Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення телекомунікацій та електронних систем
(назва відділення)

Циклова комісія комп'ютерної інженерії
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітній ступінь)

на тему: **Розробка комп'ютерної гри «SIN» з використанням C#**

Виконав: студент VII курсу, групи КІб-703

Спеціальності **123 Комп'ютерна інженерія**
(шифр і назва спеціальності)

_____ Ілля ЛЕНЧУК
(ім'я та прізвище)

Керівник Ігор ГЕНИК
(ім'я та прізвище)

Рецензент _____
(ім'я та прізвище)

Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
імені ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерної інженерії
Освітній ступінь бакалавр
Освітньо-професійна програма: Комп'ютерна інженерія
Спеціальність: 123 Комп'ютерна інженерія
Галузь знань: 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"18" травня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ленчук Ілля Михайлович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи **Розробка комп'ютерної гри «SIN» з використанням C#.**

керівник роботи Геник Ігор Степанович
(прізвище, ім'я, по батькові)

затверджені наказом Відокремленого структурного підрозділу «Тернопільський фаховий коледж Тернопільського національного технічного університетут імені Івана Пулюя» від 18.05.2026р №4/9-252.

2. Строк подання студентом роботи: 22 червня 2026 року.

3. Вихідні дані до роботи: завдання на проектування, інструкції з використання середовища C# та рушію Unity.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Загальний розділ. Розробка технічного та робочого проєкту. Спеціальний розділ. Економічний розділ. Безпека життєдіяльності, основи охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): UML-діаграма класів програми, UML-блок-схема алгоритму методу, UML-діаграма послідовності дій, UML-діаграма варіантів використання програми, UML-діаграма файлової структури програми, текст програми, таблиця техніко-економічних показників.

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Ірина ЛЕЩИК Методист, викладач		
Безпека життєдіяльності, основи охорони праці	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	18.05	
2	Збір і узагальнення інформації	21.05	
3	Написання першого розділу	25.05	
4	Розробка технічного та робочого проекту	1.06	
5	Написання спеціального розділу	8.06	
6	Розрахунок економічної частини	11.06	
7	Написання розділу охорони праці	12.06	
8	Виконання графічної частини	14.06	
9	Оформлення проекту	18.06	
10	Погодження нормоконтролю	19.06	
11	Попередній захист роботи	22.06	
12	Захист кваліфікаційної роботи	**24.06**	

7. Дата видачі завдання: 18 травня 2026 року

Студент _____ Ілля ЛЕНЧУК
(підпис) (ім'я та прізвище)

Керівник роботи _____ Ігор ГЕНИК
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

Ленчук І. М.: Розробка комп’ютерної гри «SiN» з використанням C#: кваліфікаційна робота на здобуття освітнього ступеня бакалавр, за спеціальністю 123 Комп’ютерна інженерія. Тернопіль: ВСП «ТФК ТНТУ», 2026. 102с.

Дослідження присвячене розробці комп’ютерної гри «SiN» із використанням ігрового рушія Unity та мови програмування C#. У роботі розглядаються основні етапи створення ігрового застосунку — від проєктування архітектури до реалізації функціональних можливостей і тестування готового продукту.

У ході дослідження проаналізовано сучасні інструменти для розробки ігор та визначено переваги використання Unity як універсальної платформи для створення кросплатформних проєктів. На основі проведеного аналізу було розроблено гру «SiN», у якій реалізовано ключові ігрові механіки, систему керування персонажами, взаємодію з об’єктами середовища та інтерфейс користувача.

Для реалізації логіки гри використано об’єктно-орієнтований підхід, який дозволяє ефективно організовувати взаємодію між окремими компонентами системи. Також розроблено механізми обробки подій, збереження та передачі даних між ігровими модулями.

У результаті виконаної роботи створено працездатний програмний продукт, який демонструє можливості Unity та C# для розробки сучасних комп’ютерних ігор. Проведене тестування підтвердило коректність роботи реалізованих механік, стабільність функціонування системи та прийнятний рівень продуктивності на цільових платформах. Отримані результати можуть бути використані як основа для подальшого вдосконалення гри та розширення її функціональних можливостей.

Ключові слова: комп’ютерна гра, Unity, C#, розробка ігор, ігрові механіки, програмна архітектура, користувацький інтерфейс, об’єктно-орієнтоване програмування, тестування.

ANNOTATION

Illia LENCHUK. Graduation Thesis on Topic Development of the computer game "SiN" using the C# programming language: qualification work for obtaining a Bachelor's degree in Computer Engineering. Ternopil: SSS «TPC TNTU», 2026. 102 p.

The research is dedicated to the development of the computer game "SiN" using the Unity game engine and the C# programming language. The thesis examines the main stages of creating a game application — from architectural design to the implementation of functional features and testing of the finished product.

In the course of the study, modern game development tools were analyzed, and the advantages of using Unity as a versatile platform for creating cross-platform projects were determined. Based on the conducted analysis, the game "SiN" was developed, implementing key game mechanics, a characters control system, interaction with environment objects, and a user interface.

To implement the game logic, an object-oriented approach was used, which allows for the efficient organization of interaction between individual system components. Mechanisms for event handling, data storage, and data transfer between game modules were also developed.

As a result of the completed work, a functional software product was created, demonstrating the capabilities of Unity and C# for modern computer game development. The conducted testing confirmed the correct operation of the implemented mechanics, the stability of the system's functioning, and an acceptable level of performance on the target platforms. The obtained results can be used as a basis for further improvement of the game and the expansion of its functional capabilities.

Keywords: computer game, Unity, C#, game development, game mechanics, software architecture, user interface, object-oriented programming, testing.

ЗМІСТ

ВСТУП 8
1 ЗАГАЛЬНИЙ РОЗДІЛ 9
1.1 Аналітичний огляд існуючих рішень 9
1.2 Технічне завдання 12
1.2.1 Найменування та область застосування 12
1.2.2 Призначення розробки 13
1.2.3 Вимоги до програмного забезпечення 13
1.2.4 Вимоги до програмної документації 17
1.2.5 Техніко-економічні показники 17
1.2.6 Стадії та етапи розробки 18
1.2.7 Порядок контролю та прийому 19
2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ 20
2.1 Розробка загальної структури і варіантів використання програми 20
2.2 Розробка системи класів 23
2.3 Розробка методів 34
2.4 Проектування і опис інтерфейсу користувача 39
2.5 Опис файлової структури програми 43
2.6 Тестування програми і результати її виконання 44
3 СПЕЦІАЛЬНИЙ РОЗДІЛ 48
3.1 Інструкція з інсталяції програмного забезпечення 48
3.2 Інструкція з використання тестових наборів 48
3.3 Інструкція з експлуатації програмного комплексу 48
4 ЕКОНОМІЧНИЙ РОЗДІЛ 50
4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР 50

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи 51
4.3 Розрахунок матеріальних витрат 53
4.4 Розрахунок витрати на електроенергію 55
4.5 Визначення транспортних затрат 55
4.6 Розрахунок суми амортизаційних відрахувань 56
4.7 Обчислення накладних витрат 56
4.8 Складання кошторису витрат та визначення собівартості НДР 57
4.9 Розрахунок ціни НДР 57
4.10 Визначення економічної ефективності і терміну окупності капітальних вкладень 58
5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 60
5.1 Розрахунок системи штучного освітлення для приміщення, де здійснюється розробка комп'ютерної гри «SiN» 60
5.2 Ергономічні проблеми безпеки життєдіяльності 62
ВИСНОВКИ 65
ПЕРЕЛІК ПОСИЛАНЬ 66
ДОДАТКИ 67
Додаток А. Текст програми 67

ВСТУП

Індустрія комп’ютерних ігор є однією з найбільш динамічних галузей сучасних інформаційних технологій. Постійний розвиток програмних засобів, графічних технологій та ігрових рушіїв сприяє створенню якісних ігрових продуктів різних жанрів. Одним із популярних жанрів стратегічних ігор є Tower Defense, основна ідея якого полягає в захисті визначеної території від хвиль противників шляхом розміщення оборонних споруд та керування бойовими одиницями.

Актуальність теми полягає в необхідності дослідження сучасних підходів до розробки ігрових застосунків із використанням об’єктно-орієнтованого програмування та засобів мови C#. Розробка власної гри дозволяє на практиці застосувати принципи проєктування програмного забезпечення, реалізувати ігрову логіку, механізми взаємодії об’єктів, систему керування ресурсами та штучний інтелект ігрових персонажів.

Предметом дослідження є методи та засоби створення гри «SiN» із використанням мови програмування C#, механізмів керування ігровими об’єктами та системи бойових одиниць.

Метою кваліфікаційної роботи є розробка гри «SiN» у жанрі Tower Defense із застосуванням мови програмування C#, у якій реалізовано систему створення та керування бойовими одиницями, механізми атаки противників, проходження хвиль ворогів, а також взаємодію між усіма ігровими елементами.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Одним з найбільш популярних під-жанрів ігор жанру «Стратегія» є «Tower Defense». Його основними визначними характеристиками є:

1. Можливість гравцем розміщувати вежі на ігровому полі.
2. Вежі атакують ворогів у своєму полі зору.
3. Переміщення ворогів певними я вними маршрутами та, у разі успішного досягання кінцевої цілі, нанесення гравцю поразки.

Окрім цього, розробники також розробляють додаткові можливості та механіки для гравця. Наприклад, гравець може покращувати вежі, а вежі можуть бути навіть не вежами, а й персонажами; гравець може сам переміщуватися полем та завдавати шкоди ворогам; гравець може розміщувати додаткові перешкоди на шляху ворогів тощо.

Хоч основна механіка і є простою для розуміння, що є однією з причин популярності жанру, ігровий процес залишається цікавим, оскільки вимагає від гравця постійного контролю ситуації та продумування розміщення веж наперед.

Якщо спрощувати жанри відеоігор до найпростіших визначних характеристик та їхньої реалізації, то «Tower Defense» ігри будуть одними з тих, що можуть кардинально відрізнятися між собою. Наприклад, якщо візьмемо жанр «Платформер», то основними рисами є можливість гравця переміщуватися по рівнях за допомогою об’єктів взаємодії та можливість завдавати шкоди ворогам, якщо вони, звсіно, є. Якщо почати надвати їм форму, то необхідно визначитися лише з тим, чи це буде 2D-, чи 3D-гра, і, найімовірніше це буде перший варіант. Усе інше здебілшого залишається незмінним від гри до гри.

Якщо почати робити це з «Tower Defense» грою, то ми отримаємо, що потрібно визначити, за рахунок чого гравець буде розміщувати вежі, яку

форму буде мати ігрове поле – Grid-based, spline тощо; яким буде поле зору у веж – просто радіус навколо, певний напрямок, форма тощо; якими маршрутами переміщуються вороги, визначеними наперед, згенерованими, маршрут через тайли, через spline тощо. Цей жанр надає великий простір для розмаху думок, навіть не враховуючи додаткові механіки.

Для підтвердження сказаного, є наступні існуючі рішення.

«Plants vs Zombies» – «Tower Defense» гра розроблена та випущена американською компанією PopCap Games у 2009 році. На рисунку 1.1 зображено обкладинку гри «Plants vs Zombies».

Рисунок 1.1 – Обкладинка гри «Plants vs Zombies»

У грі, гравець є власником саду, на який наступають орди зомбі. Користувач може розміщувати вежі, які тут представлені рослинами, для захисту свого саду. Гра є одніє з найпопулярніших у жанрі «Tower Defense» й отримала декілька нагород, зокрема «Стратегія року».

Ігрове поле являє собою набір клітинок, які можна використовувати для розміщення «веж». Гравець може розміщувати рослини за рахунок очок-сонць, які падають з верхньої частини екрану та зі спеціальних рослин – соняшників. Вороги наступають спарва наліво хвилями та намагають досягнути будинку. На рисунку 1.2 зображено ігрове поле гри «Plants vs Zombies».

Рисунок 1.2 – Ігрове поле гри «Plants vs Zombies»

Гра має наступні переваги:

– великий ростер «веж» зі своїми особливостями;
– різні типи полів;
– приємний аудіо-супровід;
– різні типи ворогів;
– велика кількість додаткових режимів та міні-ігор.

Не зважаючи на те, що грі вже більше десяти років, у неї все ще приємно грати та знаходити моменти, які не було помічені раніше.

«Arknights» – мобільна гра жанру «Tower Defense» з ґача-елементами. Розроблена китайською студією «Hypergryph» в 2019 році. Розповсюджується безкоштовно, але має у собі мікротранзакції, які дають можливість отримувати нових оперативників зі спеціальних паків. Варто відзначити, що цього можна досягти й без вкладання у гру грошей – просто граючи в неї.

Ігрове поле являє собою набір клітинок, які гравець може використовувати для розміщення «веж». Кожен оперативник може бути розміщений тільки на відповідних для нього тайлах, наприклад на дорозі чи височині. Вороги наступають з певних позначених точок та прямують заданими маршрутам. Гравець програє, коли втрачає певну кількість очок, які нараховують, коли ворог закінчує свій маршрут.

На рисунку 1.3 зображено ігрове поле гри «Arknights».

Рисунок 1.3 – Ігрове поле гри «Arknights»

Переваги існуючого рішення:

– велика кількість оперативників, з унікальними історіями, параметрами та можливостями;

– велика кількість наявних рівнів;

– цікавий сюжет;

– оновлення, які все ще виходять.

Недоліки існуючого рішення:

– монетизація, однак вона не є такою нав’язливою і можна комфортно грати й без неї.

Беручи до уваги наявні існуючі на ринку рішення, можна розпочинати створення технічного завдання, яке буде містити: опис, вимоги до програмного продукту та стадії розробки.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Наступний документ відноситься до розробки комп’ютерної гри «SiN» за допомогою мови C#.

Область застосування: загальне використання на персональних цифрових пристроях, які мають можливість для підтримки необхідних компонентів.

1.2.2 Призначення розробки

Мета розробки: створення демо-варіанту «Tower Defense» гри.

Призначення: для персонального користування.

Засоби розробки відеогри:

- 3D моделі – Blender;
- текстури – Blender та CSP;
- редактор коду – VS Code;
- ігровий рушій – Unity.

1.2.3 Вимоги до програмного забезпечення

Кінцевий результат розробки – відеогра, яка задовільняє такі вимоги:

1. Головне меню:

1.1. Наявність кнопки «Select Stage», яка, при виборі, переносить на панель вибору набору рівнів гри.

1.2. Наявність кнопки «About», яка, при виборі, відображає інформацію про розробника.

1.3. Наявність кнопки «Exit», яка припиняє роботу гри, при виборі.

2. Вибір набору рівнів:

2.1. Наявність кнопки «Back», яка переносить назад до головного меню.

2.2. Кнопка для вибору набору рівнів, яка переносить на панель вибору рівня.

3. Вибір рівня:

3.1. Наявність кнопки «Back», яка переносить назад на панель вибору набору рівнів.

3.2. Кнопки, які відображають кількість рівнів, їхні назви та при виборі яких – переносять на цей рівень.

4. Рівень:

4.1. Лічильник для відображення скільки гравець втратив очок та максимальне значення, яке означає поразку.

4.2. Лічильник, який відображає кількість наявних очок для розміщення веж.

4.3. Лічильник для відображення максимальної кількссості веж, які гравець може розмістити.

4.4. У нижній частині екрану мають знаходитися наявні для розміщення вежі.

4.5. Натиснення кнопки «ESC» має викликати меню паузи та призупиняти гру.

4.6. Меню паузи має містити кнопки для продовження гри та для виходу до головного меню.

5. Можливості гравця на рівні:

5.1. Гравець має мати змогу, при виборі вежі, розміщувати її на тайлах рівня.

5.2. Кожна вежа має мати свої умови для розміщення.

5.3. При виборі вежі для розміщення, мають позначатися доступні тайли для розміщення.

5.4. Якщо гравець натискає «Права кнопка миші» – має припинятися розміщення вежі.

5.5. Фіксування вежі на тайлі має відбуватися за рахунок натиснутої «Ліва кнопка миші».

5.6. При розміщеній вежі, гравець має мати змогу вибору напрямку, в якому вежа буде атакувати ворогів, за рахунок позиціонування миші по основним оссям та підтвердження вибору за рахунок «Ліва кнопка миші».

5.7. При закінченні розміщенні вежі, мають зніматися очки розміщення.

5.8. При виборі активної на полі вежі, має появлятися вікно з вибором знищення вежі.

6. Вороги:

6.1. Вороги мають появлятися на відповідних для цього тайлах.

6.2. Вороги мають переміщуватися по створеними для них маршрутах.

6.3. При успішному закінченні ворогом маршруту, мають додаватися очки поразки.

6.4. При успішному знищенні ворога, мають додаватися очки для розміщення веж.

7. Вежі:

7.1. Вежі мають мати свої умови для розміщення.

7.2. Вежі мають мати свої умови для атаки.

7.3. Вежі мають мати свої параметри.

7.4. Мати свою зону покриття атаки.

7.5. При розміщенні вежі має позначатися зона атаки, яку буде покривати вежа.

8. Інструменти для створення рівнів:

8.1. Має бути спеціальна сцена виключно в редакторі Unity, яка відіграє роль створення рівнів.

8.2. Має бути присутнім об’єкт для генерації поля, який:

– містить загальні параметри рівня: ширина, висота та стандартний тайл;

– кожен тайл має мати можливість налаштувати його параметри;

– кожен тайл має мати можливість взяти параметри з вже існуючого тайла;

– об’єкт створення рівня має мати можливість оновити візуал усіх тайлів за раз.

8.3. Об’єкт для створення шляхів для ворогів:

– має мати можливість створення повноцінних шляхів за рахунок контрольних точок;

– мають прийматися тільки контрольні точки які розміщенні по горизонталі або діагоналі;

– можливість візуально відобразити створенні шляхи.

8.4. Об’єкт для створення хвиль ворогів:

– можливість задати об’єкт ворога;

– мати можливість встановити інтервал між появою ворогів;

– можливість задати умову закінчення хвилі та початку наступної.

8.5. Об’єкт для збереження рівня:

– має зберігати результат роботи усіх минулих об’єктів розробки;

– має зберігати нові тайли, які не були створенні до того;

– можливість редагування параметрів рівня;

– можливість задання початкових значень лічильників (початкове значення очків розміщення веж, максимальна кількість втрачених очок тощо).

9. Розповсюдження та сумісність:

9.1. Гра має розповсюджуватися у вигляді інсталятора.

9.2. Сумісність:

– операційна система: Windows 10;

– процесор : 2-х ядерний з частотою 3.6 GHz;

– оперативна пам’ять: 2 GB;

– накопичувач: 1 GB;

– відеокарта: інтегрована.

– засоби вводу: клавіатура та комп’ютерна миша.

1.2.4 Вимоги до програмної документації

Документація зобов’язана супроводжуватися документами, які описують роботу програмного забезпечення та умови експлуатації. Повинна містити ключові аспекти коректного використання ПЗ.

Документація складається з:

– пояснювальної записки;

– технічного завдання;

– опису програми;

– кодів програми;

– тестування.

Документація повинна задоівльняти такі вимоги:

– відповідність ДСТУ;

– відсутність подвійних трактувань;

1.2.5 Техніко-економічні показники

Для коректної та комфортнох роботи у редакторах, які було згадано раніше, обрано робочу машину з такими параметрами

– операційна система: Windows 10;

– процесор: Ryzen 5 7600X 4.70 GHz;

– оперативна пам’ять: 32 GB;

– відеокарта: RTX 5060;

Наявність великої кількості оперативної пам’ята забезпечує можливість працювати одночасно в усіх заявлених програмах без втрати продуктивності як ПК так і розробника, що є критичним фактором для успішного виконання роботи.

Вигляд інтерфейсу ПЗ: графічний;

Мова інтерфейсу: англійська.

1.2.6 Стадії та етапи розробки

У розробці наявні наступні етапи:

1. Описова частина:

1.1. Аналітичний огляд існуючих на ринку рішень.

1.2. Аналіз та формування вимог, які має задовільняти програмне забезпечення.

1.3. Формування технічного завдання.

1.4. Концептуалізація ігрових механік та гри.

1.5. Вибір інструментів реалізації програмного забезпечення.

2. Розробка:

2.1. Створення UML-діаграм програмного забезпечення.

2.2. Написання коду.

2.3. Створення інтерфейсу взаємодії гравця.

2.4. Розробка 3D моделей персонажів.

2.5. Експортування ассетів до рушія.

2.6. Виправлення помилок, які виникають при експортуванні та використанні ассетів.

2.7. Оптимізація ассетів та гри.

2.8. Тестування готової гри.

2.9. Повторна перевірка дотримання вимог.

2.10. Збірка.

3. Фінальна підготовка:

3.1. Перевірка дотримання усіх вимог.

3.2. Написання документації.

3.3. Створення інсталяційного файлу.

3.4. Здача в експлуатацію.

1.2.7 Порядок контролю та прийому

Більшість класів у Unity наслідуються від MonoBehavior, класи яких не можна створювати через звичайні конструктори, проте їх можна «вішати» на об’єкти, які розташовуються на сцені і можуть напряму взаємодіяти з ігровим світом.

Зважаючи на фактори, було обрано метод тестування «чорної скриньки». Даний метод цілком покриває потреби тестування невеликих ігор та тестування інтерфейсу.

Під час тестування усі виявленні помилки та неточності занотовуються та описуються для подальшого виправлення розробником. Після виправлення виявлених помилок гра відправляється на повторне тестування. Цей цикл повторюється доти, доки не виправляться усі помилки.

Після того як усі помилки успішно виправлено, гра вважається готовою, та можна розпочинати створення документації та інсталятора.

Документація повинна містити усю інформацію, яка може бути необхідною для використання програмного забезпечення та відповідати усім вимогам, які було згадано раніше.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка загальної структури і варіантів використання програми

Структуру програми можна розділити на такі основні частини: частина гравця та частина розробника. Вони визначають, який функціонал містять та з чим може взаємодіти користувач та девелопер.

Частина розробника.

Ця частина містить до таких елементів:

– створення нових персонажів за рахунок наявних компонентів, створених розробниками;

– створення нових тайлів, які будуть використанні на рівнях;

– генерація поля;

– розмітка шляхів для ворогів;

– налаштування хвиль ворогів;

– створення загального рівня.

Розробник створює нових персонажів, як неігрових так і веж, за рахунок наявних компонентів. Вони мають основний скрипт, який реалізує необхідні для конкретного персонажа інтерфейси. Наприклад, якщо необхідно забезпечити переміщення по ігровому полю, то для цього є «MoveableUnit» клас, який реалзіує інтерфейс IMoveable. Саме ж переміщення відбувається за допомогою стратегій, які й містять основну логіку. Для активації логіки присутні умови в класі «MoveCondition», який наслідується від «StrategyCondition» для забезпечення поліморфізму. Один персонаж може мати декілька стратегій поведінки та змінювати її залежно від ситуації, що забезпечує можливість створювати різних акторів не лише за виглядом.

Створення тайлів має два компоненти: візуальну частину та частину з інформаціює. Перша зберігається на диск та в реєстр, оскільки вона може перевикористовуватися між рівнями. Друга частина зберігається в класі рівня.

Такий розподіл дозволяє швидко редагувати різні частини, не порушуючи загальну цілісність. Візуальну частину можна створити або напряму в директорії або під час генерації та налаштуванні рівня, в такому випадку об’єкт збереження сам знайде та збереже нові візуальні тайли. Частина з даними містить загальну інформацію про тайл, його теги тощо.

Генерацією поля займається спеціальний об’єкт зі скриптом, все що необхідно це ввести висоту та ширину, а код сам розставить тайли на свої місця. Під час створення поле заповнюється вказаним тайлом за замовчуванням, який можна замінити. Після цього, візуальна частина та дані кожного тайл налаштовуються вручну або попередньо створеними тайлами.

Для створення маршрутів ворогів, обов’язково має бути присутнім згенероване поле. Вибираючи ключові точки, які розміщенні стосовно один-до-одного по горизонталі або по діагоналі. Додатково можна візуалізувати створенні маршрути, для підтвердження правильності побудови і подальшої корекції.

Після успішного створення маршрутів, можна приступати до розробки хвиль ворогів. Кожній хвилі можна налаштувати ворогів, які були створенні раніше, умови початку, затримка між появою недругів, затримка між хвилями, умови закінчення хвилі, використанні маршрути тощо. Приклади умов початку нової хвилі: затримка, усі вороги минулої хвилі було знищено, усії недругів минулої хвилі було випущено. Кожен ворог у хвилі має свій маршрут та під час однієї хвилі різні вороги можуть прямувати різними маршрутами.

Після того, як усі минулі підсистеми було створено та налаштовано, можна зберігати загальний рівень для подальшого використання. Він практично не містить жорстких референсів на об’єкти, а лише їхні назви, які використовує під час ініціалізації, знаходячи та завантажуючи їх з реєстру. По своїй суті це просто список даних, які можна редагувати в любий момент, що полегшує виправлення помилок або конвертацію у інший формат, як наприклад json.

На рисунку 2.1 зображено приклад фінального вигляду даних рівня.

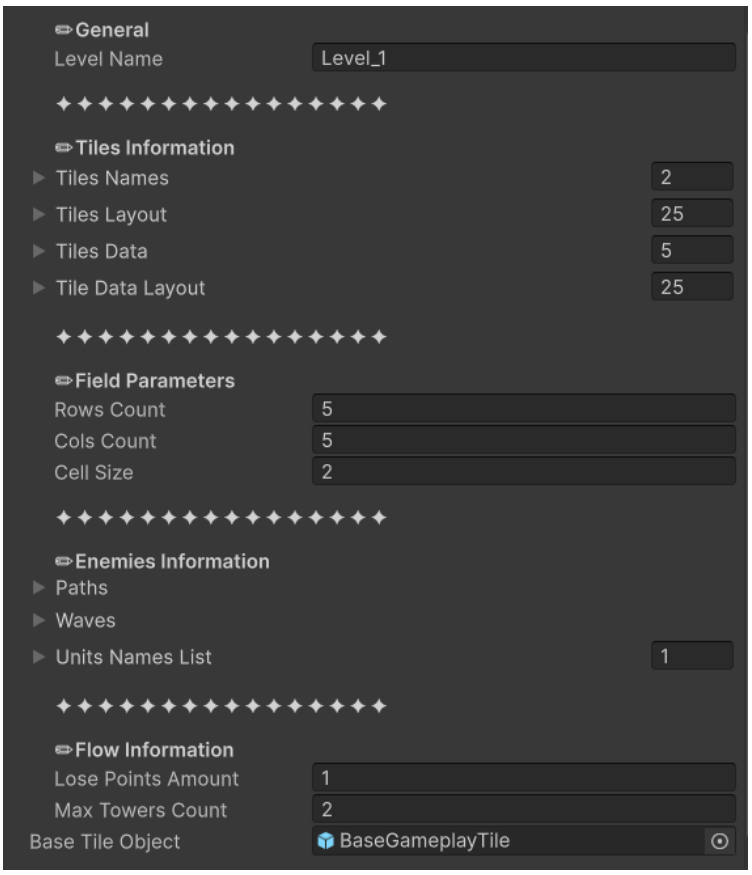


Рисунок 2.1 – Приклад фінального вигляду даних рівня

Назви полів зі стрілкою, містять розширену підінформацію про них. На рисунку 2.2 зображено розширену інформацію про тайли та їхню розміщення на рівні.

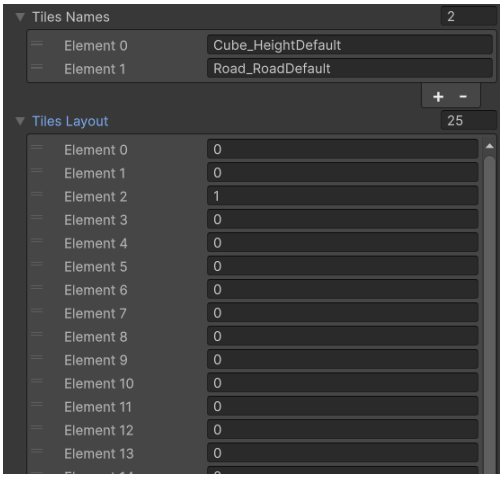


Рисунок 2.2 – Розширена інформація про тайли та їхнє розміщення на рівні

Частина гравця.

Частину гравця можна розділити на дві підчастини:

1. Головне меню та інші панелі взаємодії.
2. Ігровий процес.

Головне меню містить набір кнопок; «Select Stage», «About», «Exit», які переносять користувача між панелями та за допомогою яких можна розпочати основний ігровий цикл.

Враховуючи ігровий процес, гравець має наступні варіанти використання програми:

- запустити гру;
- переглянути панель «Про розробника»;
- вибрати набоір рівнів;
- вибрати рівень;
- розміщувати вежі;
- забирати вежі;
- призупиняти гру;
- закрити гру.

Результатом цього підрозділу є графічне представлення можливих варіантів взаємодії з програмою, наведене у додатку «Варіанти використоння програми.»

2.2 Розробка системи класів

Глобально класи можна розділити на такі групи:

1. Використовуються в редакторі.
2. Задіяні в ігровому процесі.

Класи, що використовуються в редакторі.

LevelMaker – клас для генерації та налаштування ігрового поля. Містить такі поля: поле для ширина поля, поле для висоти поля, поле для розміру тайлів, поле для поточної ширини поля, поле для поточної висоти поля, поле

для поточного розміру тайлів, посилання на зберігач, список створених тайлів та об’єкт тайлу за замовчуванням. Методи: CreateField – метод створення поля, TilesCreator – метод створення нових тайлів, IsTheSameTileSize – метод перевірки змінни розміру тайлів, IsTheSameRowsAndColumns – метод перевірки зміни кількості ширини та висоти, DeleteOldTiles – метод для видалення раніше створених тайлів, UpdateAllTiles – метод ононвлення даних призначених на тайлах, SendFieldToSaver – метод відправки поля до зберігача для подальшої роботи.

PathsMaker – метод створення маршрутів, які будуть використовуватися ворогами для переміщення. Поля: список усіх створених шляхів, список контрольних точок шляхів, посилання на об’єкт, який зберігає візуалізовані шляхи та посилання на зберігач. Методи: CalcAngle – розраховує та повертає кут між двома контрольними точками, CreateNewPaths – створює нові шляхи за контрольними точками, ChooseAndSavePath – вирішує, як створювати шлях від однієї точки до іншої та зберігає проміжні тайли, SaveStraightPath – зберігає горизонтальні та вртекальні шляхи, SaveDiagonalPath – зберігає діагональні шляхи, ClearPoints – очищує контрольні точки, ClearPaths – видаляє створенні шляхи, SendPathsToSaver – надсилає створенні шляхи до зберігача, DrawAllPaths – запускає створення візуального відображення усіх шляхів, DrawPath – «малює» маршрут, SetupLineRenderer – ініціалізує на налаштовує компонент, який відображає лінії шляхів.

SiN.Paths – клас-контейнер шляхів. Містить список елементів типу SiN.Path. Методи: AddNewPath – додає новий шлях до списку, ClearPaths – очищує список маршрутів, Length – повертає кількість елементів списку, PathsList – повертає усі маршрути списку, перегружений оператор [], для отримання маршруту за індексом.

SiN.Path – клас, який описує один маршрут. Поля: ім’я шляху, список точок маршруту SiN.Point. Методи: PathName – для отримання назви маршруту, GetFirst – повертає першу точку, GetLast – повертає останню точку,

SetPath – для ініціалізації від іншого маршруту, AddPoint – для додання точки, Length – для отримання довжини маршруту, перегружений оператор [].

WavesCreator – клас для створення та налаштування хвиль ворогів. Поля: список маршрутів, список точок появи ворогів, список тайлів з точками появи, об’єкт, який відображає точку появи, список усіх хвиль ворогів та посилання на зберігач. Методи: GetPaths – пробує взяти шляхи з зберігача, CreateSpawnpoints – створює візуальне представлення точок появи ворогів, DestroyOldSpawnpoints – знищує старі точки появи ворогів, SaveAll – запускає збереження створених хвиль, SaveWaves – надсилає зберігачу дані про хвилі, SaveEnemies – надсилає зберігачу імена ворогів, які задіяні в хвилях, GetUnitsNames – повертає імена усіх ворогів.

EditorWaves – список хвиль ворогів. Містить список елементів типу EditorWave, які описують хвилі. Методи для отримання довжини списку та перегружений оператор [].

EditorWave – описує хвилю ворогів, наслідується від BaseWave. Містить список елементів типу EditorWaveStep та методи для отримання довжини списку та перегружений оператор [].

BaseWave – містить поля: поле-умова закінчення хвилі – enum SiN.EndWaveCondition та затримка після хвилі.

EditorWaveStep – клас для одного елементу в хвилі, наслідується від BaseWaveStep. Містить посилання на ворога для появи та посилання з якої позиціх появитися – клас Spawnpoint.

BaseWaveStep – базовий клас, який описує появу одного ворога. Містить поля для назви шляху та затримки перед появою ворога.

Spawnpoint – клас, який відіграє роль позначки тайла, на якому появляються вороги. Містить моле SiN.Point, для збереження індексів тайлу.

Level – клас, що описує рівень. Поля: назва рівня, список імен задіяних тайлів, список розміщення тайлів, список тегів тайлів, список розміщення тегів, ширина поля, висота поля, розмір тайлів, шляхи та хвилі ворогів, список імен задіяних ворогів, значення втрачених очок, які означають програш,

значення максимальної кількості можливих розміщених веж. Методи: BaseInitialize – ініціалізує базові параметри: висота, ширина, розмір тайлів, InitializeFromLevel – ініціалізує дані з іншого переданого класу Level, методи для отримання значень полів.

EditorTile – описує дані тайла в редакторі. Поля: інформація, яка віжлива під час ігрового процесу TileSettigns, індекс тайла на полі SiN.Point, посилання на об’єкт, який відіграє роль позиції переміщення ворогів, посилання на компоненти, які містять меш та матеріали, посилання на об’єкт з якого взяти налаштування, прапор для взяття інформації з стороннього об’єкта, прапор на копіювання мешу, прапор на копіювання матеріалів, прапор на копіювання тегів. Методи: GetDataFromObject – розпочинає взяття інформації з стороннього об’єкта, GetAndSetMaterials – копіює матеріали, GetAndSetMesh – копіює меш, GetAndSetTags – копіює теги, SetPointIndex – встановлює індекси позиції тайла на полі, GetVisualName – повертає назву тайла для збереження.

TileSettings – дані про тайл. Поля: контейнер тегів SiN.TagsContainer, позиція точки переміщення ворогів. Методи: Tags – метод для отримання тегів тайла, WaypointLocalPosition – метод для отримання локальної точки переміщення ворогів, Initialize – метод ініціалізації класу через інший EditorTile.

SiN.Point – структура, відіграє роль точки. Містить поля для індексів по x та y. Конструктор для створення, перевантаженні оператори віднімання та додавання.

SiN.TagsContainer – клас-контейнер тегів. Містить список елементів типу enum Tags. Методи: конструктор, Count – для отримання кількості елементів, AddTag для додавання тегів, HasTag – для перевірки наяності одного тегу, GetTag – повертає тег зі списку, за назвою або індексом, якщо він є, перешружений оператор [] – для отримання тегу за індексом, HasSimilarTags – перевіряє чи є хоча б один схожий тег, GetUniqueTagsOnly – повертає список

лише унікальні теги без повторювань, TheSameTags – перевіряє чи усі теги одинакові у порівнянні з переданим списком.

Field – наслідується від Level, клас представлення поля. Містить додаткове поле – список тайлів поля. Містить методи для отримання тайлів на певних індексах.

LevelDataSO – ScriptableObject, який зберігається в реєстрі. Поля: поде Level, яке містить дані про рівень, поле для тайлу за замовчуванням. Методи: SetFieldSettings – ініціалізує дані з іншого переданого рівня.

FieldViewer – використовується для роботи та модифікації переданих значень рівня та самим рівнем. Містить поле Field. Методи: ClearLayout – очищує список розміщення тайлів, ClearTilesNames – очищує список імен задіяних тайлів, ClearTileData – комбінація попередніх двох методів, ClearPaths – очищає список маршрутів, ClearLevelName – очищає назву рівня, ClearFieldSettings – очищує базові параметри поля, AddTile – додає унікальні тайли до списку, AddTilesTags – додає унікальні набори тегів тайлів до списку, HasField – перевірка, чи є з поле з яким можна працювати.

AllSaver – зберігач ассетів. Поля: інф

ормація про рівень LevelDataSO, інформація про шляхи для зберігання AllSaverContextSO, FieldViewer для роботи та модифікації даних. Методи: SaveTiles – зберігає нові унікальні тайли, які ще не зареєстровані в реєстрі, SaveFieldLevel – зберігає в реєстр рівень, з усією інформацією.

AllSaverContext – містить інформацію, куди які ассети зберігати. Містить поля типу SaverContextSO для тайлів та рівня. Також містить методи для їх отримання.

SaverContextSO – містить інформацію про місця збереження для певної групи ассетів. Поля: посилання на папку для ассетів на диску DefaultAsset, AddressableAssetGroup – посилання на групу в реєстрі, посилання на папку префабів DefaultAsset.

Класи, задіяні в ігровому процесі.

GameplayField – клас поля, який використовується під час ігрового циклу. Поля: список об’єктів тайлів на ігровому полі, список тайлів, які є височинами GameplayTile, список тайлів, які є дорогами, список об’єктів появи ворогів, список об’єктів закінчень маршрутів, ширина поля. Методи: GetTilesContainer – повертає список тайлів, GetTileAtIndex – повертає тайл за індексом, GetTileAtIndexForAttackCoverage – повертає тайл за індексом для покриття атаки, DefinePlaceableHeightRoadTiles – визначає, які тайли можуть використовуватися для розміщення веж, HighlightTiles – позначає тайли за тегом, UnhighlightTiles – забирає позначення на тайлах за тегом.

GameplayTile – клас, відіграє роль представлення ігрового тайлі на полі. Поля: контейнер тегів тайла SiN.TagsContainer, посилання на розміщенну на тайлі вежу BaseUnit, хеш-сет ворогів на тайлі, індекс тайла SiN.Point, посилання на маркер розміщення та атаки, координати для розміщення об’єкта представлення точки появи ворогів, координати об’єкта для переміщення ворогів, подія реєстрації ворога на тайлі, подія реєстрації вежі на тайлі. Методи: RegisteredEnemies – для отримання зареєстрваних ворогів, InitChildrenData – ініціалізує дані маркерів та координати деяких об’єктів, CanUnitBePlaced – повертає чи може бути розміщена вежа на тайлі, AssignTags – призначає передані теги тайлу, SetPointIndexes – призначає індекс тайла, RegisterAsTower – реєструє вежу на тайлі, RegisterAsEnemy – реєструє ворога на тайлі, HasTag – повертає чи має тайл переданий тег, HasAnyTag – повертає чи тайл має хоча б один тег з переданих, UnregisterEnemy – видаляє вежу з зареєстрованих, UnregisterEnemy – видаляє переданого ворога з зареєстрованих, HasTower – повертає наявність вежі на тайлі, GetPoint – повертає індекс тайла, GetWorldWaypointPosition – повертає координати точки переміщення у світових координатах, GetWorldEntrancePosition – повертає координати точки появи ворогів у світових координатах, HighlightSurfaceForPlacement – позначає тайл маркером для можливого розміщення вежі, HidePlacementMarker – ховає маркер для можливого

розміщення вежі, HighlightSurfaceForAttack – позначає тайл маркером атаки, CanAttackThisTile – повертає чи може вежа, або ворог, атакувати цей тайл.

BaseUnit – базовий клас, який описує вежі та ворогів. Поля: параметри юніта StatsSO, значення отриманої шкоди, список тайлів, які покриває юніт, посилання на поточну логіку поведінки стратегії поведінки BehaviorStartegy, посилання на поточну стратегію SiN.StrategyEntry, список можливих стратегій персонажа, контейнер тегів, поле, чи може юніт діяти, значення чи юніт контролюється гравцем, подія знищення юніта. Методи: Start – виконується перед першим викликом Update, Update – виконується кожен кадр, EvaluateConditions – перевіряє умови стратегій та повертає істинну, коли зустрічає відповідність умов, ChangeStrategy – змінює стратегію на нову, якщо необхідно, HadleDamage – опрацьовує отриманну шкоду, HandleDeath – опрацьовує закінчення життєвого циклу персонажа, GetUnitTags – повертає теги юніта, GetStat – повертає значення параметри юніта за тегом, GetAttackArea – повертає список покритих тайлів, CanBeAttacked – віртуальний метод, повертає чи може юніт бути атакованим, RemoveUnit – метод, для опрацювання операцій, коли юніт забирається з ігрового поля.

SiN.StrategyEntry – запис стратегії. Поля: назва поведінки – enum BehaviorsName, логіка стратегії – BehaviorStrategy, умова стратегії – StrategyCondition.

BehaviorStrategy – абстрактний клас, логіка поведінки юніта. Містить поде назви поведінки та метод DoLogic, який приймає BaseUnit.

StartegyCondition – абстрактний клас, умова початку стратегії. Містить метод CheckCondition, який приймає BaseUnit.

GameManager – клас, який контролює ігровий цикл. Поля: посилання на свій екземпляр класу, подія зміни стану гри, подія оновлення лічильників, подія початку паузи, подія, коли дані рівня завантаженні, поточний стан гри – enum GameStates, посилання на дані рівня, посилання на об’єкт ігрового поля, значення максимальної кількості можливих втрачених очків на рівні, прапор, чи останній ворог переможений, прапор паузи. Методи: LostPointsCap – метод

отримання максимальної кількості можливих втрачених очків на рівні, Field – метод для отримання посилання на клас поля GameplayField, Awake – викликається, коли об’єкт створений на сцені, Start, SetState – метод для встановлення нового стану, HandleStateChange – опрацьовує логіку при певних станах гри, HandlePauseFlow – опрацьовує цил паузи, LoadAndInitializeLevel завантажує дані рівня через LevelLoader, AddLosePoint – додає очки програшу, CheckLoseCondition – перевірка умови програшу, CheckFinishLevelConditions – перевіряє умови закінчення рівня, CloseGame – закриває гру.

LevelLoader – клас для завантаження даних про рівень. Поля: посилання на екземпляр свого класу, подія завершення завантаження даних рівня. Містить LoadLevelData – метод для завантаження даних рівня.

LevelInstantiator – метод ініціалізації рівня. Поля: посилання на екземпляр свого класу, посилання на дані про рівень, список хендлерів для асинхронних операцій, список елементів візуальної частини тайлів, посилання на об’єкт поля, подія завершення створення поля. Методи: CreateLevel – метод запуску створення рівня, InstantiateLevel – метод ініціалізації рівня, SetupTiles – метод ініціалізації тайлів, SetupTileVisual –метод ініціалізації візуальної частини тайла, GetTiles – метод отримання тайлів, CalculateStart – метод обрахунку розмщення початкового тайла, InstantiateObject – метод створення об’єкта.

PathsManager – клас для роботи з маршрутами ворогів. Поля: посилання на екземпляр свого класу, посилання на ігрове поле, список шляхів, посилання на дані про рівень, візуальне представлення точки появи ворогів, візуальне представлення точки закінчення маршруту, список точок появи ворогів, список точок закінчення маршрутів. Методи: Init – метод ініціалізації даних, SetAllPaths – метод збереження усіх шляхів, GetNextTileRef – метод для отримання наступного тайлу маршрута, GetTileByIndex – отримання тайла за індексом, DefineEntrances – створює візуальне представлення для точок появи ворогів та точок закінчення маршрутів, InstantiateEntranceExit – створює об’єкти точок появи ворогів та закінчення маршрутів.

EnemiesManager – клас для роботи з неігровими персонажами. Поля: посилання на екземпляр свого класу, подія завершення ворогом свого маршруту, подія поразки ворога, подія завершення життєвого циклу останнього ворога, посилання на спавнер хвиль ворогів – WavesSpawner, список активних ворогів, список ассетів ворогів, прапор завершення завантаження даних про необхідних ворогів, прапор перевірки стану останнього ворога. Методи: InitializeSpawner – ініціалізує спавнер хвиль ворогів, LoadAllUnitsData – метод завантаження ассетів ворогів, HandleStateChange – метод обробки зміни стану гри, HandleEnemySpawned – опрацювання, коли ворог появився, HandleEnemyDeath – опрцювання поразки ворога, HandleEnemyCompletedPath – опрацювання успіху ворога, DisableUnit – деактивація ворога, IsLastEnemyDone – перевірка чи усі вороги закінчили свій життєвий цикл.

WavesSpawner – клас опрцювання хвиль ворогів. Поля: посилання на екземпляр свого класу, посилання на поле, посилання на хвилі ворогів – Waves, індекс поточної хвилі ворогів, кількість активних ворогів, словник ассетів ворогів, подія заверешння усіх хвиль, подія появи ворога, подія закінчення ворогом маршруту. Методи: Init – метод ініціалазації класу, BeginWaves – початок хвиль, StartWaves – запускає роботу з хвилями, SpawnWave – ініціалізує хвилю ворогів, Spawnenemy – ініціалізує ворога у хвилі, WaitForWaveEnd – опрацьовує умови закінчення хвилі, HandleEnemyCompletedPath – опрацьовує закінчення врогом маршруту.

TowersManager – клас для роботи та контролю за вежами. Поля: посилання на екземпляр свого класу, список можливих веж для розміщення – TowerData, список активних веж, кількість очків для розміщення веж, максимальне значення очків для розміщення, посилання на менеджер розміщення веж – TowerPlacementManager, прапор можливості збільшення очків, прапор збільшення очків, значення максимальної кількості можливих веж, швидкість додавання очків розміщення веж, подія запит на розміщення вежі, подія оновлення лічильника. Методи: Towers – повертає наявні для

розміщення вежі, InreasePoints – метод додавання очків протягом часу, HandleStateChange – опрацювання зміни стану гри, AddTowerToList – метод додавання активнох вежі до списку, DestroyTower – метод для знищення вежі, BeginTowerPlacement – метод, який запускає розміщення вежі, AddPoint – додає одне очко розміщення вежі, UpdatePlacementPoints – оновлює очки розміщення веж, UpdateTowersCounter – оновлює кількість можливих веж для розміщення.

TowerPlacementManager – відповідає за логіку розміщення веж. Поля: посилання на екземпляр свого класу, подія, коли місце для вежі обрано, подія, коли вежа розміщена, подія відмінни розміщення вежі, подія зменшення очків розміщення веж, інформація про вежу – TowerData, позиція курсора на екрані, стан розміщення вежі – enum TowerPlacementState, цільовий тайл для розміщення вежі, напрямок розміщення – CoverageDirection, список тайлів, які вежа буде покривати своїми атаками, посилання на елемент інтерфейсу, який появляється при виборі активної вежі, посилання на вибрану активну вежу. Методи: HandleGameStateChange – опрцювання зміни ігрового стану, SetState – встановлення стану розміщення вежі, HandleEventsInvoke – опрацювання активації необїідних подій, HandleExternalEvents – опрацювання підписок на події інших класів, StartPlacingTower – розпочинає розміщення вежі, PalcingTower – опрацьовує вибір клітинки для розміщення на полі, PlaceTowerPreview – розміщує попередній вигляд вежі на клітинці, ChoosingDirection – опрацьовує вибір напрямку, в якому буде дивитсия вежа, CalcCoverage – обрахунок покритих вежею клітинок, PalceTower – фінальне розміщення вежі, CancelPalcement – відміняє розміщення вежі, ResetVisualPlacementData – забирає маркери з екрану, ChooseActiveTower – опрацьовує вибір активної на полі вежі, CancelChoosing – відміняє вибір активної вежі, RemoveTower – видаляє вибрану активну вежу, ResetChoosing – скидає дані, які стосують даних вибору активної вежі, до даних за замовчуванням, ShootRay – створює RayCast.

AreaShape – клас, який відіграє роль покриття вежею певної зони. Містить список точок, які буде покривати вежа, відосно себе. Містить метод GetShapeCoverage, який повертає точки, які треба покрити в залежності від переданого напрямку.

TowerData – клас, який зберігає інформацію про вежі. Поля: прев'ю дані вежі – TowerDataPreview, ассет вежі – UnitSO, значення ціни розміщення вежі, спрайт для кнопки.

TowerDataPreview – містить поля: візуальний вигляд вежі під час розміщення, покриття тайлів – AreaShape, контейнер тегів-умов для розміщення, контейнер тегів-умов для атаки.

UnitSO – дані про ігрову частину вежі. Містить поля: назва вежі, параметри вежі StatSO, префаб вежі.

Приклад персонажа.

StaticUnit – клас наслідується від BaseUnit. Імплементує інтерфейс ISingleTargetAttackable з методами: CanAttack, GetAttackArea, GetFirstAttackableEnemy, GetValueFor. Містить додаткові поля: посилання на тайл, на якому розміщенно, теги за якими атакує ворогів, список можливих ворогів для атаки, подія розміщення, подія знищення. Додаткові методи: Init – ініціалізує необхідні дані для вежі, HandleAddToList – опрацьовує додавання ворогів до списку, HandleRemoveFromList – опрацювання видалення ворогів зі списку, CanAttack – визначає умову для атаки, перевизначений метод RemoveUnit.

Містить стратегію поведінки AttackStrategy, з якою спілкуються через інтерфейс ISingleTargetAttackable, умова входу – AttackCondition.

AttackCondition спілкується через інтерфейс з класом. Наслідується від StrategyCondition. Має додаткове поле – посилання на інтерфейс ISingleTargetAttackable. Перевіряє CanAttack в DoLogic.

AttackStrategy наслідується від BehaviorStartegy. Додатково містить поля: посилання на інтерфейс ISingleTargetAttackable, логіку анімації атаки –

AttackLogic та прапор затримки між атаками. Містить додатковий метод Cooldown.

AttackLogic – абстрактний клас для представлення логіки роботи анімації атак. Містить поле швидкості програвання анімації та метод DoLogic.

SingleProjectileAttack – наслідується від AttackLogic та використовується в логіці атаки StaticUnit.Містить додаткові поля: посилання на інтерфейс для спілкування ISingleTargetAttackable, посилання на об’єкт снаряду, посилання на створений об’єкт снаряду, значення зміщення під час появи, прапор програвання анімації. Містить додатковий метод DoAnimation.

Результатом підрозділу є додаток «UML-діаграма класів».

2.3 Розробка методів

Логіка методів класу StaticUnit.

Метод Update, визначенний у базовому класі BaseUnit, перевіряє умови стратегій через метод EvaluateConditions та виконує на отриманій стратегії метод DoLogic.

EvaluateConditions – перевіряє чи поточна стратегія виконує свою умову, якщо так – виходить з методу. Якщо ні – проходить по усім наявним записам стратегій та перевіряє їхні умови, очікуючи, що хоча б один поверне істинну. Якщо умова справджується, призначає нову поведінку через ChangeStrategy, яка витягує з запису логіку поведінки – BehaviorStartegy.

AttackStrategy – наслідується від BehaviorStrategy, перевизначає DoLogic. Метод працює наступним чином:

– перевіряє затримку атаки, якщо все ще діє – виходить з методу;

– бере посилання на ворога метдом GetFirstAttackableEnemy, через референс на інтерфейс ISingleTargetAttackable;

– перевіряє наявність ворога;

– перевіряє наявність логіки анімації, якщо присутня, запускає її через метод DoLogic;

– викликає метод HandleDamage на референсі ворога;

– розпочинає корутину затримки між атаками.

GetFirstAttackableEnemy імплементується в класі StaticUnit та просто повертає перший елемент масиву.

HandleDamage – віртуальний метод класу BaseUnit, який додає до поля отриманної шкоди передане значення, перевіряє чи загальна отримана шкода більша-рівна здоров'ю юніта, беручи значення через GetStat, якщо так – викликає метод HandleDeath.

GetStat – метод BaseUnit, викликає GetValueFor на полі параметрів.

HandleDeath – метод baseUnit, викликає подію закінчення життєвого циклу передаючи посилання на себе. Самим знищенням об'єкта займається EnemiesManager в методі

HandleEnemyDeath видаляє ворога зі списку активних, викликає подію, що ворог не зміг виконати свою ціль, деактивує об'єкт та видаляє його зі сцени.

SingleProjectileAttack – наслідується від AttackLogic. Метод логіки анімації DoLogic:

– перевіряє чи створений об'єкт снаряду;

– перевіряє чи вже йде анімація;

– бере позицію ворога;

– переміщує об'єкт снаряду над ворогом з певнм зміщення;

– робить снаряд активним;

– починає корутину анімації;

– під час анімації снаряд швидко переміщуєть до позиції ворога.

Логіка методів класу контролю хвиль ворогів WavesSpawner.

Корутина StartWaves працює, поки індекс поточної хвилі менший за загальну кількість хвиль. Бере дані про хвилю за порядковим індексом. Починає корутину SpawnWave та передає йому дані про хвилю ворогів.

Корутина SpawnWave працює, поки не пройде усі кроки в хвилі та спавнить відповідних ворогів на кожному кроці через метод SpawnEnemy, роблячи затримки, беручи дані з хвилі ворогів.

SpawnEnemy створює та ініціалізує ворога, підписується та викликає необхідні події, встановлює дані з UnitSO. Після появи усіх ворогів повертається до StartWaves.

Після усіх операцій, запускається корутина WaitForWaveEnd, яка очікує кінця хвилі за певними умовами.

Логіка методів розміщення веж TowerPlacementManager.

В Update перевіряється поточний стан розміщення вежі. При стані idle, нічого не відбувається.

При стані PlacingTower:

– викликається метод PlacingTower;

– кастується промінь до курсора;

– перевіряється чи є клас тайлу;

– перевіряються схожість умов розміщення вежі та тегів, які містить тайл;

– відображається покриття тайлів, атаками вежі.

Якщо при цьому стані натискається «ліва кнопка миші», то викликається метод PlaceTowerPreview, який очищає непотрібну інформація та візуальні позначення та змінює стан на ChoosingDirection.

При стані ChoosingDirection:

– викликається метод ChoosingDirection;

– розраховується кут між вежею та курсором миші;

– визначається необхідний напрямок;

– в залежності від напрямку, позначаються необхідні тайли;

Якщо під час цього натиснути «ліва кнопка миші», викликається метод PlaceTower.

Метод PlaceTower:

– створює нову вежу на сцені;

– ініціалізує її;

– викликає необхідні події, які сигналізують про те, що вежа розміщена;

– знімає усі візуальні позначення та очищає необхідні дані;

– змінює стан на Idle.

Якщо під час одного з станів натиснути «права кнопка миші», викликається метод CancelPlacement. Цей метод очщає дані, знімає позначення та змінює стан на Idle.

Якщо натсинути на активну на сцені вежу, викликається метод ChooseActiveTower. Цей метод:

– сповільнює ігровий процес;

– активовує UI-елемент вибору активної вежі та переміщує її на місце вежі відносно координат екрану;

– позначає її зону покриття;

Якщо вибрати кнопку «Remove» викликається метод RemoveTower, яка викликає метод DestroyTower у TowersManager, передаючи вежу, відновлює швидкість роботи ігрового процесу, скидає усі візульні позначення та забирає UI-елемент вибраної вежі.

При виборі кнопки «Cancel», скидаються візульні позначення та забирається UI-елемент.

Логіка створення ігрвого поля через LevelMaker в редакторі.

В інспекторі скрипта вводяться дані про поле: ширина, висота, розмір тайла. При нажатті кнопки «Create Field», створюється поле. Логіка методу CreateField:

– перевіряється чи значення розміру тайла, висоти та ширини те саме, якщо так – завершує метод;

– перевіряє наявність тайла за замовчуванням, у разі відсутності, завершує метод;

– обраховує де розмістити перший тайл, за допомогою методу CalculateStart;

– запускає створення тайлів через метод TilesCreator, передаючи нові праметри;

– присвоює нові параметра ігрового поля у відповідні поля класу.

Метод CalculateStart, визначений в інтерфейсі ITileConstructor, має наступну логіку:

– знаходить загальну ширину поля, множачи розмір тайла на ширину, яка відповідає за кількість колонок;

– знаходить висоту ігрового поля, множачи розмір тайл на висоту, яка відповідає за кількість рядків;

– розраховується координати верхнього лівого кута, значення «x» призначається розділивши загальну ширину поля на два, значення «y» – розділивши загальну висоту поля на два;

– розраховується де розмістити центр першого тайла, для значення «x» до координати «x» верхнього лівого кута додається результат дівлення розміру тайла на два, операція повторюється для «y».

Метод TilesCreator видаляє старі створені тайли, якщо вони є, та виликає метод ConstructNewTiles з ITileConstructor.

ConstructNewTiles проходить по ширині та висоті поля та на кожній ітерації створює новий тайл. При створені рядка зміщує центр наступного тайла, додаючи до координати «x», поточного створеного тайла, розмір клітинки. Для переходу на наступний рядок встановлює для «x» координати початку та для «y» віднімає розмір тайла. Усі створені тайли додаються до списку для зручності редагування та використання.

2.4 Проектування і опис інтерфейсу користувача

Інтерфейс користувача має наступні елементи:

1. Головне меню:
   - кнопка для виходу з гри;
   - кнопка перегляду інформації про розробника;
   - кнопка вибору набору рівнів.
2. Вікно «Про розробника»:
   - Інформація про розробника;
   - кнопка повернення до головного меню.
3. Вікно вибору набору рівнів:
   - кнопка повернення до головного меню;
   - кнопка для вибору набору рівнів.
4. Вікно вибору рівня:
   - кнопка повернення до вибору набору рівнів;
   - кнопки рівнів.
5. Ігровий процес:
   - панель з кнопками для вибору вежі, яка містить зображення вежі та надпис, який відображає ціну розміщення вежі;
   - лічильник кількості наявних очків розміщення веж;
   - лічильник втрачених очків;
   - лічильник кількості веж, які ще можна розмістити;
   - UI-елемент для позначення вибраної активної вежі гравцем, містить дві кнопки: кнопка для знищення вежі та кнопка виходу з режиму вибраної вежі;
   - вікно паузи з кнопками для продовження гри та виходу до головного меню.

Кнопки підв’язуються до відповідних методів з логікою у класах. Це реалізовується за допомогою події «OnClick» на вбудованому в рушій компоненті кнопки.

Для відображення тексту використовуються компоненти «TextMeshPro», який містить інструменти для роботи з текстом.

На рисунку 2.3 зображено концепція вигляду головного меню гри.

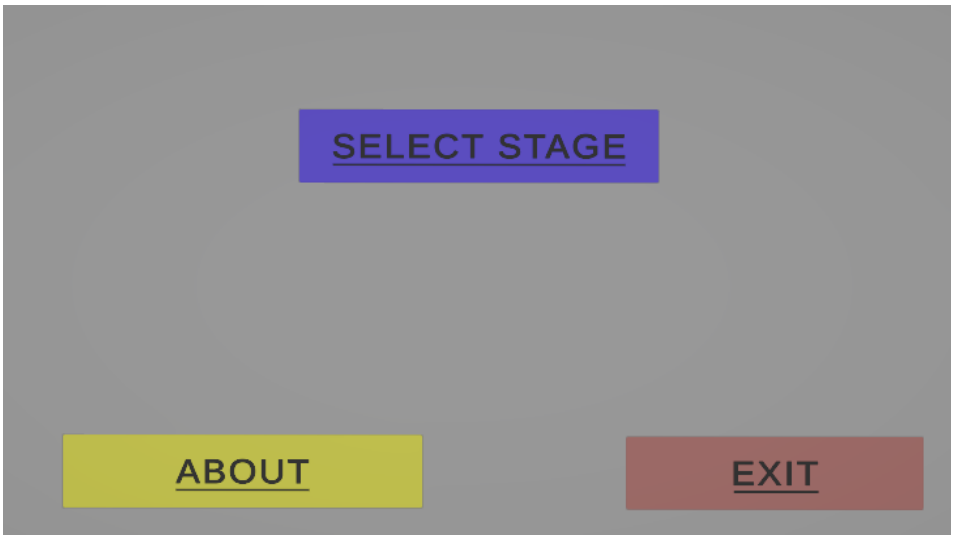


Рисунок 2.3 – Концепція вигляду головного меню гри

Згідно рисунку 2.3, кнопка «Select Stage» відповідає за перехід до вікна вибору набору рівнів, кнопка «About» – перехід до вікна інформації про розробника, кнопка «Exit» – вихід з гри.

На рисунку 2.4 зображено концепцію вікна «Про розробника».

Рисунок 2.4 – Концепція вікна «Про розробника»

Згідно рисунку 2.4, кнопка «Back» відповідає за перехід до головного меню.

На рисунку 2.5 зображено концепцію вікна вибору набору рівнів.

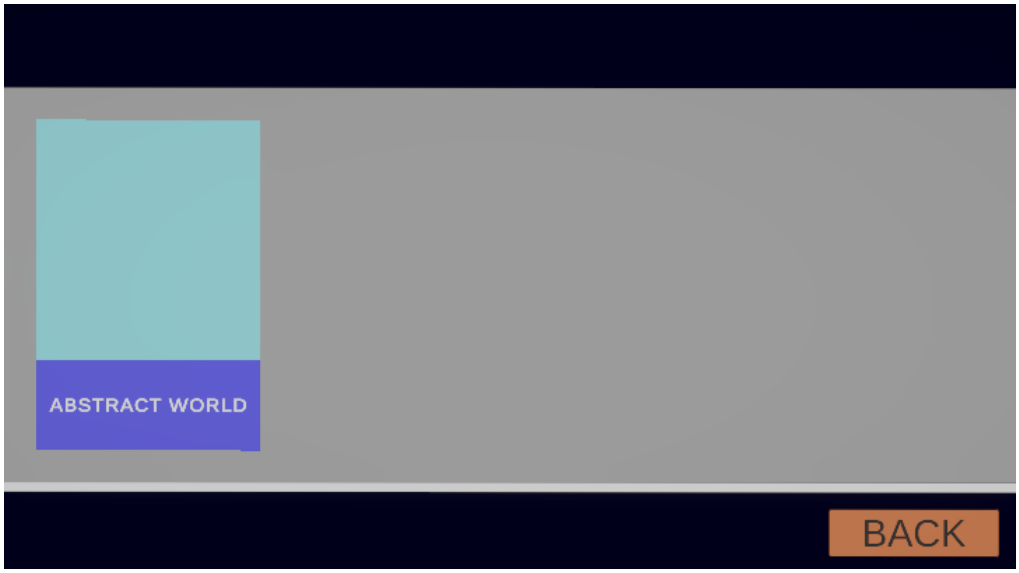


Рисунок 2.5 – Концепція вікна вибору набору рівнів

Згідно рисунку 2.5, кнопка «Back» відповідає за повернення до головного меню, кнопка з надписом «Abstract World» – один з наборів рівнів, при виборі відкриває вікно з вибором рівня.

На рисунку 2.6 зображено концепцію вікна вибору рівня.

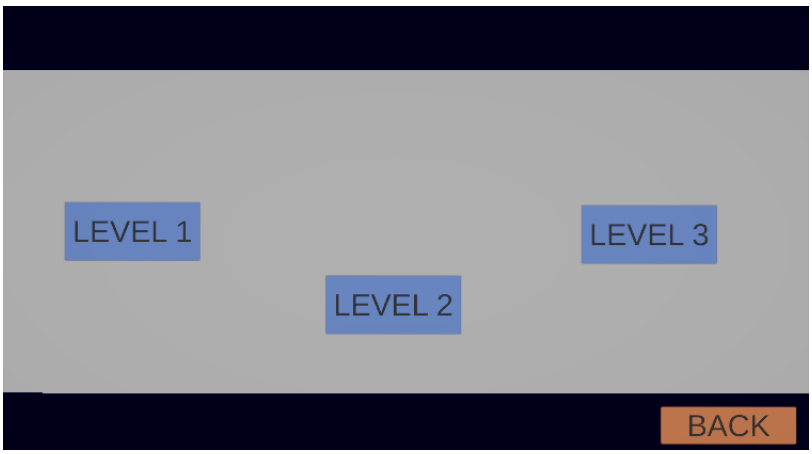


Рисунок 2.6 – Концепція вікна вибору рівня

Згідно рисунка 2.6, кнопки з надписами «Level 1», «Level 2», «Level 3» - кнопки для запуску рівня, кнопка «Back» – для повернення до вікна вибору набору рівнів.

На рисунку 2.7 зображено концепцію інтерфейсу користувача під час ігрового процесу.

Рисунок 2.7 – Концепція інтерфейсу користувача під час ігрового процесу

На рисунку 2.8 зображено концепці. меню паузи під час ігрового процесу.

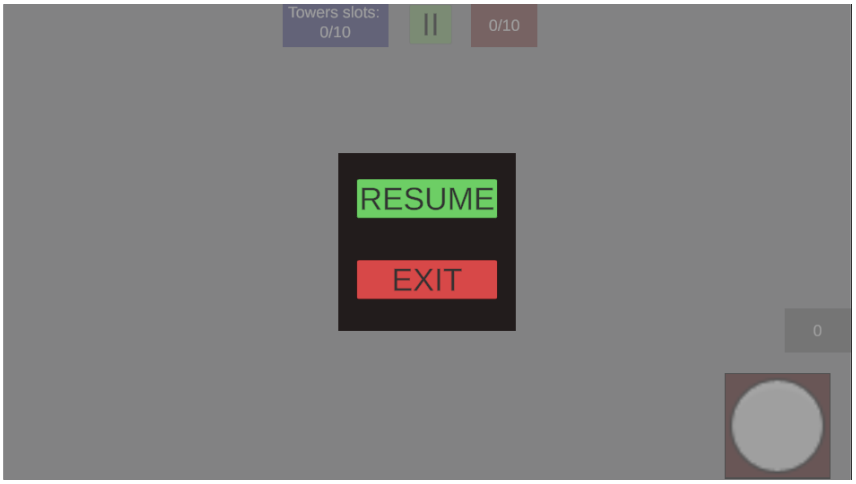


Рисунок 2.8 – Концепція меню паузи під час ігрового процесу

2.5 Опис файлової структури програми

Файлова структура гри на Unity, при виборі Windows як цільової платформи для компіляції, виглядає наступним чином:

– GameName.exe;

– GameNameData;

– UnityCrashHandler64.exe;

– UnityPlayer.dll;

– MonoBleedingEdge.

На рисунку 2.9 зображено файлову структуру гри на Unity після збірки.

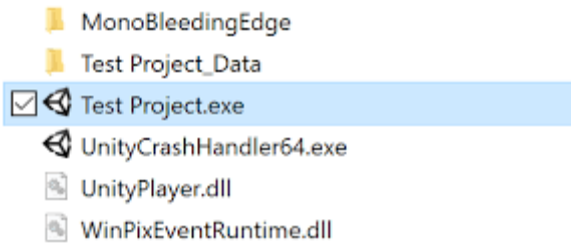


Рисунок 2.9 – Файлова структура гри на Unity після збірки

GameName.exe – виконуваний файл, який використовується для запуску гри. Має назву, яка вказується при компіляції проекту в редакторі.

GameNameData – папка з ресурсами та даними гри. Тут зберігаються глобальні налаштування проєкту: параметри фізики, Input Manager (маппінг кнопок), налаштування якості графіки, теги та шари, Time Manager тощо. Фактично це серіалізований стан усіх Unity Manager-компонентів, які існують поза сценами.

Файли з назвами: «Level1» «Level2» тощо. Це сцени, які були обрані при компіляції проекту. Цифра в кінці означає порядковий номер в налаштуваннях збірки.

Підпапка Managed – директорія з усім кодом та .Net бібліотеками.

MonoBleedingEdge – папка з компонентами Mono, які використовуються у проекті. Появляється лише, коли для бекенд скриптів використовується Mono.

Результатом підрозділу є додаток «Діаграма файлової структури.»

2.6 Тестування програми і результати її виконання

Через використання паттерна «Singleton» для менеджерів, які ініціалізуються лише під час рантайму, для тестування використовується метод тестування «чорної скриньки». Для цього створюється список функціоналу, який треба перевірити та результати, які треба отримати. Після цього можна приступати до тестування, під час цього необхідно записувати, чи були проблеми, якщо так – описувати їх так, щоб можна було виявити їх та виправити.

Далі наведено список функціоналу для перевірки та її результати:

– при виборі кнопки «About», має відкритися нове вікно, з інформацією про автора. Результат перевірки: вікно з інформацією про автора появилось, текст відображється чітко;

– при виборі кнопки «Back», у вікні з інформацією про автора, має повернутися головне меню гри. Результат перевірки: після натискання на кнопку, головне меню знову появилося на екрані;

– при виборі кнопки «Select Stage», має появитися вікно вибору набору рівнів. Результат перевірки: після натискання на кнопку, появилося вікно вибору набору рівнів;

– при виборі кнопки «Back», у вікні вибору набору рівнів, має появитися вікно головного меню. Результат перевірки: після натискання на кнопку, появилося головне меню;

– при вибору набору рівнів, у вікні наборів рівнів, має появитися вікно вибору рівня. Результат перевірки: після вибору набору рівнів, появилося вікно вибору рівнів;

– при натисканні на кнопку «Back», у вікні вибору рівня, має появитися вікно вибору набору рівнів. Результат перевірки: після вибору кнопки, появилося вікно вибору набору рівнів;

– при вибору рівня, у вікні вибору рівнів, має завантажитися рівень, створитися ігрове поле, ініціалізуватися ліильники, кнопка паузи та кнопки вибору веж. Результат перевірки: після вибору рівня, на екрані з’явилося ігрове поле, відображено лічильники, кнопка паузи та кнопки вибору веж;

– з плином часу, лічильник очків розміщення веж має збільшитися. Результат перевірки: через деякий проміжок часу, показник лічильника збільшився;

– через деякий час, мають появитися вороги, які будуть прямувати своїми маршрутами. Результат перевірки: вороги появилися на рівні та прямували за своїми маршрутами;

– при виборі кнопки паузи, гра має призупинитися, це можна замітити за тим, чи вороги все ще рухаються та чи лічильник очків розміщення веж призупинив свій ріст. Результат перевірки: після вибору кнопки паузи, з’явилося вікно меню паузи, рахунок очків розміщення веж призупинився та вороги припинили своє переміщення по ігровому полі;

– при виборі кнопки «Resume», у вікні меню паузи, гра має відновити свою роботу. Результат перевірки: після вибору варіанту «продовжити», гра відновила свою роботу;

– при виборі однієї з наявних для розміщення веж, місця наявні для розміщення, мають бути позначенні. Результат перевірки: наявні для розміщення тайли були позначенні спеціальними маркерами;

– при вибраній вежі, для розміщення, та наведенні її на позначений тайл, має появитися позначена зона, яку буде покривати вежа для атаки. Результат перевірки: після наведення вежі на можливий тайл для розміщення, появилися візуальна зона, яка позначила тайли, які буде атакувати вежа;

– при наведенні вибраної вежі на можливий для розміщення тайл та нажатті «ліва кнопка миші», місце розміщення має зафіксуватися та при русі

курсором миші, вежа має змінювати напрямок свого покриття. Результат перевірки: після наведення вежі на можливий тайл для розміщення та нажаття «ліва кнопка миші», вежа зафіксувалася на місці. При обертанні курсора навколо вежі, напрямок покриття вежі змінювався, що можна було зрозуміти по зоні покриття, яка змінювала своє розміщення;

– при зафіксованій вежі та нажатті «ліва кнопка миші», вежа має бути розміщена на тайлі фізично та має зменшитися кількість очків розміщення веж. Результат перевірки: після вибору тайла, напрямку вежі та нажатті «ліва кнопка миші», вежа появилася на ігровому полі;

– при натисканні «права кнопка миші», під час процесу розміщення вежі, операція має припинитися. Результат перевірки: під час розміщення вежі та нажатті «права кнопка миші», операція розміщення вежі на ігровому полі припинилася;

– якщо натиснути по активній на ігровому полі вежі, навколо вежі має появитися вікно з кнопками «Remove» та «Cancel». Результат перевірки: після натискання на активну на ігровому полі вежу, появилося вікно з кнопками;

– якщо при вибраній активній вежі на ігровому полі, натиснути кнопку «Cancel», UI-елемент навколо вежі має пропасти. Результат перевірки: після вибору вежі на ігровому полі, появи UI-елемента та вибору кнопки «Cancel», UI-елемент пропав;

– якщо при вибраній активній вежі на ігровому полі, натиснути кнопку «Remove», вежа має пропасти з ігрового поля. Результат перевірки: після вибору вежі на ігровому полі, появи UI-елемента та вибору кнопки «Remove», вежа пропала з ігрового поля;

– якщо на полі є активна вежа, вона має властивість атакувати та ворог проходить по її зоні атаки, останній має отримувати шкоду, при його поразці має збільшитися рахунок очків розміщення веж та має програватися анімація атаки. Результат перевірки: після розміщення вежі на ігровому полі і коли ворог зайшов у межі її атаки, програвлася анімація атаки та ворог

отримував шкоду, що привело до його поразки та збільшення рахунку розміщення веж;

– коли кількість втрачених очків досягнула своєї межі, яка відображається в лчильнику, гра має припинитися і має появитися вікно поразки. Результат перевірки: після того, як вороги завершили свої маршрути, лічильник втрачених очків збільшився та після досяганні межі, гра завершилася та появилося вікно поразки;

– якщо всі вороги на рівні були знищенні, має появитися вікно перемоги. Результат перевірки: після початку гри, розміщення веж та знищення усіх ворогів, появилося вікно перемоги;

– якщо вибрати кнопку «Back», у вікні поразки або перемоги, має появитися головне меню гри. Результат перевірки: після вибору кнопки «Back» з під вікна перемоги, появилося вікно головного меню гри;

– якщо на рівні вибрати кнопку паузи, та у ній вибрати кнопку «Exit», має появитися вікно головного меню. Результат перевірки: під час ігрового процесу, вибрано кнопку паузи, з під його меню вибрано кнопку «Exit», появилося вікно головного меню;

– якщо з під головного меню вибрати кнопку «Exit», гра має припинити свою роботу. Результат перевірки: після вибору кнопки «Exit», гра припинила свою роботу.

Після успішного тестування гри, можна приступати до виготовлення інсталятора.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з інсталяції програмного забезпечення

Гра розповсюджується у вигляді виконувального файлу, для інтсаляції необхідно виконути наступні кроки:

1. Отримати інсталятор.
2. Відкрити його.
3. Виконати необхідні кроки, які описані в ньому.
4. Дочекатися завершення інсталяції.
5. Запустити гру.

3.2 Інструкція з використання тестових наборів

Для тестування гри необхідно:

– впевнетися, що системні вимоги персонального комп’ютера вимагають вимогам програми;

– встановити гру на систему;

– запустити гру;

– дочекатися завантаження головного меню;

– відкрити список функціоналу з підрозділу «Тестування програми та результати її виконання» або «Вимоги до програмного забезпечення»;

– виконати описанні пункти у відкритій грі;

– звірити результати з описаними.

3.3 Інструкція з експлуатації програмного комплексу

Гра вимагає від системи, на якій вона запускається, такі вимоги:

– операційна система: Windows 10;

– процесор: 2-х ядерний з частотою 3.6 GHz;

– оперативна пам’ять: 2 GB;

– накопичувач: 1 GB;

– відеокарта: інтегрована.

– засоби вводу: клавіатура та комп’ютерна миша.

Також, наступні системні компоненти мають бути встановленні на системі:

– DirectX 11;

– NET Framework NET 3.5 Redistributable.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини кваліфікаційної роботи бакалавра є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності проєкту «Розробка комп'ютерної гри «SiN» з використанням C#».

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

Для визначення загальної тривалості проведення НДР зведено таблицю 4.1, в яку занесено дані витрат часу по окремих операціях технологічного процесу.

Таблиця 4.1 – Середній час виконання НДР та стадії (операції) технологічного процесу

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1	2	3	4
1	Формулювання задачі	керівник	24
2	Аналіз та проектування	керівник	48
3	Розробка основних механік	програміст	36
4	Розробка 3D моделей та анімацій	Графічний дизайнер, аніматор	120
5	Розробка контенту	Сценарист, геймдизайнер	72
6	Реалізація контенту	програміст	120

Продовження таблиці 4.1

1	2	3	4
7	Тестування	Тестувальник	72
8	Реліз	Керівник	8
Разом			500

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Оплата праці – грошовий вираз вартості і ціни робочої сили, який виступає у формі будь-якого заробітку, виплаченого керівником підприємства найманому працівникові за виконану роботу. Заробітна плата працівника залежить від кінцевих результатів його роботи, регулюється податками і максимальними розмірами не обмежується.

Основна заробітна плата розраховується за формулою 4.1:

$$З_{осн} = Т_с * К_г \quad (4.1)$$

де, $Т_с$ – тарифна ставка, грн.;

$К_г$ – кількість відпрацьованих годин.

Рекомендовані тарифні ставки: керівник проєкту – 150 грн./год., програміст – 250 грн./год., графічний дизайнер/аніматор – 150 грн./год., сценарист – 150 грн./год., геймдизайнер – 150 грн./год., тестувальник – 130 грн./год.

Основна заробітна плата становить:

- Керівника: $З_{осн1} = 150 * 80 = 12\,000$ грн.;
- Програміста: $З_{осн2} = 250 * 156 = 39\,000$ грн.;
- Графічного дизайнер/аніматор: $З_{осн3} = 150 * 120 = 18\,000$ грн.;
- Сценарист та геймдизайнер: $З_{осн4} = (150 * 72)*2 = 21\,600$ грн.;
- Тестувальник: $З_{осн5} = 130 * 72 = 9\,360$ грн.

Сумарна основна заробітна плата становить:

$З_{осн}$ = 12 000 + 39 000 + 18 000 + 21 600 + 9360 = 99 960 грн.

Додаткова заробітна плата становить 10-15 % від суми основної заробітної плати та обчисоюється за формулою 4.2:

$$З_{дод} = З_{осн} * К_{допл} , \quad (4.2)$$

де, $К_{допл}$ – коефіцієнт додаткових виплат працівникам 0,1-0,15.

Отже, додаткова заробітна плата становить:

1. Керівника: $З_{дод1}$ =12 000 * 0,1 = 1 200 грн.;
2. Програміста: $З_{дод2}$ = 39 000 * 0,1 = 3 900 грн.;
3. Графічного дизайнер/аніматор: $З_{дод3}$ =18 000 * 0,1 = 1 800 грн.;
4. Сценарист та геймдизайнер: $З_{дод4}$ = (10 800 * 0,1) * 2 = 2160 грн. кожен;
5. Тестувальник: $З_{дод5}$ = 9 360 * 0,1 = 936 грн.

Сумарна додаткова заробітна плата становить:

$З_{дод}$ = 1 200 + 3 900 + 1 800 + 2160 + 936 = 9 996 грн.

Звідси загальні витрати на оплату праці визначаються за формулою 4.3:

$$В_{о.п.} = З_{осн} + З_{дод} \quad (4.3)$$

$В_{о.п.}$ = 99 960 + 9 996 = 109 956 грн.

Необхідно визначити відрахування на соціальні заходи:

– фонд страхування на випадок безробіття – 1,6 %;

– фонд по тимчасовій втраті працездатності – 1,4 %;

– пенсійний фонд – 33,2 %;

– внески на страхування від нещасного випадку на виробництві та професійного захворювання – 1,4%.

Загальна сума зазначених відрахувань становить 37,6 %.

Отже, сума нарахувань на заробітну плату буде становити згідно формули 4.4:

$$В_{с.з.} = ФОП * 0,376, \quad (4.4)$$

Де ФОП – фонд оплати праці, грн.

$$В_{с.з.} = 109\,956 * 0,376 = 41\,343,46 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці зведено у таблиці 4.2.

Таблиця 4.2 – Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівника	Основна заробітна плата, грн			Додаткова заробітна плата, грн.	Нарах ув. На ФОП, грн.	Всього витрати на оплату праці, грн.
		Тарифна ставка, грн.	К-сть відпрацьов. год.	Фактично нарах. з/пл., грн.			
1	Керівник	150	80	12 000	1 200	-	-
2	Програміст	250	156	39 000	3 900	-	-
3	Графічний дизайнер/аніматор	150	120	18 000	1 800	-	-
4	Сценарист та геймдизайнер	130	72	21 600	2160	-	-
5	Тестувальник	130	72	9 360	936	-	-
Разом				99 960	9 960	41 343,46	151 263, 46

Загальні витрати на оплату праці становлять 151 263, 46 грн.

4.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються за наступною формулою (4.5):

$$М_{в.і.} = q_i * p_i, \quad (4.5)$$

де, q_i – кількість витраченого матеріалу і-го виду;

p_i – ціна матеріалу і – го виду.

Звідси, загальні матеріальні витрати можна визначити за формулою 4.6:

$$З_{м.в.} = \sum М_{в.і.}, \quad (4.6)$$

Проведені розрахунки занесемо у таблицю 4.3.

Таблиця 4.3 – Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витрачено матеріалів	Ціна 1-ці, грн.	Загальна сума витрат, грн
1	2	3	4	5	6
1	Процессор Ryzen 5 7600X	шт.	1	9 300	9 300
2	Відеокарта RTX 5060	шт.	1	15 600	15 600
3	Оперативна пам’ять 16 GB	шт.	2	3 200	6 400
4	Материнська плата MSI B850I	шт.	1	14 352	14 352
5	Корпус з блоком живлення Cooler Master MasterBox NR200P MAX	шт.	1	14 976	14 976
6	Накопичувачі	шт.	4	2800	11 200
7	Монітор	шт.	1	4 000	4 000

Продовження таблиці 4.3.

1	2	3	4	5	6
8	Клавіатура	шт.	1	3 600	3 600
9	Комп’ютерна миша	шт.	1	2 500	2 500
10	Графічний редактор	коп.	1	1 420	1 420
11	Ігровий рушій	коп.	1	-	-
Разом					83 348

Загальна сума матеріальних витрат становить 83 348 грн.

4.4 Розрахунок витрати на електроенергію

Витрати на електроенергію 1-ці обладнання визначається за формулою:

$$З_е = W * T * S, \quad (4.7)$$

де, W – необхідна потужність, кВт;

T – кількість годин роботи обладнання;

S – вартість кіловат-години електроенергії.

Час роботи ПК над даним проєктом становить 500 годин, споживана потужність 0,7 кВт.год., вартість 1 кВт електроенергії – 7 грн.

$$З_е = 0,5 * 500 * 7 = 1\,750 \text{ грн.}$$

4.5 Визначення транспортних затрат

Транспортні витрати слід прогнозувати у розмірі 8-10 % від загальної суми матеріальних затрат. Транспортні витрати розраховуються за формулою 4.8.

$$Т_в = З_{м.в.} * 0,08 \dots 0,1, \quad (4.8)$$

Отже, транспортні витрати будуть становити:

$$Т_в = 83\,348 * 0,09 = 7\,501 \text{ грн.}$$

4.6 Розрахунок суми амортизаційних відрахувань

Комп’ютери та оргтехніка належать до четвертої групи основних фондів. Мінімально допустимі строки їх використання 2 роки. Для визначення амортизаційних відрахувань застосовуємо формулу 4.9:

$$A = \frac{Б_в * Н_а}{150\%} * Т_А, \quad (4.9)$$

де, А – амортизаційні відрахування за звітній період, грн;

$Б_в$ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

$Н_а$ – норма амортизації, 0,04%.

Для написання програми та її тестування використовувався ПК та оргтехніка із сумарною вартістю 83 348, тому:

$$А = \frac{83\ 348 * 0{,}04}{150} * 500 = 11\ 113 \text{ грн}$$

4.7 Обчислення накладних витрат

Накладні витрати – це витрати, не пов'язані безпосередньо з технологічним процесом виготовлення продукції, а утворюються під впливом певних умов роботи по організації, управлінню та обслуговуванню виробництва. В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20 – 60 % від суми основної та додаткової заробітної плати працівників, обчислюються за формулою 4.10, де НВ – накладні витрати.

$$H_в = B_{о.п.} * 0{,}2...0{,}6, \quad (4.10)$$

$$Н_в = 109\ 956 * 0{,}4 = 43\ 982{,}4 \text{ грн.}$$

4.8 Складання кошторису витрат та визначення собівартості НДР

Кошторис витрат являє собою зведений план усіх витрат підприємства на майбутній період виробничо-фінансової діяльності.

Результат проведених вище розрахунків зведено у таблицю 4.4.

Таблиця 4.4 – Кошторис витрат на НДР

Зміст витрат	Сума, грн	В % до загальної суми
Витрати на оплату праці	109 956	37.10
Відрахування на соціальні заходи	41 343,46	13.95
Матеріальні витрати	83 348	28.12
Витрати на електроенергію	1 750	0.59
Транспортні витрати	4 915,17	1.66
Амортизаційні відрахування	11 113	3.75
Накладні витрати	43 982,4	14.84
Собівартість	296 408.03	

Собівартість ($С_в$) НДР розрахуємо за формулою 4.11:

$$С_в = В_{0.л.} + В_{с.з.} + З_{м.в.} + З_е + Т_в + А + Н_в \quad (4.11)$$

Отже, собівартість становить $С_в$ = 296 408.03 грн.

4.9 Розрахунок ціни НДР

Ціну НДР можна визначити за формулою 4.12:

$$Ц = С_в * (1 + Р_{рен.}) * (1 + ПДВ), \quad (4.12)$$

Де: $С_в$ – собівартість виконання НДР;

$Р_{рен.}$ – рівень рентабельності;

ПДВ – ставка податку на додану вартість, 20%.

Ц = 296 408.03 * (1 + 0,3) * (1 + 0,2) = 462 396.53 грн.

4.10 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Прибуток розраховується за формулою 4.13:

$$П = Ц - С_в, \quad (4.13)$$

П = 462 396.53 – 296 408.03 = 165 988.50 грн.

Економічна ефективність ($Е_р$) полягає у відношенні результату виробництва до затрачених ресурсів і розраховується за формулою 4.11, де П – прибуток; $С_в$ – собівартість.

$$Е_р = \frac{П}{С_в}, \quad (4.14)$$

$Е_р$ = 165 988.50 / 296 408.03 = 0,56.

Поряд із економічною ефективністю розраховують (формула 4.15) термін окупності капітальних вкладень $Т_р$:

$$Т_р = 1/Е_р \quad (4.15)$$

Допустимим вважається термін окупності до 5 років. В даному випадку:

$Т_р$ = 1/0,56 = 1,8

Всі дані внесемо в зведену таблицю 4.5 техніко-економічних показників.

Таблиця 4.5 – Техніко-економічні показники

№ п/п	Показник	Одиниця виміру	Значення
1	2	3	4
1	Собівартість, грн.	грн.	296 408.03
2	Плановий прибуток, грн.	грн.	165 988.50
3	Ціна, грн.	грн.	462 396.53
4	Економічна ефективність	-	0,56
5	Термін окупності, рік	рік	1,8

Загальна вартість розробки комп’ютерної гри становить 462 396.53 грн.

Зважаючи на високі показники економічної ефективності – 0,56, кошти, вкладені в проведення проектних робіт окупляться за 1,8 року.

5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

5.1 Розрахунок системи штучного освітлення для приміщення, де здійснюється розробка комп'ютерної гри «SiN»

Під час розробки комп’ютерної гри «SiN» особливу увагу необхідно приділяти вимогам до освітлення робочих приміщень. Це пов’язано з тим, що тривала робота за комп’ютером і з документацією за умов недостатнього освітлення може спричинити значне навантаження на зір. Тому слід дотримуватися таких вимог:

– природне освітлення повинно забезпечувати коефіцієнт природної освітленості (КПО) не менше 1,5 %. Для регулювання інтенсивності денного світла доцільно використовувати жалюзі;

– робоче місце з персональним комп’ютером необхідно розміщувати таким чином, щоб прямі сонячні промені не потрапляли в очі працівника;

– штучне освітлення приміщення має бути організоване за допомогою системи загального рівномірного освітлення;

– використання світильників без розсіювачів або захисних екрануючих решіток не допускається;

– освітленість робочої поверхні в зоні розміщення документів повинна становити від 300 до 500 лк.

Призначене приміщення для розробки комп’ютерної гри має розміри: 4х4 метри та висоту 3,5 метри. Необхідно розрахувати систему штучного світлення.

Вихідні дані:

– площа S = 4 * 4 = 16 м^2;

– висота стелі H = 3,5 м;

– норма освітленості E = 400 лк (середнє значення);

– тип світильників – LED-панелі, світловий потік = 4000 лм.

Для розрахунку загального світлового потоку скористаємося формулою 5.1.

$$Ф = E * S * коеф.висоти, \quad (5.1)$$

де: Ф – світловий потік;

E – освітленність (E = 400 лк);

S – площа приміщення (S = 16 м^2);

коеф.висоти (при 3,5 м = 2)[3].

$$Ф = 400 * 16 * 2 = 12\,800 \text{ лм}$$

Для обрахунку кількості світильників використовуємо формулу 5.2.

$$N = \frac{E*S*K_з*z}{Ф_{одн}*n}, \quad (5.2)$$

де: E – освітленність (E = 400 лк);

S – площа приміщення (S = 16 м^2);

$K_з$ – запас освітленості ($K_з$ = 1,5);

$Ф_{одн}$ – світлоивй потік одного світильника ($Ф_{одн}$ = 4000 лм);

n – коефіцієнт використання світлового потоку (n = 0,6).

$$N = \frac{400 * 16 * 1{,}5 * 1{,}1}{4000 * 0{,}6} = 4{,}4$$

Отже, для економічного варіанту, можна використати 5 світильників, кожен з яких буде забезпечувати по 4000 лм.

Розміщувати світильники необхідно наступним чином:

– відстань між світильниками 1 м;

– відстань світильників від стін 1 м;

– вікно, розташоване праворуч від персонального комп'ютера, буде забезпечувати додаткове освітлення.

На рисунку 5.1 зображено план розміщення світильників у приміщення розробки комп'ютерної гри.

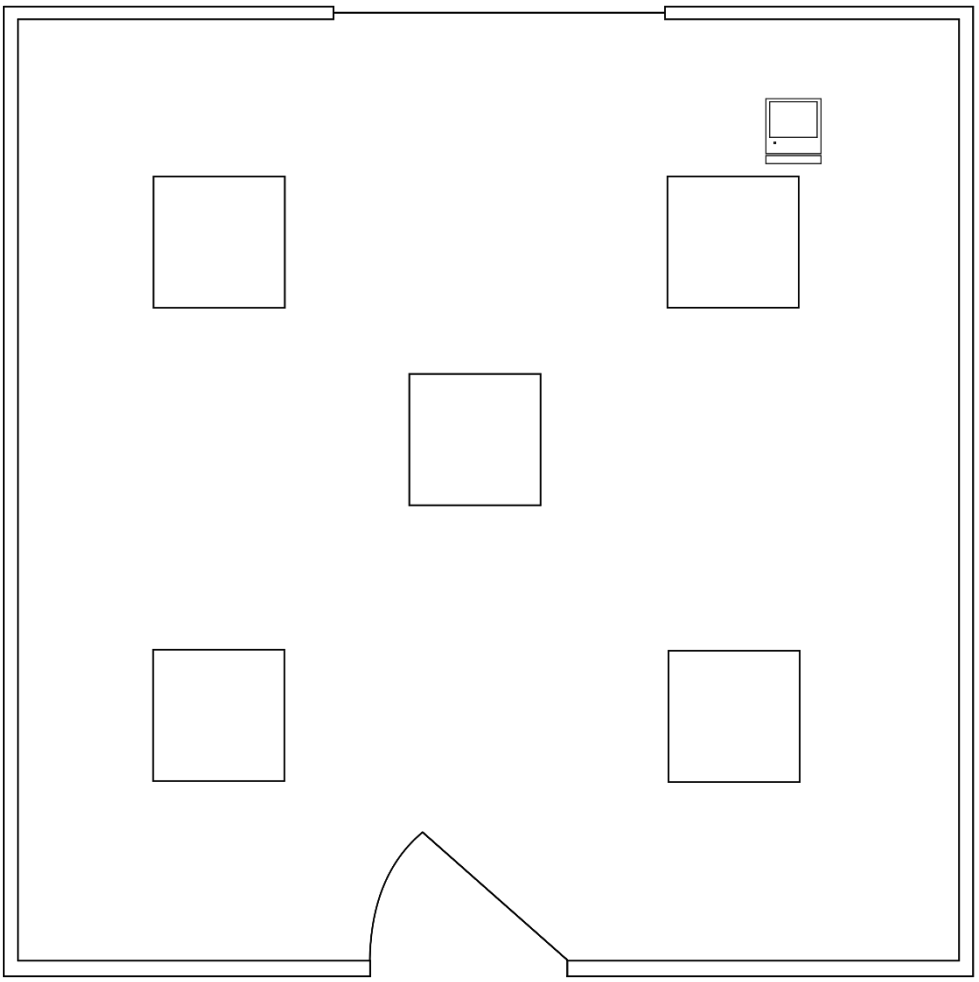


Рисунок 5.1 – План розміщення світильників у приміщення розробки комп'ютерної гри

5.2 Ергономічні проблеми безпеки життєдіяльності

Ергономіка є важливою складовою безпеки життєдіяльності людини. Вона вивчає закономірності взаємодії людини з технікою, обладнанням, виробничим середовищем та інформаційними системами з метою створення безпечних, комфортних і ефективних умов праці та життя. Ергономічні проблеми виникають тоді, коли умови діяльності не відповідають фізичним, психофізіологічним та психологічним можливостям людини. Такі невідповідності можуть призводити до зниження працездатності, підвищення втоми, професійних захворювань, травматизму та аварійних ситуацій [2].

У сучасному світі роль ергономіки постійно зростає через розвиток технологій, автоматизацію виробництва та збільшення інформаційного навантаження на людину. Тому вивчення ергономічних проблем безпеки життєдіяльності є необхідною умовою забезпечення здоров'я, працездатності та безпеки населення [2].

Невідповідність робочого місця антропометричним параметрам людини.

Однією з найпоширеніших ергономічних проблем є неправильна організація робочого місця. Якщо висота столу, стільця, розташування органів керування або інструментів не відповідають фізичним параметрам працівника, виникає надмірне навантаження на опорно-руховий апарат.

Наприклад, під час тривалої роботи за комп'ютером у неправильній позі збільшується навантаження на шийний та поперековий відділи хребта. Це може призвести до хронічних захворювань і зниження працездатності [3].

Надмірні фізичні навантаження.

У багатьох сферах діяльності працівники змушені виконувати важку фізичну роботу, підіймати та переносити вантажі, працювати у незручних позах. Особливо небезпечними є статичні навантаження, коли людина тривалий час перебуває в одній позі. У таких умовах порушується кровообіг, виникають м'язові спазми та хронічний біль.

Інформаційне перевантаження.

Сучасна людина щоденно обробляє великі обсяги інформації. Оператори складних систем, диспетчери, водії, медичні працівники, програмісти та інші спеціалісти стикаються з необхідністю швидко аналізувати інформацію та приймати рішення. У критичних ситуаціях інформаційне перевантаження може призвести до аварій, катастроф або нещасних випадків [4].

Недостатнє освітлення робочого середовища.

Освітлення безпосередньо впливає на зорову працездатність людини. Недостатня або надмірна освітленість створює небезпечні умови праці. Особливу небезпеку становлять відблиски на моніторах, різкі контрасти освітлення та мерехтіння джерел світла.

Психологічні та психоемоційні навантаження та монотонність праці.

Психологічний стан людини безпосередньо впливає на безпеку її діяльності. У стані стресу людина частіше допускає помилки, що підвищує ризик аварій та нещасних випадків. Монотонна робота характеризується багаторазовим повторенням однакових операцій протягом тривалого часу. Монотонність особливо небезпечна для операторів транспортних засобів, диспетчерів та працівників конвеєрного виробництва [5].

Шляхи вирішення.

Вирішення ергономічних проблем потребує комплексного підходу, який включає вдосконалення технічних засобів, раціональну організацію праці, створення комфортного виробничого середовища та врахування людського фактора на всіх етапах проєктування і експлуатації технічних систем. Лише за таких умов можна забезпечити високий рівень безпеки життєдіяльності, зберегти здоров'я людей та підвищити ефективність їхньої діяльності.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи розроблено комп'ютерну відеогру «SiN» з використанням мови програмування C# та ігрового рушія Unity.

У першому розділі проведено аналітичний огляд існуючих рішень на ринку, виявлено їх слабкі та сильні сторони, створено технічне завдання.

У другому розділі розроблено загальну структуру гри та варінти її використання, розроблено систему класів з полями та методами, розроблено методи, спроектовано та опиано інтерфейс користувача, описано файлову систему комп'ютерної гри, описано та проведено тестування, відображено результати тестування.

За результатами виконання другого розділу розроблено UML – діаграму варіантів використання програми, UML – діаграму послідовності дій, UML – діаграму класів програми, UML – діаграму файлової структури програми. І представлено на окремих плакатах графічної частини дипломного проекту.

У третьому розділі надано інструкцію з комп'ютерної гри, порядок тестування та експлуатації програмного комплексу.

Останній розділ описує питання охорони та техніки безпеки, яке було отримано від консультанта – викладача з охорони праці.

Дипломний проєкт містить увесь обсяг інформації та документації, необхідний для проектування та впровадження комп'ютерної гри «SiN».

ПЕРЕЛІК ПОСИЛАНЬ

1. Методичні вказівки до виконання дипломного проекту з спеціальності 5.05010101 «Обслуговування програмних систем і комплексів» напрямок «Програмування» [Електронний ресурс] – Режим доступу до ресурсу: https://eguru1.tk.te.ua/pluginfile.php/76350/mod_resource/content/1/Metod_vkaziv_DP.pdf – Дата доступу: 01.05.2026р.

2. Значення ергономіки в системі безпеки життєдіяльності людини [Електронний ресурс] – Режим доступу до ресурсу: https://studfile.net/preview/16542650/ – Дата доступу: 30.05.2026р.

3. Ергономічні вимони до організаціх робочих місць [Електронний ресурс] – Режим доступу до ресурсу: https://studies.in.ua/bjd-lapin/1142-ergonomchn-vimogi-do-organzacyi-robochih-msc.html – Дата доступу: 30.05.2026р.

4. Інформаційне перевіантаження: як не перевтомити свій мозок [Електронний ресурс] – Режим доступу до ресурсу: https://pon.org.ua/novyny/8267-nformacyne-perevantazhennya-yak-ne-perevtomiti-mozok.html – Дата доступу: 30.05.2026р.

5. Психофізіологічна суть і критерії монотонності праці [Електронний ресурс] – Режим доступу до ресурсу: https://buklib.net/books/25962/ - Дата доступу: 30.05.2026р.

ДОДАТКИ

Додаток А. Текст програми

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

public class LevelMaker : MonoBehaviour, IField, ITileConstructor, IAllSaver
{
    [SerializeField]
    [Range(1, 256)]
    int m_rowsCount = 1;

    [SerializeField]
    [HideInInspector]
    int m_currentRowsCount = 0;

    [SerializeField]
    [Range(1, 256)]
    int m_colsCount = 1;

    [SerializeField]
    [HideInInspector]
    int m_currentColsCount = 0;

    [SerializeField]
    int m_tileWHSize = 2;
```

```
        [SerializeField]
        int m_currentTileWHSize;

        [SerializeField]
        GameObject m_tileObject;

        [SerializeField]
        List<GameObject> m_tilesContainer = new List<GameObject>();

        private AllSaver m_allSaverReference;

        public void CreateField()
        {
            if (IsTheSameRowsAndColumns() &&
                IsTheSameTileSize() || !m_tileObject) return;

            Vector2 start =
((ITileConstructor)this).CalculateStart(m_colsCount, m_rowsCount,
m_tileWHSize);

            TilesCreator(m_rowsCount, m_colsCount, m_tileWHSize, start);

            m_currentRowsCount = m_rowsCount;

            m_currentColsCount = m_colsCount;

            m_currentTileWHSize = m_tileWHSize;

            Debug.Log("Field Created!");
```

```
        }

        private bool IsTheSameTileSize()
        {
            return m_currentTileWHSize == m_tileWHSize;
        }

        bool IsTheSameRowsAndColumns()
        {
            return !(m_rowsCount != m_currentRowsCount || m_colsCount !=
m_currentColsCount);
        }

        void TilesCreator(int rows, int columns, float tileWH, Vector2 start)
        {
            DeleteOldTiles();

            ((ITileConstructor)this).ConstructNewTiles(rows, columns,
tileWH, start, m_tileObject, m_tilesContainer,
this.GetComponent<Transform>());
        }

        void DeleteOldTiles()
        {
            if (m_tilesContainer.Count > 0)
            {
                for (int i = m_tilesContainer.Count() - 1; i >= 0; i--)
                {
                    if (m_tilesContainer[i])
                    {
```

```
                DestroyImmediate(m_tilesContainer[i]);
                m_tilesContainer.Remove(m_tilesContainer[i]);
            }
        }

        m_tilesContainer.Clear();
    }
}

public void UpdateAllTiles()
{
    for(int i = 0; i < m_tilesContainer.Count; i++)
    {
        if (m_tilesContainer[i].GetComponent<EditorTile>() is
EditorTile tileData)
        {
            tileData.GetDataFromObject();
        }
    }
}

public Field GetField()
{
    return new Field(m_currentRowsCount, m_currentColsCount,
        m_tilesContainer, m_tileWHSize);
}

public GameObject InstantiateObject(GameObject objectInstance,
Vector2 position)
{
```

```
        return Instantiate(objectInstance, position, Quaternion.identity);
    }

    public void SendFieldToSaver()
    {
        if(!IsReferenceToSaver())
        {
            if(FindSaver() == null)
            {
                Debug.LogError("Can't Find A Saver Object!");
                return;
            }
        }

        SetFieldInSaver();
    }

    public bool IsReferenceToSaver()
    {
        return m_allSaverReference;
    }

    public AllSaver FindSaver()
    {
        return m_allSaverReference =
FindFirstObjectByType<AllSaver>();
    }

    protected void SetFieldInSaver()
    {
```

```
                m_allSaverReference?.SetField(GetField());
            }
        }

        using System;
        using System.Collections.Generic;
        using SiN;
        using UnityEngine;

        [Serializable]
        public class Level
        {
            [Header("\u270fGeneral")]
            [SerializeField]
            private string m_levelName;

[Header("\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u27
26\u2726\u2726\u2726\u2726\u2726\u2726")]//Dividers For Inspector

            [Header("\u270fTiles Information")]

            [SerializeField]
            private List<string> m_tilesNames = new List<string>();

            [SerializeField]
            private List<int> m_tilesLayout = new List<int>();

            [SerializeField]
            private List<TagsContainer> m_tilesData = new
List<TagsContainer>();
```

```
        [SerializeField]
        private List<int> m_tileDataLayout = new List<int>();

[Header("\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u27
26\u2726\u2726\u2726\u2726\u2726\u2726")]

        [Header("\u270fField Parameters")]

        [SerializeField]
        private int m_rowsCount;

        [SerializeField]
        private int m_colsCount;

        [SerializeField]
        private int m_cellSize;

[Header("\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u27
26\u2726\u2726\u2726\u2726\u2726\u2726")]

        [Header("\u270fEnemies Information")]
        [SerializeField]
        private SiN.Paths m_paths;

        [SerializeField]
        private Waves m_waves;

        [SerializeField]
        private List<string> m_unitsNamesList;
```

```
[Header("\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u2726\u27
26\u2726\u2726\u2726\u2726\u2726\u2726")]

        [Header("\u270fFlow Information")]
        [SerializeField]
        private int m_losePointsAmount;

        [SerializeField]
        private int m_maxTowersCount;
        public Level() { }

        public Level(int rows, int cols, int tileSize)
        {
            BaseInitialize(rows, cols, tileSize);
        }

        private void BaseInitialize(int rows, int cols, int tileSize)
        {
            RowsCount = rows;
            ColsCount = cols;
            CellSize = tileSize;
        }

        public void InitializeFromLevel(Level field)
        {
            m_levelName = field.LevelName;

            m_tilesNames = field.TilesNames;
```

```
            m_tilesLayout = field.TilesLayout;

            m_paths = field.Paths;

            m_waves = field.Waves;

            m_unitsNamesList = field.UnitsNamesList;

            m_tilesData = field.TilesData;
            m_tileDataLayout = field.TilesDataLayout;

            BaseInitialize(field.m_rowsCount, field.ColsCount,
field.CellSize);
        }

        public string LevelName { get => m_levelName; set =>
m_levelName = value; }
        public List<string> TilesNames { get => m_tilesNames; set =>
m_tilesNames = value; }
        public List<int> TilesLayout { get => m_tilesLayout; set =>
m_tilesLayout = value; }

        public List<TagsContainer> TilesData {get => m_tilesData; set =>
m_tilesData = value;}
        public List<int> TilesDataLayout {get => m_tileDataLayout; set =>
m_tileDataLayout = value;}

        public int RowsCount { get => m_rowsCount; set => m_rowsCount
= value; }
```

```
        public int ColsCount { get => m_colsCount; set => m_colsCount =
value; }
        public int CellSize { get => m_cellSize; set => m_cellSize = value;
}
        public SiN.Paths Paths {get => m_paths; set => m_paths = value;}
        public Waves Waves {get => m_waves; set => m_waves = value;}

        public List<string> UnitsNamesList {get => m_unitsNamesList; set
=> m_unitsNamesList = value;}

        public int LosePointsAmount {get => m_losePointsAmount;}

        public int MaxTowersCount {get => m_maxTowersCount;}

        public void ClearPaths()
        {
            m_paths.ClearPaths();
        }
}

    using System;
    using System.Collections.Generic;
    using UnityEngine;

    [Serializable]
    public class Field : Level
    {
        [SerializeField]
        private List<GameObject> m_fieldTilesContainer = new
List<GameObject>();
```

```
        public Field(int rows, int cols, List<GameObject> tiles, int tileSize)
    : base(rows, cols, tileSize)
        {
            m_fieldTilesContainer = tiles;
        }

        public List<GameObject> FieldTilesContainer { get =>
    m_fieldTilesContainer; set => m_fieldTilesContainer = value; }

        public GameObject GetTileAtIndex(int x, int y)
        {
            return m_fieldTilesContainer[x * ColsCount + y];
        }

        public GameObject GetTileAtIndex(SiN.Point point)
        {
            return m_fieldTilesContainer[point.x * ColsCount + point.y];
        }
}

    using UnityEngine;

    [CreateAssetMenu(fileName = "Level Data", menuName =
"ScriptableObjects/TileData/New Level Data")]
    public class LevelDataSO : ScriptableObject
    {
        [SerializeField]
        private Level m_levelSettings;
```

```
    [SerializeField]
    private GameObject m_baseTileObject;

    public void SetFieldSettings(Level otherLevel)
    {
        m_levelSettings = otherLevel;
    }

    public GameObject GetBaseTileObject()
    {
        return m_baseTileObject;
    }

    public Level GetLevel()
    {
        return m_levelSettings;
    }
}
```

```
using System;
using SiN;
using UnityEngine;

[Serializable]
public class TileSettings
{
    [Header("Flag Parameters\n")]

    [SerializeField]
    private SiN.TagsContainer m_tags;
```

```
        [Header("Positioning")]

        [SerializeField]
        private Vector3 m_waypointLocalPosition;

        public TagsContainer Tags {get => m_tags; set => m_tags = value;
}
        public Vector3 WaypointLocalPosition {get =>
m_waypointLocalPosition; set => m_waypointLocalPosition = value;}

        public void Initialize(GameObject otherTile)
        {
            if (otherTile.GetComponent<EditorTile>() is EditorTile
otherSettings)
            {
                this.Tags = otherSettings.TileSettings.Tags;

                m_waypointLocalPosition =
otherSettings.TileSettings.m_waypointLocalPosition;
            }
            else
            {
                Debug.LogWarning($"No TileSettings Component On
{otherTile.name}.");
            }
        }
}
```

```
using UnityEngine;

[CreateAssetMenu(fileName = "Tile Data", menuName =
"ScriptableObjects/TileData/New Tile Data")]
public class TileData : ScriptableObject
{
    [SerializeField]
    private TileSettings m_tileSettings = new TileSettings();

    public void SetTileSettings(TileSettings otherTileSettings)
    {
        m_tileSettings = otherTileSettings;
    }

    public TileSettings GetTileSettings()
    {
        return m_tileSettings;
    }
}

using System;
using System.Collections.Generic;
using UnityEngine;

public class WavesCreator : MonoBehaviour
{
    [SerializeField]
    private SiN.Paths m_paths;

    [SerializeField]
```

```
        private List<SpawnPoint> m_spawnPoints = new
List<SpawnPoint>();

        [SerializeField]
        private List<GameObject> m_tilesWithSpawnpoints = new
List<GameObject>();

        [SerializeField]
        private GameObject m_spawnPointPrefab;

        [SerializeField]
        private AllSaver m_saver;

        [SerializeField]
        private EditorWaves m_allWaves;

        public void GetPaths()
        {
            if(!m_saver)
            {
                m_saver = FindFirstObjectByType<AllSaver>();
            }

            m_paths = m_saver.GetLevel().Paths;
        }

        public void CreateSpawnpoints()
        {
            if(m_paths.Length <= 0) return;
```

```
            DestroyOldSpawnpoints();

            if(!m_saver)
                m_saver = FindFirstObjectByType<AllSaver>();

            for(int path = 0; path < m_paths.Length; path++)
            {
                SiN.Point indexes = m_paths[path][0];

                GameObject tile = m_saver.GetField().GetTileAtIndex(indexes);

                m_tilesWithSpawnpoints.Add(tile);

                SpawnPoint point = tile.transform.GetComponentInChildren<SpawnPoint>();

                if(point == null)
                {
                    GameObject spawn = Instantiate(m_spawnPointPrefab, tile.transform.position + new Vector3(0,0,-2), Quaternion.identity);

                    spawn.name = $"Spawnpoint [{indexes.x}, {indexes.y}]";

                    spawn.transform.SetParent(tile.transform);

                    point = spawn.GetComponent<SpawnPoint>();

                    m_spawnPoints.Add(point);
                }
```

```
    }
}

public void DestroyOldSpawnpoints()
{
    if(m_spawnPoints.Count <= 0) return;

    for(int i = m_spawnPoints.Count - 1; i >= 0; i--)
    {
        DestroyImmediate(m_spawnPoints[i].transform.gameObject);
    }

    m_spawnPoints.Clear();

    m_tilesWithSpawnpoints.Clear();
}

public void SaveAll()
{
    if(m_allWaves.Count <= 0)
    {
        Debug.LogError("Waves Are Empty!");
        return;
    }

    SaveEnemies();
    SaveWaves();
}

public Waves SaveWaves()
```

```
{
    Waves newWaves = new Waves();

    newWaves.FormWavesFromEditorWaves(m_allWaves);

    m_saver.GetLevel().Waves = newWaves;

    return newWaves;
}

public void SaveEnemies()
{
    List<string> list = GetUnitsNames();

    if(list == null)
    {
        Debug.LogError("Some Unit Don't Have The Name");
        return;
    }

    m_saver.GetLevel().UnitsNamesList = list;
}

protected List<string> GetUnitsNames()
{
    List<string> units = new List<string>();

    foreach(var wave in m_allWaves._waves)
    {
        foreach(var step in wave._wave)
```

```
                {
                    string name = step._unit._unitName;

                    if(name == null || name == String.Empty) return null;

                    units.Add(name);
                }
            }

            return units;
        }
    }

    using System;
    using System.Collections.Generic;
    using UnityEngine;

    public class BaseUnit : MonoBehaviour
    {
        [SerializeField]
        protected StatsSO m_stats;

        protected float m_recievedDamage;

        protected List<GameplayTile> m_attackArea = new
List<GameplayTile>();

        protected BehaviorStrategy m_currentBehavior;

        protected SiN.StrategyEntry m_currentStrategyEntry;
```

```
        [SerializeField]
        protected List<SiN.StrategyEntry> m_strategies = new
List<SiN.StrategyEntry>();

        [SerializeField]
        private SiN.TagsContainer m_unitTags;

        public bool _canDoStuff = true;
        public bool _isControlledByPlayer = false;

        public event Action<BaseUnit> _onDied;

        public StatsSO Stats { set => m_stats = value; get => m_stats;}

        void Start()
        {
            _canDoStuff = true;
        }

        void Update()
        {
            if(!_canDoStuff) return;

            EvaluateConditions();
            m_currentBehavior?.DoLogic(this);
        }

        protected virtual void EvaluateConditions()
        {
```

```
            if(m_currentStrategyEntry != null &&
m_currentStrategyEntry._condition.CheckCondition(this)) return;

            foreach(var strategy in m_strategies)
            {
                if(strategy._condition.CheckCondition(this))
                {
                    ChangeStrategy(strategy._name);
                    return;
                }
            }

            ChangeStrategy(SiN.BehaviorsNames.None);
        }

        public virtual void ChangeStrategy(SiN.BehaviorsNames changeTo)
        {
            if(changeTo == SiN.BehaviorsNames.None)
            {
                m_currentBehavior = null;
                return;
            }

            m_currentBehavior = m_strategies.Find(strat => strat._name ==
changeTo)._logic;
        }

        public virtual void HandleDamage(float damage)
        {
            m_recievedDamage += damage;
```

```
        if(m_recievedDamage >= GetStat(SiN.StatsNames.Health))
        {
            HandleDeath();
        }
    }

    protected virtual void HandleDeath()
    {
        _onDied?.Invoke(this);
    }

    public SiN.TagsContainer GetUnitTags()
    {
        return m_unitTags;
    }

    public float GetStat(SiN.StatsNames statName)
    {
        return m_stats.GetValueFor(statName);
    }

    public List<GameplayTile> GetAttackArea()
    {
        return m_attackArea;
    }

    public virtual bool CanBeAttacked()
    {
        return true;
```

```
        }

        public virtual void RemoveUnit(){}
    }

    using UnityEngine;

    namespace SiN
    {
        public enum BehaviorsNames
        {
            None,
            MoveStrategy,
            FightStartegy
        }
    }


    public abstract class BehaviorStrategy : MonoBehaviour
    {
        public SiN.BehaviorsNames _behaviorName;
        public abstract void DoLogic(BaseUnit unit);
    }

    using System;
    using System.Collections.Generic;
    using UnityEngine;

    public class BaseUnit : MonoBehaviour
    {
```

```
        [SerializeField]
        protected StatsSO m_stats;

        protected float m_recievedDamage;

        protected List<GameplayTile> m_attackArea = new
List<GameplayTile>();

        protected BehaviorStrategy m_currentBehavior;

        protected SiN.StrategyEntry m_currentStrategyEntry;

        [SerializeField]
        protected List<SiN.StrategyEntry> m_strategies = new
List<SiN.StrategyEntry>();

        [SerializeField]
        private SiN.TagsContainer m_unitTags;

        public bool _canDoStuff = true;
        public bool _isControlledByPlayer = false;

        public event Action<BaseUnit> _onDied;

        public StatsSO Stats { set => m_stats = value; get => m_stats;}

        void Start()
        {
            _canDoStuff = true;
        }
```

```
        void Update()
        {
            if(!_canDoStuff) return;

            EvaluateConditions();
            m_currentBehavior?.DoLogic(this);
        }

        protected virtual void EvaluateConditions()
        {
            if(m_currentStrategyEntry != null &&
m_currentStrategyEntry._condition.CheckCondition(this)) return;

            foreach(var strategy in m_strategies)
            {
                if(strategy._condition.CheckCondition(this))
                {
                    ChangeStrategy(strategy._name);
                    return;
                }
            }

            ChangeStrategy(SiN.BehaviorsNames.None);
        }

        public virtual void ChangeStrategy(SiN.BehaviorsNames changeTo)
        {
            if(changeTo == SiN.BehaviorsNames.None)
            {
```

```
            m_currentBehavior = null;
            return;
        }

        m_currentBehavior = m_strategies.Find(strat => strat._name ==
changeTo)._logic;
    }

    public virtual void HandleDamage(float damage)
    {
        m_recievedDamage += damage;

        if(m_recievedDamage >= GetStat(SiN.StatsNames.Health))
        {
            HandleDeath();
        }
    }

    protected virtual void HandleDeath()
    {
        _onDied?.Invoke(this);
    }

    public SiN.TagsContainer GetUnitTags()
    {
        return m_unitTags;
    }

    public float GetStat(SiN.StatsNames statName)
    {
```

```
        return m_stats.GetValueFor(statName);
    }

    public List<GameplayTile> GetAttackArea()
    {
        return m_attackArea;
    }

    public virtual bool CanBeAttacked()
    {
        return true;
    }

    public virtual void RemoveUnit(){}
}
```

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

public class GameplayField : MonoBehaviour
{
    [SerializeField]
    private List<GameObject> m_tilesContainer = new List<GameObject>();

    [SerializeField]
    private List<GameplayTile> m_roadTiles = new List<GameplayTile>();

    [SerializeField]
```

```
        private List<GameplayTile> m_heightTiles = new
List<GameplayTile>();

        [SerializeField]
        private List<GameObject> m_entrances = new
List<GameObject>();

        [SerializeField]
        private List<GameObject> m_exits = new List<GameObject>();

        [SerializeField]
        private int m_width;

        public List<GameObject> GetTilesContainer()
        {
            return m_tilesContainer;
        }

        public GameObject GetTileAtIndex(int index)
        {
            if(m_tilesContainer != null && m_tilesContainer.Count > index)
            {
                return m_tilesContainer[index];
            }

            return null;
        }

        public GameObject GetTileAtIndex(int x, int y)
        {
```

```
        return m_tilesContainer[x * m_width + y];
    }

    public GameObject GetTileAtIndex(SiN.Point point)
    {
        return m_tilesContainer[point.x * m_width + point.y];
    }

    public GameObject GetTileAtIndexForAttackCoverage(SiN.Point
point)
    {
        if (point.x < 0 || point.y < 0 || point.y >= m_width)
            return null;

        int index = point.x * m_width + point.y;

        if (index < 0 || index >= m_tilesContainer.Count)
            return null;

        return m_tilesContainer[index];
    }

    public void SetupBaseParameters(int width)
    {
        m_width = width;
    }

    public void DefinePlaceableHeightRoadTiles()
    {
        foreach(var tile in m_tilesContainer)
```

```
            {
                GameplayTile gt = tile.GetComponent<GameplayTile>();
                if(gt.HasTag(SiN.Tags.Road)                                    &&
gt.HasTag(SiN.Tags.TowerPlaceable))
                    m_roadTiles.Add(gt);

                if(gt.HasTag(SiN.Tags.Height)                                  &&
gt.HasTag(SiN.Tags.TowerPlaceable))
                    m_heightTiles.Add(gt);
            }
        }

        public void HighlightTiles(SiN.TagsContainer tags)
        {
            if(tags.HasTag(SiN.Tags.Height))
            {
                foreach(var height in m_heightTiles)
                    height.HighlightSurfaceForPlacement();
            }
            if(tags.HasTag(SiN.Tags.Road))
            {
                foreach(var road in m_roadTiles)
                    road.HighlightSurfaceForPlacement();
            }
        }

        public void UnhighlightTiles(SiN.TagsContainer tags)
        {
            if(tags.HasTag(SiN.Tags.Height))
            {
```

```
            foreach(var height in m_heightTiles)
                height.HidePlacementMarker();
        }
        if(tags.HasTag(SiN.Tags.Road))
        {
            foreach(var road in m_roadTiles)
                road.HidePlacementMarker();
        }
    }}

using UnityEngine;
using System.Collections.Generic;
using System;

public class GameplayTile : MonoBehaviour, IPoint
{
    [SerializeField]
    private SiN.TagsContainer m_tileTags;

    [SerializeField]
    private BaseUnit m_tower;

    private HashSet<BaseUnit> m_enemies = new
HashSet<BaseUnit>();

    [SerializeField]
    private SiN.Point m_pointIndex;

    private GameObject m_placementMarker;
```

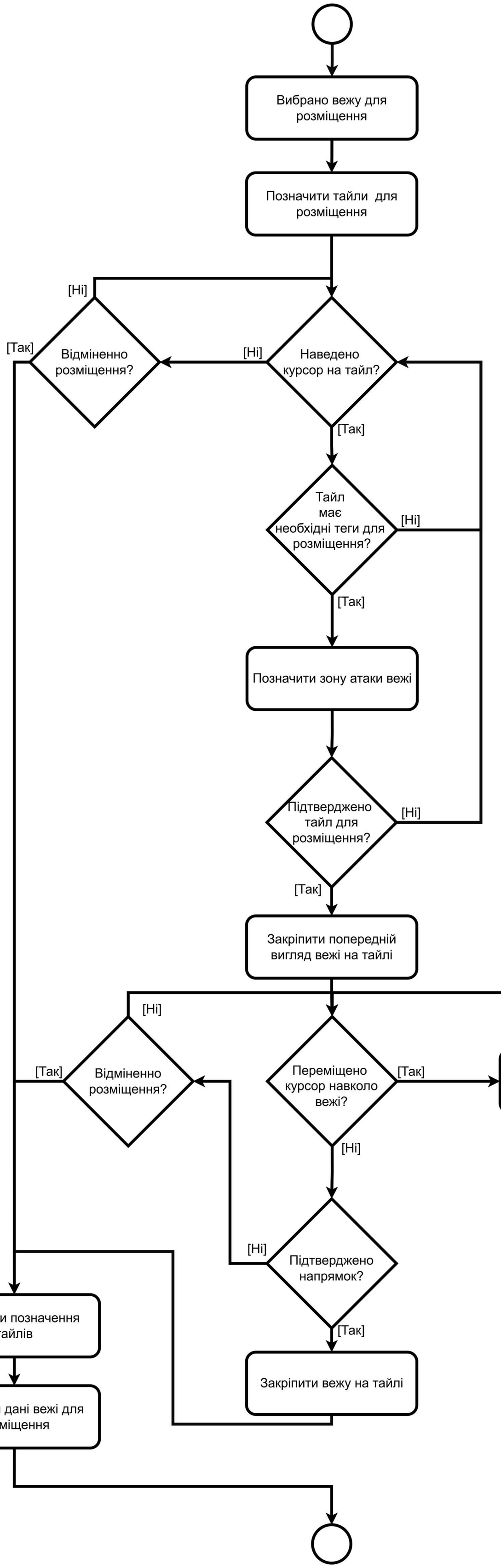
```
        private GameObject m_attackMarker;

        public Vector3 _waypointPosition;

        public Vector3 _entrancePosition;

        public event Action<BaseUnit> _enemyRegistered;

        public event Action<BaseUnit, GameplayTile> _enemyUnregistered;

        public event Action<BaseUnit> _towerRegistered;

        public HashSet<BaseUnit> RegisteredEnemies {get => m_enemies;}

        public void InitChildrenData()
        {
            _waypointPosition = transform.TransformPoint(GetComponentInChildren<Waypoint>().transform.localPosition);
            _entrancePosition = transform.TransformPoint(GetComponentInChildren<SiN.Entrance>().transform.localPosition);

            m_placementMarker = GetComponentInChildren<PlacementMarker>(true).gameObject;
            m_attackMarker = GetComponentInChildren<AttackMarker>(true).gameObject;
        }
```

```
public bool CanUnitBePlaced()
{
    if(m_tower != null) return false;
    if(!m_tileTags.HasTag(SiN.Tags.TowerPlaceable)) return false;

    return true;
}

public void AssignTags(SiN.TagsContainer tags)
{
    m_tileTags = tags;
}

public void SetPointIndexes(int x, int y)
{
    m_pointIndex.x = x;
    m_pointIndex.y = y;
}

public void RegisterAsTower(BaseUnit unit)
{
    m_tower = unit;

    _towerRegistered?.Invoke(m_tower);
}

public void RegisterAsEnemy(BaseUnit unit)
{
    m_enemies.Add(unit);
```

```
            _enemyRegistered?.Invoke(unit);
        }

        public bool HasTag(SiN.Tags tag)
        {
            return m_tileTags.GetTag(tag) == tag;
        }

        public bool HasAnyTag(SiN.TagsContainer tags)
        {
            return m_tileTags.HasSimilarTags(tags);
        }

        public void UnregisterEnemy(BaseUnit unit, GameplayTile nextTile
= null)
        {
            m_enemies.Remove(unit);

            _enemyUnregistered?.Invoke(unit, nextTile);
        }

        public void UnregisterTower()
        {
            m_tower = null;
        }

        public bool HasTower()
        {
            return m_tower? true : false;
```

```
}

public SiN.Point GetPoint()
{
    return m_pointIndex;
}

public Vector3 GetWorldWaypointPosition()
{
    return _waypointPosition;
}

public Vector3 GetWorldEntrancePosition()
{
    return _entrancePosition;
}

public void HighlightSurfaceForPlacement()
{
    if(!CanUnitBePlaced() || !m_placementMarker) return;
    m_placementMarker.SetActive(true);
}

public void HidePlacementMarker()
{
    if(m_placementMarker)
        m_placementMarker.SetActive(false);
}

public void HighlightSurfaceForAttack(bool highlight = true)
```

```
        {
            if(!m_attackMarker) return;
            m_attackMarker.SetActive(highlight);
        }

        public bool CanAttackThisTile(SiN.TagsContainer tags)
        {
            return HasAnyTag(tags);
        }
    }
```



					2026.КРБ.123.703.08.00.00 БС	
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка комп'ютерної гри «SiN» з використанням C# UML – діаграма послідовності дій	Літ. / Маса / Масштаб
Розроб.		І. ЛЕНЧУК				
Перевір.		І. ГЕНИК				Аркуш 1 / Аркушів 1
Реценз.						
Н. Контр.		В. ПРИЙМАК				ВСП "ТФК ТНТУ імені Івана Пулюя" КІб – 703п
Затверд.						

override void MoveableStrategyDoLogic(BaseUnit unit)

Початок методу

if(m_movable Reference == null)

[Так]

[Ні]

float speed = unit.GetStat(SiN.StatsNames.Speed) * Time.deltaTime;

Vector3 currentPosition = m_transform.position;

Vector3 nextPosition = m_movableReference.GetNextPositionVector();

Vector2 difference = nextPosition - currentPosition;

float distanceToNextPoint = Vector2.SqrMagnitude(difference);

float halfDistanceBetweenPoints = (Vector2.SqrMagnitude(m_movableReference. GetCurrentTilePosition() - nextPosition)) / 2;

if(distance ToNextPoint <= halfDistance BetweenPoints)

[Так]

[Ні]

m_movableReference.RegisterInNextTile();

if(m_movableReference .IsCurrentTile Occupied())

[Так]

[Ні]

m_movableReference?.SetCurrentPosition (Vector3.MoveTowards(currentPosition, nextPosition, speed));

if (distanceToNextPoint < m_distance ThresholdSqr)

[Так]

[Ні]

m_movableReference.ChangeNextPositionVector();

if(m_movableReference .IsFinished())

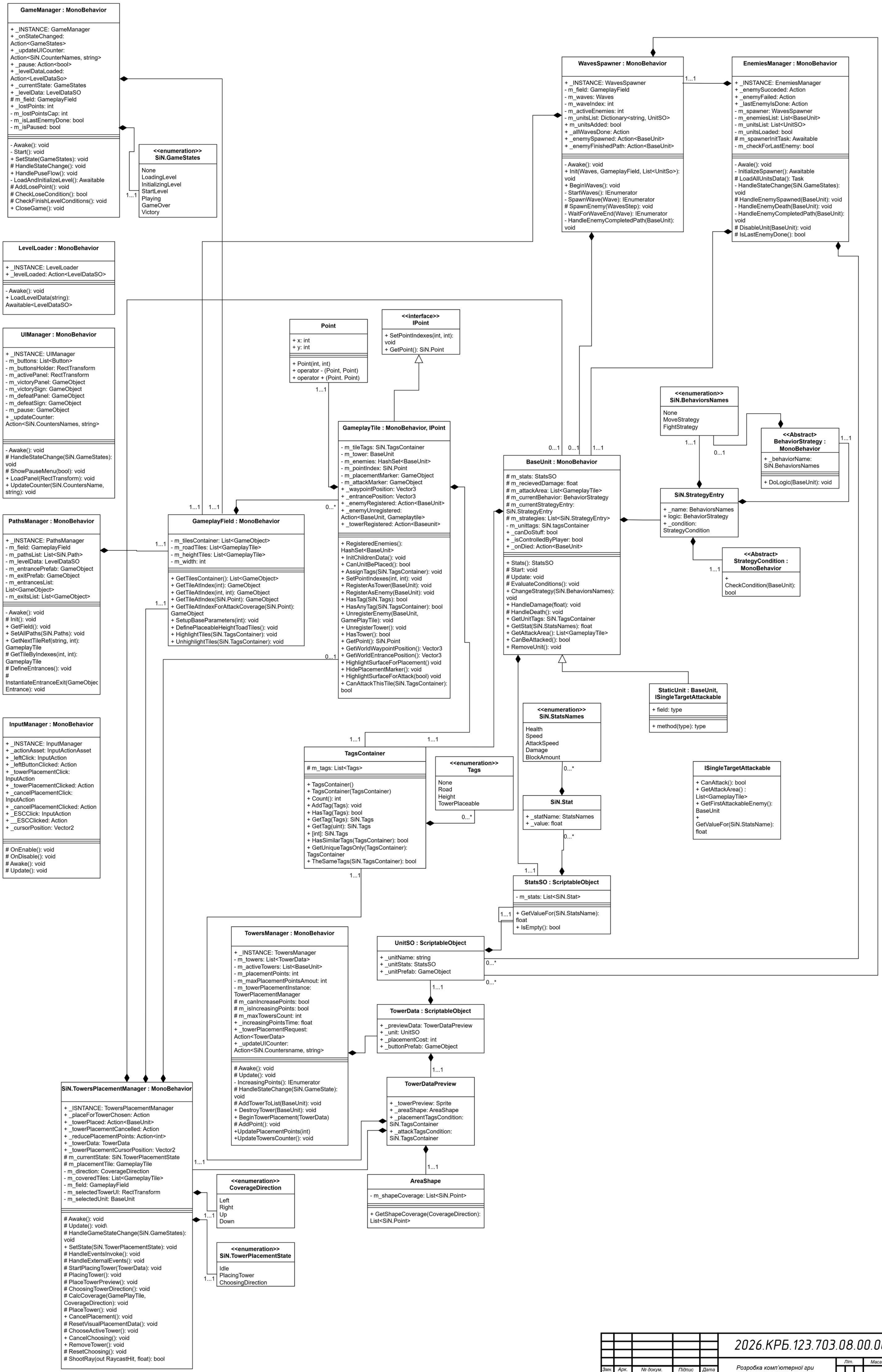
[Так]

[Ні]

m_movableReference.OnPathCompleted();

Кінець методу

					2026.КРБ.123.703.08.00.00 БС	
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка комп'ютерної гри «SiN» з використанням C# UML – блок-схема алгоритму методу	Літ. / Маса / Масштаб
Розроб.		І. ЛЕНЧУК				
Перевір.		І. ГЕНИК				Аркуш 1 / Аркушів 1
Реценз.						
Н. Контр.		В. ПРИЙМАК				ВСП "ТФК ТНТУ імені Івана Пулюя" КІб – 703п
Затверд.						



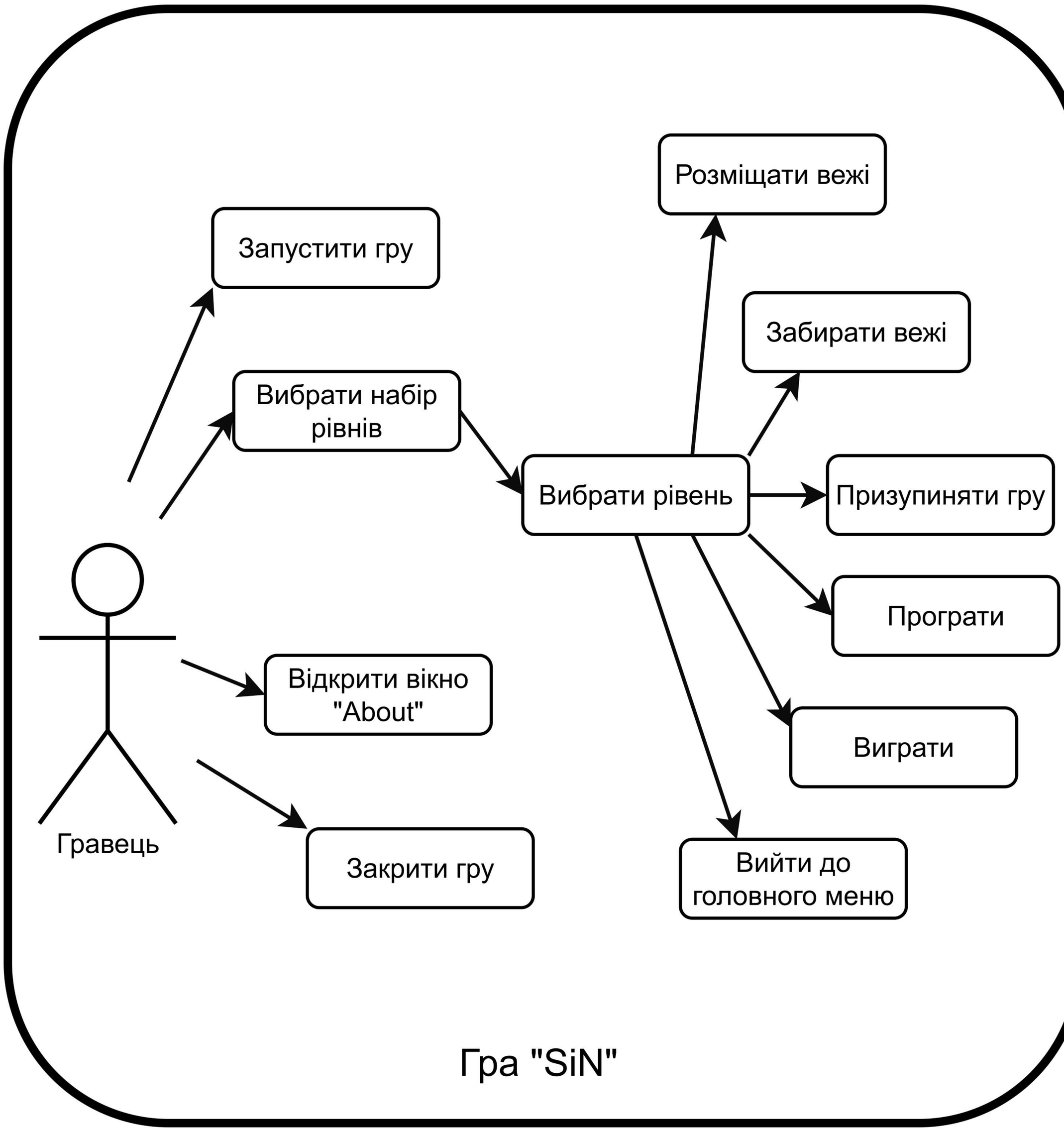
2026.КРБ.123.703.08.00.00 ДК

Змн.	Арк.	№ докум.	Підпис	Дата	2026.КРБ.123.703.08.00.00 ДК	
Розроб.		І. ЛЕНЧУК			Розробка комп'ютерної гри «SiN» з використанням C# UML-діаграма класів програми	Літ. / Маса / Масштаб
Перевір.		І. ГЕНИК				Аркуш 1 / Аркушів 1
Реценз.						ВСП "ТФК ТНТУ імені Івана Пулюя" КІб – 703п
Н. Контр.		В. ПРИЙМАК				
Затверд.						

```
public override void DoLogic(BaseUnit unit)
    {
        if(m_movableReference == null) return;

        float speed = unit.GetStat(SiN.StatsNames.Speed) * Time.deltaTime;

        Vector3 currentPosition = m_transform.position;

        Vector3 nextPosition = m_movableReference.GetNextPositionVector();

        Vector2 difference = nextPosition - currentPosition;

        float distanceToNextPoint = Vector2.SqrMagnitude(difference);

        float halfDistanceBetweenPoints =
(Vector2.SqrMagnitude(m_movableReference.GetCurrentTilePosition() -
nextPosition)) / 2;

        if(distanceToNextPoint <= halfDistanceBetweenPoints)
        {
            m_movableReference.RegisterInNextTile();

            if(m_movableReference.IsCurrentTileOccupied())
                return;
        }


m_movableReference?.SetCurrentPosition(Vector3.MoveTowards(currentPosition,
nextPosition, speed));

        if(distanceToNextPoint < m_distanceThresholdSqr)
        {
            m_movableReference.ChangeNextPositionVector();
        }

        if(m_movableReference.IsFinished())
        {
            m_movableReference.OnPathCompleted();
        }
}
```

					2026.КРБ.123.703.08.00.00 ТП	
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка комп'ютерної гри «SiN» з використанням C# Текст методу	Літ. / Маса / Масштаб
Розроб.		І. ЛЕНЧУК				
Перевір.		І. ГЕНИК				Аркуш 1 / Аркушів 1
Реценз.						
Н. Контр.		В. ПРИЙМАК				ВСП "ТФК ТНТУ імені Івана Пулюя" КІб – 703п
Затверд.						

2026.КРБ.123.703.08.00.00 ДВ



Гра "SiN"

					2026.КРБ.123.703.08.00.00 ДВ	
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка комп'ютерної гри «SiN» з використанням C# UML-діаграма діаграма варіантів використання програми	Літ. / Маса / Масштаб
Розроб.		І. ЛЕНЧУК				
Перевір.		І. ГЕНИК				Аркуш 1 / Аркушів 1
Реценз.						
Н. Контр.		В. ПРИЙМАК				ВСП "ТФК ТНТУ імені Івана Пулюя" КІб – 703п
Затверд.						

Таблиця техніко-економічних показників

Технічні показники			Економічні показники			
№ п.п.	Назва показників	Значення	№ п.п.	Назва показників	Одиниці вимірювання	Значення
1	Операційна система	Windows 10	6	Собівартість розробки	грн.	296 408. 03
2	Загальний об'єм сайту	30 Мб	7	Плановий прибуток	грн.	165 988. 50
3	Середовище розробки	Visual Studio Code	8	Ціна обслуговування	грн.	43 982, 40
4	Базаві мови програмування	C#	9	Економічна ефективність	—	1
5	Тип проекуту	Власна розробка	10	Термін окупності	рік	1. 8

Перш. застос.

Справ. №

Підп. і дата

Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № подл.

Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		І. Ленчик		
Перев.		І. Геник		
Т.контр.				
Н.контр.		Прийомак В.А.		
Затв.				

2026.КВР.123.703.08.00.00 ТБ

Розробка комп'ютерної гри «SiN» з використанням C#
Таблиця техніко-економічних показників

Літ.	Маса	Масштаб
Аркуш	Аркушів 1	

ВСП ТФК ТНТУ гр.КІБ-703
м. Тернопіль

Копіював

Формат А1