Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК _____

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “Розробка комп'ютерної 3Д гри жанру шутер з елементами штучного інтелекту”.

Спеціальність: 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ - _____

Студент ІПЗ-43

Денис ШУСТ

(дата) (підпис) (розшифровка підпису)

Науковий керівник

к. т. н., доц. Катерина МЕРКУЛОВА

(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Консультант з питань нормоконтролю

фахівець Тамара ЧАПОВСЬКА

(посада) (підпис) (дата) (розшифровка підпису)

Завідувач кафедри

д.т.н., проф. Олексій БИЧКОВ

(посада) (підпис) (дата) (розшифровка підпису)

Київ – 2021

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Освітньо-кваліфікаційний рівень бакалавр
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____(Олексій БИЧКОВ)

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
СТУДЕНТУ

Шусту Денису Олександровичу
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної бакалаврської роботи “Розробка комп'ютерної 3Д гри жанру шутер з елементами штучного інтелекту”

керівник проекту (роботи) Меркулова Катерина Володимирівна, к.т.н., доцент

затверджена наказом вищого навчального закладу від „11” листопада 2020 р. №6

2. Строк здачі студентом закінченої роботи _____

3. Вихідні дані до роботи Теоретичні та практичні навички створення комп’ютерних ігор. Теоретичний матеріал на тему штучного інтелекту, та методів його створення. Потреби користувачів, які грають у комп’ютерні ігри.

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)

1. Дослідити особливості розробки комп’ютерних ігор
2. Проаналізувати методи створення штучного інтелекту
3. Спроектувати та розробити гру
4. Реалізувати в розробленій грі штучний інтелект

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)

1. Типи машинного навчання ШІ (рис. 1.2, ст. 13)
2. State diagram (Діаграма станів) ігрового застосунку (рис. 2.1, ст. 21)
3. Діаграма компонентів об’єкту Player (рис. 2.2, ст. 22)
4. Діаграма діяльності, що описує механіку зброї (рис. 2.3, ст. 23)
5. Компоненти об’єкту гравця (рис. 2.19, ст. 40)

6. Use case діаграма (діаграма прецедентів) головного персонажа гравця (рис. 2.24, ст. 45)
7. Параметри конфігурації (табл. 3.1, ст. 56)
8. Файл конфігурації AgentConfig.yaml (рис. 3.8, ст. 58)
9. Графіки нагороди усіх трьох тренувань (рис. 3.13, ст. 62)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. **Дата видачі завдання** _____

Керівник _____(Катерина МЕРКУЛОВА)

Завдання прийняв до виконання_____(Денис ШУСТ)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Огляд теоретичного матеріалу по завданню роботи	22.09.2020	виконано
2	Аналіз методів та алгоритмів	15.10.2020	виконано
3	Аналіз аналогів та прототипів розроблюваного продукту	03.11.2020	виконано
4	Проектування гри та сюжету	19.02.2021	виконано
5	Розробка графічної складової гри	12.03.2021	виконано
6	Реалізація програмної частини гри	28.03.2021	виконано
7	Розробка штучного інтелекту та впровадження його у гру	25.04.2021	виконано
8	Аналіз готового продукту та оформлення дипломної роботи	29.05.2021	виконано

Студент – бакалавр _____(Денис ШУСТ)

Керівник роботи _____(Катерина МЕРКУЛОВА)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота містить 87 сторінок, 50 рисунків, 5 таблиць, 18 джерел та 8 додатків загальним обсягом 18 сторінок.

Тема: Розробка комп'ютерної 3Д гри жанру шутер з елементами штучного інтелекту

Мета роботи: Дослідити методи машинного навчання штучного інтелекту та створити програмний продукт – гру з використанням штучного інтелекту.

Об’єкти дослідження: Індустрія відеоігор ; Методи створення штучного інтелекту

Предмети дослідження: Ігри зі штучним інтелектом ; Машинне навчання штучного інтелекту

Завдання для досягнення мети роботи:

- Дослідити особливості розробки комп’ютерних ігор
- Проаналізувати методи створення ШІ
- Розробити гру з реалізованим ШІ

Одержані результати: Досліджено особливості розробки відеоігор. Створено та реалізовано у розробленій грі машинно навчений штучний інтелект.

Практичне значення одержаних результатів: Дана робота може використовуватися для покращення аналізу методів машинного навчання за допомогою ігрових симуляцій.

Основна частина роботи розділена на 3 розділи.

У першому розділі наведені результати дослідження предметної області та аналізу методів розробки ігор та штучного інтелекту.

У другому розділі описані методи розробки основної частини програмного продукту, а також висвітлені реалізовані алгоритми.

У третьому розділі описаний процес розробки машинно навченого штучного інтелекту та його реалізація в грі.

Ключові слова: ШТУЧНИЙ ІНТЕЛЕКТ, АЛГОРИТМ, МАШИННЕ НАВЧАННЯ, ВІДЕОГРА, НЕЙРОННА МЕРЕЖА

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа содержит 87 страниц, 50 рисунков, 5 таблиц, 18 источников и 8 приложений объемом 18 страниц.

Тема: Разработка компьютерной 3Д игры жанра шутер с елементами искусственного интеллекта.

Цель работы: Исследовать методы машинного обучения искусственного интеллекта и создать полноценный программный продукт - игру с использованием искусственного интеллекта.

Объекты исследования: Индустрия видеоигр; Методы создания искусственного интеллекта.

Предметы исследования: Игры с искусственным интеллектом; Машинное обучение искусственного интеллекта.

Задачи для достижения цели работы:

• Исследовать особенности разработки компьютерных игр

• Проанализировать методы создания ИИ

• Разработать игру с реализованным ИИ

Полученные результаты: Исследованы особенности разработки видеоигр. Создан и реализован в игре машинно обученный искусственный интеллект.

Практическое значение полученных результатов: Данная работа может использоваться для улучшения анализа методов машинного обучения с помощью игровых симуляций.

Основная часть работы разделена на 3 раздела.

В первом разделе приведены результаты исследования предметной области и анализа методов разработки игр и искусственного интеллекта.

Во втором разделе описаны методы разработки основной части программного продукта, а также освещены реализованы алгоритмы.

В третьем разделе описан процесс разработки машинно обученного искусственного интеллекта и его реализация в игре.

Ключевые слова: ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, АЛГОРИТМ, МАШИННОЕ ОБУЧЕНИЕ, ВИДЕОИГРЫ, НЕЙРОННЫЕ СЕТИ

ANNOTATION

The final qualifying bachelor's work contains 87 pages, 50 images, 5 tables, 18 sources and 8 applications of total volume 18 pages.

Topic: Development of a computer 3D shooter game with elements of artificial intelligence.

Goal: To study the methods of machine learning of artificial intelligence and create a full-fledged software product - a game with artificial intelligence.

Research objects: Industry of video games; Methods of creating artificial intelligence.

Research subjects: Games with artificial intelligence; Artificial intelligence machine learning.

Tasks to achieve the goal:

• Investigate the features of the development of computer games

• Analyze methods of creating AI

• Develop a game with implemented AI

Results: Studied the features of the development of video games. A machine learning artificial intelligence has been created and implemented in the game.

Practical significance of the obtained results: This work can be used to improve the analysis of machine learning methods using game simulations.

The main part of the work is divided into 3 sections.

The first section presents the results of a domain research and analysis of game development and artificial intelligence methods.

The second section describes methods for developing the main part of the software product, and also highlights the implemented algorithms.

The third section describes the process of developing machine learning artificial intelligence and its implementation in the game.

Keywords: ARTIFICIAL INTELLIGENCE, ALGORITHM, MACHINE LEARNING, VIDEO GAMES, NEURAL NETWORK

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Префаб – ігровий об’єкт, збережений в файлах проекту

ШІ – штучний інтелект

Скрипт – файл з кодом окремого класу

RPG - Role Play Game – Рольова гра

Агент - об’єкт, який являється носієм штучного інтелекту

ВСТУП

Ігри допомагають нам проявити позитивні емоції, такі як цікавість, оптимізм та креативність. Ці емоції залишаються в силі протягом тривалого часу після гри. Проте, ігри вже давно перестали бути призначеними просто для розваги. Сьогодні ігри - це сучасний спосіб взаємодії та навчання людей. А також потужний інструмент для дослідження наукових теорій, або штучного інтелекту.

Тема роботи є актуальною, оскільки в ній було досліджено процес створення гри та методи реалізації штучного інтелекту. Також було проаналізовано особливості машинного навчання ШІ.

Метою роботи було створити повноцінний програмний продукт – гру з використанням штучного інтелекту. Для цього було досліджено особливості розробки комп'ютерних ігор, проаналізовано методи створення ШІ та розроблено гру, реалізувавши в ній машинно навчений ШІ.

Об'єкти дослідження : Індустрія відеоігор ; Методи створення штучного інтелекту

Предмети дослідження : Ігри зі штучним інтелектом ; Машинне навчання штучного інтелекту

Говорячи про новизну одержаних результатів, потрібно сказати, що було створено абсолютно унікальну ігрову систему, зі своїми особливими механіками та наповненням. А також було одержано навчений за допомогою нейронної мережі штучний інтелект, та аналіз процесу його навчання на підібраних параметрах.

Результати робота можуть бути використані для покращення аналізу методів машинного навчання за допомогою ігрових симуляцій.

За результатами наукових досліджень, проведених у бакалаврській роботі, було опубліковано тези в збірнику узагальнених матеріалів конференції MSTIoE 2021-8 (8-ма Східно-Європейська конференція "Математичні та програмні технології Internet of Everything")

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ДОСЛІДЖЕННЯ ТЕОРЕТИЧНОЇ БАЗИ

1.1. Аналіз індустрії відеоігор

Сьогодні неможливо уявити світ без мобільних пристроїв, комп'ютерів, розумної техніки та інших проявів розвитку технічного прогресу. Технічна революція та впровадження комп'ютерів та телефонів у кожен дім дала початок розвитку ігрової індустрії. Людство зрозуміло, що машини, які були створені для полегшення життєдіяльності, можуть бути чудовими ігровими пристроями.

Розробка комп'ютерних ігор стала однією з найбільших сучасних індустрій, маючи свої напрямки та особливості. Існує незліченна кількість любителів пограти у відеогру. У них грають люди будь-якого віку та національності. Отже, попит на таку продукцію росте і це забезпечує мільйони робочих місць, та колосальні обороти коштів у даній індустрії.

Розробкою гри може займатися одна людина, а може ціла команда розробників з десятків, а подекуди й сотень людей. Кожен етап в розробці гри може відбуватися власними силами з обмеженим бюджетом, а може і спираючись на фінансування від видавця. Нерідко буває, що гру надають видавцю вже після її розробки, задля того, щоб вона набрала популярності та почала приносити прибутки.

Кожна гра розпочинається із ідеї та концепції. У ній також визначаються основні механіки гри, реалізація головної ідеї, сюжет, рівні, методи взаємодії з гравцем та інше. Після цього розробники ігор беруться за визначення стилю дизайну гри та її програмну частину. Головна ціль при розробці гри – якісно та чітко реалізувати спроектовану модель гри, слідуючи ідеї, щоб створити цікаву та унікальну гру, яка буде заохочувати гравця грати в неї.

1.2. Методи реалізації штучного інтелекту у іграх

Одним з найскладніших та водночас найдієвіших методів зробити будь-яку гру цікавішою та унікальною є реалізація в ній штучного інтелекту. У сфері інформаційних технологій створення штучного інтелекту є одним із найперспективніших напрямів. А саме ігрова індустрія являється найбільш розвинутою сферою з використанням ШІ. Впровадження ШІ у гру робить її більш повноцінною, розширюючи можливості як розробників, так і гравців.

Ігровий штучний інтелект можна реалізувати кількома методами. Простими вважаються методи, коли за дії штучного інтелекту відповідає програмний код, або певна схема, у якій чітко визначено, за яких обставин які дії має виконувати ігровий агент, наприклад на рис. 1.1

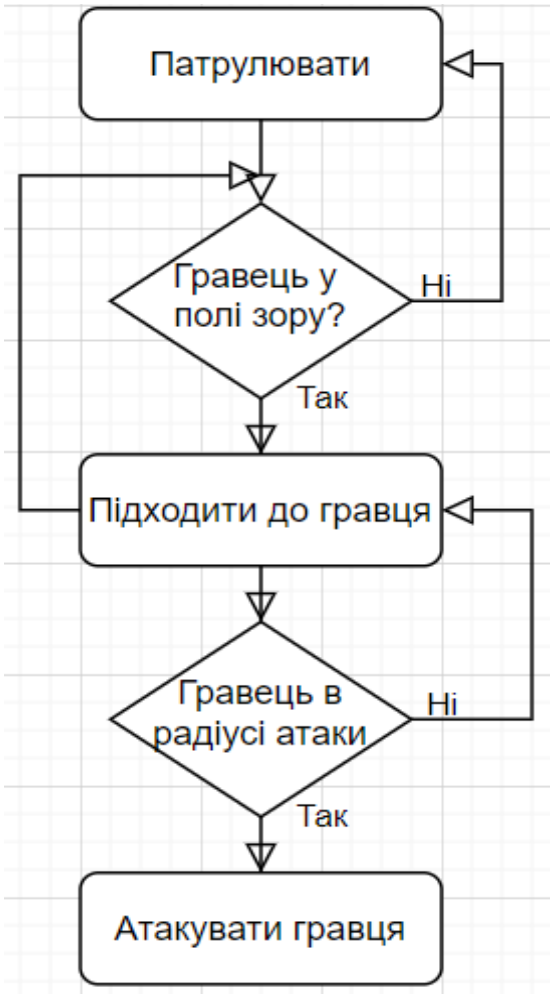

Рис. 1.1 Принцип роботи простого ігрового ШІ

Складнішим ж методом являється створення ШІ за допомогою методів машинного навчання. Проте цей метод дозволяє створити штучний інтелект таким, ніби він справді вміє мислити, кожен агент з таким інтелектом буде аналізувати середовище та виконувати дії завдяки нейронній мережі.

Для виконання завдання було створено штучний інтелект навчений за допомогою методу Reinforcement machine learning. Даний метод можна охарактеризувати як “Навчання через помилки і успіх”. При навчанні агенту даються вказівки, що робити погано, а що добре. Як саме робити те, що “добре” агент дізнається сам, аналізуючи свої попередні дії і результат за них.

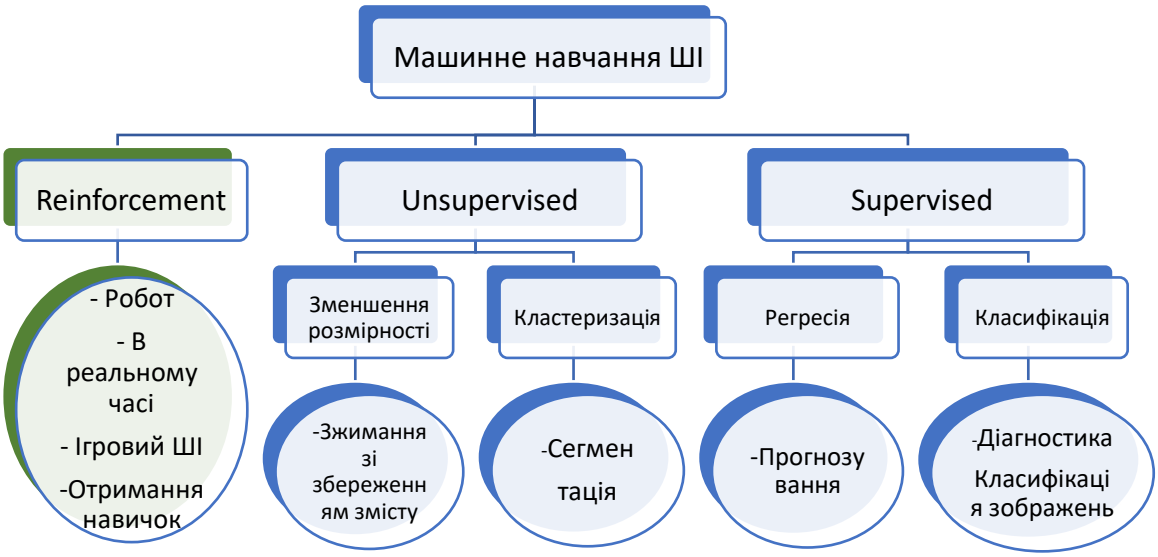

Рис. 1.2 Типи машинного навчання ШІ

1.3. Дослідження жанру та аналогів

1.3.1. Жанр розробленої гри

Гра “Hexagon Keepers” – це трьохвимірна гра жанру шутер від першого лиця. В ній містяться елементи жанру Action-RPG (бойова рольова гра).

Оскільки гра трьохвимірна, це означає, що всі події відбуваються у трьохвимірному просторі, тобто гравець може переміщувати персонажа по трьом осям координат (x,y,z).

Шутер від першої особи, або FPS, що означає First-person shooter - це жанр відеоігор, що являється підвидом жанру Шутер. В основі ігрового процесу лежить знищення ворогів різними видами зброї. Характерною особливістю гри від першого лиця, є те, що ігрова камера розташована там, де знаходяться очі ігрового персонажа, та реалізована так, що має здатність вільно рухатися, імітуючи рух голови персонажа. Таким чином відбувається імітація виду «з очей» головного героя. Саме таку реалізацію було використано при створенні гри “Hexagon Keepers”.

У грі “Hexagon Keepers”, як і в будь-якій FPS грі гравець керує одним персонажем, що може використовувати певну кількість ігрових предметів та зброї для виконання основного завдання - знищення ворогів. У шутерах гравцям зазвичай пояснюють, ким є ігровий персонаж, яка його основна ціль і чому йому потрібно боротися з противниками. У грі “Hexagon Keepers” було реалізовано подачу сюжету прямо під час гри, у коротких діалогах з ігровими персонажами.

Певною мірою гру можна віднести до жанру Action-RPG (бойова рольова гра). Action-RPG - це те ж саме, що і жанр RPG, але в ній основна увага загострена саме на такому елементі ігрового процесу, як боротьба з противниками.

Всі рівні розробленої гри являють собою суцільне поле бою, на яких майже нічого крім ворожих монстрів немає, а отже боротися потрібно постійно. Існують лише невеликі локації, на яких гравець може перепочити, завершуючи виконані завдання, просуваючись по сюжету.

Рольова відеогра, або RPG, що означає Role-Playing Game - жанр відеоігор, де основна частина ігрового процесу полягає в управлінні персонажем, які досліджують ігровий світ, виконують різноманітні завдання, та розвиваються, слідуючи сюжету.

У грі “Hexagon Keepers” потрібно керувати одним єдиним персонажем – “Блукаючим Мандрівником” , що має свою роль у ігровому світі. Гравцеві потрібно досліджувати світ, боротися з великою кількістю ворожих створінь, та дізнаватися дедалі більше деталей сюжету гри.

Ще однією особливістю створеної гри, за якою її можна віднести до жанру Action-RPG є те, що гравцю доведеться протистояти великій кількості противників, кожен з яких має особливі атрибути, що роблять їх вразливими, або ж навпаки захищеними від відповідної зброї гравця.

1.3.2. Аналоги

Досліджуючи аналоги, можна взяти за приклад гру WARFRAME - це безкоштовний науково-фантастичний шутер від третьої особи. Дана гра була розроблена Digital Extremes у 2013 році.

Події в Warframe розгортаються в далекому майбутньому, де в Сонячній системі ворогують дві сторони – “Корпус”, культ купців-мегакорпорацій із передовою робототехнікою та лазерними технологіями і “Заражені” - назва хвороби та її жертв, яка пожирає всіх. Гравці виступають у ролі древнього воїна, створеного для боротьби з таємничим ворогом.

Гравці завжди протистоять ворогам у більшій кількості. З цією метою гравці мають у своєму розпорядженні різноманітний арсенал зброї, як архаїчної, так і сучасної. Також, кожен персонаж гравця має певний набір здібностей, що значно допомагають в бою. Незважаючи на те, що це в основному гра-шутер, геймплей має певний фокус на елементи RPG-гри.

При розробці гри “Hexagon Keepers” виникло завдання зробити схожий шутер від першого лиця, але з складнішою технікою боротьби з противниками. Для цього, навідмінно від WARFRAME у якій у гравця з самого початку рівня є вся потрібна йому зброя, було зроблено так, щоб гравець мав можливість підбирати зброю з поля бою, та використовував її. Цей метод забезпечує постійну динаміку ігрового процесу, при якому гравцеві завжди потрібно шукати правильний тип зброї, та вирішувати, використати її в той же момент, чи покращити, для більшої ефективності. Також у розробленій грі було реалізовано машинно навчений ШІ.

1.4. Аналіз прототипів гри що розроблено

“Hexagon Keepers” була створена комбінацією механік двох інших раніше створених ігор, зі значними доповненнями. З однієї гри була взята основна механіка розподілу противників на різні класи та зброї з різними атрибутами, що по своєму впливають на відповідні класи противників. З іншої ж гри була взята механіка генерації противників та зброї на ігровому полі, а також спосіб взаємодії гравця зі зброєю.

Однією з ігор, що дала початок грі “Hexagon Keepers” є гра “Random War”, що створювалася у вільний від навчання час на 2 курсі. Ця гра являє собою текстову гру жанру Roguelike всі події якої відбуваються в консолі.

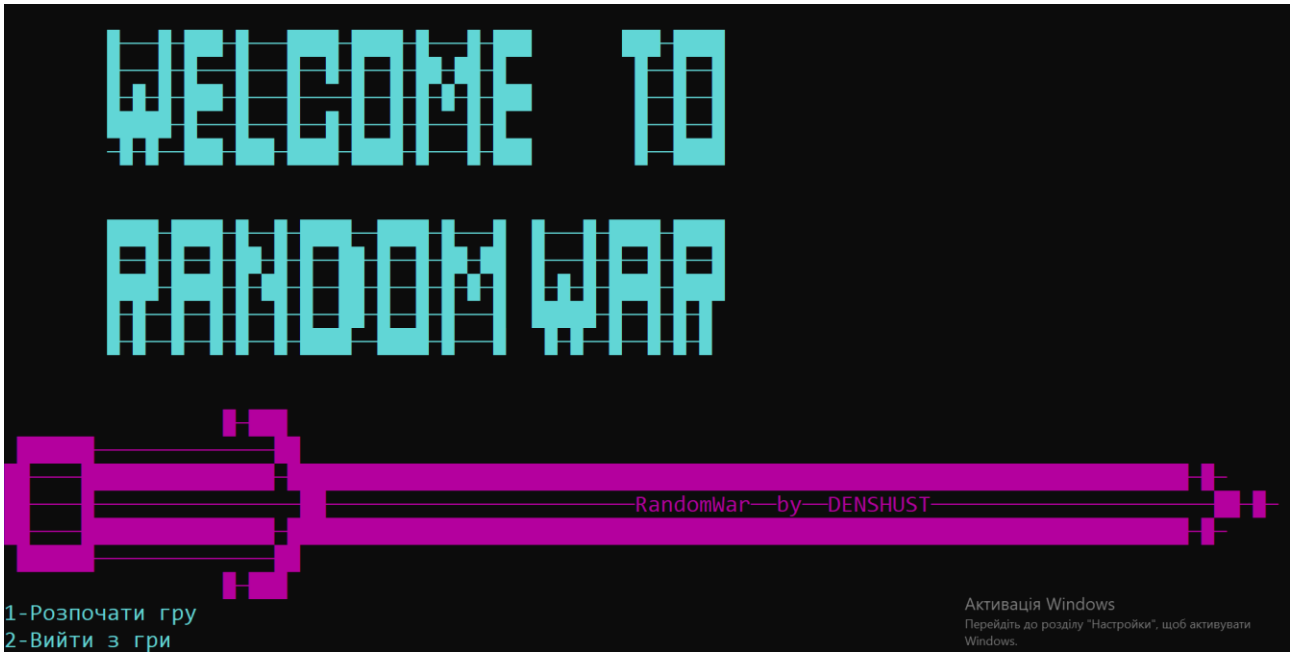

Рис. 1.3 Вітальний напис при запуску гри Random War

Roguelike, тобто rogue-подібні ігри — це піджанр рольових відеоігор, у яких майже завжди наявна випадкова генерація рівнів та/або противників, покроковий ігровий процес, та якщо персонаж помирає, то гра розпочинається з самого початку. Саме так була реалізована гра “Random War”, у якій противники генеруються у випадковому порядку, а ігровий процес відбувається покроково,

після кожної дії гравця. У даній грі є 6 типів зброї, вони продемонстровані на рисунку 1.4

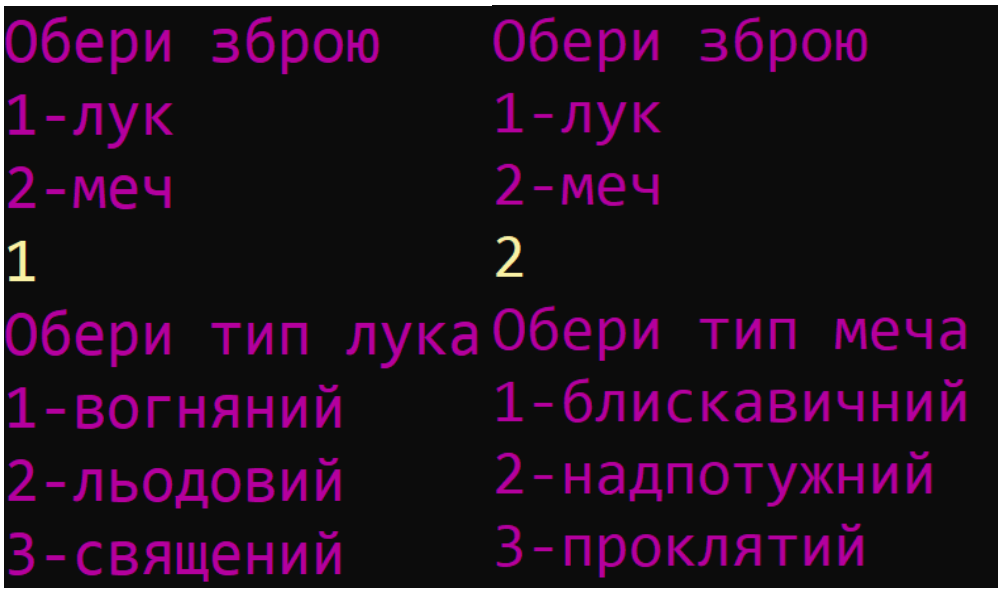

Рис. 1.4 Скріншот гри, що демонструє різноманіття зброї

Кожен тип зброї має свої параметри при виборі, наприклад будь-який лук буде мати меншу швидкість атаки, але більшу силу, ніж мечі. Різні мечі та луки також відрізняються своїми характеристиками :

```
case 1:
    sword = "блискавичний";
    heroSpeed -= 10;
    break;
case 2:
    sword = "надпотужний";
    heroDamage += 25;
    break;
case 3:
    sword = "проклятий";
    heroHealth += 200;
    heroMaxHealth += 200;
    break;
```

```
case 1:
    bow = "вогняний";
    heroSpeed -= 10;
    break;
case 2:
    bow = "льодовий";
    heroDamage += 20;
    break;
case 3:
    bow = "священий";
    heroHealth += 200;
    heroMaxHealth += 200;
    break;
```

Рис. 1.5 Частина коду гри, що відповідає за базові параметри зброї

Ще одним основним компонентом гри є те, що в ній наявні 9 типів противників, кожен з яких має особливі атрибути, і відповідно кожна зброя по різному впливає на них.

```
public void Bonys()
{
    if ((sword == "блискавичний" && monster == "ельф") || (sword
        heroSpeed -= 10;
    if ((sword == "надпотужний" && monster == "трент") || (sword
        heroDamage += 30;
    if ((sword == "проклятий" && monster == "полумяний монстр")
    {
        enemySpeed += 10;
        enemyDamage -= 10;
    }
```

Рис. 1.6 Частина коду гри, що відповідає за бонусні параметри зброї проти відповідних типів противників

Наприклад, “ельф”, “трент” і “гоблін” будуть горіти, якщо їх атакувати “вогняним луком”, а якщо атакувати “полум’яного монстра”, “демона” або “водяного монстра” “проклятим мечем”, то швидкість атаки противника і його сила атаки будуть зменшені.

Друга гра, що лягла в основу “Hexagon Keepers”, є гра “Colour Mage Arena”. Дана гра була створена для лабораторної роботи з дисципліни «Розробка мультимедійних та ігрових систем» на 4 курсі.

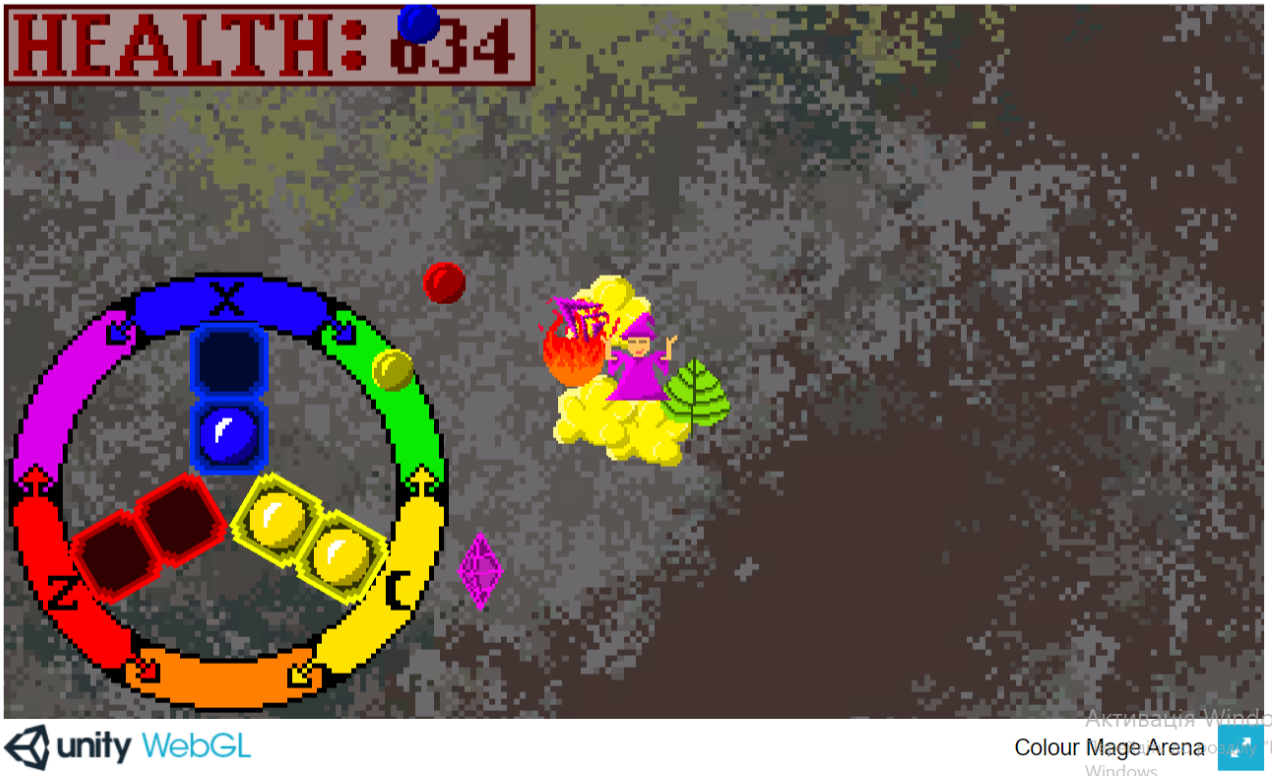

Рис. 1.7 Скріншот з геймплею гри Colour Mage Arena

Дана гра являє собою двохвимірну гру жанру Shoot 'em up, що з англійської перекладається як «перестріляти їх усіх». Це жанр відеоігор, в яких гравець, керуючи персонажем бореться з багатьма противниками за допомогою стрілянини. Зазвичай в таких іграх процес боротьби з противниками зображується у стилізованої манері. Ціллю гри є подолання усіх противників на рівні, шляхом їх знищення.

Основна механіка цієї гри, яку в певній мірі наслідувала гра “Hexagon Keepers” є слідуюча :

Гравець має в ранці по дві ячейки для пулі кожного кольору, він може збирати їх, наступаючи на них. При натисканні клавіш Z, X, C – відповідно створюється заклинання того ж кольору в руці мага, якщо гравець створив два заклинання, то вони через затримку 1 секунда, змішаються в один колір, зафарбують мага в нього, та створять стрілку відповідного кольору. Коли стрілка створена, гравець може цілитися, рухаючи мишею, та нажати на ліву кнопку

миші, щоб вистрілити. Щоб знищити противника, потрібно вистрілити в нього пулею того ж кольору, що й він.

1.5. Висновки до 1 розділу

Отже, в цьому розділі наведені результати дослідження предметної області та аналізу методів розробки ігор та штучного інтелекту. Було досліджено особливості ігрової індустрії а також методи реалізації ігрового штучного інтелекту. Було висвітлено аналоги та прототипи розробленої гри.

Ігри стали не просто забавками, а масштабною індустрією з мільйонами користувачів та розробників.

Hexagon Keepers отримала свій початок з двох інших раніше розроблених прототипів, з яких були взяті ключові механіки та ігрові об’єкти.

РОЗДІЛ 2
РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Архітектура гри

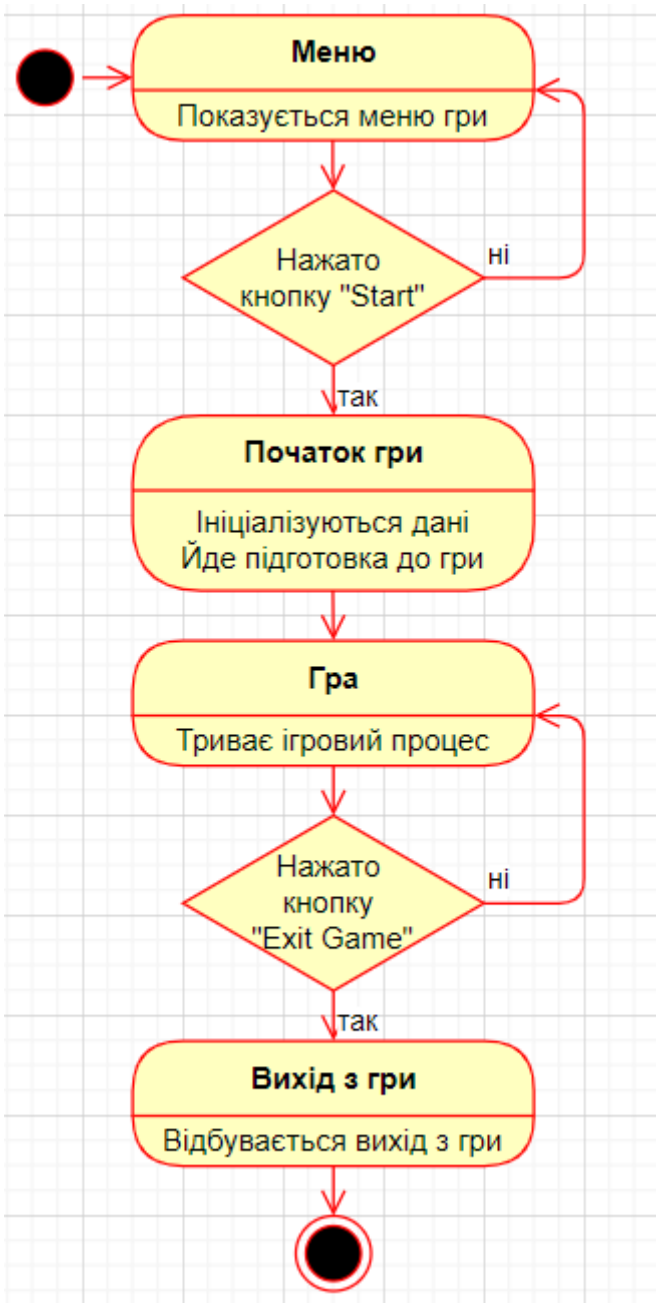

Рис. 2.1 State diagram (Діаграма станів) ігрового застосунку

На діаграмі станів зображено стани гри, починаючи головного меню, закінчуючи виходом з неї.

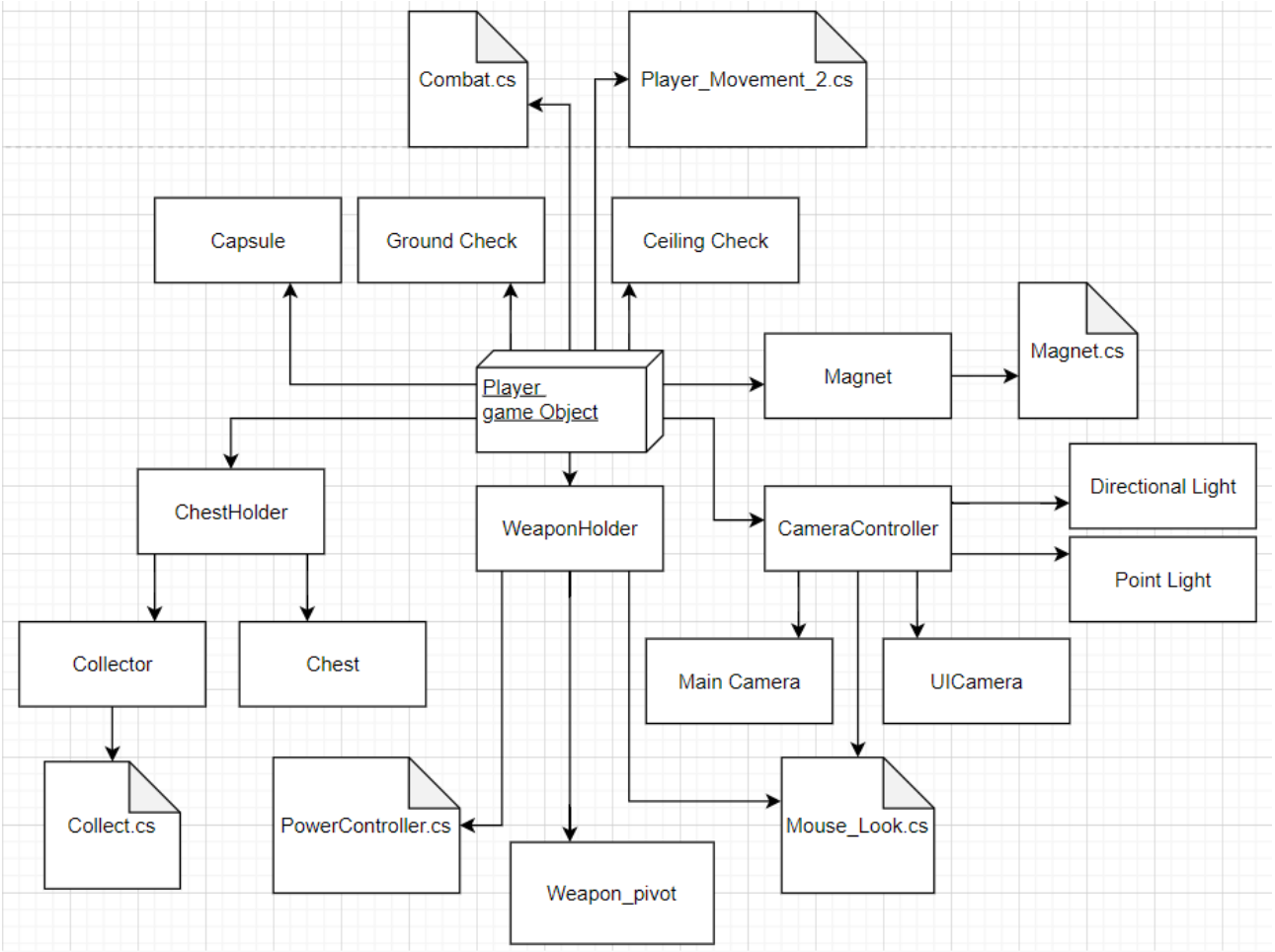

Рис. 2.2 Діаграма компонентів об’єкту Player

Діаграма на рис. 2.2 зображує структуру ігрового об’єкта Player – персонажа гравця. Прямокутниками зображені ігрові об’єкти, що є складовими Player. Значком замітки позначені скрипти відповідних ігрових об’єктів, компонентами яких ці скрипти являються.

ChestHolder – містить у собі Collector та Chest, які у свою чергу відповідають за збирання кубиків, та візуальне відображення цього процесу.

WeaponHolder – відповідає за керування зброєю та надання їй візуального супроводу у вигляді відповідного кольору.

Magnet – відповідає за притягування кубиків для збирання до об’єкту Collector

CameraController – відповідає за керування Main та UI камерами та розміщення джерел світла Point Light та Directional Light.

Рис. 2.3 Діаграма діяльності, що описує механіку зброї

На діаграмі, що на рисунку 2.3 описуються особливості механіки зброї. Виходячи з неї, можна зробити висновок, що у гравця є кілька варіантів дії при збиранні зброї :

- Відразу ж нею атакувати потрібного противника та нанести йому звичайної шкоди, після чого зброя знищиться і прийдеться знову шукати уламок
- Утриматися від використання непокращеної зброї, зібрати ще один уламок, щоб її покращити та нанести підвищену шкоду противнику та ще й зберегти зброю від знищення.

2.2. Проектування геймплею та сюжету

2.2.1. Сцена 1 (Перше знайомство)

Гравець керує персонажем – Блукаючим Мандрівником, що випадково потрапив у ігровий світ, під час своєї подорожі у пошуках магічного каменю стихій. З самого початку він зустрічає Невідомого – ігрового персонажа, що стає духовним наставником і помічником Блукаючого Мандрівника. Невідомий розповідає, що існує 6 країн, у яких колись правили 6 оберігачів шестикутника, але сталася жахлива негода і всі оберігачі втратили свої сили. Після чого, на землях країн оселилися злі створіння. Невідомий дає Мандрівнику чарівну скриню, та дає завдання знайти свій перший фрагмент зброї, щоб використати її на ворога.

Після того, як гравець збирає фрагмент зброї, ця зброя з’являється у його правій руці, та при натисканні лівої клавіші миші, ця зброя виконує удар, або постріл.

При використанні зброї на противника, противник знищується, розпадаючись на багато кубиків. При наближенні до кубиків, що лежать на землі, з’являється магічна скриня, що відкривається, та збирає у себе всі кубики поблизу ігрового персонажа.

2.2.2. Сцена 2 (Пояснення механік гри)

Після цього гравець знову повертається до Невідомого, і той пояснює, що кубики, які зібрала скриня – це ігрові ресурси, що необхідні гравцеві для проходження гри.

Невідомий навчає гравця тому, що у кожного противника є 4 атрибути, 2 з позитивними ефектами для противника та 2 з негативними для нього ж.

Невідомий вказує гравцеві, що для пробудження сили в кожного з 6 оберігачів шестикутника, потрібно зібрати певну кількість кубиків кольору, що

відповідає кольору стихії оберігача. Невідомий також застерігає, що при кожному пробудженні оберігача буде використано рівно половину всіх інший кубиків, кольорів, що не відповідають спеціальному.

2.2.3. Сцена 3 (Пробудження оберігача)

Блукаючий мандрівник підходить до оберігача, для якого він назбирав достатню для пробудження кількість кубиків відповідного кольору. Віддаючи потрібні кубики, гравець втрачає половину усіх інших кубиків. Оберігач пробуджується, повертаючи свою силу, та запалює відповідний шлях до центру великого шестикутника.

2.2.4. Сцена 4 (Пробудження останнього оберігача)

Якщо пробудити сили всіх оберігачів, то всі 6 оберігачів запалять відповідні шляхи до центру шестикутника в середині якого з’явиться камінь стихій, який і шукав головний персонаж гри.

Також, гравцеві стане відомо, що Невідомий насправді – це верховний маг усіх 6 країн. І що саме він створив формулу каменю стихій.

На цьому етапі гра вважається завершеною, але гравець досі може боротися з противниками, накопичуючи ігрові бали за збирання кубиків.

2.3. Розробка графічної складової гри

Основною стилістикою графіки гри було обрано воксельну графіку. Але попередньо для цього були створені зображення у піксельному стилі.

Піксельна графіка, або pixel art це форма цифрового мистецтва, де зображення створюються та редагуються на рівні пікселів, тобто кожен піксель зображення додається художником вручну, або за допомогою спеціальних пензликів, чи інструментів, що містяться у програмному забезпеченні для редагування графіки. При проектуванні вигляду ігрових обєктів було

використано саме таку технологію. Двохвимірний вигляд персонажів та зброї створювався у Aseprite -безкоштовному графічному редакторі для створення 2D спрайтів та анімацій.

Основною рисою pixel art-у являється його унікальний візуальний стиль, де окремі пікселі служать будівельними блоками, що складають зображення. Створюючи графіку в такому стилі, дуже важливо звертати увагу на кожен піксель, адже чим менше розширення зображення в цілому, тим значущішим стає кожен піксель. Двохвимірний вигляд персонажів та зброї для гри було виконано у розмірі 32x32 пікселя, тому, змінивши всього декілька із них, можна отримати зовсім інший результат (рис. 2.4)

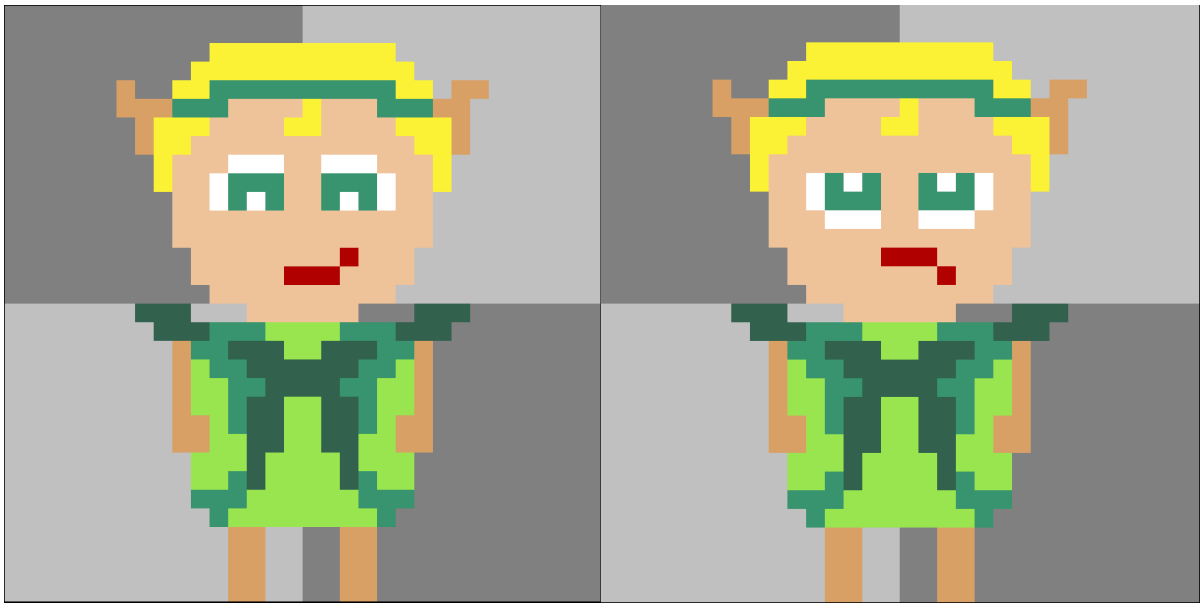

Рис. 2.4 Демонстрація значущості кожного пікселя

Такий підхід до створення графіки потребує високої концентрації та та вміння виділяти основні значущі риси створюваного образу.

Важливим інструментом для виділення основних рис та форм зображення є колір. У піксельній графіці, а особливо при створенні невеликих зображень прийнято використовувати 2-5 кольорів та їх відтінки. Це правило дотримувалося і при створенні персонажів гри.

2.2.1. Створення двохвимірного спрайту

У графічному редакторі Aseprite було створено новий спрайт з розмірами 32x32 пікселів. З прозорим фоном.

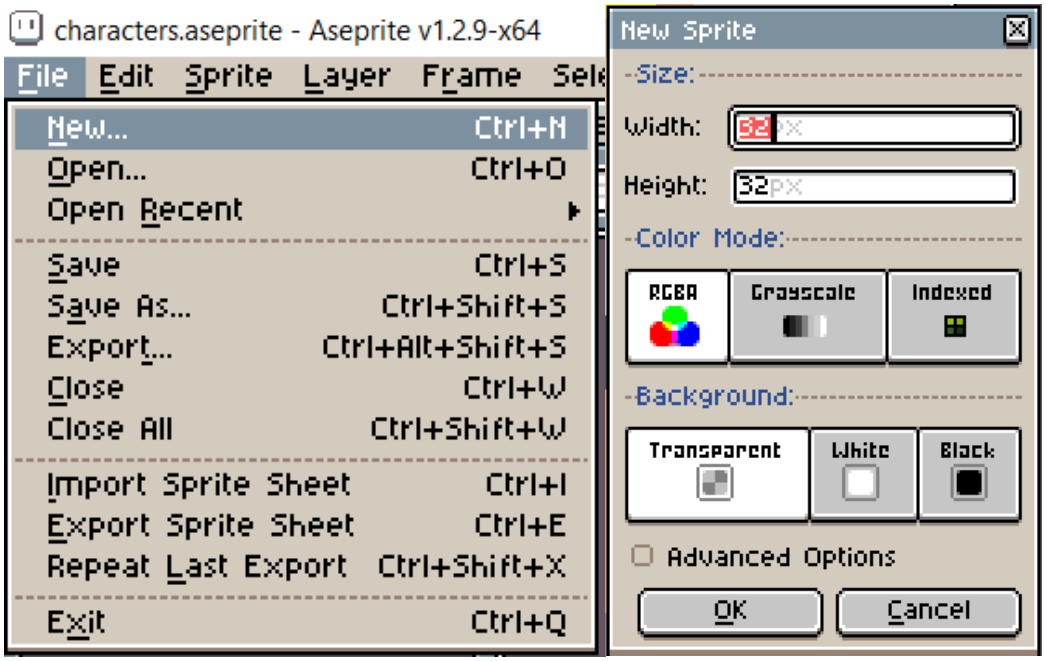

Рис. 2.5 Створення нового спрайту в Aseprite

Було створено палітру з кольорів, що підійдуть для зображення.

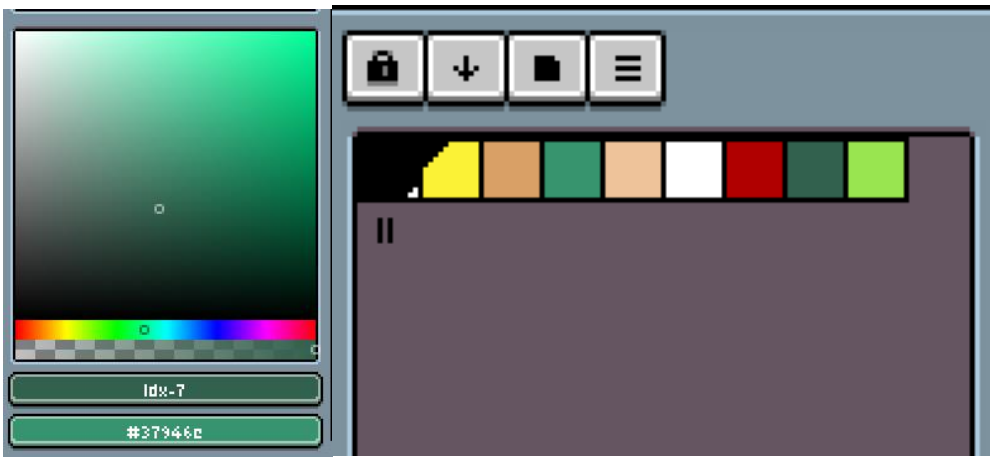

Рис. 2.6 Вибір кольору та палітра

Користуючись інструментами редактора та підібраною палітрою кольорів було створено спрайт ігрового персонажа.

2.2.2. Конвертування спрайту в трьохвимірний об’єкт

В графічному редакторі для воксельної графіки Magica Voxel було створено новий пустий проект.

Збережений спрайт персонажа у форматі картинки було переміщено у вікно програми, з відкритим пустим проектом, таким чином було створено трьохвимірний воксельний об’єкт, кожен воксель якого відповідає пікселю спрайта персонажа. Даний об’єкт 1 воксель у висоту.

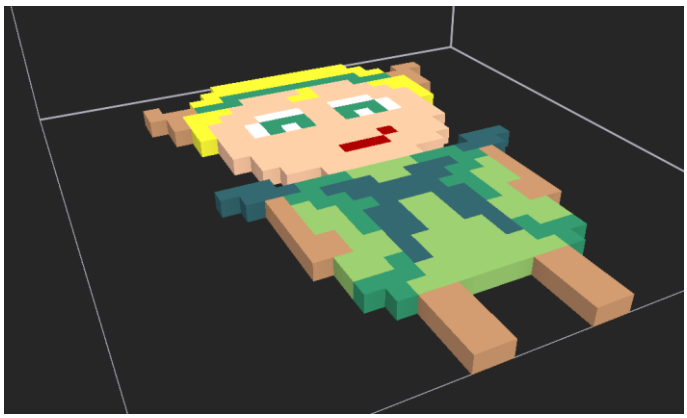

Рис. 2.7 Воксельна модель персонажа з висотою 1 воксель

Користуючись інструментами редактора було створено об’єм та рельєф ігрового персонажа.

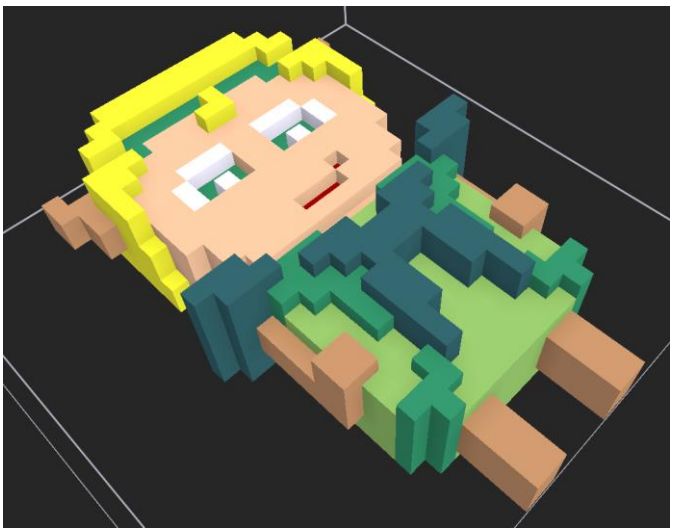

Рис. 2.8 Об’ємна воксельна модель персонажа

Інструментом для експортування воксельного об’єкту, персонажа було експортовано у форматі OBJ — формат файлу опису геометрії, започаткований Wavefront Technologies. Цей формат являється стандартом для трьохвимірних обєктів.

Експортований файл формату obj, разом з файлом формату mtl - файлом налаштування текстур для матеріалу 3D-об'єкта, та файлом формату png – палітрою текстури для матеріалу 3D-об'єкта було переміщено у відповідну папку проекту в Unity.

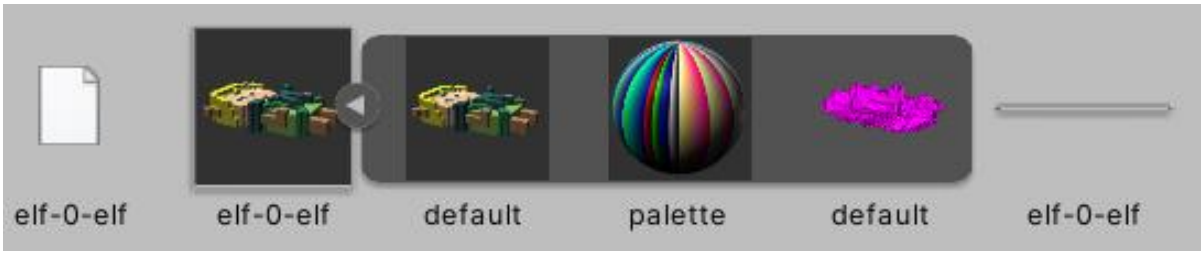

Рис. 2.9 Експортовані файли моделі в Unity

2.2.3. Створення зруйнованої моделі противника

Однією з механік гри було зроблено здатність збирати кубики з яких складається зруйнована модель противників. Для цього було зроблено такі моделі для усіх 9 противників.

Основні етапи створення зруйнованих моделей противників :

1. У воксельному редакторі Magica Voxel було створено новий проект, в який як і при конвертуванні спрайту в трьохвимірний об’єкт було імпортовано спрайт персонажа у форматі картинки

2. Конвертований об’єкт персонажа, що складається з вокселів, і у висоту 1 воксель було експортовано у форматі qb

3. У редактор воксельних об’єктів Goxel було імпортовано об’єкт противника у форматі qb.

4. Goxel - це схожий за функціоналом до Magica Voxel редактор, але з додатковими інструментами. Одним з таких інструментів було експортовано об’єкт противника у фотматі text

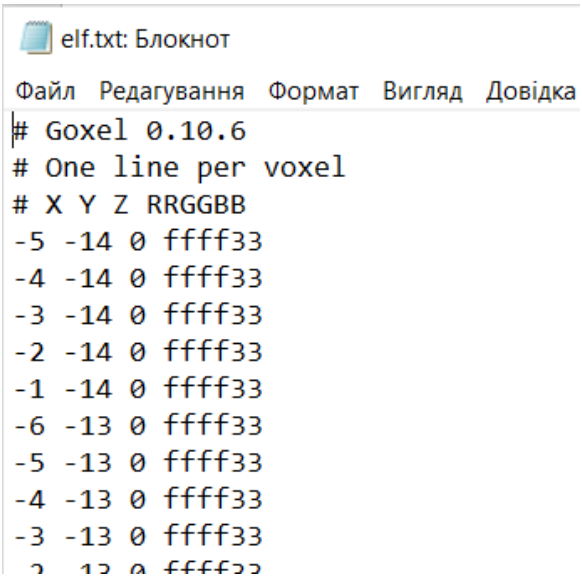

Рис. 2.10 Внутрішня структура об’єкту у текстовому форматі

5. Останнім етапом створення зруйнованої моделі противника було написання та використання скрипта (додаток А), за допомогою якого можна:

- Читати рядки такого формату: <X Y Z color>
- Створювати префаб з багатьох кубиків, визначених їх координатами та кольором.
- Для кожного кольору створювати окремий матеріал у відповідності до заданого шейдера.

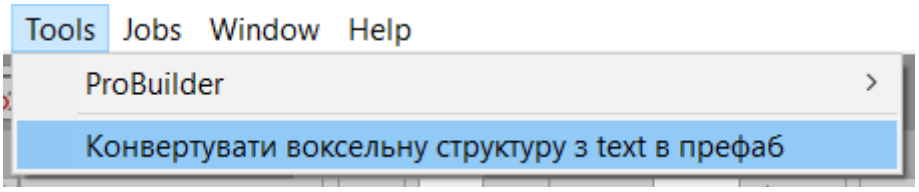

Рис. 2.11 Використання скрипта для конвертації воксельної структури в префаб

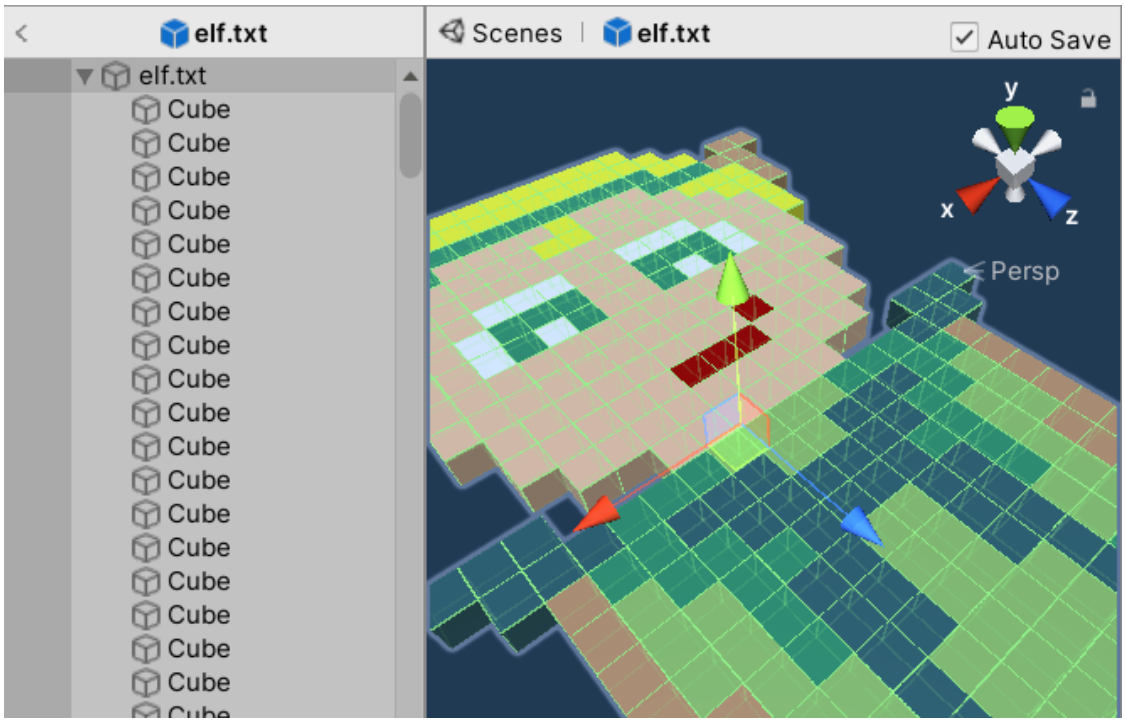

Рис. 2.12 Вигляд префаба та його структура в Unity

Об’єкти-кубики, що складають префаб мають назву Cube. Кожному такому кубику було додано компоненти Box Collider та RigidBody, що відповідають за фізичні властивості ігрового об’єкта та його взаємодію з іншими обєктами. Кубикам був наданий тег Piece(англ. Шматок), та виставлений рівень Pieces(англ. Шматки).

Окремо, було створено скрипт Collectable (додаток Б) та додано до кожного з кубиків. Цей скрипт дозволяє зберігати значення кольору кожного окремого кубика, та відповідає за механіку збирання кубиків гравцем.

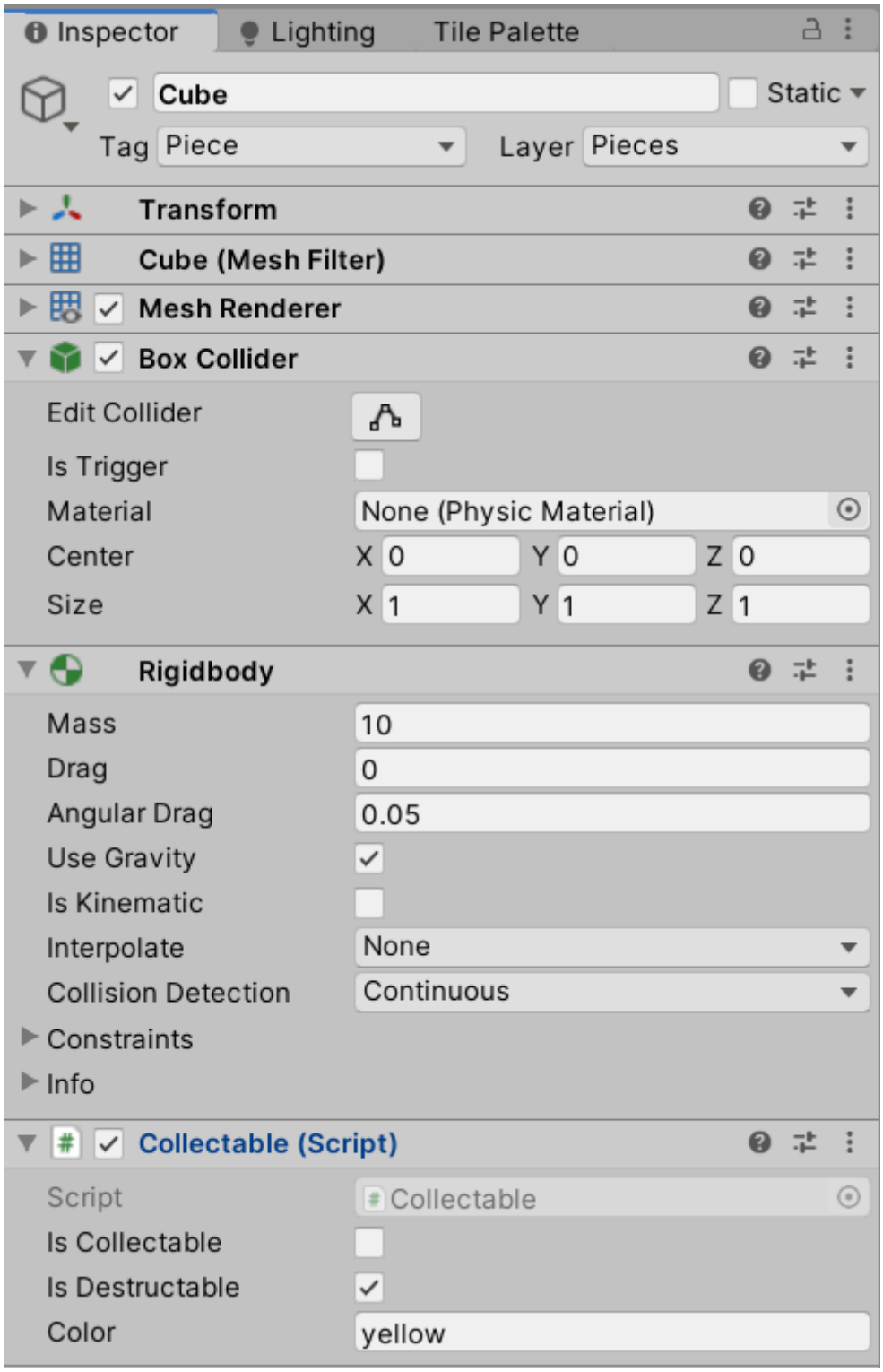

Рис. 2.13 Компоненти об’єкта Cube

2.4. Реалізація ігрових об’єктів

2.4.1. Оберігачі шестикутника

Кожен з 6 оберігачів шестикутника відповідає певній стихії та має колір, що асоціюється з даною стихією. Також, імена оберігачів походять зі слів англійською мовою, що мають логічну відповідність до стихій оберігачів. Дана відповідність відображена в таблиці 2.1 .

Таблиця 2.1

Відповідність імені оберігачів до їх стихій

Колір	Жовтий	Зелений	Фіолетовий	Червоний	Голубий	Білий
Стихія	Блискавка	Земля	Темрява	Вогонь	Лід	Світло
Ім’я англійською	**Zap**	**Chunky**	**Obscure**	**Blaze**	**Gelid**	**Holy**
Ім’я українською	Зап	Чанкі	Обскур	Блейз	Гелід	Холі
Значення імені(переклад)	Сплеск енергії	Кремезний	Тьмяний, темний	Спалах, полум’я	Льодовий	Святий

Оберігачі задумувалися як абсолютно чорні персонажі, але з явними візуальними відмінностями у формі тіла, а також з особливим забарвленням очей та одного спеціального атрибуту їх образу, що також забарвлений в колір відповідної стихії оберігача. Хоча персонажі створені з одного основного кольору, та не мають значної деталізації, було передано їх основні якості та риси характеру, використовуючи різне розміщення рук та ніг, форму тіла, а також форму та розмір очей.

Таблиця 2.2

Особливі атрибути оберігачів

Оберігач	**Zap**	**Chunky**	**Obscure**	**Blaze**	**Gelid**	**Holy**
Атрибут	Пов’язка	Нагрудний щиток	Хвіст	Колчан для стріл	Шолом	Корона
Якості персонажа	Швидкий, веселий	Сильний, серйозний	Страшний, хитрий	Запальний	Спокійний, впевнений	Добрий

Рис. 2.14 3Д Моделі оберігачів в Unity

2.4.2. Зброя

Всього у грі 6 видів зброї. Кожна зброя має свою стихію. Кожен вид зброї належить оберігачу шестикутників відповідної стихії та кольору. Зброю було названо відповідно до перекладу її стихії на латинську мову. Дана відповідність продемонстрована в таблиці 2.3 .

Таблиця 2.3

Відповідність стихії зброї до її назви

Вид зброї	Меч	Молот	Тризуб	Лук	Арбалет	Посох
Стихія	Блискавка	Земля	Темрява	Вогонь	Лід	Світло
Назва англійською	**Fulminis**	**Terra-ton**	**Tenebris**	**Ignis**	**Glacies**	**Lux-is**
Транслітерація українською	Фульмініс	Тератон	Тенебріс	Ігніс	Глейсіз	Лаксіз
Переклад з латинської	Блискавка	Земля(terra)	Темрява	Вогонь	Лід	Світло(lux)

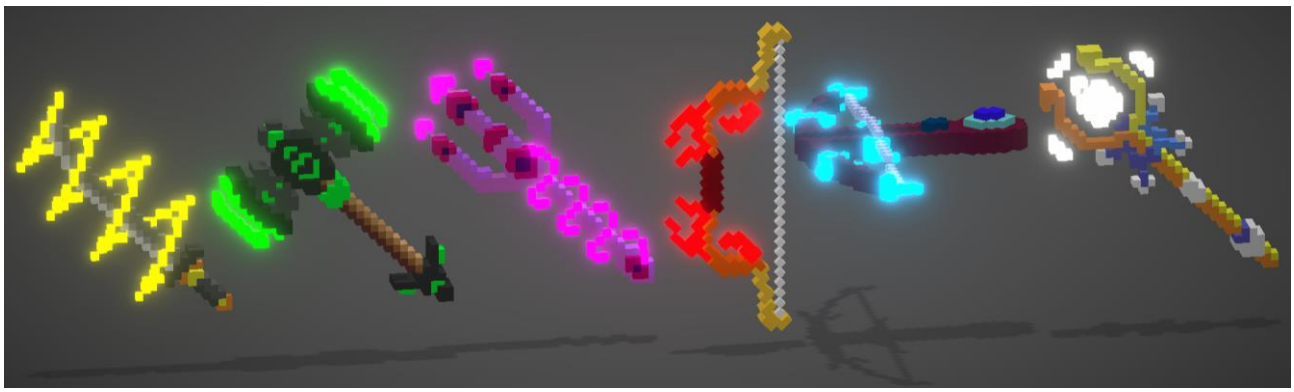

Рис. 2.15 3Д Моделі зброї в Unity

2.4.3. Противники та їх атрибути

В грі було створено 9 видів противників. Усі кольори, що були використані для їх зображення можна згрупувати в 13 кольорів. Саме такої кількості є види кубиків, які випадають при знищенні противників. Колір та кількість кубиків завжди відповідають тій кількості пікселів, що були використані для створення відповідного противника.

Рис. 2.16 3Д Моделі противників в Unity

Кожен противник має 4 атрибути, 2 з яких мають позитивний ефект для противника, та 2 – негативний ефект. Колір атрибутів вказує на ефективність використання проти певного противника зброю відповідної стихії. Кожен позитивний атрибут має свою пару серед негативних атрибутів і вона обов'язково іншої стихії.

Рис. 2.17 Атрибути противників

Атрибути в зеленому прямокутнику на рисунку вказують на те, що вони дають противнику позитивний ефект, а саме – захист від зброї стихія якої

відповідає кольору даних атрибутів. Атрибути ж в червоному прямокутнику вказують на те, що противник має підвищену вразливість до зброї відповідних стихій.

Всього є 12 атрибутів :

Позитивні:

- **Спритність**
- **Невразливість до вогню**
- **Часткова безтілесність**
- **Анти прокляття**
- **Руна захисту від світла**
- **Холоднокровність**

Негативні:

- **Слабка броня**
- **Здатність заморозитися**
- **Потойбічність**
- **Неповоротність**
- **Вразливість до вогню**
- **Вразливість до світлої магії**

На рисунку 2.18, зверху кожного противника зображені його позитивні та негативні атрибути. Розподіл атрибутів відбувався методом інтуїтивного уявлення кожного противника та протиставлення йому різної зброї з особливими стихіями.

Рис. 2.18 Відповідність атрибутів кожному противнику

Наприклад, у противників “Трент”, “Гоблін” та “Зомбі” є атрибут “неповоротність”, що пояснюється тим, що ці три персонажа завжди характеризувалися як повільні та неповороткі створіння.

У противників “Вогняна відьма” , “Водяний монстр” та “Вогняний монстр” є атрибут “невразливість до вогню”, адже двоє з цих персонажів належать до стихії вогню, а “Водяний монстр” – це ніщо інше, як створіння, яке майже на 100% складається з води, тому ніяке полум’я йому не перешкода. А от противнику “Трент”, який являє собою живе дерево, вогонь саме і завдає максимальної шкоди, тому у нього атрибут “вразливість до вогню”

Усі противники, що мають позитивний атрибут “спритність”, мають негативний атрибут “слабка броня”, адже частіше всього їм вона не потрібна, адже вони покладаються на свою швидкість та спритність.

У таблиці 2.4 наведено яким видом зброї можна завдати максимальної шкоди противнику, а від якого виду у противника буде підвищений захист.

Таблиця 2.4

Ефективність типу зброї проти певного противника

Противник\Зброя	меч	молот	тризуб	лук	арбалет	посох
Ельф	слабка броня	спритність		вразливість до вогню		руна захисту від світла
Вогняна відьма	слабка броня	спритність		невразливість до вогню	здатність заморозитися	
Вампір	слабка броня	спритність			холоднокровність	вразливість до світлої магії
Трент		неповоротність	анти прокляття	вразливість до вогню		руна захисту від світла
Зомбі		неповоротність	анти прокляття		холоднокровність	вразливість до світлої магії
Гоблін		неповоротність	анти прокляття	вразливість до вогню		руна захисту від світла
Вогняний монстр	часткова безтілесність		потойбічність	невразливість до вогню	здатність заморозитися	
Демон	часткова безтілесність		потойбічність		холоднокровність	вразливість до світлої магії
Водяний монстр	часткова безтілесність		потойбічність	невразливість до вогню	здатність заморозитися	

Зеленим кольором позначені позитивні для відповідного типу зброї, але негативні для противників атрибути, та відмічено яка зброя має максимальну ефективність проти відповідного противника

Червоним кольором позначені негативні для відповідного типу зброї, але позитивні для противників атрибути, та відмічено яка зброя має мінімальну ефективність проти відповідного противника.

Жовтим кольором відмічено, яка зброя не має ніякого особливого ефекту проти відповідного противника.

Отже, підсумовуючи, можна зробити висновок, що для кожного противника існує 2 ефективних вида зброї, 2 нейтральних вида і 2 види зброї, від яких у противника є захист. В свою чергу, у кожного виду зброї є 3 противника, проти яких цей вид зброї ефективний, 3 противника, проти яких ця зброя зовсім не ефективна, та 3 противника, проти яких цей вид зброї має середню ефективність.

2.5. Реалізація технічної частини гри

2.5.1. Керування персонажем гравця

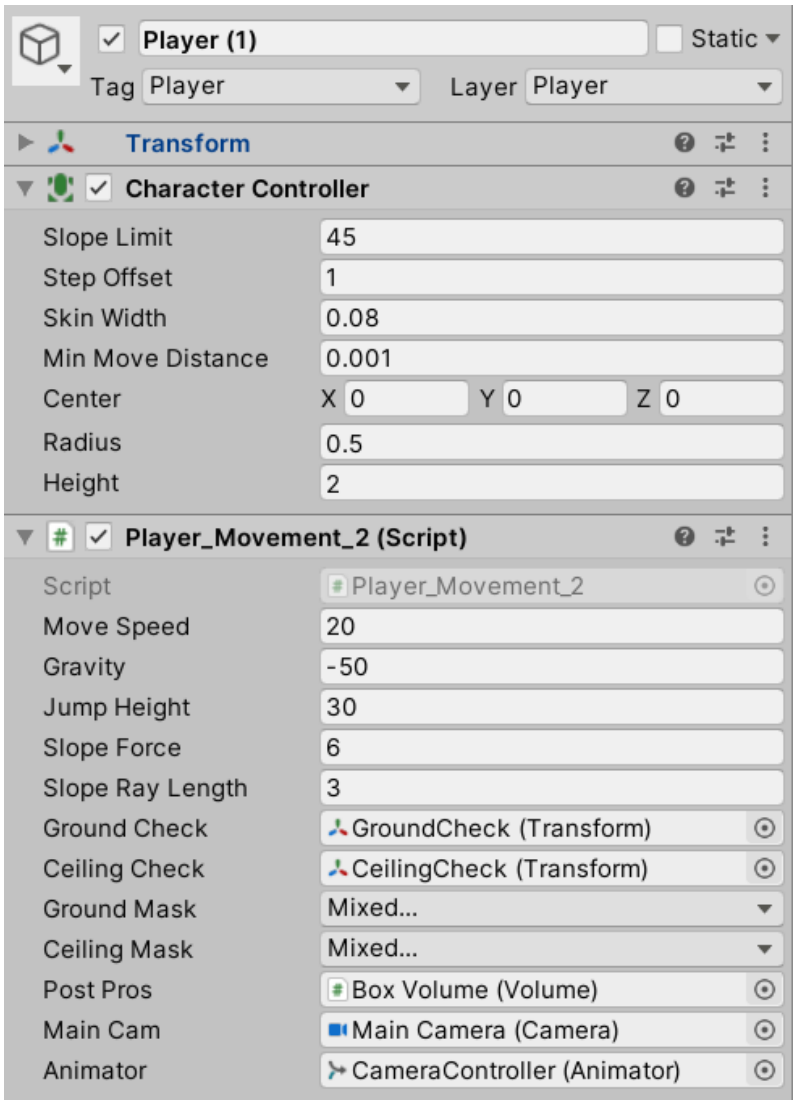

Рис. 2.19 Компоненти об’єкту гравця

Рух ігрового об’єкта – персонажа гравця було реалізовано, використовуючи компонент Character Controller. Цей компонент використовується для управління від третьої або першої особи, де не потрібна фізика, порівняно з управлінням реалізованим за допомогою Rigidbody.

Параметр Slope Limit було виставлено на 45, щоб гравець міг пересуватися по поверхнях, з нахилом до 45 градусів. Step Offset відповідає, за найбільшу висоту об’єкта, наступаючи на який, персонаж може піднятися на нього ж. Також важливими параметрами являються Radius та Height – параметри, що відповідають за розмір капсули самого компоненту Character Controller.

Для руху персонажем було написано скрипт Player_Movement_2 (Додаток В), що напряму взаємодіє з компонентом Character Controller гравця.

Скрипт має публічні поля, яким можна змінювати значення у редакторі. Move Speed визначає швидкість руху граця і за замовчуванням дорівнює 20. У полі Gravity задається сила, яка буде діяти на персонажа, коли він знаходиться у повітрі, імітуючи гравітацію. Параметр Jump Heigh визначає, яка сила буде прикладена до гравця вертикально, при натисканні кнопки стрибка. Slope Force та Slope Ray Length – параметри, для методу OnSlope(), що перевіряє, чи знаходиться гравець на схилі, щоб запобігати нетипічному руху зі схилів.

```
private bool OnSlope() // якщо на схилі
    {
        if (!isGrounded)
        {
            return false;
        }
        RaycastHit hit;
        if (Physics.Raycast(transform.position, Vector3.down, out hit,
controller.height / 2 * slopeRayLength))
        {
            if (hit.normal != Vector3.up)
                return true;
        }
        return false;
     }
```

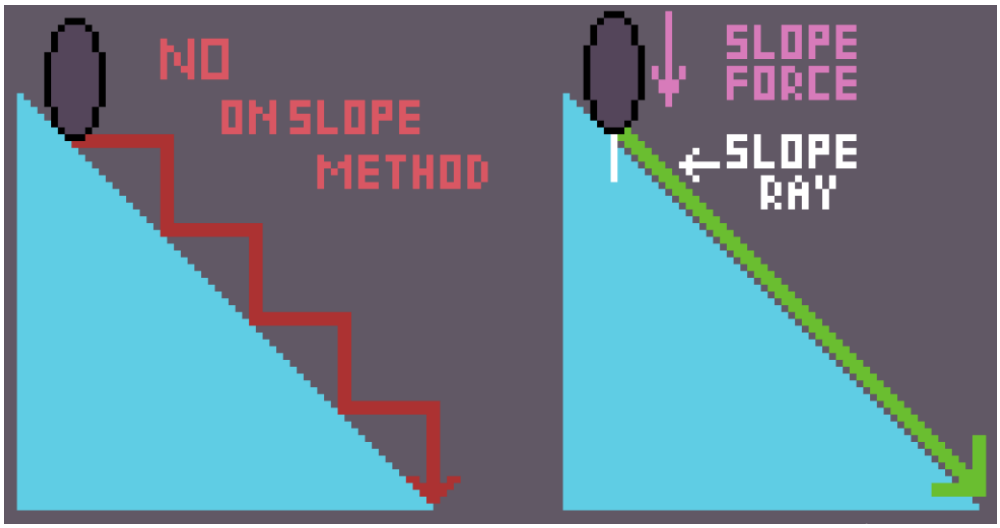

Рис. 2.20 Демонстрація роботи методу OnSlope()

Поля Ground Check та Ceiling Check містять у собі посилання на місця розміщення спеціальних колайдерів, що дозволяють зафіксувати, знаходиться гравець на підлозі, летить у повітрі, чи вдаряється головою об стелю.

Поля Ground Mask та Ceiling Mask являються фільтрами, щоб визначити об’єкти на яких рівнях будуть вважатися підлогою, а на яких стелею для гравця.

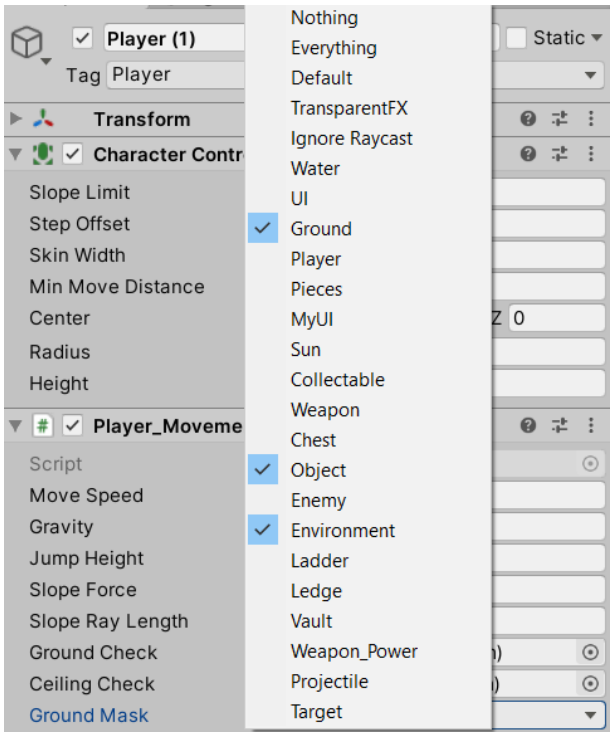

Рис. 2.21 Об’єкти на рівнях Ground, Object та Environment вважаються землею для гравця

Так як була розроблена гра від першого лиця, то ігрова камера була розміщена у верхній частині персонажа гравця, імітуючи його очі. Для руху камерою було створено окремий ігровий об’єкт CameraController, до якого було створено та приєднано скрипт Mouse_Look. Скрипт містить у собі єдиний клас з двома методами. У методі Start() відбувається блокування курсору в центрі екрану, та він робиться невидимим. У методі LateUpdate() покадрово відбувається зчитування положення мишки, та відповідне обертання ігрового об’єкту. Керування чутливістю повороту забезпечується параметром mouseSensetivity.

```
public class Mouse_Look : MonoBehaviour
{
    public float mouseSensetivity = 100f;
    public Transform playerBody;
    float xRotation = 0f;
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
    void LateUpdate()
    {
        float mouseX = Input.GetAxis("Mouse X") * mouseSensetivity ;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensetivity ;
        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -90f, 90f); // задаємо пороги
висоти погляду
        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        playerBody.Rotate(Vector3.up * mouseX);
    }
}
```

Ігровий об’єкт CameraController також відповідає за анімації стрибка, приземлення та руху камери при ходьбі персонажа. Для цього було додано компонент Animator та створено кілька анімацій. У скрипті Player_Movement_2 при виклику метода Jump(), Land() або ж при руху гравця відбувається звернення до відповідних булевих змінних в компоненті Animator об’єкта CameraController.

```
public void Land() //приземлення
{
    jumped = false;
    animator.SetBool("jump", false);
    Debug.Log("Landed");
}
```

Рис. 2.22 Звернення до булевої змінної jump при приземленні

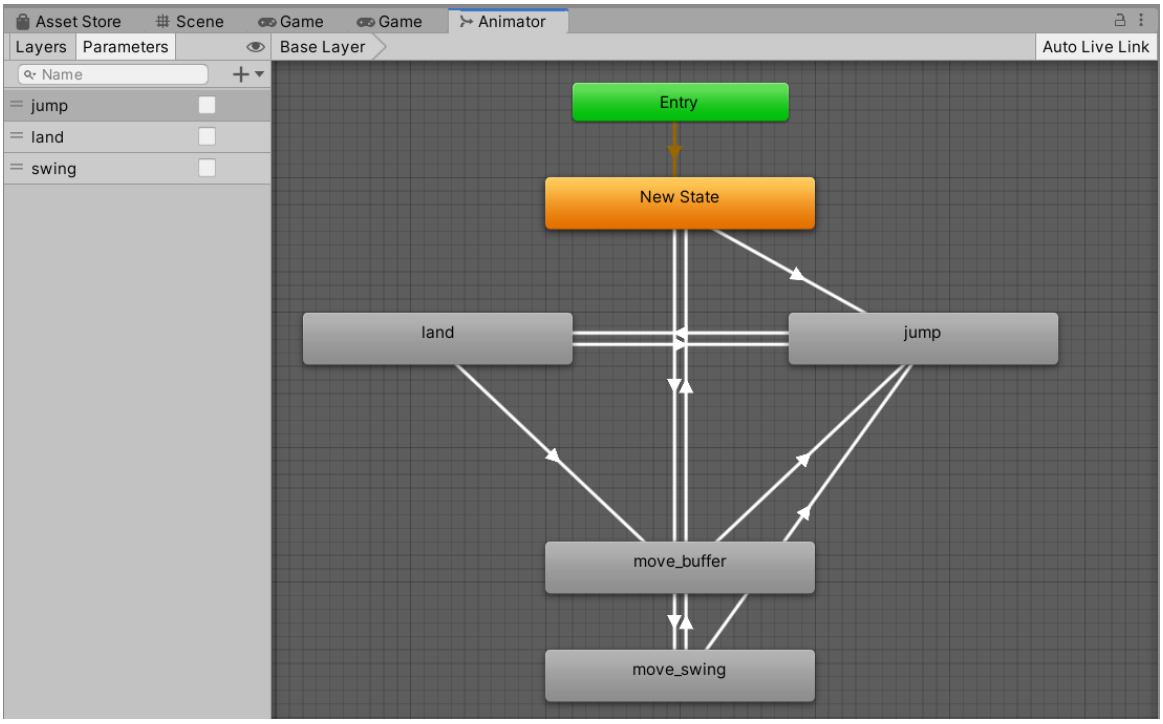

Рис. 2.23 Дерево анімацій об’єкта CameraController

З рисунка видно, що немає переходу від анімації стрибка, до анамації руху, та що в анімацію приземлення можна потрапити тільки після анамації стрибка. Це зроблено для того, щоб не було дивних похитувань камери при натисканні клавіш ходьби у повітрі, та приземлення відбувалося тільки після стрибка, а не після кожного разу коли персонаж перестає йти.

Анімація move_buffer являє собою буфер, що не дає включитися анімації покачування камери при руху move_swing протягом 15 мілісекунд. Це реалізовано для того, щоб не вмикалася анімація руху, якщо персонаж рухався менше 15 мілісекунд.

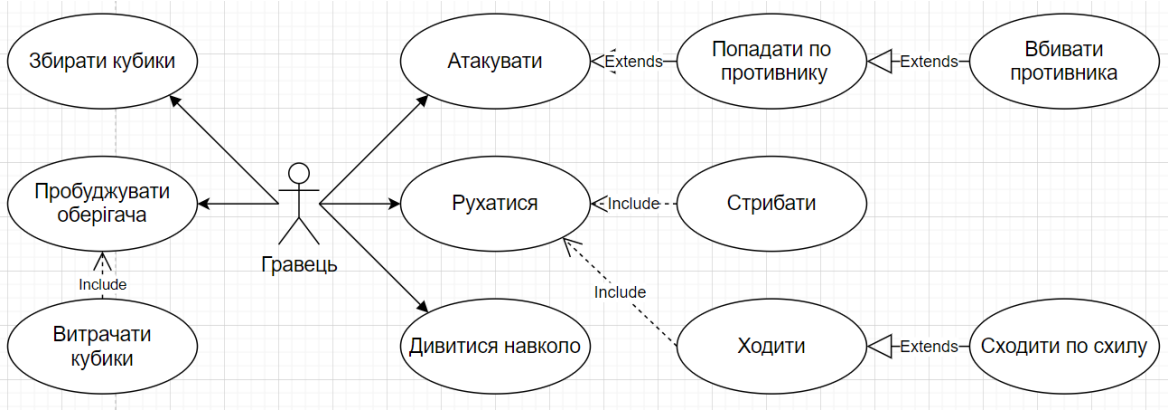

Рис. 2.24 Use case діаграма (діаграма прецедентів) головного персонажа гравця

2.5.2. Реалізація розпадання противників на кубики

Кожному противнику було додано скрипт Destructible, він був створений для реалізації смерті противника, з розпаданням тіла противника на кубики. У параметрах скрипта у кожного противника вказується його зруйнована модель, що вже складається із кубиків. Радіус та сила вибуху, при якому будуть розпадатися кубики. Також змінна N визначає на скільки випадковими будуть напрямки розлітання кубиків.

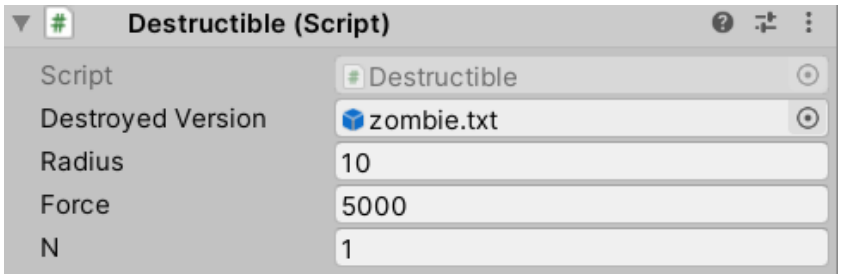

Рис. 2.25 Параметри скрипта Destructible

У скрипті Destructible було написано один метод DestroyEnemy(), при виклику якого, на місці противника відбувається створення екземпляру його моделі з кубиків, потім з точки вибуху, що вказана в даній моделі відбувається вибух, з вказаною силою та радіусом. Вибух впливає на кожен з кубиків, враховуючи випадкове зміщення, що задається змінною boomRand. В самому кінці роботи методу основний об’єкт противника знищується.

```
public void DestroyEnemy()
    {
        GameObject dv = Instantiate(destroyedVersion,
transform.position+new Vector3(0,0.5f,0), transform.rotation);
        dv.transform.Rotate(90, 0, 0);
        Transform bp = dv.GetComponent<FadeAway>().boomPoint;
        Collider[] colliders = Physics.OverlapSphere(transform.position,
radius);
        Vector3 boomPoint = new Vector3(bp.position.x+Random.Range(-
n,n),bp.position.y+0,bp.position.z+Random.Range(-n,n));
        foreach (Collider piece in colliders)
        {
            Vector3 boomRand = new Vector3(Random.Range(-n, n),
Random.Range(-n, n), Random.Range(-n, n));
            Rigidbody rb = piece.GetComponent<Rigidbody>();
            if(rb!=null)
            {
                rb.AddExplosionForce(force,boomPoint+boomRand, radius);
            }
        }
        isDestroyed = true;
        Destroy(gameObject);
    }
```

2.5.3. Реалізація механіки збирання кубиків

Для того, щоб реалізувати можливість гравця збирати уламки противників, було написано скрипт FadeAway, який було приєднано до об’єкта знищеної моделі противника. Даний скрипт відповідає за вмикання прапорця isCollectable в кожному з кубиків, для того, щоб гравець міг їх зібрати. Якщо увімкнути даний прапорець з самого початку на всіх кубиках, то гравець, при знищенні противника відразу ж почне збирати їх, ще до того як вони впадуть на землю, що робить даремним створений раніше ефект розлітання кубиків від вибуху. Саме тому у скрипті FadeAway реалізовано увімкнення прапорця через певний час від створення екземпляру знищеної моделі противника.

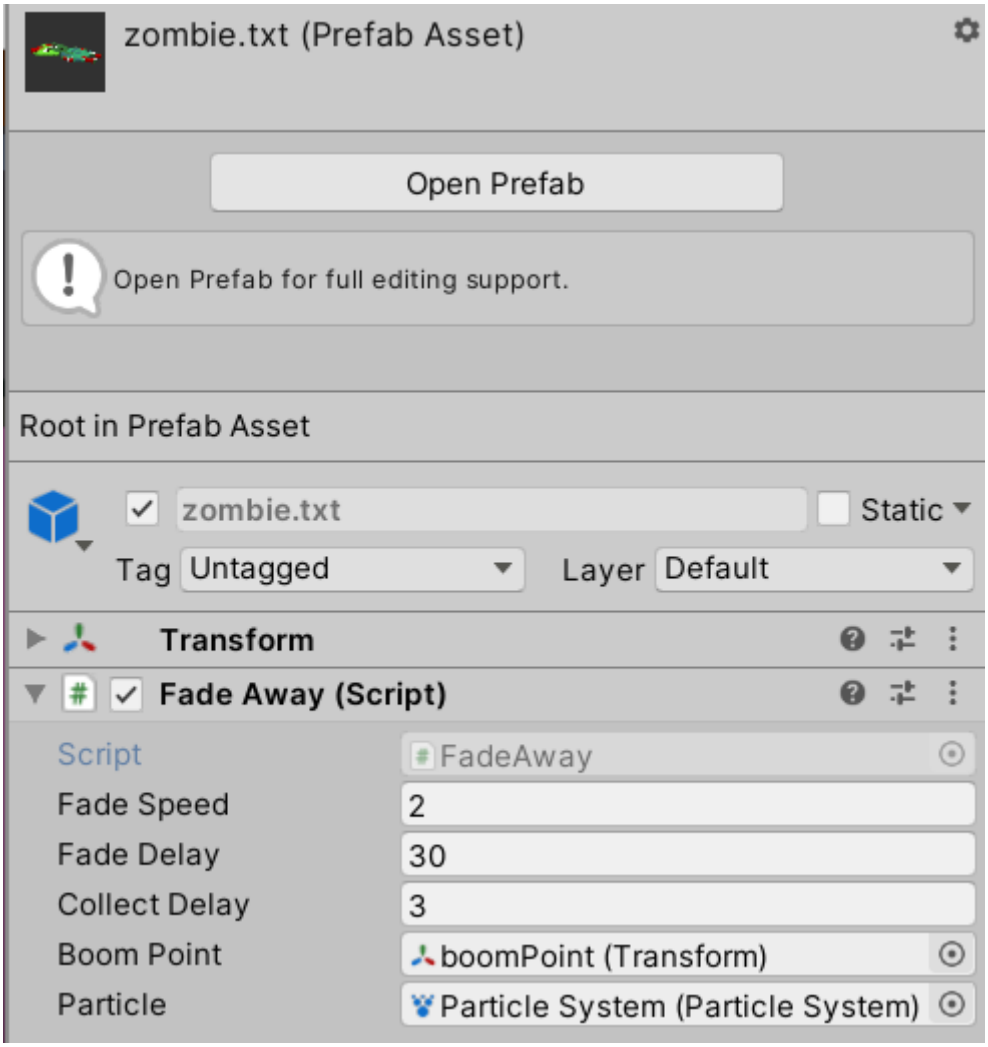

Рис. 2.26 Знищена модель з скриптом FadeAway

Змінна Collect Delay відповідає за час, який має пройти, щоб кубики можна було збирати.

Змінні Fade Speed та Fade Delay відповідають за поступове зникнення кубиків, які гравець не збирав протягом певного часу. Fade Delay визначає після скількох секунд кубики почнуть зникати, а Fade Speed визначає швидкість зникання кубиків. При зникненні кожного з кубиків буде створюватися ефект піднімання бульбашки того ж кольору, що і кубик. Кубики зникають у випадковому порядку, поки не зникнуть усі кубики окремого знищеного противника. Якщо почати підбирати кубики, то ті кубики, що в даний момент підбираються, не можуть зникнути, але при невдалому підбиранні ці кубики зникнуть з більшою швидкістю, так як інших кубиків стало менше.

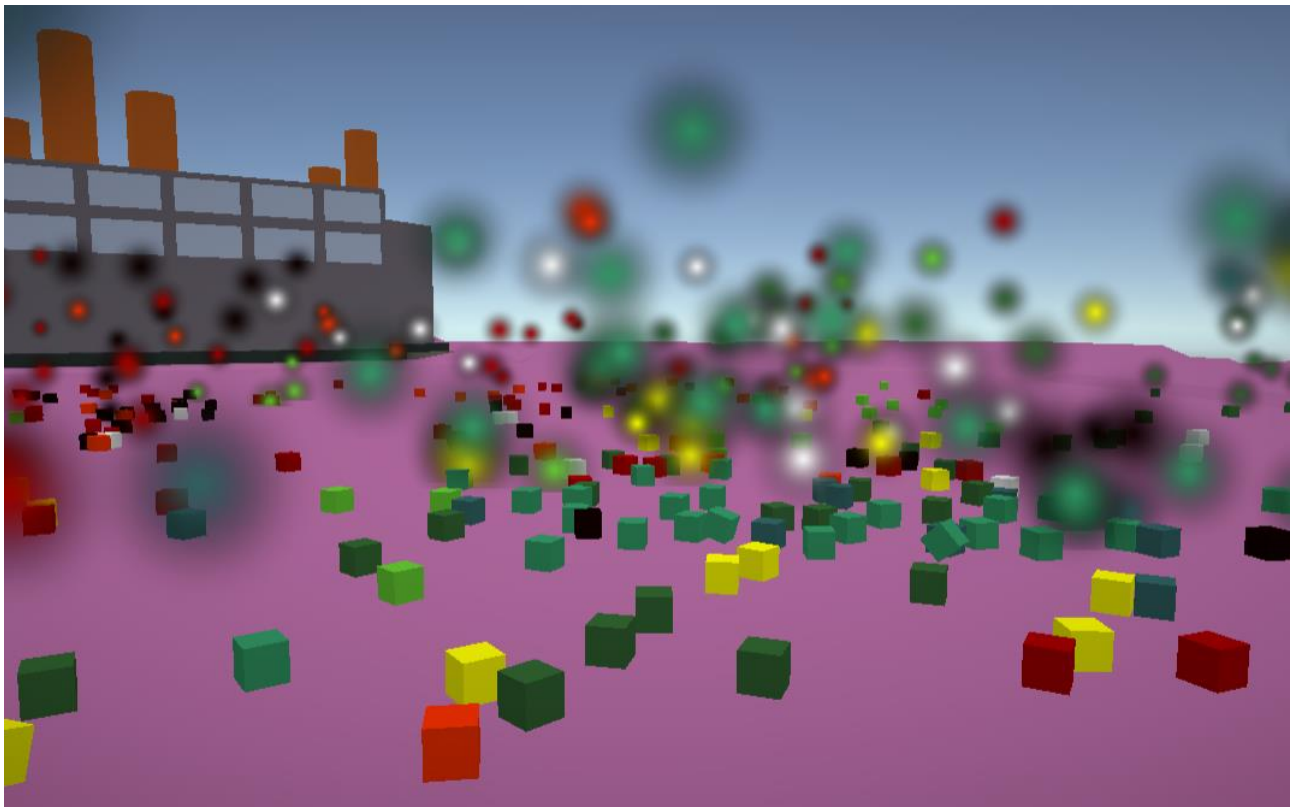

Рис. 2.27 Демонстрація зникнення кубиків з часом

Зі сторони гравця реалізація збирання кубиків полягає в тому, що було створено ігрові об’єкти Magnet та Collector (Додаток Е). Об’єкт Magnet знаходиться в центрі головного персонажа. Він має однойменний скрипт Magnet та колайдер у формі сфери. Даний скрипт відповідає за притягування кубиків, що потрапили у радіус колайдера до об’єкту Collector.

```
private void OnTriggerStay(Collider other)
    {
        if (other.gameObject.tag == "Piece")
        {
            if
(other.gameObject.GetComponent<Collectable>().isCollectable == true)
            {
                other.gameObject.layer = 13;
                other.GetComponent<MeshRenderer>().shadowCastingMode =
UnityEngine.Rendering.ShadowCastingMode.Off;
                other.GetComponent<Rigidbody>().useGravity = false; //
відключаємо гравітацію
                other.GetComponent<BoxCollider>().isTrigger = true; //
ставимо коллайдер як тріггер
```

```
                other.GetComponent<Collectable>().isDestructable = false;
// забороняємо знищення
                other.GetComponent<Rigidbody>().velocity = Vector3.zero;
// обнуляємо прискорення
                other.transform.position =
Vector3.MoveTowards(other.transform.position,
                collector.transform.position , collectspeed *
Time.deltaTime); // притягуємо до магніта
            }
        }
    }
```

В методі OnTriggerStay() скрипта Magnet відбувається перевірка того, чи об’єкти які взаємодіють в даний момент з колайделом мають тег “Piece”, якщо так, то це об’єкт – кубик. Після цього проводиться перевірка прапорця isCollectable кожного кубика. Для кубиків, які вже можна збирати відключається здатність кидати тінь, відключається гравітація, колайдер ставиться у режим isTrigger, забороняється знищення прапорцем isDestructable, обнуляється минула швидкість руху кубика та відбувається його покадрове переміщення у напрямку до об’єкту Collector зі швидкістю визначеню в змінній collectspeed.

2.6. Висновки до 2 розділу

Отже, у другому розділі були описані етапи розробки основної частини програмного продукту, а також висвітлені реалізовані алгоритми.

Було спроектовано 4 сцени, які описують як сюжет гри, так і її ігровий процес. Дані сцени будуть показуватися гравцеві по мірі його проходження гри.

Всю графічну складову, крім об’єктів оточення було створено власноруч та детально описано процес створення.

Було розроблено функціонал ігрових обєктів, визначено способи їх взаємодії, що відповідають механікам гри.

Технічну частину було реалізовано за допомогою скриптів для Unity мовою C#.

РОЗДІЛ 3
РОЗРОБКА ТА ВПРОВАДЖЕННЯ ШТУЧНОГО ІНТЕЛЕКТУ

3.1. Розробка системи для машинного навчання штучного інтелекту

3.1.1. Налаштування python та установка бібліотек

Першим кроком була установка Python на комп’ютер. Після установки потрібно зайти в командний рядок та написати там команду “py”, для того, щоб пересвідчитися в правильній установці Python

Наступним кроком, за допомогою команди “cd” було виконано перехід у папку Unity проекту, в якому буде проводитися машинне навчання штучного інтелекту

Після цього, в папці проекту було створено віртуальне середовище з назвою virtualEnv командою “py -m venv virtualEnv”. Це було зроблено, щоб інкапсулювати всі зміни та дані в межах одного проекту.

Після того, як було створено середовище, його було запущено, виконавши команду, що запускає скрипт activate.

```
D:\programs\Unity\Projects\3D Mage Diploma>virtualEnv\Scripts\activate

(virtualEnv) D:\programs\Unity\Projects\3D Mage Diploma>
```

Рис. 3.1 Команда, що запускає скрипт activate.

У віртуальному середовищі було установлено та оновлено інсталятор пакетів Python, або pip. Для цього була використана команда “py -m pip install –upgrade pip”

Було установлено пакет PyTorch – бібліотеку для машинного навчання штучного інтелекту. Дана бібліотека заснована на бібліотеці Torch. Вона використовується для реалізації комп'ютерного зору та обробки природних мов.

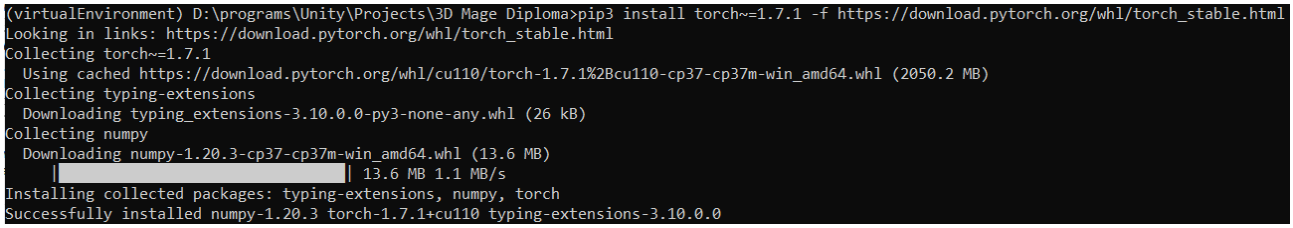

Рис. 3.2 Результат встановлення бібліотеки PyTorch

Останнім етапом установки необхідних бібліотек для створення системи для машинного навчання ШІ являлася установка пакету ml-agents.

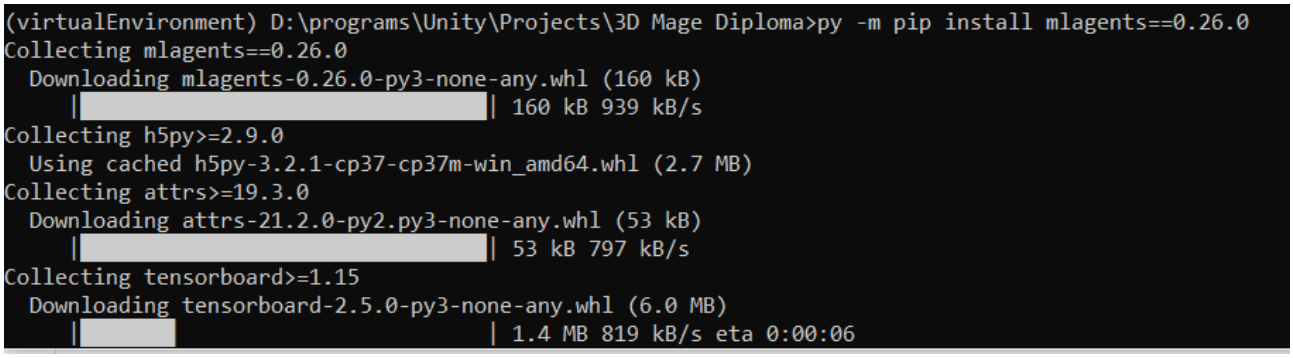

Рис. 3.3 Процес установки пакету для машинного навчання ШІ

Для того, щоб пересвідчитися, що пакет був успішно встановлений було виконано команду “mlagents-learn –help”, при виконанні якої було виведено допоміжну інформацію про команди, що стосуються машинного навчання та їх параметри.

3.1.2. Налаштування агента в unity проекті

Агентом називають об’єкт, який являється носієм штучного інтелекту, що буде навчатися. В ролі агента було взято ігровий об’єкт одного з противників, якому було додано компонент Character Controller, для реалізації руху агента.

Для реалізації взаємодії агента з бібліотеками машинного навчання, йому були додані 2 основних та 2 допоміжних компонента, що безпосередньо беруть участь у навчанні ШІ.

Одним з основних компонентів є створений скрипт MLAgent (Додаток Г).

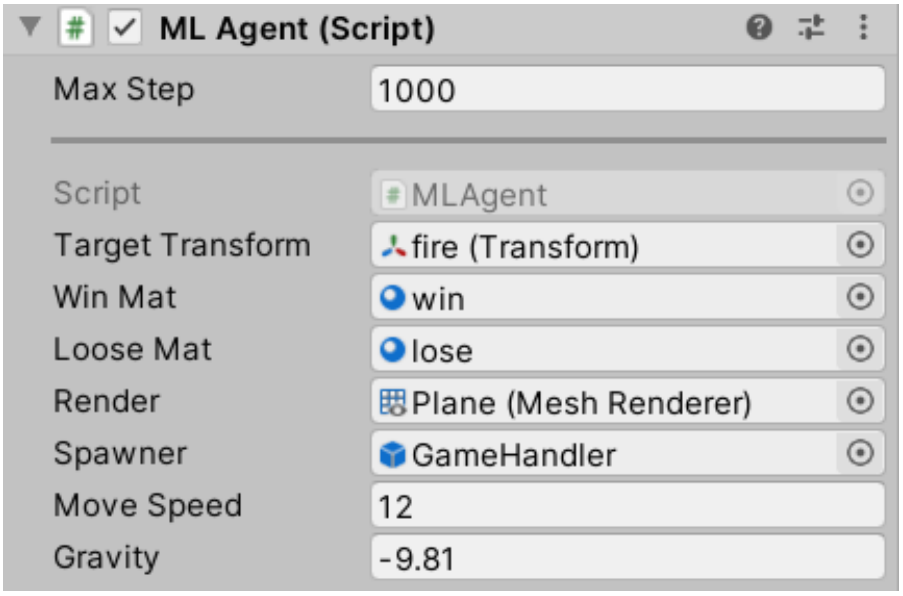

Рис. 3.4 Параметри компонента MLAgent

Даний скрипт відповідає за збирання агентом(Agent) спостережень(Observations) у середовищі(Environment), виконування дій(Actions), та отримання нагород(Rewards) за свої дії. Це відповідає методу reinforcement learning, що на рисунку 3.5

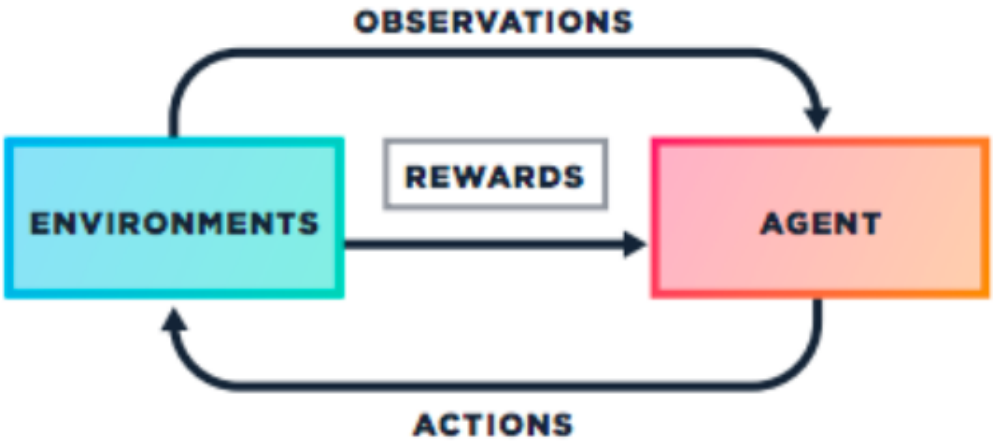

Рис. 3.5 Метод машинного навчання reinforcement learning

У скрипті MLAgent міститься 7 методів :

- Метод Initialize() відповідає за ініціалізацію певних змінних.

- Метод OnEpisodeBegin() викликається по закінченню кожного епізоду, та задає налаштування наступного.
- Метод CollectObservations(VectorSensor sensor) відповідає за налаштування того, звідки агент може брати свої спостереження.
- Метод OnActionReceived(float[] vectorAction) відповідає за налаштування того, які дії агент може виконувати.
- Метод Heuristic(float[] actionsOut) відповідає за контроль над агентом, окремо від навчання. Він дозволяє протестувати правильність роботи інших методів, без запуску процесу навчання.
- Методи OnTriggerEnter(Collider other) та OnTriggerStay(Collider other) необхідні для відслідковування взаємодії колайдерів агента з іншими ігровими об’єктами.

Наступним не менш важливим компонентом являється Behavior Parameters. У ньому було задано назву поведінки агента. Вказано значення Vector Observation Space Size, що відповідає усім можливим змінним, з яких агент може брати спостереження. У даному випадку це значення 10, оскільки у методі CollectObservations(VectorSensor sensor) скрипта MLAgents було додано відслідковування 10 змінних, а саме :

- x, y та z позиції самого агента
- x, y та z позиції цілі
- значення поточної швидкості руху агента у напрямках x, y та z
- відстань від агента до цілі

У параметрі Vector Action було вказано скільки дій може виконувати агент. А саме 2 гілки по 3 значення, тому що в методі OnActionReceived(float[] vectorAction) було вказано, що агент має змогу рухатися тільки по осі x та z (2 гілки) , та він може вирішувати, рухатися по кожній осі в сторону збільшення значення координат, зменшення значення, або ж просто стояти на місці (3 значення)

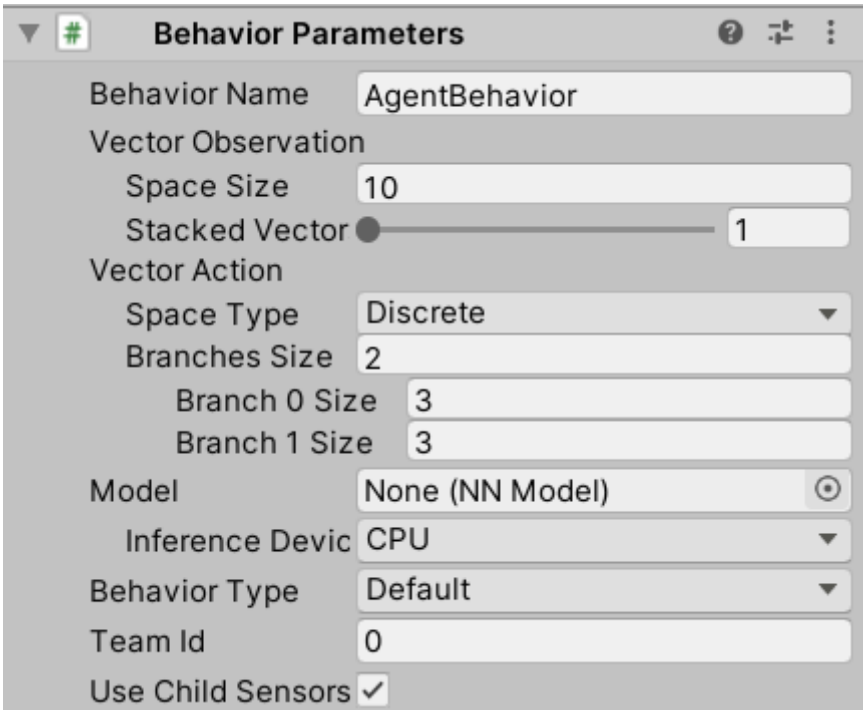

Рис. 3.6 Параметри компонента Behavior Parameters

Двома додатковими параметрами було додано Decision Requester та Ray Perception Sensor 3D, перший з яких забезпечує запити на виконання рішень агентом, а другий являє собою систему з променів з певною довжиною, які можуть повідомляти агенту, про його приближення до певних ігрових об’єктів. Ray Perception Sensor 3D забезпечує схожу з методом CollectObservations(VectorSensor sensor) роль, дозволяючи агенту робити спостереження у середовищі.

3.1.3. Створення середовища для навчання

Для того, щоб мати змогу навчати агента, було зроблено спеціальне середовище, слідуючи таким необхідним особливостям :

- Середовище має бути досить простим, щоб навчання проходило з більшою швидкістю. Для цього середовище було зроблено з використанням лише 19 шестикутників, виставлених у формі великого шестикутника.

- Ключові моменти у середовищі мають містити елементи випадковості. Це забезпечує вміння ШІ адаптуватися до різних умов. Для цього було створено додатковий скрипт GameHandler2, метод Restart() якого викликається у методі OnEpisodeBegin() скрипта MLAgents. При його виклику, відбувається випадкове розміщення елементів простору, таких як перешкоди, ціль агента та сам агент.
- Середовище має бути розроблене так, щоб унеможливити небажану та непередбачувану поведінку ШІ. Для цього був обмежений рух агента, за допомогою розміщення додаткових перешкод по периметру середовища.

Готове середовище складається з обмеженого поля з шестикутників, агента та його цілі, до якої ШІ буде вчитися шукати шлях.

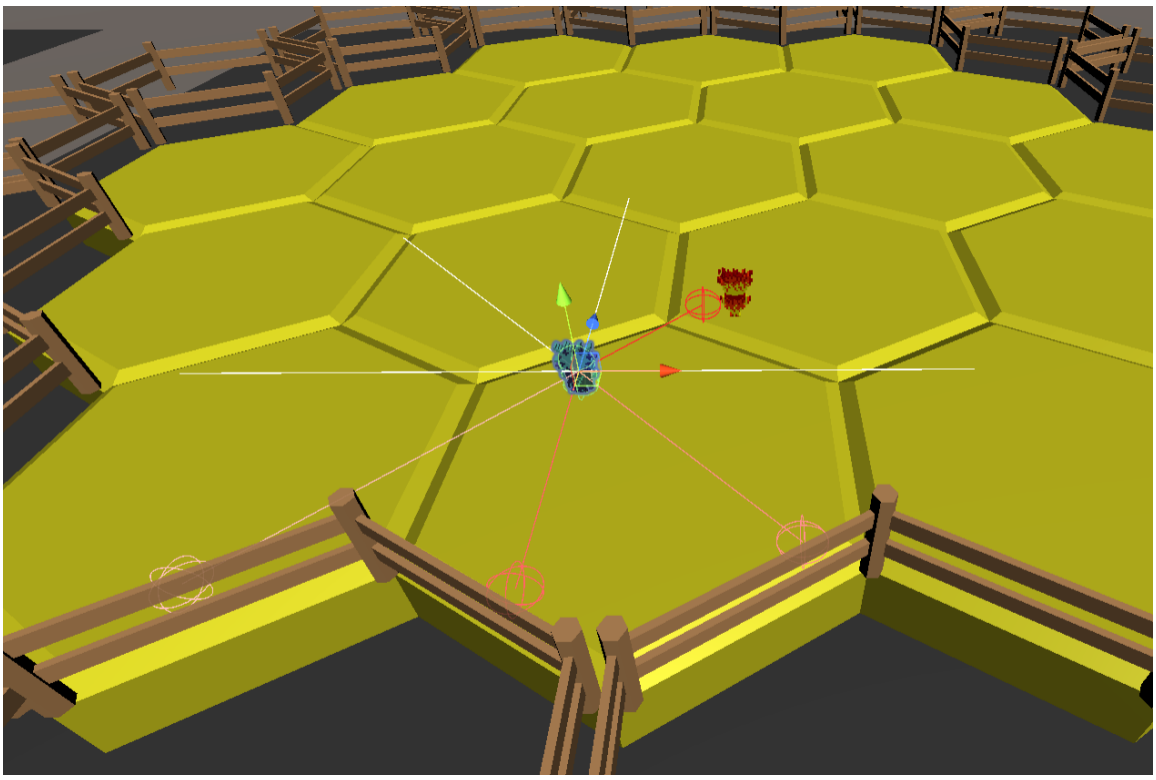

Рис. 3.7 Середовище для навчання ШІ

3.2. Аналіз параметрів та навчання штучного інтелекту

3.2.1. Аналіз параметрів файлу конфігурації

Для того, щоб машинне навчання було ефективним та взагалі мало якісь результати, було досліджено параметри для файлу конфігурації.

Таблиця 3.1

Параметри конфігурації

Параметр	Опис	Діапазон значень	Значення за замовчуванням	Обране для навчання значення
trainer_type	Тип навчання	ppo/sac/poca	ppo	ppo
summary_freq	Кількість досвіду, який потрібно зібрати перед створенням та відображенням статистики навчання.	_	50000	10000
time_horizon	Скільки кроків досвіду, потрібно зібрати кожному агенту, перш ніж додавати його в буфер досвіду.	32-2048	64	128
max_steps	Загальна кількість досвіду, який буде зібрано перед закінченням навчального процесу.	5e5 - 1e7	500000	500000
keep_checkpoints	Максимальна кількість контрольних пунктів, які слід зберігати.	_	5	5
checkpoint_interval	Кількість досвіду, зібраного між кожним контрольним пунктом	_	500000	500000
init_path	Шлях до моделі, з якої відбуватиметься навчання	_	None	None
hyperparameters -> learning_rate	Швидкість навчання для градієнтного спуску. Відповідає силі кожного кроку оновлення градієнтного спуску.	1e-5 - 1e-3	3e-4	2.0e-4

hyperparameters -> batch_size	Кількість досвіду на кожній ітерації градієнтного спуску.	32-5120	256	512
hyperparameters -> buffer_size	Кількість досвіду, який слід зібрати перед оновленням моделі.	2048 - 409600	10240	4096
network_settings -> hidden_units	Кількість одиниць у прихованих шарах нейронної мережі.	32 - 512	128	128
network_settings -> num_layers	Кількість прихованих шарів у нейронній мережі	1-3	2	2
hyperparameters -> beta	Сила регуляризації ентропії, що робить політику "більш випадковою". Збільшення цього дозволить агентам вживати більше випадкових дій, тим самим краще досліджувати середовище	1e-4 - 1e-2	5.0e-3	5.0e-4
hyperparameters -> epsilon	Впливає на те, як швидко політика може розвиватися під час навчання. Відповідає допустимому порогу розбіжності між старою та новою політикою під час оновлення градієнтного спуску. Якщо встановити це значення невеликим, це призведе до більш стабільних оновлень, але також уповільнить навчальний процес.	0.1 - 0.3	0.2	0.2
hyperparameters -> lambd	Це можна розглядати як те, наскільки агент покладається на свою поточну оцінку при обчисленні оновленої оцінки.	0.9 - 0.95	0.95	0.95
hyperparameters -> num_epoch	Кількість проходів через буфер досвіду при оптимізації градієнтного спуску. Зменшення цього забезпечить стабільніші оновлення, за рахунок повільнішого навчання.	3 - 10	3	3

Перед самим процесом навчання було створено файл конфігурації AgentConfig.yaml, який містить налаштування для машинного навчання ШІ. Файл конфігурації було створено на основі проаналізованих параметрів.

AgentConfig.yaml

D: > programs > Unity > Projects > 3D Mage Diploma > config > AgentConfig.yaml

```yaml
behaviors:
  AgentBehavior:
    trainer_type: ppo
    hyperparameters:
      batch_size: 512
      buffer_size:
      learning_rate: 2.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 500000
    time_horizon: 128
    summary_freq: 10000
```

Рис. 3.8 Файл конфігурації AgentConfig.yaml

3.2.2. Запуск та налагодження процесу навчання

Задля ефективності навчання було створено ще 5 копій розробленого середовища, ізольованих одна від одної. Для того, щоб розпочати навчання, у віртуальному середовищі в командному рядку було введено та виконано команду “mlagents-learn config/AgentConfig.yaml --run-id=Test_1”. Дана команда означає, що було розпочато навчання з назвою Test_1 та використовуючи налаштування

з файлу конфігурації AgentConfig.yaml. Після цього у Unity було натиснуто кнопку початку симуляції та навчання ШІ розпочалося.

```
[INFO] AgentBehavior. Step: 10000. Time Elapsed: 48.632 s. Mean Reward: 1.386. Std of Reward: 1.659. Training.
[INFO] AgentBehavior. Step: 20000. Time Elapsed: 74.025 s. Mean Reward: 1.044. Std of Reward: 1.492. Training.
[INFO] AgentBehavior. Step: 30000. Time Elapsed: 101.795 s. Mean Reward: 0.881. Std of Reward: 1.331. Training.
[INFO] AgentBehavior. Step: 40000. Time Elapsed: 129.157 s. Mean Reward: 0.832. Std of Reward: 1.292. Training.
[INFO] AgentBehavior. Step: 50000. Time Elapsed: 158.033 s. Mean Reward: 1.170. Std of Reward: 1.934. Training.
[INFO] AgentBehavior. Step: 60000. Time Elapsed: 185.264 s. Mean Reward: 1.228. Std of Reward: 1.952. Training.
[INFO] AgentBehavior. Step: 70000. Time Elapsed: 212.296 s. Mean Reward: 1.800. Std of Reward: 2.757. Training.
[INFO] AgentBehavior. Step: 80000. Time Elapsed: 240.866 s. Mean Reward: 1.870. Std of Reward: 2.847. Training.
[INFO] AgentBehavior. Step: 90000. Time Elapsed: 267.554 s. Mean Reward: 2.299. Std of Reward: 2.934. Training.
[INFO] AgentBehavior. Step: 100000. Time Elapsed: 296.142 s. Mean Reward: 3.077. Std of Reward: 3.497. Training.
[INFO] AgentBehavior. Step: 110000. Time Elapsed: 324.298 s. Mean Reward: 3.083. Std of Reward: 3.577. Training.
[INFO] AgentBehavior. Step: 120000. Time Elapsed: 362.608 s. Mean Reward: 3.994. Std of Reward: 3.389. Training.
[INFO] AgentBehavior. Step: 130000. Time Elapsed: 397.902 s. Mean Reward: 4.158. Std of Reward: 3.527. Training.
[INFO] AgentBehavior. Step: 140000. Time Elapsed: 433.024 s. Mean Reward: 4.159. Std of Reward: 3.972. Training.
[INFO] AgentBehavior. Step: 150000. Time Elapsed: 469.543 s. Mean Reward: 4.261. Std of Reward: 3.869. Training.
[INFO] AgentBehavior. Step: 160000. Time Elapsed: 503.927 s. Mean Reward: 4.566. Std of Reward: 4.078. Training.
[INFO] AgentBehavior. Step: 170000. Time Elapsed: 540.366 s. Mean Reward: 4.164. Std of Reward: 4.113. Training.
[INFO] AgentBehavior. Step: 180000. Time Elapsed: 574.008 s. Mean Reward: 4.920. Std of Reward: 3.705. Training.
[INFO] AgentBehavior. Step: 190000. Time Elapsed: 610.153 s. Mean Reward: 4.529. Std of Reward: 4.153. Training.
[INFO] AgentBehavior. Step: 200000. Time Elapsed: 643.794 s. Mean Reward: 5.509. Std of Reward: 4.116. Training.
[INFO] AgentBehavior. Step: 210000. Time Elapsed: 677.652 s. Mean Reward: 4.858. Std of Reward: 4.520. Training.
[INFO] AgentBehavior. Step: 220000. Time Elapsed: 712.542 s. Mean Reward: 4.580. Std of Reward: 4.363. Training.
[INFO] AgentBehavior. Step: 230000. Time Elapsed: 746.704 s. Mean Reward: 5.654. Std of Reward: 3.843. Training.
[INFO] AgentBehavior. Step: 240000. Time Elapsed: 782.850 s. Mean Reward: 5.608. Std of Reward: 4.702. Training.
[INFO] AgentBehavior. Step: 250000. Time Elapsed: 817.441 s. Mean Reward: 5.924. Std of Reward: 4.376. Training.
[INFO] AgentBehavior. Step: 260000. Time Elapsed: 851.241 s. Mean Reward: 4.795. Std of Reward: 4.273. Training.
[INFO] AgentBehavior. Step: 270000. Time Elapsed: 886.342 s. Mean Reward: 6.363. Std of Reward: 4.426. Training.
```

Рис. 3.9 Інформація, що демонструється в консолі під час навчання

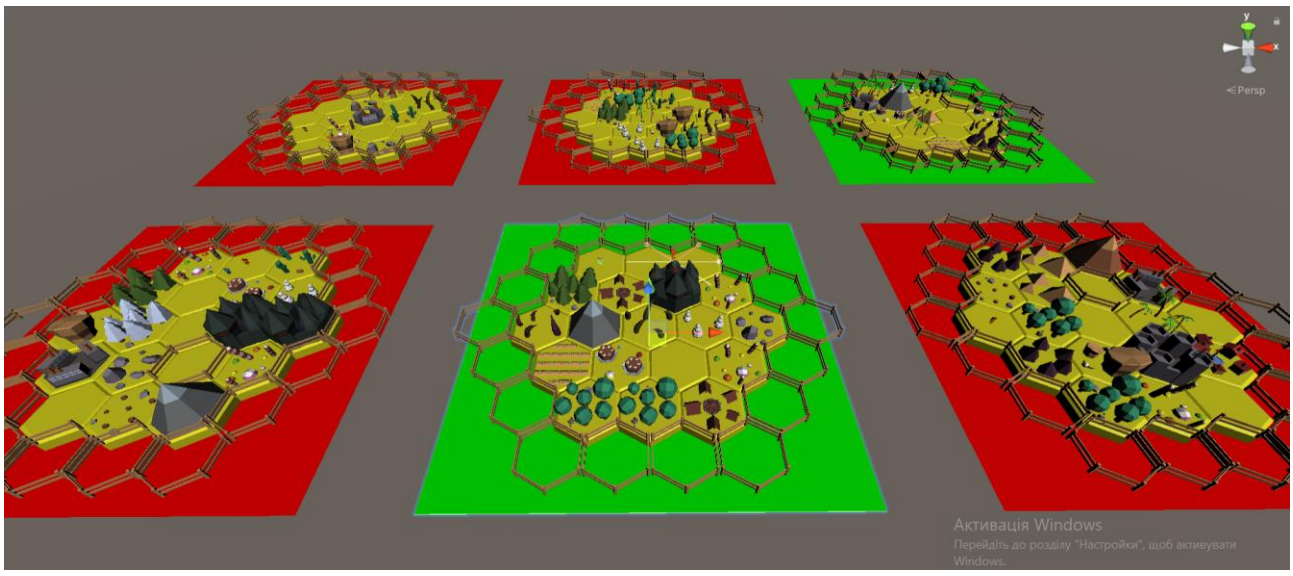

Рис. 3.10 Вигляд навчання у вікні Unity (зеленим позначені середовища, у яких агент успішно дійшов до цілі, червоним – у яких агент торкнувся до перешкоди)

Як бачимо з рис. 3.9, середнє значення нагороди за епізод змінило своє значення з найменшого - 0.832, до найбільшого 6.363, а в самій симуляції стало видно, що замість хаотичного руху агентів вони навчилися прямувати до цілі.

Щоб навчання було максимально ефективним, потрібно спостерігати за поведінкою агентів, та корегувати значення нагороди за різні дії. Наприклад, при першому запуску агенти не мають уявлення, що їм робити, вони торкаються до перешкод, та іноді, випадково знаходять ціль. Для цього, окрім негативної нагороди за дотики до перешкод, та позитивної за збирання цілі, було додано спеціальну нагороду за зменшення відстані між агентом та ціллю. Проте, трапився випадок, коли агенти розпочинали стояти від цілі на мінімальній відстані, отримуючи від цього більшу нагороду, ніж від торкання до цілі. Для вирішення даної проблеми, з кожним наступним навчанням було зменшено значення нагороди за відстань.

Для спонукання агентів швидше прямувати до цілі, було додано негативну нагороду, за кожен момент часу, коли агенти навчаються

3.3. Аналіз результатів та впровадження навченого інтелекту у гру

Для аналізу успішності навчання ШІ, у віртуальному середовище у консолі було виконано команду “tensorboard --logdir results --port 6006”, яка відповідає за відображення інформації щодо навчання у вигляді графіків, використовуючи сервіс TensorBoard.

На рис. 3.11 зображено графік, вісь Y якого відповідає за середнє значення накопичуваної нагороди агентів, а вісь X – це кількість кроків тренування агентів. Графік відображує те, що зі збільшенням кількості кроків тренування, збільшувалася і ефективність ШІ. Вони почали частіше діставатися цілі, та менше торкатися до перешкод. Графік синього кольору, що відповідає за другий запуск навчання знаходиться нижче відносно графіка першого навчання, тому що була зменшена нагорода за відстань до цілі, при сталій навченості агентів.

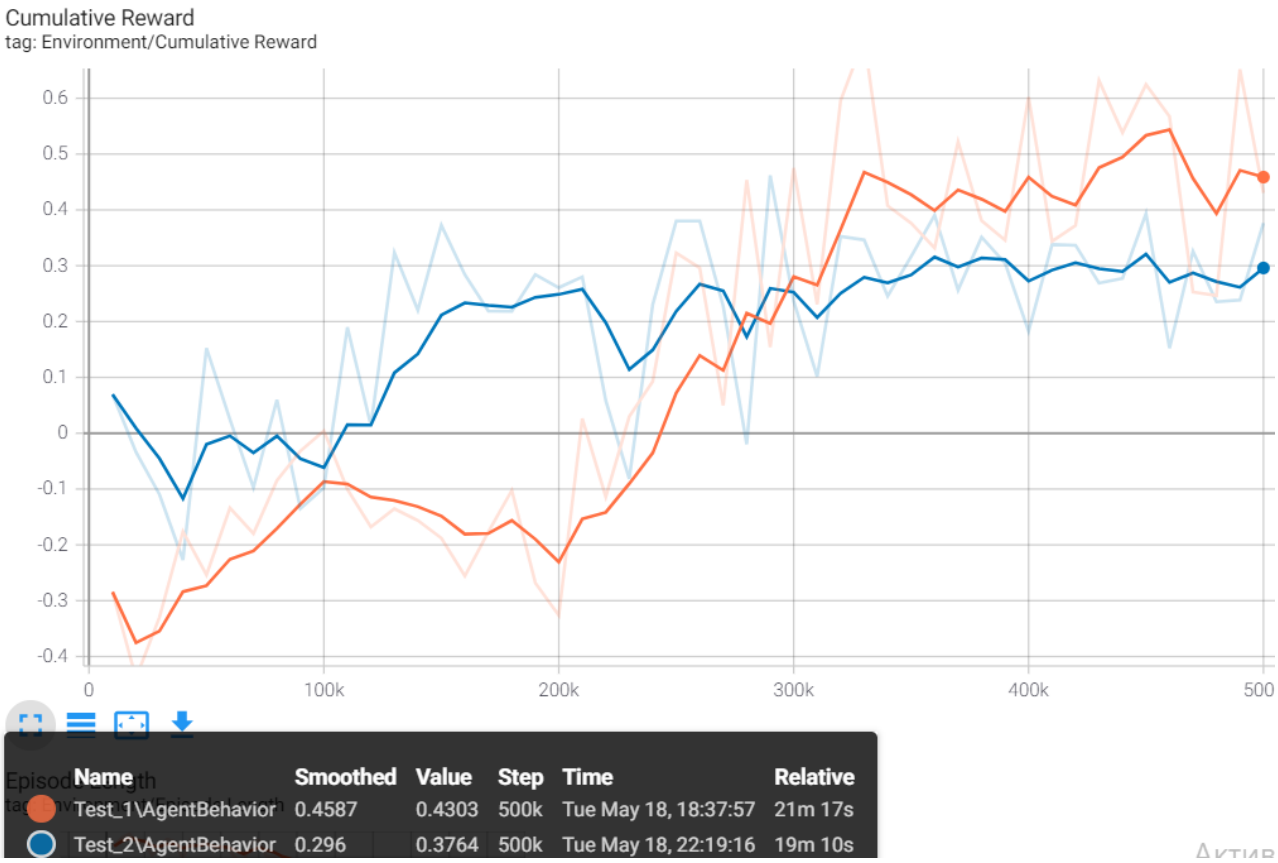

Рис. 3.11 Графіки накопичуваної нагороди перших двох тренувань

Не менш важливим графіком для аналізу успішності навчання, є графік тривалості епізодів навчання агента. Епізод вважається завершеним, коли агент торкається до цілі, або через 1000 кроків навчання. При дотику до перешкоди, епізод не завершується. Як видно на графіку, зі збільшенням кількості кроків (вісь X), зменшується час епізодів (вісь Y) , це свідчить про те, що агенти починають краще справлятися з завданням, а саме - швидше дістаються цілі.

Початок синього графіка, ніби продовжує помаранчевий графік, тому що між першим та другим запуском не було змін, що могли б повпливати на тривалість епізодів.

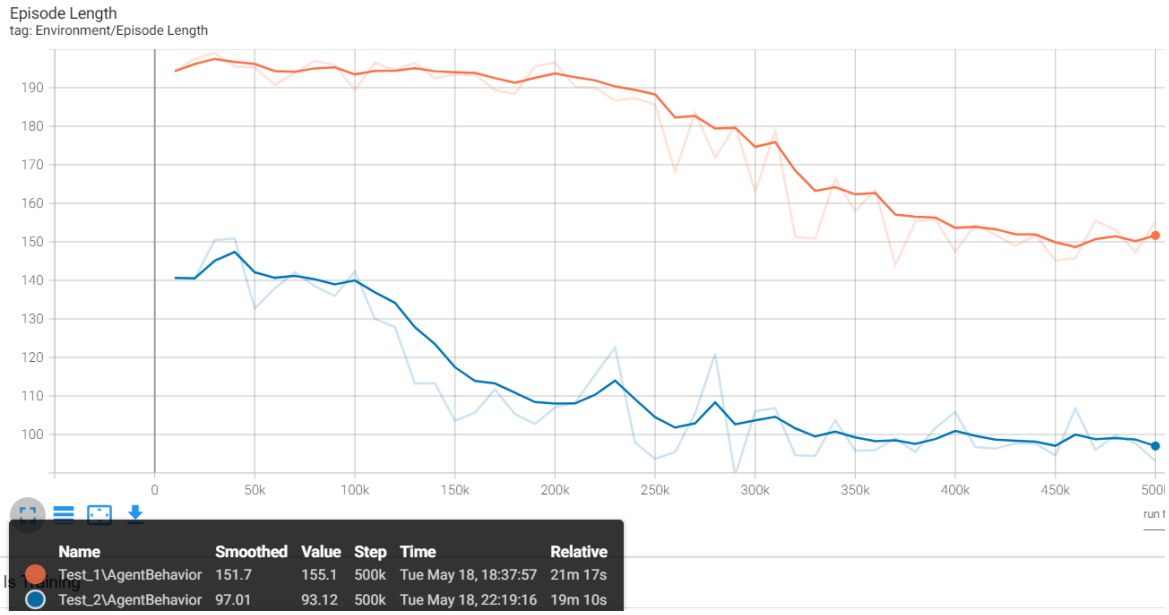

Рис. 3.12 Графіки тривалості епізодів перших двох тренувань

При третьому тренуванні була значно збільшена негативна нагорода за дотики до перешкод, тому порівняно з графіками перших двох тренувань, графік третього тренування знаходиться набагато нижче, маючи досить низькі показники нагороди.

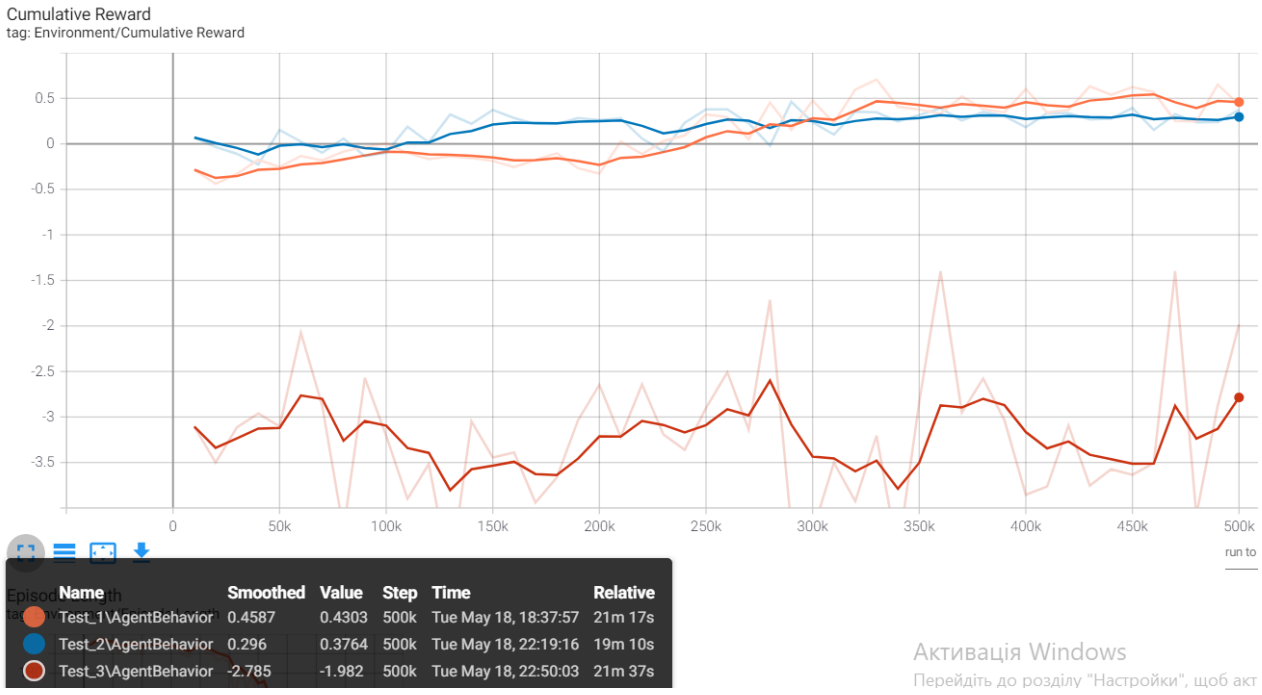

Рис. 3.13 Графіки нагороди усіх трьох тренувань

Проте, метою було зробити так, щоб персонажі менше торкалися до перешкод, а чи це вдалося видно з графіка тривалості епізодів (рис.3.14), порівняно з першими двома тренуваннями. Було досягнуто того, що тривалість епізодів зросла, а отже це значить, що агенти почали витрачати більше часу на обходження перешкод, замість пересування, торкаючись до них.

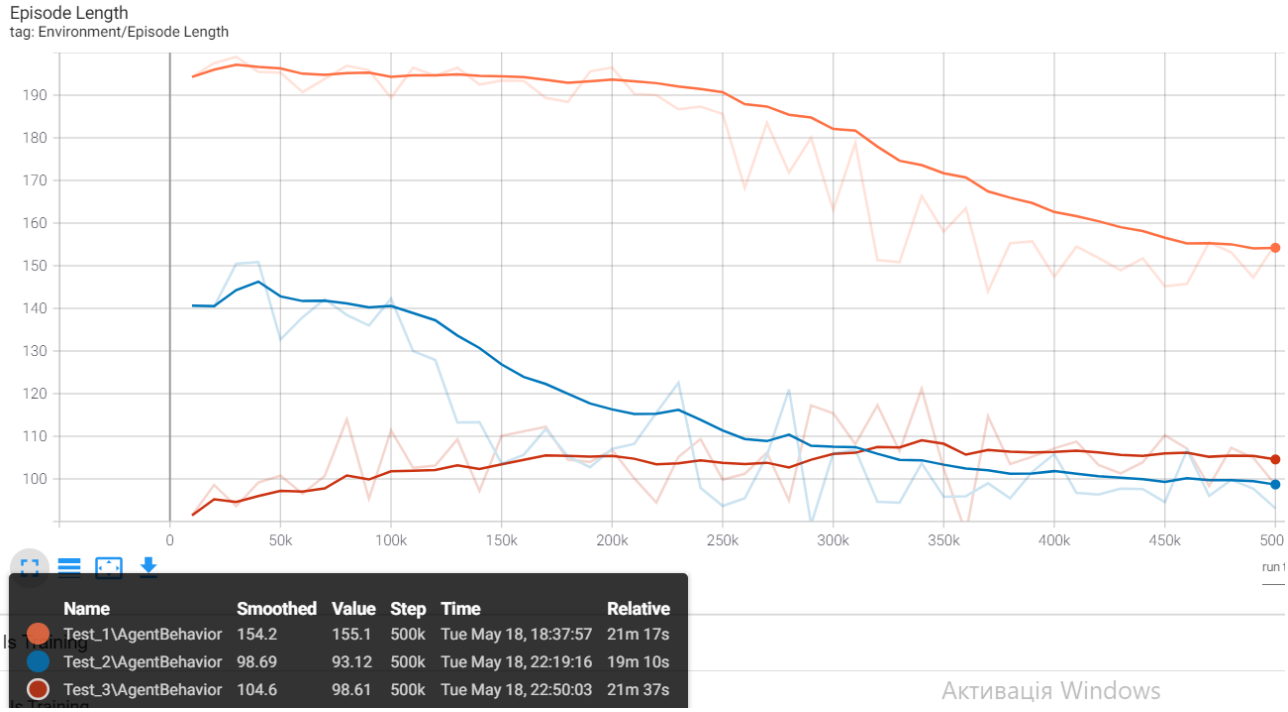

Рис. 3.14 Графіки тривалості епізодів усіх тренувань

По закінченню тренування, файли нейронної моделі зберігаються у папка_проекту \ results \ назва_тренування.

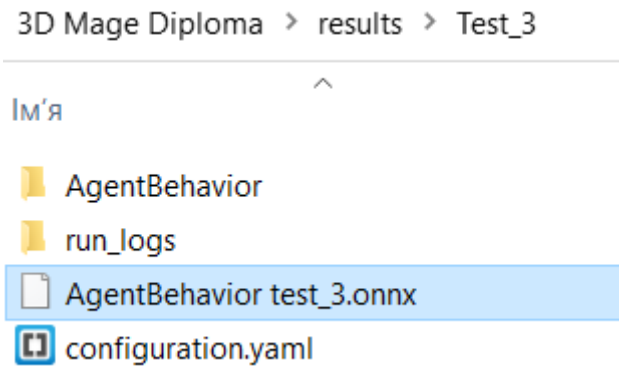

Рис. 1.15 розміщення нейронної мережі

Для того, щоб використовувати її для реалізації штучного інтелекту у грі, її потрібно перемістити у поле Model компонента Behavior Parameters агента, та у полі Behavior Type вибрати Inference Only, що вказує на те, що ми не хочемо навчати цього агента, а будемо використовувати готову нейронну модель поведінки.

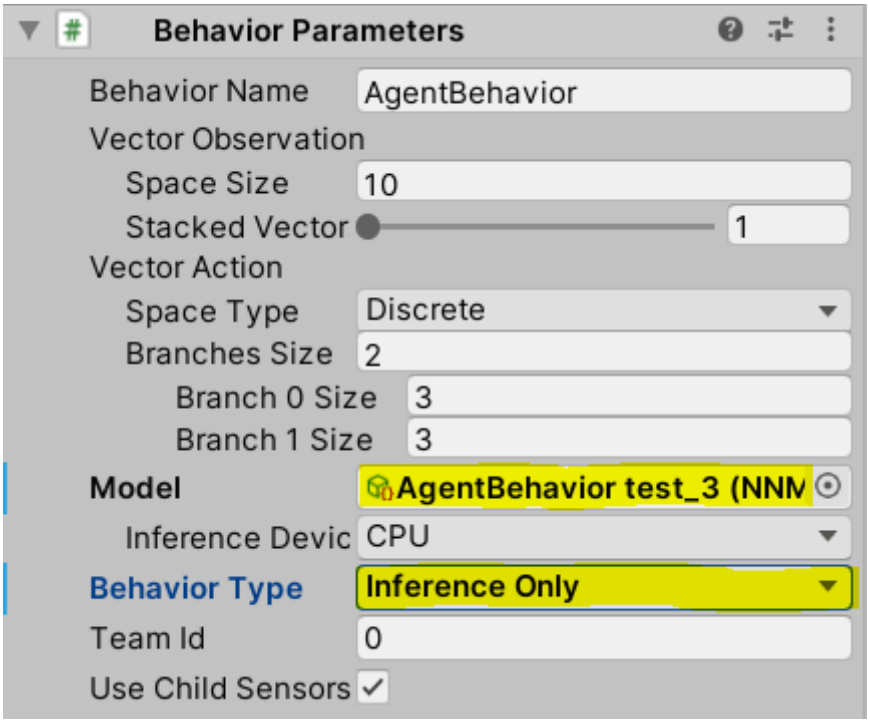

Рис. 3.16 Налаштування агента для готової нейронної моделі

Для реалізації відслідковування гравця противниками, противникам було додано компоненти агента, а капсульний колайдер гравця було поставлено на рівень Target, та додано тег Target, який був у цілі для агентів.

3.4. Висновки до 3 розділу

Отже, у третьому розділі було описано процес розробки машинно навченого штучного інтелекту та його реалізація в грі.

Було установлено необхідні для навчання бібліотеки та налаштовано віртуальне середовище в Python. Також було створено та налаштовано агента та середовище для навчання в Unity.

Було проаналізовано значення параметрів, що відповідають за конфігурацію процесу навчання. Для різних типів задач ці параметри різні, а отже було підібрано саме необхідні для отримання максимальної успішності та ефективності навчання параметри.

За допомогою графіків було проаналізовано результати навчання, та зроблено висновок, що воно пройшло успішно.

ШІ було впроваджено у гру, використовуючи натреновану модель поведінки агента.

ВИСНОВКИ

Отже, під час виконання випускної кваліфікаційної бакалаврської роботи на тему : “Розробка комп'ютерної 3Д гри жанру шутер з елементами штучного інтелекту” було проведено дослідження теоретичної бази предметної області.

Успішно вдалося досягти мети роботи, а саме : Дослідити методи машинного навчання штучного інтелекту та створити повноцінний програмний продукт – гру з використанням штучного інтелекту.

У ході досягнення мети роботи було вирішено основні завдання. Було досліджено особливості розробки комп’ютерних ігор та проаналізовано існуючі аналоги. Було також проаналізовано методи створення ШІ, серед яких було вибрано метод reinforcement machine learning. Було розроблено повноцінну 3Д гру та реалізовано в ній машинно навчений ШІ.

Об’єктами дослідження при виконанні завдання роботи були індустрія відеоігор та методи створення штучного інтелекту

Предметами дослідження при виконанні завдання роботи були ігри зі штучним інтелектом та машинне навчання штучного інтелекту.

В роботі наведені діаграми, що описують архітектуру як самої гри, так і її окремих компонентів.

Розроблена гра має свій продуманий сюжет, унікальні механіки взаємодії з противниками, 9 видів противників з особливими атрибутами, 6 видів зброї, кожен з яких має здатність покращуватися, також наявні 7 додаткових неігрових персонажів.

Результатом дослідження також стало отримання штучного інтелекту, реалізованого за допомогою методу reinforcement machine learning – машинного навчання з підкріпленням. Цей метод характерний тим, що для навчання агента використовується метод нагороди агента за правильні дії, та покарання за неправильні, в наслідок чого нейронна мережа аналізує всі вхідні дані та корегує вихідні дані таким чином, щоб агент отримував дедалі більший середній результат.

Штучний інтелект було натреновано в 3 етапи, під час кожного з яких агенти пройшли по 50000 кроків навчання. Нейронна мережа на якій тренувався ШІ має 2 приховані шари по 128 вузлів, це забезпечувало ефективне навчання та отримання якісних результатів.

Дана робота є актуальною та її результати можуть використовуватися для покращенння аналізу методів машинного навчання за допомогою ігрових симуляцій. На основі отриманих результатів можна створювати більші та складніші системи зі штучним інтелектом.

Саму ж гру можна використовувати у розважальних та навчальних цілях, оскільки в ній реалізоване досить незвичайне ведення бою з противниками, під час якого розвивається реакція та уважність гравця, коли він намагається знайти потрібні уламки зброї. Також розвивається уміння мислити гравця, оскільки йому потрібно вирішувати, використовувати зброю без покращення та втратити її, але швидше вбити противника, або ж покращити зброю, тим самим зберегти її при нанесенні критичного удару, але пожертвувати часом та здоров’ям.

У грі наявна мікроекономіка. Гравцю завжди потрібно відслідковувати кількість кубиків та планувати, якими з них жертвувати заради інших, це розвиває у гравця елементарне розуміння економіки.

Різнобарвні персонажі та зброя можуть допомогти дітям вивчити кольори, адже під час процесу гри від гравця потребується вміння гравця знати та розрізняти кольори.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. mclander. Почему люди играют в игры [Електронний ресурс] / mclander. – 18. – Режим доступу до ресурсу: https://habr.com/ru/post/319798/.

2. azazelis. игровой ИИ [Електронний ресурс] / azazelis. – 6. – Режим доступу до ресурсу: https://habr.com/ru/company/pixonic/blog/428892/.

3. vincentpierre. ml-agents [Електронний ресурс] / vincentpierre – Режим доступу до ресурсу: https://github.com/Unity-Technologies/ml-agents.

4. Жанры компьютерных игр (общая схема) [Електронний ресурс] – Режим доступу до ресурсу: https://gamesisart.ru/TableJanr.html#RPG.

5. WARFRAME [Електронний ресурс] – Режим доступу до ресурсу: https://warframe.fandom.com/wiki/WARFRAME.

6. Action role-playing game [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Action_role-playing_game.

7. denshust. Colour Mage Arena [Електронний ресурс] / denshust. – 2021. – Режим доступу до ресурсу: https://denshust.github.io/ColourMageArena/.

8. Aseprite Tutorials [Електронний ресурс] – Режим доступу до ресурсу: https://www.aseprite.org/docs/tutorial/.

9. Олександр Марголін. UML для бізнес-моделювання [Електронний ресурс] / Олександр Марголін – Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/uml-diagrams.html.

10. Pedro Medeiros. How to start making pixel art [Електронний ресурс] / Pedro Medeiros. – 2018. – Режим доступу до ресурсу: https://medium.com/pixel-grimoire/how-to-start-making-pixel-art-2d1e31a5ceab.

11. Magicavoxel [Електронний ресурс] – Режим доступу до ресурсу: https://www.voxelmade.com/magicavoxel/.

12. Unity User Manual (2019.4 LTS) [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/2019.4/Documentation/Manual/index.html.

13. Unity. Character Controller [Електронний ресурс] / Unity – Режим доступу до ресурсу: https://docs.unity3d.com/ru/2019.4/Manual/class-CharacterController.html.

14. Brackeys. THIRD PERSON MOVEMENT in Unity [Електронний ресурс] / Brackeys. – 24. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=4HpC--2iowE.

15. Нейронні мережі - шлях до глибинного навчання [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://codeguida.com/post/739.

16. Training Configuration File [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md.

17. Code Monkey. How to use Machine Learning AI in Unity! (ML-Agents) [Електронний ресурс] / Code Monkey. – 29. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=zPFU30tbyKs.

18. Acacia Developer. Fixing Slope Bouncing and Jittering [Електронний ресурс] / Acacia Developer. – 2018. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=b7bmNDdYPzU

ДОДАТКИ

Додаток А

Скрипт-парсер, для конвертування файлу формату text в префаб з кубиків

```
#if UNITY_EDITOR
using System.Collections.Generic;
using System.IO;
using UnityEditor;
using UnityEngine;

public class ParseStructure : MonoBehaviour
{
    struct VoxelsRawData
    {
        public Vector3 pos;
        public string col;

        public VoxelsRawData(Vector3 pos, string col)
        {
            this.pos = pos;
            this.col = col;
        }
    }

    [MenuItem("Tools/Конвертувати воксельну структуру з text в префаб")]
    public static void Do()
    {
        if (Selection.assetGUIDs.Length == 0 || Selection.assetGUIDs.Length > 1)
        {
            Debug.LogError("Не обрано жодний файл !");
            return;
        }

        var assetGUID = Selection.assetGUIDs[0];
        var assetPath = AssetDatabase.GUIDToAssetPath(assetGUID);

        StreamReader myFile = new StreamReader(assetPath);
        var rawVoxelDatas = ParseRawVoxelsFromFile(myFile);
        myFile.Close();

        var rootGameObject = CreateVoxelStructure(rawVoxelDatas);
        Selection.activeObject = rootGameObject;
        PrefabUtility.SaveAsPrefabAsset(rootGameObject, "Assets/NewMonsters/destructibles/"
+ Path.GetFileName(assetPath) + ".prefab");
        DestroyImmediate(rootGameObject);
    }

    private static List<VoxelsRawData> ParseRawVoxelsFromFile(StreamReader file)
    {
        var rawVoxelDatas = new List<VoxelsRawData>();

        string line;
        while ((line = file.ReadLine()) != null)
        {
            if (line[0] == '#' || line[0] == ' ')
            {
                continue;
```

```
            }
            else
            {
                var split = line.Split();
                Vector3 position = new Vector3(float.Parse(split[0]), float.Parse(split[2]),
float.Parse(split[1])); //goxel z = unity y
                rawVoxelDatas.Add(new VoxelsRawData(position, split[3]));
            }
        }

        return rawVoxelDatas;
    }

    private static GameObject CreateVoxelStructure(List<VoxelsRawData> rawVoxelData)
    {
        GameObject root = new GameObject();
        for (int i = 0; i < rawVoxelData.Count; i++)
        {
            var cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
            cube.layer = LayerMask.NameToLayer("Voxel");
            cube.transform.SetParent(root.transform);
            cube.transform.position = rawVoxelData[i].pos;
            var material = GetOrCreateVoxelMaterial(rawVoxelData[i].col);
            cube.GetComponent<Renderer>().sharedMaterial = material;
        }
        return root;
    }

    private static Material GetOrCreateVoxelMaterial(string colorCode)
    {
        Material material =
(Material)AssetDatabase.LoadAssetAtPath("Assets/NewMonsters/destructibles/" + colorCode +
".mat", typeof(Material));
        if (material == null)
        {
            material = CreateVoxelMaterial(colorCode);
        }
        return material;
    }

    private static Material CreateVoxelMaterial(string colorCode)
    {
        var material = new Material(Shader.Find("Universal Render Pipeline/Lit"));
        Color col;
        ColorUtility.TryParseHtmlString("#" + colorCode, out col);
        if (col == null)
        {
            Debug.LogError("Помилка при конвертації кольору : #" + colorCode);
        }
        else
        {
            material.color = col;
        }
        AssetDatabase.CreateAsset(material, "Assets/NewMonsters/destructibles/" + colorCode
+ ".mat");
        return material;
    }
}
#endif
```

Додаток Б

Скрипт Collectable.cs, що відповідає за здатність кубиків бути зібраними

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

public class Collectable : MonoBehaviour
{
    public bool isCollectable;
    public bool isDestructable=true;
    public string color;

    private void Start()
    {
        isCollectable = false;
        isDestructable = true;
    }

}
```

Додаток В

Скрипт Player_Movement_2, що відповідає за рух гравця

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

public class Player_Movement_2 : MonoBehaviour
{
    public float moveSpeed = 12f;
    public float gravity = -9.81f;
    public float jumpHeight = 10f;
    public float slopeForce;
    public float slopeRayLength;
    public Transform groundCheck;
    public Transform ceilingCheck;
    public LayerMask groundMask;
    public LayerMask ceilingMask;
    public Volume PostPros;
    public Camera mainCam;
    public Animator animator;
    private LensDistortion ld;
    private Vector3 move;
    private CharacterController controller;
    private Vector3 velocity;
    private Vector3 drag;
    private bool isGrounded;
    private bool touchCeiling;
    private bool jumped;

    private void Start()
    {
        controller = GetComponent<CharacterController>();
    }
```

```
    void Update()
    {
        isGrounded = Physics.CheckSphere(groundCheck.position,
groundCheck.gameObject.GetComponent<SphereCollider>().radius*gameObject.transform.localScale
.x, groundMask);
        touchCeiling = Physics.CheckSphere(ceilingCheck.position,
ceilingCheck.gameObject.GetComponent<SphereCollider>().radius *
gameObject.transform.localScale.x, ceilingMask);
        if (touchCeiling)
        {
            velocity.y = velocity.y * -1f;
        }
        if (isGrounded && velocity.y < 0) // коли падає
        {
            controller.slopeLimit = 45.0f;
            velocity.y = -2f; // швидкість початку падіння
            if (jumped)
            {
                Land();
            }

        }
        if (Input.GetButton("Jump") && isGrounded )
        {
            Jump();
        }
        if (Input.GetKeyDown(KeyCode.LeftShift)) //нажато шифт
        {
            StartCoroutine(Dash());
        }
        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");
        if (x != 0 && z != 0) //перевірка руху по діагоналі, ділення швидкості на корінь з
двох, якщо так
        {
            x = x / Mathf.Sqrt(2);
            z = z / Mathf.Sqrt(2);
        }
        if (x != 0 || z != 0)  //якщо рухається
        {
            animator.SetBool("swing", true);
        }
        else
        {
            animator.SetBool("swing", false);
        }
        move = transform.right * x + transform.forward * z;
        controller.Move(move * moveSpeed * Time.deltaTime);
        velocity.y += gravity * Time.deltaTime;
        controller.Move(velocity * Time.deltaTime);
        velocity.x /= 2 + drag.x * Time.deltaTime;
        velocity.y /= 1 + drag.y * Time.deltaTime;
        velocity.z /= 2 + drag.z * Time.deltaTime;

        if ((z != 0 || x != 0) && OnSlope())
        {
            controller.Move(Vector3.down * controller.height / 2 * slopeForce *
Time.deltaTime);
        }
    }
    private bool OnSlope() // якщо на схилі
    {
        if (!isGrounded)
        {
```

```
            return false;
        }
        RaycastHit hit;
        if (Physics.Raycast(transform.position, Vector3.down, out hit, controller.height / 2
* slopeRayLength))
        {
            if (hit.normal != Vector3.up)
                return true;
        }
        return false;
    }
    public void Jump() //прижок
    {
        velocity.y = jumpHeight;
        controller.slopeLimit = 90.0f;
        jumped = true;
        animator.SetBool("jump", true);
        Debug.Log("Jupmped");
    }
    public void Land() //приземлення
    {
        jumped = false;
        animator.SetBool("jump", false);
        Debug.Log("Landed");
    }
    IEnumerator Dash() //ривок
    {
        PostPros.profile.TryGet<LensDistortion>(out ld);
        ld.active = true;
        for (float i = 0; i > -1; i -= 0.1f)
        {
            yield return new WaitForSeconds(0.02f);
            ld.intensity.value = i;
        }
        yield return new WaitForSeconds(0.3f);
        velocity += Vector3.Scale(transform.forward, 30 * new Vector3((Mathf.Log(1f /
(Time.deltaTime * drag.x + 2)) / -Time.deltaTime), (Mathf.Log(1f / (Time.deltaTime * drag.y
+ 2)) / -Time.deltaTime), (Mathf.Log(1f / (Time.deltaTime * drag.z + 2)) / -
Time.deltaTime)));
        ld.active = false;
    }
}
```

Додаток Г

Скрипт MLAgent, що відповідає за реалізацію агента машинного навчання ШІ

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;

public class MLAgent : Agent
{
    [SerializeField] private Transform targetTransform;
    [SerializeField] private Material winMat;
    [SerializeField] private Material looseMat;
    [SerializeField] private MeshRenderer render;
    [SerializeField] private GameObject spawner;
    private float goal, death, time, dis, face, top, staying, touching;
```

```
    private CharacterController controller;
    private Vector3 move;
    private Vector3 velocity;
    public float moveSpeed = 12f;
    public float gravity = -9.81f;
    public override void Initialize()
    {
        goal = 1f;
        death = -1f;
        time = -0.01f;
        dis = 0.01f;
        staying = death / 10f;
        touching = death / 10f;

        controller = GetComponent<CharacterController>();
    }
    public override void OnEpisodeBegin()
    {
        spawner.GetComponent<GameHandler2>().Restart();
    }
    public override void CollectObservations(VectorSensor sensor)
    {
        sensor.AddObservation(transform.localPosition);
        sensor.AddObservation(targetTransform.localPosition);
        sensor.AddObservation(controller.velocity);
        sensor.AddObservation(Vector3.Distance(transform.localPosition,
targetTransform.localPosition));
    }
    public override void OnActionReceived(float[] vectorAction)
    {
        int moveX = 0;
        int moveZ = 0;
        switch (vectorAction[0])
        {
            case 0:
                moveX = 0;
                break;
            case 1:
                moveX = 1;
                break;
            case 2:
                moveX = -1;
                break;
        }
        switch (vectorAction[1])
        {
            case 0:
                moveZ = 0;
                break;
            case 1:
                moveZ = 1;
                break;
            case 2:
                moveZ = -1;
                break;
        }

        float x = moveX;
        float z = moveZ;

        if (x != 0 && z != 0) //перевірка руху по діагоналі, ділення швидкості на корінь з
двох, якщо так
        {
            x = x / Mathf.Sqrt(2);
```

```
            z = z / Mathf.Sqrt(2);
        }
        move = transform.right * x + transform.forward * z;
        controller.Move(move * moveSpeed * Time.deltaTime);
        velocity.y += gravity * Time.deltaTime;
        controller.Move(velocity * Time.deltaTime);
        AddReward(time); // за час
        AddReward(dis / Vector3.Distance(transform.localPosition,
targetTransform.localPosition));
        //Debug.Log(GetCumulativeReward());
    }
    public override void Heuristic(float[] actionsOut)
    {
        int moveX = 0;
        int moveZ = 0;
        if (Input.GetAxis("Horizontal") > 0)
            moveX = 1;
        else if (Input.GetAxis("Horizontal") < 0)
            moveX = 2;
        else
            moveX = 0;

        if (Input.GetAxis("Vertical") > 0)
            moveZ = 1;
        else if (Input.GetAxis("Vertical") < 0)
            moveZ = 2;
        else
            moveZ = 0;
        actionsOut[0] = moveX;
        actionsOut[1] = moveZ;
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Environment")
        {
            render.material = looseMat;
            SetReward(touching);

        }
        if (other.gameObject.tag == "Plane")
        {
            render.material = looseMat;
            EndEpisode();

        }
        if (other.gameObject.tag == "Target")
        {
            SetReward(goal);
            render.material = winMat;
            EndEpisode();
        }
    }
    private void OnTriggerStay(Collider other)
    {
        if (other.gameObject.tag == "Environment")
        {
            SetReward(staying);
        }
    }
}
```

Додаток Д

Скрипт що відповідає за реалізацію притягування кубиків до об’єкта Collector

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

public class Magnet : MonoBehaviour
{
    [SerializeField] private Animator myAnimController;
    public int collectspeed;
    public GameObject collector;
    public LayerMask piceMask;
    public LayerMask collectableMask;
    public bool appear;
    public bool open;
    private void Update()
    {
        appear = Physics.CheckSphere(gameObject.transform.position, 13, piceMask);
        myAnimController.SetBool("appear", appear);
        open = Physics.CheckSphere(gameObject.transform.position, 10.5f, collectableMask);
        myAnimController.SetBool("open", open);
    }
    private void OnTriggerStay(Collider other)
    {
        if (other.gameObject.tag == "Piece")
        {
            if (other.gameObject.GetComponent<Collectable>().isCollectable == true)
            {
                other.gameObject.layer = 13;
                other.GetComponent<MeshRenderer>().shadowCastingMode =
UnityEngine.Rendering.ShadowCastingMode.Off;
                other.GetComponent<Rigidbody>().useGravity = false; // відключаємо
гравітацію
                other.GetComponent<BoxCollider>().isTrigger = true; // ставимо коллайдер як
тріггер
                other.GetComponent<Collectable>().isDestructable = false; // забороняємо
знищення
                other.GetComponent<Rigidbody>().velocity = Vector3.zero; // обнуляємо
прискорення
                other.transform.position = Vector3.MoveTowards(other.transform.position,
                collector.transform.position , collectspeed * Time.deltaTime); // притягуємо
до магніта
            }
        }
    }
    private void OnTriggerExit(Collider other)
    {
        if (other.gameObject.tag == "Piece")
        {
            other.gameObject.layer = 10;
            other.GetComponent<MeshRenderer>().shadowCastingMode =
UnityEngine.Rendering.ShadowCastingMode.On;
            other.GetComponent<Rigidbody>().useGravity = true;
            other.GetComponent<BoxCollider>().isTrigger = false;
            other.GetComponent<Collectable>().isDestructable = true;
        }
    }
}
```

Додаток Е

Скрипт Collect об’єкта Collector, що відповідає за збирання кубиків різних кольорів

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class Collect : MonoBehaviour
{
    struct MyColor
    {
        public string name;
        public int count;
    }
    MyColor[] colors = new MyColor[13];
    public Text collectedText;
    string stringToPrint;
    private void Start()
    {
        colors[0].name = "red";
        colors[1].name = "orange";
        colors[2].name = "yellow";
        colors[3].name = "light green";
        colors[4].name = "dark green";
        colors[5].name = "cyan";
        colors[6].name = "light blue";
        colors[7].name = "dark blue";
        colors[8].name = "beige";
        colors[9].name = "brown";
        colors[10].name = "white";
        colors[11].name = "grey";
        colors[12].name = "black";
    }
    private void OnTriggerStay(Collider other)
    {
        if (other.gameObject.tag ==
"Piece"&&other.gameObject.GetComponent<Collectable>().isCollectable==true)
        {
            string color = other.GetComponent<Collectable>().color;
            for (int i = 0; i < colors.Length; i++)
            {
                if(color=="")
                {
                    Debug.LogError(other.GetComponent<MeshRenderer>().material);
                }
                if(color==colors[i].name)
                {
                    colors[i].count++;
                }
            }
            for (int i = 0; i < colors.Length; i++)
            {
                stringToPrint += colors[i].name + " : " + colors[i].count + "\n";
            }
            collectedText.text = stringToPrint;
            stringToPrint = "";
            Destroy(other.gameObject);
        }
    }
}
```

Додаток Ж

Скрипт SunCycle, що відповідає за візуальну зміну дня і ночі.

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

public class SunCycle : MonoBehaviour
{
    public float time;
    public TimeSpan current;
    public Transform sunTransform;
    public Light sun;
    public float intensity;
    public float speed;
    public int day;
    public Volume volume;
    private Bloom bl;
    void Update()
    {
        ChangeTime();
    }
    public void ChangeTime()
    {
        time += Time.deltaTime * speed;
        if(time>86400)
        {
            day += 1;
            time = 0;
        }
        current = TimeSpan.FromSeconds(time);
        string[] tempTime = current.ToString().Split(":"[0]);
        Debug.Log(tempTime[0] + " : " + tempTime[1]);
        sunTransform.rotation = Quaternion.Euler(new Vector3((time - 21600) / 86400 * 360,
0, 0));
        if (time < 43200)
            intensity = 1 - (43200 - time) / 43200;
        else
            intensity = 1 - ((43200 -time) / 43200 *-1);
        RenderSettings.ambientLight = Color.Lerp(new Color(0.5f,0.5f,0.5f,1), new
Color(0.9f, 0.9f, 0.9f, 1), intensity*intensity);
        volume.profile.TryGet<Bloom>(out bl);
        bl.threshold.value = Mathf.Lerp(0.5f, 3f, intensity*intensity );
        sun.intensity = intensity*0.5f;
    }
}
```

Додаток И

Taras Shevchenko National University of Kyiv

Development of a computer 3D shooter game
with elements of artificial intelligence
Software Architecture Document (SAD)

CONTENT OWNER: **Denys Shust**

DOCUMENT NUMBER: 1.0

RELEASE: 1.0

RELEASE DATE: 02.06.2021

Table of Contents

1. Introduction

1.1. Purpose

This document provides an overview of the designed game, using different architectural views to depict different aspects of the game. It is designed to to briefly summarize all the important aspects of the game architecture and capture and transmit important architectural decisions made in the game.

1.2. Scope

This Software Architecture Document provides an architectural overview of the designed 3D shooter game. The the game is made with the implementation of artificial intelligence in it.

1.3. Definitions

Artificial intelligence in the game was made using reinforcement machine learning. Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

1.4 References

Applicable references are:

- Unity ML Agents
- Machine learning
- Gaming industry
- C# Unity scripting
- PC games genres

2. Architecture Representation

This document presents the architecture as a series of diagrams. The following were used here :

- State diagram
- Component diagram
- Activity diagram
- Use-case diagram

3. Architectural Goals and Constraints

The main goal is to create a clear specification of the developed architecture of the game. The game itself must be optimized and its source code should be easy to understand, but the mechanics should be unique and complex.

4. State diagram

The state diagram gives an abstract description of the behavior of a system. It shows the states of the game, starting from the main menu and ending with it. Total states are :

- Menu
- The beginning of the game
- Game
- Exit the game

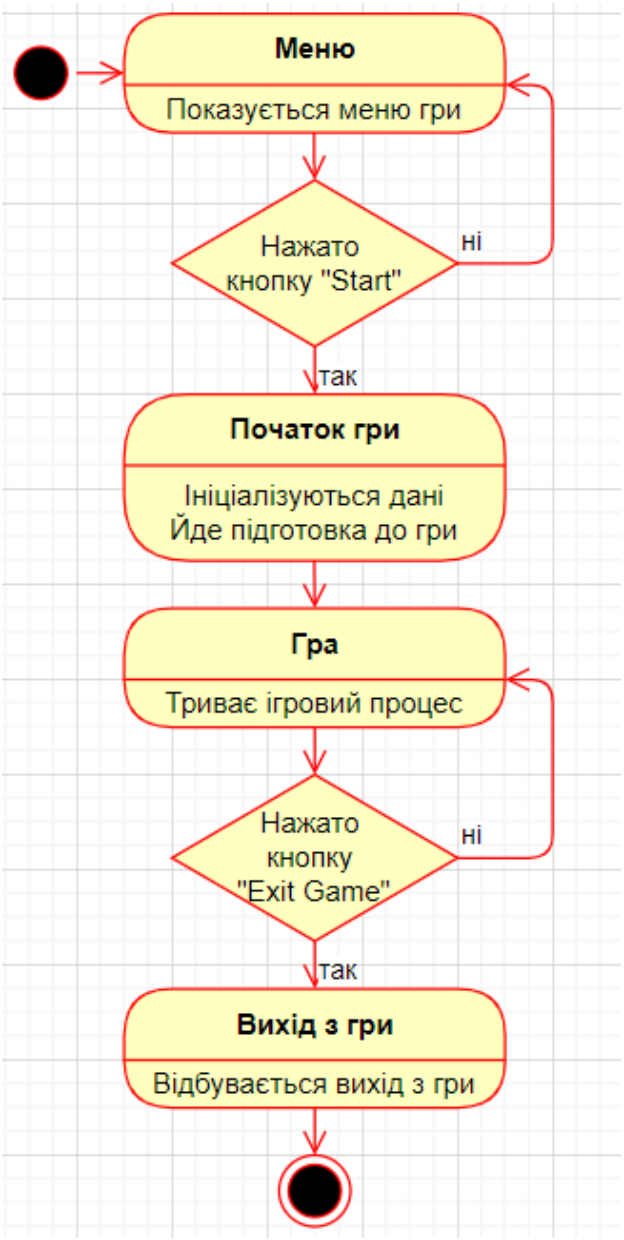

Figure 1: State diagram

5. Component diagram

The component diagram depicts the structure of the Player game object. Rectangles represent game objects that are part of the Player. The note icon indicates the scripts of the corresponding game objects, the components of which are these scripts.

ChestHolder - includes Collector and Chest, which in turn are responsible for assembling the cubes, and visually display this process.

WeaponHolder - is responsible for controlling the weapon and providing it with visual support in the form of the appropriate color.

Magnet - is responsible for attracting the assembly cubes to the Collector object

CameraController - is responsible for controlling the Main and UI cameras and placing Point Light and Directional Light light sources.

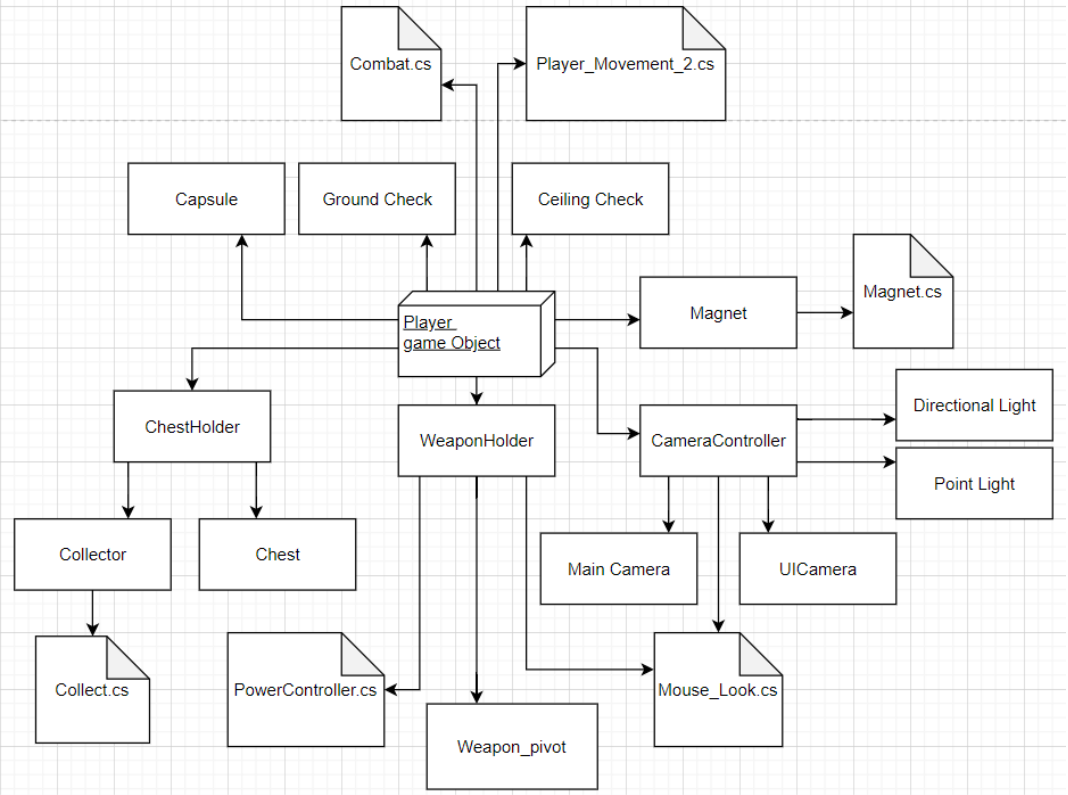

Figure 2: Component diagram

6. Activity diagram

The activity diagram describes the features of the mechanics of the weapons. Based on it, we can conclude that the player has several options for action when assembling weapons:

• Immediately attack the desired enemy with it and inflict normal damage on it, after which the weapon will be destroyed and you will have to look for the fragment.

• Refrain from using unimproved weapons, collect another piece to improve it and cause increased damage to the enemy, and also keep the weapon from destruction.

Figure 3: Activity diagram

7. Use-case diagram

The Use-case diagram is important input to the selection of the set of scenarios and/or use cases that are the focus of an iteration. It describes the set of scenariosand/or use cases that represent some significant, central functionality. It alsodescribes the set of scenarios and/or use cases that have a substantial architectural coverage (that exercise many architectural elements) or thatstress or illustrate a specific, delicate point of the architecture.

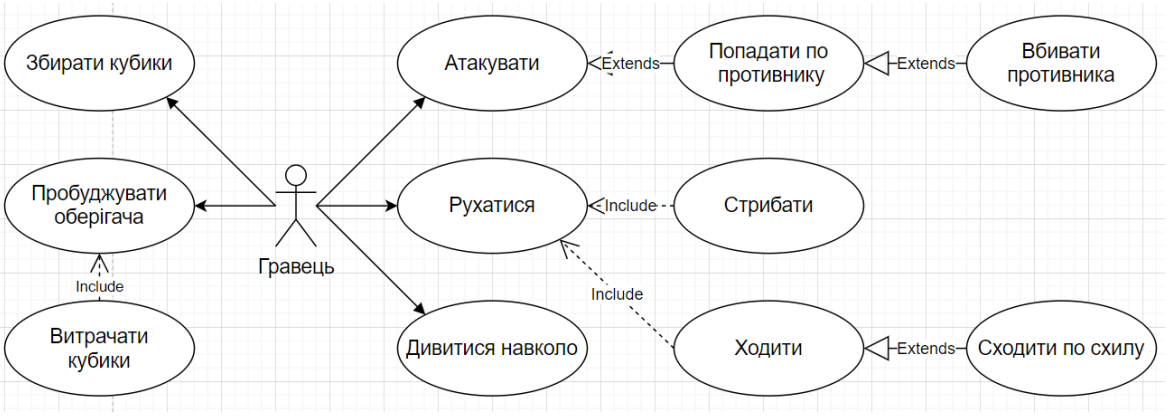

Figure 4: Use-case diagram

8. Size and Performance

- The total volume of game files should be up to 1 GB.
- The game must respond immidiately to player actions.
- The game should not have any bugs or errors.
- Enemies' artificial intelligence must work well.

9. Referenced materials

vincentpierre. ml-agents	https://github.com/Unity-Technologies/ml-agents
Action role-playing game	https://en.wikipedia.org/wiki/Action_role-playing_game
Unity User Manual (2019.4 LTS)	https://docs.unity3d.com/2019.4/Documentation/Manual/index.html
Unity. Character Controller	https://docs.unity3d.com/ru/2019.4/Manual/class-CharacterController.html
Brackeys. THIRD PERSON MOVEMENT in Unity	https://www.youtube.com/watch?v=4HpC--2iowE
Code Monkey. How to use Machine Learning AI in Unity! (ML-Agents)	https://www.youtube.com/watch?v=zPFU30tbyKs