисунок 2.3 — Граф-схема алгоритму роботи інструменту для створення ігрових рівнів.

3 ПОГРАМНА РЕАЛІЗАЦІЯ ІНСТРУМЕНТУ СТВОРЕННЯ РІВНІВ У ІГРОВИХ ЗАСТОСУНКАХ

3.1 Обґрунтування вибору мови та середовища програмування

Вибір мови програмування залежить від багатьох факторів, таких як складність вивчення, розмір спільноти, наявність обширної документації та наявності звучного середовища розробки.

Для програмної реалізації інструменту для побудови рівнів було обрано мову програмування Python C#.

В якості середовища програмування для C# було обрано Visual Studio, що має ряд зручностей, таких як: підсвічування синтаксису, автодоповнення, менеджер пакетів, інтеграція з системами контроля версій і т. д. Важливим плюсом є те, що це середовище безкоштовне для використання.

3.2 Вибір фреймворків та бібліотек

Інструмент для створення рівнів буде створено як додаток до рушія Unity. Цей вибір обумовлено тим, що даний рушій має обширні можливості для імплементації сторонніх інструментів і додатків. Крім того, він найчастіше обирається для розробки казуальних ігор, так як має обширну документацію, велику спільноту та безкоштовний у використанні.

Також при розробці гри я буду використовувати наступні бібліотеки Unity:

— unityEngine є базовим для будь-якого проєкту на платформі Unity, він містить фундаментальні класи та структури, необхідні для роботи з ігровими об'єктами, компонентами, ресурсами та управління ними;

— unityEditor, надає API для розширення можливостей редактора Unity, він необхідний для створення власних інструментів, вікон, налаштуваннь та розширеннь, які інтегруються безпосередньо в редактор;

— unityEngine.Tilemaps містить класи та функціональність для роботи з тайлмапами (Tilemaps), які широко використовуються для створення

двовимірних ігор, тайлмапи дозволяють ефективно створювати та редагувати великі двовимірні рівні, використовуючи невелику кількість графічних елементів (тайлів);

— system.Collections.Generic, надає узагальнені (generic) колекції, які є ефективними та типізованими, вони використовуються для зберігання та управління наборами даних різних типів;

— system.Linq надає мову інтегрованих запитів (Language Integrated Query), яка дозволяє виконувати складні запити та операції над колекціями даних у зручний та виразний спосіб.

Використання зазначених інструментів та бібліотек є необхідним для реалізації функціональності інструменту побудови рівнів у Unity. Вони забезпечують необхідні засоби для створення інтерактивного та зручного інтерфейсу, ефективного управління даними та інтеграції з існуючими можливостями платформи Unity.

3.3 Програмна реалізація інструменту для створення рівнів

Першим кроком у роботі інструменту для створення рівнів є створення нового пустого шаблону.

```
private void CreateEmptyArray()
{
    if (_arrayWidth <= 0 || _arrayHeight <= 0)
    {
        Debug.LogError("Width and Height must be greater than 0.");
        return;
    }

    _levelArray = new TemplateElementType[_arrayWidth, _arrayHeight];
    _cellColors = new Color[_arrayWidth, _arrayHeight];
    _isEditableArray = new bool[_arrayWidth, _arrayHeight];
```

```
        for (int x = 0; x < _arrayWidth; x++)
        {
            for (int y = 0; y < _arrayHeight; y++)
            {
                _levelArray[x, y] = TemplateElementType.Ground;
                _cellColors[x, y] = Color.white;
                _isEditableArray[x, y] = true;
            }
        }

        Debug.Log($"Created empty array of size {_arrayWidth}x{_arrayHeight}.");
    }
```

Метод CreateEmptyArray() створює новий пустий масив для рівня, який представляє шаблон у вигляді двовимірної матриці. Він ініціалізує кожну клітинку значенням за замовчуванням — типом Ground, білим кольором, та дозволяє редагування кожної комірки. Це основний етап для створення нових шаблонів рівнів.

Далі потрібно створити у інтерфейсі поле для взаємодії з масиво, для можливості його редагування. Для цього створено метод DisplayArrayEditor.

```
    private void DisplayArrayEditor()
    {
        GUILayout.BeginVertical(GUI.skin.box);

        int cellSize = Mathf.FloorToInt(20 * _zoomScale);

        for (int y = _levelArray.GetLength(1) - 1; y >= 0; y--)
```

```
        {
            GUILayout.BeginHorizontal();

            for (int x = 0; x < _levelArray.GetLength(0); x++)
            {
                Rect rect = GUILayoutUtility.GetRect(cellSize, cellSize,
GUILayout.ExpandWidth(false), GUILayout.ExpandHeight(false));

                Color buttonColor = _cellColors[x, y];
                GUI.backgroundColor = buttonColor;
                if (GUI.Button(rect, GUIContent.none) || (_isMousePressed &&
rect.Contains(Event.current.mousePosition)))
                {
                    if (_selectedObjectIndex != -1)
                    {
                        TemplateElementType previousType = _levelArray[x, y];
                        _levelArray[x, y] =
_placeableObjects[_selectedObjectIndex].Type;
                        TemplateElementType newType = _levelArray[x, y];
                        Debug.Log($"Cell changed from {previousType} to
{newType}");
                        _cellColors[x, y] =
_placeableObjects[_selectedObjectIndex].Color;
                    }
                }
                GUI.backgroundColor = Color.white;
            }

            GUILayout.EndHorizontal();
        }
```

```
    GUILayout.EndVertical();
}
```

Даний метод відповідає за відображення редактора масиву, що представляє рівень. Він відображає комірки рівня в графічному інтерфейсі Unity, дозволяючи користувачам візуально редагувати кожну клітинку, обирати типи об'єктів і їх кольори, що робить процес створення рівнів більш зручним і наочним.

Для заповнення комірок масиву певними значеннями, потрібно написати функціонал для взаємодії з ними.

```
private void DisplayPlaceableObjects()
{
    GUILayout.BeginVertical(GUI.skin.box);

    float nameWidth = Screen.width * 0.25f;
    float typeWidth = Screen.width * 0.25f;
    float colorWidth = Screen.width * 0.2f;
    float buttonWidth = 50f;

    GUILayout.BeginHorizontal();
    GUILayout.Label("Name", GUILayout.Width(nameWidth));
    GUILayout.Label("Type", GUILayout.Width(typeWidth));
    GUILayout.Label("Color", GUILayout.Width(colorWidth));
    GUILayout.Label("", GUILayout.Width(buttonWidth));
    GUILayout.EndHorizontal();

    for (int i = 0; i < _placeableObjects.Count; i++)
    {
```

```
            var obj = _placeableObjects[i];

            GUIStyle style = new GUIStyle(GUI.skin.box);
            if (_selectedObjectIndex == i)
            {
               style.normal.background = MakeTex(1, 1, new Color(115 / 255f, 115
/ 255f, 115 / 255f));
               style.normal.textColor = Color.white;
            }

            EditorGUILayout.BeginHorizontal(style);

            GUILayout.Label(obj.Name, GUILayout.Width(nameWidth));
            GUILayout.Label(obj.Type.ToString(), GUILayout.Width(typeWidth));

            Rect                    colorRect                    =
EditorGUILayout.GetControlRect(GUILayout.Width(colorWidth));
            EditorGUI.DrawRect(new Rect(colorRect.x, colorRect.y, colorWidth,
colorRect.height), obj.Color);

            GUILayout.FlexibleSpace();

            if (GUILayout.Button("Delete", GUILayout.Width(buttonWidth)))
            {
               RemoveObject(i);
               EditorGUILayout.EndHorizontal();
               break;
            }

            EditorGUILayout.EndHorizontal();
```

```
            Rect rect = GUILayoutUtility.GetLastRect();
            if (Event.current.type == EventType.MouseDown &&
rect.Contains(Event.current.mousePosition))
            {
               _selectedObjectIndex = i;
               Repaint();
            }
         }

         if (_displayNewObjectFields)
         {
            EditorGUILayout.BeginHorizontal(GUI.skin.box);

            _newObjectName = EditorGUILayout.TextField(_newObjectName,
GUILayout.Width(nameWidth));
            _newObjectType =
(TemplateElementType)EditorGUILayout.EnumPopup(_newObjectType,
GUILayout.Width(typeWidth));
            _newObjectColor = EditorGUILayout.ColorField(_newObjectColor,
GUILayout.Width(colorWidth));

            GUILayout.FlexibleSpace();

            if (GUILayout.Button("Add", GUILayout.Width(buttonWidth)))
            {
               AddNewObject();
               _displayNewObjectFields = false;
            }
```

```
            if (GUILayout.Button("Cancel", GUILayout.Width(buttonWidth)))
            {
                _displayNewObjectFields = false;
            }

            EditorGUILayout.EndHorizontal();
        }
        else
        {
            GUILayout.BeginHorizontal();
            GUILayout.FlexibleSpace();
            if (GUILayout.Button("Add New Object"))
            {
                _displayNewObjectFields = true;
            }
            GUILayout.EndHorizontal();
        }

        GUILayout.EndVertical();
    }
```

Метод DisplayPlaceableObjects() використовується для відображення списку об'єктів, які можна розміщувати в шаблоні рівня. Він дозволяє переглядати наявні елементи, їх типи, кольори, а також видаляти об'єкти зі списку, що додає гнучкості у роботі зі складовими рівня.

Після внесення змін, потрібно зберегти або оновити збереження (у випадку коли шаблон не створювався а редагувався). Тому необхідно написати метод, який буде збирати усі введені данні, і створювати запис у SO.

```
        private void CreateOrUpdateRecordInSO(string name,
TemplateElementType[,] levelArray)
        {
            if (_roomTemplateSO == null)
            {
                Debug.LogError("RoomTemplateSO not found at path:
Assets/SO/RoomTemplateSO.asset");
                return;
            }

            if (_roomTemplateSO.Templates == null)
            {
                _roomTemplateSO.Templates = new
List<RoomTemplateSO.Template>();
            }

            var existingTemplate = _selectedTemplateIndex >= 0 &&
_selectedTemplateIndex < _templates.Count
                                ? _templates[_selectedTemplateIndex]
                                : null;

            if (existingTemplate != null && name == existingTemplate.name)
            {
                existingTemplate.TemplateElement =
(TemplateElementType[,])levelArray.Clone();
                existingTemplate.ScenarioType = _selectedScenarioType;
                existingTemplate.DateAdded =
System.DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
                AssignExits(existingTemplate, levelArray);
                Debug.Log($"Updated existing template: {name}");
```

```
        }
        else
        {
            var sameNameTemplate =
_roomTemplateSO.Templates.FirstOrDefault(t => t.name == name);
            if (sameNameTemplate != null)
            {
                Debug.LogError($"Template with name {name} already exists.
Choose a different name.");
                return;
            }

            int id = _roomTemplateSO.Templates.Count > 0 ?
_roomTemplateSO.Templates.Max(t => t.id) + 1 : 1;

            RoomTemplateSO.Template newTemplate = new
RoomTemplateSO.Template
            {
                name = name,
                id = id,
                TemplateElement = (TemplateElementType[,])levelArray.Clone(),
                ScenarioType = _selectedScenarioType,
                DateAdded = System.DateTime.Now.ToString("yyyy-MM-dd
HH:mm:ss")
            };

            AssignExits(newTemplate, levelArray);
            _roomTemplateSO.Templates.Add(newTemplate);
            Debug.Log($"Added new template: {name}");
```

```
            if (existingTemplate != null)
            {
                existingTemplate.TemplateElement = _originalLevelArray;
                Debug.Log($"Restored original template:
{existingTemplate.name}");
            }
        }

        EditorUtility.SetDirty(_roomTemplateSO);
        AssetDatabase.SaveAssets();
    }
```

Даний метод створює або оновлює записи SO. Він створює запис з певними властивостями і відредагованим шаблоном, що зберігається у пам’яті застосунку і буде використовуватися надалі для побудови рівня.

Під час роботи застосунку, у момент коли відбувається запит на побудову рівня, викликається метод BuildRoom(), що реалізує функціонал створення ігрової кімнати на основі переданого шаблону. Цей метод є асинхронним (async), що дозволяє уникнути зупинки основного потоку під час створення кімнати і повертає GameObject, що представляє побудовану кімнату.

```
    public async Task<GameObject> BuildRoom(RoomTemplateSO.Template
roomTemplate, Transform parent)
    {

        AnalyzeTemplate(roomTemplate);

        if (roomTemplate == null)
        {
```

```
            Debug.LogError("Template not found");
            return null;
        }

        GameObject roomObject = new GameObject(roomTemplate.name);
        roomObject.transform.SetParent(parent);
        roomObject.transform.localPosition = Vector3.zero;

        EnemiesParent = CreateParent("Enemies", roomObject.transform);
        BuffsParent = CreateParent("Buffs", roomObject.transform);
        WallsParent = CreateParent("Walls", roomObject.transform);
        FloorsParent = CreateParent("Floors", roomObject.transform);

        RoomContext.EnemiesParent = EnemiesParent;
        RoomContext.BuffsParent = BuffsParent;

        var templateElements = roomTemplate.TemplateElement;
        var coordinates = roomTemplate.Coordinates;

        for (int i = 0; i < templateElements.GetLength(0); i++)
        {
            for (int j = 0; j < templateElements.GetLength(1); j++)
            {
                TemplateElementType elementType = templateElements[i, j];
                Vector3 position = coordinates[i, j];
                Vector3 floorPosition = new Vector3(position.x, position.y - 1,
position.z);

                _roomObjectsFactory.Build(floorPosition, FloorsParent,
TemplateElementType.Ground);
```

```
                switch (elementType)
                {
                    case TemplateElementType.DefaultWall:
                        _roomObjectsFactory.Build(position, WallsParent, TemplateElementType.DefaultWall);
                        break;
                    case TemplateElementType.Chest:
                        IChestController chestController = _chestFactory.CreateChest(position, WallsParent);
                        RoomContext.Chests.Add(chestController);
                        break;
                }
            }
        }

        BuildExits(templateElements, coordinates);
        CenterCamera(roomTemplate);
        await Task.Delay(50);
        OnNavigationCreate(roomObject.transform);

        return roomObject;
    }
```

Метод створює новий об'єкт roomObject, що буде представляти кімнату в ігровому середовищі. Він прив'язує новостворений об'єкт до батьківського об'єкта (parent), що допомагає організувати ієрархію об'єктів у сцені.

Далі створюються батьківські об'єкти для різних типів елементів, таких як стіни, підлога, вороги та бонуси. Це спрощує управління елементами кімнати та зберігає логічну структуру кімнати.

Метод проходить по всіх клітинках шаблону кімнати (templateElements) та створює відповідні об'єкти в зазначених координатах. Кожна комірка містить інформацію про тип об'єкта, що дозволяє визначати, що саме слід створити (наприклад, стіну або скриню). Використання switch дозволяє вибирати різні типи об'єктів і створювати їх із потрібними властивостями.

Метод завершує побудову кімнати, додаючи виходи (BuildExits()), центрує камеру на створеному шаблоні (CenterCamera()) та викликає метод OnNavigationCreate(), який забезпечує подальшу навігацію. Невелика затримка (await Task.Delay(50)) гарантуватиме синхронізацію у процесі створення.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ІНСТРУМЕНТУ ДЛЯ СТВОРЕННЯ ІГРОВИХ РІВНІВ

4.1 Тестування роботи інструменту для генерації рівнів

Головне меню інструменту наведено на рисунку 4.1.

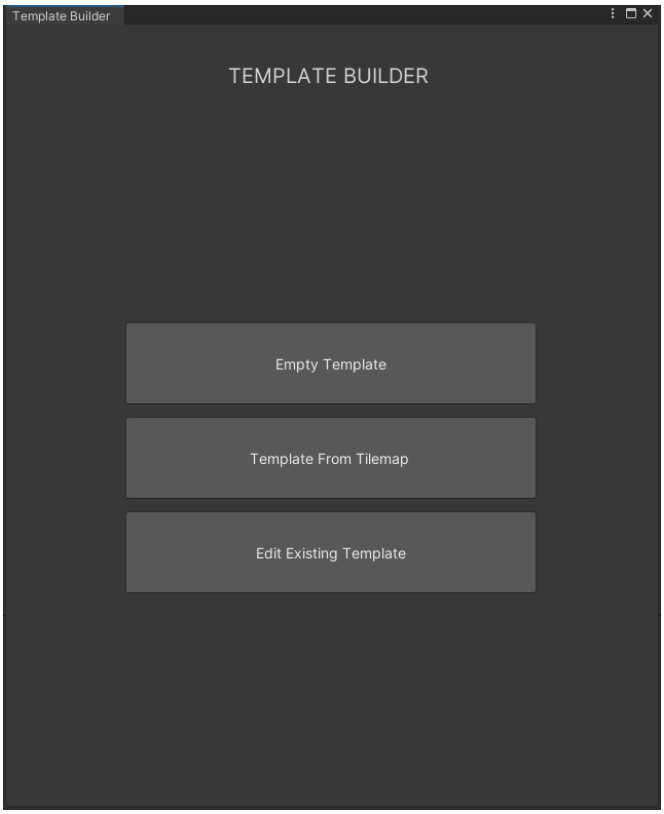

Рисунок 4.1 — Головне меню інструменту.

У меню є можливість створити новий шаблон, створити шаблон на основі TileMap, або редагувати вже створені шаблони. Якщо натиснути на кнопку “Empty Template” відкриється меню створення нового шаблону:

Рисунок 4.2 — Меню створення нового шаблону.

У цьому меню можна задати початкові розміри шаблону і створити його, або повернутися назад у головне меню.

При натисканні на кнопку “Edit Existing Template” відкриється меню вибору вже створених шаблонів:

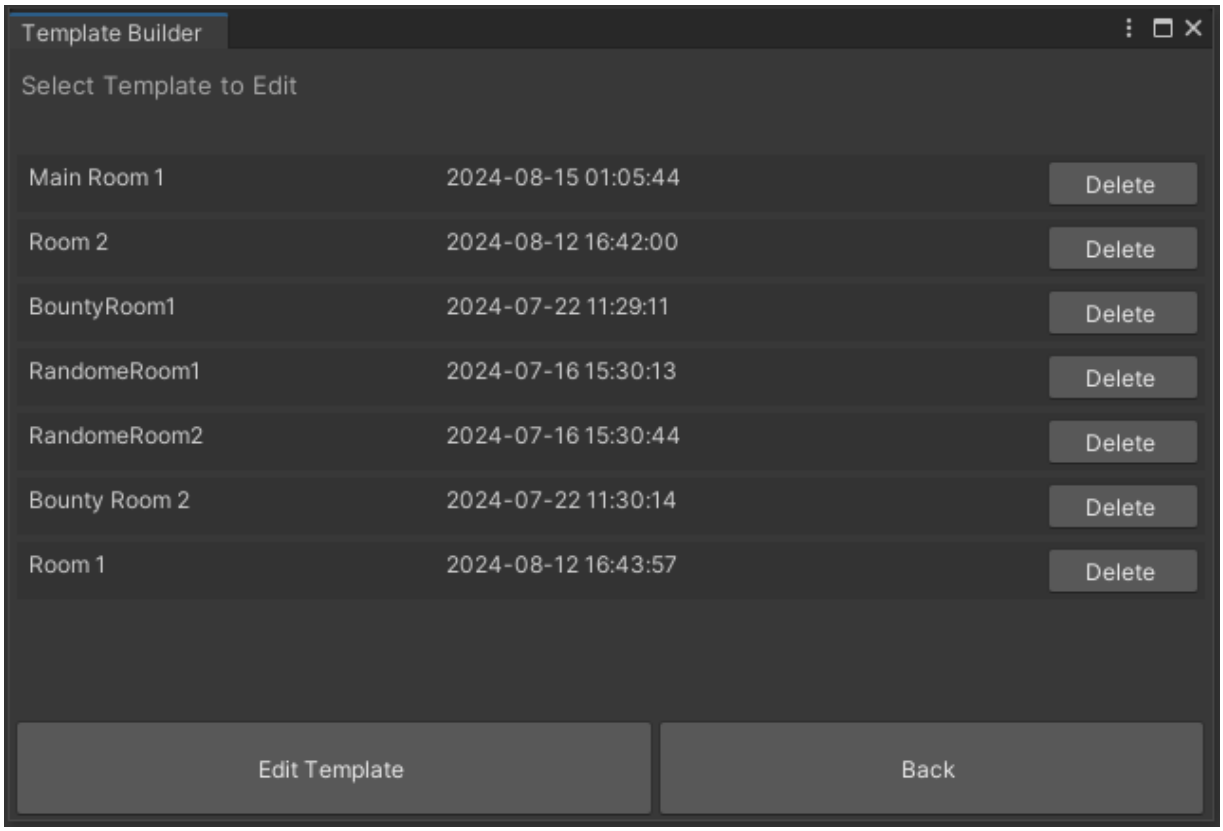

Рисунок 4.3 — Меню вибору створених шаблонів.

У цьому меню користувач має змогу обрати вже створені шаблони для редагування, побачити час їх останніх модифікацій і видалити непотрібні.

При обранні або створенні нового шаблону, відкривається вікно модифікації шаблонів:

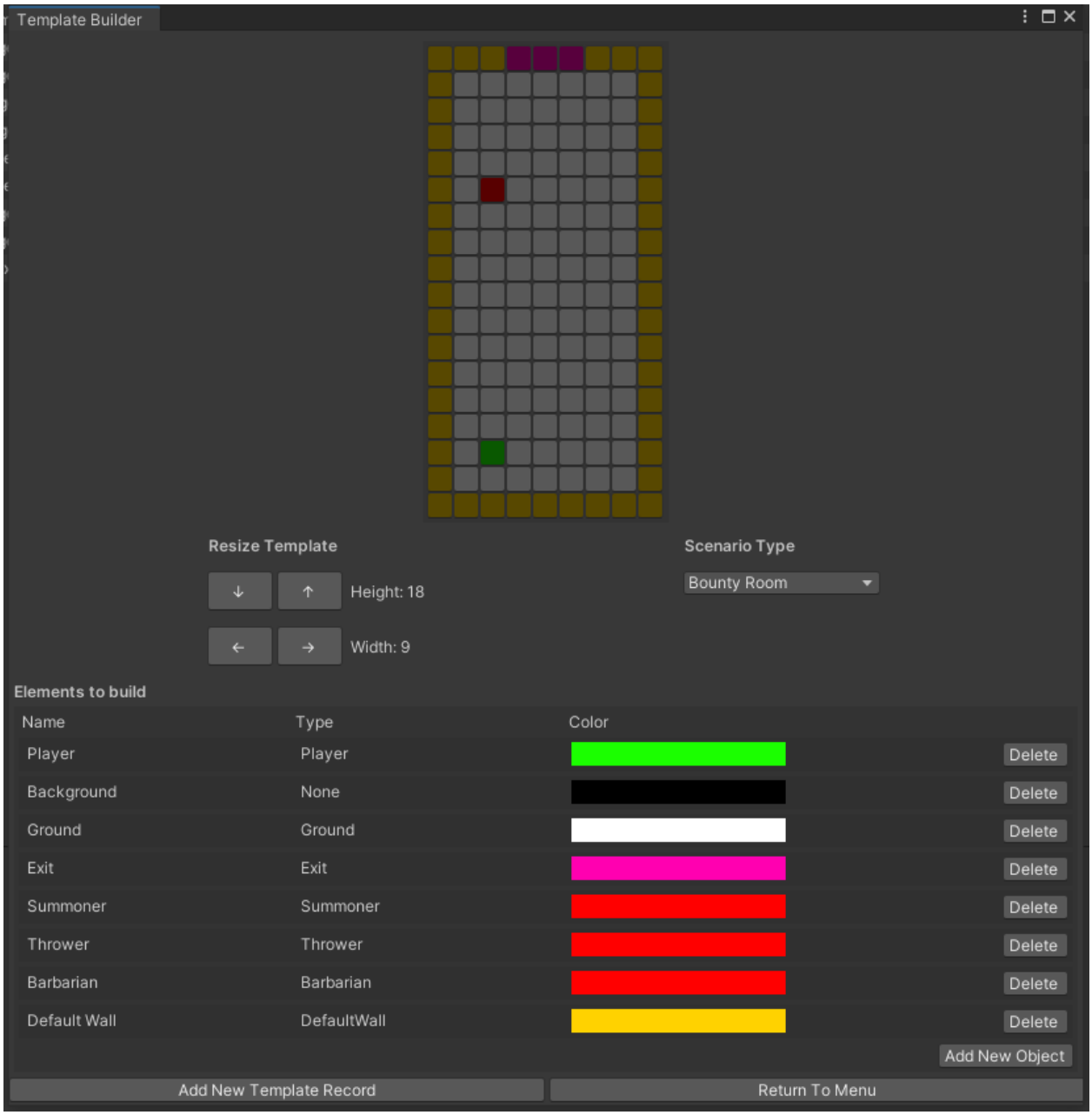

Рисунок 4.4 — Вікно модифікації шаблонів.

Дане вікно це інтерфейс взаємодії користувача з шаблоном. Саме у такому вигляді, замальовуючи картинки кольорами певного типу, користувач створює рівень. У вікні є кнопки з опціями збільшення і зменшення розмірів шаблону, встановлення типу кімнати, список із об’єктами, які можуть бути додані на карту і можливістю керування ними. При натисканні “Add New Template Record” створюється запис у SO, який має наступний вид:

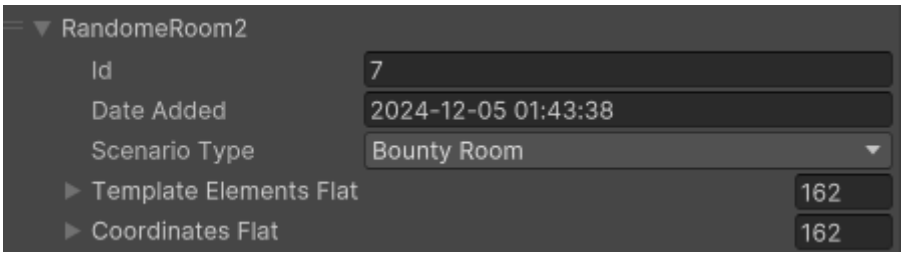

Рисунок 4.5 — вигляд SO із збереженим шаблоном.

Після того як у грі надходить запит на створення рівня, обирається певний шаблон із збережених і будується у ігровому просторі, відповідно до заповнених даних шаблону.

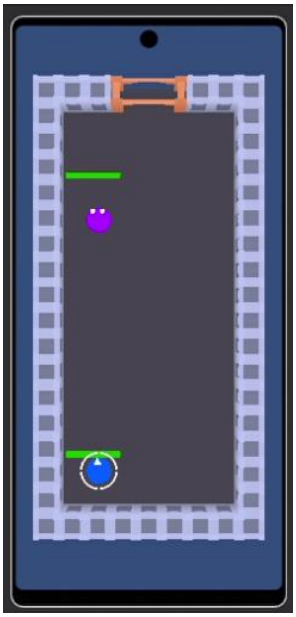

Рисунок 4.6 — побудований рівень у ігровому просторі.

4.2 Аналіз роботи інструменту для генерації рівнів

Методика тестування інструменту була спрямована на всебічну оцінку його продуктивності, ефективності та здатності спрощувати процес створення та збереження ігрових рівнів у середовищі казуальних кросплатформенних ігор. Для перевірки функціональності інструменту було створено та протестовано рівні з різними параметрами, включаючи кількість елементів, їх варіативність, а також способи розташування об'єктів у межах ігрового

простору. Такий підхід дозволив максимально наблизити умови тестування до реальних сценаріїв, з якими стикаються розробники в процесі створення ігор.

Основна мета тестування полягала у визначенні кількох ключових аспектів роботи інструменту: оптимізація використання ресурсів пам'яті, швидкість створення рівнів, ефективність збереження вихідних файлів та гнучкість інструменту у взаємодії з ігровими даними. Зокрема, було важливо оцінити, як інструмент справляється зі створенням складних ігрових рівнів з великою кількістю об'єктів, наскільки спрощується процес проєктування, а також чи дозволяє інструмент зберігати оптимізовані дані рівнів без зайвих витрат пам'яті.

Для тестування була розроблена уніфікована структура рівнів із фіксованими розмірами — 9x18 блоків, що забезпечило однакові умови для порівняння продуктивності інструменту з існуючими аналогами. Така структура рівнів була обрана як типовий приклад для казуальних ігор, де переважно використовуються сітчасті системи або шаблонні підходи для побудови локацій.

Перший етап тестування передбачав оцінку часу створення рівнів. Для цього фіксувалася тривалість процесу від початкового проєктування рівня, через розміщення об'єктів, до остаточного збереження. На практиці цей процес включав кілька ключових кроків: вибір шаблону, редагування його структурних компонентів, розташування ігрових елементів та збереження у форматі готового файлу. Оцінка часу дозволила перевірити продуктивність інструменту в порівнянні з аналогами, такими як Unity Tilemap Editor, Godot TileMap, Unreal Engine Paper2D, Tiled, AutoTileGen та DunGen.

Другий етап полягав у вимірюванні використання внутрішньої пам'яті інструментом під час процесу створення рівнів. Для цього моніторилися обсяги оперативної пам'яті, яку споживав інструмент під час виконання основних операцій — ініціалізації, редагування шаблонів, обробки елементів і їх збереження. Це дозволило оцінити, наскільки ефективно інструмент використовує ресурси системи та чи відповідає він вимогам до легковагового

програмного забезпечення для казуальних ігор, що часто працюють на мобільних пристроях або системах із обмеженою продуктивністю.

Третій етап тестування був присвячений аналізу обсягу вихідного файлу рівня. Файл, що містив збережені дані про рівень, оцінювався на предмет оптимізації — як за розміром, так і за структурою. Було перевірено, чи не містить файл надлишкових даних та чи забезпечує він ефективне зберігання інформації про розташування ігрових елементів. Менший обсяг файлу є критично важливим для кросплатформенних ігор, де зменшення розмірів ігрових ресурсів дозволяє покращити завантаження рівнів та оптимізувати використання пам'яті пристрою.

Під час тестування кожен з інструментів-аналогів був оцінений за тими ж критеріями, що й розроблений інструмент. Unity Tilemap Editor, Godot TileMap, Unreal Engine Paper2D, Tiled, AutoTileGen та DunGen є популярними рішеннями для створення ігрових рівнів, але кожен із них має свої обмеження у швидкості роботи, використанні ресурсів або оптимізації вихідних даних. Проведене тестування дозволило об'єктивно порівняти їхні можливості з результатами розробленого інструменту.

Для оцінки інструменту застосовувалися такі метрики:

— час створення рівня необхідного для створення повного рівня, включаючи розташування елементів і їх збереження;

— використання внутрішньої пам'яті споживає інструмент під час роботи;

— обсяг вихідного файлу рівня, що містить дані створеного рівня.

Тестування проводилось на рівнях розміром 9x18 блоків. Для порівняння результатів були використані інструменти-аналоги: Unity Tilemap Editor, Godot TileMap, Unreal Engine Paper2D, Tiled, AutoTileGen та DunGen. Результати представлені в таблиці 4.1.

Таблиця 4.1 — Результати тестування

Інструмент	Час створення рівня (години)	Використання внутрішньої пам’яті (МБ)	Обсяг вихідного файлу (КБ)
Розроблений інструмент	1–2	90	2.656
Unity Tilemap Editor	5-6	200	650
Godot TileMap	4-5	180	500
Unreal Engine Paper2D	6-7	300	700
Tiled	3-4	150	400
AutoTileGen	4-5	220	550
DunGen	5-6	250	600

Результати тестування інструменту показали його значну перевагу у порівнянні з популярними аналогами, що були обрані для порівняння. До основних конкурентів належать Unity Tilemap Editor, Godot TileMap, Unreal Engine Paper2D, Tiled, AutoTileGen та DunGen — інструменти, які широко використовуються у розробці ігрових рівнів. Тестування проводилося за трьома ключовими метриками: час створення рівня, використання внутрішньої пам'яті та обсяг вихідного файлу, що дозволяє комплексно оцінити ефективність інструменту в реальних умовах.

Перша метрика — час створення рівня — є критично важливим показником для розробників, оскільки швидкість побудови контенту напряму впливає на продуктивність команди і терміни випуску гри. Результати тестування демонструють, що розроблений інструмент забезпечує суттєве зниження часу на створення рівня, яке становить 1–2 години. Для порівняння, Unity Tilemap Editor потребує 56 годин, Godot TileMap — 45 годин, а Unreal

Engine Paper2D — 67 годин. Найкращий показник серед аналогів у Tiled, що становить 34 години, однак це все ще в декілька разів більше, ніж результат розробленого інструменту. Зменшення часу на 60-80% свідчить про високу продуктивність та ефективність інструменту у процесі проєктування рівнів, що дозволяє значно пришвидшити роботу над ігровим контентом.

Друга метрика — використання внутрішньої пам'яті, показала, наскільки інструмент оптимізований у контексті споживання ресурсів системи. Під час тестування розроблений інструмент споживав лише 90 МБ оперативної пам'яті, що є найнижчим значенням серед усіх перевірених аналогів. Для порівняння, Tiled, який показав найближчий результат, споживав 150 МБ, а Godot TileMap — 180 МБ. Найгірші показники були у Unreal Engine Paper2D (300 МБ) та DunGen (250 МБ). Це підтверджує ефективність запропонованого інструменту для роботи у середовищах із обмеженими ресурсами, що є особливо важливим для мобільних та кросплатформенних казуальних ігор.

Третя метрика — обсяг вихідного файлу рівня — є визначальним фактором для оптимізації зберігання та завантаження ігрових ресурсів. Вихідний файл, створений за допомогою розробленого інструменту, має розмір 2.656 КБ, що на 99,6% менше, ніж у Unity Tilemap Editor, де обсяг файлу досягає 650 КБ. Інші аналоги показали результати в межах від 400 КБ (Tiled) до 700 КБ (Unreal Engine Paper2D). Така оптимізація обсягу даних дозволяє суттєво знизити вимоги до пам'яті на пристроях користувачів, пришвидшити завантаження рівнів та покращити загальну продуктивність гри.

Загальний аналіз результатів підтвердив, що запропонований інструмент значно перевершує свої аналоги за всіма основними критеріями. По-перше, час створення рівнів знижено у декілька разів, що дозволяє мінімізувати трудовитрати на розробку ігрового контенту. По-друге, використання пам'яті оптимізоване до рівня, який робить інструмент надзвичайно ефективним для платформ із обмеженими ресурсами, зокрема мобільних пристроїв. По-третє, мінімальний обсяг вихідного файлу забезпечує компактність збережених даних та ефективну роботу з великою кількістю ігрових рівнів.

Загалом результати тестування підтвердили, що розроблений інструмент забезпечує ефективність, гнучкість і економію ресурсів, що є критично важливим для казуальних ігор. Запропонований підхід дозволяє не лише оптимізувати процес створення рівнів, але й забезпечити їхню відповідність сучасним вимогам до кросплатформенних ігрових застосунків.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку Додаток до ігрового рушія Unity для генерації рівнів у 2д іграх. Особливістю розробки є спрощення процесу розробки кросплатформенних казуальних ігрових застосунків за рахунок додатку до ігрового рушія, оптимізуючи та прискорюючи побудову рівнів і розміщення ігрових об’єктів. Аналогом розробки може бути Super Tilemap Editor - 45 $ (1800 грн).

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження табл. 5.1

Ринкові переваги					
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною.	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження табл. 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	4	4	4
Наявність аналогів на ринку	4	4	4
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	4	4
Експлуатаційні витрати	4	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	4	4	4
Фахівці з технічної і комерційної реалізації	3	4	4
Фінансування	3	4	4
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	3	4
Супровідна документація	4	3	3
Сума	45	44	45
Середньоарифметична сума балів	(45+44+45) / 3 = 44,67		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається завдяки удосконаленню метода створення ігрових рівнів за рахунок додатку до ігрового рушія.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t, \quad (5.1)$$

де M – місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ – число робочих днів за місяць, 21 днів;

t – число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	29300	1395,24	25	34880,952
Програміст	28000	1333,33	25	33333,333
Всього				68214,29

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

5.2.2 Додаткова заробітна плата розробників, які брати участь в розробці обладнання/програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 12 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 12\ \% / 100\ \% \qquad (5.2)$$

$$З_д = (68214,29 \cdot 12\ \% / 100\ \%) = 8185,71 \ \text{(грн.)}$$

5.2.3 Нарахування на заробітну плату розробників згідно діючого законодавства складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (5.3)$$

$$Н_з = (68214,29 + 8185,71) \cdot 22\ \% / 100\ \% = 16808,00 \ \text{(грн.)}$$

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.]}, \quad (5.4)$$

де Ц – балансова вартість обладнання, грн.;

Т–термін корисного використання обладнання згідно податкового законодавства, років;

$t_{вик}$ – термін використання під час розробки, місяців.

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 20000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 1,19 міс.

$$\text{А}_{\text{обл}} = \frac{20000}{2} \times \frac{1{,}19}{12} = 992{,}063 \text{ грн.}$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.5. Так як вартість ліцензійної ОС менше 20000 грн, то вона включається в розробку повністю, $B_{нем.ак.}$ = 6299 грн.

Таблиця 5.5 — Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	20000	2	1,19	992,063
Офісне обладнання (меблі)	22000	4	1,19	545,635
Приміщення	1300000	20	1,19	6448,413
Всього				7986,11

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В – вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2024 році для Вінницької області за даними Енера-Вінниця, В = 10,42 грн./кВт;

П – встановлена потужність обладнання, кВт. П = 0,3 кВт;

Ф – фактична кількість годин роботи обладнання, годин:

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,3 \cdot 8 \cdot 25 \cdot 10,42 = 562,68 \text{ (грн.)}$$

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$I_в = 68214{,}29 * 77\% / 100\% = 52525 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 68214{,}29 * 122\ \% / 100\ \% = 83221 \text{ (грн.)}$$

5.2.8 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$B_{заг} = 68214,29+8185,71+16808,00+7986,11+6299+562,68+52525+$$
$$+83221 = 243802,22 \text{ грн.}$$

5.2.9 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{B_{заг}}{\eta} \text{ (грн)}, \qquad (5.8)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 243802,22 / 0,5 = 487604 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у

потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її

можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.9)$$

де $\pm\Delta Ц_o$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

Цб – вартість програмного продукту у році до впровадження результатів розробки;

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p – коефіцієнт, який враховує рентабельність продукту;

ϑ – ставка податку на прибуток, у 2024 році ϑ =18%.

Припустимо, що при прогнозованій ціні 1100 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *100* грн. Кількість одиниць

реалізованої продукції також збільшиться: протягом першого року – на 6000 шт., протягом другого року – на 5000 шт., протягом третього року на 4000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*100 + (1100 + 100)*6000)* 0{,}8333* 0{,}35) * (1 - 0{,}18) = 1578499{,}937 \text{ грн.}$$

$$\Delta П_2 = (0*100 + (1100 + 100)*(6000+5000)* 0{,}8333* 0{,}35) * (1 - 0{,}18) = 3156999{,}874 \text{ грн.}$$

$$\Delta П_3 = (0*100 + (1100 + 100)*(6000+5000+4000)* 0{,}8333* 0{,}35) * (1 - 0{,}18) = 4304999{,}828 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 9040499,64 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0{,}05 \ldots 0{,}15$;

t – період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$$ПП = (1578499,937/(1+0,1)^1)+(3156999,874/(1+0,1)^2)+(4304999,828/$$
$$/(1+0,1)^3) = 1434999,94 + 2609090,805 + 3234410,089 = 7278500,836 \text{ грн.}$$

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 487604 = 975208,88 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП – PV, \quad (5.12)$$

$$E_{абс} = 7278500,836 – 975208,88 = 6303291,96 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.13}$$

де $T_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{(1 + 6303291{,}96/975208{,}88} - 1 = 0{,}954$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.14}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \quad (5.15)$$

$$T_{ок} = 1 / 0{,}954 = 1{,}05 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 1,05 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 487604 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 1,05 роки.

ВИСНОВКИ

У роботі було розглянуто сучасний стан розробки казуальних кросплатформенних ігрових застосунків та проаналізовано виклики, які постають перед розробниками таких продуктів. Було детально досліджено рушії та інструменти, які зазвичай використовуються в розробці, а також виявлено основні проблеми, пов'язані з використанням ресурсів та оптимізацією процесу створення контенту. Проведений аналіз показав необхідність створення спеціального інструменту для побудови ігрових рівнів, що допомагає скоротити витрати часу на розробку, а також підвищити ефективність використання апаратних ресурсів.

У другому розділі було розроблено структуру та архітектуру інструменту для побудови рівнів. Вибір використовуваних алгоритмів та технологій було обгрунтовано з огляду на вимоги до гнучкості та оптимальності. Зокрема, розроблено математичні моделі, що описують процес збереження та побудови рівнів, а також метод збереження даних, який дозволяє значно знизити вимоги до використання пам'яті.

У третьому розділі було обгурнтовано вибір об'єктно-орієнтованої мови програмування високого рівня C# та середовища програмування Visual Studio, так як вони оптимальні для реалізації поставленої задачі. Було протестовано розроблений інструмент і порівняно цифрові показники із аналогами. Аналіз роботи довів ефективність використання розробленого продукту.

У четвертому розділі виконано комерційний та технологічний аудит розробки, розраховано економічну ефективність її впровадження. Результати розрахунків свідчать про високу конкурентоспроможність програмного продукту та швидкий період окупності інвестицій.

Таким чином, мета роботи — зменшення використання ресурсів пристрою та часу на побудову ігрових рівнів за рахунок створення інструменту для побудови, зберігання та динамічного створення ігрових рівнів — була досягнута, що забезпечує важливий крок у напрямку підвищення ефективності розробки казуальних кросплатформенних ігор.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЬ

1. Крейчі В.Б., Савицька Л.А; Метод створення кросплатформенних казуальних ігрових застосунків. Молодь в науці: дослідження, проблеми, перспективи, Ukraine, mar. 2024. Available at: <https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21706>. Date accessed: 26 May. 2024.

2. Крейчі В.Б., Савицька Л.А., Крупельницький Л.В; Метод створення кросплатформенних казуальних ігрових застосунків. LIV Всеукраїнської науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії, Ukraine, mar. 2025. Available at: <https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/22730/18836>. Date accessed: 26 May. 2025.

3. Juul, J. A Casual Revolution: Reinventing Video Games and Their Players. Cambridge: MIT Press, 2012. 312 p.

4. Johnson, S. The History of Casual Games: From Puzzles to Digital Platforms. New York: Game History Press, 2019. 250 p.

5. Mäyrä, F., Alha, K. Mobile Gaming: The Next Evolution in Casual Games. London: Routledge, 2020. 296 p.

6. Ivory, J. D. Brief History of Video Games. London: Routledge, 2015. 320 p.

7. Nieborg, D. Candy Crush and Beyond: Monetization Strategies in Casual Games. Amsterdam: Digital Culture Press, 2015. 200 p.

8. Leaver, T., Willson, M. Social Casual Games: The Role of Networks. Sydney: Social Interaction Press, 2016. 280 p.

9. Paavilainen, J., Kultima, A., Mäyrä, F. Game Design Challenges in Mobile Casual Games. Tampere: Game Research Press, 2009. 220 p.

10. Unity Technologies. Unity Manual: Tilemap and 2D Tools. [Електронний ресурс]. URL: https://docs.unity3d.com/Manual/Tilemap.html.

11. Godot Documentation. TileMap Class Reference. [Електронний ресурс]. URL: https://docs.godotengine.org/en/stable/classes/class_tilemap.html.

12. Epic Games. Paper2D User Guide for Unreal Engine. [Електронний ресурс]. URL: https://docs.unrealengine.com/4.27/en-US/Paper2D.

13. Tiled Map Editor. Official Documentation. [Електронний ресурс]. URL: https://doc.mapeditor.org.

14. AutoTileGen. Automated Tilemap Generation for Unity. [Електронний ресурс]. URL: https://autotilegen.com.

15. DunGen. Procedural Dungeon Generation in Unity. [Електронний ресурс]. URL: https://assetstore.unity.com/packages/tools/level-design/dungen-157253.

16. Chow, C. Optimizing Game Performance for Mobile Devices. New York: Mobile Press, 2018. 300 p.

17. Huang, T. Effective Memory Management in Unity Games. San Francisco: Unity Insider Press, 2017. 320 p.

18. Pressman, R. S. Software Engineering: A Practitioner's Approach. Boston: McGraw-Hill, 2014. 896 p.

19. Witten, I. H., Frank, E., Hall, M. A. Data Mining: Practical Machine Learning Tools and Techniques. Amsterdam: Elsevier, 2011. 664 p.

20. Al-Tabbaa, O., Khan, S. Cross-Platform Development Challenges and Solutions. London: Tech Insight Press, 2023. 250 p.

21. Lal, P. Designing for Mobile: UX Challenges in Casual Games. Boston: UX Design Press, 2019. 260 p.

22. Barrett, J., Fry, E. Game AI Pro 3: Collected Wisdom of Game AI Professionals. Boca Raton: CRC Press, 2016. 354 p.

23. Unity Blog. Reducing Memory Usage in Unity Projects. [Електронний ресурс]. URL: https://blog.unity.com/technology/reducing-memory-usage-in-unity-projects.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

_____ проф., д.т.н. О.Д. Азаров

«29» вересня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

Методі створення кросплатформенних казуальних ігрових застосунків

08-54.МКР.009.00.000 ТЗ

Керівник роботи: к.т.н., доц. каф. ОТ:

Савицька Л.А.

Виконав: студент гр. 1КІ-23м

Крейчі В.Б.

2024

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність роботи полягає в необхідності вдосконалення методів створення казуальних кросплатформенних ігор, що залишаються популярними завдяки своїй доступності та простоті, але потребують оптимізації процесу розробки ігрових рівнів для зменшення витрат часу та ресурсів, що є критично важливим для малих команд розробників і сучасних кросплатформенних продуктів.

1.2 Наказ про затвердження теми МКР № 310 від 17.09.2024 р.

2 Мета і призначення МКР

2.1 Метою дослідження є зменшення використання ресурсів пристрою та часу на побудову ігрових рівнів за рахунок створення інструменту для побудови, зберігання та динамічного створення ігрових рівнів;

2.2 Призначення розробки — автоматизація створення та побудови ігрових рівнів у казуальних кросплатформенних іграх з використанням процедурних і шаблонних методів для оптимізації ресурсів та скорочення часу розробки.

3 Вихідні дані для виконання МКР: ігровий рушій Unity, параметри рівня (розмір 9 на 18), операційна система Windows.

4 Технічні вимоги до виконання МКР

Технічні вимоги:

— аналіз сучасних кросплатформенних казуальних ігрових застосунків та методів їх створення;

— дослідження існуючих методів створення ігрових рівнів;

— вибір оптимальних засобів створення ігрових рівнів;

— розробка прототипу інструменту для створення та побудови ігрових рівнів;

— тестування роботи інструменту для створення та побудови ігрових рівнів;

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1

Таблиця А.1 — Етапи МКР

№ з/п	Назва етапів дипломної роботи	Термін виконання		Очікувані результати
		початок	закінчення	
1	Постановка задачі роботи	19.09.2024	19.09.2024	Задачі дослідження
2	Аналіз предметної області	20.09.2024	25.09.2024	Розділ 1
3	Проєктування інструменту для побудови рівнів	26.09.2024	5.10.2024	Розділ 2
4	Програмна реалізація інструменту для побудови рівнів	06.10.2024	26.10.2024	Розділ 3
5	Розрахунок економічної частини	11.11.2024	11.11.2024	Розділ 4
6	Оформлення матеріалів до захисту МКР	12.11.2024	12.11.2024	ПЗ, графічний матеріал і презентація
7	Перевірка якості виконання магістерської роботи та усунення недоліків	13.11.2024	14.11.2024	Оформлені документи
9	Підписи супроводжувальних документів у нормоконтролера, керівника, опонента	15.11.2024	15.11.2024	Оформлені документи
10	Перевірка на антиплагіат та ШІ	17.11.2024	17.11.2024	Оформлені документи

6 Матеріали, що подаються до захисту МКР

Пояснювальна записка МКР, графічні та ілюстративні матеріали, протокол попереднього захисту роботи на кафедрі, відзив наукового керівника, рецензія рецензента, анотації до МДП українською та іноземною

мовами, нормоконтроль про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21.

ДОДАТОК Б

Архітектура інструменту створення та побудови рівнів

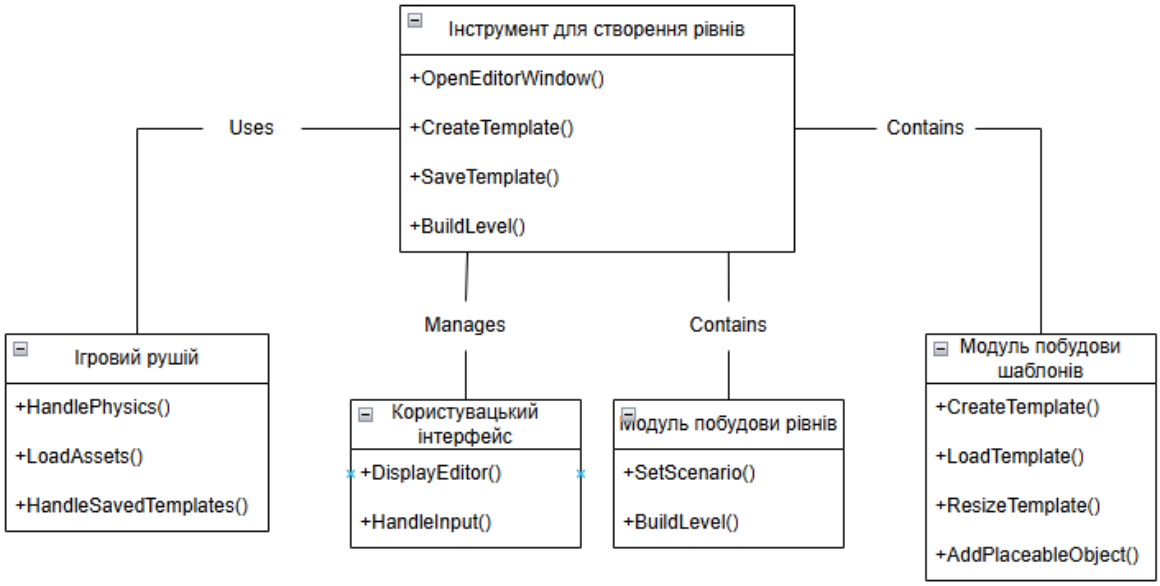

Рисунок Б.1 — Архітектура інструменту створення та побудови рівнів

ДОДАТОК В

Блок-схема алгоритму роботи інструменту для створення ігрових рівнів

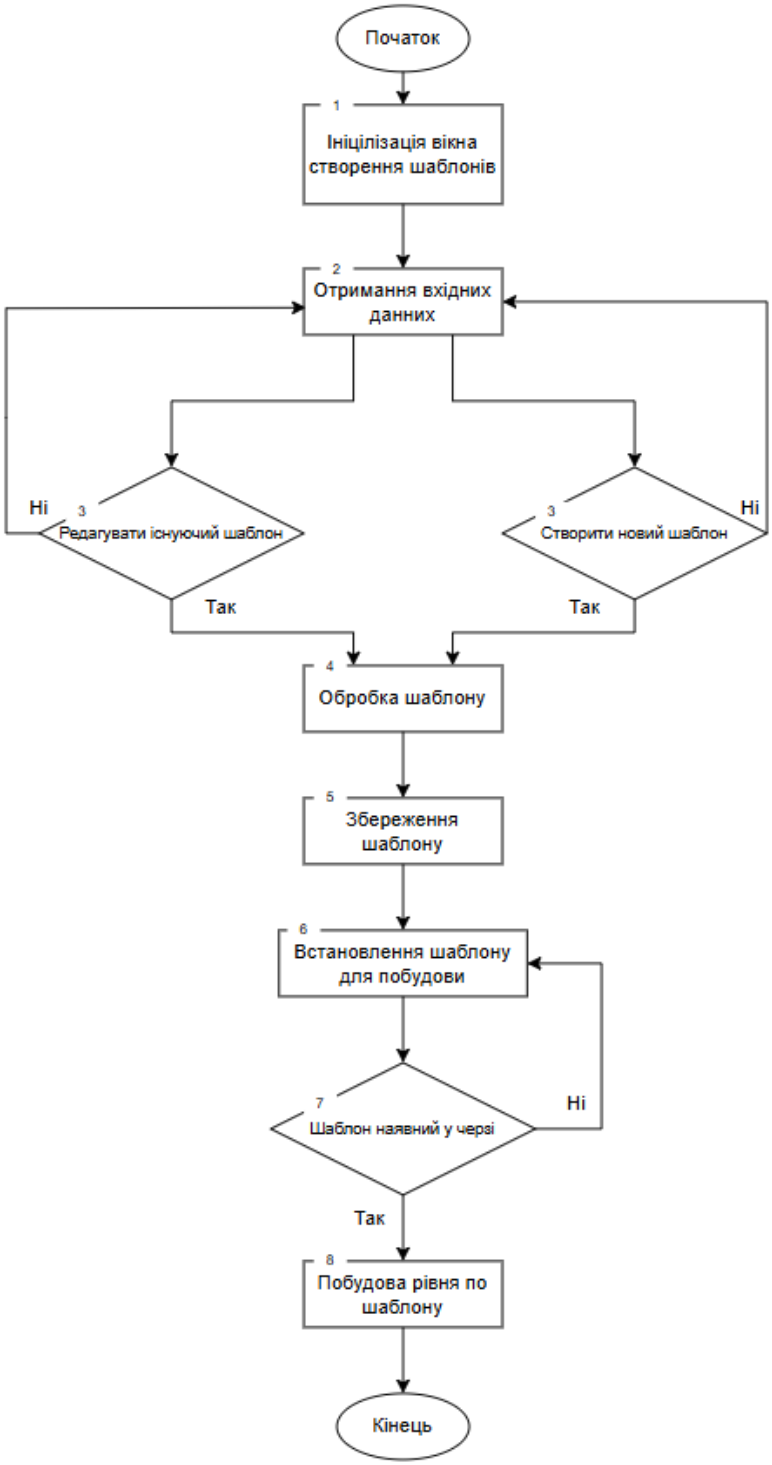

Рисунок В.1 — Блок-схема алгоритму роботи інструменту для створення ігрових рівнів

ДОДАТОК Г

Лістинг модуля створення шаблонів

```
using UnityEngine;
using UnityEditor;
using UnityEngine.Tilemaps;
using System.Collections.Generic;
using System.Linq;

public class TemplateBuilder : EditorWindow
{
    #region Fields
    private Tilemap selectedTilemap;
    private TemplateElementType[,] _levelArray;
    private TemplateElementType[,] _originalLevelArray;

    private Color[,] _cellColors;
    private bool[,] _isEditableArray;
    private Vector2 _scrollPosition;

    private List<TemplatePlacebleElements.TemplatePlacebleElement>
_placeableObjects = new
List<TemplatePlacebleElements.TemplatePlacebleElement>();
    private int _selectedObjectIndex = -1;

    private string _newObjectName = "";
    private Color _newObjectColor = Color.white;
```

```
        private TemplateElementType _newObjectType =
TemplateElementType.Ground;

        private float _zoomScale = 1f;

        private string _newRecordName = "";
        private bool _DisplayNewTemplateRecordFields = false;
        private bool _displayNewObjectFields = false;

        private TemplatePlacebleElements _templatePlacebleElementsSO;
        private TypeOfScenario _selectedScenarioType =
TypeOfScenario.DefaultRoom;

        private bool _showInitialOptions = true;
        private bool _createNewArray = false;
        private bool _createFromTilemap = false;
        private bool _editExistingTemplate = false;
        private bool _resizeArray = false;

        private bool _isMousePressed = false;

        private int _arrayWidth = 0;
        private int _arrayHeight = 0;

        private GUIStyle _headerStyle;
        private GUIStyle _buttonStyle;
```

```
        private RoomTemplateSO _roomTemplateSO;

        private List<RoomTemplateSO.Template> _templates = new
List<RoomTemplateSO.Template>();

        private int _selectedTemplateIndex = -1;


        private int _deleteIndex = -1;


        private float _lastClickTime;

        private const float DoubleClickThreshold = 0.3f;


        private const string C_TemplatePlaceableElementsPath =
"Assets/SO/TemplatePlacebleElementsSO.asset";

        private const string C_RoomTemplateSOPath =
"Assets/SO/RoomTemplateSO.asset";

        #endregion


        private void CreateEmptyArray()

    {

        if (_arrayWidth <= 0 || _arrayHeight <= 0)

        {

            Debug.LogError("Width and Height must be greater than 0.");

            return;

        }


        _levelArray = new TemplateElementType[_arrayWidth, _arrayHeight];
```

```
            _cellColors = new Color[_arrayWidth, _arrayHeight];
            _isEditableArray = new bool[_arrayWidth, _arrayHeight];

            for (int x = 0; x < _arrayWidth; x++)
            {
                for (int y = 0; y < _arrayHeight; y++)
                {
                    _levelArray[x, y] = TemplateElementType.Ground;
                    _cellColors[x, y] = Color.white;
                    _isEditableArray[x, y] = true;
                }
            }

            Debug.Log($"Created empty array of size
{_arrayWidth}x{_arrayHeight}.");
        }
        private void DisplayArrayEditor()
        {
            GUILayout.BeginVertical(GUI.skin.box);

            int cellSize = Mathf.FloorToInt(20 * _zoomScale);

            for (int y = _levelArray.GetLength(1) - 1; y >= 0; y--)
            {
                GUILayout.BeginHorizontal();
```

```
            for (int x = 0; x < _levelArray.GetLength(0); x++)
            {
                Rect rect = GUILayoutUtility.GetRect(cellSize, cellSize,
GUILayout.ExpandWidth(false), GUILayout.ExpandHeight(false));

                Color buttonColor = _cellColors[x, y];
                GUI.backgroundColor = buttonColor;
                if (GUI.Button(rect, GUIContent.none) || (_isMousePressed &&
rect.Contains(Event.current.mousePosition)))
                {
                    if (_selectedObjectIndex != -1)
                    {
                        TemplateElementType previousType = _levelArray[x, y];
                        _levelArray[x, y] =
_placeableObjects[_selectedObjectIndex].Type;
                        TemplateElementType newType = _levelArray[x, y];
                        Debug.Log($"Cell changed from {previousType} to
{newType}");
                        _cellColors[x, y] =
_placeableObjects[_selectedObjectIndex].Color;
                    }
                }
                GUI.backgroundColor = Color.white;
            }

            GUILayout.EndHorizontal();
```

```
            }


            GUILayout.EndVertical();

        }

        private void CreateOrUpdateRecordInSO(string name,
TemplateElementType[,] levelArray)

        {

            if (_roomTemplateSO == null)

            {

                Debug.LogError("RoomTemplateSO not found at path:
Assets/SO/RoomTemplateSO.asset");

                return;

            }


            if (_roomTemplateSO.Templates == null)

            {

                _roomTemplateSO.Templates = new
List<RoomTemplateSO.Template>();

            }


            var existingTemplate = _selectedTemplateIndex >= 0 &&
_selectedTemplateIndex < _templates.Count

                                   ? _templates[_selectedTemplateIndex]

                                   : null;


            if (existingTemplate != null && name == existingTemplate.name)
```

```
            {
                existingTemplate.TemplateElement =
(TemplateElementType[,])levelArray.Clone();
                existingTemplate.ScenarioType = _selectedScenarioType;
                existingTemplate.DateAdded =
System.DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
                AssignExits(existingTemplate, levelArray);
                Debug.Log($"Updated existing template: {name}");
            }
            else
            {
                var sameNameTemplate =
_roomTemplateSO.Templates.FirstOrDefault(t => t.name == name);
                if (sameNameTemplate != null)
                {
                    Debug.LogError($"Template with name {name} already exists.
Choose a different name.");
                    return;
                }

                int id = _roomTemplateSO.Templates.Count > 0 ?
_roomTemplateSO.Templates.Max(t => t.id) + 1 : 1;

                RoomTemplateSO.Template newTemplate = new
RoomTemplateSO.Template
                {
                    name = name,
```

```
                id = id,
                TemplateElement = (TemplateElementType[,])levelArray.Clone(),
                ScenarioType = _selectedScenarioType,
                DateAdded = System.DateTime.Now.ToString("yyyy-MM-dd
HH:mm:ss")
            };

            AssignExits(newTemplate, levelArray);
            _roomTemplateSO.Templates.Add(newTemplate);
            Debug.Log($"Added new template: {name}");

            if (existingTemplate != null)
            {
                existingTemplate.TemplateElement = _originalLevelArray;
                Debug.Log($"Restored original template:
{existingTemplate.name}");
            }
        }

        EditorUtility.SetDirty(_roomTemplateSO);
        AssetDatabase.SaveAssets();
    }
}
```

ДОДАТОК Д

Лістинг модуля побудови ігрових рівнів

```
using Gameplay;
using Obstacles;
using System.Collections.Generic;
using System.Threading.Tasks;
using Unity.AI.Navigation;
using UnityEngine;

public interface IRoomBuilder
{
    public Transform PlayersParent { get; set; }
    public Transform EnemiesParent { get; set; }
    public Transform BuffsParent { get; set; }
    public Transform WallsParent { get; set; }
    public Transform FloorsParent { get; set; }
    //public void OnNavigationCreate(Transform parent);
    public Task<GameObject> BuildRoom(RoomTemplateSO.Template
roomTemplate, Transform parent);
}

public class RoomBuilder : IRoomBuilder
{
    public IRoomContext RoomContext { get; set; }
    public Transform PlayersParent { get; set; }
    public Transform EnemiesParent { get; set; }
```

```
        public Transform BuffsParent { get; set; }
        public Transform WallsParent { get; set; }
        public Transform FloorsParent { get; set; }

        private INavigationFactory _navigationFactory;
        private IRoomObjectsFactory _roomObjectsFactory;
        private IChestFactory _chestFactory;

        public RoomBuilder(
            INavigationFactory navigationFactory,
            IRoomObjectsFactory roomObjectsFactory,
            IChestFactory chestFactory)
        {
            _navigationFactory = navigationFactory;
            _roomObjectsFactory = roomObjectsFactory;
            _chestFactory = chestFactory;
        }

        public async Task<GameObject>
BuildRoom(RoomTemplateSO.Template roomTemplate, Transform parent)
        {

            AnalyzeTemplate(roomTemplate);

            if (roomTemplate == null)
```

```
{
    Debug.LogError("Template not found");
    return null;
}

GameObject roomObject = new GameObject(roomTemplate.name);
roomObject.transform.SetParent(parent);
roomObject.transform.localPosition = Vector3.zero;

EnemiesParent = CreateParent("Enemies", roomObject.transform);
BuffsParent = CreateParent("Buffs", roomObject.transform);
WallsParent = CreateParent("Walls", roomObject.transform);
FloorsParent = CreateParent("Floors", roomObject.transform);

RoomContext.EnemiesParent = EnemiesParent;
RoomContext.BuffsParent = BuffsParent;

var templateElements = roomTemplate.TemplateElement;
var coordinates = roomTemplate.Coordinates;

for (int i = 0; i < templateElements.GetLength(0); i++)
{
    for (int j = 0; j < templateElements.GetLength(1); j++)
    {
        TemplateElementType elementType = templateElements[i, j];
```

```
                Vector3 position = coordinates[i, j];
                Vector3 floorPosition = new Vector3(position.x, position.y - 1,
position.z);


                _roomObjectsFactory.Build(floorPosition, FloorsParent,
TemplateElementType.Ground);


                switch (elementType)
                {
                    case TemplateElementType.DefaultWall:
                        _roomObjectsFactory.Build(position, WallsParent,
TemplateElementType.DefaultWall);
                        break;
                    case TemplateElementType.Chest:
                        IChestController chestController =
_chestFactory.CreateChest(position, WallsParent);
                        RoomContext.Chests.Add(chestController);
                        break;
                }
            }
        }

        BuildExits(templateElements, coordinates);
        CenterCamera(roomTemplate);
        await Task.Delay(50);
        OnNavigationCreate(roomObject.transform);
```

```
        return roomObject;
    }

    private void AnalyzeTemplate(RoomTemplateSO.Template
roomTemplate)
    {
        if (roomTemplate == null)
        {
            Debug.LogError("Template not found");
            return;
        }

        var templateElements = roomTemplate.TemplateElement;
        var coordinates = roomTemplate.Coordinates;

        for (int i = 0; i < templateElements.GetLength(0); i++)
        {
            for (int j = 0; j < templateElements.GetLength(1); j++)
            {
                TemplateElementType elementType = templateElements[i, j];
                Vector3 position = coordinates[i, j];

                switch (elementType)
                {
```

```
                    case TemplateElementType.Player:
                        RoomContext.PlayerSpawnPoints.Add(new
PlayerSpawnPointWithType { SpawnPoint = position, Type =
CharacterType.Warrior });
                        break;

                    case TemplateElementType.Barbarian:
                    case TemplateElementType.Summoner:
                    case TemplateElementType.Thrower:
                        RoomContext.EnemySpawnPoints.Add(new
EnemySpawnPointWithType { SpawnPoint = position, Type =
(CharacterType)elementType });
                        break;

                    case TemplateElementType.RandomBuff:
                        RoomContext.BuffSpawnPoints.Add(new
BuffSpawnPointWithType { SpawnPoint = position, Type =
(BuffType)elementType });
                        break;
                }
            }
        }
    }

    private void BuildExits(TemplateElementType[,] templateElements,
Vector3[,] coordinates)
    {
        int rows = templateElements.GetLength(0);
```

```
            int cols = templateElements.GetLength(1);
            List<Exit> exits = new List<Exit>();

            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    if (templateElements[i, j] == TemplateElementType.Exit)
                    {
                        if (j + 2 < cols &&
                            templateElements[i, j + 1] == TemplateElementType.Exit
&&
                            templateElements[i, j + 2] == TemplateElementType.Exit)
                        {
                            Vector3 centerPos = coordinates[i, j + 1];
                            GameObject exit = _roomObjectsFactory.Build(centerPos,
WallsParent, TemplateElementType.Exit);
                            exit.transform.rotation = Quaternion.Euler(0, 90, 0);

                            Exit exitComponent = exit.GetComponent<Exit>();
                            exitComponent.Transform = exit.transform;
                            exits.Add(exitComponent);

                            ICompleatedRoomTrigger trigger =
exit.GetComponent<ICompleatedRoomTrigger>();
                            RoomContext.CompleatedRoomTriggers.Add(trigger);
```

```
                templateElements[i, j] = TemplateElementType.None;
                templateElements[i, j + 2] = TemplateElementType.None;
            }
            else if (i + 2 < rows &&
                templateElements[i + 1, j] == TemplateElementType.Exit
&&
                templateElements[i + 2, j] == TemplateElementType.Exit)
            {
                Vector3 centerPos = coordinates[i + 1, j];
                GameObject exit = _roomObjectsFactory.Build(centerPos,
WallsParent, TemplateElementType.Exit);

                Exit exitComponent = exit.GetComponent<Exit>();
                exitComponent.Transform = exit.transform;
                exits.Add(exitComponent);

                ICompleatedRoomTrigger trigger =
exit.GetComponent<ICompleatedRoomTrigger>();
                RoomContext.CompleatedRoomTriggers.Add(trigger);

                templateElements[i, j] = TemplateElementType.None;
                templateElements[i + 2, j] = TemplateElementType.None;
            }
        }
    }
```

```
        }
    }

    private void CenterCamera(RoomTemplateSO.Template roomTemplate)
    {
        if (roomTemplate == null)
        {
            Debug.LogError("Template not found");
            return;
        }

        var coordinates = roomTemplate.Coordinates;
        int rows = coordinates.GetLength(0);
        int cols = coordinates.GetLength(1);

        Vector3 bottomLeft = coordinates[0, 0];
        Vector3 topRight = coordinates[rows - 1, cols - 1];
        Vector3 center = (bottomLeft + topRight) / 2;

        Camera.main.transform.position = new Vector3(center.x,
Camera.main.transform.position.y, center.z);
        Camera.main.transform.LookAt(new Vector3(center.x, 0, center.z));
    }

    private Transform CreateParent(string name, Transform parent)
```

```
        {
            GameObject parentObject = new GameObject(name);
            parentObject.transform.SetParent(parent);
            parentObject.transform.localPosition = Vector3.zero;
            parentObject.transform.localRotation = Quaternion.identity;
            return parentObject.transform;
        }

        public void OnNavigationCreate(Transform parent)
        {

            GameObject navMeshObject = new GameObject("NavMeshSurface");
            navMeshObject.transform.SetParent(parent);
            navMeshObject.transform.localPosition = Vector3.zero;

            NavMeshSurface navMeshSurface =
navMeshObject.AddComponent<NavMeshSurface>();
            RoomContext.NavMeshSurface = navMeshSurface;
            navMeshSurface.BuildNavMesh();
        }
    }
```

ДОДАТОК Е

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Методи створення кросплатформенних казуальних ігрових застосунків

Тип роботи: магістерська кваліфікаційна робота
(МКР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності
Turnitin

Оригінальність 97% Схожість 3%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Turnitin щодо роботи.

Автор роботи _____ Крейчі В.Б.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Савицька Л.А.
(підпис) (прізвище, ініціали)