Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти

на тему

“Дослідження та програмна реалізація системи процедурної генерації контенту комп’ютерної гри”

Виконав здобувач вищої освіти
ІІ курсу, групи _____
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Мельник С.О.
«____» _____ 20 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Роман МИНАЙЛЕНКО
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *123 "Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
« __ » _____ 20__ року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Мельнику Сергію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи процедурної генерації контенту комп'ютерної гри*

2. Керівник роботи *Минайленко Роман Миколайович, кандидат техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №19-13 від 17.08.2022 року

3. Строк подання роботи до захисту *10.12.2022 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Дослідження та програмна реалізація системи процедурної генерації контенту комп'ютерної гри*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Діаграма процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	25.10.2022	11.11.2022
Охорона праці	Оришака О.В., к.т.н., доцент	04.11.2022	21.11.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2022 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2022 р.	
3.	Розробка моделі компонента	20.10.2022 р.	
4.	Розробка структур даних	25.10.2022 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2022 р.	
6.	Програмування алгоритмів	10.11.2022 р.	
7.	Розрахунок економічної ефективності	13.11.2022 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2022 р.	
9.	Оформлення ПЗ	17.11.2022 р.	
10.	Попередній захист роботи	10.12.2022 р.	

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Мельник С.О. Дослідження та програмна реалізація системи процедурної генерації контенту комп'ютерної гри. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації системи процедурної генерації контенту комп'ютерної гри.

Метою розробки є дослідження та програмна реалізація системи процедурної генерації контенту комп'ютерної гри.

Об'єктом дослідження є процес процедурної генерації контенту комп'ютерної гри.

Предметом дослідження є методи та алгоритми процедурної генерації та комп'ютерної графіки для комп'ютерних ігор.

Методи дослідження базуються на методах штучного інтелекту, методах комп'ютерної графіки, методах розробки програмного забезпечення, методах розробки комп'ютерних ігор.

Результат роботи – програмна реалізація системи процедурної генерації контенту комп'ютерної гри.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на мові програмування C# у ігровому рушії Unity.

Ключові слова: комп'ютерна інженерія, процедурна генерація, комп'ютерна графіка, комп'ютерна гра.

ABSTRACT

Melnyk S.O. Research and software implementation of the computer game content procedural generation system. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2022.

In this master's thesis, software designed for a system for content procedural generation of a computer game.

The purpose of the development is the research and program implementation of a system for content procedural generation of a computer game.

The object of research is the process of procedural generation of computer game content.

The subject of research is the methods and algorithms of procedural generation and computer graphics for computer games.

Research methods are based on artificial intelligence methods, computer graphics methods, software development methods, and computer game development methods.

The result of the work is the software implementation of a system for content procedural generation of a computer game.

In the process of working on a software model an analysis of existing hardware and software was performed. All components of the software developed are fully described.

User-friendly user interface is developed. These are instructions for working with software.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the C# programming language in the Unity game engine.

Keywords: computer engineering, procedural generation, computer graphics, computer game

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

NavMesh – система для побудові маршрутів юнітів;

Nemesis – система по актоматичному створенні реалістичного світу через взаємодії з неігровими персонажами;

NPC – Non-Player Character, у ігровій індустрії, юніт, що не підпадає під контроль гравця, та має власну історію та прописаний характер;

OpanAI – система на базі нейронної мережі, що є базою для зародження розробок у сфері штучного інтелекту;

PCG – процедурна генерація;

Unity – Unity Engine, сучасний ігровий рушій, що дозволяє створювати від невеликих ігрових проектів до об’ємних розробок великих компаній;

ГПУ – графічний процесор;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер;

ЦПУ – центральний процесор.

ВСТУП

Актуальність теми. Сучасні відкриті світи в іграх – це величезні простори з реалістичними лісами та полями, містами та селами, стежками та багатосмуговими шосе. Найчастіше вони виглядають настільки вражаюче, що гравці всерйоз милуються віртуальними просторами та роблять десятки скріншотів. Фотографуючи чергову галявину у світанкових променях, гравець може і не здогадуватися, що її було згенеровано автоматично. Процедурна генерація в ігровій промисловості використовується давно. *Процедурна генерація* – це автоматичне створення ігрового контенту за допомогою різних алгоритмів, ігровий контент створюється або повністю самостійно системою, або у взаємодії з гравцями чи геймдизайнерами. Наразі відкриті світи стали настільки великими та опрацьованими, що створювати їх вручну дуже довго та дорого. Багато студій, намагаючись уникнути зриву термінів та позапланових розширень штату, використовують процедурну генерацію.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи процедурної генерації контенту комп'ютерної гри.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Дослідження існуючих системи процедурної генерації контенту комп'ютерної гри.

– Розробка методів та алгоритмів системи процедурної генерації контенту комп'ютерної гри.

– Програмна реалізація системи процедурної генерації контенту комп'ютерної гри.

Об'єктом дослідження є процес процедурної генерації контенту комп'ютерної гри.

Предметом дослідження є методи та алгоритми процедурної генерації та комп'ютерної графіки для комп'ютерних ігор.

Методи дослідження базуються на методах штучного інтелекту, методах комп'ютерної графіки, методах розробки програмного забезпечення, методах розробки комп'ютерних ігор.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Удосконалено метод процедурної генерації контенту комп'ютерної гри, що відрізняється від існуючих застосуванням еволюційних методів машинного навчання для ускладнення ігрових рівнів для гравця з часом.

2. Розроблено вітчизняний продукт процедурної генерації контенту комп'ютерної гри, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний для прискореної розробки комп'ютерних ігор.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі процедурної генерації контенту комп'ютерної гри.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація експертної системи для управління персонажами у комп'ютерній грі, є актуальною задачею, яка потребує вирішення у даній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Процедурна генерація (PCG) застосовується у комп'ютерних іграх для створення ігрового контенту та його деталей без обов'язкового втручання розробника. Ця техніка широко використовується в таких іграх, в основі яких лежать відкриті світи. Користувач може як завгодно довго досліджувати такі світи та взаємодіяти з ними.

Процедурна генерація світів принципово перевершує генерацію, що виконується вручну – оскільки на розробку потрібно менше часу та менше витрат. Розміри файлів, що завантажуються, зазвичай вдається істотно зменшити, оскільки при встановленні в грі відсутні величезні заздалегідь побудовані карти.

Як характерні приклади використання PCG називаються: генерація без участі людини підземель у пригодницьких іграх, що отримує новий світ на кожному запуску; система, що створює нові типи озброєнь у шутері у космічному сеттингу залежно від дій гравця; генерація повної, грабельної та збалансованої настільної гри; заповнення ігрового світу рослинами тощо.

Метою використання PCG може бути створення ігрового контенту без участі людини (що може бути як менш витратним, так і допомагати геймдизайнерам у вирішенні їхніх завдань), розробка інших типів ігор (покращення показників різноманітності та реграбельності), адаптація ігор під гравця «на льоту», покращення контенту за допомогою алгоритмічних рішень, а також формалізація геймдизайну як широке наукове завдання.

Процедурна генерація найбільш часто використовується в рольових комп'ютерних іграх та масових багатокористувацьких рольових онлайн-іграх.

1.2 Область застосування

Процедурна генерація контенту може використовуватися практично у всіх жанрах комп'ютерних ігор, а також для швидкого створення різних просторів у фільмах.

Види контенту, що можна генерувати:

– 3D-моделі, текстури, ефекти.

– Анімація.

– Звуки та музика.

– Рівні, ландшафти, будівлі.

– Персонажі та предмети.

– Поведінка персонажів.

– Сюжет та завдання.

– Тощо.

Причини використання процедурної генерації:

– Низька вартість – дозволяє генерувати контент в обсязі, який було б дорого (або неможливо в принципі) створювати вручну.

– Різноманітність – дивує, щоразу створювати новий контент.

– Адаптація під гравця – змінює контент, що генерується під уподобання користувачів.

– Тестування – генерує великі набори даних для тестування алгоритмів розроблюваної комп'ютерної гри.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми магістерської роботи

Процедурна генерація має в собі елемент випадковості. Алгоритм враховуватиме набір правил і випадковим чином генеруватиме контент. А створюючи контент вручну, ми маємо обмеження щодо того, чого можемо досягти за обмежений час.

Отже, в ідеалі, якщо комп'ютерна гра має певний сюжет і обмежену кількість рівнів, можна зосередитися на ручному створенні вмісту та внесенні в нього деталей для кращої взаємодії з користувачем, а не використовувати алгоритми для створення графіки.

З іншого боку, якщо комп'ютерна гра схожа на відкритий світ, де ми заохочуємо гравця одержувати новий досвід, наприклад, жити своїм життям, процедурна генерація є нашим найкращим вибором.

Крім того, процедурна генерація дійсно виділяється, коли є обмежений простір, наприклад, заповнюючи гру у 8-бітному чіпі. Розробники ігор не можуть створювати занадто багато рівнів вручну та вставляти мікросхему. Це просто з'їсть забагато пам'яті. Таким чином, вони використовують алгоритми процедурної генерації для створення випадкових карт на льоту, коли ігри завантажуються. Ця техніка використовує менше пам'яті, а також робить вміст менш передбачуваним, підсилюючи елемент веселості.

Випадки використання процедурної генерації в розробці ігор

Оскільки з часом розробникам ігор разом із геймерами стало доступно все більше вдосконаленого апаратного забезпечення, розробники почали використовувати процедурну генерацію для розробки доступності здобичі і нагород у грі. Наприклад, якщо герой йде по ворожій території, різна зброя,

нагороди та здобич місії будуть доступні героєві у випадкових місцях щоразу через процедурну генерацію.

Ще одне застосування процедурної генерації може бути наступним – ґрунтуючись на поведінці користувача у грі, логіка гри процедурно генерує ШІ-персонажів, щоб користувач залишався емоційно залученим у гру.

Розробники ігор використовують процедурну генерацію при проектуванні рівнів своєї гри. На основі неї гравець переходить до спеціальних процедурно згенерованих рівнів, щоб забезпечити захоплюючий ігровий досвід.

Процедурна генерація також використовується разом з обробкою природної мови для створення реалістичних сюжетів в іграх. Зазвичай реалізується в текстових іграх.

Сторонні інструменти та бібліотеки процедурної генерації

Розробникам ігор також доступні кілька сторонніх інструментів і движків/генераторів світу, які допомагають генерувати такий вміст, як ліси, ландшафти тощо. Модель окремої сосни можна використовувати для процедурного створення цілого соснового лісу. Це дійсно прискорює процес розробки, оскільки тепер розробник може зосередитися на історії, а не витрачати час на проектування ландшафтів і лісів.

Одними з дійсно популярних у галузі розробки комп'ютерних ігор додатками для процедурної генерації є Speed Tree та Gaia.

Speed Tree – це програмне забезпечення для моделювання та програмування рослинності, яка використовується в кіно та комп'ютерних іграх. Воно створює в режимі реального часу симуляції рослинності, лісів, листя, ферм, архітектури тощо. Його використовують деякі відомі фірми, як-от Activision, Ubisoft, Epic Games тощо.

Використання спеціалізованої технології SpeedTree, у ряді випадків, допомагає знизити навантаження на ресурси комп'ютера у порівнянні з використанням стандартних засобів ігрового двигуна; SpeedTree значно полегшує створення рослин для дизайнерів.

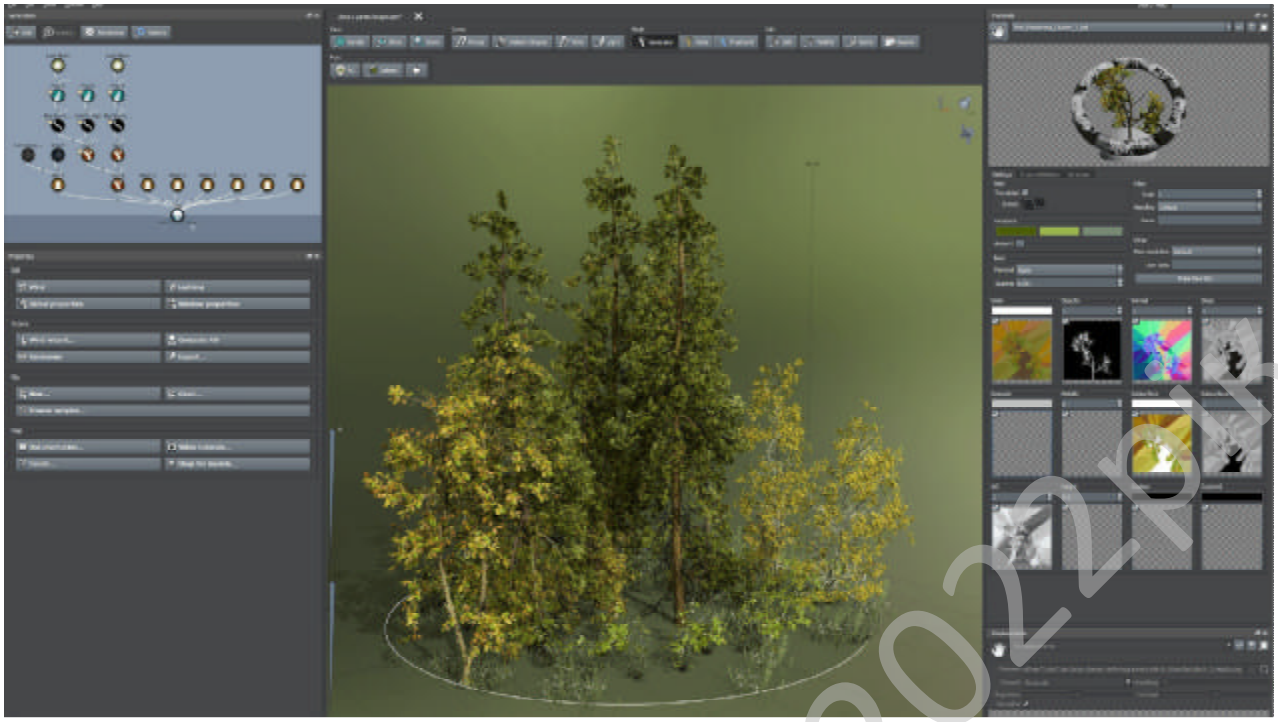

Рисунок 2.1 – Speed Tree – програмне забезпечення для процедурної генерації рослинності

Gaia – це система створення рельєфу та сцени для Unity 3D. Розробники ігор можуть використовувати передові алгоритми системи, щоб створювати прекрасні світи для своїх ігор.

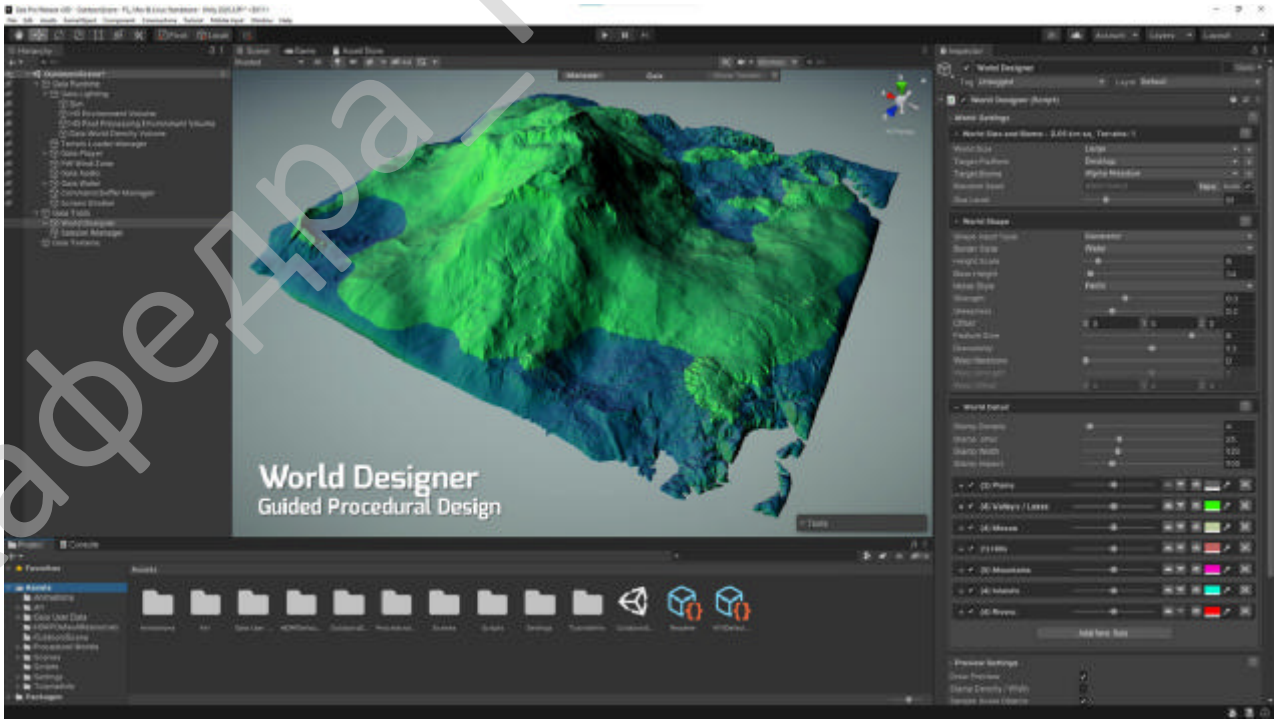

Рисунок 2.2 – Gaia – система процедурної генерації рельєфу для Unity 3D

Gaia заощаджує час, гроші та розчарування, об'єднуючи те, що зазвичай було б багатьма дорогими та складними системами, в одну просту систему на основі майстра, та автоматизує більшість зусиль, необхідних для створення придатних для гри рівнів.

Gaia підтримує алгоритми штучного інтелекту та машинного навчання.

Gaia використовується в освіті, симуляції та багатьох іграх, таких як Zenith, Last Epoch і Crowfall. Навіть NASA використовує Gaia. Gaia незмінно був одним із найпопулярніших інструментів у магазині асетів і отримав нагороду «Найкращий художній інструмент» на Unity Awards 2020.

Основні риси Gaia:

– Підтримка різних ландшафтів для великих середовищ.

– Налаштування за допомогою майстра, оптимізоване для цільової платформи.

– Модульний дизайн.

– Водна система з шейдерами води та підводними ефектами.

– Стандартний підхід Unity для максимальної сумісності з іншими інструментами.

Концепції та алгоритми процедурної генерації

Процедурна генерація PCG може бути складною річчю, її часто тісно використовують у поєднанні зі штучним інтелектом та глибоким навчанням для досягнення бажаних результатів.

Симуляція штучного життя генетичними алгоритмами

Часто в іграх з управління містом ми спостерігаємо, що разом із взаємодією користувача інші елементи гри самосвідомі та продовжують виконувати свої завдання одночасно з введенням користувача. Ця техніка широко відома як імітація штучного життя або генетичний алгоритм.

Ця процедурна техніка генерації перевіряє стан популяції в грі та продовжує генерувати нових людей на основі правил введення.

Процедурна генерація підземель

Генерація підземель є важливим прикладом процедурної генерації. У підземеллі створюються кілька кімнат розміщені на сітці, а потім намагаються з'єднатися одна з одною через коридори.

Схеми коридорів, що з'єднують кімнати, генеруються процедурно, і з такою кількістю комбінацій і можливостей кінцевий дизайн може стати справді складним. Чим вище складність, тим більше задоволення. Цей підхід до створення підземель перевіряється, якщо кімнати з'єднані правильно за допомогою «охоплюючого дерева».

Подібний алгоритмічний підхід використовується в генерації міста. Де макет сітки та транспортні засоби, що рухаються, генеруються процедурно.

Цей підхід також більш відомий як створення лабіринту. Зазвичай використовується в усіх пригодницьких іграх. Лабіринт можна пройти за допомогою алгоритмів пошуку в глибину або в ширину на графах.

Моделювання погоди за допомогою ланцюгів Маркова

Ланцюг Маркова є ймовірнісним моделюванням. Прогнозування майбутньої події на основі інформації про поточну подію. Ланцюги Маркова мають широке застосування в багатьох областях, таких як фізика, хімія, фінанси, ігри, генетика, теорія масового обслуговування тощо.

Алгоритм PageRank пошукової системи Google також базувався на марковському процесі.

За допомогою моделі ланцюга Маркова процедурно генеруються симуляції погоди. Таким чином погодні умови в грі менш передбачувані та додають більше задоволення гравцеві.

Створення та розповсюдження пожежі за допомогою процедурної генерації

Процедурні алгоритми генерації використовуються для створення диму та вогню в іграх, фільмах, а також для поширення вогню в навколишньому середовищі. Імітація спалаху та горіння будівель і будинків.

Алгоритм/концепція, яка використовується для досягнення цього, відома як Cellular Automata.

Cellular Automata означає двовимірну сітку з *n* кількістю комірок. Кожна клітинка матиме певний стан, і стан клітинки може впливати на стан сусідньої клітинки на основі певного набору правил. В ідеалі правила однакові для всіх комірок у сітці, але стан окремих комірок відрізняється.

Клітини можуть змінювати свій стан одночасно, наприклад, пожежа в конкретній комірці може також спричинити пожежу в сусідніх комірках, таким чином змінюючи їхній стан.

Створення унікальних ситуацій для героя в текстових іграх у поєднанні з обробкою природної мови

Останнім часом текстові ігри набувають великої популярності, особливо ігри, інтегровані з популярними програмами для обміну повідомленнями. Текстові ігри – це ігри, які використовують текстові символи для взаємодії з гравцями замість векторної графіки. Вони більше нагадують життєвий досвід спілкування з реальною людиною.

Герой у грі стикається з унікальними безпрецедентними ситуаціями, створеними за допомогою процедурної генерації в поєднанні з обробкою природної мови. Це підсилює елемент веселощів у грі завдяки виїмкам.

Еволюційні алгоритми для процедурної генерації

Генетичні алгоритми не генерують контент, але вони можуть бути використані для спрямування процедурного генератору до більш обмеженого заповнення та бажаного змісту.

Генетичний алгоритм – це евристика пошуку, натхненна теорією природної еволюції Чарльза Дарвіна. Цей алгоритм відображає процес природного відбору, коли для розмноження відбираються найпристосованіші особини з метою отримання нащадків наступного покоління.

На рис. 2.3 показано структурні елементи генетичного алгоритму. Основними елементами еволюційних алгоритмів є ген, хромосома та популяція.

Рисунок 2.3 – Структурні елементи еволюційних алгоритмів

Поняття природного відбору. Процес природного відбору починається з відбору найбільш пристосованих особин із популяції. Вони дають потомство, яке успадковує характеристики батьків і буде додано до наступного покоління. Якщо батьки мають кращу фізичну форму, їхні нащадки будуть кращими за батьків і матимуть більше шансів вижити. Цей процес продовжує повторюватися, і в кінці буде знайдено покоління з найбільш підготовленими особами. Це поняття можна застосувати до проблеми пошуку найкращого рішення при процедурній генерації. Ми розглядаємо набір рішень проблеми і вибираємо з них найкращі. У генетичному алгоритмі розглядаються п’ять фаз:

1. Ініціалізація початкової популяції.
2. Застосування фітнес-функції.
3. Відбір найбільш пристосованих особин.
4. Кросовер відібраних особин.
5. Мутація нових особин, створених кросовером.

Ініціалізація початкової популяції. Процес починається з набору особин, який називається популяцією. Кожна особина – це рішення проблеми, яку треба вирішити. Особина характеризується набором параметрів (змінних), відомих як гени . гени об’єднуються в рядок, утворюючи хромосому. У генетичному алгоритмі набір генів індивіда представлений за допомогою рядка в термінах

алфавіту. Зазвичай використовуються двійкові значення (рядок з 1 і 0).

Фітнес-функція. Фітнес-функція визначає, наскільки придатна особина (рішення проблеми). Імовірність того, що особина буде відібрана для розмноження, залежить від оцінки її придатності.

Відбір найбільш придатних особин. Ідея етапу відбору полягає в тому, щоб відібрати найбільш підготовлених особин і дати їм можливість передати свої гени наступному поколінню. Дві пари осіб (батьки) вибираються на основі їх фітнес-оцінки. Особини з високою пристосованістю мають більше шансів бути відібраними для розмноження.

Кросовер. Кросовер є найважливішою фазою в генетичному алгоритмі. Для кожної пари батьків, які підлягають спаровуванню, точка кросинговеру вибирається випадковим чином із генів. Потомство створюється шляхом обміну генами батьків між собою до досягнення точки кросинговеру. Нове потомство додається до популяції.

Мутація. У деяких нових нащадків, які утворюються, деякі їхні гени можуть бути піддані мутації з низькою випадковою ймовірністю. Це означає, що деякі біти в рядку бітів можна перевертати.

Мутація відбувається для підтримки різноманітності в популяції та запобігання передчасній конвергенції.

Припинення еволюційного алгоритму. Алгоритм припиняє роботу, якщо популяція зближується (не дає потомства, яке значно відрізняється від попереднього покоління). Тоді кажуть, що генетичний алгоритм забезпечив набір рішень нашої проблеми.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Unity – кросплатформове середовище розробки комп’ютерних ігор. Unity дозволяє створювати додатки, що працюють під більш ніж 20 різними

операційними системами, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-додатки та інші. Випуск Unity відбувся в 2005 році і з того часу йде його постійний розвиток.

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі.

Також Unity підтримує фізику твердих тіл і тканини, ragdoll. У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

Редактор Unity підтримує написання і редагування шейдерів. Редактор Unity має компонент для створення анімації, але також анімацію можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

Unity 3D підтримує систему Level Of Detail (скор. LOD), суть якої полягає в тому, що на далекій відстані від гравця високодеталізовані моделі замінюються на менш деталізовані, і навпаки, а також систему Occlusion culling, суть якої в тому, що у об'єктів, що не потрапляють в поле зору камери не візуалізується геометрія і колізія, що знижує навантаження на центральний процесор і дозволяє оптимізувати проект.

Не можливо не зазначити про наявність величезної бібліотеки асетів і плагінів, за допомогою яких можна значно прискорити процес розробки гри. Їх можна імпортувати і експортувати, додавати в гру цілі заготовки - рівні, ворогів, патерни поведінки. Багато асетів надаються безкоштовно, інші пропонуються за невелику суму, і при бажанні можна створювати власний контент, публікувати його в Unity Asset Store.

Проте створення масштабних, складних сцен з великою кількістю компонентів може негативно вплинути на продуктивність гри, в результаті чого розробникам доведеться витратити додатковий час і ресурси на оптимізацію, а

можливо - і видалення деяких елементів з проекту.

Крім того, додатки, створені на Unity, досить «великовагові»: навіть найпростіша піксельна гра може займати кілька сотень мегабайт на персональному комп'ютері. Так, для жорсткого диска це невеликий обсяг, але, якщо проект розробляється і для мобільних платформ, слід задуматися про оптимізацію його розміру.

На даний момент Unity підтримує дві основні мови програмування. Це за замовченням C#, а також JavaScript. Для розробки була обрана саме мова C#.

C# це об'єктно-орієнтована мова програмування зі строгою системою типизації, а також без прямого доступа до пам'яті. Синтаксис даної мови програмування є близьким до C++ та Java. Вона підтримує поліморфізм, наслідування, перевантаження операторів, тощо. Хоча ця мова не має прямого доступу до пам'яті але потрібно її звільняти і цим займається автоматичний збірник сміття. Він функціонує таким чином, що якщо деякий об'єкт перестає використовуватися то посилання да деяку ділянку пам'яті очищається, однак це не приводить до очистки самої ділянки у стеку, і в цей момент у гру вступає очисник, що відсліджує, що на дану ділянку посилань немає і очищає її.

Недоліком у використанні C# є те, що у ньому відсутня реалізація множинного наслідування класів, хоча є множинне наслудівання інтерфейсів.

Що ж до використання цієї мови програмування у ігровому рушії Unity, то кожен класс, він же у просторі рушія являється скриптом, який буде поміщений на сцену має наслідувати основний клас Unity – MonoBehaviour, або наслідуватися від класа або інтерфеса, що наслідує даний основний клас. Основний клас рушія включає в себе доступ до взаємодіє з ігровими обьєктами та їх компонентами такими як Transform, MeshRenderer, RigidBody, тощо, а також включає в себе основні методи Unity, що викликаються при деяких діях. Наприклад, старт програми, її кінець, упродовж роботи програми, абож на початку та в кінці роботи самого скрипта.

DoozyUI – це власне розширення Unity Editor, яке дозволяє легко керувати

та анімувати професійні користувальницькі інтерфейси – без необхідності знати, як кодувати. Для початківців зручний та масштабований, DoozyUI підходить для будь-чого: від розробників-аматорів до професійних студій програмного забезпечення та ігор.

Управління та анімація інтерфейсу раніше було те, що могли робити лише досвідчені розробники. Завдяки DoozyUI тепер кожен може створити власний інтерфейс Unity самостійно. Незалежно від рівня вашої майстерності, потужні редактори DoozyUI дають вам повний творчий контроль - без необхідності писати один рядок коду.

Переваги Unity:

– Низький поріг входження.

– Кросплатформність.

– Потужність – широкий набір інструментарію надає можливість розробки будь-яких проектів.

– Гнучкість і розширюваність – вам надається можливість не тільки повністю налаштовувати інтерфейс програми, а й створювати власні елементи для швидшої та зручнішої розробки.

– Велике ком'юніті – спільнота фахівців, що працюють з Unity є дуже великою, якщо потрібна допомога – завжди її можна знайти.

– Можливість розробки ігор без знання програмування – сервіс Asset Store, являє собою набір різних елементів, підібравши і «склеївши» які можна створити власний додаток в короткий термін.

2.3 Розгорнута постановка завдання

Наступним пунктом є аналіз поставленої задачі в усіх напрямках та виокремлення основних положень, що потребуватимуть розгляду. Згідно з навчальними матеріалами, а також з технічним завданням на кваліфікаційну роботу, підлягає реалізації програмне забезпечення системи процедурної генерації

контенту комп’ютерної гри. В процесі розробки даного додатку необхідно провести групування виокремлених пунктів у деякий план та паставити правильні задачі. Отже, при постановці завдання були виділені такі пункти плану:

− проведення аналізу існуючих систем, а також виявити їх позитивні та негативні сторони;

− визначити методику створення системи процедурної генерації контенту комп’ютерної гри для використання її у різних сферах. Побудувати структурну та функціональну схему;

− розробити програмне забезпечення для виконання поставленої задачі. Побудувати блок-схеми програми і підпрограм;

− розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

− сформувати висновки про разроблену систему та методики, що були використані при розробці.

Як можна побачити аналіз був проведений, пункти виділені, основна робота, яку необхідно виконати була проаналізована та записана. Наступним прококом для створення системи процедурної генерації контенту комп’ютерної гри, буде реалізація запланованого функціоналу та розробка архітектури програмного забезпечення.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

У даному програмному забезпеченні було вирішено здійснити процедурну генерацію підземель, а також вдосконалити її додаванням еволюційних алгоритмів.

Процедурну генерацію підземель можна здійснити за наступним алгоритмом:

Етап 1. Генеруємо кімнати довільним чином, але так, щоб вони не накладалися одна на одну.

Етап 2. Створюємо граф тріангуляції Делоне для кімнат, наприклад, алгоритмом Боуера-Ватсона. Приклад «сирих даних» після застосування алгоритму Боуера-Ватсона зображено на рис. 3.1.

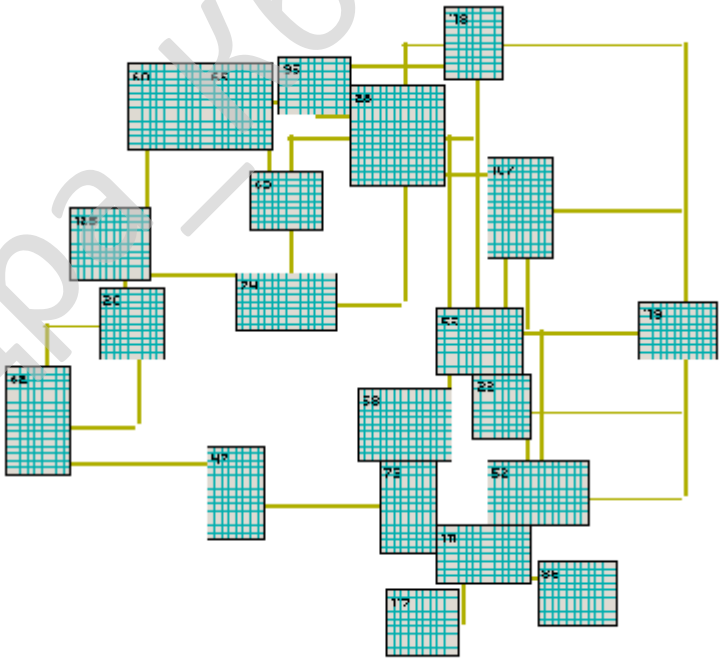

Рисунок 3.1 – Приклад ілюстрації роботи алгоритму Боуера-Ватсона

Етап 3. Створюємо з тріангуляції мінімальне кістякове дерево, наприклад алгоритмом Пріма-Краскала.

Етап 4. Створюємо список коридорів, починаючи з кожного ребра дерева з етапу 3. Дерево містить усі кімнати, тому шлях до кожної кімнати гарантовано існуватиме. Випадковим чином додаємо ребра з тріангуляції до списку. Так ми створимо у коридорах кілька петель. Ймовірність додавання кожного ребра може, наприклад варіюватися від 5% до 30%.

Етап 5. Для кожного коридору у списку використовуємо алгоритм A*, щоб знайти шляхи від початку коридору до кінця. Після знаходження одного шляху він змінює стан світу так, щоб майбутні коридори завжди могли обходити повз існуючі.

Еволюційні алгоритми пропонується застосовувати для ускладнення гри з кожним рівнем. При проходженні гравцем підземель збирається статистика його перемог. Етапи роботи еволюційного алгоритму:

Етап 1. Одержання від підсистеми процедурної генерації контенту множини підземель.

Етап 2. Одержання від статистики попередніх проходжень гри оцінки складності підземель для гравця.

Етап 3. Ініціалізація початкової популяції з множини згенерованих підземель.

Етап 4. Застосування фітнес-функції на основі одержаної статистики попередніх проходжень гри.

Етап 5. Відбір найбільш пристосованих особин – найбільш відповідних поточному рівню гри підземель на основі їх складності.

Етап 6. Кросовер відібраних особин (підземель).

Етап 7. Мутація нових особин (підземель), створених кросовером.

Етап 8. Повтор етапів 2-5 поки не виконана одна із умов зупинки (за точністю чи за часом).

Етап 9. Вибір з останньої популяції найбільш пристосованої особини –

найкращого підземелля для поточного рівня гри. Збереження підземелля у базу даних та використання у поточній грі.

Після етапу аналізу і формулювання вимог до програмного забезпечення проекту виконується створення архітектури, власне процес розробки ПЗ.

Архітектура системи ПЗ включає в себе визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури.

Проектування архітектури системи може проводитися різними методами (стандартизованим, об’єктно-орієнтованим, компонентним і ін.), кожний з яких пропонує свій шлях побудови архітектури, а саме, визначення концептуальної, об’єктної й інших моделей за допомогою відповідних конструктивних елементів.

Моя система, повинна відповідати сталим нормам функціонування ПЗ. Програма повинна мати зручний інтерфейс, користувач повинен одразу зрозуміти, яким чином виконувати ту чи іншу операцію.

Дана система має бути оптимізованою і зрозумілою у використанні, що надає можливість розробникам зробити якісний, зручний, актуальний та захоплюючий проект.

З аналізу задач, огляду методів та програмного забезпечення можна сформулювати вимоги до інформаційної системи.

Система повинна:

– мати заворожуючий геймплей;

– бути оптимізованою;

– мати зрозумілий інтерфейс;

– підтримувати декілька мов;

– бути кросплатфомною.

На початковій сцені гри зображено її назву та кілька функціональних кнопок. Їх призначення – входження та вихід з програми, перехід до налаштувань гри.

При вході у гру, ми опиняємось у безпечній локації – таборі, де головний

герой може приготуватись до подорожі у підземелля. Після проведеної підготовки починається проходження підземелля після його генерації. Підземелля включає в себе кімнати для успішного проходження яких треба виконати різні за характером та складністю завдання.

Отже перша задача створення цієї гри – реалізувати підземелля, яке буде генеруватися при загрузці відповідної сцени за певними параметрами. Щоб додати деякого різноманіття для гравця, цей лабіринт буде створено випадково. Для побудови підземелля враховано такі параметри як:

– кількість кімнат;

– матриця, в якій будуть розміщатися кімнати;

– різнотипність кімнат.

Це основні параметри які відповідають поставленим вимогам до реалізації першої поставленої задачі.

При генеруванні кожної кімнати закладається інформація що стосується власне самої кімнати та її зв’язок з усім підземеллям.

У власне кімнату закладається інформація про її тип, та в майбутньому – згенеровану подію. Крім цього кімната доповнюється інформацією, в якому напрямку є шлях до наступних кімнат підземелля, між якими є взаємозв’язок. З подальшим проходженням користувачем цієї гри, кількість кімнат буде збільшуватися залежно від рівня гравця, а точніше – скільки рівнів він зможе пройти без смертей.

Після побудови цього підземелля, постає нова задача, яка полягає в тому, щоб з’єднати ці кімнати в єдиний конструкт. Це в майбутньому наддасть можливість гравцю пересуватись між кімнатами у обраному напрямку. Щоб показати, який саме напрям слід обрати з кімнати, щоб потрапити в іншу, здійснено візуалізацію для користувача цього шляху у вигляді короткого проходу.

Наступна задача для реалізації – створення персонажу, який при загрузці гри розміщується у стартовій кімнаті підземелля. При натисканні на кнопки керування переміщенням «WASD» здійснював перехід від одної кімнати в іншу.

При цьому здійснювалась перевірка наявності та напряму переходу до іншої кімнати. У випадку коли переходу не знайдено персонаж очікує іншу команду аж поки не знайде правильний шлях переходу. За коректним виконанням цих кроків стежать такі скрипти Generation.cs та MapPlayerController.cs.

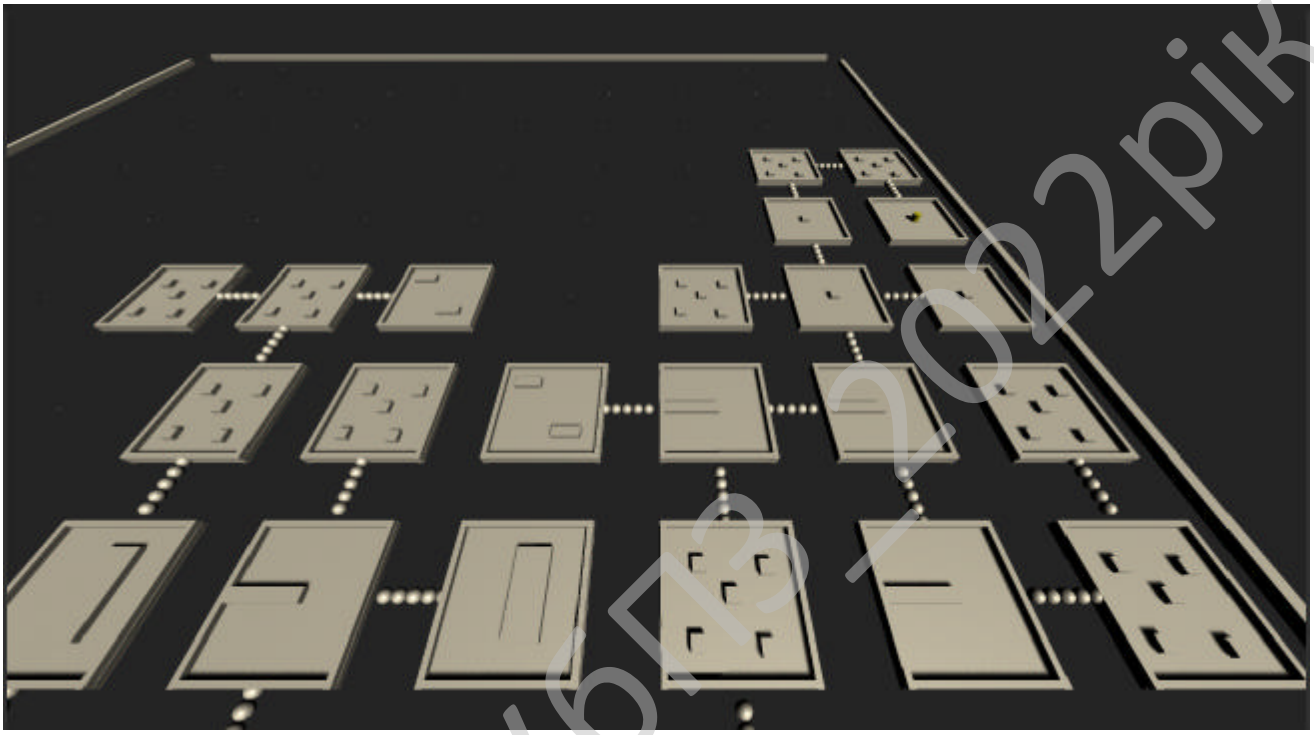

Рисунок 3.2 – Приклад побудови підземелля у розробленому програмному забезпеченні

Оскільки весь конструкт підземелля має параметри значно більші, ніж поле зору камери сцени, був реалізований скрипт CameraController.cs, який надавав би змогу оглянути всі кімнати цього підземелля та скласти певний маршрут переміщення по ньому. Крім звичного переміщення, були реалізовані додаткові можливості камери: зум, обертання та зміна кута камери нахилу для більш детального розгляду кімнат та її візуалізації.

Для урізноманітнення контенту підземелля, була створена певна система подій. За допомогою цієї системи при вході у кімнату з деякою можливістю відбувається рандомна подія, яка впливає певним чином на головного героя. Ця

дія може мати на головного героя як позитивний, так і негативний вплив:

– Збільшення чи зменшення рівня здоров’я героя.

– Ускладнення чи спрощення рівня пробудженості підземелля.

– Подія бою.

За цю механіку відповідає скрипт EventManager.cs. Його суть в тому, щоб при генерації кімнат дія обиралась рандомно. Це може бути подія про:

– Вибір дії при знаходженні артефактів чи спеціальних приміщень.

– Вибір тактики при зустрічі монстрів у різних ситуаціях.

– Вибір поведінки при комунікації з іншими авантюристами.

– Вибір тактики дії при магічному впливі.

– Перехід до бойової дії.

Крім цього створена кімната може і не мати жодного варіанту із наведеного переліку списку подій.

Кількість створених подій також контролюється цим скриптом. Окрім цього, він відповідає за спливаюче вікно, а саме інформацію в ньому, текст самої події, картинку події та кнопки вибору дії.

Вибір дії в кожній ситуації реалізовано через функціонал кнопок запропонованих в кожній події.

У випадку, коли відбувається подія бою, замість текстового вікна з вибором дій, йде перехід безпосередньо на нову сцену. Тут розміщено тільки одну кімнату з головним героєм, для якого відбувається створення різних типів ворогів. Вороги поділяються на 3 типи: вороги ближнього бою, вороги дальнього дальній бій та вороги підтримки нападаючих сил противника.

Кількість ворогів буде залежати від рівня пробудженого підземелля. Для того, щоб вийти з даної кімнати, треба знищити всіх ворогів, у іншому випадку він не зможе завершити бойову подію і продовжити свій похід по сконструйованому підземеллю.

Штучний інтелект ворогів, на даний момент виконує дві головні цілі, знайти гравця та намагатися знищити його з допомогою доступного функціоналу.

Знаходження гравців ворогами працює таким чином: коли гравець ввійде в зону контролю певного противника, то він запам’ятає його, та всіх наступних гравців що перейдуть в цю зону. Після цього штучний інтелект ворога буде ідентифікувати їх, та переміщатись до найближчого, намагаючись при цьому нанести максимальну шкоду герою, в дальньому чи ближньому бою.

Штучний інтелект ворога аналізує не тільки місцезнаходження героя, а й тип, особливості розташування ворога відносно конструкту кімнати для вибору оптимальної тактики ведення бойових дій з врахуванням своїх особливих можливостей.

На даний момент, є такі види ворогів за напрямками:

– Ближній бій (MeleeDash). Представники будуть намагатись таранитись у героя. Вони володіють більшими параметрами швидкості, ніж інші типи ворогів.

– Ближній бій (MeleePunch). Представники не такі швидкі, як перший тип. Вони володіють більшим радіусом атаки. Їх додаткова можливість - постійне обертання, що наносить значну шкоду.

– Дальній бій (Range). Представники відносяться до стрілків, що мають навики точності та регулювання швидкості польотів снарядів.

– Підтримка (Debafer). Представники мають основну ціль – завадити переміщенню та атаці героя з допомогою додаткових умінь. Їх пріоритет не атака, а зниження можливостей швидкості та атаки героя.

– Підтримка (Summoner). Представники мають ціль – триматися поодаль бойових дій та підтримувати оплічників з допомогою виклику декількох помічників із типу дальнього чи ближнього бою. У разі знищення призваних помічників – вони можуть відновити їх.

Рівень пробудженого підземелля залежить від того, скільки проходжень кімнат герой зміг здійснити. При вході у новуы кімнату, заповнюється шкала підземелля. При повному заповненні цієї шкали (5 позначок), піднімається рівень пробудженості підземелля, який впливає на рівень складності проходження.

Герой може вийти із підземелля тільки у тому випадку, коли він дослідить 90% кімнат або переможе всіх генерованих ворогів у кімнатах з бойовими подіями. Якщо ж герой не осилить проходження підземелля та помре, він зможе ожити в таборі та зможе отримати нову спробу підкорення підземелля з самого початку. Але при цьому він втратить набраний раніше прогрес.

Якщо ж гравець зможе завершити проходження всього підземелля, він повертається в зону безпечного табору і для нього згенерується нове підземелля більш ускладненого рівня. В ньому буде більше ворогів, більше кімнат та більше подій.

В майбутньому планується ввести систему предметів та продавців, які будуть впливати на характеристики героя, та надати йому можливість вдосконалитися, але ці апгрейди тимчасові, вони також пропадуть при провальному проходженні підземелля.

У розробці всього інтерфейсу користувача використовувалась платна утиліта DoozyUI з Unity Store. За її допомогою, можна розробити якісно та швидко інтерфейс з різною анімацією та звуками, що сильно зекономити час та сили у створенні стильного та зрозумілого інтерфейсу користувача.

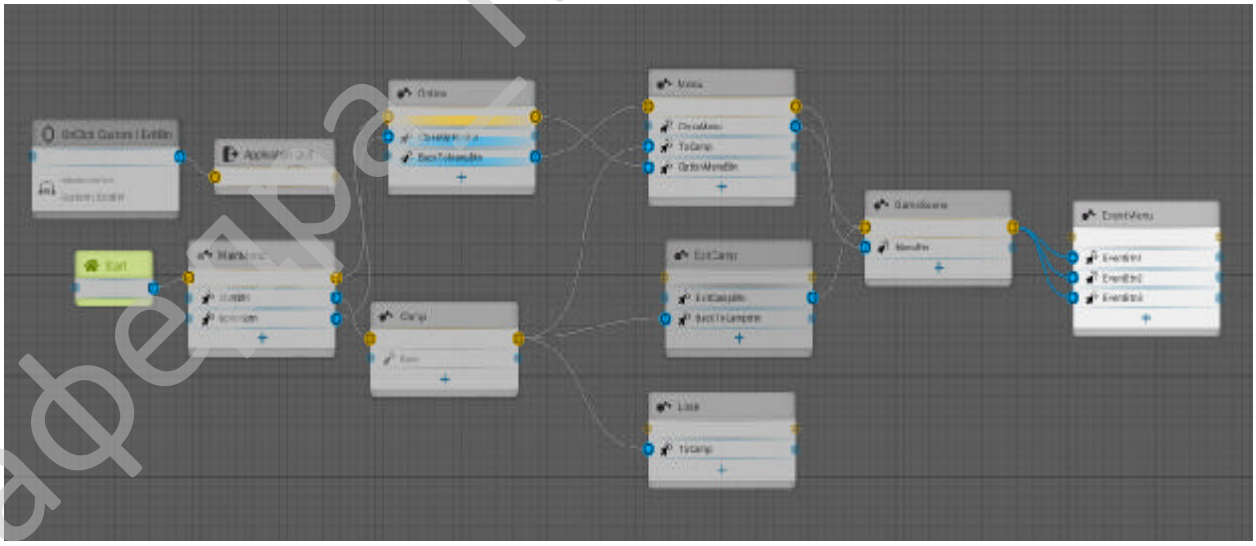

Рисунок 3.3 – DoozyUI, вікно що керує інтерфейсом користувача

3.2 Розробка структурної схеми

Розробка структурної схеми, яка відображає суть гри, ставить перед собою задачу, чітко та ясно описати структуру всіх головних процесів які проходять через весь життєвий цикл даної програми.

На рис. 3.4 зображена структурна схема розробленої системи.

Структурна система розробленого програмного забезпечення складається з таких частин:

1) Комп’ютерна гра:

– Початок ігрового процесу.

– Налаштування, передає необхідні параметри контенту у блок процедурної генерації контенту.

– Генерація контенту та взаємодія гравця з контентом.

– Проходження рівнів та нарахування балів.

– Завершення гри та виведення рейтингу.

– Оцінка складності підземелля, отримана інформація передається у блок процедурної генерації контенту у генетичний алгоритм для фітнес-функції.

2) Процедурна генерація контенту:

– Алгоритм Боуера-Ватсона, генерує граф тріангуляції Делоне для кімнат.

– Алгоритм Прима-Краскала, створює з тріангуляції мінімальне кістякове дерево.

– Генетичний алгоритм вибирає з множини згенерованих алгоритмів, такий, що відповідає за складністю потомному рівню гри.

– Створення карти підземелля.

– Збереження карти підземелля у базу даних.

Для коректної роботи програми, у першу чергу відбувається створення процедурної генерації контенту гри, випадкових подій у кімнатах та створення самого героя. На цих процесах зосереджена головна візуальна частина роботи системи.

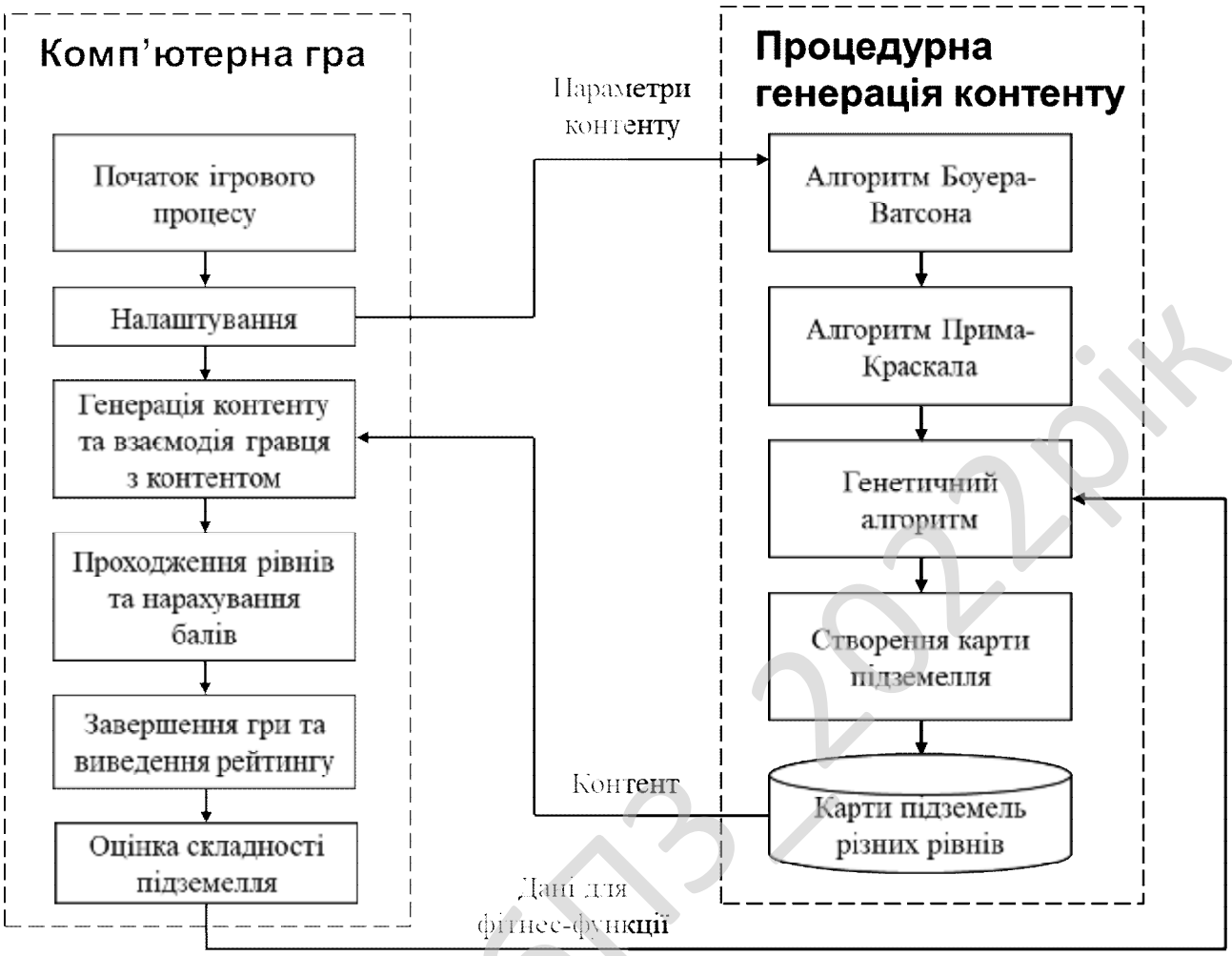

Рисунок 3.4 – Структурна схема системи

Після цих трьох кроків користувач починає своє дослідження підземелля. У кожній кімнаті може виникнути випадкова подія, на якій користувач робить свій вибір розвитку подій. Таким чином, користувач визначає наступну поведінку роботи системи.

У випадку бойової події, починається завантаження нової сцени зі створенням ворогів, яких треба знищити для подальшого його проходження.

3.3 Розробка функціональної схеми

На рис. 3.5 зображена функціональна схема розробленої системи.

Дана схема включає в себе відображення зв'язків між блоками, що

представляють собою схематичне зображення різних модулів системи.

Рисунок 3.5 – Функціональна схема системи

Був проведений аналіз всіх підсистем, що забезпечують коректну дію даного програмного забезпечення. Для початку – підсистема для взаємодії з користувачем повинна буди інтуїтивно зрозумілою та простою у використанні. Окрім даної підсистеми, існує ще декілька підсистем:

– підсистема процедурної генерації ігрового контенту.

– підсистема дій головного героя.

– підсистема бойових операцій.

Підсистема процедурної генерації ігрового контенту складається з наступних елементів:

– Підсистема налаштування випадкових дій.

– Підсистема створення кімнат та доріг між ними.

– Підсистема відбору більш складних підземель для наступних рівнів гри еволюційним алгоритмом.

Як видно на схемі представленій вище, всі існуючі підсистеми зв’язані одна з одною для коректного життя програми.

Підсистема взаємодії з користувачем пов’язана з усім інтерфейсом користувача, для подальшого розвитку програми.

Система генерації підземелля включає в себе декілька менших підсистем, які проводять всю роботу для побудови підземелля, в якому потім буде пересуватися герой, та створювати випадкові події для більшого різноманіття. Окрім створення, у цій підсистемі є функція збереження всієї інформації, яка використовується та змінюється у всіх інших підсистемах.

Система головного героя керує діями героя, що будуть виникати на протязі всієї роботи програми. Відображає стан героя, переміщення по кімнатам, бойові дії та у разі появи події в підземеллі, його вирішення.

Остання підсистема, але не по значимості – підсистема бойових операцій. Дана підсистема забезпечує появу та логіку дій противників, їх переміщення, вплив на героя та взаємозв’язок їх з іншими противниками.

3.4 Розробка діаграми процесів

На рисунку 3.6 зображена діаграма процесів, що описує роботу систему, а також є візуалізацією структурної та функціональної схем.

Дана діаграма відображає послідовність роботи основних процесів, що являються рушіями усієї моделі.

У розробленій системі є наступні процеси, взаємодія яких представлена на рисунку 3.5:

– Головне вікно програми.

– Старт гри у початковій локації.

– Налаштування.

– Процедурна генерація підземелля.
– Алгоритм Боуера-Ватсона.
– Алгоритм Прима-Краскала.
– Еволюційні алгоритми.
– Генерація випадкових подій.
– Побудова кімнат підземелля та доріг між ними.
– Переміщення героя.
– Дія героя при події.
– Переміщення до бойової сцени.
– Зачистка підземелля.
– Показ героя.

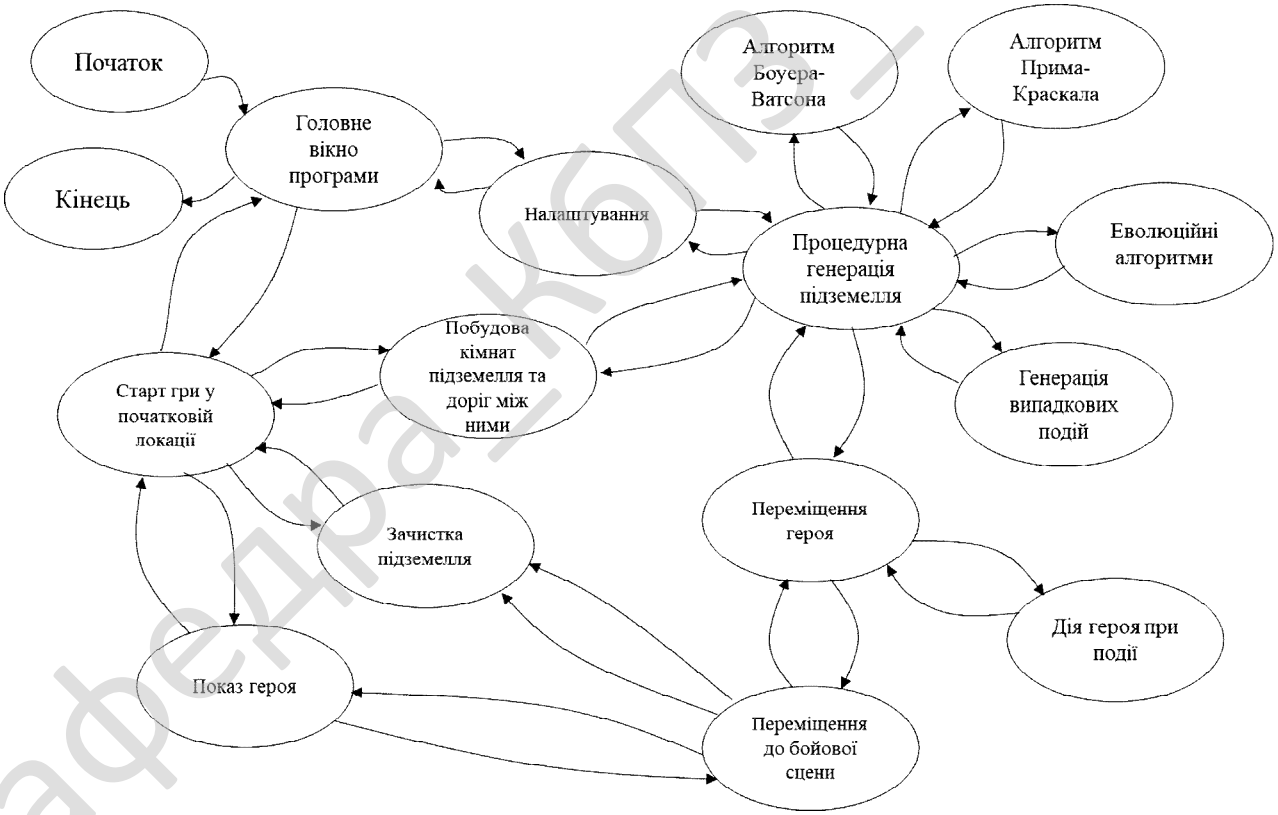

Рисунок 3.6 – Діаграма процесів системи

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Проаналізувавши вищеописані функції, слід розібрати розробку програмного забезпечення і його архітектуру. Визначити основний життєвий цикл роботи програми та її особливості, роботу підсистем, що керують різними сферами діяльності.

Робота програми починається с головного меню, в ньому є можливість перейти до налаштувань, вийти з програми чи розпочати саму гру. При натисканні на кнопку «Розпочати» відбувається завантаження сцени табору. На даний момент, це просто декоративна сцена, через яку є можливість перейти до сцени з підземеллям. Підійшовши до виходу, герой визве меню через вхід у невидимий BoxCollider, через яке можна почати загрузку сцени MainScene.

На цій сцені розпочинається зразу декілька процесів. Окрім відображення інтерфейсу користувача зі здоров’ям героя, кнопкою меню і рівнем пробудження підземелля, запускається низка процесів, які відповідають за побудову кімнат цього підземелля. Для коректної роботи цих процесів відбувається передача потрібних даних, які зберігаються у Global.cs.

За всю реалізацію побудови кімнат відповідає скрипт Generation. Для побудови кімнат, створюється матриця із об’єктів створенного класу Tile, у якому будуть зберігати координати цих об’єктів. Наступним кроком буде розміщення стартової точки для побудови першої кімнати у рандомному місці цієї матриці, у цій кімнаті створюється модель героя, прив’язується за героєм камера і відбувається побудова інших кімнат за допомогою функції CreateDungeon().

З початку роботи цієї функції йде перевірка на кількість кімнат, яка не

може перевищувати норму. Далі ми випадковим чином вибираємо напрямок ходу і перевіємо наявності Tile, якщо такий об’єкт існує, то відбувається перевірка на пустоту цього Tile. Якщо цей об'єкт пустий, то ми створюємо поточну кімнату, а попередню запам'ятовуємо і будуємо між ними шлях, по якому зможе ходити герой. Нижче представлена функція, яка відповідає за побудову деякої кімнати:

```
void CreateRoom(Tile tile)
    {
        ifStack = 0;
        tileBack = 1;
        tile.myTile = Tile.tileTypes.Room;
        int rand = Random.Range(0, roomsTypes.Count - 1);
        GameObject go = Instantiate(roomsTypes[rand], tile.transform);
        rooms.Add(go);
        go.name = (rooms.Count - 1).ToString();
        prevRoomCreation = currRoomCreation;
        currRoomCreation = go;
        SetEvent();
        roomCounter++;
    }
```

У кожної кімнати є свій скрипт, в якому розташована інформація про розташування найближчих кімнат, та функції перевірки побудови між ними шляху. У відповідності за напрямком побудови кімнати, беруться якоря, які розташовані на префабах поточної та попередньої кімнати і за допомогою короткої функції CreateWay візуально відображується через однакову дистанцію шлях між цими кімнатами.

```
 void CreateWay(GameObject start, GameObject finish, GameObject createdGameObject)
    {
        for (int i = 0; i < segmentsCreated; i++)
        {
            lerpValue += 0.2f;
             Vector3 distance = Vector3.Lerp(start.transform.position,
 finish.transform.position, lerpValue);
            GameObject go = Instantiate(createdGameObject, start.transform);
```

```
            go.transform.position = distance;
        }
    }
```

У випадку, якщо побудова кімнат зайшла в глухий кут, проходиться список у зворотному напрямку, який намагається побудувати кімнату в іншу сторону, поки не відбудеться бажана побудова.

Нижче у блок-схемі 4.1 представлена повна робота алгоритму побудови всіх кімнат цього підземелля.

Надалі необхідно описати алгоритм роботи підсистеми що керує персонажем окремо.

Герой переміщується за допомогою клавіш «WASD». Натискаючи на клавішу, відбувається перевірка поточної кімнати на наявність сусідніх кімнат у тому напрямку, куди бажає йти гравець. У випадку, коли сусідня кімната є в наявності, починає виконуватись переміщення героя, до того моменту, коли герой дійде до кімнати. При чому користувач не може переміщати героя в інші кімнати, поки він не дійде до своєї цілі і не пройде кулдаун перемішення.

```
IEnumerator WaitSeconds(int time)
    {
        if(!currentRoom.checkedRoom)
        {
            Global.instance.Danger(1);
            currentRoom.checkedRoom = true;
            if (currentRoom.isEvent)
            {
                Debug.Log("This is Event " + currentRoom.eventName);
                Global.instance.currRoom = currentRoom.GetComponent<Room>();
                EventManager.instance.SetEvent();
                if(currentRoom.eventName != "Battle")
                {
                    Global.instance.graph.SetActiveNodeByName("EventMenu");
                    Global.instance.isEventMenu = true;
                }
            }
        }
```

```
        yield return new WaitForSecondsRealtime(time);
        isMoving = false;
    }
```

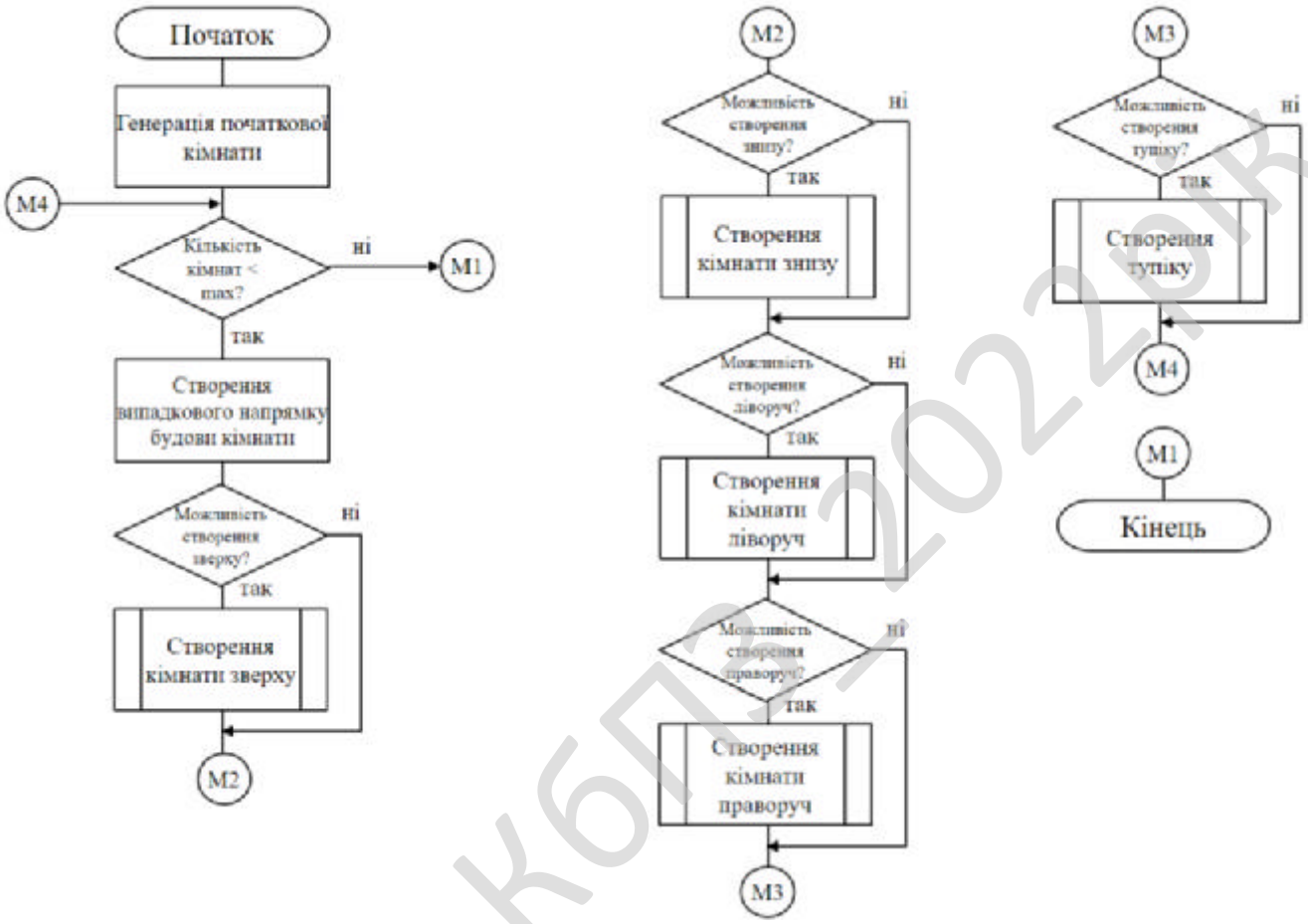

Рисунок 4.1 – Блок-схема алгоритму генерації підземелля

Нижче у блок-схемі 4.2 представлена робота переміщення головного героя у підземеллі. Обробку можливих дій та команд які відповідають за переміщення, показ можливих подій або загрузку нової сцени.

Як можна побачити, у випадку коли на поточній кімнаті є подія, після закінчення переміщення героя, йде обробка можливої події, якщо це текстова подія, через граф DoozyUI визивається нода, яка керує вікном події.

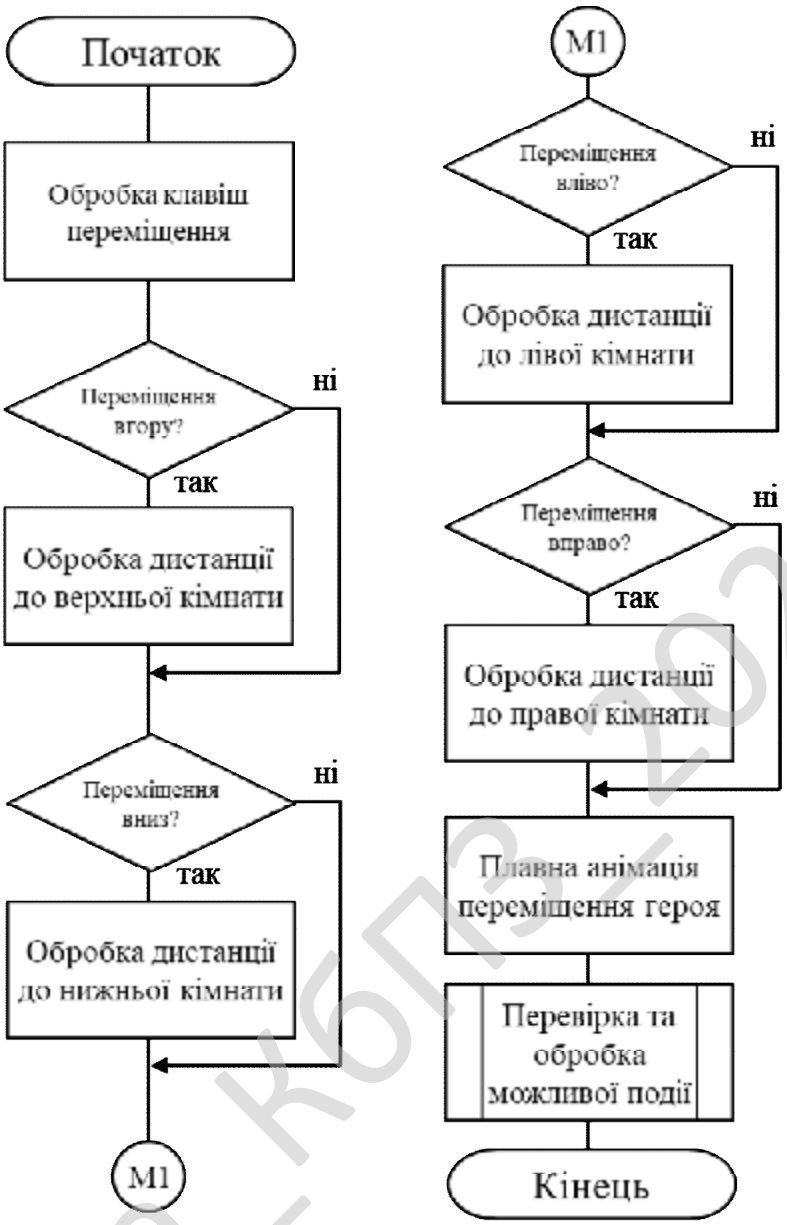

Рисунок 4.2 – Блок-схема алгоритму переміщення у підземеллі

За підсистему побудови подій відповідає EventManager. Його роль, генерувати можливі події, які будуть створюватись при побудові кімнат, контролювати встановлену кількість можливих подій, обробити та встановити відповідний текст, зображення та кнопки дій у вікні подій.

Відповідно до тексту який написаний на кнопках, відбувається встановлена дія, яка розписана у відповідному скрипту EventBtn і прив’язана до кожної з кнопок.

У випадку коли викликається подія бою відбуваються наступні дії:

– Вимикання камери на сцені з підземеллям.
– Зупинка підтримки підсистеми переміщення головного героя.
– Викликається корутина загрузки додаткової сцени бою.

```
public void Battle()
    {
        ExitCamp = false;
        StartCoroutine(LoadScene_Courutine(2, true));
        GameObject player = Instantiate(battlePlayer);
        player.transform.position = new Vector3(0, 1002, 0);
        MapCamera.SetActive(false);
        isBattle = true;
            }
IEnumerator LoadScene_Courutine(int scene, bool additiveMode)
    {
        if(additiveMode)
        {
            AsyncOperation async = SceneManager.LoadSceneAsync(scene,
LoadSceneMode.Additive);
            while (!async.isDone)
            {
                yield return null;
            }
        }
        else
        {
            AsyncOperation async = SceneManager.LoadSceneAsync(scene,
LoadSceneMode.Single);
            while (!async.isDone)
            {
                yield return null;
            }
        }
            }
```

При будь-якому переміщенню в кімнату, в яку герой не входив, підвищується рівень пробудженості підземелля, від якого залежить складність проходження підземелля. Чим вище рівень, тим складніші буть створюватись

вороги на бойовій сцені.

```
public void Danger(int step)//refact
    {
        int maxlvl = 5;
        if (step > 0)
        {
            if (dangerLvl < 5)
            {
                currLvl += 1;
                if (currLvl >= maxlvl)
                {
                    dangerLvl++;
                    dangerText.text = dangerLvl.ToString();
                    currLvl -= 5;
                    for (int i = 0; i < dangerIcons.Count; i++)
                    {
                        dangerIcons[i].SetActive(false);
                    }
                }
                DangerImage(dangerIcons[currLvl]);
            }
        }
    }
```

На камері підв’язаний скрипт за можливість переміщення та обертанні камери у певній визначеній області. На ліву кнопку миші зав’язане саме переміщення камери по всьому підземеллю. Обертання камери відбувається при нажаті на праву кнопку миші, яке дозволяє оглядіти кімнати на всі 360 градусів. Скролл колеса миші відповідає за зум у обмеженій області. Також була реалізована функція зміни кута нахилу, за цю функцію відповідає клік на колесо миші.

```
        Vector3 pos = transform.position;
        if (Input.GetMouseButton(0))
        {
            pos.x += (Input.GetAxis("Mouse X") * -1) * mouseSensivity;
            pos.z += (Input.GetAxis("Mouse Y") * -1) * mouseSensivity;
        }
        if (Input.GetMouseButtonDown(1))
        {
            mouseClickRotate = true;
        }
        if (Input.GetMouseButtonUp(1))
        {
            mouseClickRotate = false;
        }
        if (Input.GetMouseButtonDown(2))
        {
            mouseClickjudge = true;
        }
        if (Input.GetMouseButtonUp(2))
        {
```

```
            mouseClickjudge = false;
        }
        if (mouseClickjudge)
        {
            yAxis += Input.GetAxis("Mouse Y") * rotateSpeed;
            yAxis = Mathf.Clamp(yAxis, minXRotate, maxXRotate);
            transform.rotation = Quaternion.Euler(yAxis,
transform.rotation.eulerAngles.y, transform.rotation.eulerAngles.z);
        }
        if (mouseClickRotate)
        {
            xAxis += Input.GetAxis("Mouse X") * rotateSpeed;
            transform.rotation = Quaternion.Euler(transform.rotation.eulerAngles.x,
xAxis, transform.rotation.eulerAngles.z);
        }
        float scroll = Input.GetAxis("Mouse ScrollWheel");
        pos.y -= scroll * scrollSpeed * 100f * Time.deltaTime;

        pos.x = Mathf.Clamp(pos.x, panLimitStart.x, panLimitEnd.x);
        pos.y = Mathf.Clamp(pos.y, minY, maxY);
        pos.z = Mathf.Clamp(pos.z, panLimitStart.y, panLimitEnd.y);

                transform.position = pos;
```

Після загрузки сцени бою відбувається створення різних типів ворогів. У відповідності типу ворогів, складності рівня і рівня пробудженості підземелля створюється різна кількість ворогів у випадковому місці у кімнаті бою. Головна задача героя пережити бій та знищити всіх ворогів.

Після ліквідування всіх ворогів відбувається вигрузка сцени бою та перехід до головної сцени. Паралельно відбувається включення камери на сцені з кімнатами підзмелля, на тій кімнаті де зупинився герой та включається можливість подальшого переходу до інших кімнат.

```
void CreateEnemy(List<GameObject> enemyList, int counter)
    {
        for(int i = 0; i < counter; i++)
        {
            int rand = Random.Range(0, enemyList.Count - 1);
            GameObject enemy = Instantiate(enemyList[rand], room);
            enemy.transform.position = GetRandomPlace(plane);
            Global.instance.totalEnemy++;
        }
    }

    Vector3 GetRandomPlace(BoxCollider mesh)
    {
        float screenX = Random.Range(mesh.bounds.min.x + 5f,
mesh.bounds.max.x - 5f);
        float screenY = Random.Range(mesh.bounds.min.z + 5f,
mesh.bounds.max.z - 5f);
        return new Vector3(screenX, 1002, screenY);
            }
```

```
private void Update()
    {
        if(Global.instance.totalEnemy == 0)
        {
            Global.instance.UnloadBattleScene();
        }
             }
```

На кожному противнику є EnemyController, який відповідає за налаштування і можливу поведінку дій, як основні налаштування, за показники здоров'я, швидкість та сили атаки, швидкість переміщення, так і логіку дій кожного типа окремо.

Цей контроллер відстежує найближчого героя який війде до зони зору, після чого ворог почне свій рух у відповідності їх налаштування та дистації до героя і стін.

```
nearPlayers = nearPlayers.Where(item => item != null).ToList();
        if (nearPlayers.Count != 0)
        {
            nearestPlayer = nearPlayers.OrderBy(t =>
Vector2.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
        }
        else nearestPlayer = null;
        if (enemyType == "Range")
        {
            if (isCanShooting == true)
                StartCoroutine(Shoot(5f));
            isCanShooting = false;
        }
        if (enemyType == "Summoner")
        {
            if (isCanSummoning == true)
                StartCoroutine(Summoning(5f));
            isCanSummoning = false;
        }
        if (enemyType == "Debafer")
        {
            if (isCanDebaffing == true)
                StartCoroutine(Debaff(5f));
            isCanDebaffing = false;
        }
        if (nearestPlayer != null)
        {
            Movement(enemyType);
                }
IEnumerator Shoot(float time)
    {
        yield return new WaitForSeconds(time);
        if (this.enemyType == "Range")
        {
```

```
            if (nearestPlayer != null)
            {
                Vector3 pos = gameObject.transform.position;
                Quaternion rotation = gameObject.transform.rotation;
                GameObject bulletObj = Instantiate(bullet, pos, rotation);
                bulletObj.GetComponent<BulletController>().shooter =
gameObject.tag;
                bulletObj.GetComponent<BulletController>().direction = new
Vector3((nearestPlayer.position - pos).normalized.x, 0, (nearestPlayer.position -
pos).normalized.z);
            }
            isCanShooting = true;
        }
    }

    IEnumerator Summoning(float time)
    {
        yield return new WaitForSeconds(time);
        if (this.enemyType == "Summoner")
        {
            summonedEnemies = summonedEnemies.Where(item => item != null).ToList();
            if ((nearestPlayer != null) && (summonedEnemies.Count < 2))
            {
                System.Random rand = new System.Random();
                int i = rand.Next(0, enemiesToSpawn.Count);
                GameObject summonedEnemy =
Instantiate<GameObject>(enemiesToSpawn[i]);
                summonedEnemy.transform.position = this.transform.position + new
Vector3(2, 0, 2);
                summonedEnemies.Add(summonedEnemy);

            }
            isCanSummoning = true;
        }
    }

    IEnumerator Debaff(float time)
    {
        if (this.enemyType == "Debafer")
        {
            if (nearestPlayer != null)
            {
                nearestPlayer.GetComponent<PlayerController>().debaffed = true;
            }
            yield return new WaitForSeconds(time);
            nearestPlayer.GetComponent<PlayerController>().debaffed = false;
            isCanDebaffing = true;
        }
    }
```

Нижче у блок-схемі 4.3 представлена робота контроллеру ворога.

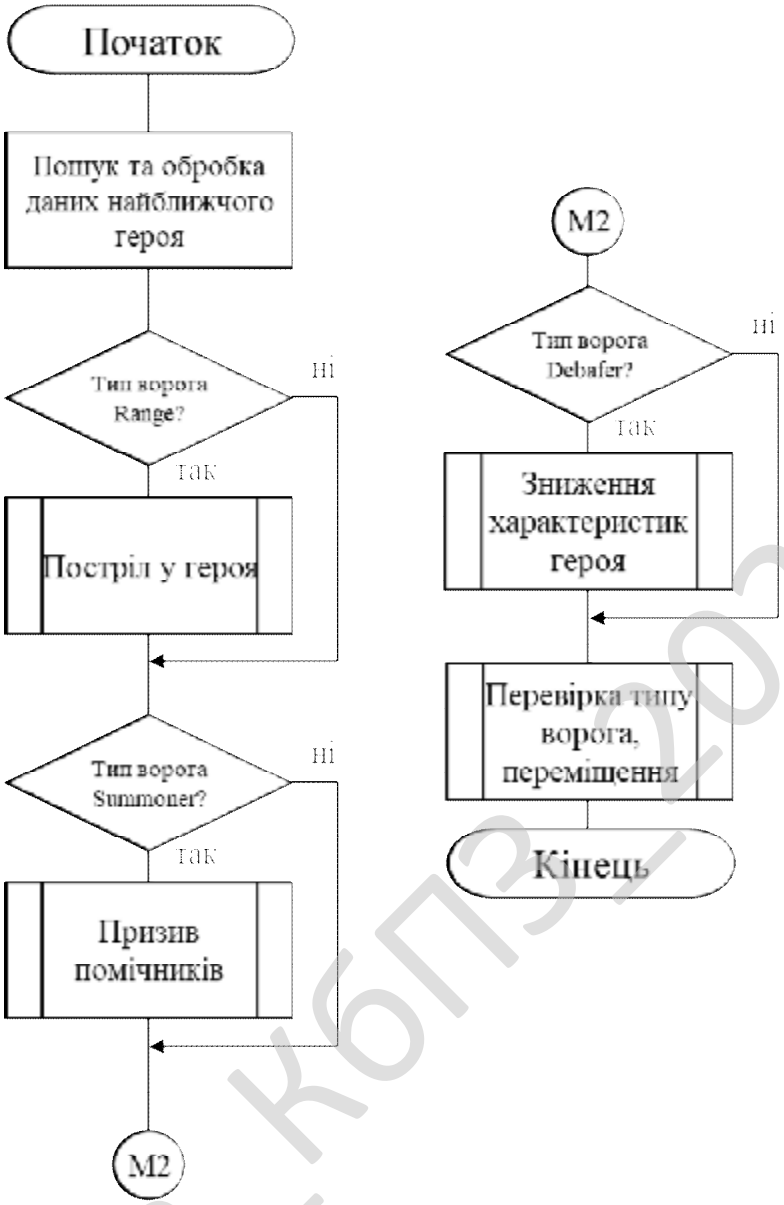

Рисунок 4.3 – Блок-схема алгоритму дії контролера супротивника

4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення пропонується захищати від несанкціонованого копіювання та поширення за допомогою прив’язки до даних про комп’ютер користувача. Це можна зробити шляхом створення прив’язки до серійного номеру компонентів комп’ютера користувача і подальшої активації програмного забезпечення.

Під час установки програмне забезпечення обчислює код активації – контрольне значення, що однозначно відповідає параметрам встановленої ОС та встановленим комплектуючим комп'ютера. Це значення передається розробнику програмного забезпечення. На його основі розробник генерує ключ активації, що підходить для активації програми лише на зазначеному комп'ютері, копіювання встановлених виконуваних файлів на інший комп'ютер призведе до непрацездатності програмного забезпечення.

Як прив'язку до комп'ютера користувача пропонується використовувати серійний номер BIOS материнської плати та серійний номер вінчестера. Для приховування від користувача даних про захист, пропонується розташовувати їх в нерозмічену область жорсткого диску та шифрувати алгоритмом RSA.

Алгоритм RSA є асиметричним алгоритмом шифрування. Асиметричний насправді означає, що він працює з двома різними ключами, тобто **відкритим ключем** і **закритим ключем.** Як видно з назви, відкритий ключ надається кожному, а закритий ключ залишається закритим.

Ідея RSA базується на тому, що велике ціле число важко розкласти на множники. Відкритий ключ складається з двох чисел, де одне число є множенням двох великих простих чисел. І закритий ключ також походить від тих самих двох простих чисел. Отже, якщо хтось може розкласти велике число на множники, приватний ключ буде зламано. Тому міцність шифрування повністю залежить від розміру ключа, і якщо ми подвоюємо або потроїмо розмір ключа, міцність шифрування зростає експоненційно. Ключі RSA зазвичай на сьогоднішній день можуть мати довжину 1024 або 2048 біт, але експерти вважають, що 1024-бітні ключі можуть бути зламані найближчим часом. Але поки це здається нездійсненним завданням.

Для прискореної реалізації алгоритму RSA в C# можна використати бібліотеку System.Security.Cryptography. Наведено приклади функцій шифрування та дешифрування даних алгоритмом RSA з використанням вищезазначеної бібліотеки.

Приклад функції шифрування даних алгоритмом RSA на мові C#:

```
        static public byte[] Encryption(byte[] Data, Parameters Key, bool
DoOAEPPadding)
        {
         try
         {
         byte[] encrData;
         using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())
           {
            RSA.ImportParameters(Key);
                    encrData = RSA.Encrypt(Data, DoOAEPPadding);
           }   return encrData;
         }
         catch (CryptographicException e)
         {
         Console.WriteLine(e.Message);
         return null;
         }
        }
```

Приклад функції дешифрування даних алгоритмом RSA на мові C#:

```
        static public byte[] decryption(byte[] Data, Parameters Key, bool
DoOAEPPadding)
        {
         try
         {
         byte[] decrData;
         using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())
            {
             RSA.ImportParameters(Key);
             decrData = RSA.Decrypt(Data, DoOAEPPadding);
            }
         return decrData;
         }
         catch (CryptographicException e)
         {
         console.writeline(e.tostring());
         return null;
         }
        }
```

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Головне завдання фінальної частини розробки програмного забезпечення даного проекту – визначити можливості його інтеграції у робочий процес.

Перший крок впровадження – визначити спосіб розповсюдження програмного забезпечення та можливостей його подальшого використання. Окремі частини даного ігрового проекту можна подати, як набір інструментів, які в подальшому можна буде використовувати як конструкт в інших проектах схожого контенту та опублікувати їх в Unity Asset Store.

У цьому магазині кожен розробник, що потребує якийсь додатковий функціонал до свого розроблюваного програмного забезпечення, може з легкістю знайти його та використати його в своїх цілях.

Інший спосіб розповсюдження – використання репозиторіїв і викладення системи у вільний доступ, як Git Hub проект. Це дозволяє розробникам самим приймати участь у оновленні та покращені розробленої моделі. Це досить функціонально, проте існує велика загроза безпеці системи, а також вірогідність копіювання викладеної системи.

Важливий етап – оформлення детальної супровідної документації для використання запропонованого функціоналу системи:

– Опис основних цілей програмного забезпечення, його призначення та можливостей.

– Детальний опис базових скриптів роботи системи.

– Опис характеристик налаштування системи: об’єктів та середовища.

Не менш важливим кроком у інтеграцію системи – створення тестової сцени, що вже є налаштованою і служить для демонстрації того, що функціонал моделі у нормі і може виконувати свої задачі. Крім цього демонстрація цієї сцени, допоможе провести рекламну промоакцію даного програмного

забезпечення, що є важливим пунктом у її просуненні та інтеграції у створення ігрових систем інших розробників.

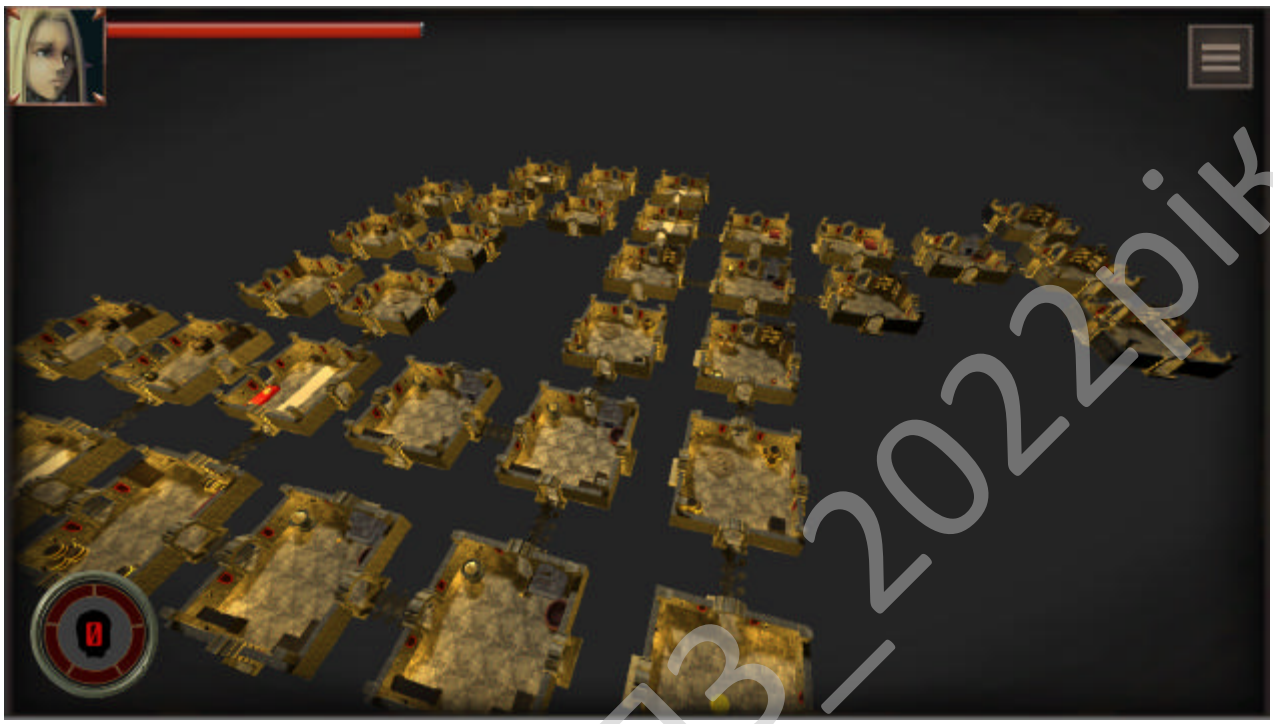

Рисунок 5.1 – Демонстраційна сцена процедурної генерації підземелля для комп’ютерної гри у розробленому програмному забезпеченні

Таким чином можна провести інтеграцію розробленої системи у робочій процес досить швидко та ефективно, не витрачаючи занадто великої кількості ресурсів та сил.

Отже, після проведення аналізу розроблюваного програмного забезпечення, розробки алгоритмів та архітектури, проведення роботи над можливою інтеграцією у виробництво, необхідно зробити висновки по роботі.

6 НАУКОВА НОВИЗНА

У магістерській роботі розроблено програмне забезпечення, яке призначено для системи процедурної генерації контенту комп'ютерної гри.

Метою роботи є дослідження та програмна реалізація системи процедурної генерації контенту комп'ютерної гри.

Об'єктом дослідження є процес процедурної генерації контенту комп'ютерної гри.

Предметом дослідження є методи та алгоритми процедурної генерації та комп'ютерної графіки для комп'ютерних ігор.

Методи дослідження базуються на методах штучного інтелекту, методах комп'ютерної графіки, методах розробки програмного забезпечення, методах розробки комп'ютерних ігор.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Удосконалено метод процедурної генерації контенту комп'ютерної гри, що відрізняється від існуючих застосуванням еволюційних методів машинного навчання для ускладнення ігрових рівнів для гравця з часом.

2. Розроблено вітчизняний продукт процедурної генерації контенту комп'ютерної гри, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний для прискореної розробки комп'ютерних ігор.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі процедурної генерації контенту комп'ютерної гри.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми магістерської роботи

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 48 днів (два місяці).

В магістерській роботі проведене дослідження та виконана програмна реалізація системи процедурної генерації контенту комп'ютерної гри.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір та системні потреби;

б) незалежність від встановлених на комп'ютері баз даних;

в) зручність у користуванні та надійність

Таблиця 7.1 - Початкові данні

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	60
3. Запланований термін розробки, днів	Fpq	48 (2 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	3

Продовження таблиці 7.1

1	2	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	1
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	_	2
13. Рівень спрацьованості колективу (1-6)	_	2
14. Ступінь вимірності процесів (1-6)	_	3
15. Необхідна надійність програмного забезпечення (1-6)	_	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	_	2
17. Складність кінцевого програмного продукту (1-6)	_	2
18. Необхідний рівень забезпечення повторного використання (1-6)	_	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	_	2
20. Вимоги до швидкодії ПП (1-6)	_	2
21. Обмеження на розміри основного сховища даних (1-6)	_	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	_	2
23. Професійний рівень аналітиків (1-6)	_	2
24. Професійний рівень програмістів (1-6)	_	2
25. Постійність складу команди розробників (1-6)	_	2
26. Досвід розробки додатків (1-6)	_	2
27. Досвід роботи з обчислювальною платформою (1-6)	_	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПО для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн	–	60000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A\ \text{Size}^B \qquad (7.1)$$

де А - коефіцієнт Боема, А=2,45;

Size - загальний об'єм відлагодженого програмного коду, тис. рядків;

B - показник ступеня, що визначається співвідношенням

$$B = 1{,}01 + 0{,}001\sum W_i \tag{7.2}$$

де Wi - сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B=1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}026$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн}=T_{ном}\, ПВ_j, \tag{7.3}$$

де $ПВ$j - добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн}= 6{,}78\cdot(0{,}88\cdot0{,}93\cdot0{,}88\cdot0{,}91\cdot0{,}95\cdot1\cdot1\cdot0{,}87\cdot1{,}22\cdot1{,}16\cdot1{,}1\cdot1{,}1\cdot$$
$$\cdot1{,}12\cdot1{,}1\cdot1\cdot1{,}1\cdot1{,}1)=9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3CT_{уточн}^{0{,}33+0{,}2(B-1{,}01)}S, \tag{7.4}$$

де C - визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S - коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3\cdot3{,}23\cdot9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot120=245 \text{ люд/день}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	245	Ф 7.1-7.4
Впровадження	13	Д13
Всього	286	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \tag{7.5}$$

де $F_{рq}$ - плановий фонд робочого часу одного спеціаліста, днів,

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні,

$$Ч = \frac{347 \cdot 1}{48\text{-}5} = 8 \text{ ставки}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо

до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	12	1080	18
Монітор	60	14	840	14
Клавіатура	30	12	360	6
Маніпулятор «мишка»	30	12	360	6
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор–маршрутизатор	30	2	60	1
Кабельні господарства ЛВС на 1 м. п.	2,5	350	875	14,58
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	65,24

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2} \quad (7.6)$$

$$Ф_{др}^{с} = \frac{65,24 \cdot 2}{1,2} = 109 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}} \quad (7.7)$$

$Ч_{ел}$=109/(48·8)=0,28 ставки

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків. Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 - Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2016, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	0,8	0,2
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,2	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,2	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	0,4	
Всього		1,6	

Продовження таблиці 7.4

Посада	Вид роботи	Час	Кількість штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	2	0,5
	Підтримка постійних клієнтів	1	
	Оформлення договорів, ведення тендерів	0,5	
	Контроль взаєморозрахунків з постачальниками	0,5	
Всього		4	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	0,5	0,2
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,3	
	Розміщення графіки і контенту на Інтернет сторінках	0,3	
Всього		1,6	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,2
	Верстка друкованих видань	0,2	
	Додрукова підготовка макетів	0,2	
	Розміщення графіки і контенту на Інтернет сторінках	0,2	
Всього		1,6	

Складемо штатний розклад виконавців у таблицю 7.5.

Таблиця 7.5 - Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (IT-менеджер)	1	18330	36660
Продакт-менеджер	0,5	14000	14000
Інженер-програміст	8	16000	256000
Інженер-електронщик	0,28	14000	7840
Інженер-системотехнік	0,2	14000	5600
Адміністратор мережі	0,2	14000	5600
Системний програміст	0,2	14000	5600
Дизайнер WEB	0,2	15000	6000
Інженер-верстальник	0,2	14000	5600
Бухгалтер-економіст	0,2	15000	6000
Всього за період розробки	$R_{сп}$=10,98	-	$Ф_{роб}$=348900

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{348900}{10,98 \cdot 48} = 662 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць.

$S_у$ – питома площа на одне робоче місце, $м^2$,

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно $8 м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн на одне робоче місце. Тобто

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де Цм – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу Інтернет магазину Компбест за 17.11.22 – джерело https://compbest.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		11771
Системний блок		7771
Процесор	Intel Core i7-2700K (S1155/4x3.5GHz /5GT/s/8MB/95 Вт)	2500
Системна плата	Huanan B75 (s1155, Intel B75, PCI-Ex16) DVI/VGA/HDMI	1100
Жорсткий диск	HDD 500 Gb SAMSUNG Barracuda HD502HJ (3.5", 500ГБ, 16МБ, SATA II-300)	1290
Оперативна пам’ять	DIMM 4096Mb DDR3 PC3-12800 Kingston, 1600MHz, 512M x 64, CL9-9-9-27, 1.65V, w/heatsink, HyperX	834
DVD-привод	DVD±RW ASUS DRW-24B5ST Black Bulk	416
Корпус	Logicpower 8702 - 550w 12cm	1411
Кардрідер внутрішній	Transcend TS-RDF8K USB 3.0	220
інше	Клавіатура, мишка	Подару-нок
Монітор	Монітор BenQ GL2450HM Black	2600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37	2800
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійного живлення	Powercom BNT-600AP USB	1400

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 - Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	8	11771	9416,8	103584,8
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2800	280	3080
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	125216,3

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400

Продовження таблиці 7.8

1	2	3	4
Група 4			
3. Обчислювальна техніка	125216	-	-
Всього по групі	125216	50	62608
Група 5,6			
4. Вимірювальні пристрої	5190	-	-
5. Транспортні засоби	143000	-	
6. Господарський інвентар	28000	-	-
Всього по групі	176190	20	35238
7. Нематеріальні активи	60000	10	6000
Разом	$К_р$= 1769406		$А_р$= 174246

Примітка: вартість автомобіля Sens (Standard+) взята по даним з автосалону «Кіровоград-Авто», джерело http://kirovograd-avto.ukravto.ua /catalog/tm-9/model-80/description, складає 143000 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де $N_е$ – Кількість екземплярів програм, шт.

$$З_о = 662 \cdot 286/60 = 3158 \text{ грн}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де $Н_q$ – норматив додаткової зарплати, %

$$З_д = 3158 \cdot 10 \cdot 0{,}01 = 316 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=22% від суми основної та додаткової зарплати

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де $Н_с$ – відрахування на соціальні потреби, %

$$С_{оц} = 0{,}01 \cdot 22(3158+316) = 764 \text{ грн}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Нг$=15% від основної зарплати

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 3158 \cdot 15 \cdot 0{,}01 = 474 \text{ грн}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де $З_{М1}$ – вартість паперу, грн., $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн., $З_{М3}$ – вартість фарби, картріджей, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 0,4 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=200 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N. \quad (7.16)$$

$$З_{М1} = 200 \cdot 0{,}4 = 80 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 10):

$$З_{М2} = \sum Ц_д., \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 33 грн./шт., DVD-R box – 49 грн./шт.

$$З_{М2} = 49 \cdot 10 = 490 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{МЗ} = \sum Ц_{з}., \tag{7.18}$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{МЗ} = 574+558+570=1702 \text{ грн.}$$

$$З_{М} = (80 + 490 + 1702)/60=38 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \tag{7.19}$$

де $Н_п$ - норматив витрат на освоєння нових мов програмування, %

$$О_п = 3158 \cdot 15 \cdot 0{,}01=474 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 60 прим.)

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \tag{7.20}$$

де $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 174246 \cdot 2/(60 \cdot 12)=484 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \tag{7.21}$$

$$С_п = 3158+316+764+474+38+474+484=5708 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності (Рп) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \tag{7.22}$$

де P_c – рівень рентабельності, %

$$П_р = 0{,}01 \cdot 50 \cdot 5708 = 2854 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	3158
2. Додаткова зарплата виконавців	$З_д$	316
3. Відрахування на соціальні потреби	$С_{оц}$	764
4. Загальногосподарські витрати	$Г_{осп}$	474
5. Витрати на матеріали	$З_М$	38
6. Освоєння нових операційних систем, мов програмування	$О_п$	474
7. Амортизація основних фондів	$А_м$	484
8. Повна собівартість програмного забезпечення	$С_п$	5708
9. Плановий прибуток	$П_р$	2854
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	8562
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	1712,4
12.Відпускна ціна програмної продукції Ц = Цп + ПДВ	$Ц$	10274,4

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 - Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн	
	Базовий	Новий
Вартість програмної продукції	–	10274
Всього капітальних витрат	–	10274

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 - Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позна-чення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на оплату праці	$З_р$	181170	31537
2. Витрати на електроенергію	$З_{ел}$	621	108
Витрати на амортизацію	$З_{ам}$	0	5137
Всього витрат за рік	$І$	181791	36782

Витрати на оплату праці:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \quad (7.23)$$

де $Т_р$ – кількість годин роботи з системою за рік, *год.*,

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення час на виконання завдання зменшилася з 1350 годин на рік до 235 годин на рік, тому витрати зменшилися з:

$$З_{р\ баз} = 1350 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 181170\ грн.$$

до

$$З_{р\ нов} = 235 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 31537\ грн.$$

Витрати на електроенергію визначаються з урахуванням спожитої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$).

$$З_{ел}=П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

Зел баз=0,2·1350·2,3=621 грн

Зел нов=0,2·235·2,3=108 грн

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 - Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	50	–	10274	–	5137
Всього відрахувань	-	–	10274	–	5137

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (8562 - 5708) \cdot 60 - (0,05 \cdot 1408000 + 0,5 \cdot 125216 +$$
$$+0,2 \cdot 176190 + 0,1 \cdot 60000) \cdot 2/12 = 142199 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{361406}{(8562 - 5708) \cdot 60 \cdot 12 / 2} = 0,35 \text{ років}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (181791\text{-}36782)\text{-}0,5 \cdot 10274 = 139872 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н} \quad (7.28)$$

$$T_{сп} = \frac{10274}{181791 - 36782} = 0{,}1 \ року$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 - Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	60
2. Повна собівартість розробленої програми	Грн.	5708
3. Ціна розробленої програми	Грн.	8562
4. Плановий прибуток від реалізації розробленої програми	Грн.	2854
5. Рентабельність програмної продукції	%	50
6. Об’єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1769406
7. Загальний прибуток від реалізації програмної продукції	Грн.	171240
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	142199
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,35
10.Об’єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	10274
11.Величина економічного ефекту у користувача програмної продукції	Грн.	139872
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,1

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Охорона праці − це: система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини в процесі трудової діяльності;

Охорона праці є складовою частиною безпеки життєдіяльності [53,54].

Законом України “Про охорону праці” [53] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [55], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби пальців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [55], та «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

- параметри повітряного середовища в приміщенні;
- вентиляція приміщення;
- освітлення приміщення;
- параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення негативного впливу комп'ютера на організм людини визначемо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

Загальний нагляд за додержанням норм охорони праці покладено на прокуратуру, спеціальний покладено на професійні спілки. За безпекою контроль праці здійснюють державні й відомчі спеціалізовані інспекції.

У Законодавстві про працю міститься вимоги і норми з виробничої санітарії, техніки безпеки та норми, що регулюють робочий час, час відпочинку, звільнення та переведення на іншу роботу, а також норми праці щодо жінок, молоді, гігієнічні норми і правила, тощо.

8.2 **Пожежна безпека**

Пожежі в приміщеннях з оргтехнікою становлять особливу небезпеку, бо поєднані з великими матеріальними збитками. Пожежа може виникнути при взаємодії горючих речовин і джерел запалювання. Горючими речовинами є будівельні та опоряджувальні матеріали, пластмасові корпуси техніки, шнури тощо. Джерелами запалювання можуть бути електронні схеми комп'ютерів, принтерів, пристроїв електроживлення, де внаслідок різних порушень виникає

перегрівання елементів, утворюються електричні іскри та дуги, здатні спричинити займання горючих матеріалів.

З метою виявлення початкової стадії займання необхідно використовувати пристрої систем автоматичного пожежогасіння там, де цього вимагають правила пожежної безпеки.

При обслуговуванні, ремонтних та профілактичних роботах використовуються різні легкозаймисті рідини, прокладаються тимчасові електропровідники, здійснюється паяння. Виникає додаткова пожежна небезпека, яка потребує відповідних заходів пожежного захисту. До засобів гасіння пожежі, призначених для локалізації невеликих займань, належать вогнегасники, сухий пісок, азбестові ковдри. Приміщення, в який встановлено комп'ютери і де немає необхідності влаштування систем автоматичного пожежогасіння, необхідно оснащувати переносними вуглекислотними з розрахунку 2 шт. на кожні 20 $м^2$ в приміщеннях. Звуковбирне облицювання стін, стель приміщень треба виконувати з негорючих та важко горючих матеріалів.

Електроустановки (можливість їх застосування, монтаж, накладка експлуатація) повинні відповідати вимогам чинних правил улаштування електроустановок, правил технічної експлуатації, електроустановок та інших нормативних документів.

Ймовірність виникнення пожежі від електротехнічного та іншого одиничного виробу не повинна перевищувати 10^{-6} на рік. При короткому замиканні в місцях з’єднання проводів опір практично дорівнює нулю, звідси величина струму досягає дуже великих значень.

Персональні комп’ютери після закінчення роботи повинні відключатися від мережі не рідше 1 разу на квартал, необхідно очищати від пилу агрегати та вузли, кабельні канали та простір між підлогами. Не дозволяється розміщувати комп’ютерні зали ЕОМ у підвалах; проводити ремонт вузлів (блоків) ЕОМ безпосередньо у залах, де знаходяться ПК (персональні комп’ютери), залишати без нагляду ввімкнену в мережу електронну апаратуру, яка використовується для

контролю ЕОМ.

Електричний струм силою 0,1 А є небезпечним для людини. Для попередження травм усе електричне обладнання повинне бути заземлене. Приступаючи до роботи необхідно перевірити справність обладнання, ізоляцію проводів і надійність заземлення. Доторкання до оголених струмоведучих і незахищених частин в електроустаткуванні забороняється. В разі виявлення порушень ізоляції електропроводів, відкритих струмоведучих частин електроустаткування або порушення заземлення треба негайно повідомити про це свого начальника для вжиття заходів щодо усунення несправності. Проводити самому ремонт електроустаткування забороняється.

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 - Розміри приміщення

Найменування	**Значення, м**
Ширина	2,6
Довжина	2,6
Висота	3

Таблиця 8.2 - Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	**Одиниця виміру**	**Нормативне значення***	**Фактичне значення**
Площа, S	$м^2$	не менше 6.0	6,76
Обсяг, V	$м^3$	не менше 20.0	20,3

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних

машин).

У зазначеному приміщенні працює 1 особа. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста відповідають нормативним вимогам (Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»).

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів

мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

Таблиця 8.3 - Оптимальні і фактичні значення параметрів мікроклімату

Пора року	**Оптимальні для Іа**			**Фактичні**		
	Температура, °С	**Вологість,%**	**Швидкість повітря, м/с**	**Температура,°С**	**Вологість%**	**Швидкість повітря, м/с**
Холодна	22-24	40-60	0,1	23-24	45-55	0,11
Тепла	23-25	50-70	0,1	23-24	55-65	0,1

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер *Xerox Phaser 3020BI*, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

Працю працівника, який постійно працює за комп’ютером, згідно ДБН В.2.5 – 28 – 2006 р можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для

такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 лк. Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

- розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;
- мікроклімат відповідає нормативному значенню;
- акустичні умови роботи не перевищують нормативних значеннь;

Таким чином можна припустити, що основною причиною можнивого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: наочне знайомство персоналу з шляхами для евакуації людей із приміщення відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов’язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання.

8.5 Розрахункова частина

Проводемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 2,6 м, довжина – 2,6 м, висота – 3 м.

У зазначеному приміщенні працює 1 особа.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F = E \cdot S \cdot K \cdot Z / n,$$

де: F - світловий потік, що розраховується, Лм;

E - нормована мінімальна освітленість, Лк; E = 300 Лк;

S - площа освітлюваного приміщення (у нашому випадку S=2,6 х 2,6 = 6,76 м^2);

K - коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку K = 1,5);

Z - відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку Z = 1,1);

n - коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

$$i = S / (h \cdot (A + B)),$$

де: S - площа приміщення, S = 6,76 м^2;

h - розрахункова висота підвісу, *h* = 3 м (співпадає з висотою стелі, т.я. лампи освітлення закріплюються на стелі);

A - ширина приміщення, *A* = 2,6 м;

B - довжина приміщення, *B* = 2,6 м.

Підставимо всі значення у формулу та визначемо індекса приміщення: i=0,43.

Знаючи індекс приміщення, за знаходимо *n* = 0,23 (з табличних даних коефіцієнтів використання світлового потоку (*n*) світильників з відповідним типом лампам). Підставимо всі значення у формулу, визначемо світловий потік: *F=14548 Лм.*

Для розрахунку дудемо використовувати світлодіодні панелі *LED панель 42Вт 6000К SUNLED 000000127*, світловий потік яких $F_л$ *= 3990 Лм.*

Число ламп визначається по формулі:

$$N = F / F_л$$

де: *F* - світловий потік,

$F_л$ - світловий потік однієї лампи.

Підставимо всі значення у формулу та визначемо індекса приміщення: *N= 14548/ 3990=3,6 шт.*

Приймаємо необхідну кількість ламп 4 шт.

8.6 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для системи процедурної генерації контенту комп'ютерної гри.

У магістерській роботі наведені теоретичне узагальнення й рішення наукового завдання дослідження методів процедурної генерації контенту комп'ютерної гри.

Рішення даного завдання полягало у вирішенні наступних задач:

– Було проведене дослідження існуючих системи процедурної генерації контенту комп'ютерної гри.

– На основі проведеного дослідження розроблено методи та алгоритми для системи процедурної генерації контенту комп'ютерної гри.

– Використовуючи розроблені методи та алгоритми, створена програмна реалізація системи процедурної генерації контенту комп'ютерної гри.

Розроблені під час виконання магістерської роботи алгоритми дозволяють успішно вирішувати завдання процедурної генерації контенту комп'ютерної гри.

При розробці програмного забезпечення було використано об'єктно-орієнтований підхід та використано методи штучного інтелекту.

Програму розроблено на мові програмування C# у середовищі Visual Studio 2019 та Unity Engine. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються всі необхідні інструкції для застосування розробленого програмного забезпечення.

Розроблене програмне забезпечення пропонується захищати від несанкціонованого копіювання та поширення за допомогою прив'язки до даних про комп'ютер користувача. Як прив'язку до комп'ютера користувача пропонується використовувати серійний номер BIOS материнської плати та серійний номер вінчестера. Для приховування від користувача даних про захист, пропонується розташовувати їх в нерозмічену область жорсткого диску та шифрувати алгоритмом RSA.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 139872 грн. З урахуванням вартості розробки програми та обладнання, строк окупності становить 0,1 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Довідник по C # | Microsoft Docs, Ключові слова C #, Директиви препроцесора, Параметри компілятора C #

2. Архітектура персонального комп'ютера, Тема 3 - Загальні принципи архітектури комп’ютерів, 3.1 Принципи побудови комп’ютера. Архитектура Фон Неймана, 3.3 Архітектура і структура ПК

3. Седерхольм, Д. Пуленепробиваемый дизайн. Библиотека специалиста / Д. Седерхольм. - СПб.: Питер, 2012. - 304 с.

4. Джозеф Хокинг, Unity в дії. Глава 10 Звукові ефекти та музика 242с.

5. Джозеф Хокинг, Unity в дії. Глава 11 Об'єднання фрагментів в готову гру 267с.

6. Джозеф Хокинг, Unity в дії. Глава 12 Розгортання ігор на пристроях гравців 298с.

7. Е. Таненбаум, Архітектура комп’ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, глава 5. Рівень архітектури команд 334с.

8. Е. Таненбаум, Архітектура комп’ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Глава 8. Архітектури комп'ютерів паралельної дії 556с.

9. Е. Таненбаум, Архітектура комп’ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Додаток А. Двійкові числа 663с.

10. Е. Таненбаум, Архітектура комп’ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Додаток Б. Числа з плаваючою точкою 674с.

11. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 2. Лінійна алгебра 44с.

12. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 3. Теорія ймовірності і теорія інформації 61с.

13. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 5.

Основи машинного навчання 96с.

14. Бхаргава А. Грокаем алгоритми. (Пітер)

15. Кормен Томас Х., Лейзерсон Чарльз І., Ривест Рональд Л., Штайн Кліффорд, Алгоритми. Побудова і аналіз

16. «Плоский дизайн»: с чего начать? Пять основных принципов Flat дизайна [Електронний ресурс] – Режим доступу до ресурсу: http://powerbranding.ru/design/flat-design-june13/.

17. «Альтернативы для замены ArcGIS» [Електронний ресурс] – Режим доступу до ресурсу: https://ruprogi.ru/software/arcgisdesktop.

18. «.NET Framework Guide» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/framework/.

19. «Common Language Runtime (CLR) overview» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/standard/clr.

20. Рихтер Д. CLR via C# Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. – Киев: Питер, 2008. – 656 с. – (2).

21. Троелсен Э. Язык программирования C# и платформа .NET 4.5 / Эндрю Троелсен. – Киев: вильямс, 2014. – 1312 с. – (6).

22. «Visual Studio» [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/.

23. «Windows Forms overview» [Електронний ресурс] – 10. Режим доступу до ресурсу: https://docs.microsoft.com/enus/dotnet/framework/winforms/windows-forms-overview.

24. «Entity Framework Documentation» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/ef/.

25. «SQL Server 2017» [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/ru-ru/sql-server/sql-server-2017.

26. «About ArcGIS» [Електронний ресурс] – Режим доступу до ресурсу: https://www.esri.com/ru-ru/arcgis/about-arcgis/overview.

27. «Understanding Onion Architecture» [Електронний ресурс] – Режим

доступу до ресурсу: https://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/understandingonion-architecture.html.

28. «Концептуальная модель системы» [Електронний ресурс] – Режим доступу до ресурсу: http://studepedia.org/index.php?vol=1&post=2123.

29. «MVC, MVP and MVVM Design Pattern» [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvmdesign-pattern-6e169567bbad.

30. «UnitOfWork And Repository Pattern » [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@utterbbq/c-unitofwork-and-repository-pattern305cd8ecfa7a.

31. «What is ArcGIS, and where is it available at IU?» [Електронний ресурс] – Режим доступу до ресурсу: https://kb.iu.edu/d/avzt.

32. Кузін А.В «Базы данных, 5-е издание» / Кузін А.В., Левонисова С.В. – К. : «Академия», 2012. – 317 с.

33. Гольцман В.І. «MySQL 5.0. Библиотека программиста» / Гольцман В.І. – К. : «Питер», 2010. – 253 с.

34. Бен Сміт «Beginning JSON» / Бен Сміт – К. : «Apress», 2015. – 324 с.

35. Sedgewick, Robert. Algorithms in C++. Parts 5: Graph Algorithms. 3rd Ed. Addison-Wesley, 2002.

36. Роберт Седжвік. Фундаментальные алгоритмы на C++. Части 1-4: Анализ/Структуры данных/Сортировка/Поиск. - К.: Издательство ДиаСофт?, 2001.

37. Роберт Седжвік. Фундаментальные алгоритмы на C++. Часть 5: Алгоритмы на графах. - К.: Издательство ДиаСофт?, 2002.

38. Роберт Седжвік. Фундаментальные алгоритмы на C. Части 1-4: Анализ/Структуры данных/Сортировка/Поиск. - К.: Издательство ДиаСофт?, 2003.

39. Роберт Седжвік. Фундаментальные алгоритмы на C Часть 5:

Алгоритмы на графах. - К.: Издательство ДиаСофт?, 2003.

40. Джон Макгрегор, Девід Сайке. Тестирование объектно-ориентированного программного обеспечения. Практическое пособие. - К.: Издательство ДиаСофт?, 2002.

41. Knuth, Donald E. The Art of Computer Programming: Fundamental Algorithms. 3rd Ed. Addison-Wesley, 1997.

42. Knuth, Donald E. The Art of Computer Programming: Seminumerical Algorithms. 3rd Ed. Addison-Wesley, 1998.

43. Knuth, Donald E. The Art of Computer Programming: Sorting and Searching. 2nd Ed. Addison-Wesley, 1998.

44. L"Ecuyer, Pierre. "Efficient and Portable Combined Random Number Generators." Communications of the ACM, Vol. 31 (1988), pp. 742-749, 774.

45. Nelson, Mark. The Data Compression Book. M& T Publishing, 1991.

46. Park, S.K., and K.W. Miller. "Random Number Generators: Good Ones are Hard to Find." Communications of the ACM, vol. 31 (1988), pp. 1192-1201.

47. Pham, Thuan Q. and Pankaj K. Garg. Multithreaded Programming with Win32. Prentice Hall, 1999.

48. Pugh, William. "Skip Lists: A Probabilistic AItemative to Balanced Trees." Communications of the ACM, Vol. 33 (1990), pp. 668-676.

49. Robbins, John. Debugging Applications. Microsoft Press, 2000.

50. Wood, Derick. Data Structures, Algorithms, and Performance. Addison-Wesley, 1993.

51. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://goo.su/9AkQ

52. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

53. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. -

Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

54. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

55. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

56. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. - Кіровоград: КІСМ, 1997. - 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358

57. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

58. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : "Колос", 1973. – 238 с.

59. Центр післядипломної освіти та підвищення кваліфікації. - Режим доступу до ресурсу: *https://cpo.stu.cn.ua*

60. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2022. - 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення 19.09.22).

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРМ-123.22.0016.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Мельник С.О.			Дослідження та програмна реалізація системи процедурної генерації контенту комп'ютерної гри	Літ.	Аркуш 1 / Аркушів 6
Перевірив		Минайленко Р.М.				М	
Н. Контр.		Гермак В.С.				ЦНТУ КІ-21М1,4	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи процедурної генерації контенту комп’ютерної гри.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. №19-13 від 17.08.2022 року).

3 Мета та призначення розробки

Метою магістерської роботи є дослідження та програмна реалізація системи процедурної генерації контенту комп’ютерної гри.

4 Джерела розробки

Джерелом цієї магістерської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки

програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– процедурну генерацію контенту комп'ютерної гри;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Мова програмування C#, середовище розробки Unity.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2022 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи повинна бути розглянуті умови праці програмістів під час розробки програмного забезпечення.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 85 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі магістерської роботи. Постановка задачі на виконання магістерської роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень магістерської роботи.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання магістерської роботи на попередній захист 10.12.2022 р.

11.2 Подання магістерської роботи на захист .12.2022 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи
за другим (магістерським) рівнем вищої освіти
_____ Р.М. Минайленко

Дослідження та програмна реалізація системи процедурної генерації контенту комп’ютерної гри

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 49

Літера: РП

Кропивницький – 2022 року

Основна програма

Файл CameraController.cs – управління камерою

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

public class CameraController : MonoBehaviour
{
    public float panSpeed = 20f;
    public float panBorderThickness = 10f;
    public Vector2 panLimitStart;
    public Vector2 panLimitEnd;

    public float scrollSpeed = 20f;
    public float minY = 20f;
    public float maxY = 120f;

    public float rotateSpeed = 20f;
    public float minXRotate = 45f;
    public float maxXRotate = 90f;
    public bool mouseClickjudge = false;

    public bool mouseClickRotate = false;
    float xAxis, yAxis;

    public float mouseSensivity = 1f;

    void Update()
    {
        Vector3 pos = transform.position;
        if (Input.GetMouseButton(0))
        {
            pos.x += (Input.GetAxis("Mouse X") * -1) * mouseSensivity;
            pos.z += (Input.GetAxis("Mouse Y") * -1) * mouseSensivity;
        }
        if (Input.GetMouseButtonDown(1))
        {
            mouseClickRotate = true;
        }
        if (Input.GetMouseButtonUp(1))
        {
            mouseClickRotate = false;
```

```
        }
        if (Input.GetMouseButtonDown(2))
        {
            mouseClickjudge = true;
        }
        if (Input.GetMouseButtonUp(2))
        {
            mouseClickjudge = false;
        }
        if (mouseClickjudge)
        {
            yAxis += Input.GetAxis("Mouse Y") * rotateSpeed;
            yAxis = Mathf.Clamp(yAxis, minXRotate, maxXRotate);
            transform.rotation = Quaternion.Euler(yAxis,
transform.rotation.eulerAngles.y, transform.rotation.eulerAngles.z);
        }
        if (mouseClickRotate)
        {
            xAxis += Input.GetAxis("Mouse X") * rotateSpeed;
            transform.rotation =
Quaternion.Euler(transform.rotation.eulerAngles.x, xAxis,
transform.rotation.eulerAngles.z);
        }
        float scroll = Input.GetAxis("Mouse ScrollWheel");
        pos.y -= scroll * scrollSpeed * 100f * Time.deltaTime;

        pos.x = Mathf.Clamp(pos.x, panLimitStart.x, panLimitEnd.x);
        pos.y = Mathf.Clamp(pos.y, minY, maxY);
        pos.z = Mathf.Clamp(pos.z, panLimitStart.y, panLimitEnd.y);

        transform.position = pos;
    }
}
```

Файл EventBtn.cs – обробка подій натиснення на кнопки

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

public class EventBtn : MonoBehaviour
{
    [SerializeField] Text buttonText;

    public void SetBtnText(string Text)
    {
        buttonText.text = Text;
    }

    public void CallEvent()
    {
        Global.instance.isEventMenu = false;
        switch (buttonText.text)
        {
            case "Search":
                {
                    //add materials
                    Global.instance.Danger(1);
                    break;
                }
            case "Set a trap":
                {
                    Global.instance.Danger(1);
                    break;
                }
            case "Execute":
                {
                    Global.instance.EventHealth(-20);
                    Global.instance.Danger(-1);
                    break;
                }
            case "Destroy":
                {
                    Global.instance.Danger(1);
                    break;
                }
            case "Look around":
                {
                    Global.instance.Danger(1);
```

```
            break;
        }
    case "Valor":
        {
            Global.instance.EventHealth(-20);
            Global.instance.Danger(1);
            break;
        }
    case "Cunning":
        {
            Global.instance.Danger(1);
            break;
        }
    case "Pick up":
        {
            //add materials
            break;
        }
    case "Inspect":
        {
            //add materials
            Global.instance.Danger(1);
            break;
        }
    case "Help":
        {
            Global.instance.EventHealth(20);
            Global.instance.Danger(-1);
            break;
        }
    case "Use Magic":
        {
            Global.instance.EventHealth(-20);
            break;
        }
    case "Examine the equipment":
        {
            //add materials
            break;
        }
    case "Use":
        {
            Global.instance.EventHealth(20);
            break;
        }
    case "Retreat":
```

```
                {
                    break;
                }
            case "Interrogate":
                {
                    Global.instance.Danger(-1);
                    break;
                }
            case "Workaround":
                {
                    Global.instance.EventHealth(-20);
                    Global.instance.Danger(1);
                    break;
                }
            case "Caution":
                {
                    Global.instance.Danger(1);
                    break;
                }
            default:
                {
                    Debug.Log("?");
                    break;
                }

        }
    }
}
```

Файл EventManager.cs – обробка подій

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

public class EventManager: MonoBehaviour
{
    public static EventManager instance;
    public enum eventTypes {Battle, Artefacts, Mosters, People, Magic}
    [SerializeField] private int eventCouter =
System.Enum.GetNames(typeof(eventTypes)).Length;

    [SerializeField] Text eventMessage;
    [SerializeField] Image eventImage;
    [SerializeField] GameObject button1;
    [SerializeField] GameObject button2;
    [SerializeField] GameObject button3;

    [SerializeField] List<string> eventText = new List<string>();
    [SerializeField] List<Sprite> eventImages = new List<Sprite>();
    [SerializeField] List<string> eventTextButton1 = new List<string>();
    [SerializeField] List<string> eventTextButton2 = new List<string>();
    [SerializeField] List<string> eventTextButton3 = new List<string>();

    private void Awake()
    {
        instance = this;
    }

    public string CreateEvent()
    {
        int rand = Random.Range(0, eventCouter);
        switch(rand)
        {
            case 1:
                {

                    return (eventTypes.Artefacts).ToString();
                }
            case 2:
                {
                    return (eventTypes.Mosters).ToString();
                }
            case 3:
```

```
            {
                return (eventTypes.People).ToString();
            }
        case 4:
            {
                return (eventTypes.Magic).ToString();
            }
        default:
            {
                Debug.Log("Battle");
                return (eventTypes.Battle).ToString();
            }
    }
}

public void SetEvent()
{
    Room r = Global.instance.currRoom;
    switch (r.eventName)
    {
        case "Battle":
            {
                //battle
                Global.instance.Battle();
                break;
            }
        case "Artefacts":
            {
                //artef
                SetText(0, 3, 0, 2, 0);
                break;
            }
        case "Mosters":
            {
                //monster
                SetText(3, 6, 2, 4, 1);
                break;
            }
        case "People":
            {
                //people
                SetText(6, 9, 4, 6, 2);
                break;
            }
        case "Magic":
            {
```

```
                    //magic
                    SetText(9, eventText.Count,6, eventTextButton1.Count, 3);
                    break;
                }
            }
        }

    void SetText(int startEventText, int endEventText, int startEventBtnText,
int endEventBtnText, int image)
    {
        int rand = Random.Range(startEventText, endEventText);
        eventMessage.text = eventText[rand];
        eventImage.sprite = eventImages[image];
        int randBtn1 = Random.Range(startEventBtnText, endEventBtnText);
        button1.GetComponent<EventBtn>().SetBtnText(eventTextButton1[randBtn1]);
        int randBtn2 = Random.Range(startEventBtnText, endEventBtnText);
        button2.GetComponent<EventBtn>().SetBtnText(eventTextButton2[randBtn2]);
        int randBtn3 = Random.Range(startEventBtnText, endEventBtnText);
        button3.GetComponent<EventBtn>().SetBtnText(eventTextButton3[randBtn3]);
        Debug.Log((eventTypes.Magic).ToString());
    }
}
```

Файл Generation.cs – генерація лабіринту з кімнатами

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

public class Generation : MonoBehaviour
{
    [SerializeField] private GameObject tileObj;
    [SerializeField] private List<GameObject> tiles = new List<GameObject>();
    [SerializeField] private List<GameObject> roomsTypes = new
List<GameObject>();
    [SerializeField] private GameObject startTile;//
    [SerializeField] private Transform dungeon;

    [SerializeField] private int height = 10;
    [SerializeField] private int width = 10;

    [SerializeField] private int maxRooms;
    [SerializeField] int roomCounter;
    [SerializeField] int currEventedRooms = 0;
    [SerializeField] private List<GameObject> rooms = new List<GameObject>();
    [SerializeField] GameObject currRoomCreation;
    [SerializeField] GameObject prevRoomCreation;

    void SetEvent()
    {
        int eventRooms = (int)((rooms.Count-3)*0.4);
        if(currEventedRooms < eventRooms)
        {
            int rand = Random.Range(0, 1);
            switch(rand)
            {
                case 0:
                    {
                        Room room = currRoomCreation.GetComponent<Room>();
                        room.isEvent = true;
                        room.eventName = EventManager.instance.CreateEvent();
                        currEventedRooms++;
                        break;
                    }
                default:
                    {
                        break;
                    }
            }
```

```
        }
    }
    void CreateGrid()
    {
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                GameObject go = Instantiate(tileObj, dungeon);

                go.name = (x, y).ToString();
                go.transform.position = new Vector3(15 * y, 0, -x * 15);
                tiles.Add(go);
                Tile tile = go.GetComponent<Tile>();
                tile.x = x;
                tile.y = y;
            }
        }
    }
    void GetStartRoom()
    {
        int rand = Random.Range(0, tiles.Count);
        tiles[rand].GetComponent<Tile>().myTile = Tile.tileTypes.Room;
        GameObject startRoom = Instantiate(startTile, tiles[rand].transform);
        GameObject player = Instantiate(Global.instance.GetPlayer());
        player.transform.position = startRoom.transform.position;
        Camera.main.transform.position = player.transform.position + new
Vector3(0, 15, -6);
        Global.instance.MapCamera = Camera.main.gameObject;
        rooms.Add(startRoom);
        Global.instance.SetStart(startRoom);
        currRoomCreation = startRoom;
        currRoomCreation.name = (rooms.Count - 1).ToString();
        roomCounter++;
    }
    void Start()
    {
        CreateGrid();
        GetStartRoom();
        CreateDungeon();
    }

    [SerializeField] int ifStack = 0;
    [SerializeField] int tileBack = 1;
    void CreateDungeon()
    {
```

```
        while (roomCounter < maxRooms)
        {
            int rand = Random.Range(1, 6);
            switch (rand)
            {
                case 1:
                    {
                        //up
                        if(currRoomCreation != null)
                        {
                            if (tiles.Find(x => x.name ==
(currRoomCreation.transform.parent.GetComponent<Tile>().x - 1,
currRoomCreation.transform.parent.GetComponent<Tile>().y).ToString()) != null)//
parent sometimes didnt found
                            {
                                Tile tile = tiles.Find(x => x.name ==
(currRoomCreation.transform.parent.GetComponent<Tile>().x - 1,
currRoomCreation.transform.parent.GetComponent<Tile>().y).ToString()).GetCompone
nt<Tile>();
                                Room room =
currRoomCreation.GetComponent<Room>();
                                if (tile.myTile == Tile.tileTypes.None)
                                {
                                    CreateRoom(tile);
                                    prevRoomCreation.GetComponent<Room>().upDoor
= true;

prevRoomCreation.GetComponent<Room>().nearestUpRoom = currRoomCreation;

room.CheckWay(currRoomCreation.GetComponent<Room>(),
prevRoomCreation.GetComponent<Room>(), way, rand);
                                }
                                else
                                {
                                    room.cantUp = true;
                                    if (room.CheckIfStack())
                                    {
                                        ifStack++;
                                        currRoomCreation = rooms[rooms.Count - 1
- ifStack].GetComponent<GameObject>();
                                    }
                                }
                            }
                            else
                            {
```

```
                                currRoomCreation.GetComponent<Room>().cantUp =
true;
                                if
(currRoomCreation.GetComponent<Room>().CheckIfStack())
                                {
                                    ifStack++;
                                    currRoomCreation = rooms[rooms.Count - 1 -
ifStack].GetComponent<GameObject>();
                                }
                            }
                        }
                        break;
                    }
                case 2:
                    {
                        //down
                        if(currRoomCreation != null)
                        {
                            if (tiles.Find(x => x.name ==
(currRoomCreation.transform.parent.GetComponent<Tile>().x + 1,
currRoomCreation.transform.parent.GetComponent<Tile>().y).ToString()) != null)//
parent sometimes didnt found
                            {
                                Tile tile = tiles.Find(x => x.name ==
(currRoomCreation.transform.parent.GetComponent<Tile>().x + 1,
currRoomCreation.transform.parent.GetComponent<Tile>().y).ToString()).GetCompone
nt<Tile>();
                                Room room =
currRoomCreation.GetComponent<Room>();
                                if (tile.myTile == Tile.tileTypes.None)
                                {
                                    CreateRoom(tile);

prevRoomCreation.GetComponent<Room>().downDoor = true;

prevRoomCreation.GetComponent<Room>().nearestDownRoom = currRoomCreation;

room.CheckWay(currRoomCreation.GetComponent<Room>(),
prevRoomCreation.GetComponent<Room>(), way, rand);
                                }
                                else
                                {
                                    room.cantDown = true;
                                    if (room.CheckIfStack())
                                    {
                                        ifStack++;
```

```
                                    currRoomCreation = rooms[rooms.Count - 1
- ifStack].GetComponent<GameObject>();
                                }
                            }
                        }
                        else
                        {
                            currRoomCreation.GetComponent<Room>().cantDown =
true;
                            if
(currRoomCreation.GetComponent<Room>().CheckIfStack())
                            {
                                ifStack++;
                                currRoomCreation = rooms[rooms.Count - 1 -
ifStack].GetComponent<GameObject>();
                            }
                        }
                    }
                    break;
                }
            case 3:
                {
                    //left
                    if(currRoomCreation != null)
                    {
                        if (tiles.Find(x => x.name ==
(currRoomCreation.transform.parent.GetComponent<Tile>().x,
currRoomCreation.transform.parent.GetComponent<Tile>().y - 1).ToString()) !=
null)// parent sometimes didnt found
                        {
                            Tile tile = tiles.Find(x => x.name ==
(currRoomCreation.transform.parent.GetComponent<Tile>().x,
currRoomCreation.transform.parent.GetComponent<Tile>().y -
1).ToString()).GetComponent<Tile>();
                            Room room =
currRoomCreation.GetComponent<Room>();
                            if (tile.myTile == Tile.tileTypes.None)
                            {
                                CreateRoom(tile);

prevRoomCreation.GetComponent<Room>().leftDoor = true;

prevRoomCreation.GetComponent<Room>().nearestLeftRoom = currRoomCreation;

room.CheckWay(currRoomCreation.GetComponent<Room>(),
prevRoomCreation.GetComponent<Room>(), way, rand);
```

```
                                }
                                else
                                {
                                    room.cantLeft = true;
                                    if (room.CheckIfStack())
                                    {
                                        ifStack++;
                                        currRoomCreation = rooms[rooms.Count - 1
- ifStack].GetComponent<GameObject>();
                                    }
                                }
                            }
                            else
                            {
                                currRoomCreation.GetComponent<Room>().cantLeft =
true;
                                if
(currRoomCreation.GetComponent<Room>().CheckIfStack())
                                {
                                    ifStack++;
                                    currRoomCreation = rooms[rooms.Count - 1 -
ifStack].GetComponent<GameObject>();
                                }
                            }
                        }
                        break;
                    }
                case 4:
                    {
                        //right
                        if(currRoomCreation != null)
                        {
                            if (tiles.Find(x => x.name ==
(currRoomCreation.transform.parent.GetComponent<Tile>().x,
currRoomCreation.transform.parent.GetComponent<Tile>().y + 1).ToString()) !=
null)// parent sometimes didnt found
                            {
                                Tile tile = tiles.Find(x => x.name ==
(currRoomCreation.transform.parent.GetComponent<Tile>().x,
currRoomCreation.transform.parent.GetComponent<Tile>().y +
1).ToString()).GetComponent<Tile>();
                                Room room =
currRoomCreation.GetComponent<Room>();
                                if (tile.myTile == Tile.tileTypes.None)
                                {
                                    CreateRoom(tile);
```

```
prevRoomCreation.GetComponent<Room>().rightDoor = true;

prevRoomCreation.GetComponent<Room>().nearestRightRoom = currRoomCreation;

room.CheckWay(currRoomCreation.GetComponent<Room>(),
prevRoomCreation.GetComponent<Room>(), way, rand);
                                }
                                else
                                {
                                    room.cantRight = true;
                                    if (room.CheckIfStack())
                                    {
                                        ifStack++;
                                        currRoomCreation = rooms[rooms.Count - 1
- ifStack].GetComponent<GameObject>();
                                    }
                                }
                            }
                            else
                            {
                                currRoomCreation.GetComponent<Room>().cantRight
= true;
                                if
(currRoomCreation.GetComponent<Room>().CheckIfStack())
                                {
                                    ifStack++;
                                    currRoomCreation = rooms[rooms.Count - 1 -
ifStack].GetComponent<GameObject>();
                                }
                            }
                        }
                        break;
                    }
                case 5:
                    {
                        if (prevRoomCreation != null)
                        {
                            tileBack++;
                            if ((rooms.Count - 1) - tileBack >= 0)
                            {
                                currRoomCreation = prevRoomCreation;
                                prevRoomCreation = rooms[rooms.Count - 1 -
tileBack];
                            }
                            else if (rooms.Count - tileBack == 0)
```

```
                            {
                                currRoomCreation = rooms[0];
                            }
                        }
                        else
                        {
                            Debug.Log("no prev room from tupic");
                        }
                        break;
                    }
                }
            }
        }
        [SerializeField] private GameObject way;
        void CreateRoom(Tile tile)
        {
            ifStack = 0;
            tileBack = 1;

            tile.myTile = Tile.tileTypes.Room;

            int rand = Random.Range(0, roomsTypes.Count - 1);
            GameObject go = Instantiate(roomsTypes[rand], tile.transform);
            rooms.Add(go);
            go.name = (rooms.Count - 1).ToString();

            prevRoomCreation = currRoomCreation;
            currRoomCreation = go;
            SetEvent();
            roomCounter++;
        }
    }
```

Файл Global.cs – ініціалізація глобальних змінних

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using Doozy.Engine.Nody.Models;
using UnityEngine.UI;

public class Global : MonoBehaviour
{
    public bool ExitCamp = false;

    public bool isEventMenu = false;
    public static Global instance;
    [SerializeField] bool isBattle = false;
    public GameObject MapCamera;
    [SerializeField] GameObject mapPlayer;
    [SerializeField] GameObject battlePlayer;
    [SerializeField] GameObject startRoom;
    public Graph graph;

    [SerializeField] Slider healthBar;
    public int currentHealth = 100;
    public Room currRoom;
    public int dangerLvl = 0;
    [SerializeField] Text dangerText;
    [SerializeField] List<GameObject> dangerIcons = new List<GameObject>();
    [SerializeField] int currLvl = 0;
    public int totalEnemy = 0;
    [SerializeField] GameObject optionBtn;
    void DangerImage(GameObject go)
    {
        for (int i = 0; i < dangerIcons.Count; i++)
        {
            dangerIcons[i].SetActive(false);
        }
        go.SetActive(true);
    }
    public void Danger(int step)//refact
    {
        int maxlvl = 5;
        if (step > 0)
        {
            if (dangerLvl < 5)
            {
```

```
                currLvl += 1;
                if (currLvl >= maxlvl)
                {
                    dangerLvl++;
                    dangerText.text = dangerLvl.ToString();
                    currLvl -= 5;
                    for (int i = 0; i < dangerIcons.Count; i++)
                    {
                        dangerIcons[i].SetActive(false);
                    }
                }
                DangerImage(dangerIcons[currLvl]);
            }
        }
        else
        {
            if (dangerLvl > 0 || currLvl != 0)
            {
                currLvl -= 1;
                if (currLvl < 0)
                {
                    dangerLvl--;
                    dangerText.text = dangerLvl.ToString();
                    currLvl = 4;
                    for (int i = 0; i < dangerIcons.Count; i++)
                    {
                        dangerIcons[i].SetActive(false);
                    }
                }
                DangerImage(dangerIcons[currLvl]);
            }
        }
    }

    public GameObject SetStart(GameObject go)
    {
        startRoom = go;
        startRoom.GetComponent<Room>().checkedRoom = true;
        return startRoom;
    }
    public GameObject GetStart()
    {
        return startRoom;
    }
    public GameObject GetPlayer()
    {
```

```
    return mapPlayer;
}
public GameObject BPlayer()
{
    return battlePlayer;
}
public void SetHealth(int health)
{
    healthBar.value = health;
}
public void EventHealth(int health)
{
    PlayerController player = battlePlayer.GetComponent<PlayerController>();
    if ((currentHealth + health)< player.maxHealth)
        currentHealth += health;
    else
        currentHealth = player.maxHealth;

    SetHealth(currentHealth);
}
[ContextMenu("Battle")]
public void Battle()
{
    ExitCamp = false;
    StartCoroutine(LoadScene_Courutine(2, true));
    GameObject player = Instantiate(battlePlayer);
    player.transform.position = new Vector3(0, 1002, 0);
    MapCamera.SetActive(false);
    isBattle = true;
}
public void CheckExitCamp()
{
    graph.SetActiveNodeByName("ExitCamp");
}
public void StayAtCamp()
{
    ExitCamp = false;
}
public void LoadMainScene()
{
    StartCoroutine(LoadScene_Courutine(1, false));
    SetHealth(currentHealth);
}
public void LoadCampScene()
{
    optionBtn.SetActive(true);
```

```
        StartCoroutine(LoadScene_Courutine(0, false));
        ExitCamp = false;
        isBattle = false;
    }
    public void UnloadBattleScene()
    {
        SceneManager.UnloadSceneAsync("BattleScene");
        MapCamera.SetActive(true);
    }
    IEnumerator LoadScene_Courutine(int scene, bool additiveMode)
    {
        if(additiveMode)
        {
            AsyncOperation async = SceneManager.LoadSceneAsync(scene,
LoadSceneMode.Additive);
            while (!async.isDone)
            {
                yield return null;
            }
        }
        else
        {
            AsyncOperation async = SceneManager.LoadSceneAsync(scene,
LoadSceneMode.Single);
            while (!async.isDone)
            {
                yield return null;
            }
        }
    }
    public void MenuBtn(bool check)
    {
        Debug.Log("check is " + check);
        isEventMenu = check;
    }
    public bool GetBattle()
    {
        return isBattle;
    }
    private void Awake()
    {
        instance = this;
    }
}
```

Файл MapPlayerController.cs – управління ігровою мапою

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

public class MapPlayerController : MonoBehaviour
{
    [SerializeField] bool isMoving = false;
    [SerializeField] Room currentRoom;
    [SerializeField] int speed = 3;
    [SerializeField] private bool goUp = false;
    [SerializeField] private bool goLeft = false;
    [SerializeField] private bool goDown = false;
    [SerializeField] private bool goRight = false;
    void Start()
    {
        currentRoom = Global.instance.GetStart().GetComponent<Room>();
    }
    void Update()
    {
        if (!Global.instance.GetBattle() && !Global.instance.isEventMenu)
        {
            if (!isMoving)
            {
                if (Input.GetKey(KeyCode.W))
                {
                    if (currentRoom.upDoor)
                    {
                        goUp = true;
                    }
                    else
                    {
                        Debug.Log("no room up");
                    }
                }
                else if (Input.GetKey(KeyCode.A))
                {
                    if (currentRoom.leftDoor)
                    {
                        goLeft = true;
                    }
                    else
                    {
                        Debug.Log("no room left");
                    }
```

```
                }
                else if (Input.GetKey(KeyCode.S))
                {
                    if (currentRoom.downDoor)
                    {
                        goDown = true;
                    }
                    else
                    {
                        Debug.Log("no room down");
                    }
                }
                else if (Input.GetKey(KeyCode.D))
                {
                    if (currentRoom.rightDoor)
                    {
                        goRight = true;
                    }
                    else
                    {
                        Debug.Log("no room right");
                    }
                }
            }
            if (goUp)
            {
                isMoving = true;
                if (Vector3.Distance(gameObject.transform.position,
currentRoom.nearestUpRoom.transform.position) > 0.1f)
                {
                    gameObject.transform.position =
Vector3.MoveTowards(gameObject.transform.position,
currentRoom.nearestUpRoom.transform.position, speed * Time.deltaTime);
                }
                else
                {
                    goUp = false;
                    currentRoom =
currentRoom.nearestUpRoom.GetComponent<Room>();
                    StartCoroutine(WaitSeconds(1));
                }
            }
            if (goLeft)
            {
                isMoving = true;
```

```
                if (Vector3.Distance(gameObject.transform.position,
currentRoom.nearestLeftRoom.transform.position) > 0.1f)
                {
                    gameObject.transform.position =
Vector3.MoveTowards(gameObject.transform.position,
currentRoom.nearestLeftRoom.transform.position, speed * Time.deltaTime);
                }
                else
                {
                    goLeft = false;
                    currentRoom =
currentRoom.nearestLeftRoom.GetComponent<Room>();
                    StartCoroutine(WaitSeconds(1));
                }
            }
            if (goDown)
            {
                isMoving = true;
                if (Vector3.Distance(gameObject.transform.position,
currentRoom.nearestDownRoom.transform.position) > 0.1f)
                {
                    gameObject.transform.position =
Vector3.MoveTowards(gameObject.transform.position,
currentRoom.nearestDownRoom.transform.position, speed * Time.deltaTime);
                }
                else
                {
                    goDown = false;
                    currentRoom =
currentRoom.nearestDownRoom.GetComponent<Room>();
                    StartCoroutine(WaitSeconds(1));
                }
            }
            if (goRight)
            {
                isMoving = false;
                if (Vector3.Distance(gameObject.transform.position,
currentRoom.nearestRightRoom.transform.position) > 0.1f)
                {
                    gameObject.transform.position =
Vector3.MoveTowards(gameObject.transform.position,
currentRoom.nearestRightRoom.transform.position, speed * Time.deltaTime);
                }
                else
                {
                    goRight = false;
```

```
                    currentRoom =
currentRoom.nearestRightRoom.GetComponent<Room>();
                    StartCoroutine(WaitSeconds(1));
                }
            }
        }
    }
    IEnumerator WaitSeconds(int time)
    {
        if(!currentRoom.checkedRoom)
        {
            Global.instance.Danger(1);
            currentRoom.checkedRoom = true;
            if (currentRoom.isEvent)
            {
                Debug.Log("This is Event " + currentRoom.eventName);
                Global.instance.currRoom = currentRoom.GetComponent<Room>();
                EventManager.instance.SetEvent();
                if(currentRoom.eventName != "Battle")
                {
                    Global.instance.graph.SetActiveNodeByName("EventMenu");
                    Global.instance.isEventMenu = true;
                }
            }
        }
        yield return new WaitForSecondsRealtime(time);
        isMoving = false;
        // add animation of next rooms
    }
}
```

Файл Room.cs – створення ігрової кімнати

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

public class Room : MonoBehaviour
{
    public bool cantLeft = false;
    public bool cantRight = false;
    public bool cantUp = false;
    public bool cantDown = false;

    public bool upDoor = false;
    public bool rightDoor = false;
    public bool downDoor = false;
    public bool leftDoor = false;

    public GameObject upAnchor;
    public GameObject rightAnchor;
    public GameObject downAnchor;
    public GameObject leftAnchor;

    private bool isStack = false;
    int segmentsCreated = 5;
    float lerpValue = 0;

    public GameObject nearestUpRoom;
    public GameObject nearestRightRoom;
    public GameObject nearestDownRoom;
    public GameObject nearestLeftRoom;

    public GameObject plane;
    public bool checkedRoom = false;
    public bool isEvent = false;

    public string eventName;
    public bool CheckIfStack()
    {
        if (cantUp == true && cantDown == true && cantLeft == true && cantRight
== true)
            isStack = true;
        return isStack;
    }
    // 1-up 2-down 3-left 4-right
    public void CheckWay(Room curr, Room prev, GameObject way, int rand)
```

```
    {
        lerpValue = 0;
        if (rand == 1)
        {
            curr.downDoor = true;
            curr.nearestDownRoom = prev.gameObject;
            CreateWay(prev.upAnchor, curr.downAnchor, way);
        }
        else if (rand == 2)
        {
            curr.upDoor = true;
            curr.nearestUpRoom = prev.gameObject;
            CreateWay(prev.downAnchor, curr.upAnchor, way);
        }
        else if (rand == 3)
        {
            curr.rightDoor = true;
            curr.nearestRightRoom = prev.gameObject;
            CreateWay(prev.leftAnchor, curr.rightAnchor, way);
        }
        else if (rand == 4)
        {
            curr.leftDoor = true;
            curr.nearestLeftRoom = prev.gameObject;
            CreateWay(prev.rightAnchor, curr.leftAnchor, way);
        }
    }

    void CreateWay(GameObject start, GameObject finish, GameObject
createdGameObject)
    {
        for (int i = 0; i < segmentsCreated; i++)
        {
            lerpValue += 0.2f;
            Vector3 distance = Vector3.Lerp(start.transform.position,
finish.transform.position, lerpValue);
            GameObject go = Instantiate(createdGameObject, start.transform);
            go.transform.position = distance;
        }
    }
}
```

Файл PlayerController.cs – управління грою

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

public class PlayerController : MonoBehaviour
{
    public int maxHealth = 100;
    public float speed = 100;
    public Rigidbody rb;
    public GameObject bullet;
    public Vector3 bulletDirection;
    public Ray cameraRay;
    public float rayLength;
    public Plane groundPlane;
    public Vector3 pointToLook;
    public bool debaffed;
    public float debaffMultiplier;

    public void Update()
    {
        if(!Global.instance.ExitCamp)
        {
            float h = Input.GetAxis("Horizontal");
            float v = Input.GetAxis("Vertical");
            if (debaffed == true)
            {
                debaffMultiplier = 0.5f;
            }
            else debaffMultiplier = 1f;
            Vector3 tempVect = new Vector3(h, 0, v);
            tempVect = tempVect.normalized * (speed * debaffMultiplier) *
Time.deltaTime;
            rb.MovePosition(rb.transform.position + tempVect);
            groundPlane = new Plane(Vector3.up, Vector3.zero);
            cameraRay = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (groundPlane.Raycast(cameraRay, out rayLength))
            {
                pointToLook = cameraRay.GetPoint(rayLength);//find another way
to set camera;
                Debug.DrawLine(cameraRay.origin, pointToLook, Color.cyan);
                transform.LookAt(new Vector3(pointToLook.x,
transform.position.y, pointToLook.z));
            }
```

```
        }
        if(Global.instance.GetBattle())
        {
            if (Input.GetMouseButtonDown(0))
            {
                Vector3 pos = gameObject.transform.position;
                Quaternion rotation = gameObject.transform.rotation;
                GameObject bulletObj = Instantiate(bullet, pos, rotation);
                bulletObj.GetComponent<BulletController>().shooter =
gameObject.tag;
                bulletObj.GetComponent<BulletController>().damage = 20 *
debaffMultiplier;
                bulletObj.GetComponent<BulletController>().direction = new
Vector3(((Vector3)pointToLook - pos).normalized.x, 0, ((Vector3)pointToLook -
pos).normalized.z);
            }
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if ((other.CompareTag("Bullet")) &&
((other.gameObject.GetComponent<BulletController>().shooter == "Enemy") ||
(other.gameObject.GetComponent<BulletController>().shooter == "Boss")))
        {
            Global.instance.currentHealth -=
(int)(other.gameObject.GetComponent<BulletController>().damage/debaffMultiplier)
;
            Global.instance.SetHealth(Global.instance.currentHealth);
            Destroy(other.gameObject);
            if ( Global.instance.currentHealth <= 0)
            {
                Destroy(gameObject);
                Global.instance.graph.SetActiveNodeByName("Lose");
            }
        }
        else if (other.CompareTag("MeleeWeapon"))
        {
            Global.instance.currentHealth -=
(int)(other.gameObject.GetComponentInParent<EnemyController>().meleeDamage /
debaffMultiplier);
            Global.instance.SetHealth(Global.instance.currentHealth);
            if (Global.instance.currentHealth <= 0)
            {
                Destroy(gameObject);
                Global.instance.graph.SetActiveNodeByName("Lose");
```

```
            }
        }
        else if ((other.CompareTag("Enemy") &&
(other.gameObject.GetComponent<EnemyController>().enemyType == "Melee_Dash")))
        {
            Debug.Log("che za huinya");
            Global.instance.currentHealth -=
(int)(other.gameObject.GetComponent<EnemyController>().meleeDamage /
debaffMultiplier);
            Global.instance.SetHealth(Global.instance.currentHealth);
            if (Global.instance.currentHealth <= 0)
            {
                Destroy(gameObject);
                Global.instance.graph.SetActiveNodeByName("Lose");
            }
        }
        else if (other.gameObject.tag == "ExitCamp")
        {
            Debug.Log("ExitCamp");
            Global.instance.CheckExitCamp();
            Global.instance.ExitCamp = true;
        }
    }
}
```

Файл EnemySpawner.cs – створення суперників у грі

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

public class EnemySpawner : MonoBehaviour
{
    [SerializeField] List<GameObject> meleeEnemy = new List<GameObject>();
    [SerializeField] List<GameObject> rangeEnemy = new List<GameObject>();
    [SerializeField] List<GameObject> supportEnemy = new List<GameObject>();

    [SerializeField] BoxCollider plane;
    [SerializeField] Transform room;

    [SerializeField] int meleeEnemyCounter;
    [SerializeField] int rangeEnemyCounter;
    [SerializeField] int supportEnemyCounter;

    private void Awake()
    {
        meleeEnemyCounter += (Global.instance.dangerLvl / 2);
        rangeEnemyCounter += (Global.instance.dangerLvl / 2);
        supportEnemyCounter += (Global.instance.dangerLvl / 5);
    }
    void Start()
    {
        CreateEnemy(meleeEnemy, meleeEnemyCounter);
        CreateEnemy(rangeEnemy, rangeEnemyCounter);
        CreateEnemy(supportEnemy, supportEnemyCounter);
    }

    void CreateEnemy(List<GameObject> enemyList, int counter)
    {
        for(int i = 0; i < counter; i++)
        {
            int rand = Random.Range(0, enemyList.Count - 1);
            GameObject enemy = Instantiate(enemyList[rand], room);
            enemy.transform.position = GetRandomPlace(plane);
            Global.instance.totalEnemy++;
        }
    }

    Vector3 GetRandomPlace(BoxCollider mesh)
    {
```

```
        float screenX = Random.Range(mesh.bounds.min.x + 5f, mesh.bounds.max.x -
5f);
        float screenY = Random.Range(mesh.bounds.min.z + 5f, mesh.bounds.max.z -
5f);
        return new Vector3(screenX, 1002, screenY);
    }

    private void Update()
    {
        if(Global.instance.totalEnemy == 0)
        {
            Global.instance.UnloadBattleScene();
        }
    }
}
```

Файл EnemyAim.cs – створення цілей для супротивників у грі

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

public class EnemyAim : MonoBehaviour
{
    public EnemyController enemyController;

    private void OnTriggerEnter(Collider other)
    {
        Debug.Log("OnTriggerEnter");
        if (other.CompareTag("Player"))
        {
            if (!enemyController.nearPlayers.Contains(other.gameObject))
            {
                enemyController.nearPlayers.Add(other.gameObject);
            }
        }
        if ((other.CompareTag("Wall")))
        {
            if (!enemyController.nearWalls.Contains(other.gameObject))
            {
                enemyController.nearWalls.Add(other.gameObject);
            }
        }
    }

    private void OnTriggerStay(Collider other)
    {
        if ((other.CompareTag("Wall")))
        {
            if (!enemyController.nearWalls.Contains(other.gameObject))
            {
                enemyController.nearWalls.Add(other.gameObject);
            }
        }
    }
    private void OnTriggerExit2D(Collider2D collision)
    {
        Debug.Log("OnTriggerExit");
        if ((collision.CompareTag("Enemy")) || (collision.CompareTag("Boss")))
        {
            if (enemyController.nearPlayers.Contains(collision.gameObject))
            {
```

```
                enemyController.nearPlayers.Remove(collision.gameObject);
            }
        }
        if ((collision.CompareTag("Wall")))
        {
            if (enemyController.nearWalls.Contains(collision.gameObject))
            {
                enemyController.nearWalls.Remove(collision.gameObject);
            }
        }
    }
}
```

Файл BulletConroller.cs – управління кулями

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

public class BulletController : MonoBehaviour
{
    public float speed = 15.0f;
    public Vector3 direction = Vector3.zero;//check
    public float damage = 20f;
    public string shooter;
    public float delay = 3.0f;

    void Start()
    {
        StartCoroutine(WaitAndDestroy(delay));
    }

    private IEnumerator WaitAndDestroy(float delay)
    {
        yield return new WaitForSeconds(delay);
        Destroy(gameObject);
    }

    void Update()
    {
            if (direction != Vector3.zero)//check this if
            {
                gameObject.transform.position += direction * speed *
Time.deltaTime;
            }
    }
}
```

Файл EnemyController.cs – управління супротивниками у грі

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

public class EnemyController : MonoBehaviour
{
    public int health = 100;
    public float speed = 0;
    public string enemyType;
    public bool isDash;
    public bool isPunch;
    public List<GameObject> enemiesToSpawn = new List<GameObject>();
    public List<GameObject> summonedEnemies = new List<GameObject>();//check
    public Vector3 playerPrevPosition;
    public float meleeDamage = 20f;
    /// <summary>
    public float RangeToPlayer;
    public float RangeToWall;
    /// </summary>
    public GameObject bullet;
    public Vector3 bulletDirection;
    public List<GameObject> nearPlayers = new List<GameObject>();
    public List<GameObject> nearWalls = new List<GameObject>();
    public Transform nearestWall;//check
    public Transform nearestPlayer;//check
    public bool isCanShooting = true;
    public bool isCanSummoning = true;
    public bool isCanDebaffing = true;
    public Vector3 EnemyPlayerVector = new Vector2();
    public Vector3 EnemyWallVector = new Vector2();
    public float maxDistance;

    // Update is called once per frame
    void Update()
    {
        nearPlayers = nearPlayers.Where(item => item != null).ToList();
        if (nearPlayers.Count != 0)
        {
            nearestPlayer = nearPlayers.OrderBy(t =>
Vector2.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
        }
```

```
            else nearestPlayer = null;
            if (enemyType == "Range")
            {
                if (isCanShooting == true)
                    StartCoroutine(Shoot(5f));
                isCanShooting = false;
            }
            if (enemyType == "Summoner")
            {
                if (isCanSummoning == true)
                    StartCoroutine(Summoning(5f));
                isCanSummoning = false;
            }
            if (enemyType == "Debafer")
            {
                if (isCanDebaffing == true)
                    StartCoroutine(Debaff(5f));
                isCanDebaffing = false;
            }
            if (nearestPlayer != null)
            {
                Movement(enemyType);
            }
        }

        public void MovementRange(string type)
        {
            if (nearestPlayer != null)
            {
                Vector3 tempVect = Vector3.zero;
                Vector3 pos = gameObject.transform.position;
                Quaternion rotation = gameObject.transform.rotation;
                nearPlayers = nearPlayers.Where(item => item != null).ToList();
                if (nearPlayers.Count != 0)
                {
                    nearestPlayer = nearPlayers.OrderBy(t =>
    Vector3.Distance(transform.position,
    t.transform.position)).FirstOrDefault().transform;
                }
                else nearestPlayer = null;
                nearWalls = nearWalls.Where(item => item != null).ToList();
                if (nearWalls.Count != 0)
                {
                    nearestWall = nearWalls.OrderBy(t =>
    Vector3.Distance(transform.position,
    t.transform.position)).FirstOrDefault().transform;
```

```
            }
            else nearestWall = null;
            if ((nearestPlayer != null) || (nearestWall != null))
            {
                if (nearestPlayer != null)
                {
                    transform.LookAt(nearestPlayer.transform.position);
                    if (Vector3.Distance(pos, nearestPlayer.position) <
RangeToPlayer)
                    {
                        EnemyPlayerVector = (pos -
(Vector3)nearestPlayer.position).normalized;
                    }
                    else if (Vector3.Distance(pos, nearestPlayer.position) >
RangeToPlayer)
                    {
                        EnemyPlayerVector = ((Vector3)nearestPlayer.position -
pos).normalized;
                    }
                }
                if (nearestWall != null)
                {
                    if (Vector3.Distance(pos, nearestWall.position) <
RangeToWall)
                    {
                        EnemyWallVector = (pos -
(Vector3)nearestWall.position).normalized;
                    }
                }
                if ((EnemyPlayerVector != null) && (EnemyWallVector != null))
                {
                    tempVect = (EnemyWallVector + EnemyPlayerVector).normalized
/ 2;
                }
                else if ((EnemyPlayerVector != null) || (EnemyWallVector !=
null))
                {
                    if (EnemyPlayerVector != null)
                        tempVect = EnemyWallVector;
                    else if (EnemyWallVector != null)
                        tempVect = EnemyPlayerVector;
                }
            }
            if (tempVect == Vector3.zero)
            {
```

```
            }
            else tempVect = tempVect.normalized * speed * Time.deltaTime;
            maxDistance = Vector3.Distance(transform.position,
nearestPlayer.position);
            transform.position = transform.position + new
Vector3(Vector3.ClampMagnitude(tempVect, maxDistance).x, 0,
Vector3.ClampMagnitude(tempVect, maxDistance).z);
        }
    }

    public void MovementMeleeDash(string type)
    {
        if (nearestPlayer != null)
        {
            Vector3 tempVect = Vector3.zero;
            Vector3 pos = gameObject.transform.position;
            Quaternion rotation = gameObject.transform.rotation;
            nearPlayers = nearPlayers.Where(item => item != null).ToList();
            if (nearPlayers.Count != 0)
            {
                nearestPlayer = nearPlayers.OrderBy(t =>
Vector3.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
            }
            else nearestPlayer = null;
            nearWalls = nearWalls.Where(item => item != null).ToList();
            if (nearWalls.Count != 0)
            {
                nearestWall = nearWalls.OrderBy(t =>
Vector3.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
            }
            else nearestWall = null;
            if ((nearestPlayer != null) || (nearestWall != null))
            {
                if (nearestPlayer != null)
                {
                    transform.LookAt(nearestPlayer.transform.position);
                    if (Vector3.Distance(pos, nearestPlayer.position) >
RangeToPlayer)
                    {
                        EnemyPlayerVector = ((Vector3)nearestPlayer.position -
pos).normalized;
                    }
                    else
                    {
```

```
                        EnemyPlayerVector = ((Vector3)nearestPlayer.position -
pos).normalized;
                        isDash = true;
                        StartCoroutine(DashOff(2f));
                    }
                }
                if (nearestWall != null)
                {
                    if (Vector3.Distance(pos, nearestWall.position) < 3f)
                    {
                        EnemyWallVector = (pos -
(Vector3)nearestWall.position).normalized;
                    }
                }
                if ((EnemyPlayerVector != null) && (EnemyWallVector != null))
                {
                    tempVect = (EnemyWallVector + EnemyPlayerVector).normalized
/ 2;
                }
                else if ((EnemyPlayerVector != null) || (EnemyWallVector !=
null))
                {
                    if (EnemyPlayerVector != null)
                        tempVect = EnemyWallVector;
                    else if (EnemyWallVector != null)
                        tempVect = EnemyPlayerVector;
                }
            }
            if (tempVect == Vector3.zero)
            {

            }
            else
            {
                if (isDash == false)
                {
                    tempVect = tempVect.normalized * speed * Time.deltaTime;
                    maxDistance = Vector3.Distance(transform.position,
nearestPlayer.position);
                    transform.position = transform.position + new
Vector3(Vector3.ClampMagnitude(tempVect, maxDistance).x, 0,
Vector3.ClampMagnitude(tempVect, maxDistance).z);
                }
                else
                {
                    if (playerPrevPosition == Vector3.zero)
```

```
                    playerPrevPosition = nearestPlayer.position;
                tempVect = (playerPrevPosition -
transform.position).normalized;
                tempVect = tempVect.normalized * speed * 2f *
Time.deltaTime;
                maxDistance = Vector3.Distance(transform.position,
playerPrevPosition);
                transform.position = transform.position + new
Vector3(Vector3.ClampMagnitude(tempVect, maxDistance).x, 0,
Vector3.ClampMagnitude(tempVect, maxDistance).z);
            }
        }
    }
}

    public void MovementMeleePunch(string type)
    {
        if (nearestPlayer != null)
        {
            Vector3 tempVect = Vector3.zero;
            Vector3 pos = gameObject.transform.position;
            Quaternion rotation = gameObject.transform.rotation;
            nearPlayers = nearPlayers.Where(item => item != null).ToList();
            if (nearPlayers.Count != 0)
            {
                nearestPlayer = nearPlayers.OrderBy(t =>
Vector3.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
            }
            else nearestPlayer = null;
            nearWalls = nearWalls.Where(item => item != null).ToList();
            if (nearWalls.Count != 0)
            {
                nearestWall = nearWalls.OrderBy(t =>
Vector3.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
            }
            else nearestWall = null;
            if ((nearestPlayer != null) || (nearestWall != null))
            {
                if (nearestPlayer != null)
                {
                    if (Vector3.Distance(pos, nearestPlayer.position) >
RangeToPlayer)
                    {
```

```
                        EnemyPlayerVector = ((Vector3)nearestPlayer.position -
pos).normalized;
                    }
                    else
                    {
                        EnemyPlayerVector = ((Vector3)nearestPlayer.position -
pos).normalized;
                        isPunch = true;
                        StartCoroutine(PunchOff(4f));
                    }
                }
                if (nearestWall != null)
                {
                    if (Vector3.Distance(pos, nearestWall.position) < 3f)
                    {
                        EnemyWallVector = (pos -
(Vector3)nearestWall.position).normalized;
                    }
                }
                if ((EnemyPlayerVector != null) && (EnemyWallVector != null))
                {
                    tempVect = (EnemyWallVector + EnemyPlayerVector).normalized
/ 2;
                }
                else if ((EnemyPlayerVector != null) || (EnemyWallVector !=
null))
                {
                    if (EnemyPlayerVector != null)
                        tempVect = EnemyWallVector;
                    else if (EnemyWallVector != null)
                        tempVect = EnemyPlayerVector;
                }
            }
            if (tempVect == Vector3.zero)
            {

            }
            else
            {
                if (isPunch == false)
                {
                    tempVect = tempVect.normalized * speed * Time.deltaTime;
                    maxDistance = Vector3.Distance(transform.position,
nearestPlayer.position);
```

```
                    transform.position = transform.position + new
Vector3(Vector3.ClampMagnitude(tempVect, maxDistance).x, 0,
Vector3.ClampMagnitude(tempVect, maxDistance).z);
                }
                else
                {
                    if (playerPrevPosition == Vector3.zero)
                        playerPrevPosition = nearestPlayer.position;
                    tempVect = (playerPrevPosition -
transform.position).normalized;
                    tempVect = tempVect.normalized * speed * 2f *
Time.deltaTime;
                    maxDistance = Vector3.Distance(transform.position,
playerPrevPosition);
                    transform.position = transform.position + new
Vector3(Vector3.ClampMagnitude(tempVect, maxDistance).x, 0,
Vector3.ClampMagnitude(tempVect, maxDistance).z);
                    transform.RotateAround(transform.position, Vector3.up, 360 *
Time.deltaTime);
                }
            }
        }
    }

    public void MovementSummoner(string type)
    {
        if (nearestPlayer != null)
        {
            Vector3 tempVect = Vector3.zero;
            Vector3 pos = gameObject.transform.position;
            Quaternion rotation = gameObject.transform.rotation;
            nearPlayers = nearPlayers.Where(item => item != null).ToList();
            if (nearPlayers.Count != 0)
            {
                nearestPlayer = nearPlayers.OrderBy(t =>
Vector3.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
            }
            else nearestPlayer = null;
            nearWalls = nearWalls.Where(item => item != null).ToList();
            if (nearWalls.Count != 0)
            {
                nearestWall = nearWalls.OrderBy(t =>
Vector3.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
            }
```

```
            else nearestWall = null;
            if ((nearestPlayer != null) || (nearestWall != null))
            {
                if (nearestPlayer != null)
                {
                    transform.LookAt(nearestPlayer.transform.position);
                    if (Vector3.Distance(pos, nearestPlayer.position) <
RangeToPlayer)
                    {
                        EnemyPlayerVector = (pos -
(Vector3)nearestPlayer.position).normalized;
                    }
                    else if (Vector3.Distance(pos, nearestPlayer.position) >
RangeToPlayer)
                    {
                        EnemyPlayerVector = ((Vector3)nearestPlayer.position -
pos).normalized;
                    }
                }
                if (nearestWall != null)
                {
                    if (Vector3.Distance(pos, nearestWall.position) <
RangeToWall)
                    {
                        EnemyWallVector = (pos -
(Vector3)nearestWall.position).normalized;
                    }
                }
                if ((EnemyPlayerVector != null) && (EnemyWallVector != null))
                {
                    tempVect = (EnemyWallVector + EnemyPlayerVector).normalized
/ 2;
                }
                else if ((EnemyPlayerVector != null) || (EnemyWallVector !=
null))
                {
                    if (EnemyPlayerVector != null)
                        tempVect = EnemyWallVector;
                    else if (EnemyWallVector != null)
                        tempVect = EnemyPlayerVector;
                }
            }
            if (tempVect == Vector3.zero)
            {

            }
```

```
            else tempVect = tempVect.normalized * speed * Time.deltaTime;
            maxDistance = Vector3.Distance(transform.position,
nearestPlayer.position);
            transform.position = transform.position + new
Vector3(Vector3.ClampMagnitude(tempVect, maxDistance).x, 0,
Vector3.ClampMagnitude(tempVect, maxDistance).z);
        }
    }

    public void MovementDebafer(string type)
    {
        if (nearestPlayer != null)
        {
            Vector3 tempVect = Vector3.zero;
            Vector3 pos = gameObject.transform.position;
            Quaternion rotation = gameObject.transform.rotation;
            nearPlayers = nearPlayers.Where(item => item != null).ToList();
            if (nearPlayers.Count != 0)
            {
                nearestPlayer = nearPlayers.OrderBy(t =>
Vector3.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
            }
            else nearestPlayer = null;
            nearWalls = nearWalls.Where(item => item != null).ToList();
            if (nearWalls.Count != 0)
            {
                nearestWall = nearWalls.OrderBy(t =>
Vector3.Distance(transform.position,
t.transform.position)).FirstOrDefault().transform;
            }
            else nearestWall = null;
            if ((nearestPlayer != null) || (nearestWall != null))
            {
                if (nearestPlayer != null)
                {
                    if (Vector3.Distance(pos, nearestPlayer.position) >
RangeToPlayer)
                    {
                        EnemyPlayerVector = ((Vector3)nearestPlayer.position -
pos).normalized;
                    }
                    else
                    {
                        EnemyPlayerVector = ((Vector3)nearestPlayer.position -
pos).normalized;
```

```
                        isPunch = true;
                        StartCoroutine(PunchOff(4f));
                    }
                }
                if (nearestWall != null)
                {
                    if (Vector3.Distance(pos, nearestWall.position) < 3f)
                    {
                        EnemyWallVector = (pos -
(Vector3)nearestWall.position).normalized;
                    }
                }
                if ((EnemyPlayerVector != null) && (EnemyWallVector != null))
                {
                    tempVect = (EnemyWallVector + EnemyPlayerVector).normalized
/ 2;
                }
                else if ((EnemyPlayerVector != null) || (EnemyWallVector !=
null))
                {
                    if (EnemyPlayerVector != null)
                        tempVect = EnemyWallVector;
                    else if (EnemyWallVector != null)
                        tempVect = EnemyPlayerVector;
                }
            }
            if (tempVect == Vector3.zero)
            {

            }
            else
            {
                if (isPunch == false)
                {
                    tempVect = tempVect.normalized * speed * Time.deltaTime;
                    maxDistance = Vector3.Distance(transform.position,
nearestPlayer.position);
                    transform.position = transform.position + new
Vector3(Vector3.ClampMagnitude(tempVect, maxDistance).x, 0,
Vector3.ClampMagnitude(tempVect, maxDistance).z);
                }
                else
                {
                    if (playerPrevPosition == Vector3.zero)
                        playerPrevPosition = nearestPlayer.position;
```

```
                    tempVect = (playerPrevPosition -
transform.position).normalized;
                    tempVect = tempVect.normalized * speed * 2f *
Time.deltaTime;
                    maxDistance = Vector3.Distance(transform.position,
playerPrevPosition);
                    transform.position = transform.position + new
Vector3(Vector3.ClampMagnitude(tempVect, maxDistance).x, 0,
Vector3.ClampMagnitude(tempVect, maxDistance).z);
                    transform.RotateAround(transform.position, Vector3.up, 360 *
Time.deltaTime);
                }
            }
        }
    }

    public void Movement(string type)
    {
        if (type == "Range")
        {
            MovementRange(type);
        }
        else if (type == "Melee_Dash")
        {
            MovementMeleeDash(type);
        }
        else if (type == "Melee_Punch")
        {
            MovementMeleePunch(type);
        }
        else if (type == "Summoner")
        {
            MovementSummoner(type);
        }
        else if (type == "Debafer")
        {
            MovementDebafer(type);
        }
    }
    IEnumerator Shoot(float time)
    {
        yield return new WaitForSeconds(time);
        if (this.enemyType == "Range")
        {
            if (nearestPlayer != null)
            {
```

```
                Vector3 pos = gameObject.transform.position;
                Quaternion rotation = gameObject.transform.rotation;
                GameObject bulletObj = Instantiate(bullet, pos, rotation);
                bulletObj.GetComponent<BulletController>().shooter =
gameObject.tag;
                bulletObj.GetComponent<BulletController>().direction = new
Vector3((nearestPlayer.position - pos).normalized.x, 0, (nearestPlayer.position
- pos).normalized.z);
            }
            isCanShooting = true;
        }
    }

    IEnumerator Summoning(float time)
    {
        yield return new WaitForSeconds(time);
        if (this.enemyType == "Summoner")
        {
            summonedEnemies = summonedEnemies.Where(item => item !=
null).ToList();
            if ((nearestPlayer != null) && (summonedEnemies.Count < 2))
            {
                System.Random rand = new System.Random();
                int i = rand.Next(0, enemiesToSpawn.Count);
                GameObject summonedEnemy =
Instantiate<GameObject>(enemiesToSpawn[i]);
                summonedEnemy.transform.position = this.transform.position + new
Vector3(2, 0, 2);
                summonedEnemies.Add(summonedEnemy);

            }
            isCanSummoning = true;
        }
    }

    IEnumerator Debaff(float time)
    {
        if (this.enemyType == "Debafer")
        {
            if (nearestPlayer != null)
            {
                nearestPlayer.GetComponent<PlayerController>().debaffed = true;
            }
            yield return new WaitForSeconds(time);
            nearestPlayer.GetComponent<PlayerController>().debaffed = false;
            isCanDebaffing = true;
```

```
        }
    }

    IEnumerator DashOff(float time)
    {
        yield return new WaitForSeconds(time);
        isDash = false;
        playerPrevPosition = Vector3.zero;
    }

    IEnumerator PunchOff(float time)
    {
        yield return new WaitForSeconds(time);
        isPunch = false;
        playerPrevPosition = Vector3.zero;
    }

    public void OnTriggerEnter(Collider other)
    {
        if ((other.CompareTag("Bullet")) &&
(other.gameObject.GetComponent<BulletController>().shooter == "Player"))
        {
            health = health - 20;
            Destroy(other.gameObject);
            if (health <= 0)
            {
                Destroy(gameObject);
                Global.instance.totalEnemy--;
            }
        }
    }
}
```